# Cours VB.NET

Par Philippe Lasserre

Date de publication : 8 janvier 2011

Dernière mise à jour : 15 avril 2020

Il s'agit d'un cours de Visual Basic.Net © de Microsoft complet, pour débutants (pas de prérequis) ou programmeurs voulant passer à la version .net.
Autres contributions du même auteur:
**Cours sur Chart permettant d'afficher des graphiques**
**Pour voir et télécharger LDF: logiciel de compta en Shareware, cliquer ici.**
**Télécharger un dictionnaire médical gratuit pour Word.**
philippe@lasserrelyon.fr

Nouveau : MAJ avec VB 2010, Nouvelles rubriques : en WPF

**Commentez**

# I - Introduction

Il s'agit d'un cours de Visual Basic.Net © de Microsoft, pour débutants (pas de pré requis) ou de programmeur voulant passer à la version .net.

Le cours est basé sur VB 2005 (Framework 2), VB 2008 (Framework 3.5), VB 2010 (Framework 4).
VB 2003 (Framework 1) est progressivement abandonné, car il contenait des fautes de jeunesse.
Les versions Express (Gratuites) sont privilégiées.

Visual Basic.Net apporte une puissance inégalée et nécessite une rigueur importante, mais il devient vite complexe et technique. La documentation et les livres sont totalement hermétiques pour les novices et rebutent totalement les débutants. Les articles sur le Web sont très techniques et traitent d'emblée de problèmes complexes, ils sont nécessaires, mais pas pour le débutant. J'explique donc dans ce cours, à ma manière, très simplement, comment créer un programme afin de permettre un bon démarrage même à celui qui n'a jamais fait d'informatique.(Je traite des programmes Windows: Windows Forms et WPF mais pas ASP Web).J'encourage par ce cours sans prétention, à développer ses propres programmes.

**Soyez un utilisateur actif**

- Retournez les bugs et erreurs et même les fautes d'orthographe. Mon site serait-il parfait ? J'en doute !! Merci de vos critiques.
- Adressez-moi vos idées, du code original, des infos à mettre sur le site.
- Ou simplement indiquez-moi que vous avez lu mon cours, cela fait toujours plaisir et m'incite à poursuivre.

Merci à developpez.com, à ses membres qui m'ont aidé (à Guillaume en particulier) à ceux qui m'envoient un petit mot, et à ceux qui me donnent un coup de main.

Cours constamment remis à jour : voir la date de la version en début d'article.

**Questions à l'auteur :** je ne peux pas répondre à toutes les questions particulières et spécifiques, car je n'ai pas d'expérience poussée dans tous les aspects de VB, et les questions sont très nombreuses, aussi je vous conseille d'utiliser les forums developpez.com

## II-A - Qu'allons-nous étudier ?

Ce cours est un cours de Visual Basic.Net développé par Microsoft.

Nous étudierons principalement : LES APPLICATIONS WINDOWS. (les WindowsForms) et les WPF.


Les applications WindowsForms et WPF sont des programmes directement exécutables qui utilisent des fenêtres: des programmes de traitement de texte, d'image, de musique, des jeux, de petits utilitaires, des logiciels métiers (médicaux)…

Nous laisserons de côté les applications 'Web' (en ASP qui utilisent les WebForms) et qui permettent de créer des sites Internet, les applications 'console'.

Les versions étudiées sont VB 2005 (Framework 2), VB 2008 (Framework 3.5), VB 2010 (Framework 4).
VB 2003 (Framework 1) est progressivement abandonné, car il contenait des fautes de jeunesse.
Les versions Express (Gratuites) sont privilégiées.

## II-B - Quel plan de cours suivrons-nous ?

Nous étudierons donc comment créer une application Windows.

Nous étudierons la notion d'objet, d'événement, d'instruction, procédure et module.(Section III).

Nous étudierons l'IDE ou interface de développement (Section IV).

Nous étudierons le langage Visual Basic (Section V).

Nous verrons les Classes VB (Section VI).

Nous utiliserons 'contrôles' WindowsForms pour créer l'interface utilisateur (Section VII).

Nous découvrirons la manière de créer une application Windows Forms.(Section IX).

Nous utiliserons les WPF pour créer l'interface utilisateur (Section XI).

Nous apprendrons à faire de la programmation objet et à créer une classe (Section XIII.)

Nous verrons comment utiliser les bases de données. (Section XV.)

## II-C - Quels logiciels utiliser ?

Historique : il y avait **Visual Basic.Net 2003** de Microsoft en 2003 !!


Il fonctionnait avec le Framework 1.1.

En 2005 il y a eu **Visual Basic 2005** de Microsoft et le Framework 2.0.

Ce produit .Net était maintenant mature, l'environnement de développement magique, les quelques points noirs de la version 2003 ont été corrigés.


En 2008 il y a eu **Visual Basic 2008** de Microsoft et le Framework 3.5 qui permettait d'utiliser les WPF.


En avril 2010 il y a **Visual Basic 2010** de Microsoft et le Framework 4.

**VisualStudio** (payant) contient Visual Basix C#. Mais il existe aussi la version **Visual Basic Express** version allégée, mais très bien et GRATUITE, en français.

Est-il possible d'utiliser les éditions Express à des fins commerciales ?
Oui. Il n'y a aucune restriction liée aux licences pour les applications créées à l'aide des éditions Express.

Cette réponse (pour VB express 2008) est indiquée sur le site de Microsoft : http://msdn.microsoft.com/fr-fr/express/default.aspx

Ce cours utilise donc Visual Basic 2005 2008 et 2010 Express.

Si vous débutez, installez et utilisez sans hésitation **Visual Basic Express 2010 GRATUIT**
Nous abandonnons VB 2003, la première version en Net, qui avait quelques gros défauts.

Où trouver Visual Basic 2010 Express ?
Cliquer sur le lien:
**http://www.microsoft.com/express/downloads/** Dans la liste de lien, cliquer sur 'Visual Basic Express 2010'.
Puis dans la liste 'Select language", choisissez"French", une fenêtre pop-up démarre.

Autre gratuit: **SharpEditor** venant du monde du libre.

*SharpEditor*

Voici le site,**SharpDevelop** le configurer pour qu'il marche en VB (il supporte VB et C#).

## II-D - Quelle configuration est nécessaire ?

Pour développer avec Visual Studio 2005 VB 2005 VB 2010.

il faut Windows XP ou 2000 (non vérifié pour VB 2010) ou Vista ou Windows 7 avec au minimum 1 Go de mémoire vive. Un grand écran (vu le nombre de fenêtres) est conseillé.

Les exécutables fonctionnent sous Windows 98 (pour VB 2003), XP (à vérifier pour VB 2008 et 2010), Vista, Windows 7.

## II-E - À propos de l'auteur

**LASSERRE Philippe** est médecin généraliste exerçant en groupe dans le Rhône (à Toussieu), il développe des logiciels depuis des années.

Il n'est pas informaticien.

Il a travaillé avec des ordinateurs :

ZX81, New-Brain, Ti-99, TO7, Vic20, Oric, Apple II, puis PC avec l'aide de Bill.

Il a utilisé le Basic Microsoft, le QuickBasic le Visual Basic de Microsoft ® de la version 1 à la version VB6 et VB.Net, a fait un peu d'assembleur Z80 il y a longtemps.

Il a fait partie de MEDITRA, association de médecins informatisés du Rhône pionnière en la matière à l'époque, il a été cofondateur d'un club d'informatique local (Microzon) où on programmait dur !!

Ensuite il a écrit des logiciels, pour cela outre le côté technique informatique, il a beaucoup travaillé sur le dossier médical informatisé, les plans de soins.

Plein d'idées et de projets, un seul problème : il n'y a que 24 heures dans une journée.

Il est l'Auteur de :

**CREEMED**, il y a quelques années. C'était un utilitaire pour Medigest ® Dos.

**MEDIWIN®** distribué par Polytel , écrit en VB6, logiciel de gestion complète de cabinet médical dont il est le coauteur.

Logiciel agréé Sesam-Vitale, très complet, innovant, incluant les notions de "dossier vivant", "procédures de soins" (contenu médical validé par des thèses), travaillant avec la base de médicament BCB de Résip©, contenant le dictionnaire de la SFMG.
Ce logiciel n'est plus développé.

**LDF** logiciel de comptabilité en Shareware.**Télécharger Ici.**
Il distribue gratuitement un dictionnaire de termes médicaux pour Word.

Il a créé un site pour son association de plongée sous-marine (EmbellieBulle.fr) sous SPIP.
Il est fana de musique de cinéma de photographie et de voyages.

Vous pouvez envoyer un mail à Mr LASSERRE à l'adresse : **lasserre.philippe@wanadoo.fr**

# III - Principe et structure des programmes

## III-A - Les 'Objets'

VB utilise la notion d'OBJETS'.

Pour bien comprendre ce qu'est un objet, nous allons prendre des exemples dans la vie courante puis nous passerons à des exemples dans Visual Basic.


Voir la vidéo : **au format 'Flash'**> ou **au format 'Avi'** en Visual Basic 2005.

La vidéo (identique à celle du chapitre sur les Classes) contient :

1) Objets, Classes ;

2) Références, espaces de noms (à visionner plus tard).

### III-A-1 - Dans la vie courante

Ma voiture est un **objet**, cet objet existe, on peut l'utiliser.

Ma voiture fait partie de "Les voitures", du type, du genre "Les voitures". "Les voitures" c'est une classe (Class) qui a ses caractéristiques.
"Les voitures" ont une couleur, un moteur…, elles roulent en transportant des passagers…

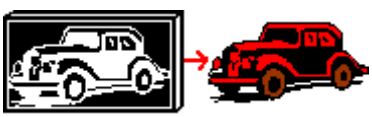
mais je ne peux pas utiliser "Les voitures", la Classe; pour me déplacer, il faut avoir un objet "voiture".

**Avec la Classe je vais créer des Objets.**

Pour fabriquer ma voiture, je prends les caractéristiques de la class "Les voitures" (c'est comme un moule, une usine) et je fabrique une voiture, je la nomme *'MaVoiture'*.

```
Dim MaVoiture As New Lesvoitures
```

*MaVoiture* est maintenant un nouvel objet de **type** 'Les voitures'.



Class --> Objet

Type 'Les voitures'--> Objet 'Mavoiture'

Un Objet est créé selon un 'modèle' qu'on appelle une Classe.

On dit aussi qu'il faut **instancier** un objet à partir de la Classe.

'Mavoiture' est une **instance** de la classe 'Les voitures'.

(On dit aussi une '**occurrence**' de la classe).

**De manière générale, une classe est une représentation abstraite de quelque chose.**
**Tandis qu'un objet est un exemple utilisable de ce que représente la classe.**

Remarque

Avec la même classe on peut instancier plusieurs Objets.

**Propriétés (Attributs)**

Prenons *MaVoiture*. 

Elle a des propriétés : une marque, une couleur, une puissance…

Pour indiquer la couleur de ma voiture on utilise la notation :

```
MaVoiture.couleur
```

Syntaxe : *Objet.Propriété* (Il y a un point entre les 2 mots).

Pour modifier la couleur et avoir une voiture verte on écrit :

```
MaVoiture.couleur= "Vert"
```

Et la voiture devient verte !! 

*MaVoiture.Phares.Avant* indique les phares avant de la voiture.

*MaVoiture.Phares.Avant.Allumé* indique l'état des phares (Allumé ou non)

Si je fais :

*MaVoiture.Phares.Avant.Allumé=True* (Vrai) cela allume les phares.

**Méthodes**

*MaVoiture* fait des choses : elle roule par exemple.

Pour faire rouler la voiture j'appelle la méthode 'Roule'.

*MaVoiture.Roule*

Syntaxe : *Objet.Méthode* (il y a un point entre les 2 mots).

Si c'est possible pour cette méthode je peux indiquer à quelle vitesse la voiture doit rouler :

*MaVoiture.Roule(100)* 'j'ai ajouté un paramètre.

Le paramètre est un renseignement envoyé avec la méthode.

Il est possible parfois d'indiquer en plus si la voiture doit rouler en marche avant ou en marche arrière.

*MaVoiture.Roule(10, Arriere)*

Il y a donc 2 manières d'appeler la méthode Roule : avec 1 ou 2 paramètres, on dit que la méthode est surchargée, chaque manière d'appeler la méthode s'appelle 'signature'.

Première signature: *MaVoiture.Roule(100)*

Seconde signature: *MaVoiture.Roule(10, Arriere)*

**événement**

Des événements peuvent survenir sur un objet.

*MaVoiture_démarre* est un événement, quand la voiture se met en route (si par exemple j'ai fait *MaVoiture.Roule(10, Arriere)),* cet événement *démarre* se déclenche automatiquement.

**Interface et implémentation**

Ce que je vois de l'objet, c'est son interface exemple: la méthode *Roule* fait partie de l'interface d'une voiture. Par contre ce qui fait rouler physiquement la *voiture* se nomme implémentation, c'est le moteur, ce n'est ni visible ni accessible.

Si je crée une voiture je vais faire l'implémentation, je vais fabriquer le moteur, mais si je suis simplement utilisateur de l'objet voiture, je vais me contenter d'utiliser l'interface.

**Visibilité**

Quand un objet est créé, il est visible et utilisable, uniquement dans la zone où il a été défini.

**Relation entre Objets**

*Héritage*

Une Classe (un moule) peut hériter d'une autre classe (d'un autre moule).

La classe Voiture hérite de la classe *Véhicule*, cela veut dire qu'une voiture est un véhicule; la classe voiture aurait les propriétés de la classe Véhicule.

*Contenant-contenu*

On peut créer une Classe qui contient des Objets, une classe qui se compose d'objets. On parle de composition.

L'objet *Voiture* contient 4 objets Roue.

On dit que l'objet **encapsule** le contenu.

*Collections*

Les collections sont des groupes d'objets semblables qui peuvent être énumérés.

Un parc automobile contient X Voitures, chaque voiture a un numéro d'item:

*ParcVoiture.item(1)* pour la première Voiture ;

*ParcVoiture.item(2)* pour la seconde Voiture.

**Espace de noms**

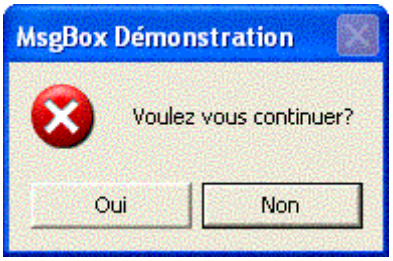
Un espace de noms regroupe des objets qui appartiennent au même 'domaine'. Cela sert a différencier des objets qui ont même nom, mais ne font pas partie du même domaine. Si je parle de 'Porte' ce terme n'a pas la même signification si je suis dans l'espace de noms 'Composant électronique' (on y trouve des portes logiques) ou l'espace de noms 'Maison'.

Tout cela c'est simpliste, mais voilà, vous avez compris ce qu'est un objet !

## III-A-2 - Dans Visual Basic.net

Une application Windows se compose de fenêtres (nommées aussi formulaires) dans lesquelles se trouvent des contrôles (bouton, liste, texte…).

Exemple de fenêtre avec 2 boutons, une zone de texte (un label) et une icône :




Dans une application Windows, il y a aussi des lignes de code utilisant des variables pour faire des calculs.

**En Visual Basic.Net tout est objet :**

- les fenêtres (on dit les formulaires) ;

- les variables ;

- les contrôles (bouton, liste, image, case à cocher…).

Il faut un **moule** pour faire un objet. Le moule c'est une **Classe**.

Le moule va servir à créer un objet,on dit **une instance**.

On peut créer, **instancier** une multitude d'objets avec le même moule.

Pour créer, démouler un objet, on utilise les mots-clés *Dim* et *As New*.

```
Dim objet As New Classe
```

*New* est un *constructeur*.

Exemple : créer une fenêtre (un formulaire) :

Je dessine une fenêtre *FormDémarrage* (c'est la Classe, le moule)

puis

```
Dim F As New FormDémarrage
```

Crée une fenêtre qui se nomme 'F' à partir du moule, du modèle (FormDémarrage) que j'ai dessiné.

Autre exemple :

```
Dim B As New Button
```

Créer un bouton nommé 'B' avec les attributs habituels des boutons (Class Button)

Troisième exemple

Comment créer une variable nommée Mavariable pouvant contenir un entier (Integer)

```
Dim MaVariable As New Integer
```

*Dim MaVariable As Integer* 'est correct aussi.

Ici, pour une variable, on remarque que New peut être omis

**Tout objet a des propriétés.**

On utilise la syntaxe : *Objet.Propriété* (il y a un point entre les 2 mots).

Si F est une fenêtre, *F.BackColor* indique la couleur de fond de la fenêtre.

S'il y a un bouton, la couleur de fond du bouton sera :

```
Bouton.BackColor
```

ou

```
F.Bouton.BackColor
```

Noter la syntaxe : la couleur du bouton qui est dans la fenêtre F.

En fait une propriété c'est une sorte de variable.

Comment modifier cette propriété ?

```
Bouton.BackColor=Color.Red
```

'modifie la couleur de fond du bouton


Autre exemple :

La propriété Visible : si elle a la valeur True (Vraie) l'objet est visible, si elle est à False l'objet n'est pas visible.

```
Bouton.Visible=False
```

fait disparaitre le bouton.

=Ici il y a un bouton invisible !! oui, oui !!

**Les objets ont des méthodes parfois.**

Une méthode agit sur l'objet ou fait agir l'objet.

Prenons un exemple simplifié.

Les listes (liste déroulante) ont des lignes (Items) et une méthode Clear qui permet de les vider.

Si je veux vider toutes les lignes d'une liste nommée Liste1, je fais :

```
Liste1.Items.Clear()
```

 Les propriétés et méthodes se nomment les membres d'un objet.

**Certains objets ont des événements.**

Reprenons notre bouton. Quand l'utilisateur clique dessus, l'événement Bouton_Click survient.

Ce sont les objets contrôles (bouton, case à cocher…)et les formulaires qui ont des événements.

**Interface et implémentation**

Ce que je vois de l'objet, c'est son interface (le nom des propriétés, méthodes…) exemple : la méthode Clear fait partie de l'interface d'une ListBox. Par contre le code qui effectue la méthode (celui qui efface physiquement toutes les lignes de la listBox), ce code se nomme implémentation, lui n'est ni visible ni accessible.

**Visibilité**

Quand un objet est créé, il est visible et utilisable, uniquement dans la partie du programme où il a été défini.

Par exemple habituellement, je peux voir et modifier la couleur d'un bouton uniquement dans le code de la fenêtre ou il est situé.

Pour les variables on parle de portée : la variable peut être locale (Private) ou de portée générale ('Public') visible partout.

**Relation**

*Héritage*

Une Classe (un moule) peut hériter d'une autre classe (d'un autre moule).

La classe *Button* hérite de la classe *Control*, cela veut dire qu'un bouton est un contrôle.

Si je crée un bouton, il aura les caractéristiques de la classe Button, mais aussi de la classe Control.

*Contenant-contenu*

Une Classe peut contenir d'autres classes.

Je peux décider qu'un Objet *Rectangle* va contenir 4 Objets *Point*

*Collections*

Les collections sont des groupes d'objets semblables qui peuvent être énumérés.

Une fenêtre Windows (on dit un 'formulaire' contient une collection nommée 'Controls' composée de tous les objets (boutons, List, texte) contenus dans la fenêtre :

```
maFenetre.Controls.item(1)
```

contient par exemple le premier bouton

```
maFenetre.Controls.item(2)
```

contient par exemple une liste.

**En résumé**

Une classe est un élément logiciel qui décrit un type de données abstrait. Un type de données abstrait est un ensemble d'objets définis par une liste d'opérations et les propriétés de ces opérations Une classe est un élément logiciel qui décrit les caractéristiques d'un ensemble d'objets. En programmation orientée une classe déclare des propriétés communes à un ensemble d'objets. La classe déclare des attributs représentant l'état des objets et des méthodes représentant leur comportement. Une classe représente donc une catégorie d'objets. Il apparait aussi comme un moule ou une usine à partir de laquelle il est possible de créer des objets. On parle alors d'un objet en tant qu'instance d'une classe (création d'un objet ayant les propriétés de la classe). (www.techno-science.net)

En Visual Basic.net tout est objet.

Les Classes sont des types d'objets.

Pour créer (instancier) un objet à partir d'une Classe, il faut utiliser les mots clé Dim …As New :

```
Dim Objet As New Class
```

**Un objet a :**

- des propriétés ;
- des méthodes ;
- des événements.



*Attention, par abus de langage, on emploie parfois indifféremment les mots 'Classe' et 'Objet', mais il est préférable de ne pas confondre le modèle et l'objet lui-même.*

**Espace de noms**

Les objets sont regroupés dans des **bibliothèques d'objets**, dans des 'espaces de noms'.

Il faut parfois importer la bibliothèque avant d'utiliser l'objet :

```
'Importe l'espace de noms Systel.IO
Imports System.IO

'On peut maintenant utiliser l'objet 'File':
Fl = File.Exists("vessaggi.gif")
```

Si l'import n'a pas été fait, **System.IO.File.Exists()** est accepté aussi.

**Lexique anglais=>français.**

New = Nouveau.

## III-B - Programmation événementielle : le premier programme

**Nous allons comprendre la programmation événementielle : comment fonctionne Visual Basic.**

- Ce que voit l'utilisateur.
- Ce qu'a fait le développeur pour arriver à ce résultat.

Voir la vidéo au format 'Flash':  ou au format AVI  en Visual Basic 2005

### III-B-1 - Principes de la programmation VB

Le programmeur va dessiner l'interface utilisateur (fenêtre, bouton, liste…), il va ensuite uniquement écrire les actions à effectuer quand certains événements se produisent sur cette interface.

C'est Visual Basic qui va entièrement s'occuper de la gestion des événements.

### III-B-2 - Exemple : le premier programme

Il affiche 'Bonjour' quand on clique sur un bouton.

Ce n'est pas original : le premier programme, dans tous les cours d'informatique, permet d'afficher 'Bonjour' (ou 'Hello Word').

**Que voit l'utilisateur du programme ?**

L'utilisateur final, celui qui utilise le logiciel, voit une fenêtre avec un bouton, S'il appuie sur ce bouton il voit s'afficher "Bonjour".


**Que se passe-t-il dans le programme ?**

Quand l'utilisateur clique sur le bouton, cela déclenche automatiquement un événement. (Button1_Click), cet événement contient du code qui affiche "Bonjour".

**Que doit faire le programmeur pour arriver à ce résultat ?**

Pour atteindre ce résultat, le programmeur va dessiner la fenêtre, le bouton, la zone d'affichage du texte (un label) puis il va simplement indiquer dans l'événement Button_Click d'afficher "Bonjour".

*Le fait de déterminer la procédure à appeler ou de réaliser l'appel est entièrement pris en charge par VB.*

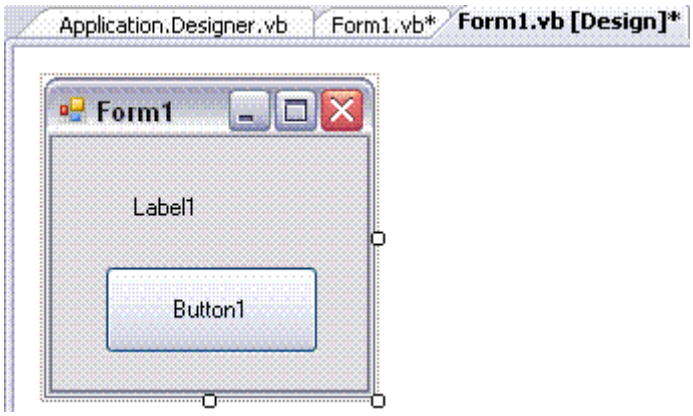
## III-B-3 - En pratique, que fait le programmeur ?

Voir la vidéo : **au format 'Flash'**> ou **au format 'Avi'** en Visual Basic 2005.

Le programmeur est en mode 'conception' (ou mode Design) : il écrit le programme.

### III-B-3-a - Il dessine l'interface utilisateur

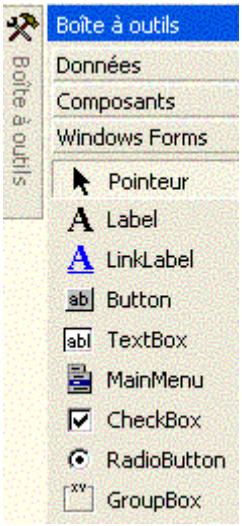
(Ce que verra l'utilisateur final, c'est l'interface utilisateur : une fenêtre avec des boutons, des listes, du texte…) :




Il ouvre un projet : une fenêtre 'Form1' apparait.

Il ajoute un bouton.

Pour cela il utilise la Boite à outils :




Il clique sur 'Boite à Outils' à gauche, bouton Windows Forms, puis bouton 'Button', il clique dans Form1, déplace le curseur sans lâcher le bouton, puis lâche le bouton de la souris : le dessin d'un bouton apparait.

Pour l'exemple,Il ajoute un label.

Un label est un contrôle qui permet d'afficher un texte.

Comme pour le bouton, il clique sur 'Boite à Outils' à gauche, bouton Windows Forms, bouton 'Label' et met un contrôle label sur la fenêtre.

## III-B-3-b - Il écrit le code correspondant aux événements

Il double-clique sur le bouton qu'il a dessiné :

Une fenêtre de conception de code s'ouvre et il apparait :

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
 Button1.Click

End Sub
```

Cela correspond à la **procédure événement** en rapport avec l'événement 'On a cliqué sur le bouton1'.

Quand le programme fonctionne, quand l'utilisateur du logiciel clique sur le bouton1, le code situé entre Private Sub Button1Click et End Sub est effectué.

**Une procédure** est un ensemble de lignes de code qui commence par Sub et se termine par End Sub (ou Function… End Function).

Comment indiquer dans cette procédure d'afficher "Bonjour" ?

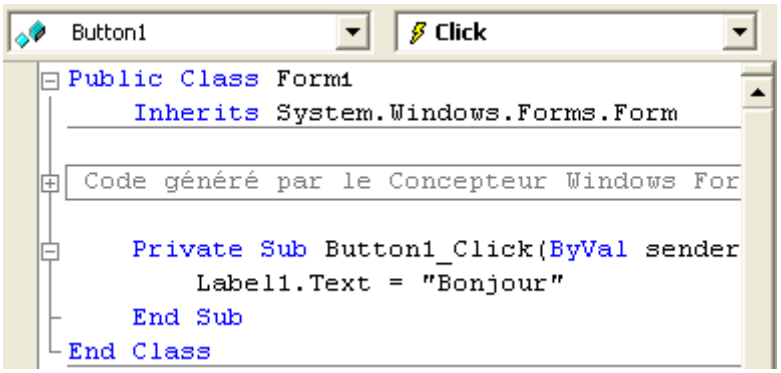
Le label possède une propriété nommée '.text' qui contient le texte à afficher.

Il faut taper le code qui modifie cette propriété '.text' , qui y met la chaine de caractères "Bonjour":

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
 Button1.Click

        Label1.Text = "Bonjour"
```

```
End Sub
```

Cela donne :




Voilà votre premier programme est écrit.

**Comment exécuter ce programme ?**

Il est possible de tester immédiatement le programme en mode débogage, sans quitter l'environnement de développement.

Utiliser le menu 'Déboguer' puis 'Démarrer' qui lance l'exécution du programme.

On peut aussi taper sur F5 pour lancer le programme.

Ou plus simplement cliquer sur la flèche :

C'est plus rapide, lancer l'exécution avec le premier bouton, le second servant à arrêter temporairement l'exécution, le troisième à terminer l'exécution.

**En mode exécution**

L'utilisateur voit bien une fenêtre avec un bouton, s'il clique dessus, "Bonjour" s'affiche.

Quand le programme est totalement écrit, terminé, testé, il est possible de le compiler et ainsi de créer un fichier exécutable (possédant une extension '.exe') qui fonctionne de manière autonome en dehors de l'environnement de développement.

C'est ce **fichier exécutable** qui est fourni à l'utilisateur.

Par opposition le code écrit par le programmeur, composé d'instructions Visual Basic, se nomme **le code source**.

**En résumé**

Le programmeur utilise des outils de dessin pour construire une interface utilisateur : des fenêtres avec des contrôles dessus: menus, boutons, case à cocher…

VB, pour chaque fenêtre ou pour chaque contrôle, génère une liste d'événements, (événement lié au chargement d'une fenêtre, événement lié au fait de cliquer sur un bouton, événement survenant quand on modifie un texte…).

Il suffit, dans la procédure événement qui nous intéresse, d'écrire le code qui doit être effectué lorsque cet événement survient.

Comme nous l'avons vu le code sert à agir sur l'interface (Afficher un texte, ouvrir une fenêtre, remplir une liste, un tableau), mais il peut aussi effectuer des calculs, évaluer des conditions et prendre des décisions, travailler en boucle de manière répétitive et ainsi effectuer les tâches nécessaires.

## III-C - Les instructions, les procédures : les 'Sub', les 'Function'

Qu'est-ce qu'une instruction, une procédure ?

Quelle différence entre les procédures liées aux événements ? Non liées ? :

Les 'Sub', les 'Functions'.

### III-C-1 - Les instructions

Une instruction est le texte tapé au clavier dans le 'code source' et permettant d'effectuer une opération, une déclaration, une définition.
Elle contient des mots-clés, des opérateurs, des variables, des constantes et des expressions des appels à des fonctions ou des méthodes. On verra cela en détail.

```
Dim A As Integer
```

est **une instruction** (de déclaration).

```
A = 1
```

est aussi une instruction qui effectue une opération.

C'est habituellement une **'ligne de code exécutable'…**

Une instruction est exécutée lorsque le programme marche.

**Plusieurs instructions peuvent se suivre sur une même ligne**, séparées par ':'

```
Dim B As String : B="Bonjour"
```

**Si une ligne est très longue, on peut passer à la ligne grâce à ' _'**

(caractère 'Espace' puis caractère "_" puis immédiatement après, passage à la ligne) :

```
Dim B, C As String
B = "Bonjour monsieur ": C= _
"le professeur"
```

est équivalent à :

```
Dim B, C As String
B = "Bonjour monsieur ": C= "le professeur"
```

En VB 2010, après certains mots, il peut y avoir continuation de ligne implicite (plus besoin de _ après la virgule, une parenthèse ouvrante, après & ou { ou = ou +…).

Quand un programme tourne, les instructions sont effectuées ligne après ligne.

```
1  Dim B As String
2  B="Bonjour"
3  Dim A As Integer
4  A= 3
5  A= A + 1
```

La ligne 1 est exécutée puis la ligne 2 puis la 3, la 4…

Bien que l'on puisse avoir des numéros de ligne, ils ne sont plus utilisés actuellement et non visibles :

```
Dim B As String
B="Bonjour"
Dim A As Integer
A= 3
A= A + 1
```

Pour mettre **des commentaires** dans un programme, on le fait précéder de ' (on peut aussi utiliser le mot REM en début de ligne).

```
'Ceci est un commentaire,
'Cela n'est pas une instruction.
REM Ceci est aussi un commentaire.
```

Le commentaire ne sera pas exécuté.

Il peut aussi, à partir de VB 2005 , y avoir des commentaires en XML, ils sont dans ce cas précédés de ''' (3').

```
''' <summary>
''' fonction Calculant le total
''' </summary>
''' <param name="custName"></param>
''' <param name="amount"></param>
''' <remarks> </remarks>
Overloads Sub calcul(ByVal custName As String, ByVal amount As Single)
    ' Insert code to access customer record by customer name.
End Sub
```

```
calcul(
calcul (custName As String, amount As Single)
fonction Calculant le total
```

## III-C-2 - Les procédures

Une procédure est un ensemble d'instructions, de lignes de code, un groupement d'instructions bien définies effectuant une tâche précise.

Les procédures sont bien délimitées.

**Il y en a de 2 sortes.**

**Les procédures Sub**

Elles débutent par le mot *Sub* et se terminent par *End Sub*.

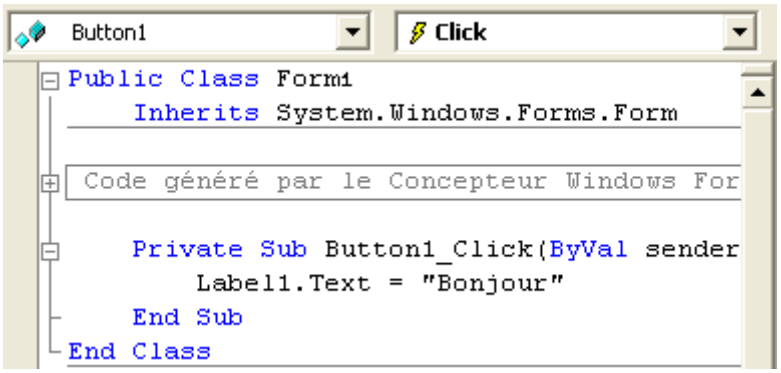
**Les procédures Function**

Elles débutent par *Function* et se terminent par *End Function*.

Exemple :

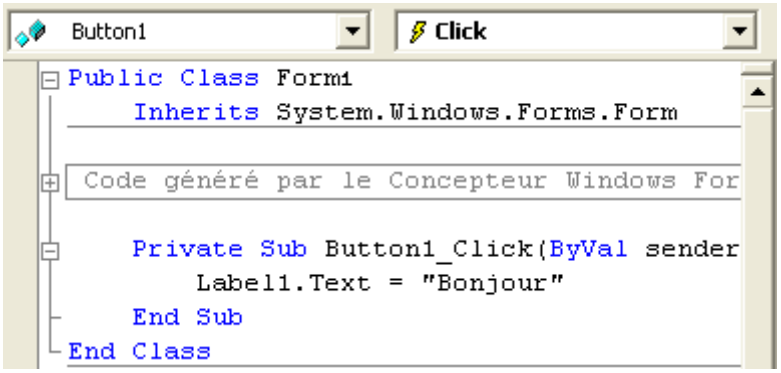
```
Sub Maprocédure

   A=1

End Sub
```

Exemple concret d'une procédure : la procédure Button_Click du premier programme. (Celui qui affiche 'Bonjour', elle ne contient qu'une ligne de code. Le mot Sub est précédé de Private, on verra plus loin ce que cela signifie.




Vous avez vu que l'on peut dessiner l'interface, une fenêtre Form1 par exemple. En mode conception, après avoir dessiné l'interface, on doit avoir accès aux procédures.

*Si on double-clique sur la fenêtre, on a accès aux procédures événement liées à cette fenêtre, si on double-clique sur un objet (bouton, case à cocher…), on voit apparaitre les procédures événement de ce contrôle.*




Quand on voit ces procédures, on peut y inclure du code.

Nous allons voir qu'il y a 2 types de procédures : les procédures liées aux événements et celles qui ne sont pas liées.

## III-C-3 - Procédures liées aux événements

Si on **double-clique sur le fond d'une fenêtre (en vb 2010)**,(celle du programme 'Bonjour') on voit apparaitre **les procédures liées** à cette fenêtre et aux contrôles contenus dans cette fenêtre :

```
Public Class Form1



Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)  _
Handles MyBase.Load

            Label1.Text = ""

End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles Button1.Click

        Label1.Text = "Bonjour"

End Sub

End Class
```

Détaillons :

```
Public Class Form1

End Class
```

Ce n'est pas une procédure, mais la 'Classe' définissant la fenêtre.

En VB 2003, il y a une 'région' que vous déroulez, en cliquant sur le petit +, vous pouvez y lire le code permettant de créer la fenêtre, les contrôles… C'est généré automatiquement par VB. (Le chapitre VII-C sur les formulaires explique en détail le code généré par VB, mais c'est un peu complexe pour les débutants pour le moment !!).

En VB 2005 2008 et 2010, cette partie générée par VB n'est pas visible directement.

Il faut comprendre qu'à un formulaire (fenêtre) et aux contrôles qui sont dans ce formulaire correspond du code généré par VB. Ce code (sur lequel vous n'intervenez habituellement pas) permet de créer le formulaire et les contrôles.

Chaque fenêtre a une procédure *Form_Load* qui est exécutée lorsque la fenêtre est chargée, on y met généralement le code initialisant la feuille.

```
Private Sub Form1_Load

End Sub
```

Il y a bien d'autres procédures liées à la fenêtre.

Dérouler la liste box en haut à gauche de la fenêtre de code, cliquer sur **(Form1 events),** si vous déroulez maintenant la liste à droite vous aurez **tous les événements qui génèrent une procédure :**

*Load* Lors du chargement de la fenêtre ;

*Unload* Lors du déchargement de la fenêtre ;

*Activated* Lorsque la fenêtre devient active ;

*GotFocus* Lorsque la fenêtre prend le focus ;

*Resize* Lorsque la fenêtre est redimensionnée.

```
Private Sub  Button1_Click

End Sub
```

C'est **la procédure liée au bouton** et qui contient le code à effectuer quand l'utilisateur clique sur le bouton.

C'est là que l'on écrit le code qui doit s'effectuer lorsque l'utilisateur clique sur le bouton.

De la même manière que pour la fenêtre, vous pouvez voir dans la liste en haut, tous les événements liés aux boutons qui génèrent une procédure :

*Click* Lorsque l'utilisateur clique sur le bouton ;

*DoubleClick* Lorsque l'utilisateur double-clique sur le bouton ;

*MouseDown* 'se déclenche si appui du bouton gauche de la souris ;

*MouseUp* 'se déclenche si relâchement du bouton gauche de la souris.

*On voit donc que le formulaire (la fenêtre) et tous les contrôles d'une application ont chacun des procédures pour chaque événement qui peut survenir.*

## III-C-4 - Procédures non liées

Parfois on a besoin de code qui fait une tâche particulière, qui est utilisé à plusieurs endroits et qui **n'est pas liée à un événement**.

On crée dans ce cas **une procédure indépendante des événements**.

Le système des procédures permet aussi de découper un problème complexe en quelques fonctions moins complexes et indépendantes les unes des autres.

Un programme vb est donc composé de procédures dont l'exécution est déclenchée par des événements (ouverture d'une fenêtre, clic sur un bouton…), ces procédures en appellent d'autres qui en appellent d'autres…

Ces procédures sont en fait des **sous-programmes :** si une ligne appelle une procédure, le programme 'saute' au début de la procédure, il effectue le code de la procédure puis revient juste après la ligne qui avait appelé la procédure et continue les lignes suivantes.

Exemple : plusieurs fois dans le programme j'ai besoin de calculer la surface d'un cercle à partir de son rayon et de l'afficher sur un label.

Plutôt que de retaper dans chaque procédure le code, je peux créer une procédure 'Sub' nommée *AfficheSurfaceCercle*.

Il suffit ensuite si nécessaire d'appeler la procédure qui effectue le calcul et affiche le résultat puis revient effectuer le code situé après l'appel.

**Comment appeler une procédure ?**

**Avec :**

```
Call NomdeProcedure()
```

ou par

```
NomdeProcedure()
```

Call est facultatif.

Noter les parenthèses après le nom de la procédure.

## III-C-5 - Procédures 'Sub'

**Comment créer cette procédure Sub ?**

Dans la fenêtre de code, tapez :

```
Sub AfficheSurfaceCercle
```

puis validez. Vous obtenez :

```
Sub AfficheSurfaceCercle()

End sub
```

Le code de la procédure est compris entre le *Sub* et le *End Sub*.

Pour que le calcul se fasse, il faut fournir (transmettre de la procédure qui appelle à la procédure Sub) la valeur du rayon.

Pour indiquer que la Sub doit recevoir un **paramètre** (un **argument** en VB) ajouter entre les parenthèses :

```
Sub AfficheSurfaceCercle( Rayon as Single)
```

Cela signifie qu'il existe une procédure qui reçoit comme paramètre une variable de type Single (Réel simple précision) contenant le Rayon.

Ajouter le code :

```
Label.text =(3.14*Rayon*Rayon).ToString
```

**Que fait cette ligne ?**

Elle fait le calcul: '3.14*Rayon*Rayon' ('*' signifie multiplier), on transforme le résultat en chaine de caractères (grâce à '.ToString') que l'on met dans la propriété .text du label : Cela affiche le résultat. (On verra toute cette syntaxe en détail ultérieurement.)

On obtient :

```
Sub AfficheSurfaceCercle( Rayon as Single)

    Label.text =(3.14*Rayon*Rayon).ToString

End sub
```

**Comment appeler cette Sub ?**

N'importe quelle procédure pourra appeler la Sub AfficheSurfaceCercle en envoyant la valeur du rayon afin d'afficher la surface du cercle dans un label.

Exemple d'appel pour un rayon de 12 :

```
AfficheSurfaceCercle(12)
```

Affiche dans le label : 452.16.

## III-C-6 - Procédures 'Function'

Parfois on a besoin que la procédure retourne un résultat, un seul, qu'elle donne en retour un résultat à la procédure appelante. Dans ce cas on utilise une Fonction.

Exemple : je veux créer une fonction à qui je fournis un rayon et avoir en retour la surface d'un cercle.

**Comment créer cette Function ?**

Tapez *Function SurfaceCercle* puis validez, ajouter *(Rayon As Single)*

Tapez *Return 3.14*Rayon*Rayon*

Ce que la fonction doit retourner est après *Return* (ce que la procédure doit renvoyer à la procédure appelante).

On obtient la fonction complète :

```
Function SurfaceCercle( Rayon as Single)

    Return 3.14*Rayon*Rayon

End Function
```

**Comment appeler cette Function ?**

Dans la procédure qui appelle, il faut une variable pour récupérer la valeur retournée par la Fonction :

```
S= NomdelaFonction()
```

N'importe quelle procédure pourra appeler la fonction et obtenir le résultat dans la variable S par exemple pour un rayon de 12 :

```
Dim S As Single

S=SurfaceCercle(12)
```

On appelle la fonction SurfaceCercle en envoyant le paramètre '12', ce qui fait qu'à l'entrée de la fonction, Rayon=12, le calcul est effectué et le résultat du calcul (452.16) est retourné grâce à Return. S récupère ce résultat.

Après l'appel de cette fonction, S est égal à 452.16.

Il est possible de spécifier le type retourné par la fonction :

```
Function SurfaceCercle( Rayon as Single) As Single
```

*As Single* en fin de ligne après () indique que la fonction retourne un Single (un nombre en simple précision). Il faut donc que la variable qui reçoit la valeur retournée (S dans notre exemple) soit aussi un Single.

Il existe une **autre manière de retourner le résultat d'une fonction**, reprenons l'exemple précédent, on peut écrire :

```
Function SurfaceCercle( Rayon as Single)

     SurfaceCercle= 3.14*Rayon*Rayon

     Exit Function

End Function
```

Ici on utilise le nom de la fonction pour retourner le résultat, avec un signe '='.

Utilisez plutôt la méthode Return.

*Exit Function* permet aussi de sortir de la fonction, cela a le même effet que Return sauf que Return peut être suivi d'un argument de retour (et pas *Exit Function*).

## III-C-7 - Module standard

La Sub **AfficheSurfaceCercle** affiche le résultat dans le formulaire où elle est située.

Par contre la fonction **SurfaceCercle** est d'intérêt général, n'importe quelle procédure doit pouvoir l'appeler, de plus elle n'intervient pas sur les contrôles des formulaires et n'est donc pas liée aux formulaires.

On la placera donc dans un **module standard** qui est un module du programme qui ne contient que du code. (Pas d'interface utilisateur)

Pour créer un module standard Menu Projet>Ajouter un module.

Y mettre les procédures.

## III-C-8 - Private Public

Avant le mot *Sub* ou *Function* on peut ajouter :

*Private* indiquant que la procédure est accessible uniquement dans le module.

C'est donc une procédure privée.

Les procédures liées aux événements d'une feuille sont privées par défaut.

*Public* indiquant que la procédure est accessible à partir de toute l'application.

S'il n'y a rien devant Sub la procédure est publique

Exemple :

```
Private Function SurfaceCercle( Rayon as Single)

    Return 3.14*Rayon*Rayon

End Function
```

## III-C-9 - Remarques

Pour sortir d'une procédure Sub avant la fin, utiliser Exit Sub (Exit Function pour une fonction).

Quand vous appelez une procédure, il faut toujours mettre des parenthèses même s'il n'y a pas de paramètres.

```
FrmSplash.ShowDialog ()
```

Éventuellement on peut faire précéder l'appel du mot-clé Call, mais ce n'est pas obligatoire.

```
Call FrmSplash.ShowDialog ()
```

Nommage

Quand vous créez une procédure utilisez "la casse Pascal" pour créer les noms de routine:

la première lettre de chaque mot est une majuscule (c'est donc une convention).

```
Sub CalculTotal()
```

## III-C-10 - Lexique anglais=>français

Call = Appel.

Return= Retour.

Private= Privé.

Show= spectacle, exposition.

To show= montrer.

# III-D - Les modules

## III-D-1 - Qu'est-ce qu'un module ?

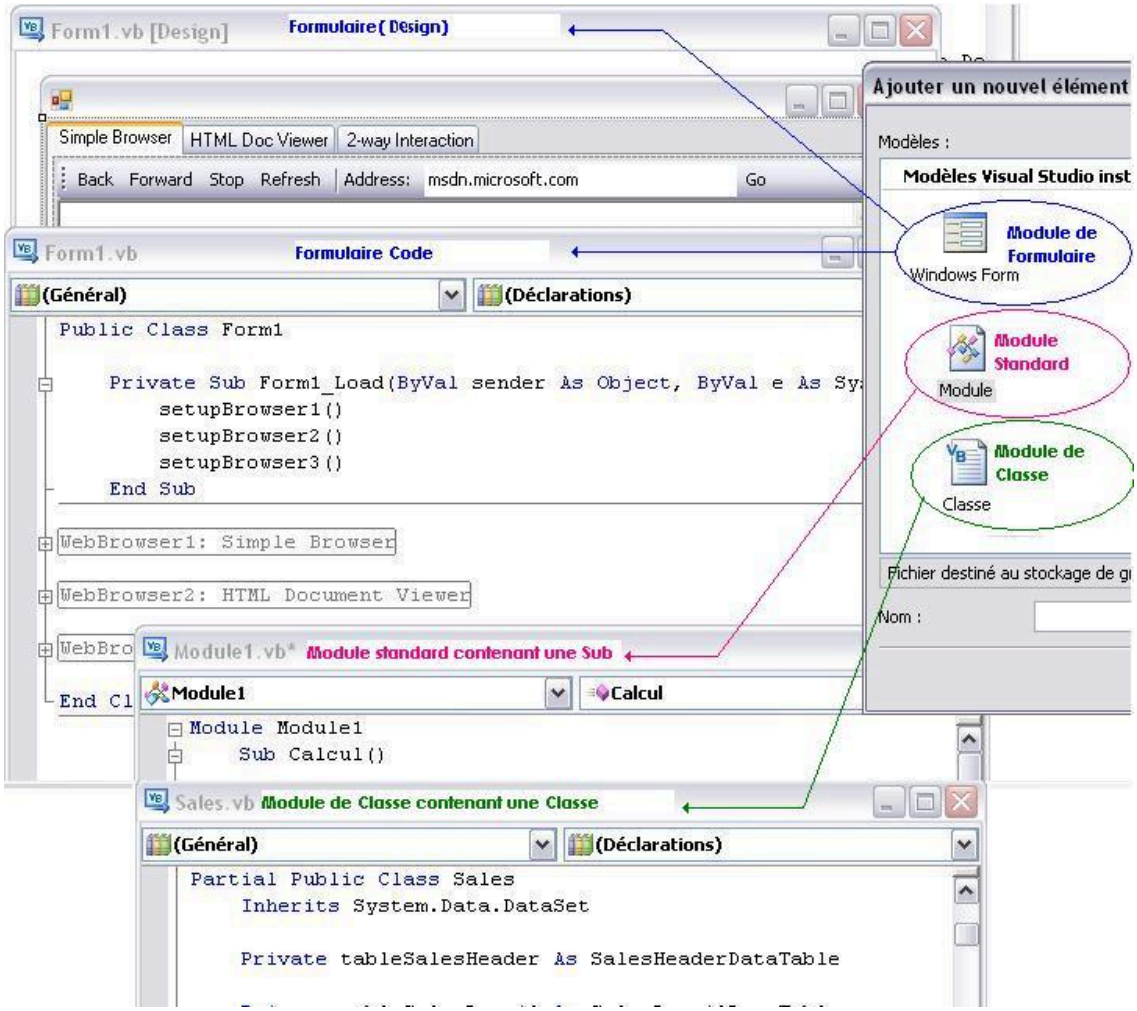
On a vu qu'un programme est décomposé en modules, chaque module contenant des procédures.

Chaque module correspond physiquement à un fichier '.vb'.

**Il existe**

- les modules de formulaire ;
- les modules standards ;
- les modules de 'Classe'.

### Comment se présentent-ils ?




### Un programme Visual Basic comporte donc :

- Les 'Modules de Formulaires' contenant :
  - le dessin des fenêtres de l'interface utilisateur (ou formulaire)contenant les contrôles (boutons, listes, zones de texte, cases à cocher…) ;
  - le code qui comprend :
  les procédures liées aux événements de la feuille (Button_Click…),
  les procédures indépendantes des événements. Ce sont des Sub() ou des Function().
  Exemple :

```
Class Form1         'Nom du Formulaire

Inherits System.Windows.Forms

Public A as String

    ……

Private Button1_Click 'Procédure liée à un événement

 …

End Sub
```

```
Sub MaRoutine          'Procédure indépendante
 ….
End Sub
End Class
```

**Un programme Visual Basic comporte donc :**

- les modules standards.
  Ils servent de stockage de procédures. Procédures "d'intérêt général".
  Ces procédures sont des Sub() ou des Function() qui peuvent être appelées à partir de n'importe quel endroit (pourvu qu'elles soient 'Public').
  Ils peuvent aussi servir à déclarer les objets ou déclarer les variables 'Public' qui seront utilisées donc accessibles par la totalité du programme.
  Exemple :

```
Module Module1         'Nom du Module
    Public A as String
    ……
 Sub MaRoutine         'Procédure indépendante
 ……
 End Sub
End Module
```

**Un programme Visual Basic comporte donc :**

- les modules de Classe.
  Ils ont vocation à fabriquer des objets, on verra cela plus loin (Chapitre sur la programmation objet).
  Exemple :

```
Class MaClasse            'Nom de la Classe
    Public A as String
    ……
End Class
```

**Un programme Visual Basic comporte donc :**



*On remarque que les Class, formulaires, Modules, Sub, Functions sont délimités par :*

*une ligne de début comportant le type et le nom du module ;*

*une ligne de fin contenant End et le Type.*

Exemple :

```
Module Module1         'Nom du Module
….
End Module
```

```
Sub MaRoutine        'Procédure

 ……

End Sub
```

## III-D-2 - Comment créer un module standard

Faire Menu Projet puis Ajouter un module. Donner un nom au module. C'est Module1.vb par défaut.

```
Module Module1        'Nom du Module

….

End Module
```

On remarque que le module est bien enregistré dans un fichier '.vb'.

Un module standard ne contient que du code.

**Comment ajouter une Sub dans un module Standard ?**

Taper Sub Calcul puis valider, cela donne :

```
Sub Calcul()

End Sub
```

**Remarque** Les Sub, Functions et Modules sont utilisés dans un type de programmation dite 'procédurale' où on découpe le code. Il existe un autre type de programmation dit 'Objet' ou on crée et on utilise des Objets, on verra cela plus tard.

## III-D-3 - Lexique anglais=>français

Return = Retour.

## III-E - Notion de programmation 'procédurale' et de programmation 'objet'

**Il y a deux manières de travailler en VB.NET.**

- En programmation '**Procédurale**'
  Chaque problème est décomposé en 'Fonctions'(Les Subs et Fonctions).
  La programmation structurée découpe les problèmes en fonctions (Sub et Function). Ce découpage s'il est systématiquement employé aboutit à la programmation fonctionnelle qui consiste en un emboîtement de fonctions que l'on peut voir comme des "boites noires" que l'on peut imbriquer les unes dans les autres.
  Chaque fonction contient du code VB qui permet d'effectuer le travail dévolu à la fonction.
  Ces fonctions sont stockées dans des modules standards (ou dans les modules de formulaire).
  Dans une application en programmation 'procédurale' il y a habituellement :
  des modules de formulaires ;
  des modules standard contenant des Sub et Function.
  N. B. j'utilisais, dans la précédente version du cours, le terme de programmation 'fonctionnelle' pour une programmation utilisant des Sub et Fonction. Dans Wikipedia la programmation fonctionnelle c'est autre chose aussi je parle maintenant de programmation 'procédurale'…
- En programmation '**Objet**'

On verra cela plus tard : on crée ses propres objets dans des modules de Classe, on utilise les membres (Propriétés et méthodes) de ces objets pour programmer.
Dans une application en programmation 'Objet' il y a habituellement :
des modules de formulaires ;
des modules de classe permettant de créer des Objets.
Grâce aux Classes (qui contiennent le code), on crée des objets.
Ensuite on utilise les propriétés et méthodes des objets.

*De toute façon, dans les 2 cas, que se soit dans des Sub ou des Classes, on utilise du code Visual Basic.*

La mode est à la programmation Objet !!

# IV - Environnement de développement : les EDI/IDE

## IV-A - IDE Visual Studio 2008 (Microsoft)

C'est l'**I**ntegrated **D**evelopment **E**nvironment (IDE): Environnement de développement intégré de Visual Basic Express 2008 de Microsoft. Il permet de dessiner l'interface (les fenêtres, les boutons, List, Image…) et d'écrire le code VB. Chez nous, on peut aussi dire **EDI** (Environnement de Développement Intégré).

L'IDE de Visual Basic 2008 est identique à celle de VB 2005, bien meilleur que celle de VB 2003 et l'Édition Express' (version légère par rapport à Visual Studio) est GRATUITE. Donc pas d'hésitation, chargez et utilisez VB Express 2008.

**Charger sur ce lien VB Express 2008**

Pour la version française, dans le cadre bleu 'Visual Basic Edition Express' dérouler la liste et choisir 'French' puis 'Download'.

Vous pouvez voir une vidéo sur l'IDE 2005 (c'est la même que pour la version 2008).

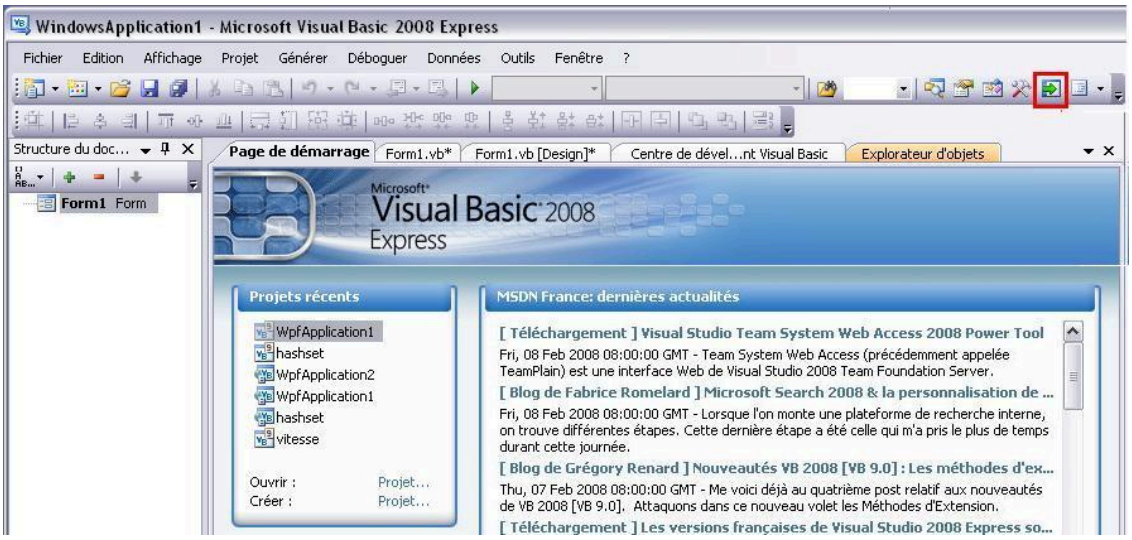

Voir la vidéo : **au format 'Flash'**> ou **au format 'Avi'** en Visual Basic 2005.

(En flash, il y a un arrêt au milieu : patientez. En Avi ne pas tenir compte des avertissements qui déclarent que le fichier n'est pas valide).

**Fenêtre Projet**

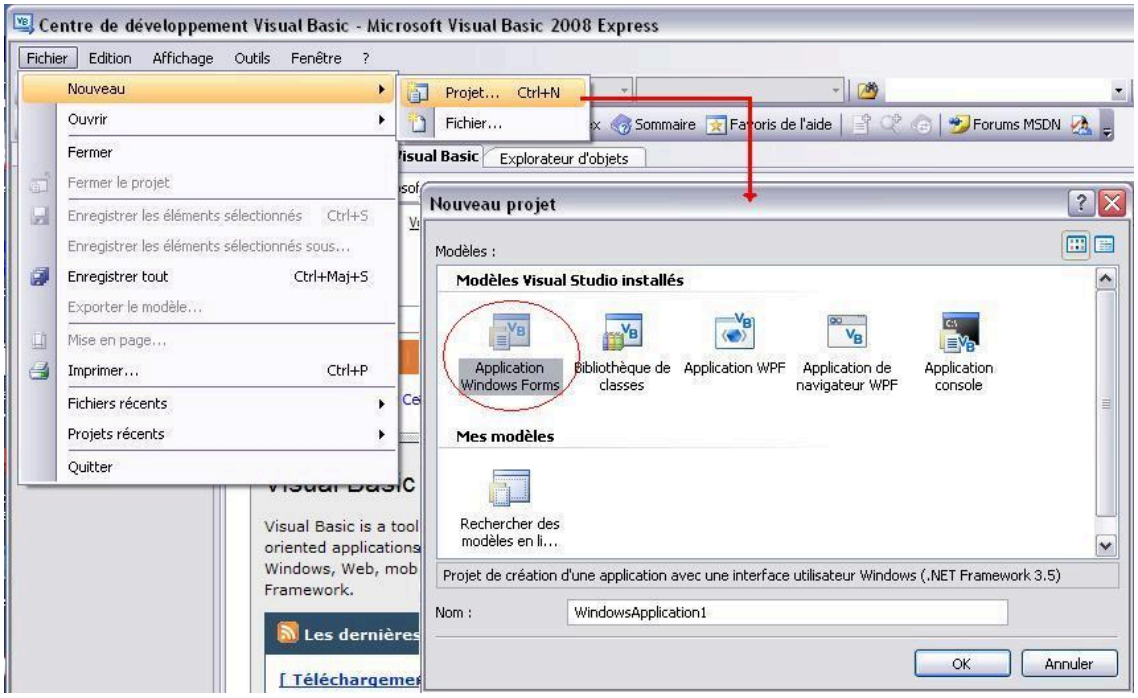
Quand on lance VB.net 2008, on ouvre l'IDE dans laquelle la fenêtre centrale charge la page du centre de développement Visual Basic de MSDN (site Microsoft), il faut être connecté à Internet.


En cliquant sur le bouton 'flèche verte' en haut à droite, on affiche la Page de démarrage "Start Page" qui permet d'ouvrir un projet existant Ouvrir (Recent Projects ou Open dans la version anglaise) ou de créer un nouveau projet :Créer (Create dans la version anglaise).




On constate que les diverses fenêtres sont accessibles par des onglets. L'IDE de VB 2008 diffère peu de celui de VB 2005.

Pour créer un nouveau projet Visual Basic, il faut choisir 'Créer' à gauche ou passer par le menu 'Fichier' puis 'Nouveau' puis 'Projet' . La fenêtre suivante s'ouvre :




On a le choix à partir de VB 2008 de créer l'interface utilisateur : en Windowsforms (basé sur GDI+), interface habituelle, bien connue ou en WPF interface vectorielle élaborée n'existant pas avant VB 2008.

## IV-A-1 - Interface 'Windows Forms'

Choisir l'icône 'Application Windows forms', puis donner un nom au projet, enfin valider sur 'OK'.

(Le chemin de l'emplacement du projet n'est pas modifiable ici, il est par défaut ' C:\Documents and Settings\Nom Utilisateur\Mes documents\Visual Studio 2008\ Projects\MonProjet'.)

On remarque qu'on aurait pu choisir 'Application WPF', on y reviendra.

**Dans un nouveau projet, créer ou ajouter une fenêtre 'WinForm'.**

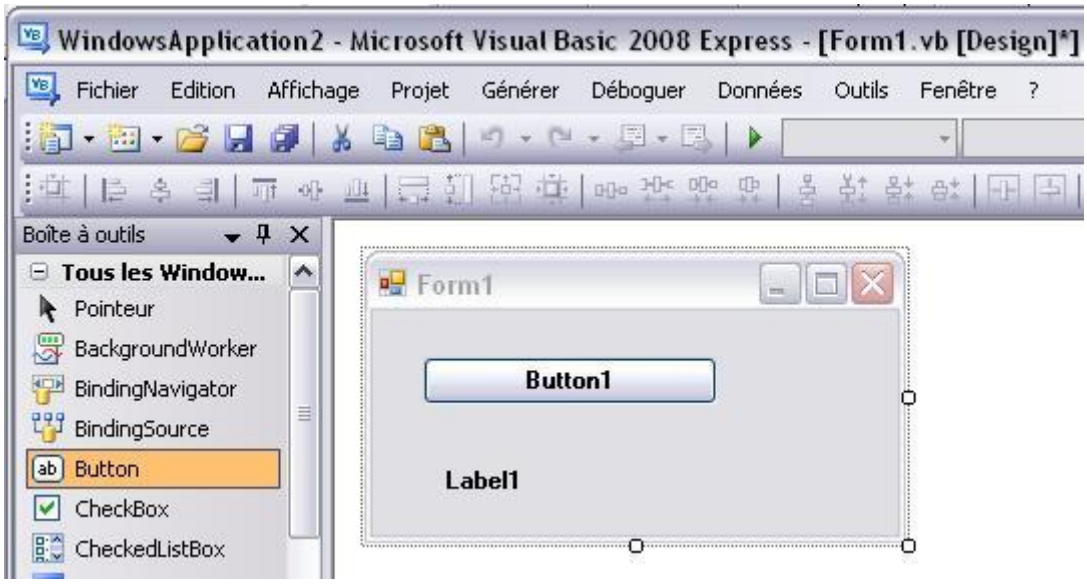
Pour ajouter une fenêtre (un formulaire) Menu Project, Ajouter un formulaire Windows ( 'Add a WindowsForms' en version anglaise ) :


Cliquer sur Windows Form, une fenêtre (un formulaire) Form2 vide apparait (Form1 était le nom du premier formulaire).

Il y a des fenêtres toutes faites pour accélérer le travail (les templates) comme les 'Écrans de démarrage' les 'Formulaire Explorateur'…

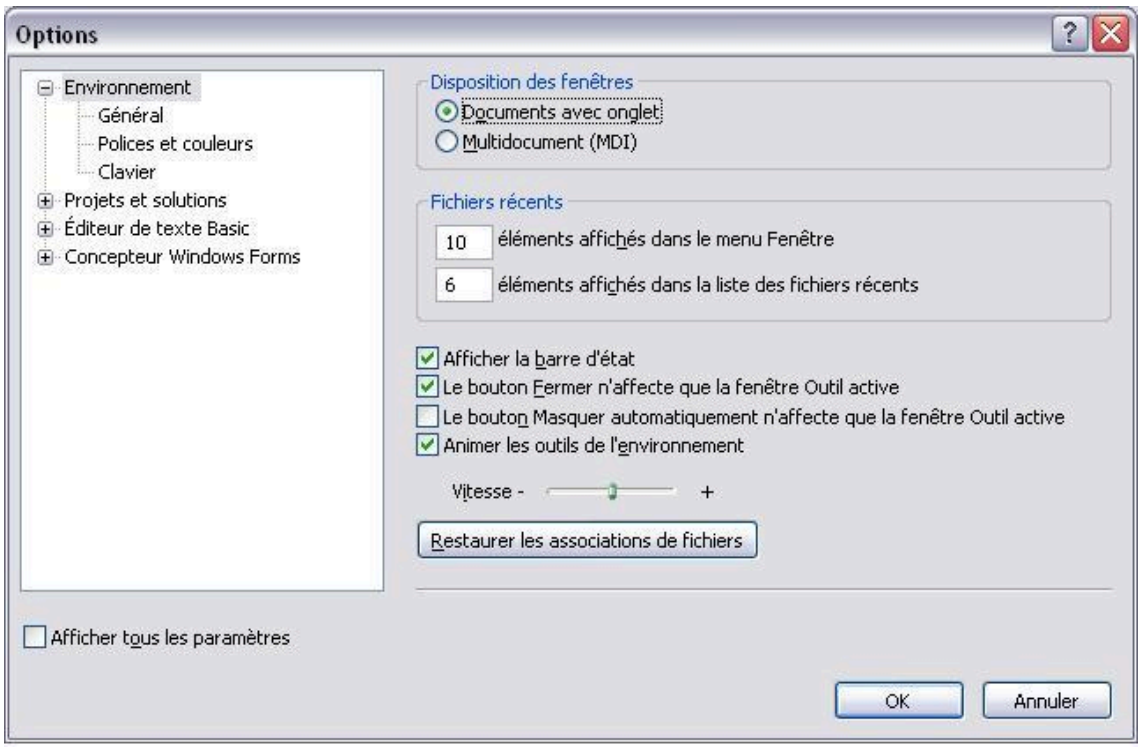
**Designer**

La zone de travail se trouve au centre de l'écran : c'est l'onglet Form1.vb[Design] ci-dessous qui donne donc accès au dessin de la feuille (du formulaire), on peut ajouter des contrôles, modifier la taille de ces contrôles…

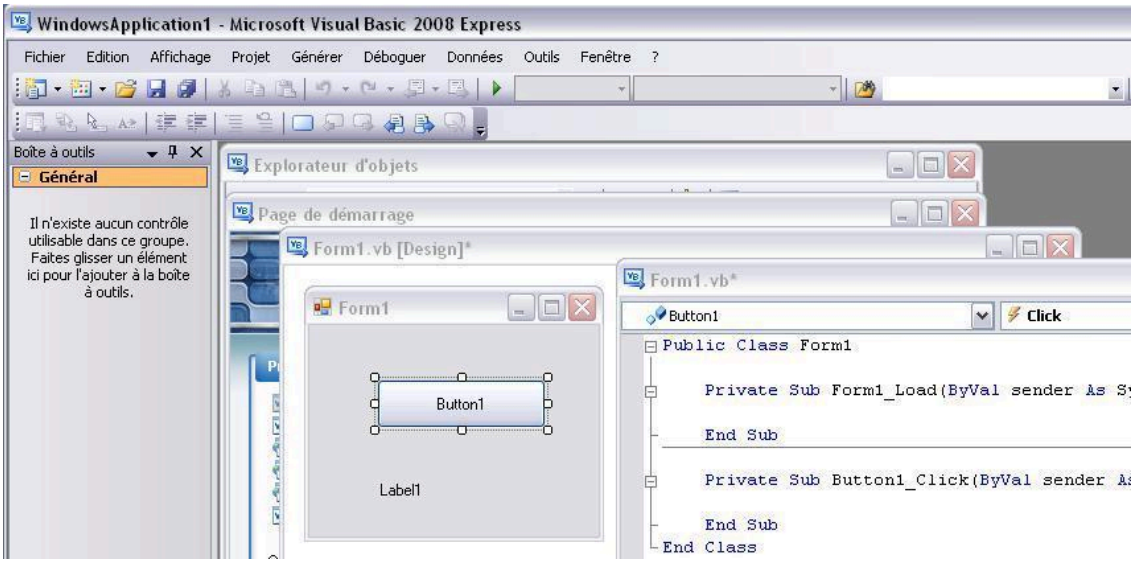



**On peut passer en mode 'Multidocument Mdi'** (comme en VB6) au lieu du mode 'Onglet'.

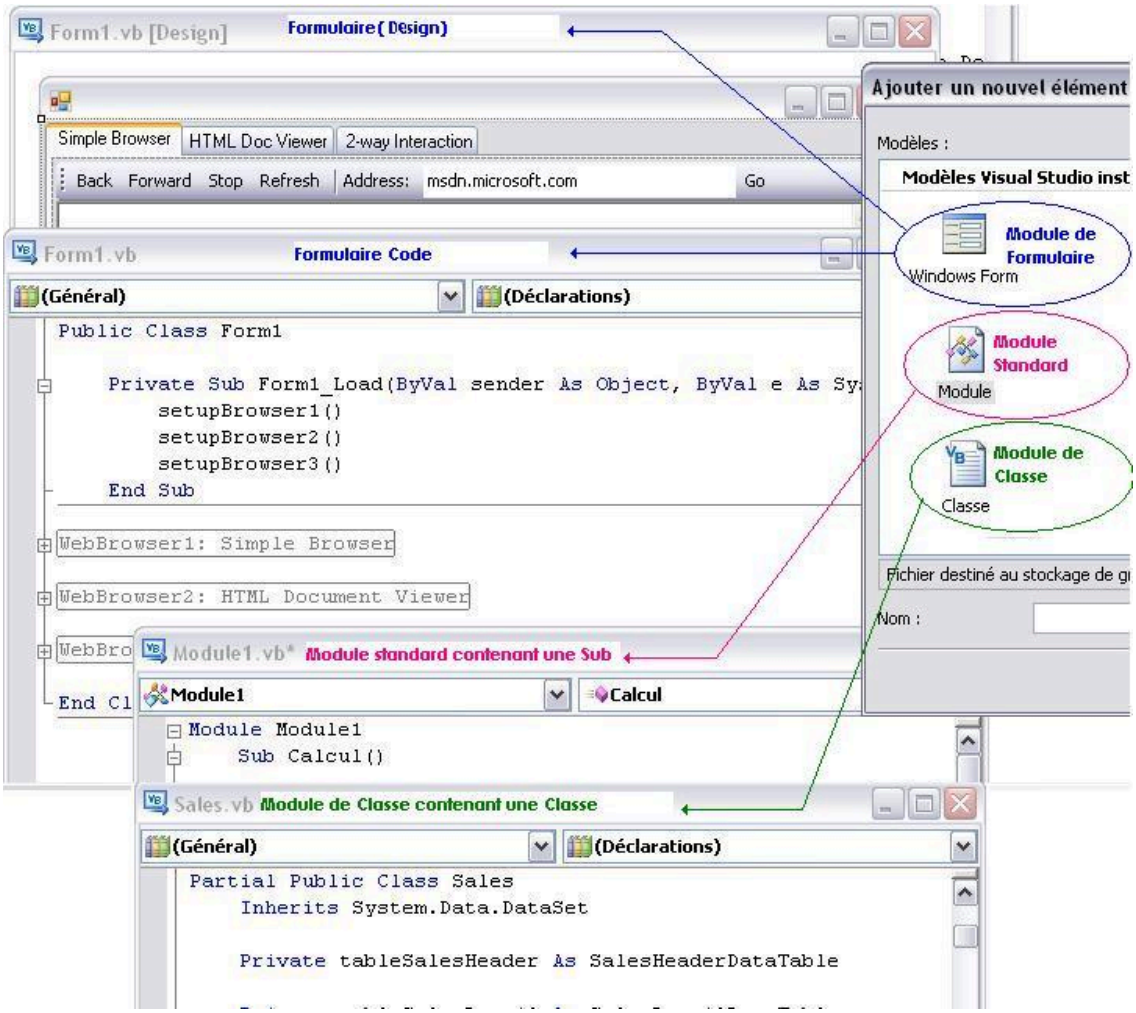
(Passer par le menu 'Outils' puis 'Options…' puis bouton 'Multidocument (Mdi)'.)




On obtient un mode multidocument avec plusieurs fenêtres.



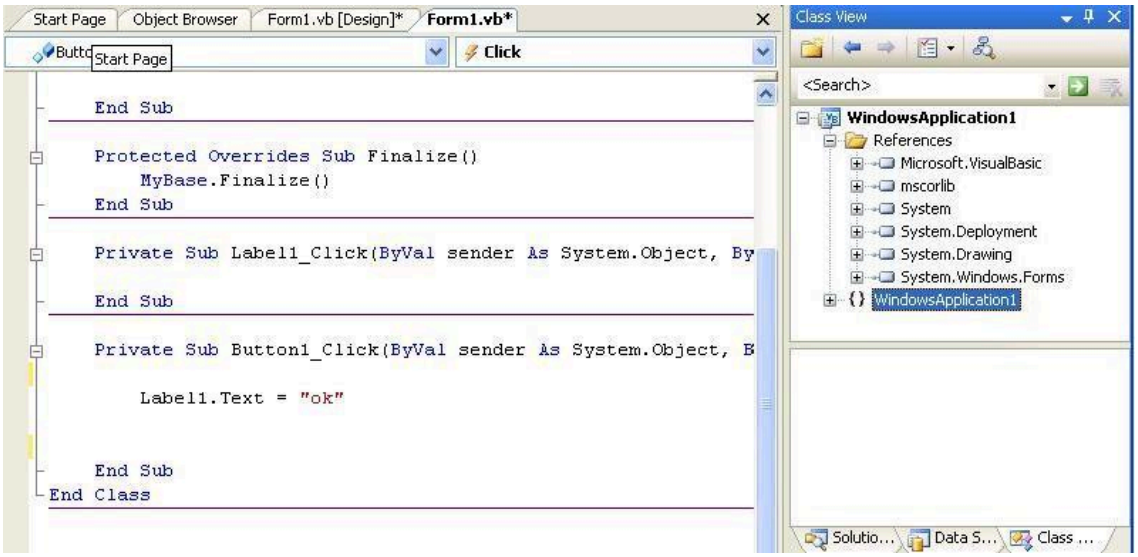

Exemple en mode Mdi montrant les 3 types de modules.




À noter que si on utilise le menu 'Projet' puis 'Ajouter…' cela permet d'ajouter un formulaire, un module standard, un module de Classe.

**Voir les procédures.**

L'onglet Form1.vb donne accès aux procédures liées à Form1.




On peut 'taper' du code dans les procédures.

La liste déroulante de gauche donne la liste des objets, celle de droite, les événements correspondants à cet objet.

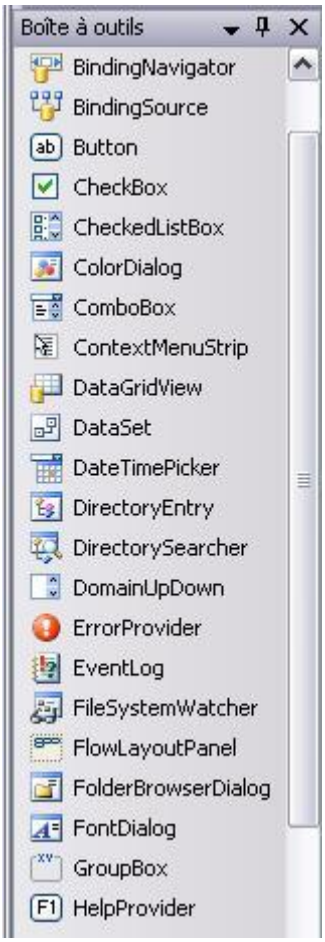
**Il est possible en double-cliquant dans le formulaire ou un contrôle de se retrouver directement dans le code de la procédure correspondant à cet objet.**

Ici on voit la procédure Button1_Click liée au Button1 de la fenêtre de Design.

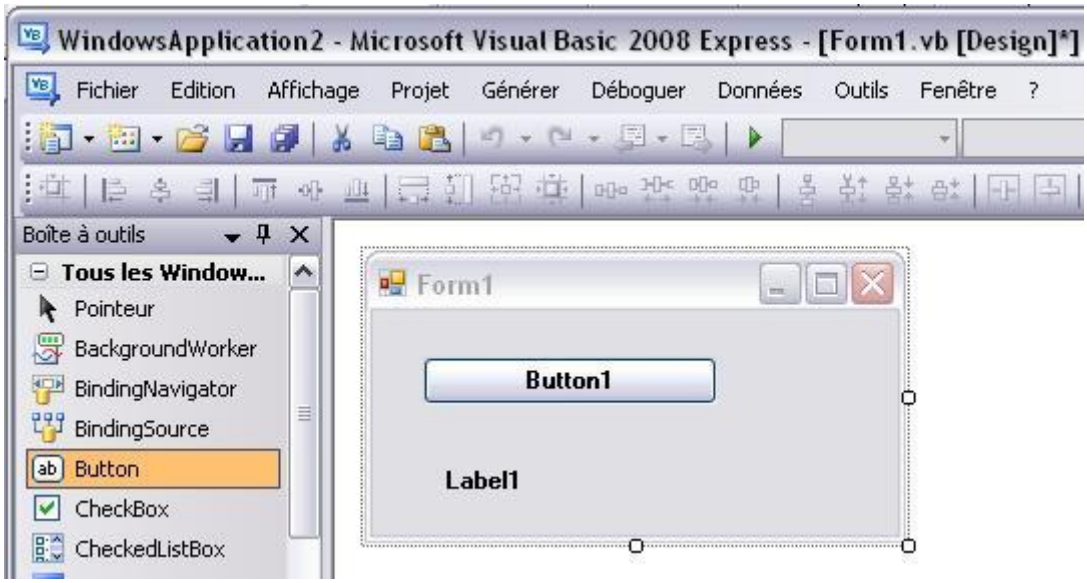
**Ajouter des contrôles au formulaire 'Winform'**

Ajouter un bouton par exemple :

Cliquer sur Boite à outils (Toolbox) à gauche, les contrôles apparaissent tous ou classés par ordre alphabétique.

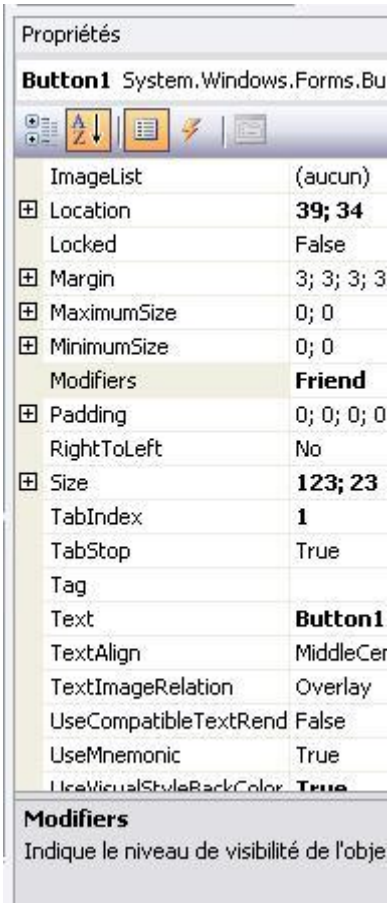



Cliquer sur 'Button' dans la boite à outils, cliquer dans la Form, déplacer le curseur sans lâcher le bouton, puis lâcher : un bouton apparait.




**Modifier les propriétés d'un contrôle ou du formulaire.**

Quand un formulaire ou un contrôle est sélectionné dans la fenêtre Design, ses propriétés sont accessibles dans la fenêtre de 'Propriétés' (Properties) à droite en bas : ici ce sont les propriétés du contrôle 'Button1' qui sont visibles (Text, Location…) on peut modifier directement les valeurs.




En bas de la fenêtre propriétés, il y a une explication succincte de la propriété sélectionnée (si elle n'apparait pas, clic droit sur la propriété puis dans le menu 'Description').

Exemple

Si au niveau de la ligne 'Text' des propriétés du bouton, j'efface 'Button1' et que je tape 'OK', dans le designer, le texte écrit sur le bouton deviendra 'OK'.

**Le déplacement des contrôles ou l'accès aux principales tâches est facile.**

La croix à gauche permet de déplacer le contrôle, la petite flèche à droite permet d'ouvrir un menu qui donne accès aux tâches les plus fréquentes.


**L'alignement automatique des contrôles**

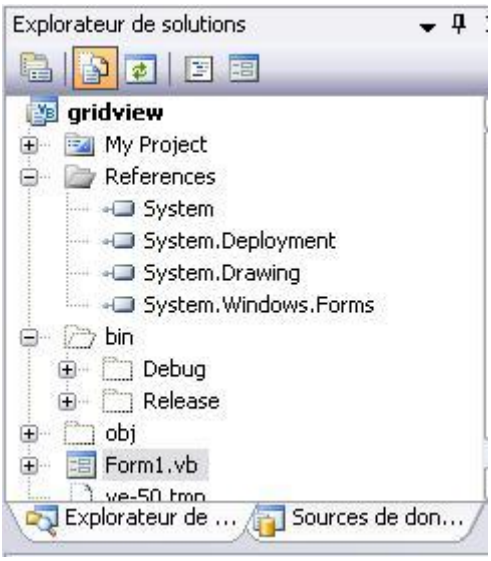
Si on modifie la taille ou l'emplacement d'un contrôle, VB signale par un trait bleu que le contrôle modifié et le contrôle voisin sont alignés :


**Renommer un nom : modification automatique**

On nomme cela '**Refactoring**': cliquer sur une variable, puis bouton droit, dans le menu cliquer sur 'Renommer'. Modifier le nom de la variable, valider. Dans toute la Classe la variable est renommée.

**Voir tous les composants d'un projet**

Pour cela il faut utiliser la fenêtre **Explorateur de solutions** en haut à droite, elle permet de voir et d'avoir accès au contenu du projet (pour voir tous les fichiers, il faut cliquer sur le deuxième bouton en haut) : gridview est le nom du programme.




**MyProjet :** double-cliquer dessus, vous ouvrirez la fenêtre 'propriétés du projet'.

**Références** qui contient les dll chargées. Pour atteindre les références, on peut aussi passer par le menu 'Projet' puis 'Propriétés' ou double-cliquer sur 'MyProjet' puis choisir l'onglet 'Références'.

**Form1.vb** est un formulaire (une fenêtre). Les formulaires, modules de classe ou standard sont tous des '.vb' Il suffit de double-cliquer dessus pour les ouvrir.

Si on ouvre la sous-liste de Form1.vb (en cliquant sur le '+'), on voit :

**Form1.Designer.vb** (qui montre le code qui crée le formulaire, on n'a pas à y toucher) ;

**Form1.resx** (le fichier de ressources).

Il suffit de cliquer sur la ligne Form1 dans l'explorateur de solution pour voir apparaitre la Form1 dans la fenêtre principale.

Si on clique sur un espace de noms dans la liste Références, cela montre l'arborescence des Classes.

### Tester son logiciel

On peut tester le projet grâce à :  lancer l'exécution avec le premier bouton (mode 'Run', le second servant à arrêter temporairement l'exécution (mode 'Debug'), le troisième à terminer l'exécution (Retour au mode 'Design' ou 'Conception').

Quand on est en arrêt temporaire en mode 'Debug', la ligne courante, celle qui va être effectuée, est en jaune :

```
For i=0 To 100
 Label1.Text=i.ToString
Next i
```

```
For i=0 To 100
 Label1.Text=i.ToString
Next i
```

Si on tape la touche F10 (exécution pas à pas), la ligne 'Label1.Text=i.ToString' est traitée et la position courante passe à la ligne en dessous.

En mode Debug, on peut modifier une ligne et poursuivre le programme qui tiendra compte de la modification (sauf pour les déclarations). On parle d''Edit and continue'.

La **sauvegarde du projet** se fait comme dans tous les logiciels en cliquant sur l'icône du paquet de disquettes.

On peut **compiler** le programme pour créer un exécutable par le menu Générer ('Build'). Le code présent dans l'IDE est le code **source**, après compilation le fichier exécutable contient du code **exécutable**.

### Projet

Dans la terminologie VB, **un projet** est une application en cours de développement.

Une '**solution**' (Team Project) regroupe un ou plusieurs projets (c'est un groupe de projets). Il n'y en a pas dans la version express.

En VB express on parle donc uniquement de projet, en fait ,VB crée aussi une solution de même nom.

### Fichiers, Chemins des sources

Si vous regardez dans ' C:\Documents and Settings\Nom Utilisateur\Mes documents\Visual Studio 2008\ Projects \MonProjet')les fichiers correspondant à un projet VB :
sous Windows 7) le programme est accessible dans 'Document/Visual Studio/Projects/Database/Database (Database étant le nom du programme). (En effet sous Windows 7 'Documents ans Settings' n'est pas accessible !! il faut passer par le répertoire 'Document' de l'utilisateur en cours.

**MonProjet.sln** est le fichier solution.(Pas de solution en VB express, que des projets.)

**MonProjet.psess** est le fichier de performance (pas toujours présent).

**MonProjet.suo** est le fichier de User solution.

Dessous existe un répertoire nommé aussi MonProjet qui contient:

**MonProjet.vbProj** le fichier de projet.

**Form1.vb** contient un formulaire et ses procédures.

**MyClasse.vb** contient par exemple des classes.

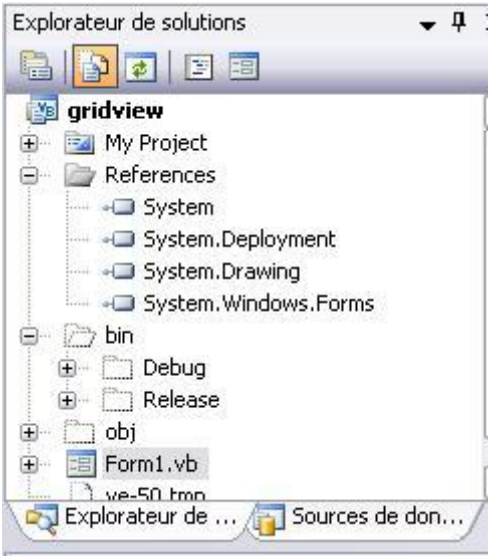
**Form1.Designer.vb** contient le code qui crée la fenêtre et les contrôles.

Il a encore les sous-répertoires \Bin, il y a aussi un répertoire \Obj et un répertoire \MyProjet

Si on compile le projet l'exécutable est dans un sous répertoire \Bin,

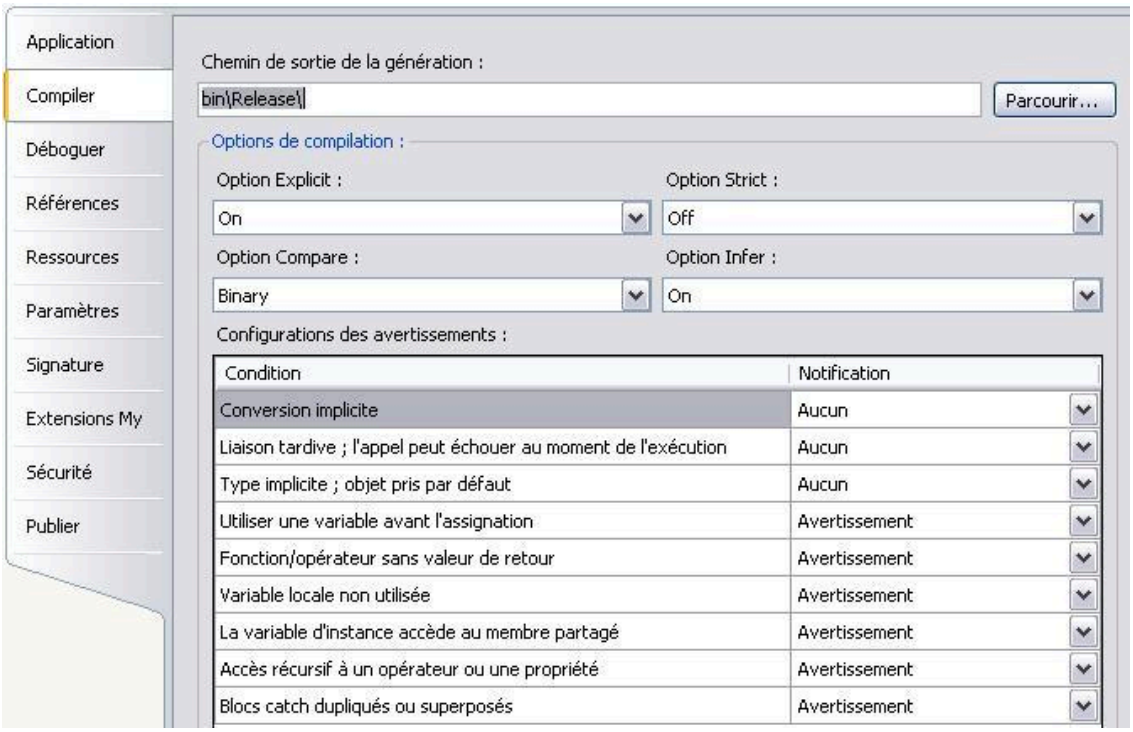
**Propriétés du projet**

Toutes les propriétés de l'application peuvent être modifiées dans le 'Projet Designer' (**Propriétés du projet**), pour l'atteindre, il faut double-cliquer sur 'My Project' dans l'explorateur de solutions :




Une autre manière d'ouvrir le 'Projet Designer' est de passer par les menus 'Projet' puis 'Propriétés de…'

On retrouve dans le projet designer :




Le nom de l'application, son icône, la fenêtre de démarrage, celle de fin. (Application)

Les Option Strict, Explicit compare et la nouvelle Option Infer. (Compiler).

Les références (dll liées au projet).

Les paramètres (valeurs liées à l'application).

Les ressources (texte, image, son) utilisées dans le programme.

La signature et la sécurité.

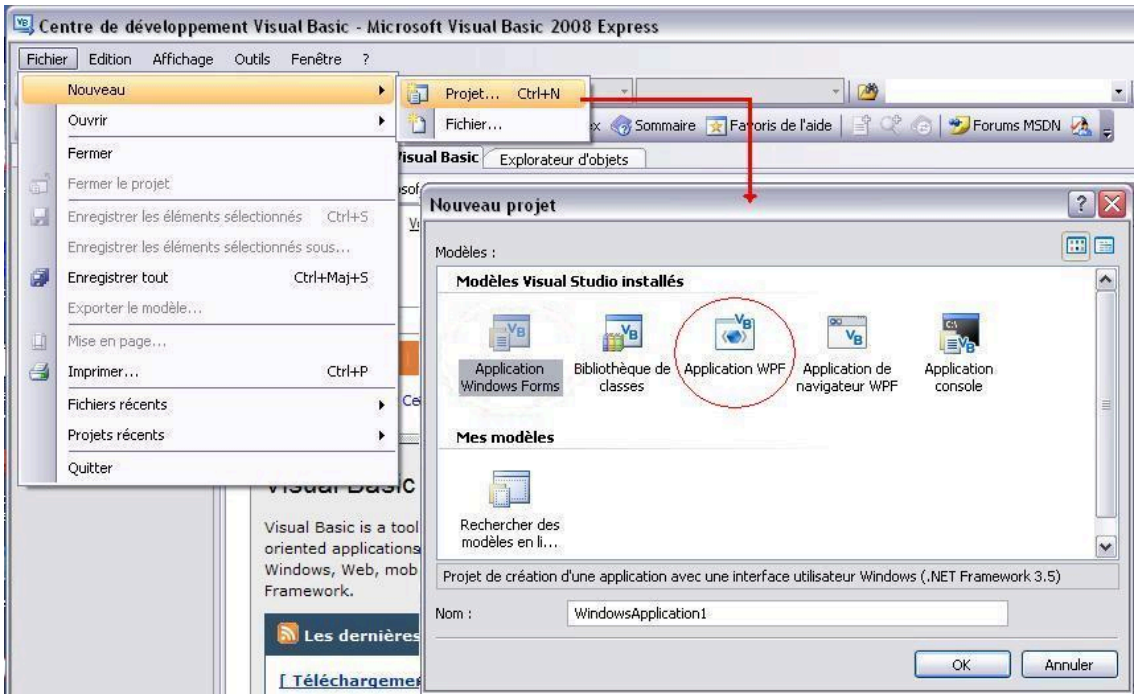
Les Extension My (nouveauté 2008).

Les paramètres relatifs à la publication (distribution et installation).

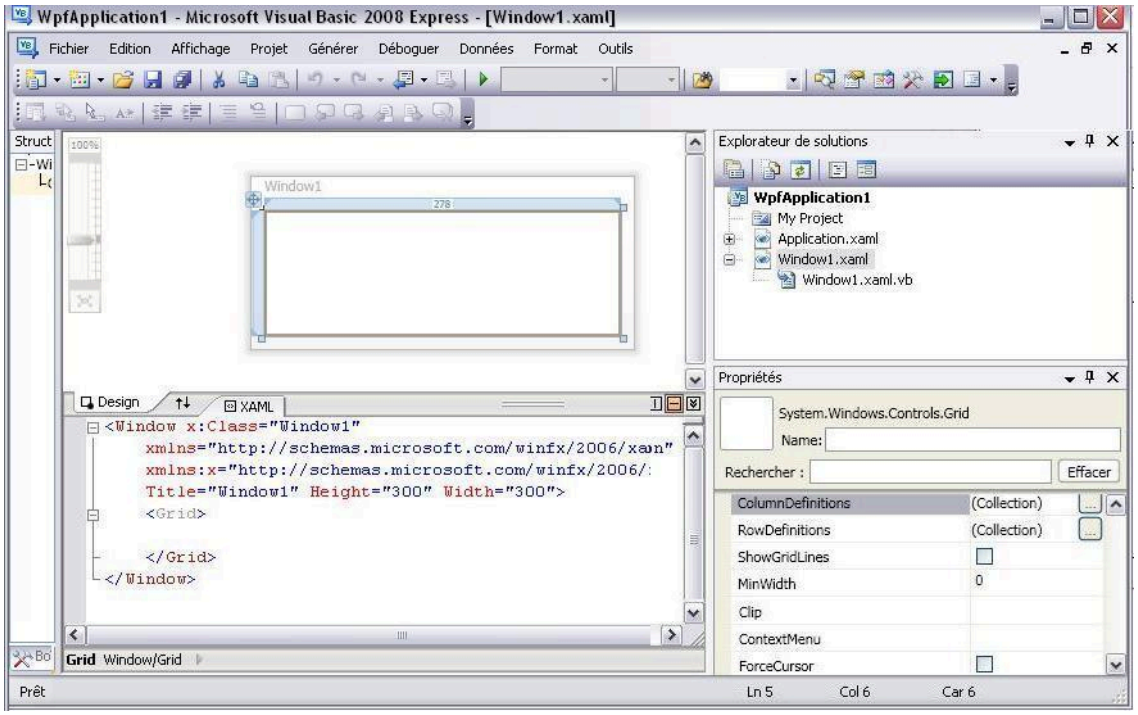
## IV-A-2 - Interface WPF

Plutôt que de travailler avec les Windows Forms (formulaire habituel utilisant GDI+), en VB 2008 on peut utiliser un mode graphique vectoriel extrêmement performant pour dessiner les formulaires et contrôles : pour cela on utilise les WFP (Windows Presentation Foundation).

Pour cela : menu 'Fichier', 'Nouveau', 'Projet'.




On choisit 'Application WPF', on se retrouve dans un nouvel environnement :

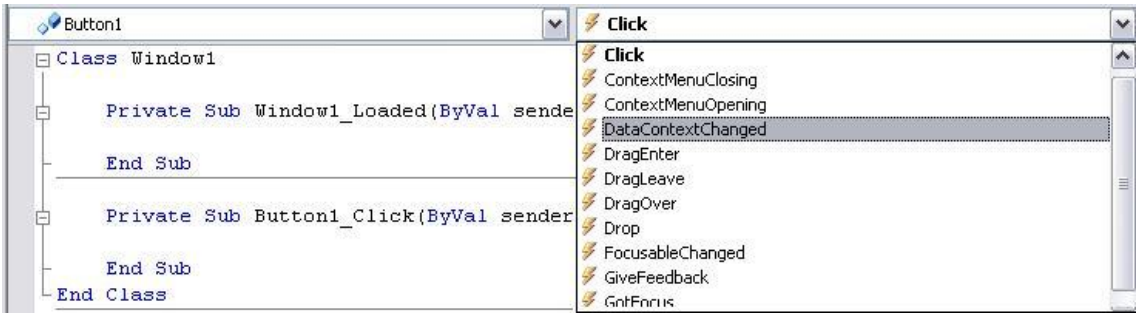



Les formulaires et contrôles sont différents de ceux des Windows Forms, ainsi que les propriétés des objets graphiques.

Il y a le 'designer' en haut qui permet de dessiner l'interface que verra l'utilisateur. Le designer génère un fichier XAML en bas qui décrit en XML l'interface.

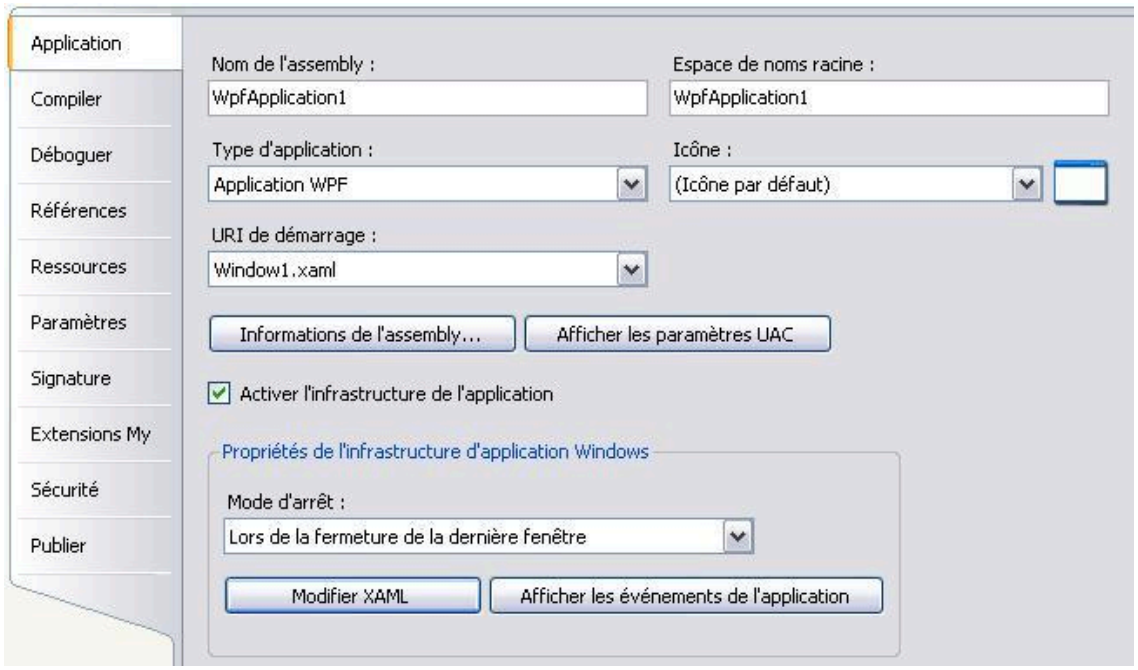
Dans la version Express, on peut dessiner des interfaces simples, les interfaces extrêmement élaborées (dégradé de couleur, animation…) peuvent être écrites en code XAML ou en utilisant un programme extérieur payant (Expression Blend). Voir le chapitre sir les WPF.

Si on double-clique sur un bouton, par exemple, on se retrouve dans la procédure événement correspondante :




On se rend compte que les événements là aussi ne sont pas les mêmes que pour les WindowsForm.

Il y a aussi d'autres modifications comme dans les propriétés du projet :



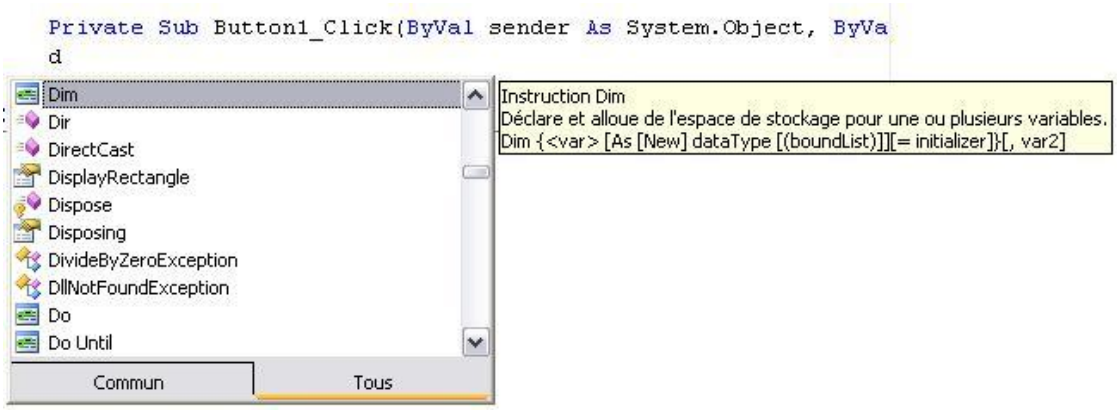

Voir le chapitre sur les WPF.

## IV-A-3 - Vb propose des aides

Quand on tape du code, VB affiche, des aides.

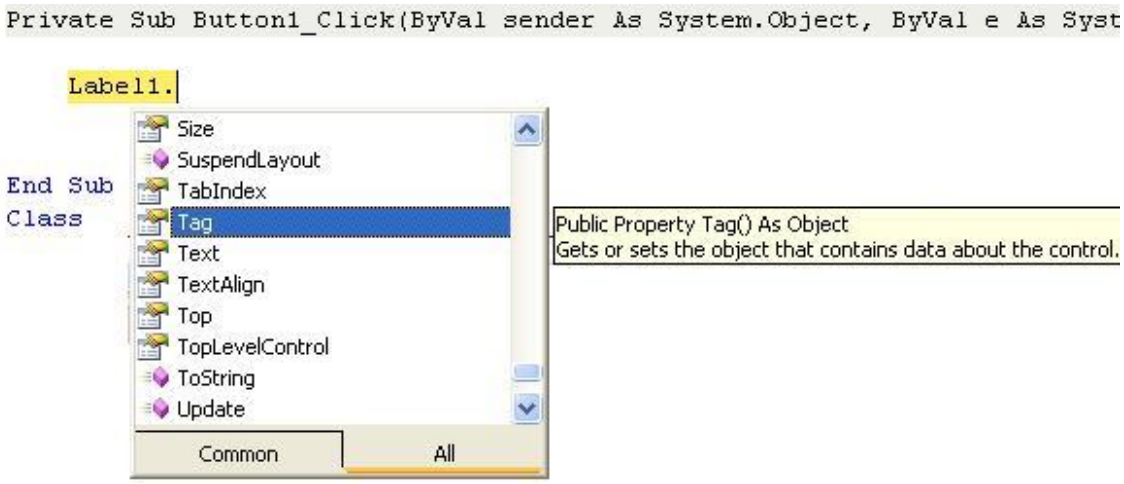
Dès que je tape une lettre VB propose dans une liste des mots.

Exemple, je tape 'd', il affiche 'Dim', 'Dir'… de plus si je me mets sur un des mots, il ouvre une petite fenêtre d'explication sur le mot avec sa syntaxe.




*VB permet de choisir dans une liste une des propriétés d'un objet.*

Exemple : je tape le nom d'un label nommé label1 puis je tape un point, cela me donne la liste des propriétés du label.




Quand je 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 par App

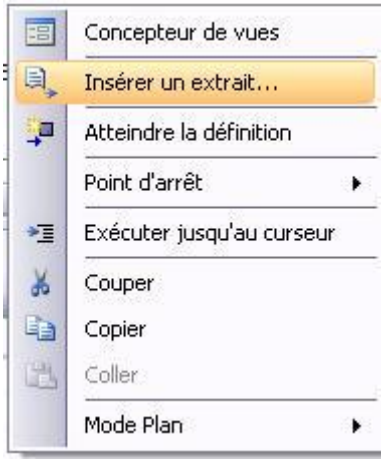
AppActivate

AppDomain

*VB fournit des exemples de code.*

Les **Extraits** (Snippets, bride, morceau de code) permettent d'insérer du code tout fait.

Dans le code d'une procédure, le clic droit de la souris ouvre un menu.




Cliquer sur 'Insérer un extrait' (Insert Snipper). Puis par menu successif vous obtiendrez le code que vous cherchez.

*Vb propose des solutions pour corriger les erreurs de code.*

Si je veux afficher une valeur numérique (avec option Strict=On), il y a erreur, VB me propose la correction :


*Il existe une abondante documentation.*

### Sur le Net: Msdn Framework 3.5

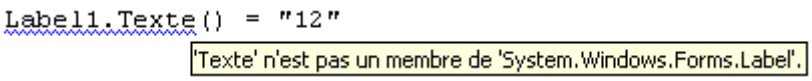
(http://msdn.microsoft.com/fr-fr/library/aa139616.aspx)

Dans l'IDE, 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. Enfin il est toujours possible de rechercher des informations par le menu '?'

```
Public Function Trim(ParamArray trimChars() As Char) As String
Public Function Trim() As String
 Supprime toutes les occurrences d'un jeu de caractères spécifié dans un tableau à partir du début et de la fin de cette instance.
```

**Erreur dans l'écriture du code.**

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 où se trouve l'erreur.



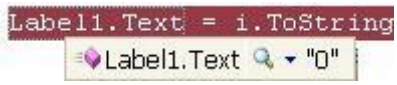

Ici la propriété 'Text' a été mal orthographiée.

Si je lance le programme en mode 'Run' et qu'il y a des erreurs, Vb me le signale et répertorie les erreurs dans la liste des tâches en bas. Vb propose des solutions pour corriger les erreurs de code. (Voir plus haut.)

**Mode débogage (mode BREAK)**

Une fois lancée l'exécution (F5), puis stoppée (par Ctrl +Alt +Pause ou sur un point d'arrêt), on peut

- voir la valeur d'une propriété d'un objet en le pointant 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.

**F8** permet l'exécution pas à pas (y compris des procédures appelées).

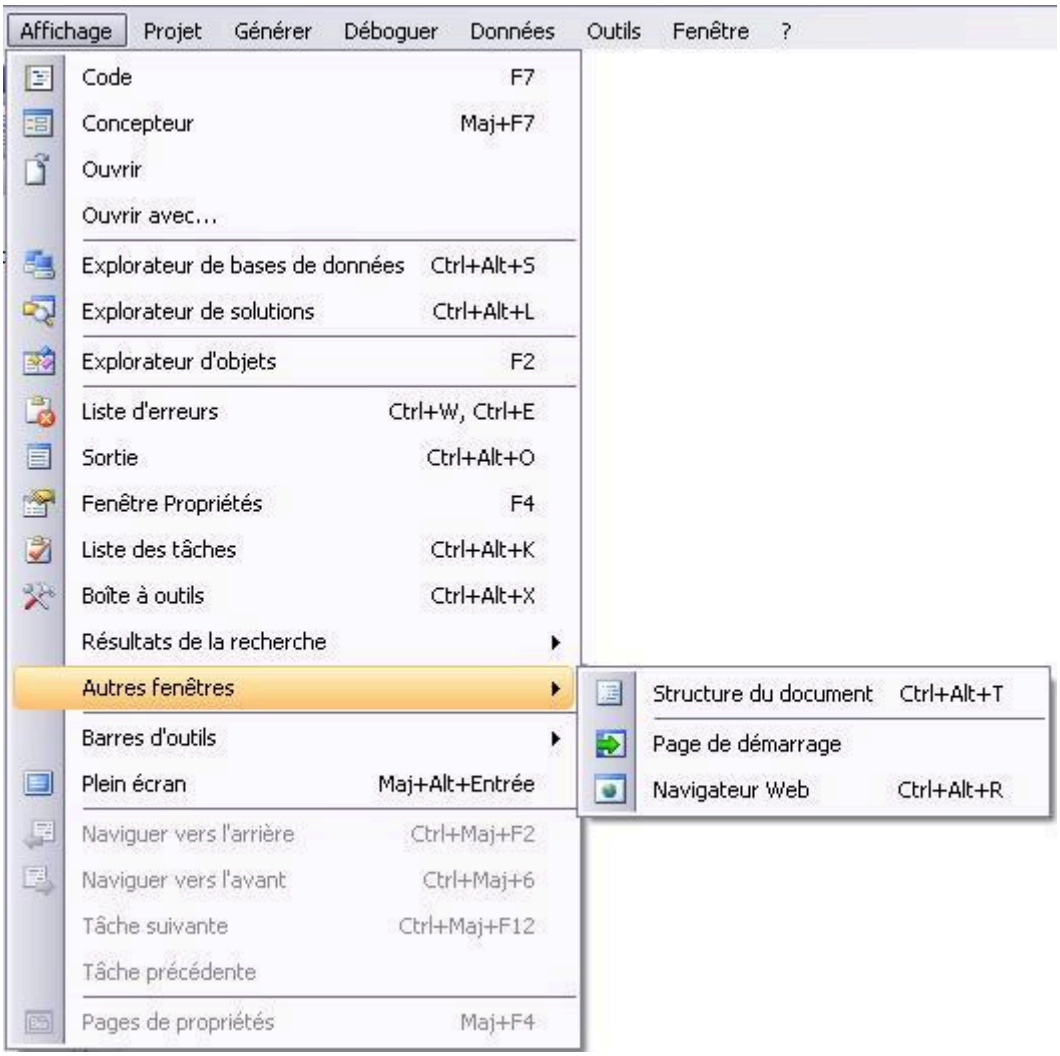
**F10** permet le pas à pas (sans détailler les procédures appelées).

**Maj+F11** exécute jusqu'à la fin de la procédure en cours.

En cliquant sur le bouton droit de la souris, on peut exécuter jusqu'au curseur (Run To Cursor), voir la définition, la déclaration de ce qui est sous le curseur (Atteindree la définition : Go To Definition)…


Il y a un chapitre sur le débogage pour apprendre à trouver les erreurs de code.

**On peut grâce au menu 'Affichage' avoir accès à plein de choses :**




## IV-B - Visual Basic 2010 Express

C'est l'**I**ntegrated **D**evelopment **E**nvironment (IDE) : Environnement de développement intégré de Visual Basic Express 2010 de Microsoft. Il permet de dessiner l'interface (les fenêtres, les boutons, List, Image…) et d'écrire le code VB. Chez nous, on peut aussi dire **EDI** (Environnement de Développement Intégré).

L'IDE de Visual Basic 2010 est similaire à celle de VB 2005 et VB 2008.VB 2010 Express et est **GRATUIT**. Donc pas d'hésitation, chargez et utilisez VB Express 2010.

Où trouver Visual Basic 2010 Express ?
Cliquer sur le lien :
**http://www.microsoft.com/express/downloads/** Dans la liste de liens, cliquer sur 'Visual Basic Express 2010'
Puis dans la liste 'Select language', choisissez "French", une fenêtre pop-up démarre.

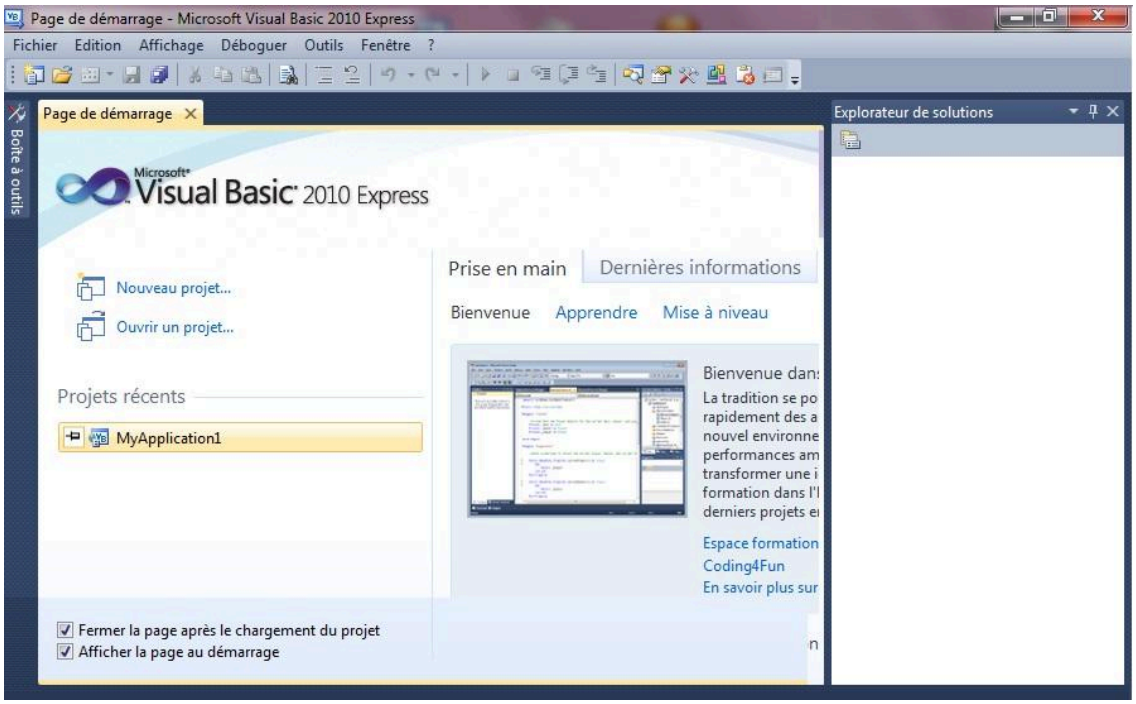
Est-il possible d'utiliser les éditions Express à des fins commerciales ?
Oui. Il n'y a aucune restriction liée aux licences pour les applications créées à l'aide des éditions Express.
Cette réponse (pour VB express 2008) est indiquée sur le site de Microsoft : http://msdn.microsoft.com/fr-fr/express/default.aspx

Vb 2010 utilise le Framework 4 : **Voir la documentation Msdn du Framework 4**

Mais on peut choisir d'utiliser le Framework 2, 3, 3.5, 4 : menu 'Projet', 'Propriété de…', onglet 'Compiler', en bas liste:'Framework cible'.

## Page de démarrage

Quand on lance VB.net 2010, on affiche la **Page de démarrage**.



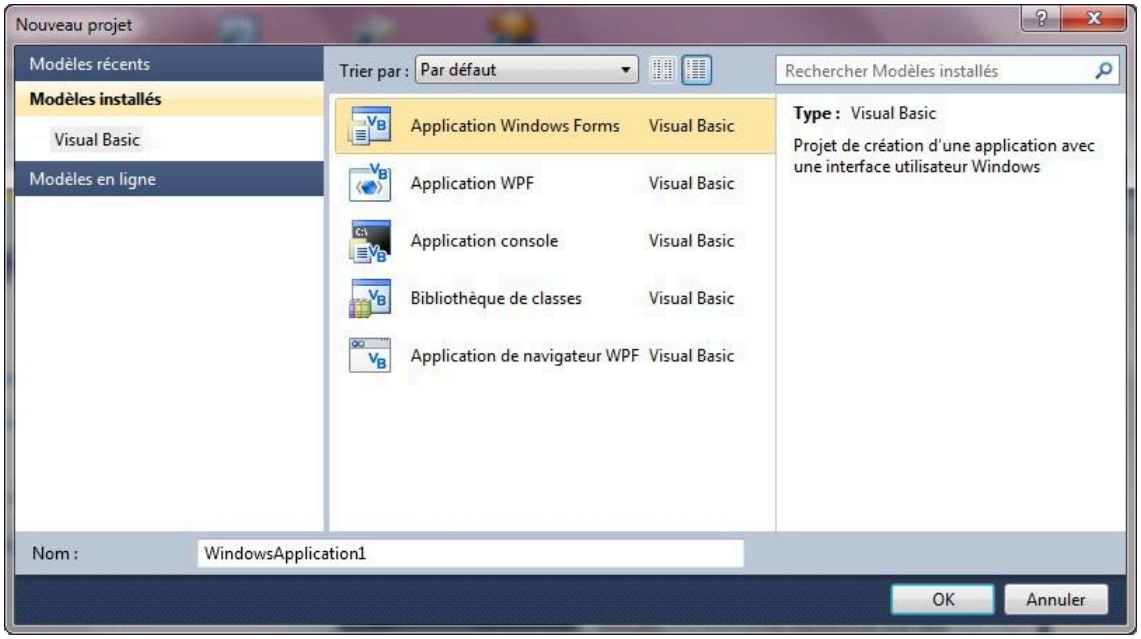

On a le choix entre :
- Nouveau projet… ;
- Ouvrir un projet… ;
- Projets récents.

Quand le pointeur est sur un élément de la liste Projets récents, ce dernier est mis en surbrillance et une icône de **punaise** s'affiche. Cliquer sur la punaise "épingle" le projet à la liste, afin qu'il reste dans sa position actuelle ultérieurement.
Si vous ouvrez plein de projets, votre projet punaisé restera visible dans la liste.

En bas de la page de démarrage, il y a 2 options pour cette page :
"Fermer la page après le chargement du projet" ;
"Afficher la page au démarrage".

## Créer un nouveau projet

Pour créer un nouveau projet Visual Basic, il faut choisir 'Nouveau projet' dans le menu démarrage ou passer par le menu 'Fichier' puis 'Nouveau Projet'. La fenêtre suivante s'ouvre :




Il faut choisir 'Application Windows Forms' ou 'Application WPF'.
On peut aussi choisir 'Modèle en ligne' à gauche pour avoir une liste (courte) de modèle de programme.

On a donc le choix (à partir de VB 2008) de créer l'interface utilisateur : en Windowsforms (basé sur GDI+), interface habituelle, bien connue ou en WPF interface vectorielle élaborée n'existant pas avant VB 2008.

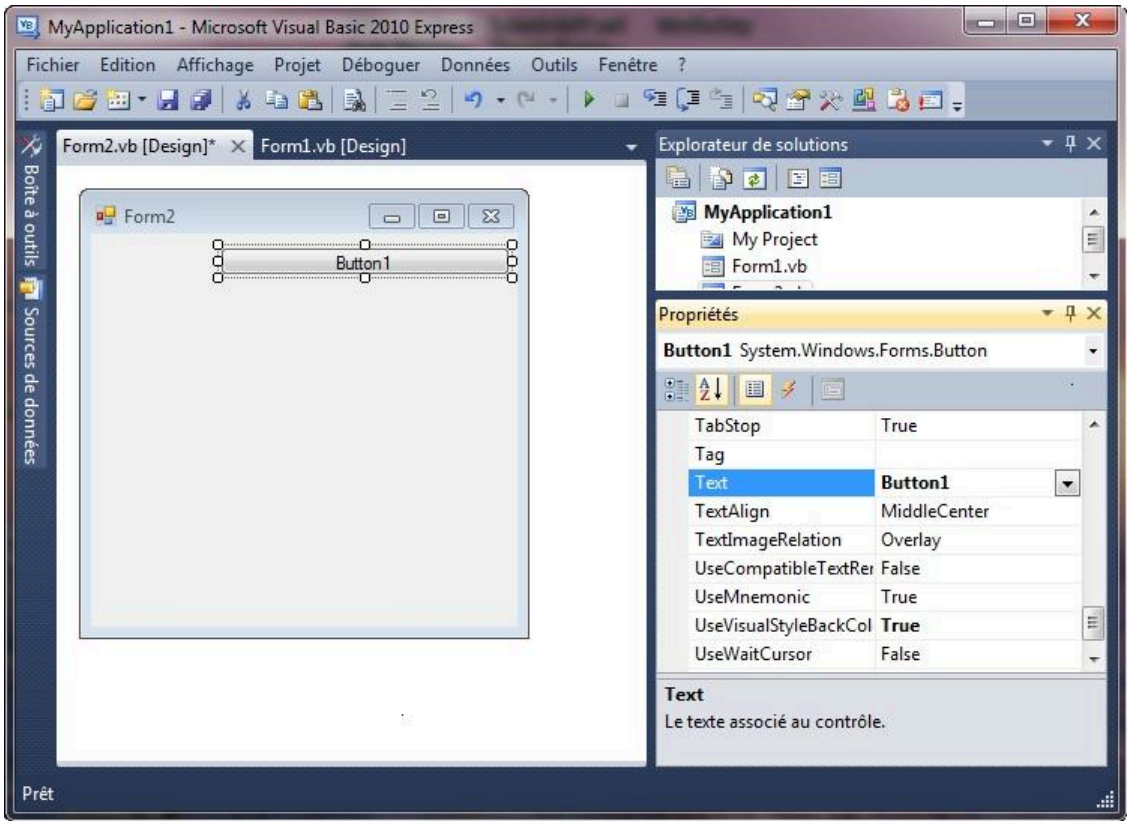
## IV-B-1 - Interface 'Windows Forms'

Choisir l'icône 'Application Windows forms', puis donner un nom au projet, enfin valider sur 'OK'.

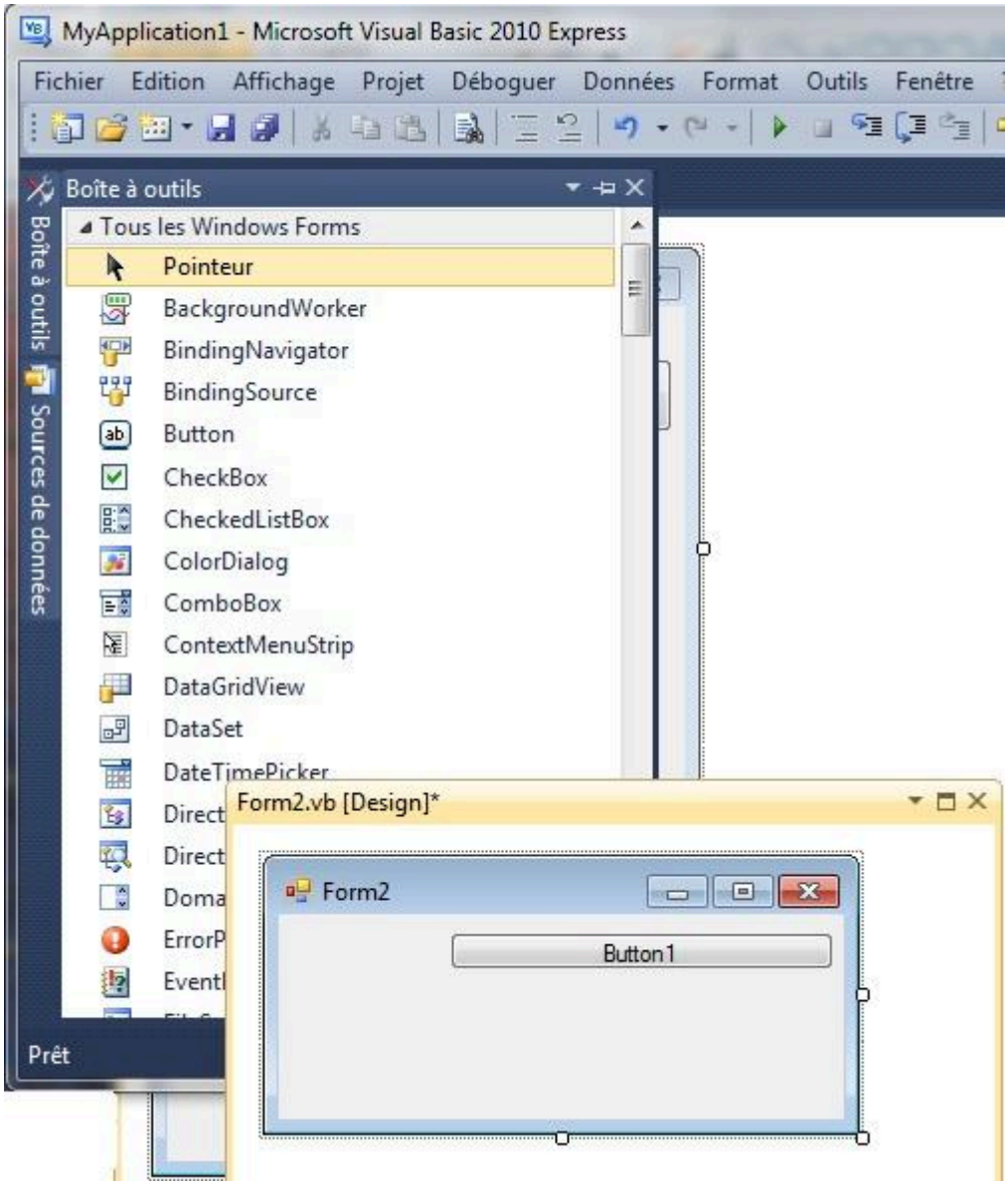
(Le chemin de l'emplacement du projet n'est pas modifiable ici, il est par défaut ' C:\Documents and Settings\Nom Utilisateur\Mes documents\Visual Studio 2010\ Projects\MonProjet')
C:/Utilisateurs/Philippe/Mes document/ Visual Studio 2010/Projet sous Windows 7
Avec l'explorateur : Documents=> Visual Studio 2010=>Projet.

On remarque qu'on aurait pu choisir 'Application WPF', on y reviendra.

# IV-B-1-a - Fenêtre Projet

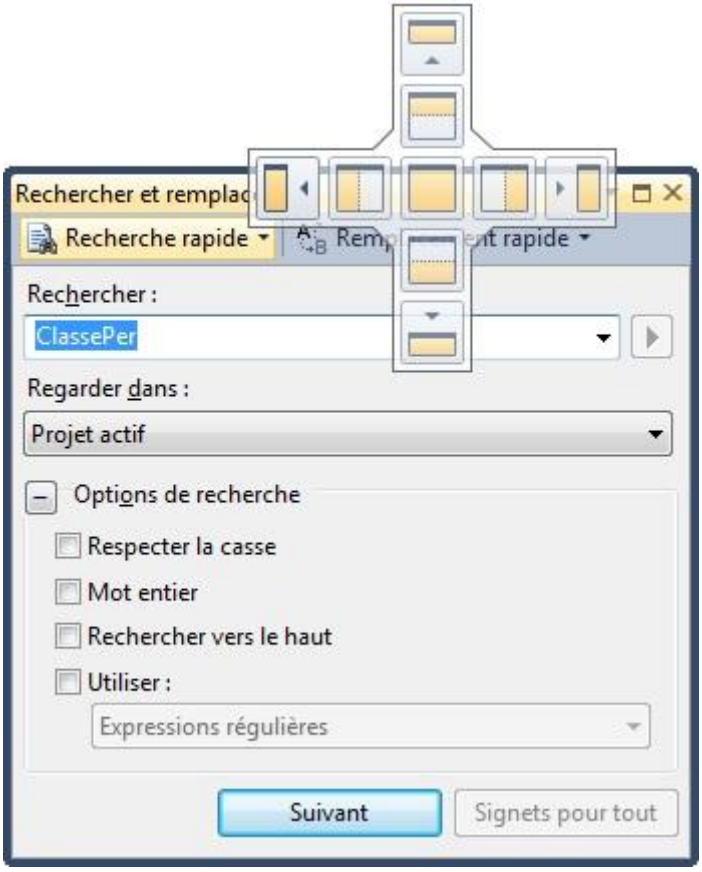



On constate que les diverses fenêtres (pour chaque Form du projet ou chaque module de code) sont accessibles par des onglets.
À droite en haut il y a la fenêtre 'Explorateur de solution' ou se trouve les divers éléments du projet.
À droite en bas il y a la fenêtre 'Propriétés' contenant les propriétés de l'objet pointé à droite.
(Si on est sur un bouton à droite, ce sont les propriétés du bouton qui sont à gauche).




Vous pouvez ancrer les fenêtres correspondant aux onglets aux extrémités de la fenêtre de l'IDE ou les déplacer n'importe où sur le Bureau (sur un second moniteur aussi).
Ci-dessus la fenêtre de Form2 est détachée de l'IDE (il suffit de cliquer déplacer l'onglet).

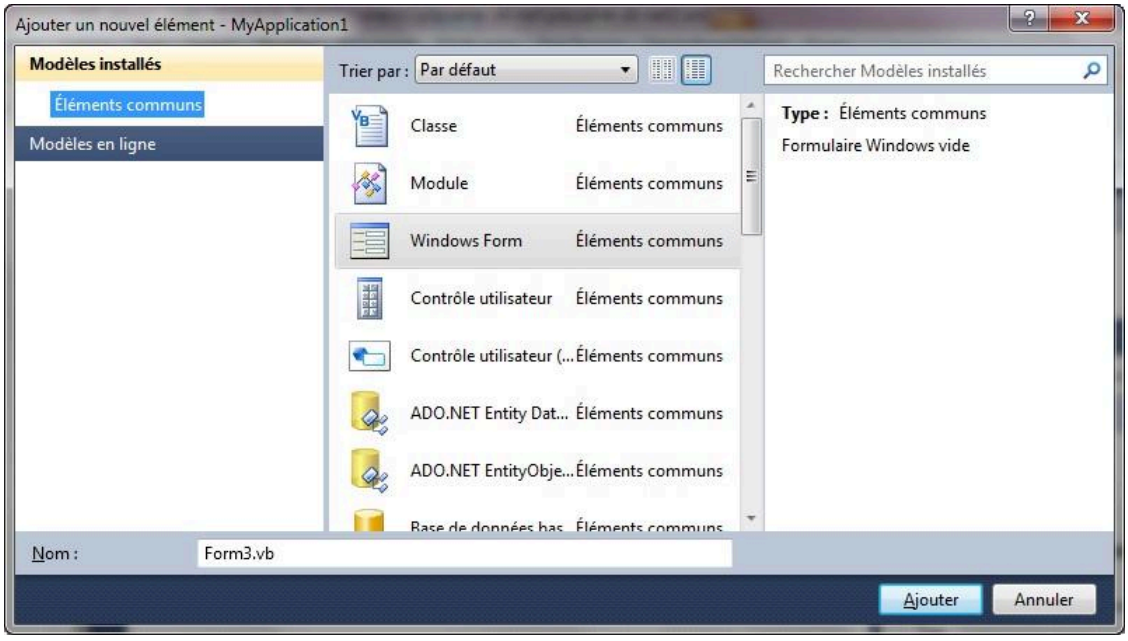
Quand on déplace une fenêtre de l'IDE, une 'croix' s'affiche, il suffit de déplacer le curseur dans un élément de la croix qui symbolise une position dans l'IDE (en haut en bas, à droite…) et de lâcher le bouton de la souris pour que la fenêtre se positionne en haut, en bas, à droite… dans l'IDE.




## IV-B-1-b - Créer ou ajouter une fenêtre 'WinForm'

Dans un nouveau projet, créer ou ajouter une fenêtre 'WinForm'.

Pour ajouter une fenêtre (un formulaire) Menu 'Project', 'Ajouter un formulaire Windows' ( 'Add a WindowsForms' en version anglaise) :



Cela permet d'ajouter un formulaire, mais aussi un module standard, un module de Classe.

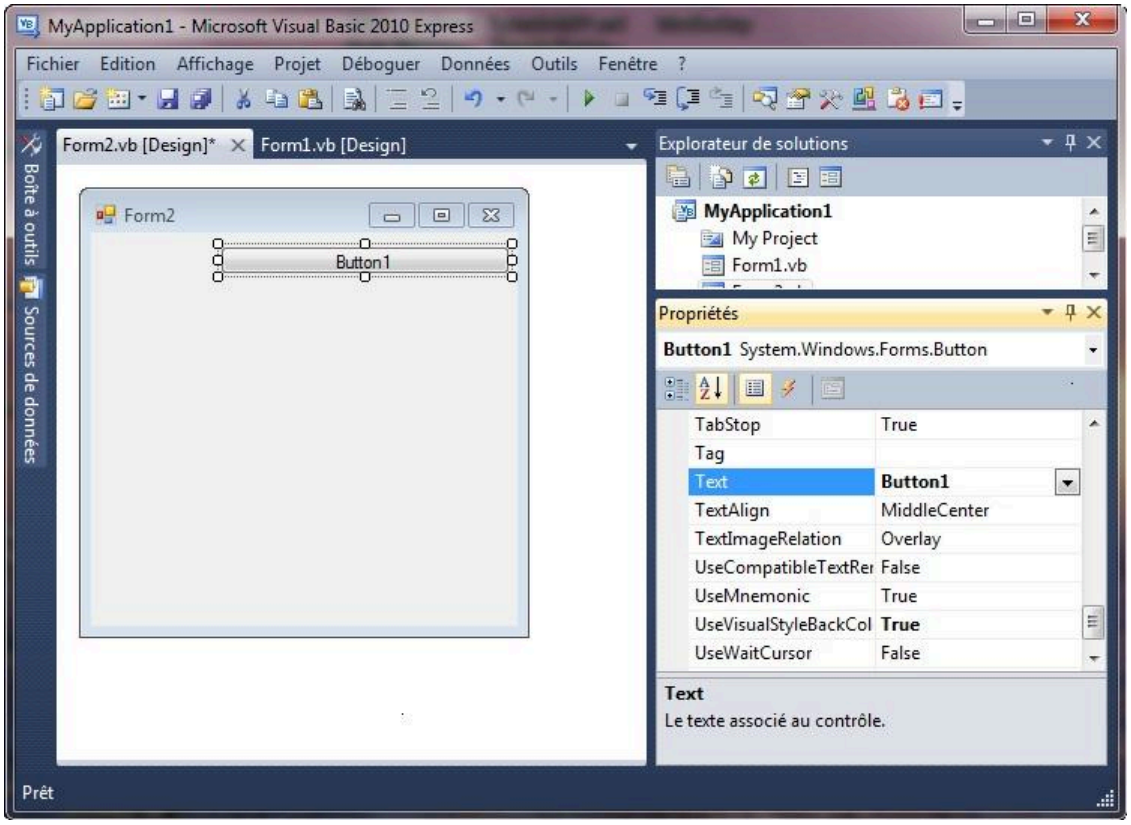
Cliquer sur Windows Form, une fenêtre (un formulaire) Form2 vide apparait (Form1 était le nom du premier formulaire).

Il y a des fenêtres toutes faites pour accélérer le travail (les templates) comme les 'Ecran de démarrage' les 'Formulaire Explorateur'…

## IV-B-1-c - Le concepteur (Designer)

C'est la zone permettant de dessiner l'interface utilisateur : les fenêtres, controles…

La zone de travail se trouve au centre de l'écran : c'est l'onglet **Form1.vb[Design]** ci-dessous qui donne donc accès au dessin de la feuille (du formulaire); on peut ajouter des contrôles, modifier la taille de ces contrôles…

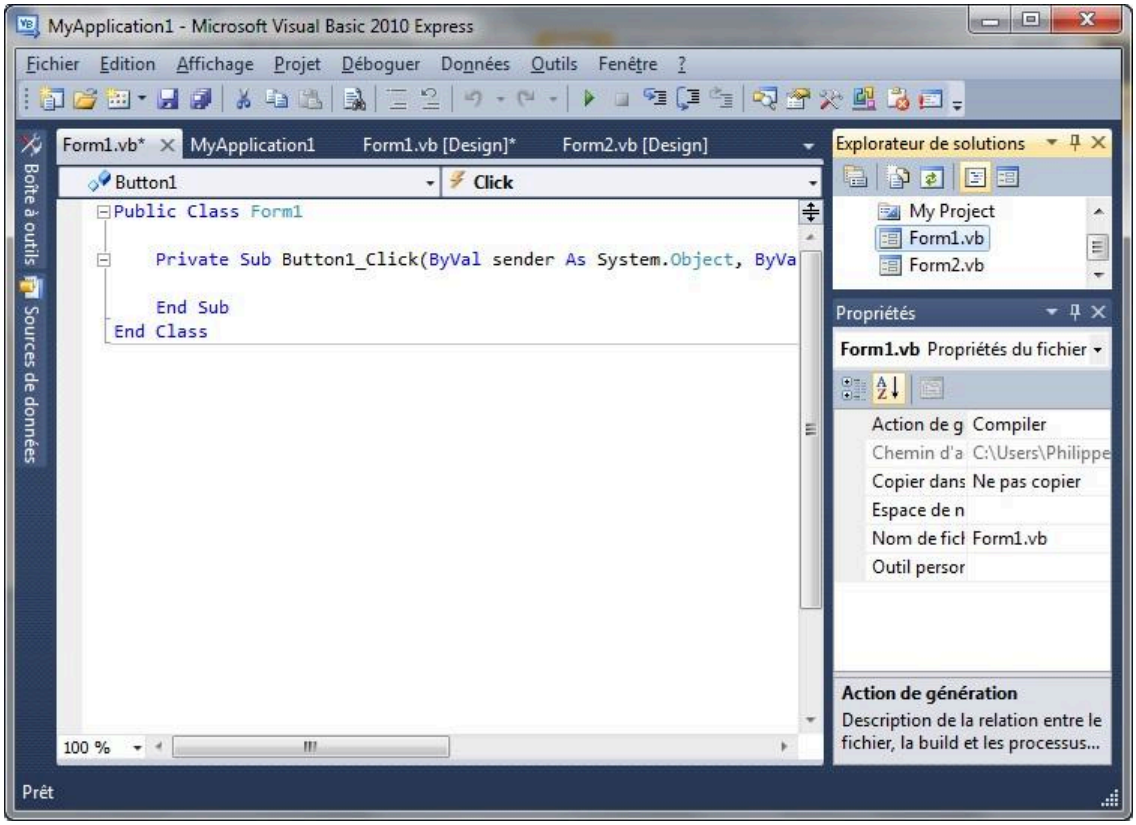



## IV-B-1-d - Les procédures

Elles contiennent le code en visual basic. On se souvient que pour chaque événement de chaque objet visuel il existe une procédure.

Dans l'explorateur de solution à droite, cliquer sur Form1.vb puis sur l'icône 'Afficher le code' (survoler les petites icônes, c'est la 4$^{e}$), cela donne accès aux procédures liées à Form1.

**Il est aussi possible en double-cliquant dans le formulaire ou un contrôle de se retrouver directement dans le code de la procédure correspondant à cet objet.**

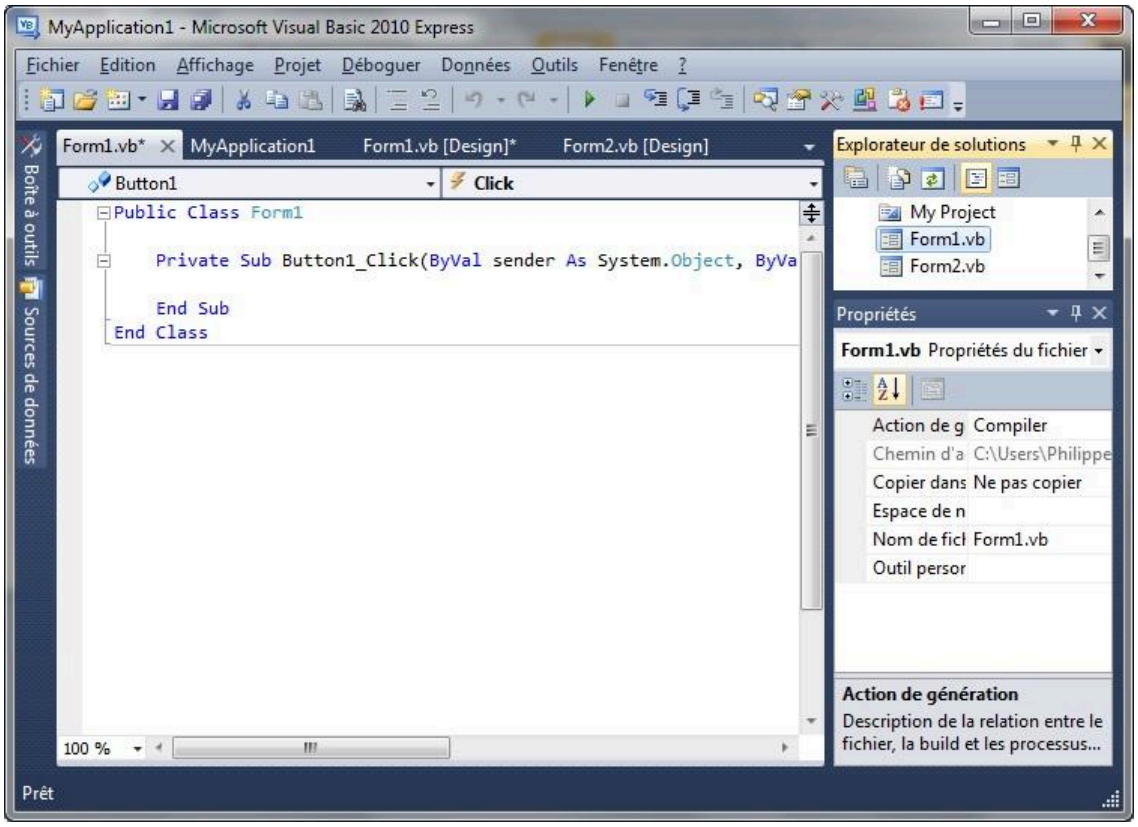



Dans toute fenêtre de code ou de texte, vous pouvez effectuer rapidement un zoom avant ou arrière (agrandir ou réduire la taille des caractères) en appuyant sur la touche CTRL tout en déplaçant la roulette de défilement de la souris.

On peut 'taper' du code dans les procédures. Dès que vous tapez quelques caractères Vb ouvre des listes vous proposant la suite (voir aide).

La liste déroulante de gauche au-dessus de la procédure donne la liste des objets, celle de droite, les événements correspondants à cet objet.

Ici on voit la procédure Button1_Click liée au Button1 de la fenêtre de Design :




**Il est possible de faire de la saisie ou de l'insertion multiple.**
Pour cela il faut appuyer sur ALT puis sélectionner à la souris une colonne sur plusieurs lignes:

```
b=
c=
d=
```

Si ensuite on tape du code , il apparait sur toutes les lignes :

```
b=d+1
c=d+1
d=d+1
```

Insertion de texte : sélectionnez plusieurs lignes puis tapez dans la sélection de zone pour insérer le nouveau texte dans chaque ligne sélectionnée.
Collage : collez le contenu d'une sélection de zone dans une autre.
Zones de longueur nulle : effectuez une sélection verticale de zéro caractère de largeur pour créer un point d'insertion multiligne où insérer du texte sur toutes les lignes en même temps.

**Quand on clique sur une variable, cette variable est surlignée** dans l'ensemble du code :
CTL+MAJ+Flèche haut ou Flèche bas permet de passer à la variable surlignée suivante ou précédente.

```
Dim b As Integer
Button1.Text = CType(b, String)

b = b + 1
```

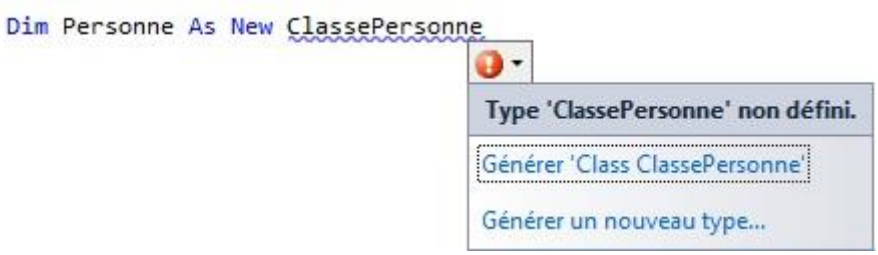
**Renommer un nom : modification automatique**

On nomme cela '**Refactoring**': cliquer sur une variable, puis bouton droit, dans le menu cliquer sur 'Renommer'. Modifier le nom de la variable, valider. Dans toute la Classe la variable est renommée.

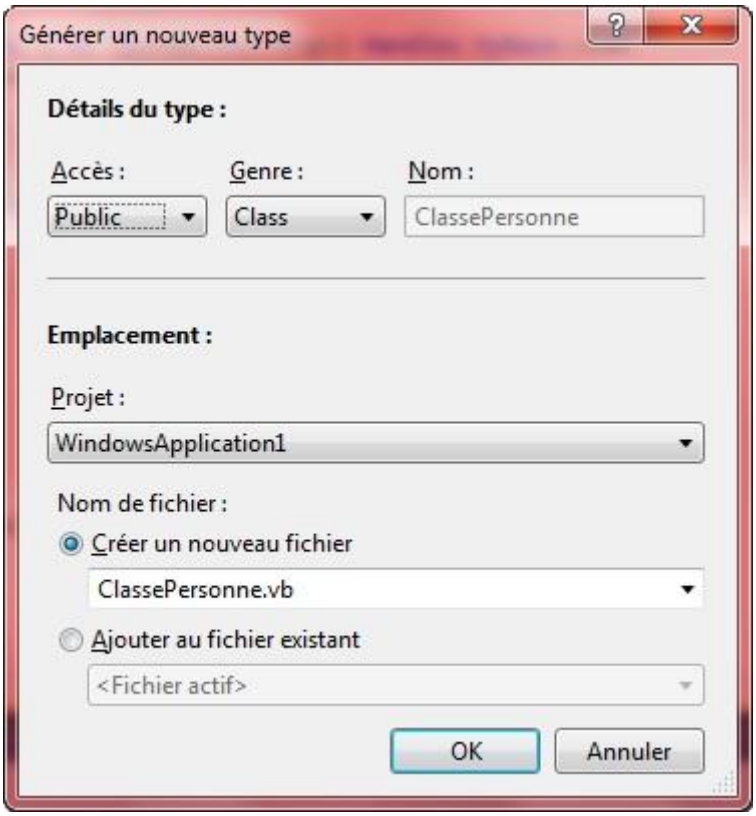
Si une ligne de code est trop longue, on peut ajouter un caractère de continuation de ligne "_" et passer à la ligne. En VB 2010, après certains mots il peut y avoir continuation de ligne implicite (plus besoin de _ après la virgule, après &, après une parenthèse ouvrante, après { ou = ou + ou Is…).

```
Public Function GetUsername(ByVal username As String,
                            ByVal delimiter As Char,
                            ByVal position As Integer) As String

    Return username.Split(delimiter)(position)
End Function
```

**En vb 2010, si dans le code, on utilise une classe qui n'existe pas, vb souligne le nom de la classe et vous propose (en cliquant sur le bouton dessous) de créer une classe, une structure ou un Enum vide :**

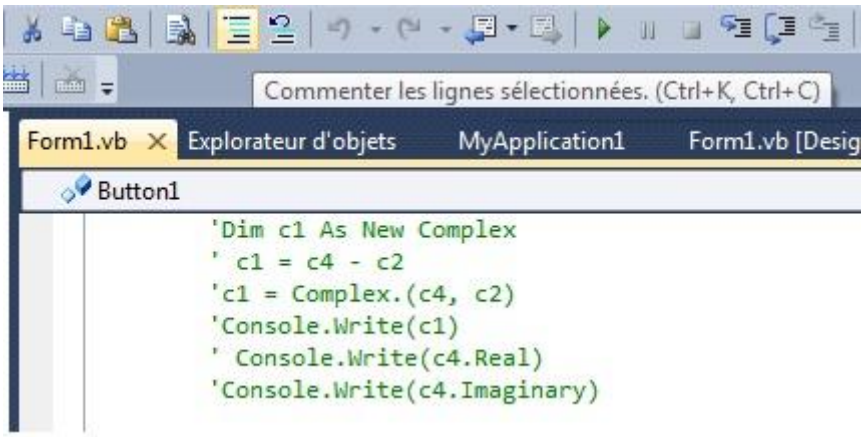



. Cliquez sur 'générer un nouveau type".

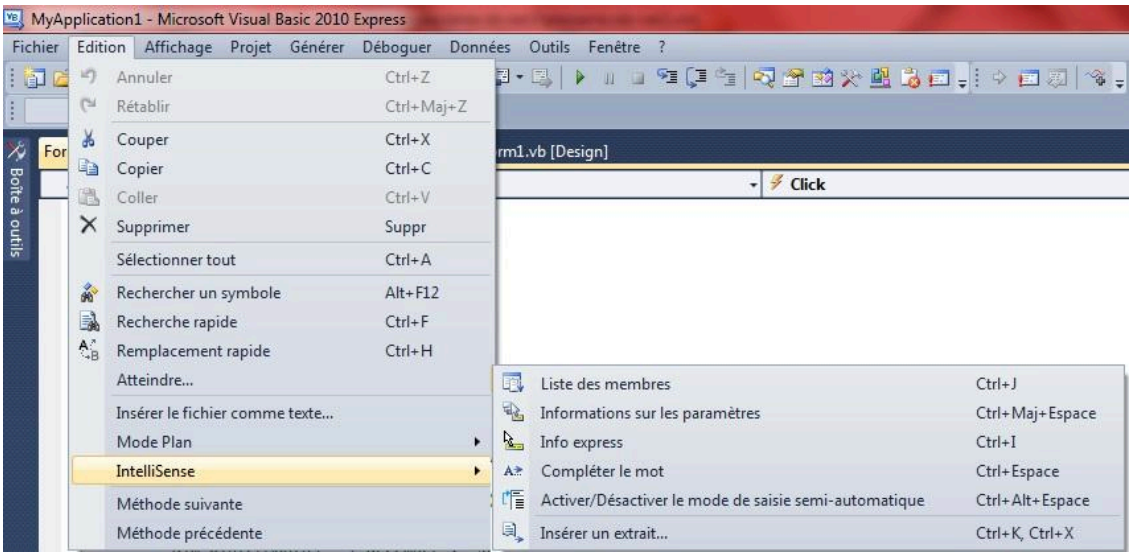



Il existe une **'Recherche rapide'** très puissante accessible par le menu 'Édition'.


Il y a bien sur les habituels boutons '**Couper**', '**Copier**', '**Coller**', mais aussi **le bouton qui transforme les lignes sélectionnées en commentaire** (ou l'inverse).



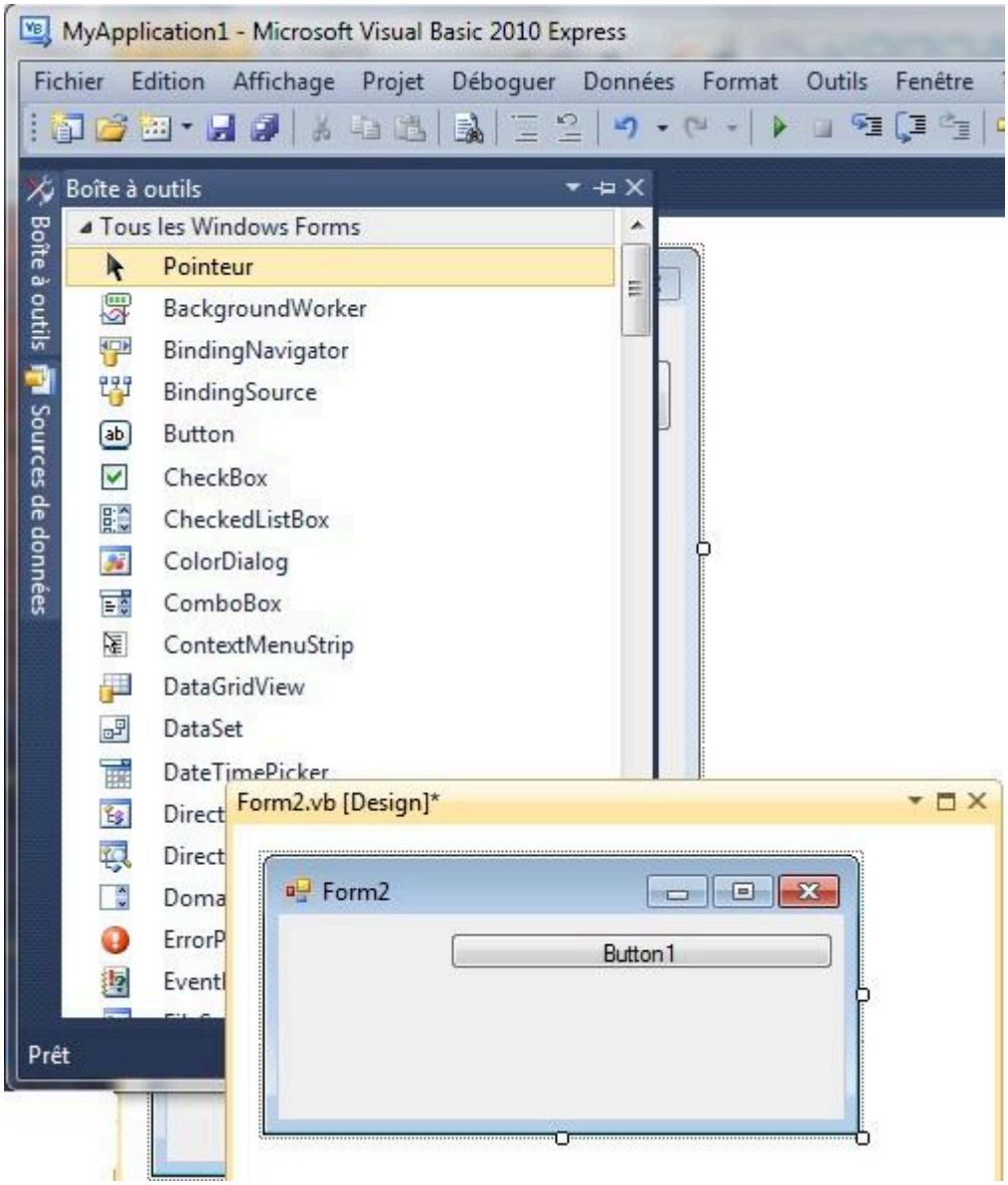

Le menu 'Édition' puis '**IntelliSense**' donne accès aux aides à l'écriture




## IV-B-1-e - Ajouter des contrôles au formulaire

Ajouter un bouton par exemple.

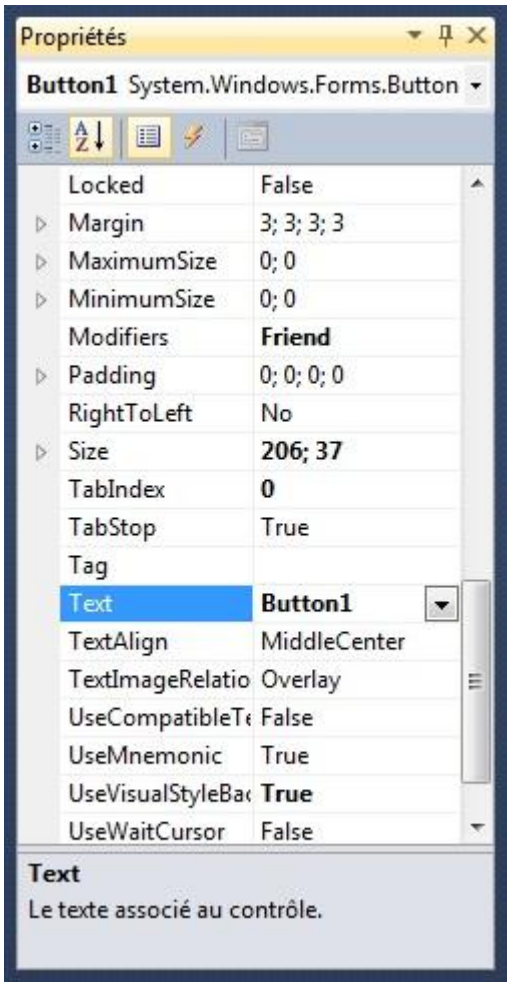
Passer sur Boite à outils (Toolbox) à gauche, les contrôles apparaissent tous ou classés par ordre alphabétique.




Cliquer sur 'Button' dans la boite à outils, cliquer dans la Form, déplacer le curseur sans lâcher le bouton, puis lâcher : un bouton apparait.

**Modifier les propriétés d'un contrôle ou du formulaire**

Quand un formulaire ou un contrôle est sélectionné dans la fenêtre Design, ses propriétés sont accessibles dans la fenêtre de 'Propriétés' (Properties) à droite en bas: ici ce sont les propriétés du contrôle 'Button1' qui sont visibles (Text, Location…) on peut modifier directement les valeurs.




En bas de la fenêtre propriétés, il y a une explication succincte de la propriété sélectionnée (si elle n'apparait pas, clic droit sur la propriété puis dans le menu 'Description').

Exemple

Si au niveau de la ligne 'Text' des propriétés du bouton, j'efface 'Button1' et que je tape 'OK', dans le designer, le texte écrit sur le bouton deviendra 'OK'.

En haut de cette fenêtre de propriété, il y a un bouton avec un éclair donnant accès aux événements de l'objet.

**Le déplacement des contrôles**

Plus de croix ni de petite flèche pour déplacer le contrôle ou ouvrir un menu. Pour déplacer le contrôle, appuyer bouton gauche dedans puis déplacer.

**L'alignement automatique des contrôles**

Si on modifie la taille ou l'emplacement d'un contrôle, VB signale par un trait bleu que le contrôle modifié et le contrôle voisin sont alignés :

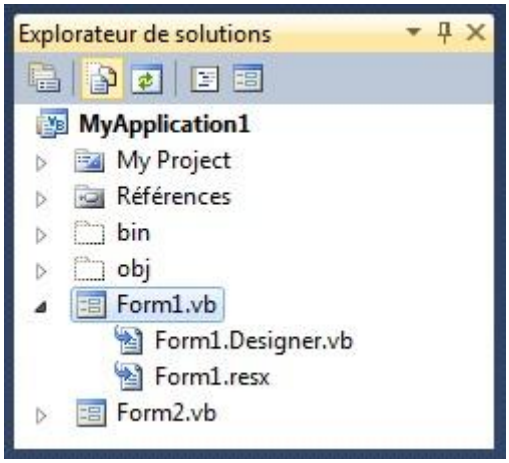

**Disposition des contrôles**

Le menu 'Outils', 'Personnaliser' permet d'ajouter une barre de disposition :

Cela permet (comme avec le menu Format) d'aligner les contrôles sur la grille, d'aligner les côtés, la taille, les espacements, d'uniformiser les tailles, de centrer dans la fenêtre…

## IV-B-1-f - Voir tous les composants d'un projet

Pour cela il faut utiliser la fenêtre **Explorateur de solutions** en haut à droite, elle permet de voir et d'avoir accès au contenu du projet (pour voir tous les fichiers, il faut cliquer sur le deuxième bouton en haut) :




**MyProjet :** double-cliquer dessus, vous ouvrirez la fenêtre 'propriétés du projet'.

**Références** qui contient les dll chargées. Pour atteindre les références, on peut aussi passer par le menu 'Projet' puis 'Propriétés' ou double-cliquer sur 'MyProjet' puis choisir l'onglet 'Références.

**Form1.vb** est un formulaire (une fenêtre). Les formulaires, modules de classe ou standard sont tous des '.vb' il suffit de double-cliquer dessus pour les ouvrir.

Si on ouvre la sous-liste de Form1.vb (en cliquant sur le petit triangle), on voit :

**Form1.Designer.vb** (qui montre le code qui crée le formulaire, on n'a pas à y toucher) ;

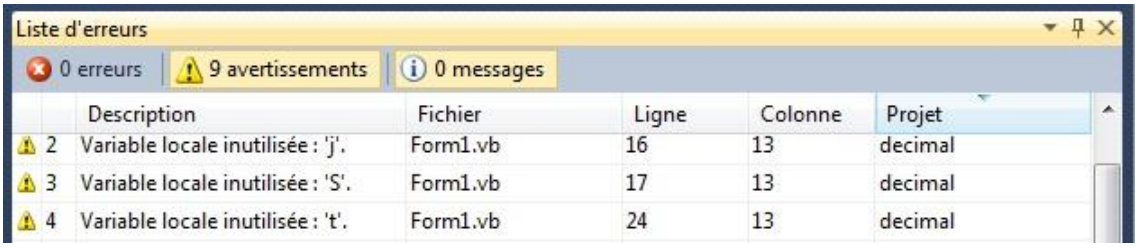
**Form1.resx** (le fichier de ressources).

Il suffit de cliquer sur la ligne Form1 dans l'explorateur de solution pour voir apparaitre la Form1 dans la fenêtre principale.

Si on clique sur un espace de noms dans la liste Références, cela montre l'arborescence des Classes.

## IV-B-1-g - Tester son logiciel

La **Liste d' erreurs**, en bas indique les erreurs de syntaxe (erreurs et avertissements).




On peut tester le projet grâce à :  lancer l'exécution avec le premier bouton (mode 'Run'), le second servant à arrêter totalement l'exécution: Retour au mode 'Design' (ou 'Conception').
Pour arrêter le programme temporairement en mode 'Debug' il faut un point d'arrêt ou faire Ctrl+Pause (ce qui ouvre une autre fenêtre !!).

En cliquant sur  (visible lors de l'exécution) on peut aussi suspendre l'exécution.
Par contre on peut mettre un point d'arrêt quand le programme tourne en cliquant dans la marge grise avant le code.

Quand on est en arrêt temporaire en mode 'Debug', la ligne courante, celle qui va être effectuée, est en jaune :

```
Private Sub Button1_Click(ByVal sender As System.Object,
    Dim b As Integer
    b = 2
    Button1.Text = "ok"
End Sub
```

Si on tape la touche F8 (exécution pas à pas), la ligne est traitée et la position courante passe à la ligne en dessous.
F5 relance l'exécution.

En mode Debug, on peut modifier une ligne et poursuivre le programme qui tiendra compte de la modification (sauf pour les déclarations). On parle d''Edit and continue'.

## IV-B-1-h - Sauvegarde, Projet, chemin

La **sauvegarde du projet** se fait comme dans tous les logiciels en cliquant sur l'icône du paquet de disquettes.

On peut **compiler** (on dit 'générer') le programme pour créer un exécutable par le menu 'Générer' ('Build'). Le code présent dans l'IDE est le code **source**, après compilation le fichier exécutable contient du code **exécutable**.

**Projet**

Dans la terminologie VB, **un projet** est une application en cours de développement.

Une '**solution**' (Team Project) regroupe un ou plusieurs projets (c'est un groupe de projets). Il n'y en a pas dans la version express.

En VB express on parle donc uniquement de projet, en fait ,VB crée aussi une solution de même nom.

**Fichiers, Chemins des sources**

Sous Windows XP regardez dans ' C:\Documents and Settings\Nom Utilisateur\Mes documents\Visual Studio 2010\Projects\MonProjet'
Sous Windows 7 le programme est accessible dans 'Document/Visual Studio 2010/Projects/MonProgramme'.

**MonProjet.sln** est le fichier solution. (Pas de solution en VB express, que des projets.)

**MonProjet.psess** est le fichier de performance (pas toujours présent).

**MonProjet.suo** est le fichier de User Option.

Dessous existe un répertoire nommé aussi MonProjet qui contient :

**MonProjet.vbProj** le fichier de projet ;

**Form1.vb** contient un formulaire et ses procédures ;

**MyClasse.vb** contient par exemple des classes ;

**Form1.Designer.vb** contient le code qui créer la fenêtre et les contrôles.

Il a encore les sous répertoires \Bin, il y a aussi un répertoire \Obj et un répertoire \MyProjet.

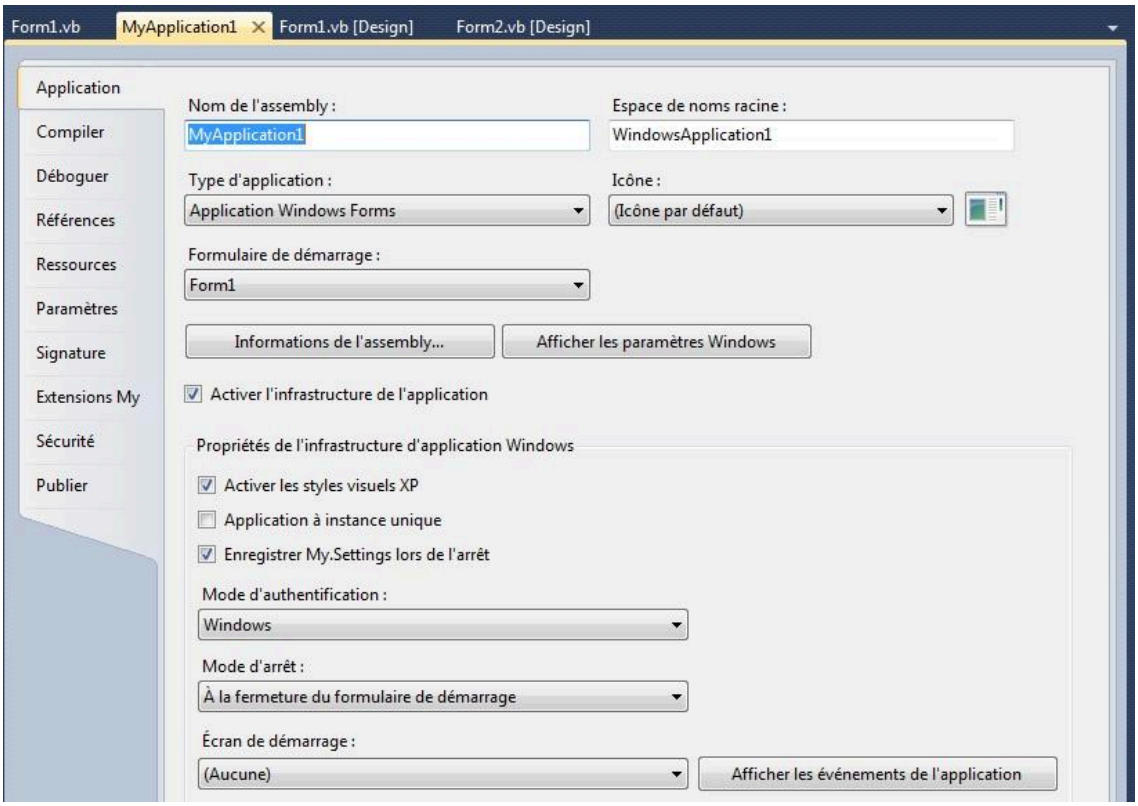
Si on compile (génère) le projet l'exécutable est dans un sous répertoire \Bin, Vb 2010 choisit automatiquement la configuration **Debug** (compilée avec des informations de débogage symboliques et aucune optimisation) lorsque vous cliquez sur Démarrer dans le menu Déboguer et la configuration **Release** (ne contient pas d'informations de débogage relatives aux symboles et est entièrement optimisée) lorsque vous utilisez le menu Générer.
Les exécutables générés (fichier .exe) sont respectivement dans /bin/Debug et /bin/Release.

## IV-B-1-i - Propriétés du projet

Toutes les propriétés de l'application peuvent être modifiées dans le 'Projet Designer' (**Propriétés du projet**), pour l'atteindre, il faut double-cliquer sur 'My Project' dans l'explorateur de solution.

Une autre manière d'ouvrir le 'Projet Designer' est de passer par les menus 'Projet' puis 'Propriétés de…'

On retrouve dans le projet designer :




Le nom de l'application, son icône, la fenêtre de démarrage, celle de fin. (Application)

Les Option Strict, Explicit compare et Option Infer.(Onglet Compiler).

Les références (dll liées au projet).

Les paramètres (valeurs liées à l'application).

Les ressources (texte, image, son) utilisées dans le programme.

La signature et la sécurité.

Les Extension My (nouveauté 2008).

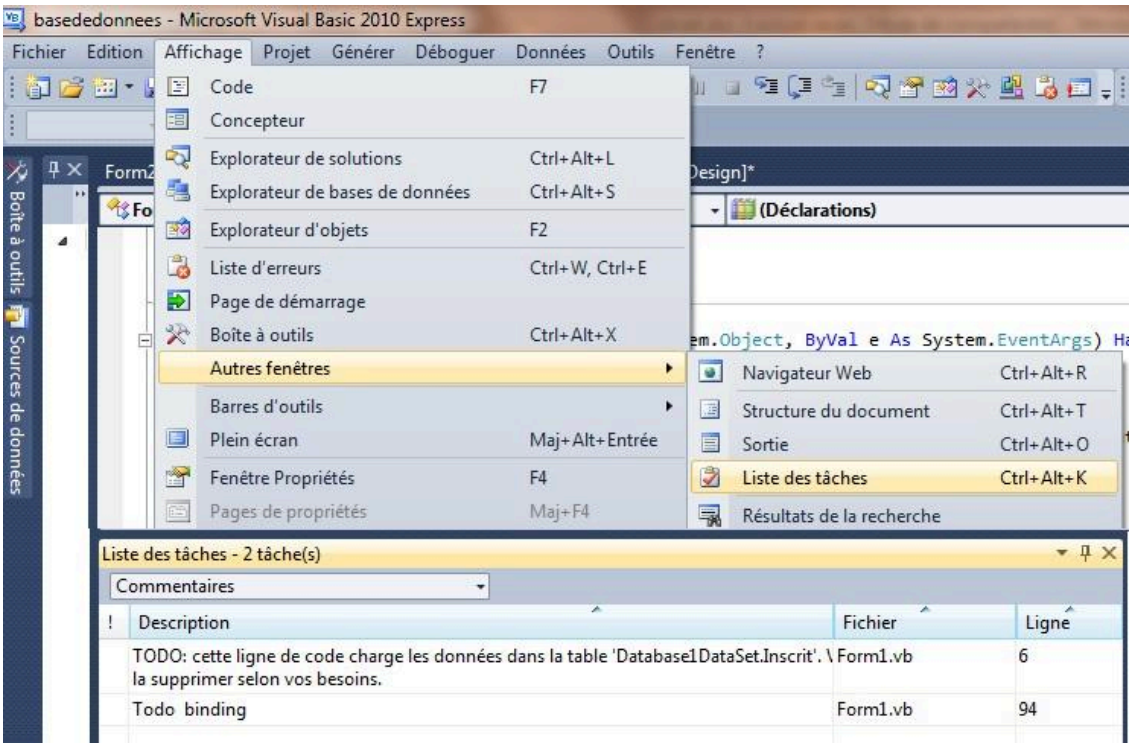
Les paramètres relatifs à la publication (distribution et installation).

## IV-B-1-j - Autre

On peut mettre des commentaires précédés de '.

```
' Commentaire non exécutable
'TODO ne pas oublier de modifier …
```

Si le premier mot du commentaire est TODO, comme ci-dessus, ce commentaire sera affiché dans la liste des tâches : (Menu Affichage, Autres fenêtres, Liste des tâches).



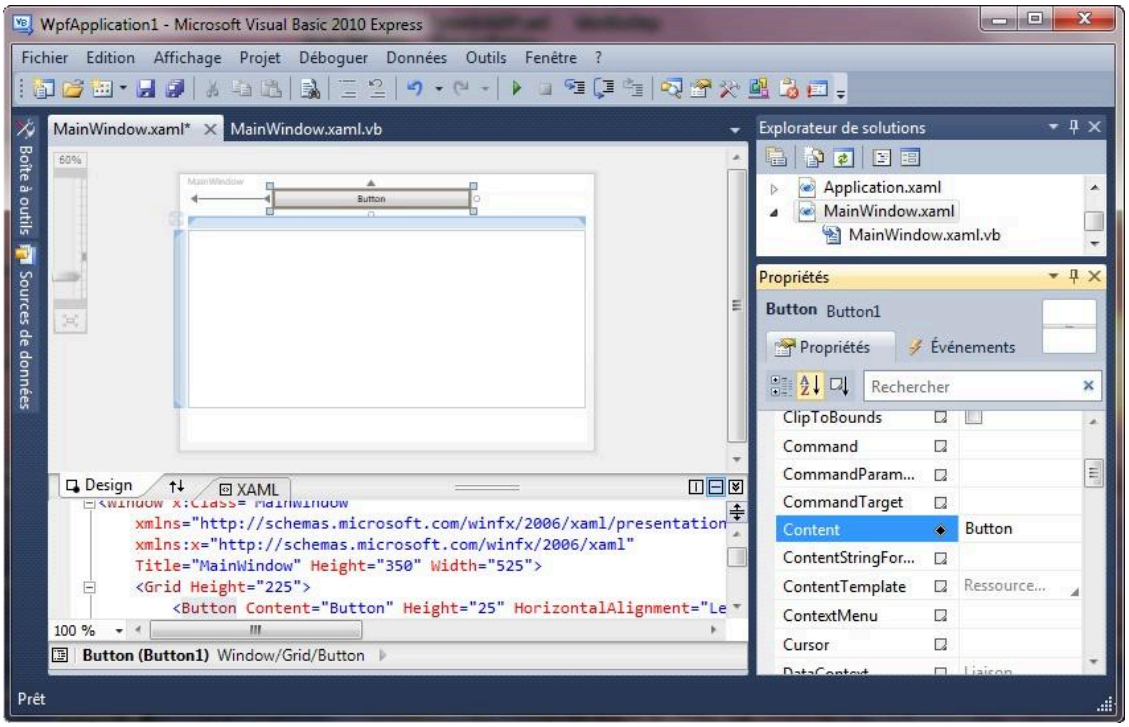

C'est bien pratique pour ne pas oublier des modifications à faire.

## IV-B-2 - Interface WPF

Plutôt que de travailler avec les Windows Forms (formulaire habituel utilisant GDI+), en VB 2008 on peut utiliser un mode graphique vectoriel extrêmement performant pour dessiner les formulaires et contrôles : pour cela on utilise les WFP (Windows Presentation Foundation).

Pour cela : menu 'Fichier', 'Nouveau', 'Projet'.

On choisit 'Application WPF', on se retrouve dans un nouvel environnement :




Les formulaires et contrôles sont différents de ceux des Windows Forms, ainsi que les propriétés des objets graphiques.

Il y a le 'designer' en haut qui permet de dessiner l'interface que verra l'utilisateur. Le designer génère un fichier XAML en bas qui décrit en XML l'interface.

Dans la version Express, on peut dessiner des interfaces simples, les interfaces extrêmement élaborée (dégradé de couleur, animation…) peuvent être écrites en code XAML ou en utilisant un programme extérieur payant (Expression Blend). Voir le chapitre sur les WPF.

Si on double-clique sur un bouton, par exemple, on se retrouve dans la procédure événement correspondante.

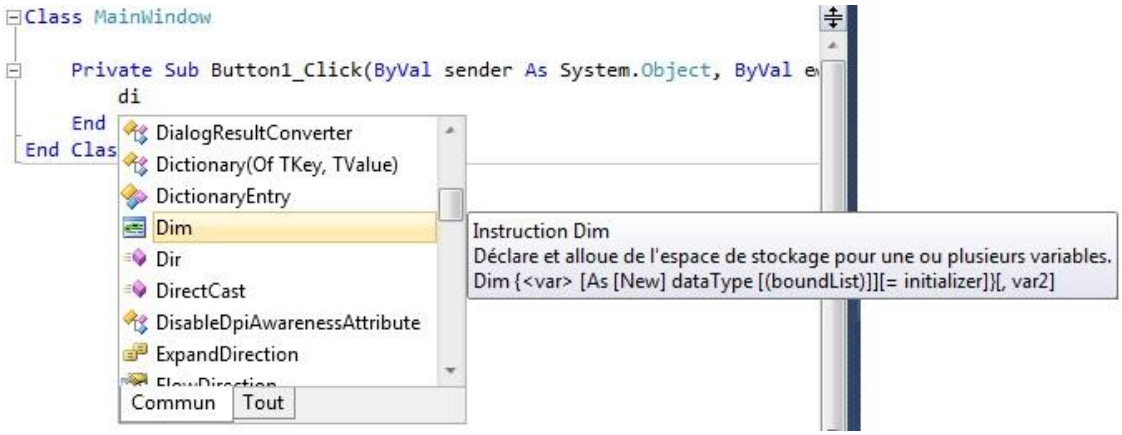
On se rend compte que les événements là aussi ne sont pas les mêmes que pour les WindowsForm.

Il y a aussi d'autres modifications comme dans les propriétés du projet : voir le chapitre sur les WPF.

## IV-B-3 - Vb propose des aides

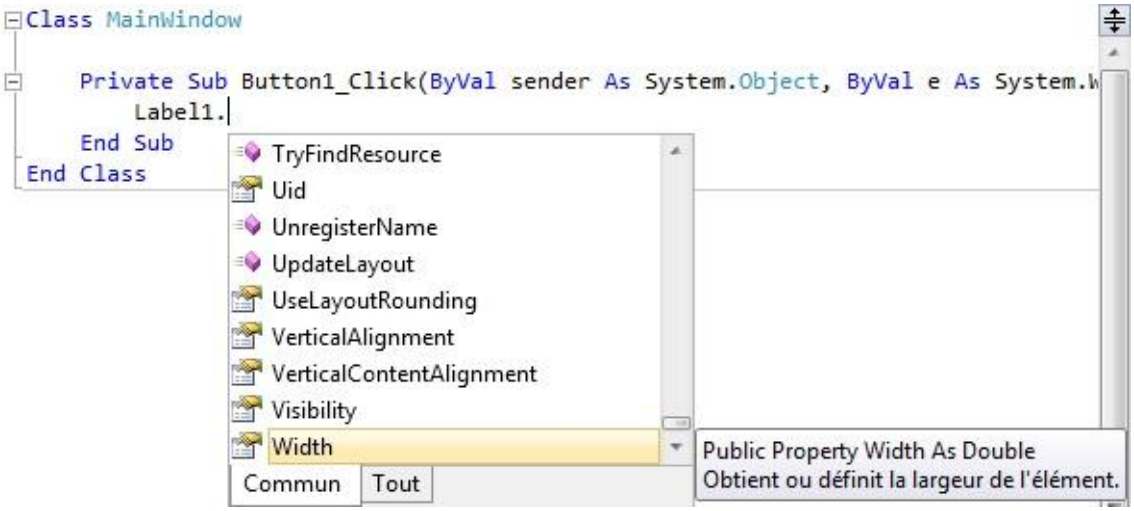
Quand on tape du code dans une procédure, VB affiche, des aides : dès que je tape une lettre VB propose dans une liste des mots.

Exemple, je tape 'di', il affiche 'Dim', 'Dir'…, de plus si je me mets sur un des mots, il ouvre une petite fenêtre d'explication sur le mot avec sa syntaxe.



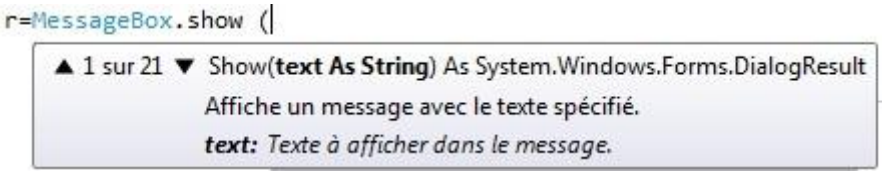

*VB permet de choisir dans une liste une des propriétés d'un objet.*

Exemple : je tape le nom d'un label nommé label1 puis je tape un point, cela me donne la liste des propriétés du label. Quand on pointe une propriété, un rectangle donne une explication sur cette propriété.




*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.

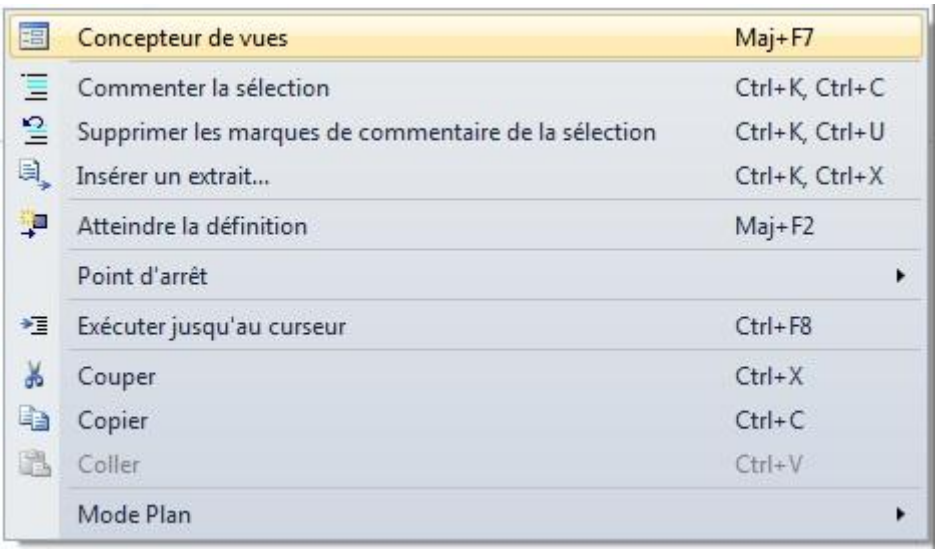



*En plus il affiche les différentes manières d'utiliser les paramètres* (les différentes signatures), on peut les faire défiler avec les petites flèches du cadre blanc.

*VB aide à compléter des mots.*

Si je tape 'App' Vb affiche la liste des mots commençant par App: Objet, variables…

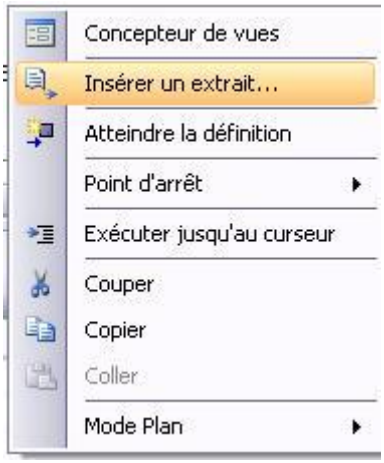
Dans le code, cliquez avec le bouton droit ouvre un menu et donne accès à :




*VB fournit des exemples de code.*

Les **Extraits** (Snippets, bride, morceau de code) permettent d'insérer du code tout fait.

Dans le code d'une procédure, le clic droit de la souris ouvre un menu.

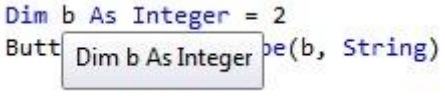



Cliquer sur 'Insérer un extrait' (Insert Snipper). Puis par menu successif vous obtiendrez le code que vous cherchez.

*Il existe une abondante documentation:*
**http://msdn.microsoft.com/fr-fr/library/dd831853%28v=VS.100%29.aspx**

Dans l'IDE, VB donne accès à l'aide sur un mot-clé. 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.
Enfin il est toujours possible de rechercher des informations par le menu '?'.

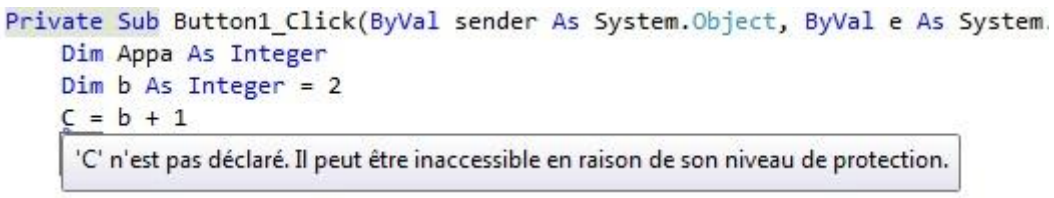
Si le curseur passe sur un mot-clé, un carré affiche la définition de la fonction, le type de la variable, sa déclaration.



### Erreur dans l'écriture du code

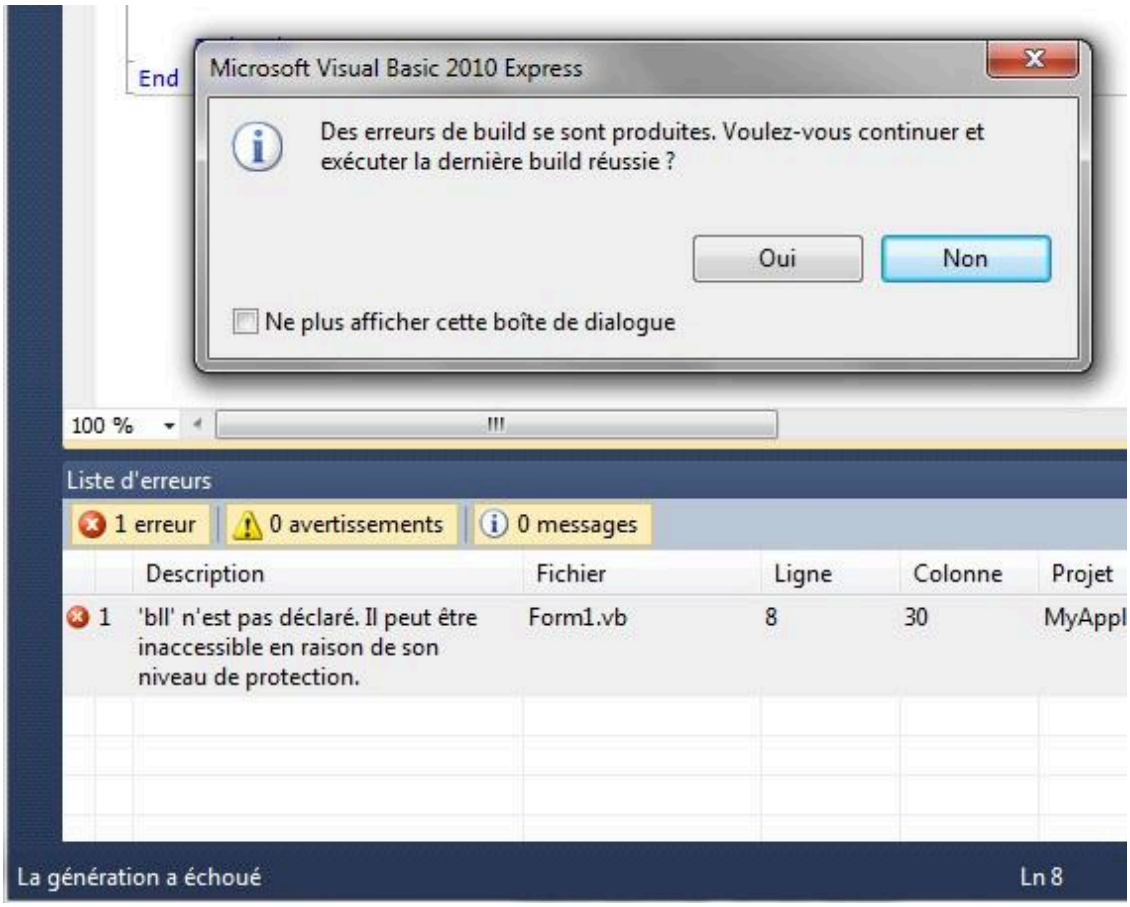
Vb propose des solutions pour corriger les erreurs de code :

Si je fais une erreur Vb la souligne en ondulé bleu, si je mets le curseur dessus il m'explique l'erreur :




S'il y a un soulignement rouge, mettre le curseur dessus affiche un bouton avec un point d'exclamation qui ouvre une fenêtre donnant la correction de l'erreur :

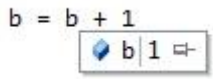

Si je lance le programme en mode 'Run' et qu'il y a des erreurs, Vb me le signale et répertorie les erreurs dans la liste des tâches en bas.




**Mode débogage (mode BREAK)**

Une fois lancée l'exécution (F5), puis stoppée (par Ctrl +Pause ou sur un point d'arrêt), ou en cliquant sur  on peut :

voir la valeur d'une propriété d'un objet, d'une variable en le pointant avec la souris :




Il s'affiche un petit cadre donnant la valeur de la propriété d'un objet.
Si on clique sur la punaise, l'affichage de la variable et de sa valeur devient permanent et la valeur est mise à jour.

Quand on clique sur une variable, cette variable est surlignée dans l'ensemble du code :

```
Dim b As Integer
Button1.Text = CType(b, String)

b = b + 1
```

**F8** permet l'exécution pas à pas (y compris des procédures appelées).

**MAJ F8** permet le pas à pas (sans détailler les procédures appelées).

**CTRL+F8** exécute jusqu'au curseur.

Il y a un chapitre sur le débogage pour apprendre à trouver les erreurs de code.

## IV-C - IDE SharpDevelop (logiciel libre en open spource)

C'est l'IDE (Integrated Development Environment) : Environnement de développement intégré GRATUIT, en open source, alternative à VisualStudio.

Depuis sa version 2 #develop est un très bon produit.
Il manque certaines choses.
Pas de "Edit and continue" : si on suspend l'exécution et qu'on modifie le code, la modification n'est pas prise en compte si on continue l'exécution, débogueur moins performant). On en est à la version 3.2 (version 4 en bêta).

Sharpevelopp (#Develop) sera toujours gratuit.

C'est un logiciel libre en open source (GPL).

### IV-C-1 - Où le trouver ? Comment l'installer ?

**SharpDevelop 3.2 (version stable)**

Si ce n'est pas fait, télécharger et installer le **FrameWork 3.5**. (Impérativement en premier).

Télécharger et installer le *SDK*.

C'est le Kit de Développement Microsoft .NET Framework : SDK du Framework 3.5.

Télécharger et installez SharpDevelop 3.2 (le 14/4/2010).

**Télécharger SharpDevelop 3.2 (gratuit)**

L'installer en exécutant le fichier d'installation.

Le Framework, le SDK et #develop suffisent pour faire des programmes.
La version 3.2 permet de travailler en VB avec les winforms.

**SharpDevelop 4 (version beta)**

Si ce n'est pas fait, télécharger et installer le **FrameWork 4**. (Impérativement en premier.)

Télécharger et installer le *SDK*.

C'est le Kit de Développement Microsoft .NET Framework : SDK du Framework 4.

Télécharger et installez SharpDevelop 4 bêta (le 14/12/2010).

**Télécharger SharpDevelop 4 (gratuit)**

L'installer en exécutant le fichier d'installation.

Le Framework, le SDK et #develop suffisent pour faire des programmes.

La version 4 bêta permet aussi de travailler avec WPF. Au 27/12/2010 elle n'est pas finalisée : impossible de créer facilement une Sub événement.

**Lien**

**Site SharpDevelop**

## IV-C-2 - Fenêtre Projet Windows Forms

Didacticiel en anglais pour les Windows Forms : http://community.sharpdevelop.net/blogs/mattward/articles/FeatureTour.aspx.

Lancer SharpDevelop

**Au lancement de l'application, on peut :**

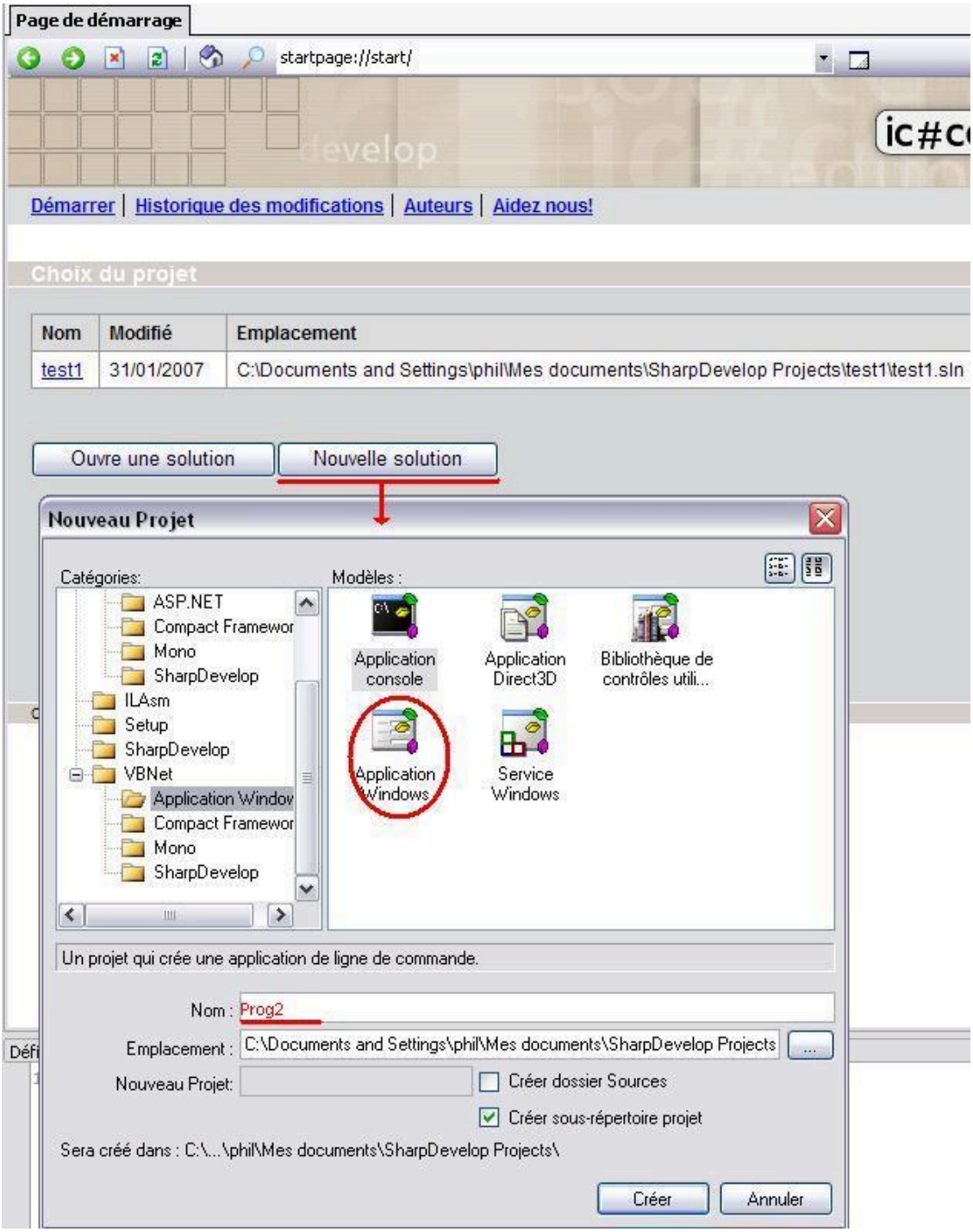
- ouvrir une solution existante: Bouton 'Ouvrir une solution' (ou cliquer sur le nom d'un projet récent en haut) ;
- créer un nouveau projet (une nouvelle solution).

Si l'on veut rajouter des fichiers à notre projet faire :
'Fichier'-'Ouvrir'-'Fichier' et catégorie VB

Détaillons la création d'un nouveau projet.

Bouton 'Nouvelle solution' ou

Menu 'fichier'-'Nouveau'-'Solution'

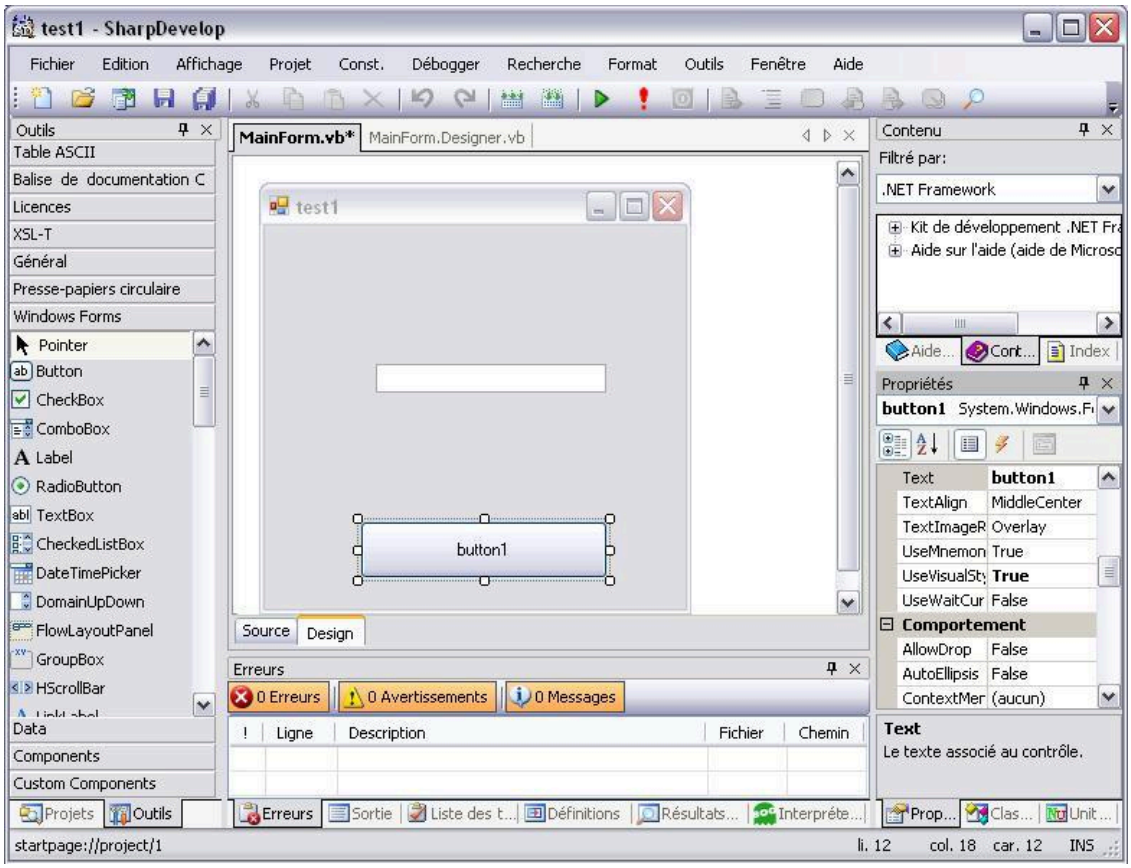



Sélectionner la catégorie 'VBNET' et choisir le type d'application à créer. (Dans le cas d'une création d'un projet Visual Basic, il faudra choisir dans les 'Modèles' : Application Windows.) On remarque que #Develop permet aussi d'écrire du C#, du C++ du ILAsm un setup du Python, Ruby….
On peut utiliser les Windows Forms (version 3.2) ou WPF (version 4).

Puis il faut donner un nom au projet (il n'y a pas de nom par défaut), modifier si nécessaire le chemin de l'emplacement du projet qui est par défaut ' C:\Documents and Settings\NomUtilisateur\Mes documents \SharpDevelop Projects' (cocher si nécessaire 'Créer le répertoire source') enfin valider sur le bouton 'Créer'. Une fenêtre 'MainForm' apparait.

Si, comme dans notre exemple, on a tapé 'Prog2', #develop crée une 'solution' nommée 'SolutionProg2' (ensemble, groupe de projets) contenant un projet (Prog2) contenant un formulaire nommé 'MainForm'.

L'écran principal se présente ainsi :




Au centre, sont visibles les écrans du code et des formulaires ; on peut changer d'écran grâce aux onglets du haut. Ici on voit 'MainForm'.

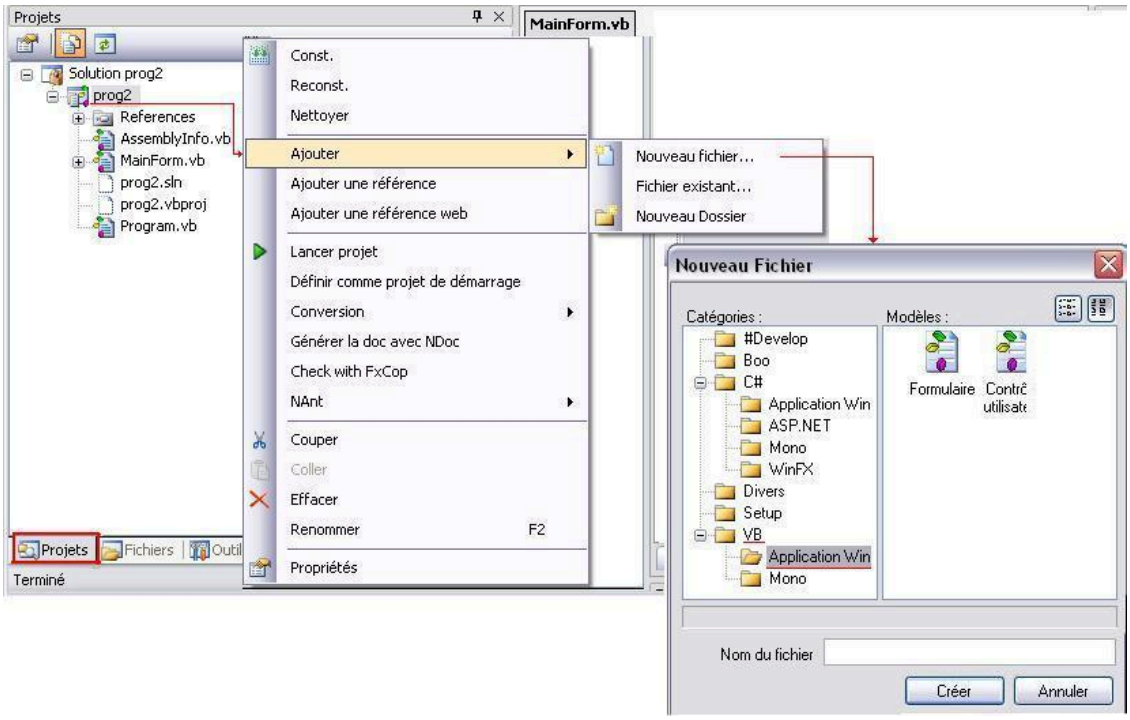
À gauche, les onglets du bas donnent accès au projet en cours (les solutions, projets, formulaires, autres fichiers : ressources, assembly…) ou aux outils : Table ascii, Presse papier et surtout (si on a un formulaire au centre et non du code) aux objets (bouton, texteBox, ListBox…).

À droite, en bas, les classes et surtout la fenêtre de Propriétés (Name, Text…) de l'objet sélectionné au centre.

En bas les fenêtres de 'sortie' (affichage de la console) liste des 'erreurs' des 'tâches', définitions', 'Résultat des recherches'…

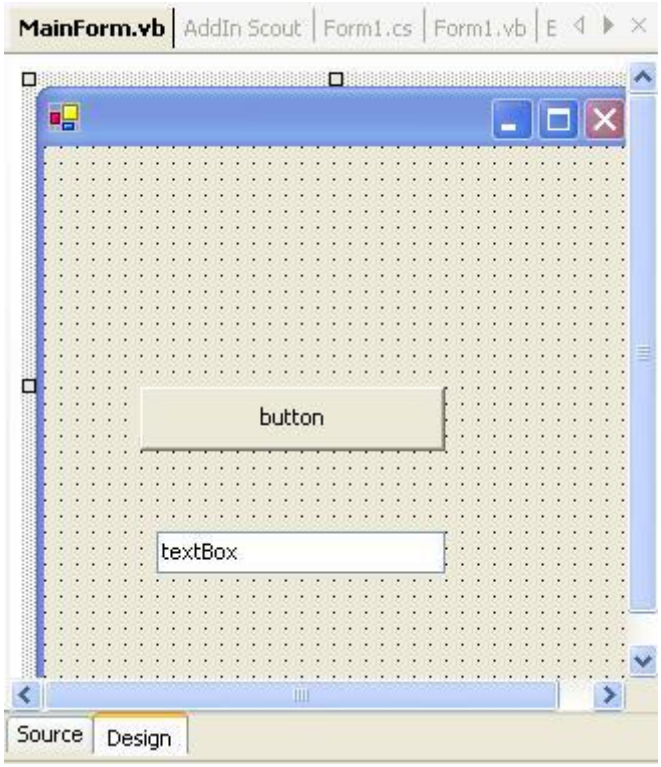
## IV-C-2-a - Dans un nouveau projet, créer une fenêtre

Pour ajouter une fenêtre (un formulaire) ouvrir le gestionnaire de projet et solution (Onglets en bas à gauche), il donne le nom de la solution (solutionprog2) et du projet (prog2 ici). Cliquer avec le bouton droit sur prog2 puis dans les menus sur 'Ajouter', 'Nouveau fichier'. Cela ouvre la fenêtre 'Nouveau fichier'.




Dans la fenêtre qui s'ouvre, à gauche, choisir 'VB' puis 'Application Windows', à droite 'Formulaire', taper un nom de formulaire (Form1 par exemple) puis 'Créer', une fenêtre 'Form1' apparait. La première fenêtre qui s'ouvre automatiquement quand on crée un projet se nomme 'MainForm'.

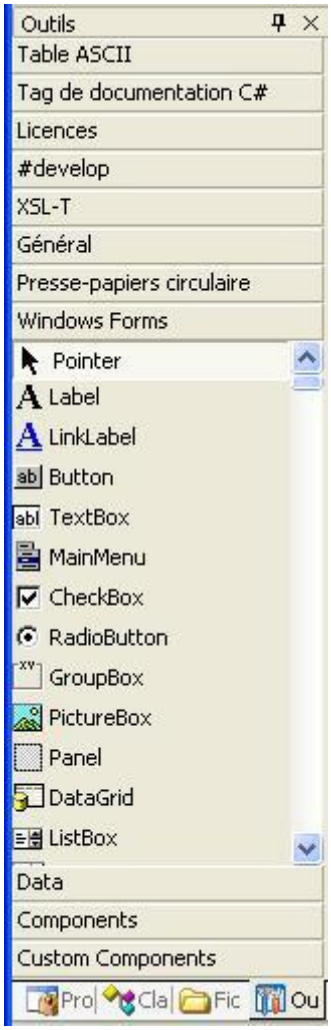
La zone de travail se trouve au centre de l'écran : on voit les onglets MainForm, Form1.vb pour chaque formulaire (fenêtre)



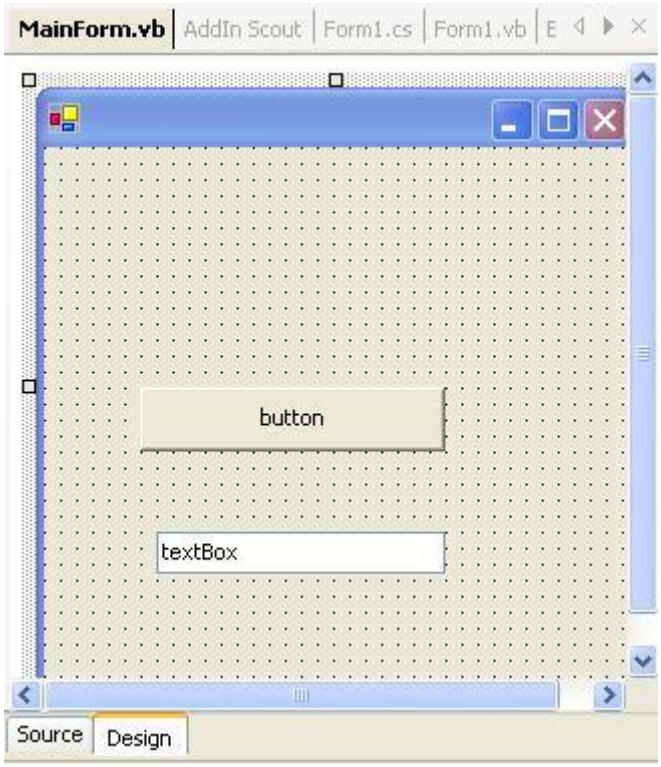

En bas les onglets 'Source' et 'Design' permettent de passer de l'affichage du code('Source') à la conception de l'interface utilisateur ('Design') : affichage de la fenêtre et de ses contrôles permettant de dessiner l'interface.

# IV-C-2-b - Ajouter des contrôles au formulaire

Ajoutons un bouton par exemple :




Cliquer sur l'onglet 'Outils' à gauche en bas , bouton 'Windows Forms', puis sur 'Button', cliquer dans la MainForm, déplacer le curseur sans lâcher le bouton, puis lâcher le bouton :



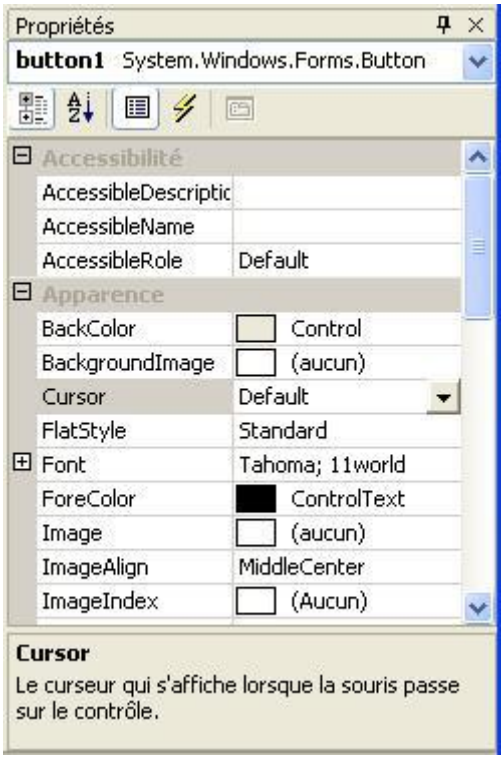

## IV-C-2-c - Modifier les propriétés d'un contrôle ou du formulaire

Quand une feuille ou un contrôle est sélectionné dans la fenêtre Design, ses propriétés sont accessibles dans la fenêtre de propriétés à droite en bas. (Si elles ne sont pas visibles, cliquer sur l'onglet 'Propriétés' en bas.)

Ici ce sont les propriétés du contrôle 'Button1' (BackColor, Image, Texte…).

Un petit texte d'aide concernant la propriété en cours apparait en bas.

(On peut modifier les propriétés directement.)




## IV-C-2-d - Voir les procédures

L'onglet 'Source' en bas donne accès aux procédures (au code) liées à Form1.


La combo déroulante de droite donne la liste des objets. Si on en choisit un, le pointeur va sur les procédures liées à cet objet.

Malheureusement, contrairement à Visual Studio, la combo de gauche ne contient que les formulaires et pas les objets. Par exemple, on aura MainForm, mais pas Label1… Du coup la recherche se fait directement dans la combo de droite et c'est forcément beaucoup moins clair dès qu'il y a beaucoup de contrôles sur un formulaire…

Il est possible en double-cliquant dans le formulaire ou sur un contrôle de se retrouver directement dans le code de la procédure correspondant à cet objet.

Si la procédure n'existe pas (ButtonClick par exemple),double-cliquez sur le bouton pour la créer.

Pour créer les autres procédures événements, utiliser le bouton  qui est sur la fenêtre des propriétés à droite, il fait apparaitre la liste des événements, double-cliquant sur un événement cela permet d'ouvrir la fenêtre de code et de créer les procédures.

## IV-C-2-e - Voir tous les composants d'un projet, les classes

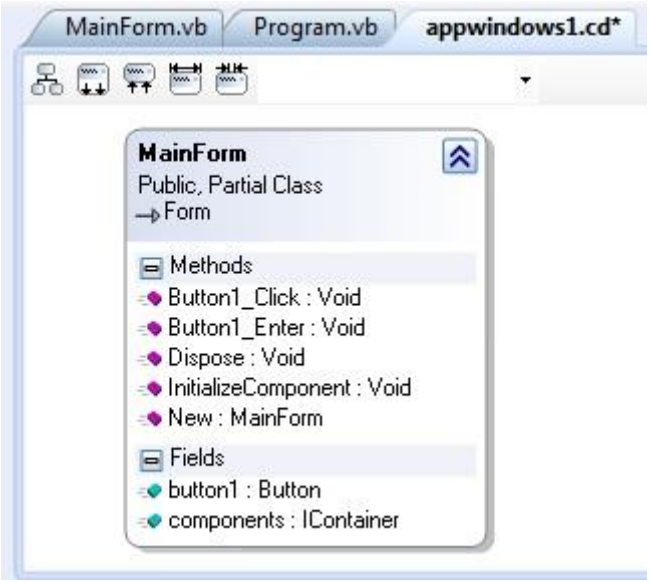
Pour cela il faut utiliser La fenêtre Projet à gauche (Si elles ne sont pas visibles, cliquer sur l'onglet 'Propriétés' en bas), elles permettent de voir et d'avoir accès au contenu du projet.

Le gestionnaire de projet et solution donne le nom de la solution (solutionprog2) et du projet (prog2 ici). Cliquer sur les '+' pour développer : vous verrez apparaitre les formulaires, les modules… et :

Références qui contient les espaces de nom ;

Assembly : info nécessaire pour générer le projet…

Le menu 'Fichier', 'Afficher les diagrammes de Classe' permet de voir les Classes du projet sous forme de diagramme :




## IV-C-2-f - Remarque relative aux fenêtres de l'IDE

Pour faire apparaitre une fenêtre qui a disparu (fenêtre projet par exemple) utiliser le menu 'Affichage' puis 'projet'.

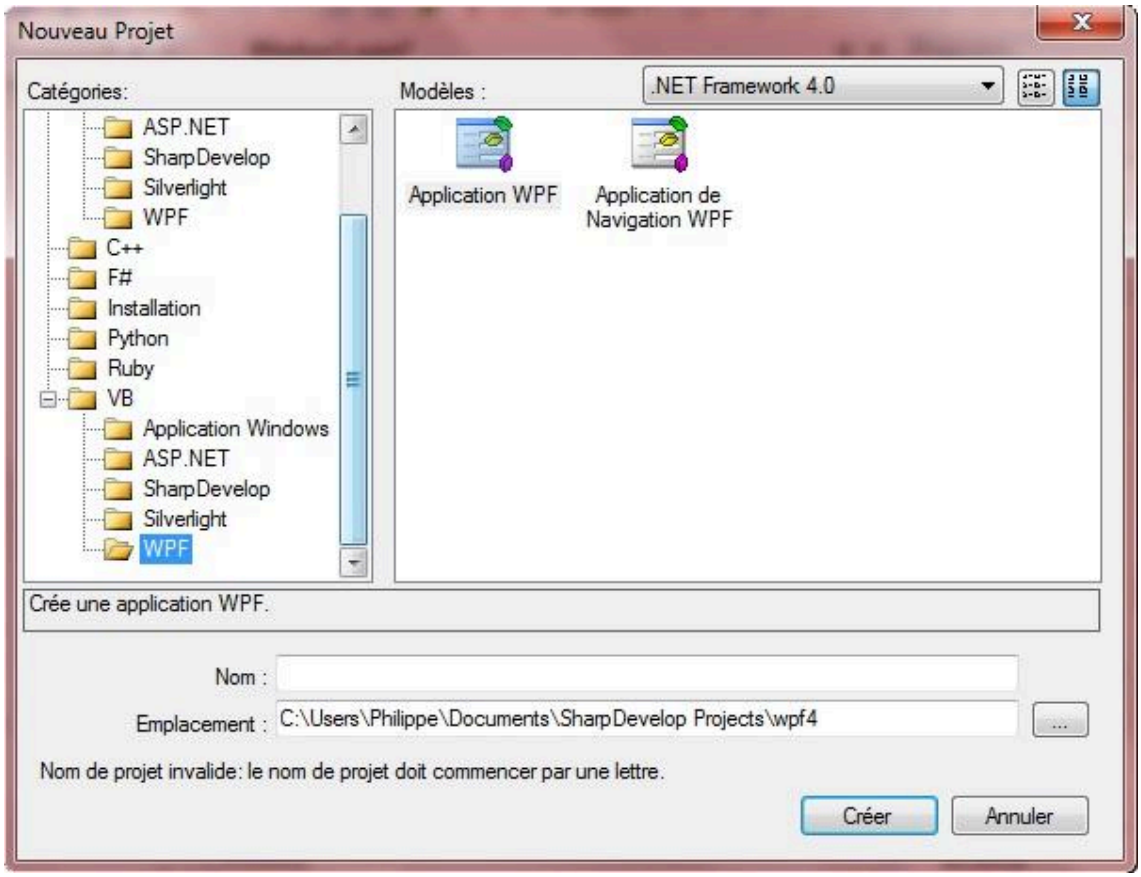
Quand la fenêtre est ancrée (accrochée aux bords), le fait de la déplacer avec sa barre de titre la 'dé ancre', et elle devient autonome.

Pour la 'réancrer', il faut double-cliquer dans sa barre de titre.

## IV-C-3 - Interface WPF

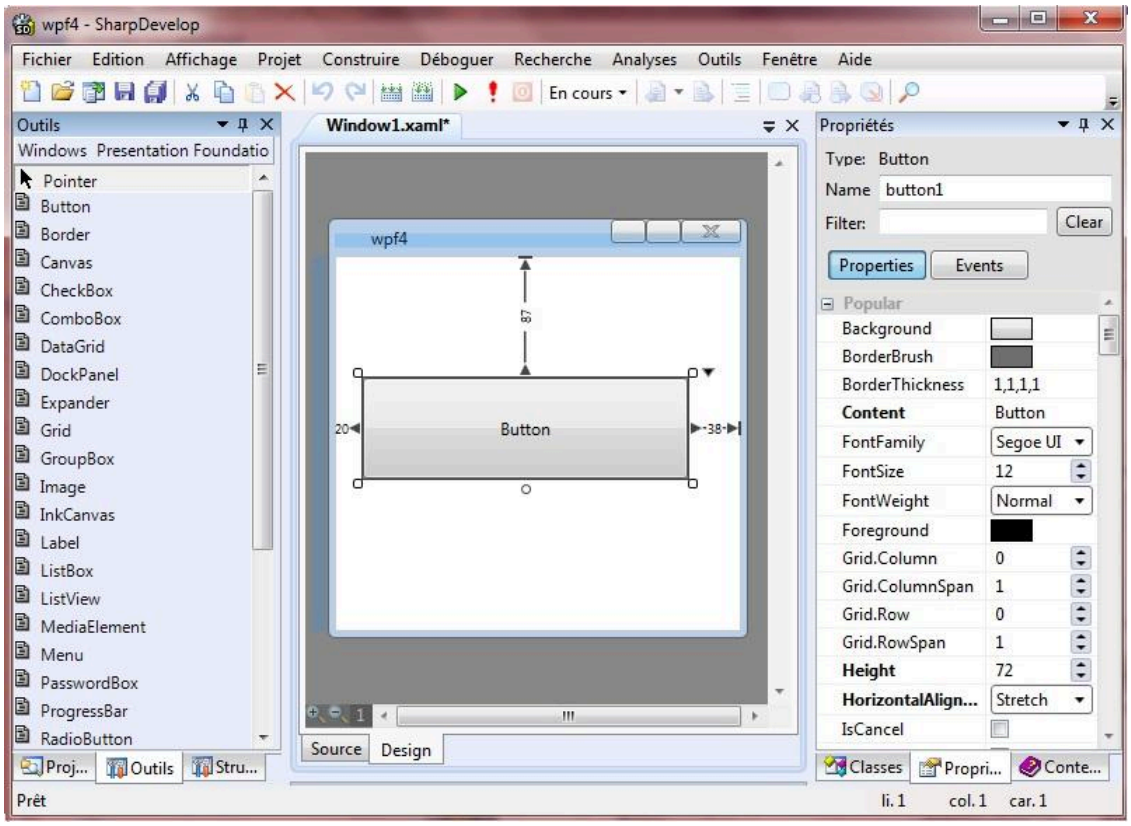
Interface WPF est présente à partir de la version 4.

Faire 'Fichier', 'Nouveau projet' :



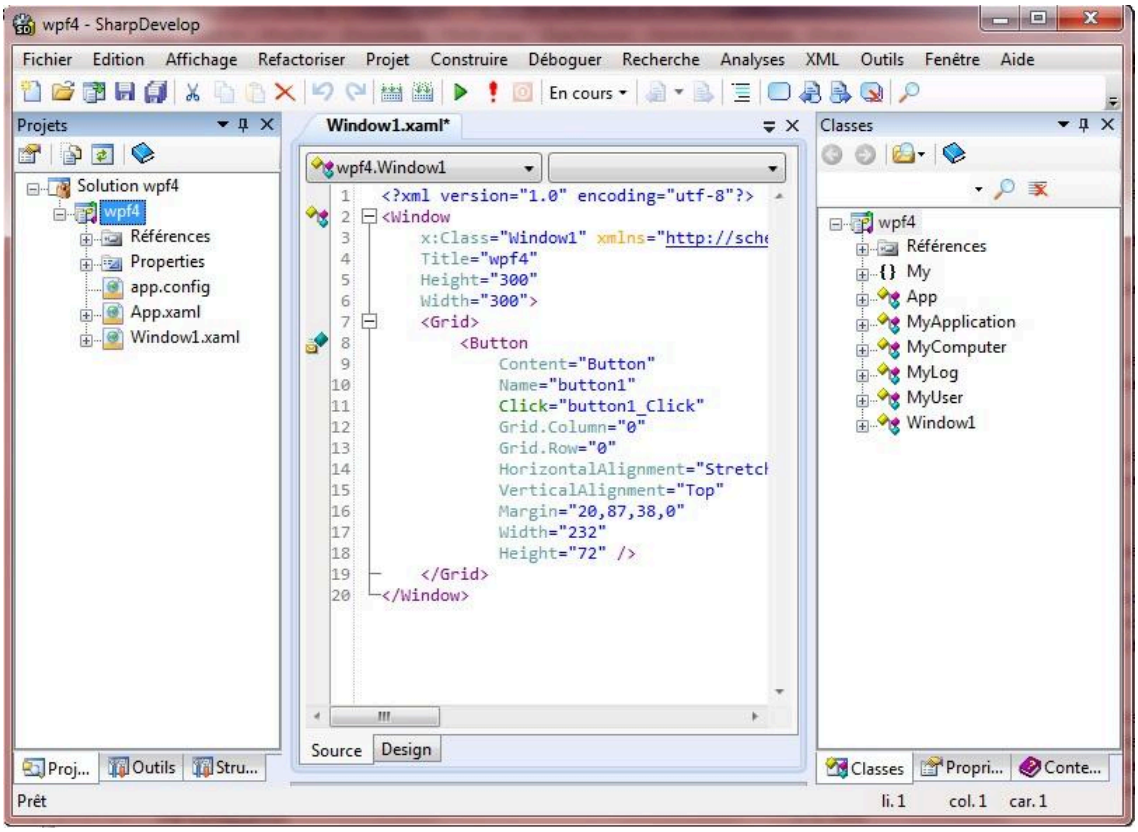

Choisir 'VB', 'WPF' à gauche puis 'Application WPF' à droite, donner un nom en bas.

On se trouve dans l'environnement wpf :




L'onglet 'Window1.xaml' permet de voir le 'Designer'.
Le menu 'Affichage', 'Outils' permet d'avoir la liste des contrôles à gauche: On peut prendre un contrôle, un bouton par exemple, et le déposer dans la fenêtre centrale.
Si on clique sur un contrôle, on voit à droite ses propriétés.

En bas du designer l'onglet 'Source' permet d'avoir accès au code Xaml décrivant l'interface. Ici on voit le code xaml correspondant au bouton :




En bas à gauche, on peut cliquer sur l'onglet 'Projet'.
S'affiche les composants du projet, donnant accès à 'Windows.xaml.vb' contenant le code 'behind' visual basic.

Raccourcis clavier :

Ctrl+Entrée : Aller à la définition;
Ctrl + H: Rechercher;
Ctrl + G: Aller à.
Voir http://wiki.sharpdevelop.net/KeyboardShortcuts.ashx.

## IV-C-4 - Tester son logiciel

On peut compiler le projet avec le premier bouton ci-dessous. Créer le projet avec le second. Lancer l'exécution avec le bouton flèche verte (débogueur actif), le point d'exclamation lance l'exécution sans débogage, le rond à droite (qui devient rouge pendant l'exécution) sert à terminer l'exécution.

La liste déroutante permet de choisir la configuration des fenêtres de l'IDE :

Défaut : c'est les fenêtres habituelles précédemment décrites ;

Débogage : ouvre les fenêtres: variables locales, points d'arrêt, modules chargés… ;

Texte simple: uniquement les fenêtres centrales ;

Éditer : ouvre la fenêtre Edit Layout.

La sauvegarde du projet se fait comme dans tous les logiciels en cliquant sur l'icône du paquet de disquettes.

## IV-C-5 - Fichiers, Chemins des sources

Avant, en #develop 1 :

.prjx est le fichier de projet.

.cmbw est le fichier solution.

Avec Sharpdevelop 2 c'est comme en VB : les solutions sont maintenant des fichiers .sln

.vb sont tous les fichiers Visual Basic (Feuille module…)

Les sources sont par défaut dans ' C:\Documents and Settings\NomUtilisateur\Mes documents\SharpDevelop Projects'

Si on compile le projet l'exécutable est dans un sous-répertoire \Bin\Debug ou \Bin\Realese.

Si vous avez plusieurs versions du framework sur votre machine (version 1.0, version 1.1, voire version 2.0 bêta), il vous est possible de choisir le compilateur dans les options du projet.

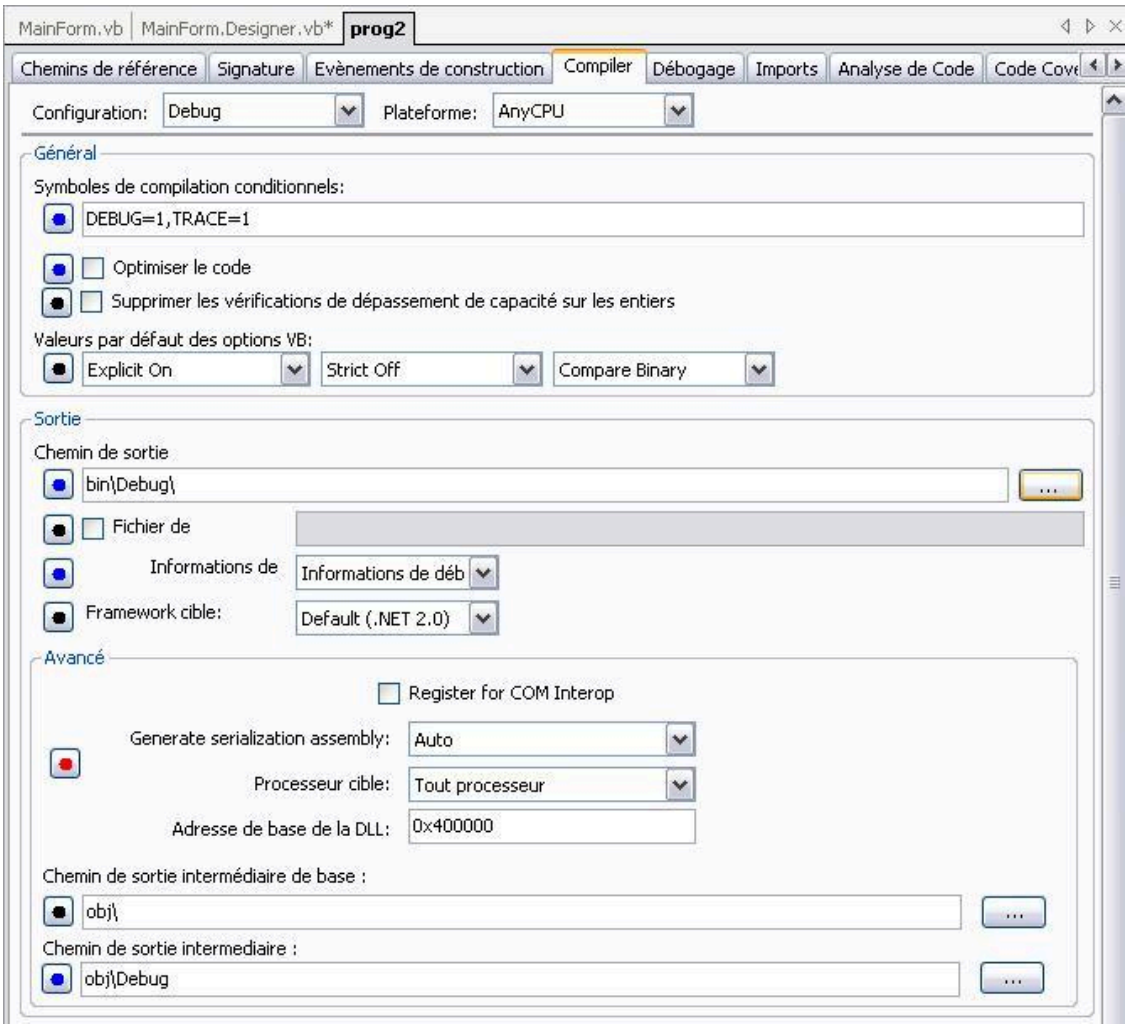
Visual Studio 2003 à version 1.1 du framework.

Visual Studio 2005 à version 2.0 du framework.

## IV-C-6 - Propriétés du projet

Menu 'Projet', 'Option du projet' permet l'accès aux propriétés du projet en cours.

Le quatrième onglet (Compiler) est le plus intéressant :




On peut :

compiler le programme en mode 'Debug' ou 'Release' ;

forcer le programmeur à travailler en Option Strict= On (empêcher les conversions automatiques) ;

Option Explicit=On (Forcer la déclaration des variables) ;

Choisir le Framework avec lequel on travaille (1 ou 2, pas le trois encore) ;

…

Dans l'onglet Import, on peut importer des espaces de noms.

## IV-C-7 - #Develop propose des aides

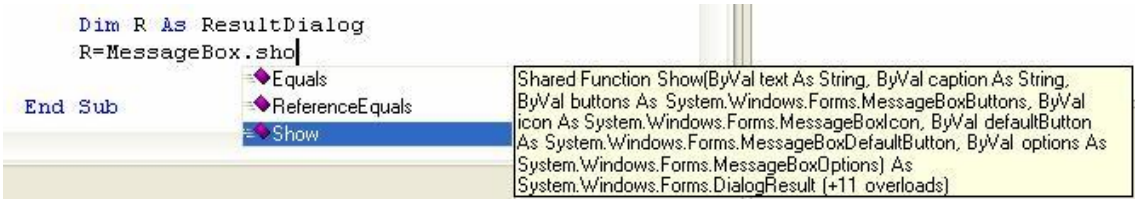
La fenêtre d'aide à droite donne accès à des aides :

de #develop en anglais, non à jour !! ;

du Framework ;

de zipLib.

Si vous avez installé le SDK (SDK Framework .Net et/ou SDK Direct X), vous avez accès à l'aide (partie en haut à droite de l'écran), et donc également à l'intellisense, qui affiche les propriétés, les méthodes des objets, les paramètres des fonctions, des types… des différents objets.



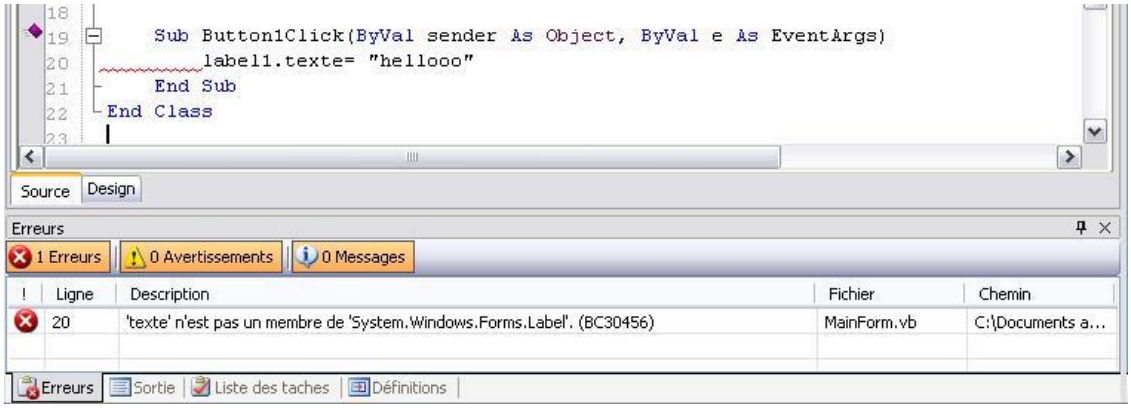

Ici par exemple on a tapé MessageBox. , la liste des membres (Equals, Show…) est affichée.

## IV-C-8 - Erreur de compilation

Si on fait une faute dans le code, elle est détectée lorsque l'on lance l'exécution.

Ici on a tapé 'Texte' à la place de 'Text'.

La ligne en cause est soulignée en rouge et la fenêtre des erreurs située en bas s'ouvre, elle indique et décrit l'erreur :




L'aide dynamique à droite propose des liens en rapport avec le contexte.

## IV-C-9 - Erreur d'exécution : Exceptions

S'il y a une erreur d'exécution (division par zéro par exemple), l'exécution s'arrête et la fenêtre d'exception s'ouvre :


On peut choisir d'arrêter le programme, de continuer, d'ignorer.

## IV-C-10 - Débogage

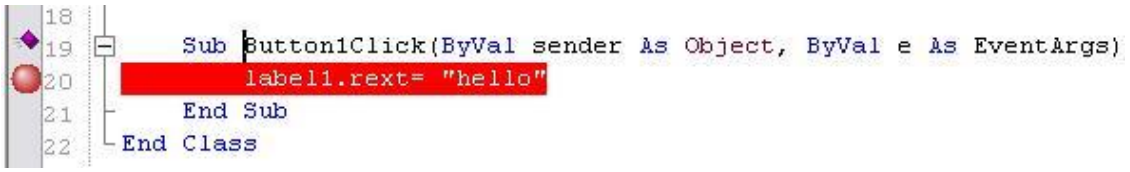
Le débogueur est maintenant intégré dans la version 2.

Une fois l'exécution lancée, on peut :

suspendre l'exécution par ALT+CTRL+B , reprendre par F6

**Ajouter des points d'arrêt**

Grâce à des points d'arrêt (pour définir un point d'arrêt en mode de conception, cliquez en face d'une ligne dans la marge grise, cela fait apparaitre un rond et une ligne rouge. Quand le code est exécuté, il s'arrête sur cette ligne).



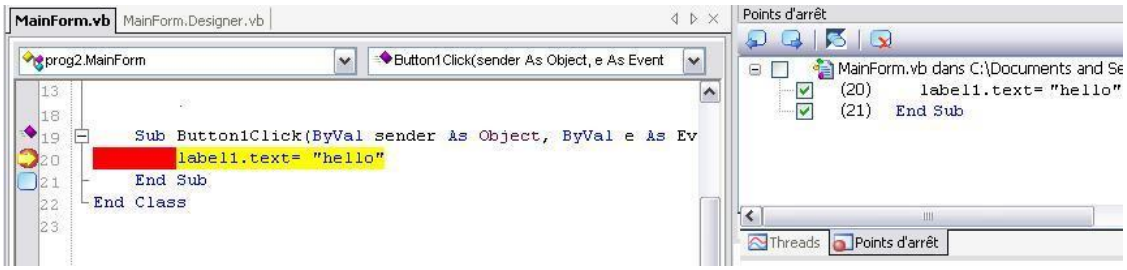

(Recliquer sur le rond pour l'enlever).

**Ajouter des marques pages**

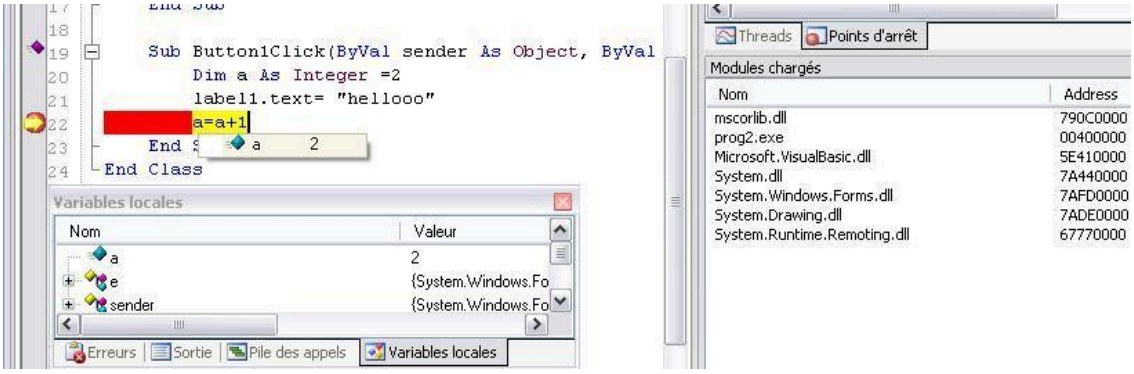
On peut ajouter des marques pages, en cliquant (quand on est sur la ligne à marquer) sur le petit carré bleu de la barre d'outils :

 Ensuite, on peut se déplacer de marque en marque avec les 2 boutons qui suivent.

En mode 'Run', si on clique sur l'onglet 'Points d'arrêt' à droite, on voit la liste des points d'arrêt et des marques, on peut rendre inactif tous les points d'arrêt (3e bouton) ou un seul en le décochant dans la liste.




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




En plus en mode Run, la fenêtre 'Variables locales' située en bas affiche la valeur de toutes les variables de la procédure. (Y compris 'e' et 'sender' qui sont les paramètres de la Sub.)

Enfin à droite on peut voir les modules chargés et les threads.

**Exécution pas à pas**

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.

On peut aussi utiliser les boutons : 

> ⚠ *Attention, il n'est pas possible de modifier les fichiers sources à partir du moment où vous avez démarré le débogage.*

Fonctions présentes dans #develop 1, mais pour l'instant absente dans #develop 2: C++ NProf Wix NAnt, générateur de MessageBox.

Créer un installateur (en anglais).

## IV-C-11 - Conclusion

Programme permettant de faire du VB.net gratuitement (rapport qualité/prix infiniment élevé).

**CONCLUSION D'UN UTILISATEUR**

SharpDevelop est un IDE agréable à utiliser, pour le développement des programmes .NET, en mode WYSIWYG.

Il est possible d'atteindre un niveau de qualité équivalent à Visual Studio ou à Borland C# Builder en faisant une installation complète. Très ouvert, on peut lui rajouter des plugins. Certains programmes externes peuvent être utilisés également avec Visual Studio ou Borland C# Builder.

SharpDevelop est en perpétuelle évolution.

Un forum permet de déposer le descriptif des erreurs rencontrées, mais également de vos demandes de modifications, si vous pensez à une évolution qu'il serait bien que SharpDevelop possède. En plus vous pouvez récupérer le code source et pouvez donc modifier à loisir l'IDE.

Bien sûr, pour les débutants, il manque les assistants de Visual Studio (Crystal report, ADO .NET…). Le problème avec les assistants est qu'une fois qu'on pratique un peu, ils deviennent vite une gêne, et souvent, il faut repasser derrière eux, pour enlever le superflu de code qu'ils ont écrit (souvent ils n'optimisent pas le code).

Il manque également la partie UML de Visual Studio Architecte, mais là on attaque le haut du panier des développeurs.

**Par contre, SharpDevelop apporte en plus :**

- aide à la génération automatique des MessageBox ;
- aide à la conversion C# vers VB.NET et de VB.NET vers C# ;
- aide à la génération d'expression régulière.

**Il fournit les logiciels :**

- NDoc : permet de faire des fichiers d'aide compilée au format MSDN, à partir de lignes commentées dans le code ;
- NUnits : permet de faire des tests unitaires (!) ;
- SharpQuery : permet de se connecter aux bases de données .

## IV-C-12 - J'ai besoin d'aide

Comment créer facilement un installateur (SetUp) avec #develop ? Certains utilisateurs utilisent **Inno Setup**, d'autres **Excelcior delivery**.

Comment utiliser NDoc NUnits ?

Comment utiliser simplement des ressources ?

Comment utiliser des bases de données ?

Qui utilise le menu 'Outils' et ses différentes options ?

Merci à Fabrice SAGE pour son aide.

Merci à Hubert WENNEKES, CNRS Institut de Biologie de Lille pour son aide.

**Remarque pour les forts**

On peut s'étonner qu'il n'y ait pas Handles Button1.Click à la fin de la ligne suivante (comme dans VB 2005) :

```
Sub Button1Click(ByVal sender As Object, ByVal e As EventArgs)
```

```
End Sub
```

En fait si on va voir dans InitializeComponent, il y a un AddHandler après la description du bouton.

```
Private Sub InitializeComponent()

….

AddHandler Me.button1.Click, AddressOf Me.Button1Click
```

# V - Langage Visual Basic

## V-A - Introduction

Nous allons étudier :

Le langage Visual Basic.Net qui est utilisé dans les procédures.

**Comme nous l'avons vu, le langage Visual Basic sert à**

- agir sur l'interface (Afficher un texte, ouvrir une fenêtre, remplir une liste, un tableau, poser une question).
  Exemple afficher 'Bonjour" dans un label :

```
Label1.Text="Bonjour"
```

- effectuer des calculs, des affectations en utilisant des variables.
  Exemple: Mettre dans la variable B la valeur de A+1

```
B=A+1
```

- faire des tests avec des structures de décision: évaluer des conditions des comparaisons et prendre des décisions.
  Exemple: SI A=1 …

```
If A=1 Then… End If
```

- travailler en boucle pour effectuer une tâche répétitive.
  Exemple faire 100 fois…

```
For I=0 To 100…. Next I
```

**Comme nous l'avons vu, le langage Visual Basic sert à**

Tout le travail du programmeur est là.

Dans VB.Net nous avons à notre disposition deux sortes de choses:

### V-A-1 - Les Classes du framework

Le Framework (un framework est un ensemble de classes) en est à sa version 4 en VB 2010.

Les classes du Framework permettront de créer des objets de toutes sortes: objet 'chaine de caractères', objet 'image', objet 'fichier'… On travaille sur ses objets en utilisant leurs propriétés, leurs méthodes.

Il existe des milliers de classes: les plus utilisées sont les classes 'String' (permettant de travailler sur des chaines de caractères), Math (permettant d'utiliser des fonctions mathématiques), Forms (permettant l'usage de formulaires, de fenêtres et donnant accès aux contrôles: boutons, cases à cocher, listes…).

Elles sont communes à tous les langages utilisant le FrameWork (VB, C#, C… ).

Ces classes ont de multiples méthodes (rappel de la syntaxe: Classe.Methode).

Exemple d'utilisation de la Class TextBox (contrôle contenant du texte) et de sa méthode Text :

```
TextBox1.Text="Hello"
```

'Affiche "Hello" dans le Textbox.

Parfois la Classe n'est pas chargée par défaut au démarrage de VB, il faut dans ce cas 'l'importer' en haut du module (au-dessus de Public Class…). Si par exemple, je veux utiliser Sin() de la classe Math, il faut écrire en haut du module :

```
Imports System.Math
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles MyBase.Load
        'Je peux utiliser Sin
        Console.WriteLine(Sin(1).ToString)

End Sub
End Class
```

Si on n'a pas importé Math, on peut quand même utiliser Sin en tapant Math.Sin().

## V-A-2 - Les instructions de Microsoft.VisualBasic

Vb permet d'utiliser **des instructions Visual Basic**; seul VB peut les utiliser et de lui seul (pas C#).

Il s'agit d'instructions, de mots-clés qui ont une syntaxe similaire au basic, mais qui sont du VB.Net.

Exemple :

```
A = Mid(MaString, 1, 3)
```

'Mid retourne une partie de la chaine de caractères.

Il y a aussi les **Classes de compatibilité VB6**. Elles ne dépayseront pas ceux qui viennent des versions antérieures de VB, car elles reprennent la syntaxe utilisée dans VB6 et émulent les fonctions VB6 qui ont disparu de VB.Net. Ce sont des fonctions VB6 qu'on ajoute à VB.net par souci de compatibilité, mais ce n'est pas du VB.Net. Il faut les oublier !

L'outil d'import automatique de VB6 vers VB.Net en met beaucoup dans le code. Il faut à mon avis éviter de les utiliser, car ce n'est pas vraiment du VB. Ce cours 'pur' VB.Net n'en contient pas.

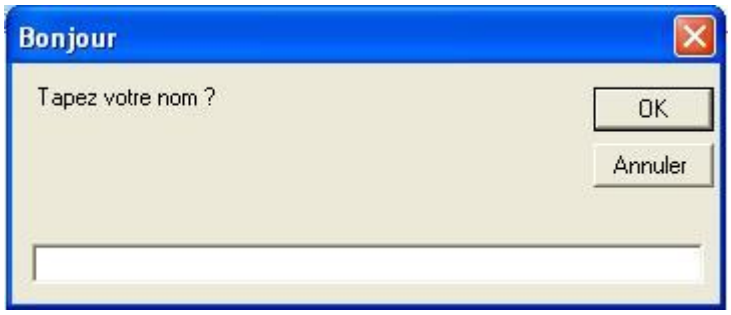
Pour le moment cela peut paraitre un peu compliqué, mais ne vous inquiétez pas, cela va devenir clair.

## V-A-3 - Saisir, Afficher

Dans l'étude du langage VB, on s'occupe du code, on ne s'occupe pas de l'interface (les fenêtres, les boutons, l'affichage du texte…), mais parfois, on a besoin, pour faire fonctionner des exemples de code, de saisir des valeurs, de les afficher :

- **Saisir une valeur**, pour cela on utilise une InputBox, c'est une boite qui s'ouvre, l'utilisateur y tape un texte puis il clique sur 'OK' ; on retrouve ce qu'il a tapé dans la variable Réponse.

```
Réponse= InputBox()
```




- **Afficher des résultats**, pour le moment on affichera du texte de la manière suivante :

dans une fenêtre, dans des TextBox :

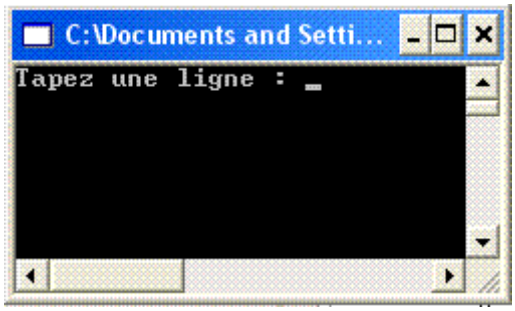
```
TextBox1.Text="TextBox1"
```


ou un label :

```
Label1.Text="Résultat"
```

sur la console :

```
Console.WriteLine ("Résultat")
```




ou

dans une Boite de message :

```
MsgBox("Bonjour")

'ou
MessageBox.Show ("Bonjour")
```

# V-B - Les 'Algorithmes'

Ici nous allons étudier les principes généraux de la programmation, ils sont valables pour tous les langages de programmation. Ici il faut simplement comprendre le principe de ce qui est expliqué.

## V-B-1 - Pour écrire un programme

**Pour écrire un programme, aller du problème à résoudre à un programme exécutable, il faut passer par les phases suivantes :**

- Analyse du cahier des charges
  Il doit être clair, exhaustif, structuré.
- Analyse générale du problème
  Il existe des méthodes pour professionnels (MERISE, JACKSON…), nous utiliserons plutôt l'analyse procédurale: le problème global est découpé en sous-problèmes nommés fonctions. Chaque fonction ne contient plus qu'une partie du problème. Si une fonction est encore trop complexe, on itère le processus par de nouvelles fonctions à un niveau plus bas.
  Cela s'appelle la 'Conception structurée descendante'. La 'Conception ascendante' existe aussi: en assemblant des fonctions préexistantes, on résout le problème: attention, il faut que les fonctions préexistantes soient cohérentes. (Pour le moment on ne fait pas de programmation objet).
- Analyse détaillée
  Chaque fonction est mise en forme, la logique de la fonction est écrite dans un pseudo langage (ou pseudocode) détaillant le fonctionnement de la fonction. Ce pseudocode est universel, il comporte des mots du langage courant ainsi que des mots relatifs aux structures de contrôle retrouvées dans tous les langages de programmation.
- Codage
  Traduction du pseudocode dans le langage que vous utilisez.
- Test
  Car il faut que le programme soit valide.

**Exemple simpliste**

- Analyse du cahier des charges.
  Création d'un programme affichant les tables de multiplication, d'addition, de soustraction.
- Analyse générale du problème.
  Découpons le programme en diverses fonctions:
  Il faut créer une fonction 'Choix de l'opération', une fonction 'Choix de la table', une fonction 'TabledeMultiplication', une fonction 'TabledAddition', une fonction 'Affiche'…
- Analyse détaillée.
  Détaillons la fonction 'TabledeMultiplication'
  Elle devra traiter successivement (pour la table des 7 par exemple)
  1X7
  2X7
  3X7…
  Voici l'algorithme en pseudocode.

```
Début

    Pour i allant de 1 à 10

        Ecrire (i*7)

    Fin Pour

Fin
```

**Exemple simpliste**

- Codage.
  Traduction du pseudocode en Visual Basic, en respectant la syntaxe du VB.

```
Sub MultiplicationPar7

Dim i As Integer

For i=1 to 10

    Call Affiche(i*7)

next i.

End Sub
```

**Exemple simpliste**

- Test
  Ici il suffit de lancer le programme pour voir s'il marche bien…

Pour des programmes complexes, il existe d'autres méthodes.

## V-B-2 - Définition de l'algorithme

**Un problème est traitable par informatique si :**

- on peut parfaitement définir les données (entrées) et les résultats (sorties) ;
- on peut décomposer le passage de ces données vers ces résultats en une suite d'opérations élémentaires exécutables.

*L'algorithme décrit le processus de résolution d'un problème permettant de décrire les étapes vers le résultat.*

*L'algorithme détaille, le fonctionnement de ce passage et en décrit la logique.*

L'algorithme est une succession de tests, décisions et actions dans le but de décrire le comportement d'une entité (objet, programme, personne). Définition du Dicodunet.

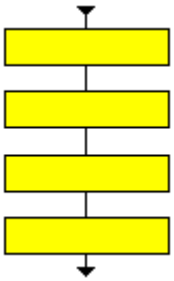
On écrit bien 'algorithme' et non 'algorythme'.Le mot "algorithme" vient du nom du mathématicien arabe Al Khuwarizmi (latinisé au Moyen Âge en Algoritmi).

Pour représenter l'algorithme et ses opérations, on utilisait les organigrammes (dessins avec des cases et des flèches qui ne permettaient de visualiser que des problèmes très simples); maintenant on utilise du pseudocode (composé d'instructions généralistes).
La mise en œuvre de l'algorithme consiste en l'écriture de ces opérations dans un langage de programmation (le Visual Basic pour nous).
On utilise l'anglicisme **implémentation** pour désigner cette mise en œuvre.

Étudions cette logique valable pour tous les langages de programmation (ceux qui sont des langages impératifs).

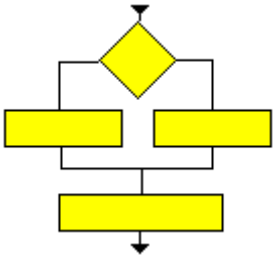
**Pour représenter n'importe quel algorithme, il faut disposer des trois possibilités suivantes :**

- la **séquence** qui indique que les opérations doivent être exécutées les unes après les autres.



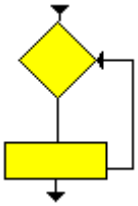
**Pour représenter n'importe quel algorithme, il faut disposer des trois possibilités suivantes :**

- le **choix** qui indique quelles instructions doivent être exécutées en fonction de circonstances.



**Pour représenter n'importe quel algorithme, il faut disposer des trois possibilités suivantes :**

- la **répétition** qui indique que des instructions doivent être exécutées plusieurs fois.



**Pour représenter n'importe quel algorithme, il faut disposer des trois possibilités suivantes :**

Exemple d'algorithme principalement composé d'une répétition:

```
Pour i allant de 1 à 10
        Ecrire (i*7)
Fin Pour
```

Voyons cela en détail.

Le langage algorithmique et son pseudocode ne sont pas vraiment standardisés, chacun écrit le pseudocode à sa manière, aussi vous verrez des notations différentes dans les divers cours d'algorithme. Cela n'a pas d'importance, car un programme en pseudocode ne sera jamais exécuté sur une machine.

L'intérêt d'étude des algorithmes est didactique: elle permet de comprendre la logique d'un programme totalement indépendamment du langage: ce qui suit est valable en C++, Delphi, Java, Visual Basic, Assembleur… Comme on est dans un cours de VisualBasic, je donnerais pour chaque notion le pseudocode, mais aussi l'équivalent en Visual Basic.

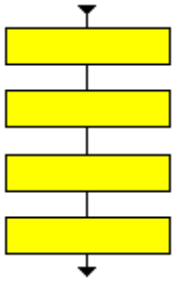
## V-B-3 - Structures élémentaires

Il y a les **structures de données** (dans quoi sont stockées les données?) dans des :
-Constantes ;

-Variables ;
-Tableaux ;
-Collections ;
-Listes, Graphe, Arbre…
Il y a les **structures de contrôles** (Comment fonctionne le code ?) :
-Séquence ;
-Condition ;
-Boucle.

## V-B-3-a - Séquences

Au sein d'un programme, la structure est généralement séquentielle.



On fait de la programmation **impérative**, on travaille sur le modèle de la **machine de Turing**, avec une mémoire centrale et des instructions qui modifient son état grâce à des assignations successives.

Le code est fait d'une succession de lignes (ou instructions) qui seront lues et traitées les unes après les autres.

Instruction 1

Instruction 2

Instruction 3

…

Quand le programme s'exécute, il le fait de haut en bas, Instruction 1 sera exécuté puis instruction 2 puis instruction 3…

En VB on peut mettre plusieurs instructions sur la même ligne, séparées par ":"

Instruction1 : Instruction2

## V-B-3-b - Variables, 'Type' de variable

Les 'Variables' contiennent les informations les données nécessaires au déroulement du programme (c'est le même sens qu'en mathématiques, à la différence qu'en informatique une variable ne contient qu'une valeur).

Chaque variable a un **Nom** (identifiant) et un **Type**. Ce dernier indique la nature de l'information que l'on souhaite mettre dans la variable.

Un type indique :

- la nature de l'information (un chiffre ? du texte ?) ;

- les valeurs que peut prendre la variable (un entier, un réel…) ;

-les opérations possibles (addition, concaténation…).

Exemple : le Type 'Entier' (Integer en VB) peut contenir une valeur entière, positive ou négative, les opérations possibles sont l'addition, la soustraction, la multiplication…Ainsi si je crée une variable de type Entier, je sais que je ne pourrai y mettre qu'un entier et que je pourrai faire une addition avec, je ne pourrai pas y mettre de caractères.

Les types disponibles sont :

**Type numérique**

'Entier', 'réel'… (Integer, Single en VB) Exemple d'un entier: 123 ;

**Type alphanumérique**

'Caractère' (Char en VB) contient 1 caractère Exemple d'un caractère: 'a' (avec des guillemets)

'chaine de caractères',(String en VB), contient plusieurs caractères. Exemple: 'toto' (avec des guillemets) ;

**Booléen** (Boolean en VB) ne peut contenir que 'Vrai' ou 'Faux' ;

**Objet** (Object en VB) ;

**Monétaire** (Décimal en VB) ;

**Date** (Date en VB) ;

**Matrice, nombre imaginaire** (depuis VB 2010).

À partir des types précédents, on peut créer des **types complexes** (ou structurés) :

- les **Tableaux** (Array) qui contiennent plusieurs éléments ;

- les **Collections** qui contiennent plusieurs éléments aussi.

Exemple : la variable nommée 'Total' contient un réel dans un programme de comptabilité.

On remarque qu'il ne faut pas confondre 1 qui est une valeur numérique( sans guillemets) et "1" qui est le caractère '1' (avec des guillemets).

**Utilisation des variables**

Les variables numériques serviront à faire des calculs.

Les variables alphanumériques (String et Char du VB) serviront entre autres à manipuler et afficher du texte.

Comment afficher les résultats de calcul ?

On apprendra à transformer des variables numériques en variables alphanumériques.

Pour utiliser une variable, il faut qu'elle existe, **il faut donc la créer**, on dit il faut la **déclarer**.

Dans un algorithme : 'Variable A en Numérique' 'crée une variable nommée A et de Type Numérique.

En VB : 'Dim A As Integer' 'crée une variable nommée A et de Type Integer.

On peut aussi **initialiser une variable,** c'est-à-dire définir sa valeur initiale.

Pour cela on peut utiliser **un littéral :** c'est une donnée utilisée directement.

X <- 2 veut dire : donner à la variable X la valeur 2 (2 est une littéral).

## V-B-3-c - Constantes

Comme une variable, une 'Constante' a un Nom (identifiant) et un Type. Elle contient une valeur: un nombre, une chaine de caractères…

**Son contenu ne peut pas être modifié.**

Exemple :'Constante A en Numérique =12'.

En VB : 'Const A As Integer =12'.

On la déclare et on l'initialise en même temps.

Ensuite on ne peut pas modifier sa valeur, on ne peut que la lire. Les constantes sont utilisées dans les programmes pour conserver des valeurs fixes qui n'ont pas à changer. Si j'ai un programme d'astronomie, je créerai une constante contenant la vitesse de la lumière pour faire mes calculs (elle a peu de chances de changer !).

## V-B-3-d - Affectation (ou Assignation)

C'est une instruction consistant à **donner une valeur à une variable**.

En langage algorithmique on l'indique par '<-'

X <- 2 veut dire : donner à la valeur X la valeur 2 (2 est une littéral).

Z <- X veut dire : donner à la variable Z la valeur de la variable X.

Z <- X+1 veut dire : donner à la variable Z la valeur de la variable X à laquelle on ajoute 1 (Z prendra la valeur 2+1 =3).

Cela revient à **évaluer l'expression de droite** et à en mettre la valeur dans la variable de gauche.

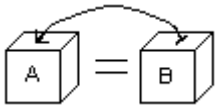
En VB le signe d'affectation est '=' on écrit donc :

```
Z=X+1
```



*Attention le signe '=' utilisé en VB est ambiguë et n'a donc pas le même sens qu'en mathématiques.*

Exemple Visual Basic: A=B




*Attention ce n'est pas une égalité, mais une affectation.*

L'affectation ne marche que si le type de variable est le même :

Variable A en Numérique

Variable B en Numérique

B<-12

A<-B 'fonctionne, car B contient 12, on met 12 dans A

Variable A en Numérique

Variable B en Alphanumérique

B<-'toto'

A<-B 'ne fonctionne pas, car on tente de mettre le contenu de B qui est alphanumérique dans une variable numérique.

**L'affectation sert à effectuer des calculs**

Variable A en Numérique.

A<-3+4-2 'L'expression à droite est évaluée et son résultat est affecté à la variable A.

Ici les + - sont des opérateurs, il y en a d'autres : * (multiplier), / (diviser)….

## V-B-3-e - Booléens

On a parfois besoin de savoir si une assertion est vraie ou Fausse (True ou False).

Pour stocker une information de ce type, on utilise une variable de type booléen. Une variable de ce type ne peut contenir que True ou False.

Le terme booléen vient de "l'algèbre de Boole", cette algèbre ne travaille que sur les valeurs 1 ou 0 (True ou False).

Soit B une variable booléenne.

On peut écrire B<-True (B=True en VB).

On peut aussi tester cette variable.

Si B=False alors (If B=False Then… en VB).

L'expression après 'Si' est évaluée, si elle est vraie 'alors' se produit.

Autre exemple

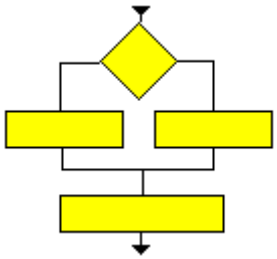
Si C=2 alors… (If C=2 Then …en VB).

L'expression C=2 est évaluée, si C est effectivement égal à 2, C=2 est évalué et prend la valeur True, dans ce cas le programme se poursuit après Then.

si C est différent de 2, C=2 est évalué et prend la valeur False, dans ce cas le programme ne se poursuit pas après Then.

## V-B-3-f - Choix : Si…Alors

Le programme doit pouvoir choisir parmi deux ou plusieurs possibilités en fonction d'une condition :



```
Si Condition Alors

      Action 1

Sinon

      Action 2

Fin Si
```

Si Condition est vraie Action 1 est effectuée, sinon Action 2 est effectuée.

Parfois il n'y a pas de seconde branche :

```
Si Condition Alors

      Action 1

Fin Si
```

ou sur une seule ligne :

```
    Si Condition Alors Action 1
```

Il peut y avoir plusieurs conditions imbriquées :

```
Si Condition 1 Alors
    Si Condition 2 Alors

            Action 1

    Sinon

            Action 2

    Fin Si
```

```
Sinon

      Action 3

Fin Si
```

Noter bien le retrait des lignes de la seconde condition afin de bien visualiser la logique du programme :

Action 2 est effectuée si la Condition 1 est remplie et la Condition 2 n'est pas remplie.

En VB cela correspond à l'instruction IF THEN

```
If Condition 1 Then

      Action 1

Else

      Action 2

End If
```

Remarque sur les conditions

Une condition contient deux valeurs et un opérateur :

Si C>2 Alors est correcte.

Si B=3 Alors est correcte.

Si 2<B<7 Alors est incorrecte, car il y a deux opérateurs, il faut dans ce cas utiliser plusieurs conditions et des opérateurs logiques.

Si B>2 Et B<7 Alors est correcte (If B>2 And B<7 Then en Visual Basic).

La condition est évaluée.

Exemple : soit l'expression Si C>2 Alors , elle sera évaluée, si C contient 3, C>2 est vérifié donc Vrai.

Exemple : trouver le plus grand nombre entre x et y et le mettre dans max

```
Variable x en Numerique

Variable y en Numerique

Variable max en Numerique

Si x>y Alors

    max<-x

Sinon

    Max<-y

Fin Si
```

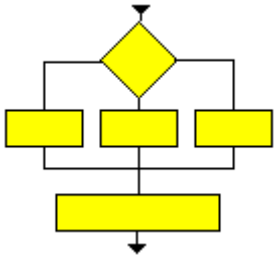
En VB

```
Dim x As Integer
```

```
Dim y As Integer

DIm max As Integer

if x>y Then

    max=x

Else

    max=y

End if
```

## V-B-3-g - Choix : Décider entre

Il est parfois nécessaire d'effectuer un choix parmi plusieurs solutions :



```
Décider Entre
      Quand Condition 1 Alors

            Action 1

      FinQuand


      Quand Condition 2 Alors

            Action 2

      FinQuand

      …

      …

      Autrement

            Action 4

      FinAutrement

FinDécider
```

Si la condition 1 est remplie, Action 1 est effectuée puis le programme saute après FinDécider.

Si la condition 1 n'est pas remplie, on teste la condition 2…

Si aucune condition n'est remplie, on saute à Autrement, on effectue Action 4.

On pourrait aussi parler de sélection :

```
Sélectionner.

      Le cas : condition 1
```

```
            Action 1
      Le cas : condition 2
            Action 2
      …
      Les autres cas
FinSélectionner
```

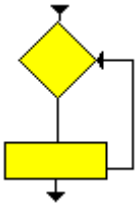
En VB cela correspond à

```
Select Case Valeur
      Case condition 1
            Action 1
      Case condition 2
            Action 2
      …
      Case Else
            Action 4
End Select
```

Si Valeur=Condition 1 Action 1 est effectuée,si Valeur=Condition 2 Action 2 est effectuée…

## V-B-3-h - Répétitions : Pour…Répéter

Permet de répéter une séquence un nombre de fois déterminé :



Le cas le plus classique est :

```
Pour I variant de 0 à N Répéter
      Action
FinRépéter
```

I prend la valeur 0, 'Action' est effectuée,

puis I prend la valeur 1, Action est effectuée,

puis I prend la valeur 2…

cela jusqu'à N.

La boucle tourne N+1 fois (car ici on commence à 0 ).

Cela se nomme une itération.

Intérêts ?

Au lieu de faire :

Afficher (1*7)

Afficher (2*7)

Afficher (3*7)

Afficher (4*7)

…

on remarque qu'un élément prend successivement la valeur 1, 2, 3…

Une boucle peut faire l'itération :

```
Pour i allant de 1 à 10 Répéter

    Affiche (i*7)

Fin répéter
```

La variable dite 'de boucle' prend bien les valeurs 1 puis 2 puis 3… ; elle est utilisée dans le corps de la boucle.

Une instruction Sortir permet de sortir prématurément de la boucle.

En VB

```
For i=0 To N

      …

Next i
```

L'instruction Exit For permet de sortir prématurément de la boucle.

On peut aussi boucler en parcourant tous les éléments d'une collection.

(Une collection est une liste d'objets, liste de taille variable en fonction de ce qu'on ajoute ou enlève.)

```
Pour Chaque élément de la liste

      Action

Fin Pour
```

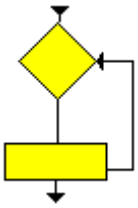
En VB :

```
For Each élément In list
```

```
Next
```

## V-B-3-i - Répétitions : Tant que

Permet de faire une boucle sans connaitre le nombre d'itérations à l'avance.



```
Tant Que Condition

      Action

Fin Tant Que
```

L'action qui est dans la boucle doit modifier la condition afin qu'à un moment 'Tant que' ne soit pas vérifié et que l'on sorte de la boucle. Sinon la boucle tourne sans fin.

Pour plus cadrer avec la réalité :

```
Faire tant que condition

      Action

Boucler
```

En VB :

```
Do while Condition

      Action

Loop
```

Il existe une boucle équivalente :

```
Répéter

      Action

Jusqu'à Condition


En VB :

Do

      Action

Loop Until Condition
```

Une instruction Exit Do permet de sortir prématurément de la boucle.

## V-B-3-j - Logique : Et, Ou, Non

Une condition contient deux valeurs et un opérateur :

Si C>2 Alors… est correcte.

Si B=3 Alors est correcte.

Si 2<B<7 Alors est incorrecte, car il y a deux opérateurs, il faut dans ce cas utiliser plusieurs conditions et des opérateurs logiques :

Si B>2 Et B<7 Alors est correcte (If B>2 And B<7 Then en Visual Basic)

La condition est évaluée.

Exemple : Soit l'expression Si C>2 Alors , elle sera évaluée, si C contient 3, C>2 est vérifiée donc Vrai.

**ET**

On a vu Si B>2 Et B<7 Alors

Il existe aussi :

**OU**

Si B>2 Ou B<7 Alors

et

**NON**

Si NON(B>2) Alors est équivalent à Si B<=2 Alors

En VB on utilise les termes **AND OR NOT**.

## V-B-3-k - Les Sauts

Un saut dans le code correspond à 'Aller à'.

Cela permet de 'sauter' vers un label (une étiquette= un endroit du code).

Exemple :

```
Variable A en Numérique

Variable B en Numérique

Variable C en Numérique

B<-12

A<-B

Aller à Poursuivre

C=11
```

```
Étiquette Poursuivre

A<-A+1
```

Le programme saute de 'Aller à Poursuivre' à 'Étiquette Poursuivre', il n'exécute pas C=11.

En VB on utilise **GoTo** pour faire le saut, pour créer une étiquette, on lui donne un nom et on ajoute ':'…

```
MonEtiquette:

GoTo monetiquette
```

On verra que les sauts ne sont pratiquement plus utilisés.

## V-B-3-l - Programmation structurée

Avant on écrivait :

```
Variable A en Numérique
Variable B en Numérique
Variable C en Numérique

B<-12
A<-B

Si A=B Aller à Poursuivre1
  C<-1
Étiquette Poursuivre1

Si A<>B Aller à Poursuivre2

C<-2

Étiquette Poursuivre2
```

On faisait des sauts dans tous les sens!! Code illisible, non structuré.

Maintenant, on structure et on n'utilise pas de 'Aller à'.

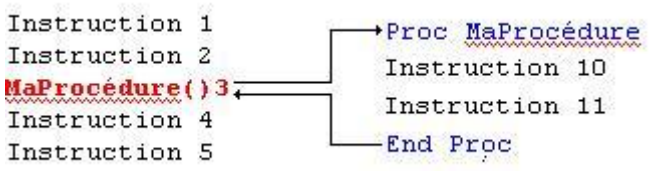
```
Variable A en Numérique
Variable B en Numérique
Variable C en Numérique

B<-12
A<-B

Si A=B Alors
    C<-1
Sinon
    C<-2
Fin Si
```

## V-B-3-m - 'Sous-programme' ou 'procédure'

On a déjà vu cette notion.

Quand on appelle une procédure, le logiciel 'saute' à la procédure, il effectue celle-ci puis revient effectuer ce qui suit l'appel.

Dans un algorithme, une procédure commence par le mot Proc et se termine par End Proc.




Le programme effectuera les instructions 1, 2, 3, 10, 11, 4, 5.

Une opération complexe peut être découpée en plusieurs procédures ou sous-programmes plus petits et plus simples qui seront appelés. Chaque procédure résout un problème.

Et VB les sous-programmes (ou procédures) sont des Sub ou des Function. Pour appeler une procédure, on utilise Call NomProcedure() ou NomProcedure()

On peut fournir aux procédures **des paramètres**, ce sont des variables qui seront transmises à la procédure.

Exemple

Création d'une Procédure 'MaProcédure' recevant deux paramètres :

```
Sub MaProcédure(paramètre1, paramètre2)

…

End Sub
```

Exemple d'appel de la procédure 'Maprocédure' en envoyant deux paramètres :

Call MaProcédure(premierparamètre, secondparamètre)

Exemple : si j'écris Call MaProcédure(2,3)

dans la procédure MaProcédure paramètre1=2 et paramètre2=3.

Il est nécessaire de définir le type des paramètres :

Sub MaProcédure(paramètre1 As Integer, paramètre2 As Integer) indique que la procédure attend deux entiers.

Il y a deux manières d'envoyer des paramètres :

- Par valeur : (By Val)c'est la valeur, le contenu de la variable qui est envoyé.

- Par référence : (By Ref) c'est l'adresse (le lieu physique où se trouve la variable) qui est envoyée. Si la Sub modifie la variable, cette modification sera visible dans la procédure appelante après le retour.

Parfois on a besoin que la procédure appelée retourne une valeur, dans ce cas il faut créer une fonction :

```
Function MaFonction()  As Integer    'MaFonction qui retourne un entier

…

    Return Valeur

End Function
```

Pour appeler la fonction :

ValeurRetournée=MaFonction()

Donc ValeurRetournée est aussi un entier.

Exemple de fonction: créer une fonction qui retourne le plus petit nombre :

```
Fonction Minimum (x en Numerique, y en Numérique) en numérique
Si x<y Alors

    Retourner x

Sinon

    Retourner y

Fin Si

Fin Fonction
```

Pour l'utiliser :

```
Variable Min en Numerique

Min<-Minimum (5,7) 'Appelle la fonction en donnant les 2 paramètres 5 et 7.
```

Min contient maintenant 5

En Vb

```
Function Minimum(x As Integer, y As Integer) As Integer

    If x<y Then

        Return x

    Else

        Return y

    End If

End Function
```

Pour l'utiliser :

```
Dim Min As Integer

Min= Minimum (5,7)
```

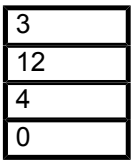
La fonction résout un problème et plus précisément à partir de données, calcule et fournit un résultat.

## V-B-3-n - Tableaux

Un tableau de variables permet de stocker **plusieurs variables de même type** sous un même nom de variable, chaque élément étant repéré par un **index** ou indice.

C'est une suite finie d'éléments.

Soit un tableau A de quatre éléments :




Pour accéder à un élément, il faut utiliser l'indice de cet élément.

L'élément d'index 0 se nomme A[0] et contient la valeur 3.

On remarque que le premier élément est l'élément d'index 0 (ZÉRO).

L'élément d'index 1 se nomme A[1] et contient la valeur 12.

Quand on crée un tableau, il a un nombre d'éléments bien défini : 4 dans notre exemple d'index 0 à 3.

Pour donner une valeur à un des éléments, on affecte la valeur à l'élément.

```
A[2] <- 4
```

Pour lire une valeur dans un tableau et l'affecter à une variable x :

```
x <- A[2]
```

Traduction en VB

```
Dim  A(4) As Integer 'on déclare le tableau
A(2)=4

x = A(2)
```

Pour parcourir tous les éléments d'un tableau, on utilise une boucle.

Exemple : afficher tous les éléments du tableau tab qui contient n éléments.

```
début
          Pour i allant de 0 à n-1 Répéter
              écrire(tab[i])
          Fin Répéter
fin
```

En VB :

```
        For i=0 to n-1

             Affiche ( tab(i))     'routine d'affichage

        Next i
```

Il existe des tableaux multidimensionnels avec plusieurs index :

Voyons les index de chaque élément :

| | | |
|---|---|---|
| B(0,0) | B(0,1) | B(0,2) |
| B(1,0) | B(1,1) | B(1,2) |
| B(2,0) | B(2,1) | B(2,2) |

B[1,0] désigne l'élément de la seconde ligne, première colonne.

Voyons par exemple, le contenu de chaque élément :

| | | |
|---|---|---|
| 3 | 12 | 0 |
| 18 | 4 | 5 |
| 12 | 2 | 8 |

Ici B[1,0] =18

En VB on utilise des parenthèses : B(1,0) =18

## V-B-3-o - Collection

Une collection permet de stocker plusieurs variables ou objets, chaque élément étant repéré par un index ou indice. Mais la collection n'a pas de nombre d'éléments précis au départ, elle ne contient que les éléments qu'on y ajoute.

Soit la collection Col, au départ elle est vide.

J'ajoute des éléments (ou items) à cette collection.

Col.Ajouter ("Toto")

Voici la collection :

| |
|---|
| Toto |

La collection a maintenant 1 élément.

```
Col.Ajouter("Lulu")

Col.Ajouter("Titi")
```

| |
|---|
| Toto |
| Lulu |
| Titi |

La collection a 3 éléments maintenant.

Col.Retirer(2) enlève l'élément numéro 2.

| |
|---|
| Toto |
| Titi |

La collection n'a plus que 2 éléments maintenant.

On voit que le nombre d'éléments n'est pas connu à l'avance, il varie en fonction des éléments ajoutés (ou retirés).

Un élément est repéré par un indice.

En VB

```
Col.Add    'Ajoute un élément

Col.Remove 'Enlève un élément
```

Il existe des collections avec des clés permettant de retrouver la valeur d'un élément rapidement.

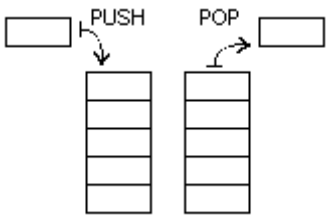
## V-B-3-p - Pile et Queue

Une Pile (ou stack) est une collection de type LIFO (Last In, First Out). Dernier entré, premier sorti.

Ce type de stack (pile) est très utilisé en interne par les programmes informatiques : on stocke dans une stack les adresses de retour des procédures appelées, au retour on récupère l'adresse du dessus.

**Push** insère un objet en haut de la pile.

**Pop** enlève et retourne un objet en haut de la pile.

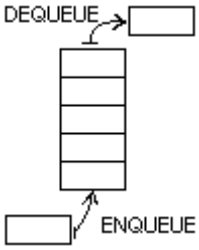



On peut utiliser une pile dans un programme pour gérer le déplacement de l'utilisateur dans un arbre, les éléments en cours sont stockés par Push, pour remonter en chemin inverse, on Pop.

Une 'Queue' est une collection de type FIFO (First In, First Out). Premier arrivé premier servi.

C'est la queue devant un cinéma, le premier arrivé, prend son billet le premier.

Les éléments stockés dans Queue sont insérés à une extrémité; les éléments extraits le sont à l'autre extrémité.




Le nombre d'éléments de la queue est géré automatiquement.

Généralement on a les possibilités suivantes :

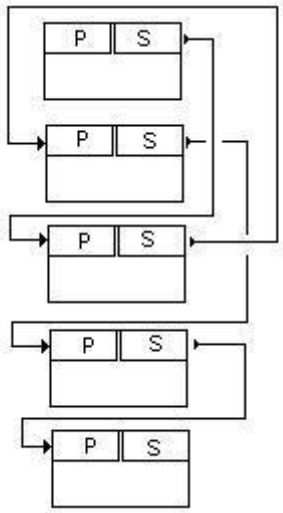
**DeQueue** supprime et retourne l'objet de début de liste

**EnQueue** ajoute un objet en fin de liste

**Peek** retourne l'objet de début sans le supprimer

## V-B-3-q - Liste chainée

Une liste chainée est une liste d'éléments non classée. Dans chaque enregistrement il y a, outre le contenu de l'enregistrement, la localisation, l'adresse, l'index de l'enregistrement précédent et de l'enregistrement suivant.




Ainsi on peut parcourir la liste en allant d'enregistrement en enregistrement. Il existe des algorithmes pour ajouter ou supprimer un enregistrement. La liste peut être ouverte ou fermée (le dernier enregistrement bouclant sur le premier).

## V-B-3-r - Notion de Clé

Quand on classe une série importante de données, on peut utiliser la notion de clé/Valeur (Key/Value).

Ici on utilise comme clé le numéro du département et comme valeur, le nom du département.

| Clé | Valeur |
|---|---|
| 69 | Rhone |
| 75 | Paris |
| 83 | Var |
| 1 | Ain |

Si on a la clé, on peut retrouver la valeur correspondante.

Autre exemple: La clé peut être le nom, prénom.

## V-B-3-s - Notion de Hachage

Une fonction de hachage est une fonction qui fait subir une succession de traitements à une donnée fournie en entrée pour en produire une empreinte servant à identifier la donnée initiale.

**Donnée d'entrée=> Fonction de hachage => Empreinte.**

Le résultat d'une fonction de hachage peut être appelé selon le contexte empreinte, somme de contrôle (dans le cas de la CRC), résumé, condensé, condensat ou encore empreinte cryptographique (dans le cadre de la cryptographique). On l'appelait aussi autrefois aussi signature.

Les fonctions de hachage sont utiles en cryptographie où elles sont utilisées pour chiffrer une donnée initiale.

Mot de passe: "GftUi6h77"=> Fonction de hachage => Empreinte : "4587213399545684246847"

C'est l'empreinte qui va être enregistrée. Quand l'utilisateur rentre le mot de passe, on le 'hache" et on compare l'empreinte du mot de passe tapé par l'utilisateur avec l'empreinte enregistrée pour voir si le mot tapé est bon. Ainsi à aucun moment le mot de passe est en clair.

Ces fonctions de hachage sont aussi très utilisées pour accéder rapidement à une donnée contenue dans un grand nombre de données.

Ceci grâce aux tables de hachage (ou hash tables en anglais).

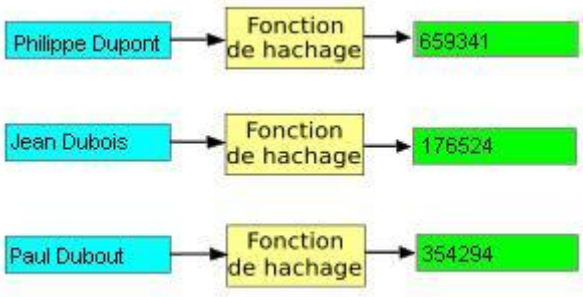
Un exemple classique et simpliste de fonction de hachage est la fonction modulo n : Si on dispose d'un grand nombre de données, à mettre dans un tableau de n cases, on pourra ranger l'élément numéro i dans la case i modulo n. Ainsi, pour aller chercher notre donnée, nous n'avons plus besoin de parcourir tous les éléments jusqu'à trouver l'élément i : Il suffit de parcourir les éléments contenus dans la case i modulo n. Si les données initiales étaient réparties uniformément, le temps de recherche en moyenne est divisé par n. En pratique, on utilise des fonctions de hachage bien plus complexes.

Le hachage est un nombre qui permet la localisation des éléments dans une table.

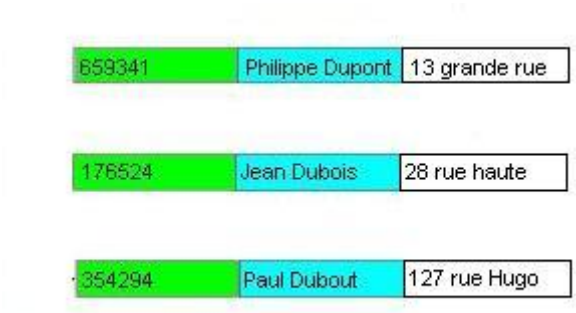
Exemple

Nous avons une série de noms et adresses, nous voulons rapidement trouver l'adresse correspondant à un nom sans avoir à faire une boucle qui compare le nom cherché avec chaque élément du tableau pour le trouver.

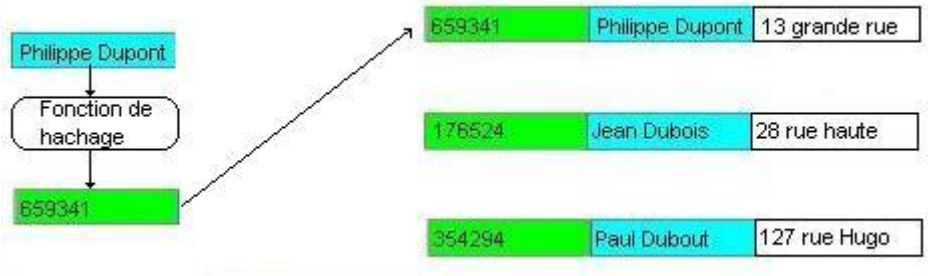
Pour chaque nom la fonction de hachage, va créer un numéro (empreinte).




On crée des enregistrements indexés par ledit numéro (empreinte), chaque enregistrement contenant l'adresse.

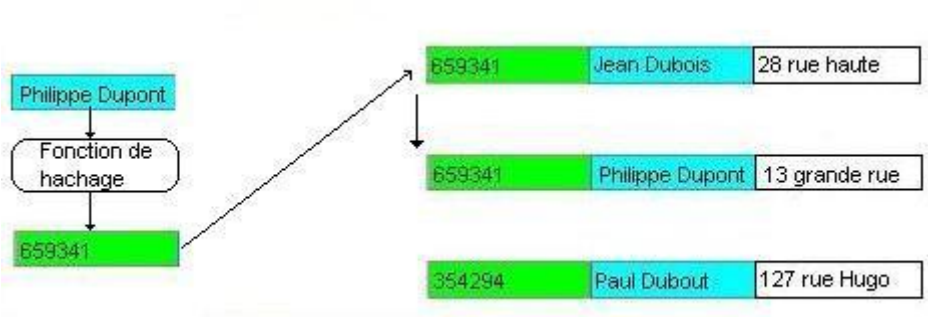



Si maintenant on cherche un nom, on calcule son empreinte, ce qui nous donne l'index de l'enregistrement que l'on trouve rapidement.



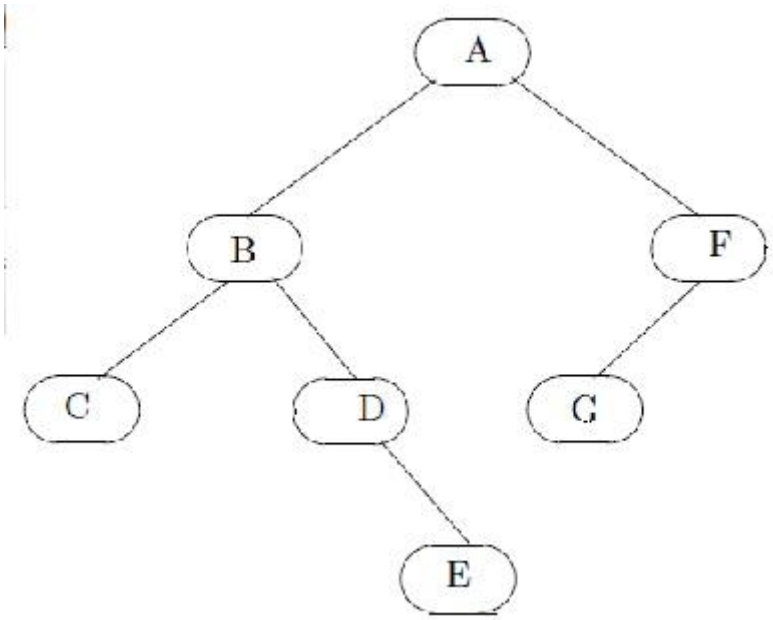

Si la fonction de hachage est uniforme, cela veut dire que pour des entrées différentes, il n'y a jamais la même empreinte. Ce qui est la plupart du temps impossible.

Deux noms peuvent donc donner la même empreinte parfois (on parle de collision). Dans ce cas, on range les enregistrements ayant la même empreinte les uns à la suite des autres (sous forme de liste chainée). Si le premier enregistrement n'est pas le bon, on regarde le suivant.




## V-B-3-t - Arbre

Structure de données. Désigne une forme de diagramme qui modélise une hiérarchie.




Un arbre est constitué d'un ensemble de **nœuds** et de **branches** reliant les nœuds. On peut parler de feuilles pour les éléments, les objets qui sont en bout de branches.
Le premier nœud d'un arbre (celui sans branche entrante) se nomme **la racine**.
Un arbre est unidirectionnel et sans boucle.

Un exemple de données en 'arbre' : sur un disque dur, les répertoires, sous-répertoires et fichiers ont une structure en arbre.

## V-B-3-u - Erreur d'exécution : Notion de 'Sécurisation' du code

Erreur d'exécution

Si le code exécute Z=X/Y avec Y=0 (pour des entiers) , comme la division par zéro est impossible, le logiciel s'arrête.

Il s'agit d'une 'Erreur d'exécution' (dépassement de capacité).

En VB on dit qu'une exception est levée.

Il appartient au programmeur, une fois l'algorithme écrit, de le sécuriser : des instructions doivent protéger certaines parties du code afin d'éviter d'effectuer des opérations incohérentes.

```
Si Y <> 0 Alors
    Z= X/Y
Fin Si
```

Si vous testez cela en VB, bien utiliser des Integers, avec des Single le comportement est différent.

## V-B-3-v - Récursivité

Une procédure est récursive si elle peut s'appeler elle-même.

```
Sub Calcul()

'...
```

```
    Calcul()

'…

End Sub
```

Pourquoi utiliser la récursivité ?
Une procédure récursive découpe le problème en morceaux plus petits et s'appelle elle-même pour résoudre chacun des plus petits morceaux, elle résout une petite partie du problème elle-même.

## V-B-3-w - Flag et variables d'état

Un Flag (ou drapeau) est une variable utilisée pour enregistrer un état, la valeur de cet état servant à déclencher ou non des actions. C'est une manière de retenir qu'un événement s'est produit.

Si le drapeau est abaissé, les voitures roulent…

Exemple : utiliser un Flag pour sortir d'une boucle.

On utilise flagSortir.

```
flagSortir=Faux

Tant que  flagSortir =Faux

      Si on doit sortir de la boucle, on met la valeur de flagSortir à Vrai

Boucler
```

En VB :

```
flagSortir=Faux

Do while flagSortir =Vrai

     ' Si on doit sortir de la boucle, on met la valeur de flagSortir à Vrai

Loop
```

Tant que flagSortir =Faux la boucle tourne.

On peut généraliser cette notion en parlant de variable d'état.

Une variable d'état sert à décrire l'état du programme.

Exemple

fileIsOpen est une variable indiquant si un fichier est ouvert ou fermé.

## V-B-3-x - Compilation, interprétation

Ici on n'est pas à proprement parler dans l'algorithmie.

Le texte que vous écrivez pour construire un programme est le code source.

Tout langage doit obligatoirement être traduit en langage machine (le langage du processeur) pour que ce programme soit exécutable.

Il existe deux stratégies de traduction :

- le programme traduit les instructions au fur et à mesure qu'elles se présentent (à la volée) au cours de l'exécution. Cela s'appelle l'interprétation ;

- le programme commence par traduire l'ensemble du programme (programme source) en langage machine, constituant ainsi un deuxième programme (un deuxième fichier) distinct physiquement et logiquement du premier, c'est le fichier exécutable. Cela s'appelle la compilation. Ensuite, pour exécuter le programme, on exécute l'exécutable, ce second programme.

Les premiers langages Basic étaient interprétés. Un langage interprété était plus maniable : on exécutait directement son code au fur et à mesure qu'on le tapait, sans passer à chaque fois par l'étape supplémentaire de la compilation. Un programme compilé s'exécute beaucoup plus rapidement qu'un programme interprété : le gain est couramment d'un facteur 10, voire 20 ou plus.

Le VB.Net est un langage compilé.

Le code source est dans des fichiers '.vb' et l'exécutable est un '.exe'. On verra que dans l'environnement de développement vb présente les avantages d'un langage interprété (exécution pas à pas, modification du source en cours de débogage…) On peut aussi créer un exécutable autonome.

Les choses sont plus complexes, car en vb, entre le source et l'exécutable il y a un code 'intermédiaire'.

## V-B-4 - Grandes stratégies

Pour trouver des réponses à un problème, on va choisir une grande stratégie et utiliser des structures élémentaires pour l'exécuter.

**Algorithme direct ou explicite**
Ici l'algorithme utilise la seule voie possible, évidente mathématiquement.
Exemple : pour calculer les racines d'une équation du second degré, l'algorithme calcule le déterminant puis en fonction de sa valeur (plusieurs cas), il calcule les racines.
Il est possible de découper un problème complexe en plusieurs problèmes plus simples.

**Algorithme glouton**
Un algorithme glouton suit le principe de faire, étape par étape, un choix optimum local, dans l'espoir d'obtenir un résultat optimum global.
Par exemple, dans le problème du rendu de monnaie (rendre une somme avec le moins possible de pièces), l'algorithme consiste à répéter le choix de la pièce de plus grande valeur qui ne dépasse pas la somme restante; c'est un algorithme glouton.
Nous faisons beaucoup d'algorithmes gloutons !

**Diviser pour régner**
Diviser pour régner est une technique algorithmique consistant à diviser un problème de grande taille en plusieurs sous-problèmes analogues.
Les algorithmes Diviser pour régner appliquent deux stratégies principales. La première est la récursivité sur les données: on sépare les données en deux parties, puis on résout les sous-problèmes (par la même fonction), pour enfin combiner les résultats. La seconde stratégie, la récursivité sur le résultat, consiste elle à effectuer un prétraitement pour bien découper les données, puis à résoudre les sous-problèmes, pour que les sous-résultats se combinent d'eux-mêmes à la fin.
Exemple : les algorithmes de tri (trier un tableau par exemple).

**Recherche exhaustive**
Cette méthode utilisant l'énorme puissance de calcul des ordinateurs consiste à regarder tous les cas possibles.

Exemple : rechercher une clé pour pénétrer dans un site (c'est mal !!): on teste à l'aide d'une boucle toutes les clés possibles, on parle de 'force brute'.

**Algorithme aléatoire, approches successives**
Certains algorithmes utilisent des recherches aléatoires, ou par approches successives donnant de meilleurs résultats que des recherches directes ou explicites. Exemple : ceux dits de Monte-Carlo et de Las Vegas.

**Les heuristiques**
Pour certains problèmes, les algorithmes ont une complexité beaucoup trop grande pour obtenir un résultat en temps raisonnable. On recherche donc une solution la plus proche possible d'une solution optimale en procédant par essais successifs. Certains choix stratégiques doivent être faits. Ces choix, généralement très dépendants du problème traité, constituent ce qu'on appelle une heuristique. Les programmes de jeu d'échecs, de jeu de go fonctionnent ainsi.

## V-B-5 - Quelques algorithmes

Les algorithmes les plus étudiés sont :
- Algorithmes simples :
…Inversion de variables,
…Recherche d'une valeur dans un tableau,
…Tri,
…Problème du voyageur de commerce ;
- Algorithmes utilisant la récursivité ;
- Algorithmes mathématiques ;
- Algorithmes informatiques complexes :
…Compression,
…Cryptage,
…Graphisme,
….

## V-B-5-a - Recherche dans un tableau

Soit un tableau A() de 4 éléments :

| |
|---|
| 3 |
| 12 |
| 4 |
| 0 |

Je veux parcourir le tableau pour savoir s'il contient le chiffre '4'.

Il faut faire une itération afin de balayer le tableau : la variable dite de boucle ( I ) va prendre successivement les valeurs: 0 ,1 ,2 ,3. (Attention : dans un tableau de 4 éléments, l'index des éléments est 0,1,2,3)

```
Pour I variant de 0 à 3 Répéter

      …

FinRépéter
```

Dans la boucle il faut tester si la valeur de l'élément du tableau est bien la valeur cherchée.

```
Pour I variant de 0 à 3 Répéter

      Si A(I)= 4 Alors…

FinRépéter
```

Si on a trouvé la bonne valeur, on met un flag (drapeau) à Vrai.

```
flagTrouvé<-Faux

Pour I variant de 0 à 3 Répéter

      Si A(I)= 4 Alors flagTrouvé<-Vrai

FinRépéter
```

Ainsi si après la boucle flagTrouvé= Vrai, cela veut dire que le chiffre 4 est dans le tableau.

En VB :

```
flagTrouve=False

For I=0 To 4

    If A(I)=4 Then flagTrouve=True

Next I
```

## V-B-5-b - Tri de tableau

Pour trier un tableau de chaines de caractères (des prénoms par exemple), il faut comparer 2 chaines contiguës, si la première est supérieure (c'est-à-dire après l'autre sur le plan alphabétique: "Bruno" est supérieur à "Alice"))on inverse les 2 chaines, sinon on n'inverse pas. Puis on recommence sur 2 autres chaines en balayant le tableau jusqu'à ce qu'il soit trié.

Tableau non trié :

| Bruno |
|---|
| Alice |
| Agathe |

On compare les lignes 1 et 2, on inverse

| Alice |
|---|
| Bruno |
| Agathe |

On compare les lignes 2 et 3, on inverse

| Alice |
|---|
| Agathe |
| Bruno |

Le tableau n'étant pas encore trié, on recommence :

On compare les lignes 1 et 2, on inverse

| Alice |
|---|
| Agathe |
| Bruno |

On compare les lignes 2 et 3, on n'inverse pas.

Le tableau est trié.

Tout l'art des routines de tri est de faire le moins de comparaisons possible pour trier le plus vite possible.

On a utilisé ici le Bubble Sort (ou tri à bulle), on le nomme ainsi, car l'élément plus grand remonte progressivement au fur et à mesure jusqu'au début du tableau comme une bulle. ("Agathe" est passé de la troisième à la seconde puis à la première position).

Une boucle externe allant de 1 à la fin du tableau balaye le tableau N fois (La boucle varie de 0 à N-1), une seconde boucle interne balaye aussi le tableau et compare 2 éléments contigus (les éléments d'index j et j+1)et les inverse si nécessaire. La boucle interne fait remonter 1 élément vers le début du tableau, la boucle externe le fait N fois pour remonter tous les éléments.

```
Pour i allant de 0 à N-1
    Pour j allant de 0 à N-1
        Si T(j)>T(j+1) Alors
            Temp<-T(j)
            T(j)<-T(j+1)
            T(j+1)<-Temp
        Fin Si
    Fin Pour
 Fin Pour
```

En Visual Basic :

```
Dim i, j , N  As Integer    'Variable de boucle i, j ; N= nombre d'éléments-1
Dim Temp  As String
N=4  'tableau de 5 éléments.
Dim T(N) As String  'élément de 0 à 4
For i=0 To N-1
    For j=0 To N-1
        If T(j)>T(j+1) then
            Temp=T(j): T(j)=T(j+1):T(j+1)=Temp
        End if
    Next j
 Next i
```

*Remarque : pour inverser le contenu de 2 variables, on doit écrire*

Temp=T(j): T(j)=T(j+1): T(j+1)=Temp (L'instruction qui faisait cela en VB6 et qui se nommait Swap n'existe plus)

Cet algorithme de tri peut être optimisé, on verra cela plus loin.

Les algorithmes s'occupent aussi de décrire la manière de rechercher des données dans des tableaux, de compresser des données… Nous verrons cela au fur et à mesure.

## V-B-6 - Lexique anglais=>français

If = Si.

Then= Alors

Step=Pas

Do (To)= faire

While= tant que

Until= Jusqu'à ce que.

Loop= boucle

## V-C - L'affectation

C'est l'instruction la plus utilisée en programmation.

On peut aussi utiliser le terme 'Assignation' à la place de l'affectation.

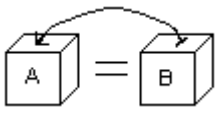
'Variable = Expression' est une affectation, **c'est le signe '=' qui indique l'affectation.**

L'affectation transfère le résultat de l'expression située à droite du signe 'égal' dans la variable (ou la propriété) à gauche du signe égal.

Exemple

A=B est une affectation (ou assignation).

A=B affecte la valeur de la variable B à la variable A, la valeur de B est mise dans A.




Si

A=0

B=12

A=B entraine que A=12 (B n'est pas modifié).

**Si nécessaire l'expression à droite du signe = est évaluée, calculée avant d'être affectée à la variable de gauche.**

Si

A=0

B=12

A=B+2 entraine que A=14

L'affectation permet donc de faire des calculs :

Si nombrdHeure=100 et tauxHoraire=8

paye= nombredHeure * tauxhoraire

paye prend la valeur 800 (notez que '*' , l'étoile veut dire : multiplication.)



*Attention dans le cas de l'affectation "=" ne veut donc pas dire 'égal'.*

A=A+1 est possible

Si A=1

A=A+1 entraine que A=2

On verra qu'il existe des variables numériques ('Integer' 'Single') et alphanumériques (chaine de caractères ou 'String'), l'affectation peut être utilisée sur tous les types de variables.

Le second membre de l'affectation peut contenir des constantes, des variables, des calculs dans le cas de variables numériques.

A=B+2+C+D

On ne peut pas affecter une variable d'un type à une variable d'un autre type :

si A est numérique et B est alphanumérique (chaine de caractères) A=B n'est pas accepté.

Écriture compacte :

A=A+1 peut s'écrire de manière plus compacte : A += 1

A=A*2 peut s'écrire de manière plus compacte : A *= 2

A=A&"Lulu" pour une variable chaine de caractères peut s'écrire de manière plus compacte : A &= "Lulu"

L'affection marche pour les objets, leurs propriétés…

Bouton1.BackColor= Bouton2.BackColor

Signifie que l'on donne au Bouton1 la même couleur de fond que celle du bouton2: on affecte la valeur BackColor du Bouton2 au Bouton1.



*Attention le signe '=' signifie par contre 'égal' quand il s'agit d'évaluer une condition, par exemple dans une instruction If Then (Si Alors)*

If A=B then 'signifie: Si A égal B alors…

**Permutation de variables**

C'est un petit exercice.

J'ai 2 variables A et B contenant chacune une valeur.

Je voudrais mettre dans A ce qui est dans B et dans B ce qui est dans A.

Si je fais

A=B

B=A

Les 2 variables prennent la valeur de B !!

Comment faire pour permuter?

Et bien il faut utiliser une variable intermédiaire C qui servira temporairement à conserver le contenu de la variable A :

C=A

A=B

B=C

| | A | B | C |
|---|---|---|---|
| Départ | 1 | 2 | 0 |
| C=A | 1 | 2 | 1 |
| A=B | 2 | 2 | 1 |
| B=C | 2 | 1 | 1 |

Voilà, on a bien permuté.

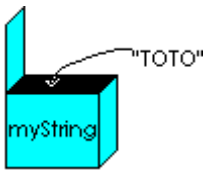
## V-D - Les variables : généralités

C'est le même sens qu'en mathématiques.

Une variable sert à stocker un nombre, du texte (chaine de caractères), une date, un objet…

Une variable a un **nom** et un **type** qui indiquent ce que l'on peut y mettre (Elle a aussi une 'portée' : où et quand sera-t-elle visible et utilisable ?).

Si myString est une variable de type chaine de caractères (ou String): je peux y mettre une chaine de caractères ("TOTO" par exemple)




myString="TOTO"

Si age est une autre variable de type numérique, je peux y stocker un nombre (45,2 par exemple).

age=45,2

## V-D-1 - Nom des variables

On peut utiliser dans les noms de variable des majuscules ou des minuscules, mais pour VB il n'y a pas de différence.

Exemple

On ne peut pas déclarer une variable VB et une variable vb.

Si on déclare une variable nommée 'VB' et si ultérieurement on tape 'vb', le logiciel le transforme automatiquement en 'VB'.

- On peut mettre des chiffres et des lettres dans les noms de variable, mais pas de chiffres en premier caractère.
  2A n'est pas un nom valide
  Nom2 l'est.
- Certains caractères de ponctuation ('…) ne peuvent pas être utilisés, d'autres "_" et "-" sont permis :
  nom_Utilisateur est valide.
  L'espace n'est pas permis.
- Bien sur, les mots clé de VB ne peuvent pas être utilisés : On ne peut pas nommer une variable Form ou BackColor

Il est conseillé de donner des noms explicites qui rappellent ce que contient la variable :

nom_Utilisateur est explicite, pas NU.

Parfois on indique en début de nom, par une lettre, le type de variable ou sa portée (la zone de code où la variable existe).

s_Nom 'Le s indique qu'il s'agit d'une variable String (chaine de caractères)

iIndex 'Le i indique qu'il s'agit d'une variable Integer (Entier)

gNomUtilisateur 'g indique que la variable est globale

Il est possible de 'forcer' le type de la variable en ajoutant un caractère spécial à la fin du nom de la variable.

Dim c$ = "aa" ' $ force c à être une variable String.

De même % force les Integer, & les long…

Cette notation est moins utilisée et non recommandée.

Voir en annexe, en bas de page des recommandations supplémentaires.

Nommage

*On conseille, quand le nom d'une variable est composé de plusieurs mots, de mettre la première lettre de chaque mot en majuscule, sauf pour le premier mot.*

Exemple

nom_Utilisateur

## V-D-2 - Déclaration, initialisation

Avant d'utiliser une variable, il faut la déclarer, la créer, pour cela on utilise l'instruction Dim :

```
Dim a As Integer    'Déclare une variable nommée 'a' de type Entier
```

Avant la ligne, a n'existait pas, après le DIM, a existe et contient 0.

L'instruction **Dim** crée la variable et lui alloue un espace de stockage.

Il faut aussi parfois l'**initialiser**, c'est-à-dire lui donner une valeur de départ : a=3

On peut déclarer et initialiser en même temps :

```
Dim a As Integer =3
```

Rien n'empêche d'utiliser une expression, une fonction pour initialiser.

```
Dim a As Integer = CalculMaVariable(4,3)
```

Ici pour initialiser la variable, on appelle une fonction CalculMaVariable qui retourne une valeur pour a.

*Il est toujours préférable d'initialiser rapidement une variable.*

On peut déclarer plusieurs variables en même temps; il faut les séparer par des virgules :

```
Dim a, b, c As Integer        'a, b et c sont maintenant des variables de type integer.
```

Bien sûr, on ne peut pas déclarer plusieurs fois un même nom de variable :

```
Dim a As Integer
Dim a As String' <= non accepté
```

On peut affecter à une variable une valeur de même type que le type de la variable :

```
Dim a As Integer
 a=2  'OK
 a= "Philippe"  'Non, car on ne peut pas mettre une chaine de caractères dans une variable de
 type Integer
```

## V-D-3 - En pratique : Exemple

Les variables après avoir été déclarées, vont servir à stocker des données, à effectuer des calculs, on peut afficher ensuite leur valeur.

Exemple simpliste d'utilisation de variables :

```
    Dim varA As Integer         'Création d'une variable varA

    Dim varB As Integer         'Création d'une variable varB

    Dim total As Integer        'Création d'une variable total


    varA=3                      'Mettre '3' dans a


    varB=2                      'Mettre '2' dans b


    total=varA + varB           'Mettre dans la variable 'total' la valeur de varA et de varB


    LabelTotal.Text= total.ToString    'Afficher le total dans un label
```

Un label est une zone permettant d'afficher du texte, pour afficher dans un label il faut transformer la variable total qui est un entier en chaine de caractères (à l'aide de ToString) puis mettre la string dans le texte du label.

Noter bien la différence entre :

```
Dim total As Integer
```

total= 123 'Total est une variable numérique (Integer ou Entier) contenant le nombre 123

et

```
Dim total2 As String
```

total2= "123" 'Total2 est une variable string contenant la chaine de caractères "123" c'est-à-dire les caractères "1" , "2" et "3"

*On peut afficher les chaines de caractères (ou String), pas les variables numériques.*

*On fait des calculs avec les variables numériques.*

Il faudra donc parfois convertir le contenu d'une variable d'un type dans un autre type ( convertir une variable numérique en String pour l'afficher par exemple ou convertir une variable String en numérique pour faire un calcul). On apprendra cela plus loin.

L'exemple simpliste juste au-dessus le montre: il faut convertir total qui est un entier en string pour l'afficher.

**Concernant les variables numériques**

Les variables numériques peuvent être **signées** (accepter les valeurs négatives ou positives) ou **non signées** (Comme le Type 'Byte' qui ne contient que des valeurs positives). Les variables numériques peuvent contenir des **entiers**(comme les Integer) ou des **réels** (comme les Single). Pour les réels dans le code, le séparateur est le point:

## V-D-4 - Les différents types de variables

En Visual Basic :

| Nom : | Contient : |
|---|---|
| Boolean | Contient une valeur booléenne (logique): True ou False. |
| Byte | Contient les nombres entiers de 0 à 255 (sans signe) |
| Short | Entier signé sur 16 bits (-32768 à 32768) |
| Integer | Entier signé sur 32 bits (-2147483648 à 2147483647) |
| Long | Entier signé sur 64 bits (-9223372036854775808 à 9223372036854775807) |
| BigInteger | Entier signé très grand (sans limite supérieure ou inférieure) (VB2010) |
| Single | Nombre réel en virgule flottante (-1,401298 *10^-45 à 1,401298 10^45) |
| Double | Nombre réel en virgule flottante double précision. (…puissance 324) |
| Decimal | Nombre réel en virgule fixe grande précision sur 16 octets. |
| Char | 1 caractère alphanumérique |
| String | chaine de caractères de longueur variable (jusqu'a 2 milliards de caractères) |
| DateTime | Date plus heure |
| Object | Peut contenir tous les types de variables, mais aussi des contrôles, des fenêtres… |
| Structure | Ensemble de différentes variables définies par l'utilisateur. |
| | Depuis la version 2005 il y a aussi les Unsigned ( non signé: pas de valeur négative): |
| UInteger | Entier codé sur 32 bits pouvant prendre les valeurs 0 à 4 294 967 295. |
| ULong | Entier codé sur 64 bits :0 à 18 446 744 073 709 551 615 |
| UShort et | Entier sur 16 bits 0 à 65 535. |
| SByte | Byte, mais signé. Codé sur 1 octet, valeur de -128 à 127 |
| Complex | Nombre complexe (en VB2010) |

## V-D-5 - Les Boolean

Contient une valeur booléenne : True ou False (Vrai ou Faux pour les sous doués en anglais!).

Exemple :

```
Dim myBoolean As Boolean

myBoolean = True
```

Après déclaration un Boolean a la valeur False.

## V-D-6 - Variable entière

| | |
|---|---|
| Byte | Contient les nombres entiers de 0 à 255 (sans signe). |
| Short | Entier sur 16 bits (-32768 à 32768) |
| Integer | Entier sur 32 bits (-2147483648 à 2147483647) |
| Long | Entier sur 64 bits (-9223372036854775808 à 9223372036854775807) |
| BigInteger | Entier signé très grand (sans limite supérieure ou inférieure) (VB2010) |

Pour une variable entière il n'y a pas de possibilité de virgule!! attention, une division de 2 entiers donne un entier.

Pour le mathématicien, les Integer correspondent aux nombres entiers naturels : Entiers positif ou négatif et 0 (…-3 -2 -1 0 1 2 3…), mais avec une limite.



*Attention, les variables numériques en informatique ne peuvent pas contenir de nombre infiniment grand ( sauf les BigInteger): Il y a une limite maximum : un Integer est par exemple codé sur 32 bits ce qui fait qu'il peut varier de -2147483648 à 2147483647. Si on utilise une valeur trop grande, une erreur se produit. Les entiers peuvent être positifs ou négatifs (Sauf les Bytes et les 'Unsigned': UInteger, ULong, UShort).*

Plus on augmente le type (Long au lieu de Integer) plus on peut y mettre des grands nombres. Mais cela prend aussi plus de place et le traitement des opérations est plus long.

Les processeurs travaillant sur '32 bits', utilisez plutôt les Integer (qui sont codés sur 32 bits aussi), c'est plus rapide, que les short.

On utilise largement les 'Integer' comme variable de boucle, Flag, là ou il y a besoin d'entier…(Les calculs sur les réels en virgule flottante sont beaucoup plus lent.)

Exemple:

```
Dim i As Integer

i=12
```

Après déclaration une variable numérique contient 0.

Le type de données Byte est utilisé pour contenir des données binaires (octet codant de 0 à 255) non signé.

## V-D-7 - Variable réelle

Un réel peut avoir une partie fractionnaire: 1,454 est un réel.

Pour le mathématicien, les Single, Double… correspondent aux nombres réels ou fractionnaires: mais avec des limites ( sur la précision et le fait qu'ils ne sont pas infinis).

Single, Double, Decimal.

Single Nombre réel en virgule flottante (-1,401298 *10^-45 à 1,401298 10^45)

Double Nombre réel en virgule flottante double précision. (-1,79769313486231570E+308 et -4,94065645841246544E-324 pour les valeurs négatives et entre 4,94065645841246544E-324 et 1,79769313486231570E+308 pour les valeurs positives)

Decimal Nombre réel en virgule fixe grande précision sur 16 octets: Avec une échelle 0 (aucune décimale), la plus grande valeur possible correspond à +/-79 228 162 514 264 337 593 543 950 335. Avec 28 décimales, la plus grande valeur correspond à +/-7,9228162514264337593543950335

**Les variables en virgule flottante ou notation scientifique**

(Single, Double)

La variable peut être positive ou négative.

Le 'Double' est, bien sûr, plus précis et peut atteindre des nombres plus grands que le 'Single'.

Le 'Single' comporte 7 chiffres significatifs maximum.

Le 'Double' comporte 18 chiffres significatifs maximum.

Le nombre est codé en interne sous forme scientifique, exemple:1,234568E+008.

Mais en pratique, on travaille et on les affiche de manière habituelle, en notation normale avec un point comme séparateur décimal :

```
Dim poids As Single

poids=45.45
```

Format scientifique, mantisse et exposant

Voici 3 nombres :

14500000
0,145
0,0000145

**Ils comportent tous les 3, deux informations :**

- le nombre entier 145
- la localisation du premier chiffre par rapport à la virgule

8
-1
-5 dans nos exemples.

**Donc un réel peut être stocké sous la forme d'un couple :**

- partie entière ;
- localisation de la virgule.

Il est codé en interne avec une mantisse (la partie entière) et un exposant (position de la virgule), sous la forme mmmEeee, dans laquelle mmm correspond à la mantisse (chiffres significatifs: partie entière) et eee à l'exposant (puissance de 10).

En fait, en notation scientifique (en codage interne) un chiffre précède toujours la virgule: 1,234568E+008.



*Attention, les variables numériques réelles ne peuvent pas contenir de nombre infiniment grand: Il y a une limite maximum comme pour les entiers. La valeur positive la plus élevée d'un type de données Single est 3,4028235E+38 et celle d'un type de données Double est 1,79769313486231570E+308. Si on dépasse cette valeur VB le signale en déclenchant une erreur.*

Quand on travaille avec des nombres ayant beaucoup de chiffres significatifs, il peut y avoir des erreurs d'arrondi. Le type 'Single' comporte par exemple une mantisse de 7 chiffres significatifs seulement. Si on utilise des nombres (même petit: avec un exposant négatif par exemple)avec 8 chiffres significatifs il peut y avoir des erreurs d'arrondi.

**Le type en Virgule fixe.**

Le type en Virgule fixe (Decimal) prend en charge jusqu'à 29 chiffres significatifs et peut représenter des valeurs jusqu'à 7,9228 x 10^28. Ce type de données est particulièrement adapté aux calculs (par exemple financiers) qui exigent un grand nombre de chiffres, mais qui ne peuvent pas tolérer les erreurs d'arrondi.
Il est codé sur 128 bits sous forme d'un entier et une puissance de 10.

Les Calculs en Decimal sont 10 fois plus lents que les calculs en Single, mais il n'y a pas d'erreur d'arrondi avec les décimals.

*Pour les calculs financiers ont utilisera les 'Decimal'.*

*Pour les petits calculs du genre résultats d'examen biologique, on utilisera les 'Single' ou les 'Double' qui sont les plus rapides.*

Pour les variables de boucle, les index, on utilise habituellement des Integers.

## V-D-8 - String, Char

Le type 'String' peut contenir une 'chaine de caractères' (alphanumérique) comme du texte. La longueur de la chaine n'est pas fixe et une String peut avoir un nombre de caractères allant de 0 jusqu'à environ 2 milliards de caractères.

Les chaines de longueur fixe n'existent pas (plus).

Le Type 'Char' contient un seul caractère. On utilise souvent des tableaux de 'Char'.

Pour information Char et String contiennent en interne **le code des caractères** au format Unicode (dans la variable,chaque caractère est codé sur 2 octets) et pas de l'ASCII ou de l'ANSI…(ancien codage où chaque caractère était codé sur un octet).

Les premiers caractères ont le même code Unicode et Ascii.

Exemple :

| Caractère | Code |
|---|---|
| "a" | 65 |
| "b" | 66 |
| " " | 32 |

Il y a aussi des caractères non affichables :

| | | |
|---|---|---|
| RC | 13 | retour chariot |
| LF | 10 | Line Feed |
| | 9 | Tabulation |

Pour passer à la ligne, on utilise les codes 13 puis 10. Il y a une constante toute faite pour cela: **ControlChars.CrLf**.

## V-D-9 - Place occupée en mémoire

Les types de variables ont un nom en VisualBasic et un nom dans le Framework.

Exemple

Integer et System.Int32 sont équivalents pour designer le type 'entier', Integer est le type VB, System.Int32 est le type 'NET' correspondant. On peut utiliser l'un ou l'autre.

Exemple de place occupée par une variable (et le nom de sa Classe dans NET).

| Type VB | Place occupée | Type NET correspondant |
|---|---|---|
| Boolean | 2 octets | System.Boolean |
| Byte | 1 octet | System.Byte |
| Short | 2 octets | System.Int16 |
| Integer | 4 octets | System.Int32 |
| Long | 8 octets | System.Int64 |
| Single | 4 octets | System.Single |
| Double | 8 octets | System.Double |
| Decimal | 16 octets | System.Decimal |
| Date | 8 octets | System.DateTime |
| Char | 2 octets | System.Char |
| Objet | 4 octets | System.Objet |
| String | dépend de la chaine | System.String |

La méthode GetType permet de savoir de quel type, de quelle Classe est une variable.

```
Dim x As String ="a"

MessageBox.Show(x.GetType.ToString)     'Affiche: Systeme.String
```

Prend le type de x, le transforme en String, l'affiche dans une MessageBox (Noter qu'il faut initialiser x avec une valeur avant de faire GetType).

## V-D-10 - Type primitif, littéral

Mise à part Objet, Structure, Class tous les autres types sont dit 'Primitif'( Byte, Boolean, Short, Integer, Long, Single, Double, Decimal, Date, Char, String).

- Tous les types primitifs permettent la création de valeurs par l'écriture de littéraux. Par exemple, i=123 ou i=123I (le I force 123 a être entier) est un littéral de type Integer.
- Il est possible de déclarer des constantes des types primitifs.
- Lorsqu'une expression est constituée de constantes de type primitif, le compilateur évalue l'expression au moment de la compilation. C'est plus rapide.

**Un littéral**: c'est une donnée utilisée directement; une valeur numérique ou en toutes lettres par opposition à une variable.

```
Dim i As Integer
i=4   '4 est un littéral, c'est ici un integer
```

On voit que le littéral est un Integer en passant la souris dessus.

```
Dim i As Integer
i=100000000  'valeur non acceptée, car trop grande pour un littéral integer
Dim j As Long
j=100000000  'accepté, car le littéral est un Long
```

Attention, si je tape :

```
i=1.4
```

1.4 est un Double, il sera converti en Integer pour être affecté à i est on aura dans i la valeur 1.
Le signe entre la partie entière et fractionnaire est le '.' dans un littéral et cela force le littéral à être un Double.

On peut forcer le type d'un littéral en ajoutant une lettre :

```
i= 42L   'le 'L' indique que 42 est un Long
i= 42I   'le 'I' indique que 42 est un Integer
i= 42D   'le 'D' indique que 42 est un Decimal
i= 42S   'le 'S' indique que 42 est un Single
C = "A"c  'le 'c' force "A" à être une Char et non une String
```

## V-D-11 - Nullable

### Type Nullable

Les types Par Valeur peuvent être étendus afin d'accepter une valeur normale habituelle ou une valeur Null (Nothing en VB). On peut déclarer un type Nullable de 3 manières :

```
Dim MyInteger As Nullable (Of Integer)
```

Mais aussi :

```
Dim MyInteger? As Integer

Dim MyInteger As Integer?
```

C'est le '?' qui force la variable Nullable.

Sur la seconde déclaration, la variable est MyInteger

Autre exemple :

```
Dim MyBol As Boolean?
```

MyBol pourra prendre la valeur True, False et Nothing.

Cela a de l'intérêt quand on travaille avec les bases de données qui ont des champs qui contiennent un Null et avec Linq.

La propriété HasValue permet de voir si la variable a une valeur autre que Nothing (Valeur retrouvée dans Value).

```
Dim a? As Integer
        If a.HasValue = True Then
            MsgBox(a.Value.ToString)
        End If
```

## V-D-12 - Choix des noms de variables

- La plupart des noms sont une concaténation de plusieurs mots, utilisez des minuscules et des majuscules pour en faciliter la lecture.
- Pour distinguer les variables et les routines (procédures), utilisez la casse Pascal (CalculTotal) pour les noms de routine (la première lettre de chaque mot est une majuscule).
- Pour les variables,la première lettre des mots est une majuscule, sauf pour le premier mot (documentFormatType).
- Le nom des variables booléennes doit contenir Is qui implique les valeurs Yes/No ou True/False, Exemple fileIsFound.
- Même pour une variable à courte durée de vie qui peut apparaitre uniquement dans quelques lignes de code, utilisez un nom significatif. Utilisez des noms courts d'une seule lettre, par exemple i ou j, pour les index de petite boucle uniquement.
- N'utilisez pas des nombres ou des chaines littérales telles que For i = 1 To 7. Utilisez plutôt des constantes nommées, par exemple For i = 1 To Nombre_jour_dans_semaine, pour simplifier la maintenance et la compréhension.
- Utilisez des paires complémentaires dans les noms de variables telles que min/max, begin/end et open/close ou des expressions min max si nécessaire en fin de nom.

# V-E - Variables 'String' et 'Char'

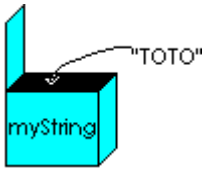
## V-E-1 - Variables 'String'

Il faut déclarer une variable avant de l'utiliser, pour cela on utilise l'instruction Dim.

```
Dim MyString As String
```

Déclare une variable nommée MyString et qui peut contenir une chaine de caractères.

Cette variable peut être utilisée pour contenir une chaine de caractères.




```
MyString= "TOTO"
```

'On met la chaine de caractères "TOTO" dans la variable MyString.

On peut afficher le contenu de la chaine dans un label (zone présente dans une fenêtre et où on peut afficher du texte) par exemple :

```
        Label.text = MyString
```

Cela affiche 'TOTO' dans le label.

Remarquons que pour définir une chaine de caractères il faut utiliser des "" : Ce qui est entre" et " est la chaine de caractères. On parle ici de chaine littérale: une représentation textuelle d'une valeur particulière.



*Après avoir été créée, une String contient 'Nothing' c'est-à-dire rien (même pas une chaine vide: ""); il faudra l'initialiser pour qu'elle contienne quelque chose.*

```
Dim str As String     'str contient Nothing
'Testons si str contient Nothing
If IsNothing(str) then Console.Write ( "contient Nothing")
```

(pas le texte "Nothing"!! mais la valeur Nothing qui signifie qu'elle ne pointe sur rien.

```
str= ""               'str contient "" : chaine vide de longueur 0

str= "TOTO"           'str contient "TOTO"
```

Notez bien l'importance des guillemets :

A est la variable A

"A" est une chaine de caractères contenant le caractère "A"

Exemple :

```
Dim A As String= "Visual"

Dim B As String= "Basic"

Label.text = "A+B"  affiche bêtement la chaine  "A+B"

Label.text =  A+B    affiche "VisualBasic" 'on affiche les variables.
```

Notez enfin que " ", l'espace est un caractère à part entière.

Si je veux inclure un caractère " dans la chaine, il faut le doubler pour qu'il ne soit pas considéré comme caractère de fin de chaine :

```
A=" Bonjour ""Monsieur"" "    'Cela affiche :  Bonjour "Monsieur"
```

On peut initialiser la variable en même temps qu'on la déclare.

```
Dim Chaine as string = "Toto"
```

On peut déclarer plusieurs variables d'un même type sur une même ligne.

```
Dim x, y, z As String    'Déclare 3 variables 'String'
```

On utilise GetType pour connaitre le type d'une variable.

```
x.GetType.ToString

y.GetType.ToString

z.GetType.ToString
```

donne

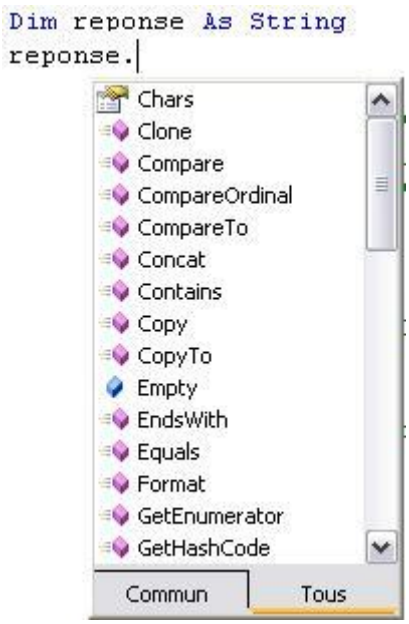
```
System.String

System.String

System.String
```

Ce qui prouve que les 3 variables sont bien des Strings.

## V-E-1-a - La Classe System.String

Le type System.String ou String (chaine de caractères) est une Classe du Framework, qui a des méthodes.

*Pas besoin de connaitre toutes les méthodes, il suffit de déclarer la String par 'Dim réponse As String') de taper réponse puis "." et vous voyez apparaitre toutes les propriétés et méthodes :*




Voyons par exemple la méthode **.ToUpper**

Elle retourne la chaine de caractères en majuscules.

```
        str=str.ToUpper()
```

Si str contenait "abc" il contiendra "ABC"

**.ToLower** transforme par contre la chaine en minuscules.

Quel intérêt ?

Exemple

Je dois comparer 2 String pour savoir si elles sont égales, la première a été saisie par l'utilisateur et je ne sais pas si l'utilisateur a tapé en majuscules ou en minuscules.

Si je compare A = "Vb" et B= "vb" elles sont différentes.

Si je compare A.ToLower et B.ToLower elles sont égales.

**.Trim**

Permet de supprimer des caractères en début et fin de chaine.

```
Dim a As String = "#@Informatique@#"

Dim b As Char() = {"#", "@"}    'b est un tableau de Char contenant les caractères à supprimer.

a=a.Trim(b)  Donne a= "Informatique"
```



*Attention : Bien utiliser Char() qui est un tableau de caractères pour définir les caractères à supprimer.*

(Dim b As String= "#@" est déconseillé, car produisant des résultats curieux.)

Pour enlever les espaces avant et après la chaine (cas le plus fréquent) :

```
s=" Bonjour "

s=s.Trim(" ")    'donne s="Bonjour"
```

Attention avec Option Strict= On, s=s.Trim("") n'est pas accepté, car le paramètre de Trim doit être une Char, la String" "n'est pas transformée (castée) en char. Il faut écrire s=s.Trim(" "c) ou s=s.Trim(CChar("")).

Il existe aussi **TrimStart** et **TrimEnd** pour agir seulement sur le début ou la fin de la chaine.

**Length**

Length : Taille d'une chaine en nombre de caractères.

Afficher la taille de la chaine "VB"

```
Dim s As String= "VB"

MsgBox(s.Length.ToString) 'Affiche 2
```

**Concat**

Concaténation de plusieurs chaines : mise bout à bout :

```
s=String.Concat(a,b)
```

Il est plus rapide de faire : s= a & b

(s= a+b fait la même chose, mais est déconseillé, on réserve '+' pour l'addition de numériques).

Concat est très pratique quand on veut mettre bout à bout tous les éléments d'un tableau :

```
Dim s() As String = {"hello ", "my ", "friend ", "to "}
Dim c As String = String.Concat( s())
```

### Insert

Insère une chaine dans une autre.

```
Dim s As String= "VisualBasic"

s= s.Insert(6,"  " ) 'Donne s= "Visual  Basic"
```

Noter : le premier caractère a la position 0.

### Remove

Enlève des caractères à une certaine position dans une chaine.

```
Dim s As String = "VisualBasic"

s = s.Remove(2, 7) 'Donne s= "Viic"
```

### Replace

Remplace dans une chaine de départ, toutes les occurrences d'une chaine par une autre.

Resultat=ChaineDépart.Replace(chaineARemplacer,chaineQuiRemplace)

```
Dim s As String= "Visual_Basic"

s= s.Replace("_"," " ) 'Donne s= "Visual  Basic"
```

Autre exemple

L'utilisateur a tapé une date, mais avec comme séparateur des ".", comme on le verra plus loin, il est nécessaire d'utiliser plutôt les "/", pour cela on utilise Replace

```
Dim ladate as string= "12.02.1990"

ladate= ladate.Replace(".","/" ) 'Donne ladate= "12/02/1990"
```

### Split

Découpe en plusieurs sous chaines une chaine de départ, cela par rapport à un séparateur.

Exemple

Je récupère dans un fichier une chaine de mots ayant pour séparateur ";", je veux mettre chaque mot dans un tableau.

chaine contenant les mots séparés par ";"

```
Dim s As String= "Philippe;Jean ;Toto"

Dim separateur As Char = ";"

Dim nom() As String
```

```
nom=s.Split(separateur)
```

Donne :

```
        nom(0)= "Philippe"

        nom(1)= "Jean"

        nom(2)= "Toto"
```

*Remarque : quand on déclare le tableau nom(), on ne donne pas le nombre d'éléments, c'est Split qui crée autant d'éléments qu'il faut.*

En Framework 2, on peut utiliser plusieurs séparateurs différents :

```
nom=s.Split( New Char() {" "c, ","c, "."c }) 'ici on a 3 séparateurs: l'espace, la virgule et le
 point.
```

le ci-après chaque séparateur veut dire Char, car les séparateurs sont des caractères.

On peut ajouter 2 paramètres permettant d'indiquer le nombre de lignes maximum et forcer l'élimination des lignes vides.

```
Dim sep() As Char={" "c, ","c, "."c}

Dim nom() As String = S.Split ( sep, 100, StringSplitOptions.RemoveEmptyEntries )
```

Allons encore plus loin: avant et après le séparateur, il peut y avoir des espaces.

Il faut dans ce cas utiliser la méthode Split de la classe Regex :

```
Imports System.Text.RegularExpressions

Dim S As String = "abc ; def ; ghi"

' On crée un Regex

Dim R As New Regex("\s*;\s*")

' décomposition de ligne en champs

Dim Nom As String() = R.Split(S)
```

### .Join

Concatène tous les éléments d'un tableau et peut ajouter des séparateurs.

Si myLines() est un tableau de String, je veux ajouter ces lignes bout à bout en les séparant d'un retour à la ligne.

```
Dim myText As String = String.Join ( ControlChars.CrLf, myLines)
```

### .IndexOf .LastIndexOf

Indique le numéro du caractère, la position (la première occurrence) ou une chaine à chercher est trouvée dans une autre. Recherche en commençant par la fin avec LastIndexOf.

```
Dim a As String= "LDF.EXE"

Dim r As Char()={"."}

a.IndexOf(r)  retourne 3
```



*Se souvenir : le premier caractère est en position 0 en .Net.*

.LastIndexOf retourne la dernière occurrence.

.IndexOfAny .LastIndexOfAny (Framework 2)

Indique le numéro du caractère, la position (la première occurrence) ou une chaine à chercher est trouvée dans une autre avec en plus possibilité d'indiquer la position de départ.

```
Dim a As String= "LDF.EXE"

Dim r As Char()={"."}

a.IndexOfAny(r)  recherche à partir du début de chaine.

a.IndexOfAny(r,2)  recherche à partir du deuxième caractère.
```

Autre exemple: On recherche ici plusieurs caractères (en fait un tableau de Char)

```
Dim str As String ="gfdjzak;,vdqsygeak"

Dim start As Integer =2

Dim at As Integer

Dim count As Integer =5

Dim target As String = "ou" 'chaine à chercher

Dim anyOf As Char() = target.ToCharArray() 'on transforme la chaine en tableau de char

at = str.IndexOfAny(anyOf, start, count)
'on cherche le tableau de Char anyOf dans str à partir de la position start et sur count
 caractères.
```

**.Compare**

Compare 2 chaines :

```
Dim rep As Integer

rep=String.Compare(a,b)
```

Retourne un entier.

```
      -1 si a<b

      0 si a=b

      1 si a>b
```

On peut comparer des sous-chaines et indiquer la sensibilité à la casse (Framework 2) :

```
Dim myStr1 As [String] = "My Uncle Bill"
```

```
Dim myStr2 As [String] = "My uncle bill"
Dim r As Integer = String.Compare(myStr1, 2, myStr2, 2, 10,
 StringComparison.CurrentCultureIgnoreCase)
```

Ici on compare 10 caractères en commençant par le deuxième caractère de chaque chaine en mode insensible à la casse (majuscules=minuscules).
Voir ci-dessous, le chapitre comparaison.

**.Equals**

Permet de comparer 2 chaines. Retourne True si elles sont égales.

```
Dim b As Booleen= String.Equals("aaa", "AAA")
'b=False
```

On peut ajouter un paramètre pour signifier la culture ou indiquer de ne pas tenir compte de la case comme ici :

```
Dim b As Booleen= String.Equals("aaa", "AAA", StringComparison.CurrentCultureIgnoreCase)
'b=True
```

**.Contains**

Permet de savoir si une chaine apparait dans une autre: (Framework 2)

```
Dim trouve As Boolean

trouve = a.Contains( "123")
```

Retourne True ou False

**.Substring**

Extrait une partie d'une chaine.

Le premier paramètre indique la position de départ; le second, le nombre de caractères à extraire.

```
Dim a As String= "Informatique"

MessageBox.show(a.Substring(2,3)) 'Affiche  for
```

Le premier paramètre indique la position du caractère où doit commencer la sous-chaine, en commençant à la position 0. (les caractères sont comptés 0, 1, 2, 3….

Le second paramètre la longueur de la sous-chaine.

Exercice 1: comment obtenir les 4 caractères de droite :

```
Dim a As String= "Informatique"

MessageBox.show(a.Substring(A.Length-4)) 'Affiche  ique
```

Ici on omet le second paramètre,la longueur de la sous-chaine, va jusqu'a la fin de la chaine.

Exercice 2 : comment obtenir les 3 caractères de gauche :

```
Dim a As String= "Informatique"
```

```
MessageBox.show(a.Substring(0, 3)) 'Affiche  inf
```

### .Chars

Une chaine peut être perçue comme un tableau de caractères (instances Char) ; vous pouvez extraire un caractère particulier en faisant référence à l'index de ce caractère par l'intermédiaire de la propriété Chars. Par exemple :

```
Dim maString As String = "ABCDE"
Dim monChar As Char
monChar = maString.Chars(3) ' monChar = "D"
```

On peut créer des chaines avec la Classe String :

```
myString = New String(" ", 15)     'Créer une chaine de 15 espaces
```

### .PadRight

Aligne les caractères de cette chaine à gauche et remplit à droite en ajoutant un caractère Unicode spécifié pour une longueur totale spécifiée.

```
Dim str As String
Dim pad As Char
str = "Nom"
pad = Convert.ToChar(".")
Console.WriteLine(str.PadRight(15, pad)) ' Affiche Nom……….
```

PadLeft fait l'inverse.

### .StartsWith() et EndsWith()

Permettent de tester si une string commence ou se termine par une string, retourne True ou False.

Tester si la String s commence par "abc" et se termine par "xyz" :

```
If s.StartWith ("abc") And s.EndWith ("xyz") Then
```

En VB 2005, on peut ajouter un argument gérant la culture ou la casse.

Voir aussi **String.Format** dans le chapitre : Afficher correctement du texte.

### .IsNull , IsNullOrEmpty() Framework 2

Il est parfois nécessaire de vérifier si une chaine est égale à Nothing ou de longueur égale à 0 (vide).

```
If S Is Nothing AndOr S.length=0 Then
```

Ou

```
If String.IsNullOrEmpty( S) Then
```

À partir de vb 2010 existe aussi String.**IsNullOrWhiteSpace** qui est égal à True si la chaine est Null Empty ou ne contient que des espaces :

```
If String.IsNullOrWhiteSpace(value) Then…
'Est équivalent à (mais plus rapide):
If String.IsNullOrEmpty(value) OrElse value.Trim().Length = 0 Then….
```

# V-E-1-b - Les instructions 'Visual Basic'

CONSEIL: Si vous débutez, laissez de coté ces instructions Visual Basic: elles font double emploi avec la classe String, elles ne sont pas toujours cohérentes avec le reste et cela embrouille.

Utilisez donc uniquement la classe String.

Les instructions VB, elles, sont bien connues des 'anciens' et font partie intégrante de VisualBasic; elles sont parfois plus simples. Mais elles ne fonctionnent pas comme des Objets, mais comme des instructions.

Elles font partie de l'espace de noms 'Microsoft.VisualBasic', il est 'chargé' par défaut et il n'y a pas lieu de l'importer. Par contre quand certains 'mots' sont communs à plusieurs classes ou instructions, il peut y avoir ambiguïté et il faut utiliser dans ce cas la syntaxe complète (avec l'espace de nom). Cela semble le cas pour left qui est un mot-clé Vb, mais aussi une propriété des contrôles. Pour lever l'ambiguïté, il faut écrire Microsoft.VisualBasic.left(C,i) par exemple.

Ces méthodes font souvent double emploi avec les méthodes de la classe String.



*Attention : le premier caractère est en position 1 dans les instructions VB.*

**Mid**

Permet de récupérer une sous-chaine.

```
MaString = "Mid Demonstration"
a = Mid(MaString, 1, 3) ' Retourne "Mid".
```

Retourne 3 caractères à partir du premier

Le premier paramètre indique la position du caractère où doit commencer la sous-chaine, en commençant à la position 1. (les caractères sont comptés 1, 2, 3…; on rappelle qu'avec SubString la sous-chaine, commence à la position 0.

```
a = Mid(MaString, 14)
```

Retourne "tion": du 14e à la fin (pas de 3e argument)

Mid permet aussi de remplacer une string dans une string

```
Mid(MaString, 1, 3) = "Fin"
```

=> MaString="Fin Demonstration"

**Left, Right** (Pas d'équivalent dans le Framework)

Retourne x caractères de gauche ou de droite :

```
a=Right(MaString,2)
```

a="on"

```
a=Microsoft.VisualBasic.Left(MaString,2)
```

a="Mi"

Notez bien que, pour lever toute ambiguïté avec les méthodes 'Left' d'autres classes, il faut indiquer Microsoft.VisualBasic.Left.

**Len**

Retourne la longueur de la chaine:

```
MyLen = Len(MaString)
```

Retourne 17.

**LTrim, RTrim**

Enlève les espaces à gauche ou à droite d'une chaine.

```
    a=LTrim("  RRRR")
```

a="RRR"

**InStr**

Retourne un entier spécifiant la position de début de la première chaine à l'intérieur d'une autre.

```
n=InStr(1,"aaaaRaa","R")        'retourne 5
```

Recherche à partir du premier caractère, à quelle position se trouve 'R' dans la chaine "aaaaRaa"

Si la chaine n'est pas trouvée, retourne 0.

**InStrRev**

Recherche aussi une chaine, mais de droite à gauche. La position de départ est le 3e argument.

InStrRev (Ch1, Ch2 , PosDépart)

**StrComp** Compare 2 chaines.

**Space**

Retourne une chaine d'espace: Space(10) retourne " "

**StrDup**

Retourne une chaine de caractères par duplication d'un caractère dont on a spécifié le nombre.

```
maString = StrDup(5, "P") ' Retourne "PPPPP"
```

**Asc**

Retourne le code de caractère du caractère. Il peut être compris entre 0 et 255 pour les valeurs du jeu de caractères codés sur un octet (SBCS) et entre -32 768 et 32 767 pour les valeurs du jeu de caractères codés sur deux octets (DBCS). La valeur retournée dépend de la page de codes.

AscW retourne le code Unicode du caractère entré. Il peut être compris entre 0 et 65 535.

```
    x=Asc("A")
```

retourne 65

```
x=Asc("ABCD")
```

retourne 65 : seul le premier caractère est pris en compte.

### Chr et ChrW

Retourne le caractère associé au code de caractère.

```
Chr(65)
```

retourne "A" 'cela dépend de la page de code.

On peut donner le numéro du caractère en hexadécimal, dans ce cas on le fait précéder de &H

```
Chr(&H20)
```

est équivalent de Chr(32) et retourne un caractère " ".

ChrW retourne le caractère correspondant à l'Unicode.

### GetChar

Retourne le caractère d'une chaine à une position donnée.

```
Dim maString As String = "AIDE"
Dim monChar As Char
monChar = GetChar(maString, 3) ' monChar = "D"
```

### LCase Ucase

Retourne la chaine en minuscules ou majuscules :

```
Lowercase = LCase(UpperCase)
```

### Lset Rset

Retourne une chaine alignée à gauche avec un nombre de caractères.

```
Dim maString As String = "gauche"
Dim r As String
r = LSet(maString, 2) ' Retourne "ga"

Si la chaine de départ est plus courte que la longueur spécifiée, des espaces sont ajoutés.
r = LSet(maString, 8) ' Retourne "gauche  "
```

### StrRevers

Retourne une chaine ou les caractères ont été inversés :

```
Dim maString As String = "STRESSED"
```

```
Dim revString As String

revString = StrReverse(maString)     ' Retourne "DESSERTS"
```

Marrant l'exemple !

**Filter** (VB2005)

Passe les Strings d'un tableau dans un autre tableau, si elles contiennent ou non une chaine.

```
TableauResultat= Filter( TableauChaine, Match, Include, Compare)
```

Match: chaine à chercher.

Include: Filtre sur la présence ou non de la chaine à chercher.

Compare en binaire ou en texte (majuscules = minuscules dans ce cas)

```
Dim TestStrings(2) As String
TestStrings(0) = "Ici"
TestStrings(1) = "Si"
TestStrings(2) = "si"
Dim subStrings() As String    'chaine des résultats

subStrings = Filter(TestStrings, "i", True, CompareMethod.Text)
'Retourne "Ici","Si","si"

subStrings = Filter(TestStrings, "si", True, CompareMethod.Binary)
' Retourne "si".

subStrings = Filter(TestStrings, "si", False, CompareMethod.Binary)
'Retourne "Ici","Si"
```

**Like**

Instruction hyper puissante: Like, elle compare une chaine String avec un modèle (Pattern), elle permet de voir si la chaine contient ou ne contient pas un ou des caractères, ou une plage de caractères. (c'est l'équivalent des expressions régulières du Framework)
UTILISER PLUTÔT le REGEX.

result = String Like Pattern

Si string correspond à pattern, la valeur de result est True ; s'il n'y a aucune correspondance, la valeur de result est False. Si string et pattern sont une chaine vide, le résultat est True. Sinon, si string ou pattern est une chaine vide, le résultat est False.

L'intérêt de Like est que l'on peut y mettre des caractères génériques:

? veut dire tout caractère unique.

* veut dire * ou plusieurs caractères.

# veut dire tout chiffre.

[caractères] veut dire tout caractère présent dans la liste.

[!caractères] veut dire tout caractère NON présent dans la liste.

- trait d'union permet de spécifier un début et une fin de plage.

Exemple :

```
Dim R As Boolean
R = "D" Like "D" ' Est-ce que "D" est égal à "D"? => True.


R = "F" Like "f" ' Est-ce que "F" est égal à "f"? => False.


R = "F" Like "FFF" ' Est-ce que "F" est égal à "FFF"? => False.


R = "cBBBc" Like "c*c" ' Est-ce que "cBBBc" répond au pattern (avoir un "c" au
'début, un  "c" à la fin, et des caractères au milieu? Retourne True.


R = "J" Like "[A-Z]" ' Est-ce que "J" est contenu dans les caractères allant de
'  A à Z? Retourne True.


R = "I" Like "[!A-Z]" ' Est-ce que "I" n'est PAS dans les caractères allant de
'  A à Z? Retourne  False.


R = "a4a" Like "a#a" ' Est-ce que "a4a" commence et finit par un
' "a" et à un nombre entre les 2? Retourne True.


R = "bM6f" Like "b[L-P]#[!c-e]" ' Est-ce que "bM6f"

'commence par  "b",

'a des caractères entre L et P

'un nombre

'se termine par un caractère non compris entre c et e

'retourne True
```

## V-E-1-c - Un exemple

Combinaison de chaines de caractères, de variables.

Souvent, on a besoin d'afficher une combinaison de chaines littérales, le contenu de variables, des résultats de calcul, c'est possible.

Exemple

Pour afficher dans un label 'Le carré de X est X2', avec une valeur dans la variable x :

```
Dim X As Integer = 2

Label1.Text= "Le carré de " & X & "  est " & X * X
```

Ce qui est entre guillemets est affiché tel quel. C'est le cas de "Le carré de" et de "est"

Ce qui n'est pas entre guillemets est évalué, le résultat est affiché. C'est le cas de X et X*X

Pour ne faire qu'une chaine on ajoute les bouts de chaines avec l'opérateur '&'.

Notez l'usage d'espace en fin de chaine pour que les mots et les chiffres ne se touchent pas.

```
Dim X As Integer

X=2

Label1.Text= "Le carré de " & X & "  est " & X * X
```

Affiche dans le label: "Le carré de 2 est 4".

## V-E-1-d - Comparaison de caractères (Option Compare)

On peut comparer 2 String avec :

```
les instructions '=', '>', '<':
```

```
Dim s1 As String ="ABCD"

Dim s2 As String ="XYZ"
```

Dans ce cas s1<s2 est vraie.

Car par défaut **Option Compare Binary**

Les caractères sont classés dans un ordre croissant (l'ordre de leur code Unicode).

Voyons l'ordre des certains caractères particuliers :

" " +,-./ 0123456789 :;ABCDEF abcdef èéê

On constate que l'ordre est espace puis quelques caractères spéciaux, les chiffres, les majuscules puis les minuscules, les accentués (voir le tableau d'Unicode).

Ainsi B<a

En utilisant Option Compare Binary, la plage [A-E] correspond à A, B, C, D et E.

Avec **Option Compare Text**

Les caractères sont classés dans un ordre qui reflète plus la réalité d'un texte :

Tous les types de a: A, a, À, à, puis tous les types de b: B, b…

Avec Option Compare Text, [A-E] correspond à A, a, À, à, B, b, C, c, D, d, E et e. La plage ne correspond pas à Ê ou ê parce que les caractères accentués viennent après les caractères non accentués dans l'ordre de tri.

Ainsi B>a

L'ordre des caractères est donc défini par Option Compare et aussi les paramètres régionaux du système sur lequel s'exécute le code.

On peut modifier Option Compare soit dans les propriétés de l'application (Menu 'Projet' puis 'Propriétés de ' puis onglet 'Compiler') ou dans un module en ajoutant en haut 'option Compare Text'.

**Grandes règles de comparaison**

La comparaison s'effectue de gauche à droite.

La comparaison s'effectue sur le premier caractère de chaque chaine.

Si le premier caractère est identique, la comparaison se fait sur le deuxième caractère…

"zz" > "za" est vrai

En cas de chaine du type "zz" et "zzz" , la seconde est supérieure.

"zz" < "zzz" est vrai.

Il y a quelques pièges.

Si je veux créer des chaines du genre 'un nombre puis le mot string' et qu'elles soient classées dans un ordre logique pour l'humain.

Je vais taper: "1string", "2string", "10string", "11string", "100string"

Le classement par Vb sera 'surprenant', car les chaines seront classées dans cet ordre :

"100string", "10string", "11string", "1string","2string"

Pourquoi? c'est l'application stricte des règles de comparaison: regardons le troisième caractère des 2 premières chaines (les 2 premiers caractères étant égaux), "0" est bien inférieur à "s" donc "100string" < "10string" est vrai !!

Pour résoudre le problème et obtenir un classement correct, il faut écrire des blocs numériques de même longueur et alignés à droite:

Écrire 010string et non 10string.

"001string", "002string", "010string", "011string", "100string" ' ici le tri est dans le bon ordre.

## V-E-1-e - Comparaison avec Equals et String.Compare

**Equals** retourne un Boolean égal à True si les 2 chaines sont égales.

Cette méthode effectue une comparaison ordinale respectant la casse (majuscules et minuscules ne sont pas égales) et non spécifique à la culture.

```
Dim myStr1 As [String] = "My Uncle Bill"
Dim myStr2 As [String] = "My uncle bill"
Dim r A Boolean= myStr1.Equals(MyStr2)

'Autre syntaxe:
Dim r A Boolean= String.Equals(MyStr1, MyStr2)

'Avec la première syntaxe, on peut ajouter des options de comparaison :
Dim r A Boolean= myStr1.Equals(MyStr2, CurrentCultureIgnoreCase)
```

**String.Compare** compare 2 chaines et retourne un Integer qui prend la valeur :
0 si les 2 chaines sont égales.
inférieur à 0 si string1 est inférieur à string2.
supérieur à 0 si string1 est supérieur à string2.

```
Dim myStr1 As [String] = "My Uncle Bill"
 Dim myStr2 As [String] = "My uncle bill"
        Dim r As Integer = String.Compare(myStr1, myStr2)
        MessageBox.Show(r.ToString)
```

Par défaut la culture en cours est utilisée et la comparaison est comme avec Option Compare=Binary: le code Unicode est utilisé.

```
'Affiche 1, car "A"<"a"
 MessageBox.Show(String.Compare("A", "a").ToString)
```

On peut ajouter **des options de comparaison :** IgnoreCase, IgnoreSymbol, IgnoreNonSpace, IgnoreWidth, IgnoreKanaType et StringSort.

```
Dim myStr1 As [String] = "My Uncle Bill"
 Dim myStr2 As [String] = "My uncle bill"
        Dim r As Integer = String.Compare(myStr1, myStr2, ignoreCase:=True)
        MessageBox.Show(r.ToString)

'Autre syntaxe: on a 2 options
Dim r As Integer = String.Compare(myStr1, myStr2, CompareOptions.IgnoreCase And
 CompareOptions.IgnoreSymbol)
```

On peut même comparer dans une autre culture :

```
Dim myComp As CompareInfo = CultureInfo.InvariantCulture.CompareInfo
Dim r As Integer= myComp.Compare(myStr1, myStr2))
```

Ici on compare 10 caractères en commençant par le deuxième caractère de chaque chaine en mode insensible à la casse (majuscules=minuscules).
Les options font partie de l'énumération StringComparison et pas de CompareOptions comme plus haut.

```
Dim myStr1 As [String] = "My Uncle Bill"
Dim myStr2 As [String] = "My uncle bill"
Dim r As Integer = String.Compare(myStr1, 2, myStr2, 2, 10,
 StringComparison.CurrentCultureIgnoreCase)
```

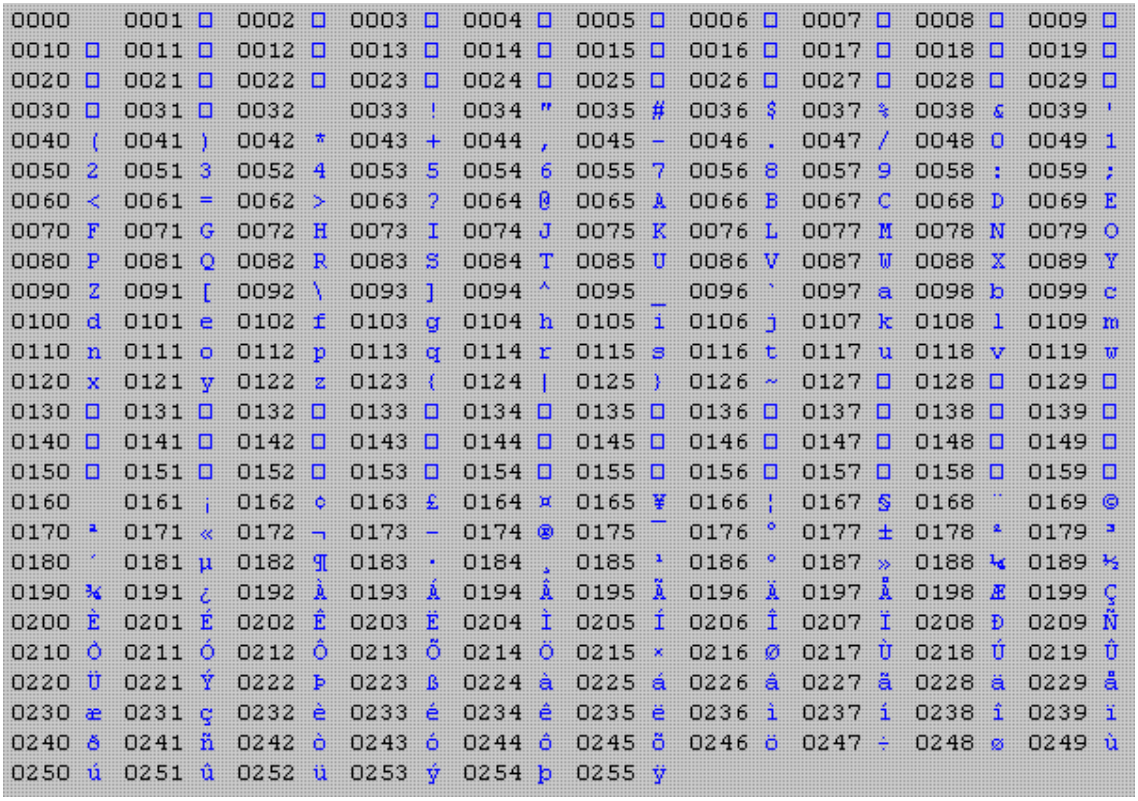
## V-E-1-f - Unicode

Les variables 'String' sont stockées sous la forme de séquences de 16 bits (2 octets) non signés dont les valeurs sont comprises entre 0 et 65 535. Chaque nombre représente un caractère Unicode. Une chaine peut contenir jusqu'à 2 milliards de caractères.

L'Unicode est donc un codage de caractères sur 16 bits qui contient tous les caractères d'usage courant dans les langues principales du monde.

Les premiers 128 codes (0-127) Unicode correspondent aux lettres et aux symboles du clavier américain standard. Ce sont les mêmes que ceux définis par le jeu de caractères ASCII (ancien codage sur un octet). Les 128 codes suivants (128-255) représentent les caractères spéciaux, tels que les lettres de l'alphabet latin, les accents, les symboles monétaires et les fractions. Les codes restants sont utilisés pour des symboles, y compris les caractères textuels mondiaux, les signes diacritiques, ainsi que les symboles mathématiques et techniques.

Voici les 255 premiers :




Le petit carré indique un caractère non imprimable (non affichable), certains caractères sont des caractères de contrôle comme le numéro 9 qui correspondant à tabulation, le numéro 13 qui correspond au retour à la ligne…

## V-E-2 - Variables 'Char'

Les variables Char contiennent un caractère et un seul, un caractère est stocké sous la forme d'un nombre de 16 bits (2 octets) non signé dont les valeurs sont comprises entre 0 et 65 535. Chaque nombre représente un seul caractère Unicode. Pour les conversions entre le type Char et les types numériques, il y a les fonctions AscW et ChrW qui peuvent être utilisées…

L'ajout du caractère 'c' à un littéral de chaine force ce dernier à être un type Char. À utiliser surtout si Option Strict (qui force à être strict…) est activé.

Exemple:

```
Option Strict On
'…
Dim C As Char
C = "A"c

'Autre manière de faire:
C=CChar("A")
'On convertit la String "A" en Char
```

Après déclaration une variable Char contient '' c'est-à-dire un caractère vide.

**String.ToCharArray**: Permet de passer une string dans un tableau de Char :

```
Dim maString As String = "abcdefghijklmnop"
Dim maArray As Char() = maString.ToCharArray
```

La variable maArray contient à présent un tableau composé de Char, chacun représentant un caractère de maString.

Pour mettre le tableau de Char dans une String :

```
Dim maNewString As String (maArray)

String.Chars():
```

vous pouvez extraire un caractère particulier en faisant référence à l'index de ce caractère par l'intermédiaire de la propriété Chars. Par exemple :

```
Dim maString As String = "ABCDE"

Dim monChar As Char

monChar = maString.Chars(3) ' monChar = "D"
```

Un caractère est-il numérique ? un chiffre ? une lettre ? un séparateur ? un espace ?

```
    Dim chA As Char
        chA = "A"c
        Dim ch1 As Char
        ch1 = "1"c
        Dim str As String
        str = "test string"

        Console.WriteLine(chA.CompareTo("B"c))          ' Output: "-1" '  A est plus petit que B
        Console.WriteLine(chA.Equals("A"c))             ' Output: "True" ' Egal?
        Console.WriteLine(Char.GetNumericValue(ch1))    ' Output: 1     'Convertir en valeur
numérique (double)
        Console.WriteLine(Char.IsControl(Chr(9)))       ' Output: "True"'  Est une caractère de
contrôle?
        Console.WriteLine(Char.IsDigit(ch1))            ' Output: "True"'  Est un chiffre
        Console.WriteLine(Char.IsLetter(","c))          ' Output: "False"' Est une lettre
        Console.WriteLine(Char.IsLower("u"c))           ' Output: "True" ' Est en minuscules
        Console.WriteLine(Char.IsNumber(ch1))           ' Output: "True" ' Est un nombre
        Console.WriteLine(Char.IsPunctuation("."c))     ' Output: "True" ' Est un caractère de
ponctuation
        Console.WriteLine(Char.IsSeparator(str, 4))     ' Output: "True" ' Est un séparateur
        Console.WriteLine(Char.IsSymbol("+"c))          ' Output: "True" ' Est un symbole
        Console.WriteLine(Char.IsWhiteSpace(str, 4))    ' Output: "True" ' Est un espace
        Console.WriteLine(Char.ToLower("M"c))           ' Output: "m"     ' Passe en minuscules
```

Existe aussi IsLetterOrDigit, IsUpper.

Bien sûr, si 'Option Strict= On', il faut ajouter .ToString à chaque ligne :

```
Console.WriteLine(Char.ToLower("M"c).ToString)
```

On note que l'on peut tester un caractère dans une chaine : Char.IsWhiteSpace(str, 4)

Autre manière de tester chaque caractère d'une String :

```
Dim V as string

For Each C As Char in V    'Pour chaque caractère de V…

      C…

Next
```

Ici la String est considérée comme une collection de Char. (C'est aussi une collection de String)

Mais on verra plus loin les collections et les boucles For Each.

Conversions Char <->Unicode

On rappelle que l'Unicode est le mode de codage interne des caractères.

```
Dim monUnicode As Short = Convert.ToInt16 ("B"c) ' le code Unicode  de B est 66.
Dim monChar As Char = Convert.ToChar (66) '   monChar="B"
```

Pour savoir si un caractère a un code Unicode précis il y a 2 méthodes :

if MyChar=Convert.ToChar(27) then…

ou

if AscW(MyChar)=27 then…

Si vous souhaitez utiliser Asc et Chr de VisualBasic :

```
Dim monAscii As Short = Asc("B") 'Asc donne le code ASCII ou l'Unicode (Ascw fait de même ?)
Dim monChar As Char= Chr(66) 'Char retourne le caractère qui a le code ASCII donné.
```

## V-E-3 - Et les chaines de longueur fixe

Débutant s'abstenir.

On a vu que les chaines de longueur fixe n'existent pas en VB.NET (compatibilité avec les autres langages oblige), ET ON S'EN PASSE TRÈS BIEN, mais il y a moyen de contourner le problème si nécessaire.

On peut créer une chaine d'une longueur déterminée (comme paramètres pour appeler une API par exemple) par :

```
Dim Buffer As String
Buffer = New String(CChar(" "), 25)
'appel
UserName = Left(Buffer, InStr(Buffer, Chr(0)) - 1) 'on lit jusqu'au caractère
'chr(0) qui est en interne le dernier caractère de la chaine
```

On peut aussi utiliser la Classe de compatibilité VB6: à éviter++

(Il faut charger dans les références du projet Microsoft.VisualBasic.Compatibility et Compatibility Data)

```
Dim MaChaineFixe As New VB6.FixedLengthString(100)

'pour l'initialiser en même temps:
Dim MaChaineFixe As New VB6.FixedLengthString(100, "hello")
```

Pour afficher la chaine fixe, utilisez MaChaineFixe.ToString.

Mais pour mettre une chaine dans cette chaine de longueur fixe!! galère pour trouver!!!

MaChaineFixe.Value="ghg"

Enfin ce type de chaine fixe ne peut pas être utilisé dans les structures, mais il y a un autre moyen pour les structures. On verra cela plus loin.

Donc les chaines fixes sont à éviter.

# V-E-4 - Regex, expressions régulières

Débutant s'abstenir.

Les expressions régulières sont une manière de rechercher (de remplacer, d'extraire) une sous-chaine ou un modèle d'une chaine de caractères.

On a un modèle, on veut voir si une chaine contient des parties répondant à ce modèle. Simplement pour voir ou pour séparer des sous-chaines, remplacer…

Exemple: une chaine ne contient-elle que les lettres 'abc', commence-t-elle par 'hello' ou 'HELLO', est-elle une adresse mail valide, un code postal valide? Comment découper une chaine ayant comme séparateur des chiffres? Comment remplacer tous les caractères 'wxyz' par des '-'?….

## V-E-4-a - Principe du regex

Les Regex servent à :
- vérifier la syntaxe d'une chaine de caractères ;
- remplacer une partie de la chaine par une autre chaine ;
- découper une chaine de caractères ;
-extraire certaines sous-chaines.

Pour expliquer le principe, on va **comparer une chaine String avec un modèle**, un 'Pattern', nommé 'REGEX'.

Cela permet de vérifier la syntaxe d'une chaine de caractères. La chaine de caractères à examiner respecte-t-elle un motif (le Regex) décrivant la syntaxe attendue ?

Nécessite **l'import d'un espace de noms** en haut du module :

```
Imports System.Text.RegularExpressions
```

**Premier exemple très simple: Voir si une String ne contient que des chiffres.**

Il faut dans un premier temps **instancier un Regex contenant le motif**. Comme exemple nous allons utiliser un motif permettant de voir si la String contient uniquement des chiffres :

```
Dim rg As New Regex("[0-9]")
```

Notez que le motif est entre guillemets. Le motif [0-9] signifie: tous les caractères entre 0 et 9.
Ensuite on va utiliser la propriété **IsMatch** du Regex rg pour voir si la String à vérifier ("4545896245" ici comme exemple) répond au motif. Elle retourne True si la String répond au motif.

```
rg.IsMatch("4545896245")  'retourne True, car il n'y a que des chiffres.
```

Pour afficher :

```
MsgBox(rg.IsMatch("45gdGRj1"))  'Affiche False dans une Box
```

**Second exemple pas simple: Voir si une String contient une adresse mail valide.**

Il faut dans un premier temps instancier un Regex contenant le motif. Comme exemple nous allons utiliser un motif permettant de voir si une adresse mail est valide :

```
Dim rg As New Regex("^([\w]+)@([\w]+)\.([\w]+)$")
```

Bonjour le motif !!!!
Ensuite on va utiliser la propriété IsMatch du Regex pour voir si la String à vérifier répond au motif. Elle retourne True si la String a bien la syntaxe d'une adresse mail.

```
MsgBox(rg.IsMatch("philippe@lasserrelyon.fr")) 'affiche True
```

C'est donc extrêmement puissant !! mais on l'a compris tout l'art est d'écrire le motif !!

## V-E-4-b - Caractères pour modèle regex

**Principaux caractères composant les motifs :**

**hello** veut dire le texte 'hello'
\ Caractère d'échappement: \. veut dire un point
**^** Début de ligne
**$** Fin de ligne
**.** N'importe quel caractère
**|** Alternative: toto|lulu veut dire toto ou lulu
**( )** Groupement
**-**Intervalle de caractères: a-c veut dire a b ou c
**[ ]** Ensemble de caractères
**[^]** Tout sauf un ensemble de caractères
Après le caractère ou un groupe de caractères, on peut indiquer le nombre de caractères :
**+** 1 fois ou plus
**?** 0 ou 1 fois
***** 0 fois ou plus
**{x}** x fois exactement
**{x,}** x fois au moins
**{x, y}** x fois minimum, y maximum

**Il y a aussi des métacaractères :**

**\s** Caractère d'espacement (espace, tabulation, saut de page, etc)
**\S** Tout ce qui n'est pas un espacement
**\d** Un chiffre
**\D** Tout sauf un chiffre
**\w** Un caractère alphanumérique
**\W** Tout sauf un caractère alphanumérique
**\r** retour chariot
Il y a plein d'autres caractères et métacaractères…

## V-E-4-c - Exemples

**Petits exemples**

**[\.]** contient un point car "." ne signifie pas 'n'importe quel caractère
il y a un caractère d'échappement avant.
**^z$** contient uniquement z (entre début et fin).
**es$** finit par "es"
**^.$** contient un seul caractère (entre début et fin)
**^(i|A)** commence par i ou A
**^((b)|(er))** commence par b ou er
**^[a-c]** commence par a,b ou c
**[a-zA-Z]** caractères en majuscules ou minuscules
**[A-Z]+** un mot en majuscules

**[A-Z]{1,7}** un mot en majuscules de 1 à 7 caractères
**[0-9]** contient un chiffre
**[^0-9]** tous sauf un chiffre
**\d+** entier positif
**[+-]?d+** entier positif ou négatif , le signe + ou - est facultatif
**^[^y]** ne commence pas par y
**^(o)+** commence par un ou plusieurs o
**^(a)*** peut ou non commencer par a
**a{2,4}** deux, trois ou quatre fois "a"
**[12\^]** chaine contenant les chiffres 1 ou 2 ou le symbole ^
**^[0-9]+-[0-9]+$** 2 nombres séparés par un tiret: 55-4589 4586-85
**[aeiou]\d** Une voyelle et un chiffre: a2 i5
**\d+** des chiffres (un ou plusieurs)

**Validité de différent nom:**
^[pP]hilip(pe)?$ True si philip Philip philippe Philippe

**Validité d'une adresse ip :**
^(25[0-5]|2[0-4]\d|[0-1]?\d?\d)(\.(25[0-5]|2[0-4]\d|[0-1]?\d?\d)){3}$

## V-E-4-d - Divers utilisations de Regex

### Validation d'une chaine de caractères : IsMatch

```
Funtion  SiValideMail (Adresse As String) As Boolean

Dim rg As New Regex("^([\w]+)@([\w]+)\.([\w]+)$")
Return rg.IsMatch(Adresse)

End Function
```

Retourne True si la chaine envoyée à la fonction est une adresse mail valide.

### Remplacement dans une chaine de caractères : Replace

```
Funtion  Remplace (Chaine As String) As String

Dim rg As New Regex("hello|salut|buenas dias")
Return rg.Replace(Chaine, "bonjour")

End Function
```

Retourne une chaine où hello, salut, buenas dias ont été remplacés par bonjour…

### Découpage d'une chaine de caractères : Split.

Split permet de découper une chaine de caractères et de la mettre dans un tableau en utilisant l'expression régulière comme séparateur.

```
Dim ch As String ="az45er78ty"

Dim rg As New Regex("\d+")
Dim t() As String= rg.Split(ch)
```

ch ="az45er78ty"', retourne t(1)="az" t(2)="er" t(3)="ty"
Retourne un tableau de chaines découpées à partir de ch avec comme séparateur les chiffres (séparateur : 1 ou plusieurs chiffres).

### Extraire des chaines de caractères : Matches.

Matches permet d'extraire les séquences de caractères correspondant à un motif. Retourne une MatchCollection qui a une propriété Count indiquant le nombre d'éléments retournés. Cette MatchCollection en lecture seule est composée de 'Match' qui ont les propriétés 'Value' (chaine retournée) 'Index' (position: numéro caractère) et 'Length'.

On va extraire les chaines de 2 caractères (caractères de a à z, pas les ',').

```
Dim ch As String ="az,er,ty"

Dim rg As New Regex("[a-z]{2}")
Dim mac As MatchCollection = rg.Matches(ch)

For Each m As Match In mac
 MsgBox( m.Value &" en position " & m.Index)
Next
```

### Extraire des mots d'une chaine: Matches.

Matches permet d'extraire les séquences de lettres donc des mots. Retourne une MatchCollection qui a une propriété Count indiquant le nombre d'éléments retournés.

Le motif pourrait être [A-Za-z]+, mais il y a des problèmes avec les accentués qui ne font pas partie de a-z!! il faut utiliser le motif: (\p{Lu}|\p{Ll})+ (Explication: \p{LU}: caractères Unicode majuscules, \p{Ll}: caractères Unicode minuscules).

```
Dim ch As String ="Ceci est un cours vb"

Dim rg As New Regex("(\p{Lu}|\p{Ll})+")
Dim mac As MatchCollection = rg.Matches(ch)

For Each m As Match In mac
 MsgBox( m.Value &" en position " & m.Index)
Next
```

Le motif "\b(?!(le|un|une|et|de|la)\b)(\p{Lu}|\p{Ll})+" permet en plus d'éliminer les un, une, le, la…

### Méthodes statiques

On peut utiliser une autre syntaxe (pour Replace, Match, Matches) avec une méthode statique (sans instanciation du regex):

```
Funtion  Remplace (Chaine As String) As String

Return Regex.Replace("salut", "bonjour")

End Function
```

### Options

Les méthodes statiques ou non peuvent avoir un ou plusieurs arguments optionnels (les séparer par 'Or') :
RegexOptions.IgnoreCase : ignore la case oui, oui…
RegexOptions.IgnorePatternWhitespace : ignore l'espace la tabulation, nouvelle ligne
RegexOptions.Compiled : accélère, car compile le regex.
RegexOptions.MultiLine : applique les caractères de début et fin à chaque ligne.

```
Funtion  Remplace (Chaine As String) As String

Return Regex.Replace("salut", "bonjour" , RegexOptions.IgnoreCase Or RegexOptions.Compiled)
```

```
End Function
```

ou en instanciant le regex :

```
Dim rg As New Regex("salut", RegexOptions.IgnoreCase Or RegexOptions.Compiled)
```

On pourrait écrire des livres sur les expressions régulières !!! Pour trouver des motifs, voir : d**es centaines de chaines de motif toutes faites.**

## V-E-5 - StringBuilder

Débutant s'abstenir.

Les opérations sur les Strings peuvent être accélérées, il faut pour cela utiliser les StringBuilder.

Exemple d'une opération coûteuse en temps :

```
Dim s As String = "bonjour"

s += "mon" + "ami"
```

Le Framework va créer 3 chaines en mémoire avec toutes les pertes en mémoire et en temps que cela implique. (il crée une chaine "bonjour" puis il crée une chaine "bonjour mon" puis…
On dit que le type String est **immutable**.
Pour l'exemple précédent, cela ralentit peu, mais dans une boucle qui concatène 10 000 chaines !!

Pour effectuer des opérations répétées sur les string, le framework dispose donc d'une classe spécialement conçue et **optimisée** pour ça : System.Text.StringBuilder.

Pour l'utiliser, rien de plus simple :

```
Dim sb As new System.Text.StringBuilder()

    sb.Append("bonjour")

    sb.Append("mon ami")

    Dim s As String

    s = sb.ToString()
```

Il y a création et utilisation d'une seule chaine : sb La méthode *ToString* de la classe StringBuilder renvoie la chaine qu'utilise en interne l'instance de StringBuilder.

Pour comparer 2 StringBuilder utiliser la méthode Equals plutôt que =.

À partir de vb 2010 il existe **StringBuilder.Clear**.

## V-F - Variables numériques


4815162342

Une variable numérique peut contenir des données numériques.

**On a vu qu'une variable numérique peut être entière :**

- Integer (entier signé) ;
- Short (entier court signé) ;
- Long (Entier long signé) ;
- Byte (entier non signé de valeur 0 à 255).

**À partir de VB2005 il y a aussi :**

- UInteger (entier non signé) ;
- UShort (entier court non signé) ;
- ULong (Entier long non signé) ;
- SByte (entier signé).

**Une variable numérique peut aussi être un fractionnaire :**

- Single (virgule flottante simple précision) ;
- Double (virgule flottante double précision) ;
- Decimal (virgule fixe haute précision).

**À partir de vb 2010 il y a en plus :**

- BigInteger ( Entier signé très grand (sans limite supérieure ou inférieure) (VB2010) ) ;
- Complex (Nombre complexe).

On déclare une variable numérique avec Dim.

```
Dim i As Integer
```

Après déclaration une variable numérique contient 0.

on peut initialiser en même temps qu'on déclare :

```
Dim i As Integer= 3
```

Si la variable est numérique, il faut la transformer en String avant de l'afficher :

```
Dim I As Integer=12

Label.Text = I.ToString
```

**.ToString** fait partie des méthodes de la classe String. Il y en a d'autres :

**.GetType** retourne le type de la variable

```
Dim i As Integer
i=3 ' Il faut initialiser i avant d'utiliser GetType

Label1.Text = i.GetType.ToString  'Affiche: System.Int32
```

**.MaxValue .MinValue** donne le plus grand et le plus petit nombre possible dans le type de la variable.

On verra qu'on peut utiliser des opérateurs + - * / .

```
Dim I As Integer=2
```

```
Dim J As Integer

J=I+3 ' J est égal à 5, car on affecte à J la valeur I+3
```

On rappelle que le séparateur est le point :

J=1.2 veut dire J=1,2 en bon français !!

de même pour Str et Val du VisualBasic.

Par contre pour les instructions du Framework (CType, TryCaste, Cint…), le séparateur est celui de la culture (',' en culture française, '.' en culture us).

## V-F-1 - La Classe Math du Framework

Pour qu'elle soit disponible, il faut d'abord importer l'espace de noms 'Math' du FrameWork :

Pour cela il faut taper en haut de la fenêtre (au-dessus de public Class) :

```
Imports System.Math
```

Si on n'a pas importé l'espace de nom, il faut ajouter Math. avant le nom de la fonction. Exemple :

```
R=Math.Abs(N)
```

On verra plus loin ce que cela signifie.

```
Dim N As Single

Dim R As Single


R=Abs(N)     'retourne la valeur absolue

             'Si N=-1.2  R=1.2

R=Sign(N)    'retourne le signe

             'Si N=-1.2  R=-1 (négatif) ; retourne 1 si nombre positif


R=Round(N)   'retourne le nombre entier le plus proche

             ' N=1.7     R=2

             ' N=1.2     R=1

             ' N=1.5     R=2
```

Pour Round, si nombre <0.5 retourne 0 si > ou = à 0.5 retourne 1; c'est la manière d'arrondir les nombres en Euros, comme sur la feuille d'impôts.

On peut donner en second paramètre le nombre de digits : Math.Round(Valeur, 2)donne 2 décimales après la virgule.

```
R=Truncate(N)
```

Retourne la partie entière (enlève les chiffres après la virgule, arrondie à l'entier le plus proche en allant vers zéro).

'N=1.7 R=1

```
R=Floor(N)
```

Retourne le plus grand entier égal ou inférieur (arrondi à l'entier inférieur le plus proche en allant vers l'infini négatif).

N=1.7 R=1

```
R=Ceiling(N)
```

Retourne le plus petit entier égal ou supérieur. (arrondi à l'entier supérieur le plus proche en allant vers l'infini positif).

N=1.2 R=2

```
R=Max(2,3)
```

Retourne le plus grand des 2 nombres.

Retourne 3

```
R=Min(2,3)
```

Retourne le plus petit des 2 nombres.

Retourne 2

```
R=Pow(2,3)
```

Retourne 2 puissance 3.

Retourne 8

```
R=Sqrt(9)
```

Retourne la racine carrée.

Retourne 3

longResult = Math.BigMul(int1, int2) 'BigMul donne le résultat de la multiplication de 2 entiers sous forme d'un long.

intResult = Math.DivRem(int1, int2, Reste) DivRem donne le résultat (intResult) de la division de int1 par int2 et retourne le reste (Reste), cela pour des entiers.

Il existe aussi Log, Log10, Exp.

Bien sur il y a aussi Sin Cos Tan, Sinh Cosh Tanh (pour hyperbolique) Asin Acos Atan Atan2.

Prenons un exemple :

```
Imports System.Math
Dim MonAngle, MaSecant As Double
MonAngle = 1.3 '  angle en  radians.
MaSecant = 1 / Cos(MonAngle) ' Calcul la sécante.
```

On remarque que les angles sont en radians.

Rappel:2pi=360° ; Angle en radians= (2pi/360)*Angle en degrés.

## V-F-2 - Les instructions du langage VisualBasic

**Int** et **Fix** qui suppriment toutes deux la partie fractionnelle et retournent l'entier.

```
Dim R As Single= 1.7

Int(R)     'retourne 1
```

Si le nombre est négatif, Int retourne le premier entier négatif inférieur ou égal au nombre, alors que Fix retourne le premier entier négatif supérieur ou égal au nombre. Par exemple, Int convertit -8,4 en -9 et Fix convertit -8,4 en -8.

## V-F-3 - Dépassement de capacité, 'Non Nombre'

Testé en VB2005

On a vu que , codées sur un nombre de bits défini, les variables numériques ne peuvent pas avoir des valeurs très très grandes. MaxValue donne le plus grand nombre possible dans le type de la variable. (MinValue le plus petit nombre) Que se passe-t-il , si on dépasse la valeur maximum ?

- Si on affecte à une variable entière une valeur supérieure à .MaxValue cela déclenche une erreur (on dit **une exception** de type OverFlow) et cela plante.
- Si on affecte à une valeur à virgule flottante(un Single par exemple), une valeur supérieure à .MaxValue, la variable prend la valeur 'infinie' ( +ou - infinie: Single.NegativeInfinity ou Single.PositiveInfinity)).

Exemple

**IsInfinity, IsNegativeInfinity, IsPositiveInfinity** permettent de tester si le résultat d'un calcul dépasse les valeurs autorisées pour le Type virgule flottante.

```
Dim s As Single = 2147483647 ^ 33

If Single.IsInfinity(s) Then MsgBox("infinie")
```

s prend la valeur Single.PositiveInfinity.

Les opérations en virgule flottante retournent **NaN** pour signaler que le résultat de l'opération est non défini. Par exemple, le résultat de la division de 0,0 par 0,0 est NaN.

On peut tester une expression par **IsNan**.

Exemple :

```
If Single.IsNaN(0 / 0) Then
```

## V-F-4 - Problème de précision

Integer Single ou Decimal ? Précision ou rapidité ?

Vu le système de codage en interne, on a vu qu'avec les variables en virgule flottante, comme les Single par exemple, certaines valeurs ou calculs sont représentés avec une certaine approximation, infime, mais réelle.
Souvent le calcul est exact, mais parfois (rarement en pratique courante) on peut avoir un infime manque de précision.
Voici un exemple: on a 0,0001, avec une boucle on l'additionne dix mille fois :

```
        Dim i As Integer
        Dim k As Single
        Dim j As Single = 0.0001

        For i = 1 To 10000
            k = k + j
        Next

        MsgBox(k.ToString)' Affiche 1.000054

        Dim k1 As Decimal
        Dim j1 As Decimal = 0.0001

        For i = 1 To 10000
            k1 = k1 + j1
        Next

        MsgBox(k1.ToString)' Affiche 1
```

Avec des Decimal on obtient bien 1, mais avec des Single on obtient 1,000054 !!
Cela peut poser des problèmes si on compare le résultat du calcul avec 1.

Avec les entiers la précision est parfaite.

Dans notre exemple le calcul est DIX fois plus long avec les Decimal qu'avec les Single.

## V-F-5 - BigInteger

Un BigInteger est un Entier signé très grand (sans limite supérieure ou inférieure).
Il apparait dans vb 2010.

La valeur, en théorie, n'a pas de limites supérieure ou inférieure.

Il faut charger dans les références (passer par propriétés du projet) pour charger System.Numerics puis Importer cet espace.

Instanciation :

```
Imports System.Numerics
'Instanciation directe avec new
Dim MyBitInteger As New BigInteger(17903)
'Si variable avec virgule, la partie après la virgule sera tronquée.

'À partir d'un long
Dim MylongValue As Long = 631548935
Dim MyBigInteger2 As BigInteger = Mylong

Dim MyBigInteger As BigInteger = CType(64312.65d, BigInteger)
```

On peut utiliser les opérations mathématiques de base telles que l'addition, la soustraction, la division, la multiplication (+ -* /), la négation et la négation unaire.
Vous pouvez également comparer deux valeurs .Comme les autres types intégraux, BigInteger prend en charge également les opérateurs de bit, de décalage vers la droite et de décalage vers la gauche And, Or et XOr Il existe aussi Add, Divide, Multiply, Negate et **Subtract**.
**Sign**, retourne une valeur qui indique le signe d'une valeur BigInteger.
**Abs** retourne la valeur absolue d'une valeur BigInteger.
**DivRem** retourne à la fois le quotient et reste d'une opération de division.
**GreatestCommonDivisor** retourne le plus grand diviseur commun de deux valeurs BigInteger.

Exemple: Creation de 2 BitInteger (2190 Puissance 2 et 42656*35); affichage du plus grand commun diviseur.

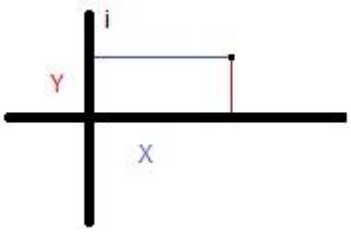
```
Dim n1 As BigInteger = BigInteger.Pow(2190, 2)
Dim n2 As BigInteger = BigInteger.Multiply(42656, 35)

Console.WriteLine("Le plus grand commun diviseur de {0} et de {1} est {2}.", _
                  n1, n2, BigInteger.GreatestCommonDivisor(n1, n2))
```

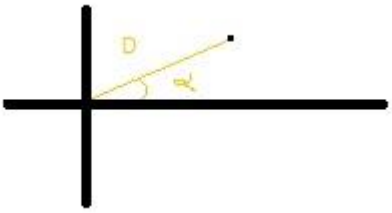
Les calculs avec les BigInteger sont lents (20 fois plus lent qu'avec les Single pour 10 000 additions par exemple). Comme ce sont des entiers, il ne devrait pas y avoir d'erreur d'arrondi.

## V-F-6 - Nombre complexe

Les nombres complexes sont une notion mathématique (je les avais étudiés en terminal S il y a quelques années). Ils sont utilisés dans certains calculs en génie électrique.

Un nombre complexe comprend une partie réelle et une partie imaginaire.
Un nombre complexe z s'écrit sous la forme suivante : **z = x + yi**, où x et y sont des nombres réels, et i est l'unité imaginaire qui a la propriété i2 = -1.
La partie réelle du nombre complexe est représentée par x, et la partie imaginaire du nombre complexe est représentée par y. Un nombre complexe peut être représenté comme un point dans un système de coordonnées à deux dimensions, appelé plan complexe.La partie réelle est positionnée sur l'axe des abscisses (axe horizontal), et la partie imaginaire est positionnée sur l'axe des ordonnées (axe vertical).
Tout point peut également être exprimé, en utilisant le système de coordonnées polaires.
Un point est caractérisé par deux nombres :
- sa grandeur, qui est la distance entre le point et l'origine (autrement dit, 0,0) ; -sa phase, qui est l'angle entre le véritable axe et la ligne tirée entre l'origine et le point.




En vb x et y (coordonnées cartésiennes) sont des 'Double'.
Les propriétés Real et Imaginary retournent la part réelle et imaginaire du nombre complexe.
La magnitude (d) et la phase (alpha exprimé en radians) sont des 'Double'.
(Pour convertir des degrés en radians, multiplier par Math.Pi/180). Les propriétés Magnitude et Phase retournent d et alpha.

Il faut charger dans les références (passer par propriétés du projet) pour charger System.Numerics puis Importer cet espace.

```
Imports System.Numerics

Module Example
   Public Sub Main()
      ' Creationn d'un complexe 11+ 6i .
      Dim c1 As New Complex(11, 6)
      Console.WriteLine(c1) 'Affiche  (11, 6)

      ' Assigne un  Double à un complex .
      Dim c2 As Complex = 3.1416
      Console.WriteLine(c2) 'Affiche  (3.1416, 0)


      ' Assign la valeur retournée .
      Dim c3 As Complex = Complex.One + Complex.One
      Console.WriteLine(c3) 'Affiche  (2, 0)
```

```
        ' Instancie un complex à partir des coordonnées polaires.
        Dim c4 As Complex = Complex.FromPolarCoordinates(10, .524)
        Console.WriteLine(c4) 'Affiche  (8.65824721882145, 5.00347430269914)

       'Affichage coordonnées cartésiennes
        Console.Write(c4.Real)
        Console.Write(c4.Imaginary)
        'Affichage coordonnées polaires
        Console.WriteLine("   Magnitude: {0}", c4.Magnitude)
        Console.WriteLine("   Phase:     {0} radians", c4.Phase)

    End Sub
End Module
```

Opérateurs

Les opérations sur les nombres complexes obéissent à des règles mathématiques particulières (voir un cours de maths). Vb connait ces règles.
En plus de quatre opérations arithmétiques fondamentales (+ - / * ou Add, Substrat, Divise, Multiply), vous pouvez élever un nombre complexe à une puissance spécifiée (Pow), rechercher la racine carrée d'un nombre complexe (Sqrt) et obtenir la valeur absolue d'un nombre complexe (Abs).
Vous pouvez obtenir l'inverse (Negate) le Log et les valeurs trigonométriques (Cos, Sin…). Enfin on peut comparer avec Equals et =.

```
Dim c4 As New Complex(1, 1)
        Dim c2 As New Complex(2, 2)
        Dim c1 As New Complex
        ' c1 = c4 - c2
        c1 = Complex.Subtract(c4, c2)
        ' ou c1=c4-c2
        Console.Write(c1)
```

Attention, les valeurs étant des doubles il peut y avoir des problèmes d'arrondis: perte de précision lors de certaines opérations ce qui peut poser des problèmes au cours de comparaisons.

Pour formater une impression de nombre complexe, on peut utiliser ToString ou le ComplexFormatter :

```
Dim c1 As Complex = New Complex(12.1, 15.4)
      Console.WriteLine("Formatting with ToString():       " +
                        c1.ToString())
      Console.WriteLine("Formatting with ToString(format): " +
                        c1.ToString("N2"))
      Console.WriteLine("Custom formatting with I0:        " +
                        String.Format(New ComplexFormatter(), "{0:I0}", c1))
      Console.WriteLine("Custom formatting with J3:        " +
                        String.Format(New ComplexFormatter(), "{0:J3}", c1))
                        ' The example displays the following output:
'    Formatting with ToString():       (12.1, 15.4)
'    Formatting with ToString(format): (12.10, 15.40)
'    Custom formatting with I0:        12 + 15i
'    Custom formatting with J3:        12.100 + 15.400j
'Merci Microsoft pour cet exemple
```

## V-G - Conversion, séparateur décimal

On a vu qu'on peut afficher les chaines de caractères (des 'String'), par ailleurs, on fait des calculs avec les variables numériques (Integer, Single…).

On a donc besoin sans arrêt de faire des calculs avec des variables numériques et de transformer le résultat en String pour l'afficher et vice versa.

Est-il possible de convertir une variable d'un type à un autre ? OUI !!

On aura donc besoin de savoir transformer des variables de tous types en d'autres types.

## V-G-1 - Conversion numérique vers String

**Quel intérêt de convertir ?**

Après avoir effectué un calcul, on veut afficher un résultat numérique.

On ne peut afficher que des Strings (chaine de caractères) dans un label ou un TextBox par exemple.

Aussi, il faut transformer cette valeur numérique en chaine avant de l'afficher, on le fait avec la méthode ".ToString":

```
Dim i As Integer=12         'On déclare une variable I qu'on initialise à 12

Label.text = i.ToString
```

La valeur de i est transformée en String puis affectée à la propriété Text du label, ce qui affiche '12'

On verra plus loin qu'on peut ajouter des paramètres.

Il existe aussi **Cstr :**

```
Dim i As Integer=12         'On déclare une variable I qu'on initialise à 12

Label.text = CStr(i)
```

## V-G-2 - Conversion String vers numérique

À l'inverse une chaine de caractères peut être transformée en numérique.

Par exemple, l'utilisateur doit saisir un nombre, il saisit un nombre dans une boite de saisie (InputBox), mais il tape des caractères au clavier et c'est cette chaine de caractères qui est retournée, il faut la transformer en numérique Integer grâce à **CInt**.

```
Dim s as String

Dim i as Integer

s= InputBox ("Test", "Taper un nombre") 'Saisie dans une InputBox  d'un nombre par l'utilisateur.

's contient maintenant une chaine de caractères, "45" par exemple

i= CInt(S)      'on transforme la chaine s en Integer
```

On peut aussi utiliser **Parse :**

```
Dim s as String

Dim i as Integer

s= InputBox ("Test", "Taper un nombre") 'Saisie dans une InputBox  d'un nombre par l'utilisateur.
```

```
's contient maintenant une chaine de caractères, "45" par exemple

i= Integer.Parse(s)     'on transforme la chaine s en Integer
```

Bizarre cette syntaxe!! en fait c'est le type Integer qui a une méthode (Parse) qui transforme une chaine en entier.

**On peut aussi utiliser, et c'est plus simple, CType** pour convertir n'importe quel type en n'importe quel type :

Il suffit de donner à cette fonction la variable à modifier et le type à obtenir.

```
Dim i As Integer

Dim s As String= "12"

 i=CType(s,Integer)   ' s est la variable à modifier, Integer est le type à obtenir.
```

i contient maintenant l'entier 12.

Voilà ces quelques instructions devraient suffire pour un usage courant !! Mais il en existe d'autres.

## V-G-3 - Tous les modes de conversion

**CType pour tout.**

CType peut aussi servir à convertir de la même manière un single en double, un Short en Integer…

Il est donc possible de convertir un type de variable en un autre.

Il suffit de donner à cette fonction la variable à modifier et le type à obtenir.

```
Dim d As Double = 2.65

 Dim i As Integer

 i=CType(d,Integer)     'conversion d'un Double en entier
```

```
Dim d As Double = 2.65

 Dim s As String

 s=CType(d,String)     'conversion  d'un Double  en String
```

```
Dim d As Integer = 2

 Dim S As Single

 S=CType(d, Single)     'conversion d'un Integer en Single
```

Pour les forts.

**DirectCast** fait de même, mais on doit utiliser une variable ByRef.

i=DirectCast(s,Integer) 'S doit être ByRef.

Par contre DirectCast nécessite que le type d'exécution d'une variable objet soit identique au type spécifié.

```
' nécessite Option Strict Off.
Dim Q As Object = 2.37   ' crée un objet contenant un double.
```

```
Dim K As Integer =CType(Q, Integer)        'Marche

Dim J As Integer = DirectCast(Q, Integer)    ' échoue
```

DirectCast échoue, car le type d'exécution de Q est Double. CType réussit, car Double peut être converti en Integer, mais DirectCast échoue, car le type d'exécution de Q n'est pas encore Integer.

**TryCast** à partir de VB 2005 (Framework 2) 


TryCast fonctionne comme DirectCast, mais retourne Nothing si la conversion est impossible (et ne plante pas! autrement dit, il ne lève pas d'exceptions).

```
Dim chaine As String = TryCast(b, String)
 If IsNothing(chaine) Then…
```

**Fonctions spécifiques**

CType fait toutes les conversions, mais on peut aussi utiliser des fonctions qui sont spécifiques au type de la variable de retour : le nom de ces fonctions contient le nom du type de la variable de retour.

```
CBool()   'Pour convertir en Booléen
CByte()   'Pour convertir en octet
CChar()   'Pour convertir en Char
CDate()   'Pour convertir en Date
CDbl()    'Pour convertir en Double
CDec()    'Pour convertir en Decimal
CInt()    'Pour convertir en Integer
CLng()    'Pour convertir en Long
CObj()    'Pour convertir en Objet
CShort()  'Pour convertir en  Short
CSng()    'Pour convertir en Single
CStr()    'Pour convertir en String
'Et en VB 2005
CSByte()  'Pour convertir en SByte
CUShort() 'Pour convertir en  UShort
CUInt()   'Pour convertir en UInteger
CULng()   'Pour convertir en ULong
```

Exemple CDbl retourne un 'Double'.

```
Dim I As Integer=123

Dim D As Double

D=CDbl(I)    'donnera D=123  D est un Double (réel double précision)
```

Ces fonctions sont plus rapides, car elles sont spécifiques.

Remarque

Les fonctions CInt et CLng arrondissent les parties décimales égales à 0,5 au nombre pair le plus proche. Par exemple, 0,5 s'arrondit à 0 et 1,5 s'arrondit à 2. Bizarre !!

**Val et Str (de MicroSoft.VisualBasic) existe aussi:**

Ouf pour les anciens !!

Ces fonctions permettent aussi la conversion String=>Numérique et Numérique=>String

**Val** donne la valeur numérique d'une expression String.

```
Dim i As Integer

i=Val("5")     ' i=5
```

Val s'arrête au premier caractère non numérique.

Val("12er") retourne 12

Val reconnaît le point (et pas la virgule).

```
Dim i As Double

i=Val("5.45")     ' donnera i=5,45

i=Val("5,45")     ' donnera i=5
```

**Str** transforme une valeur numérique en String :

```
Dim s As String

s=Str(1999)     ' s=" 1999"
```

Noter bien : Str ajoute un espace à gauche ou le signe'-' si le nombre est négatif.

Str ne reconnaît que le point comme séparateur décimal. Pour utiliser les autres séparateurs internationaux, il faut utiliser la fonction CStr().

**La Classe System.Convert**

La Classe System.Convert permet la conversion d'un type de base vers un autre:

.ToString en fait partie

Exemple

Pour convertir un Single en Byte (entier 8 bits non signé)

.ToByte

Pour convertir un Byte en Single:

.ToSingle

```
singleVal = System.Convert.ToSingle(byteVal)
```

En Decimal

.ToDecimal

On a des méthodes pour pratiquement convertir tous les types en tous les types. Cherchez !!

On verra plus loin, la fonction **Format** utilisée pour convertir une valeur numérique en une chaine de caractères généralement destinée à l'affichage en imposant un formatage: vous pouvez mettre un format pour l'affichage des

dates, des heures, un format pour les monnaies ou les nombres (nombre de chiffres affichés, séparateur…) Ce n'est pas à proprement parler une conversion, mais plutôt une mise en forme.

```
Dim nb As Single = 12.23
MsgBox( Format (nb, "000,000.000")  'Affiche 000 012.230
```

## V-G-4 - Pour résumer et faire très simple, retenir

ToString pour les conversions en String des variables numériques(pour afficher).

CType pour convertir tout en tout.

Le fait de convertir d'un type dans un autre s'appelle 'effectuer un cast'

## V-G-5 - Conversion Explicite et Implicite

À noter que dans cette page, on a étudié la conversion **Explicite :** elle permet de forcer la conversion vers un type à l'aide de mots-clés. C'est l'option par défaut de VB (pour le voir : menu 'Projet', 'Propriétés de …', Onglet 'Compiler').

Exemple

```
Dim d As Double = 2.65

Dim i As Integer

 i=CType(d,Integer)
```

Il existe aussi la conversion **Implicite** effectuée automatiquement sans syntaxe particulière et de manière transparente.

VB peut le permettre (Si **Option Explicit= Off** dans la configuration ).

Exemple :

```
Option Explicit Off

Dim d As Double = 2.65

Dim i As Integer

 i=d  'Pour affecter à i, un Integer, le Double d, Vb a transformé le double d en Integer.
 ' Transformation effectuée  automatiquement  et sans qu'on le voie.
```

On verra que ce mode de travail Implicite n'est pas recommandé.

## V-G-6 - Conversion restrictive, erreur



*Attention, la conversion est dite restrictive si le type final ne peut pas convertir toutes les valeurs possibles du type de départ.*

Si je convertis un Single en Integer, la partie décimale peut être tronquée, c'est une conversion restrictive.

L'inverse (conversion Short en Single par exemple) est dite étendue.

## V-G-7 - Erreur de dépassement de capacité dans les calculs

Voyons le code suivant qui semble correct :

```
Dim i As Integer = 1000000000
Dim j As Long
        j = i * 10
        MsgBox(j)
```

J'ai un grand nombre dans un integer, comme je le multiplie par 10 et que cela risque de dépasser le MaxValue dans Integer, je mets le résultat dans un Long. Pourtant il y a une erreur à l'exécution !!
Explication: quand l'expression j=i*10 est exécutée, l'expression de droite (i*10) est exécutée en premier, comme i est un Integer et '10' aussi, le résultat est mis dans un Integer (c'est là qu'il y a dépassement de capacité) puis casté en Long pour être affecté à J.

Pour éviter cela, il faut travailler directement en Long ou bien écrire 'j=i*10L': le L force 10 a être un Long et le calcul est effectué en Long.

## V-G-8 - Séparateur décimal : le point, la virgule, Culture

On rappelle aussi que le séparateur d'un **littéral** est le point (un littéral sert à donner une valeur à une variable) :

```
Dim s As Single

s= 456.67
```

Les fonctions **Val** (conversion d'une String en numérique) et **Str** (conversion d'un numérique en String), de Visual Basic, ne reconnaissent que le point (.) comme séparateur décimal.

```
Dim s As Single

s=Val ("123.4")     'est accepté, c'est 123,4 en français.
```

Les fonctions **CDbl, CType, CSng** ou **Parse** ainsi que **ToString** utilisent le séparateur des paramètres locaux de la machine . Ils reconnaissent la culture.

**Le symbole de séparateur décimal (ainsi que celui des milliers ) est donc spécifique à la culture.**

- En France, sur votre ordinateur, le séparateur décimal est la virgule.

  ```
  Dim s As Single

  s = CType("123,4", Single)

  Console.Out.WriteLine(s.ToString) 'affiche sur la console s transformé en String
  ```

  **Le symbole de séparateur décimal (ainsi que celui des milliers ) est donc spécifique à la culture.**

  Affiche: '123,4'

  **Le symbole de séparateur décimal (ainsi que celui des milliers ) est donc spécifique à la culture.**

  Par contre s = CType("123.4", Single) est refusé.

  **Le symbole de séparateur décimal (ainsi que celui des milliers ) est donc spécifique à la culture.**
- Au Usa le séparateur décimal est le point.

```
s = CType("123.4", Single) est accepté

Console.Out.WriteLine(s.ToString)

'Affiche '123.4'
```

**Le symbole de séparateur décimal (ainsi que celui des milliers ) est donc spécifique à la culture.**

On remarque donc que ToString utilise aussi le séparateur spécifique à la culture.

```
Console.Out.WriteLine(s.ToString)
```

Affiche: '123,4' en France

Lors de l'utilisation d'autres séparateurs décimaux (applications internationales, par exemple), convertissez la chaine en nombre à l'aide de la fonction CDbl ou CType CSng ou Parse.

**Pour voir quel est le séparateur en cours:**

Menu Démarrer->Paramètres->Panneau de configuration>Options régionales et linguistiques.

Obtient le séparateur décimal en fonction des paramètres locaux de la machine par du code.

```
SeparateurDécimal = NumberFormatInfo.CurrentInfo.NumberDecimaleparator
```

**On peut modifier le CultureInfo**

On peut, si on est en CultureInfo Français, afficher en mode Us.

```
Dim i As Single = 45.78

' Afficher dans la CultureInfo courante: Français

Console.WriteLine(i.ToString)    'Affiche 45,78

' Créer un CultureInfo en anglais  U.S.
Dim us As New CultureInfo("en-US")


' Afficher sur la console  en CultureInfo Us.
Console.WriteLine(i.ToString("c", us))    'Affiche 45.78
```

Il s'agit ici d'une surcharge de ToString , "c" signifie NumberFormatInfo.

## V-G-9 - IsNumeric

On utilise la fonction IsNumeric pour déterminer si le contenu d'une variable peut être évalué comme un nombre.

Exemples :

```
Dim MyVar As Object
Dim R As Boolean

MyVar = "45"
R = IsNumeric(MyVar) ' R= True.
'…
MyVar = "678.92"
R = IsNumeric(MyVar) ' R= True.
'…
MyVar = "45 kg"
```

```
R = IsNumeric(MyVar) ' R= False.
```



*'Attention le dernier exemple indique que "45 kg" n'est pas purement numérique, mais Val("45 kg") retourne 45 sans déclencher d'erreur, car Val transforme les caractères numériques à partir de la gauche, en s'arrêtant dès qu'il y a un caractère non numérique.*

## V-G-10 - Lexique anglais=>français

To Cast = Mouler, couler.

Type = Type, genre.

To parse = analyser.

## V-H - Les 'Tableaux'


C'est un beau tableau, mais en VB, ce n'est pas ça un tableau !!

**Les tableaux permettent de regrouper des données de même type.**

Les tableaux vous permettent de faire référence à un ensemble de variables par le même nom et d'utiliser un numéro, appelé **index** ou **indice**, pour les distinguer.

Comment déclarer un tableau :

```
Dim Tableau(3) As Integer
```

déclare un tableau de 4 entiers

On remarque que, dés la déclaration du tableau, le nombre d'éléments est bien défini et restera toujours le même. Après **As** on indique le type utilisé dans le tableau.

Dim Tableau(3) As Integer entraine la création des variables 'Integer' suivante :

Tableau (0)

Tableau (1)

Tableau (2)

Tableau (3)

Contenu du tableau:

| |
|---|
| 0 |
| 0 |
| 0 |
| 0 |

soit 4 éléments.

Noter que comme c'est un tableau d'entier, juste après la création du tableau les éléments sont initialisés à 0.



*Le tableau commence toujours par l'indice 0.*

*Le nombre d'éléments dans le tableau est toujours égal à l'indice de dimension + 1 (ou l'indice du dernier élément+1)*

Dim Tableau(3) comporte 4 éléments (éléments d'index 0 à 3).

Si j'exécute **Tableau(4)=5**, cela plante et me donne le message d'erreur suivant:
*L'exception System.IndexOutOfRangeException n'a pas été gérée*
*"L'index se trouve en dehors des limites du tableau."*
En effet l'élément Tableau (4) n'existe pas (Le tableau comporte 4 éléments, éléments d'index 0 à 3); l'index 4 est trop grand.
On fait parfois cette erreur quand on utilise une variable comme index dans une boucle par exemple et qu'on a mal calculé la valeur maximum de l'index de boucle.

```
Tableau(1)= 12
```

permet d'affecter le nombre 12 au 2e élément du tableau.

| |
|---|
| 0 |
| 12 |
| 0 |
| 0 |

```
Dim S As Integer
S=Tableau(1)
```

permet d'affecter à la variable S le 2e élément du tableau.

Un tableau peut avoir plusieurs dimensions :

```
Dim T(2,2)      ' 3 X 3 éléments
```

Pour un tableau à 2 dimensions, le premier argument représente les lignes, le second les colonnes.

Voyons pour chaque élément du tableau le numéro de ligne et celui de la colonne: (pas le contenu des éléments ici, mais leurs index)

| | | |
|---|---|---|
| élément:0 | élément:0 | élément:0,2 |
| élément:1 | élément:1 | élément:1,2 |
| élément:2 | élément:2 | élément:2,2 |

Exemple

La première ligne comporte les 3 éléments: T(0,0) T(0,1) et T(0,2)

Pour mettre 33 dans l'élément central :

```
Dim T(2,2) As Integer

T(1,1)=33
```

voyons le contenu du tableau :

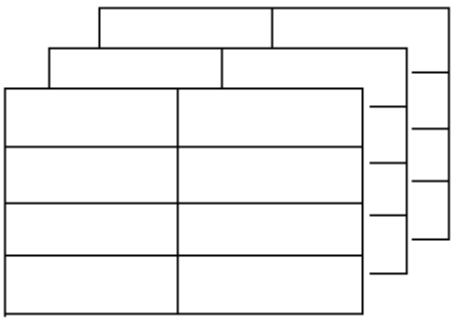
| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 33 | 0 |
| 0 | 0 | 0 |

Il est possible de créer des tableaux à 3, 4 …dimensions.

Exemple :

```
Dim T(3,1,2)
```

crée un tableau de 4X2X3 éléments.



On peut créer des tableaux de tableaux :

```
Dim T(2),(2)
```

Il a autant d'éléments que le tableau T (2,2) (mais pour l'accès à un élément, ils fonctionnent plus vite).

Il est possible de créer des tableaux avec tous les types de variables (y compris les structures).

```
Dim Mois(11) As String    'tableau de String de 12 éléments
```

Notez que dans ce cas (après la ligne Dim )les éléments contiennent Nothing, car le tableau contient des String et quand on déclare une String, elle contient Nothing au départ.

On peut initialiser un tableau (Donner une valeur aux éléments).

En effet après déclaration d'un tableau, il contient :

- la valeur 0 si c'est un tableau de numérique ;

- Nothing si c'est un tableau de String ou d'Objet.

```
Dim mois(11) As String

'mois (1) contient Nothing

mois(0)="Janvier"

mois(1)="Février"

mois(2)="Mars"
```

On peut aussi l'initialiser lors de sa déclaration :

```
Dim Mois() As String ={Janvier,Février,Mars}

' Crée un tableau de type String().
Dim winterMonths = {"December", "January", "February"}

' Crée un tableau de type Integer()
Dim numbers = {1, 2, 3, 4, 5}

'Crée un tableau de Double
Dim b = {1, 2, 3.5}

'Attention création d'un tableau d'OBJECT
Dim d = {1, "123"}

'Création de tableau à plusieurs dimensions et de tableau de tableau
    Dim e = {{1, 2, 3}, {4, 5, 6}} 'Integer(,)
    Dim f = {({1, 2, 3}), ({4, 5, 6})} 'Integer()() (jagged array)
```

On remarque ici que le nombre d'éléments n'est pas indiqué, comme on initialise 3 éléments, le tableau en aura 3. On peut même se passer d'indiquer le type (à partir du deuxième exemple), le compilateur déduit le type à partir des littéraux. On nomme cela l'inférence de type.
À part quand on utilise Linq, je pense qu'il faut mieux indiquer explicitement le type de variable.

Autre syntaxe :

```
'Déclaration
Dim t As String()

'On instancie et on initialise
t = New String(1) {"One", "Two"}
' on affecte au tableau un nouveau tableau de String contenant "One" et "Two"


Dim R(,) as Integer ={{0, 1}, {1, 2}, {0, 0}, {2, 3}}
```

Dans le premier exemple, on fait les choses en deux étapes, on déclare puis on dimensionne (instanciation) et on initialise un tableau 't'. Dans le second exemple, n déclare et on initialise en même temps un tableau à 2 dimensions, remarquez qu'on rentre les éléments 2 à 2.(Equivalent à R(0,0)=0 R(0,1)=1 R(1,0)=1 R(1,1)=2 …)

**Redim** permet de redimensionner un tableau (modifier le nombre d'éléments d'un tableau existant), si on ajoute **Preserve** les anciennes valeurs seront conservées (Array.Resize fait de même, voir plus bas).

*Attention, on ne peut pas modifier le nombre de dimensions ni le type des données. Un tableau à 2 dimensions de 20 fois 20 string pourra être redimensionné en tableau de 30 fois 30 String, mais pas en tableau d'entiers ou à 3 dimensions.*

```
Dim T(20,20) As String

…

Redim Preserve T(30,30)
```

Il est possible d'écrire Dim T( , ) As String

Dim T( , ) As String 'Sans donner les dimensions du tableau : il est déclaré, mais n'existe pas, car T(1,1)="toto" déclenche une erreur. Il faut avant de l'utiliser écrire Redim T(30,30), (sans remettre As String).

Certaines instructions, comme Split (qui découpe une String pour la mettre dans un tableau), redimensionnent elles-mêmes le tableau au nombre d'éléments nécessaires.

```
Dim Nom() as String

Nom=S.Split(Separateur)
```

**Erase** efface le tableau et récupère l'espace.

```
    Erase Tableau
```

Erase Tableau (équivalent à tableau= Nothing ).

**Clear** réinitialise le tableau (remise à 0 d'un tableau de numérique par exemple).

```
    Array.Clear(t, 2, 3)
```

Réinitialisation tableau t à partir de l'élément 1 et pour 3 éléments.

**Comment parcourir un tableau?**

Pour parcourir un à un tous les éléments d'un tableau, on utilise une boucle:

Exemple: créer un tableau de 11 éléments et mettre 0 dans le premier élément, 1 dans le second, 2 dans le troisième…

```
Dim T(10) As Integer

Dim i As Integer


For i = 0 To 10     'Pour i allant de 0 à 10

         T(i)=i

Next i
```

La variable de boucle i est utilisée pour parcourir le tableau: on utilise l'élément T( i ) donc successivement T(1) puis T(2)…et on affecte i donc 1 puis 2 puis 3…

On peut aussi utiliser For Each:( un tableau hérite de la classe System.Array)

```
Dim amis() As String = {"pierre", "jean", "jacques", "toto"}

For Each nom As String In amis

    Console.Out.WriteLine(nom)

Next
```

L'exemple affiche sur la console (menu Affichage->Fenêtre->Sortie) les noms qui sont dans le tableau.

VB alloue de l'espace mémoire pour chaque élément créé. Ne dimensionnez pas un immense tableau si vous avez besoin d'un tableau de 4*4, car cela utilise de la mémoire inutilement.

## V-H-1 - Un tableau est un objet de type Array

La Classe Array (tableau) a des propriétés et des méthodes que l'on peut utiliser.

Créons 2 tableaux et examinons les principales méthodes.

```
Dim a(3) As String

Dim b(3) As String

b=a          'Copie le tableau a dans b

b=a.copy     'Est équivalent
```



*Attention: il copie les références (l'adresse, l'endroit ou se trouve la variable) et non pas la valeur de cette variable, ce qui fait que si vous modifiez b(3), a(3) sera aussi modifié.*

Car lorsque vous assignez une variable tableau à une autre, seul le pointeur (l'adresse en mémoire) est copié. Donc en fait a et b sont le même tableau.

Pour obtenir une copie 'indépendante' dans un nouveau tableau faire :

```
b=a.clone
```

Dans ce cas si vous modifié a(2), b(2) ne sera pas modifié.

Par contre a(1)=b(1) n'affecte que l'élément a(1).

Soit un tableau Mois()

**Clear**

Array.Clear(Mois,0,2) Efface 2 éléments du tableau Mois à partir de l'élément 0.

**Reverse**

Array.Reverse(Mois, 1, 3) inverse les 3 éléments à partir de l'élément 1.

**Copy**

Array.Copy(Mois,1,Mois2,1,20) copie 20 éléments de Mois vers Mois2 à partir du 2e élément.

Array.ConstrainedCopy fait la même chose, mais annule tout si la copie n'est pas effectuée intégralement.

De même: mySourceArray.CopyTo(myTargetArray, 6) copie TOUS les éléments de la source dans la destination à partir d'un index dans la destination.

**Sort**

Array.sort(Mois) Trie le tableau Mois

Malheureusement cette méthode marche sur des tableaux unidimensionnels uniquement.

Au lieu d'utiliser un tableau à 2 dimensions (sur lequel la méthode 'Sort' ne marche pas, on peut ruser et créer 2 tableaux et surcharger la méthode sort pour trier les 2 tableaux (un servant de clé, le second d'items):

Array.Sort(myKeys, myValues) (Voir un exemple plus bas).

**Equals** compare 2 tableaux.

**Binarysearch** recherche un élément dans un tableau trié unidimensionnel.(algorithme de comparaison binaire performant sur tableau trié)

Exemple :

```
I=Array.BinarySearch(Mois, "Février") 'retourne I=1  se souvenir le premier élément est Mois(0)
```

BinarySearch effectue une recherche dichotomique : il regarde l'élément du milieu, si l'élément cherché est plus petit, il regarde l'élément du milieu du haut du tableau…

C'est rapide, mais le tableau doit être trié.

S'il trouve un élément, il retourne son index.

Si la recherche échoue, il retourne un nombre négatif, si on effectue un Not sur ce nombre retourné, on a l'index où on doit insérer l'élément.

**IndexOf**

Recherche un objet spécifié dans un tableau unidimensionnel (trié ou non), retourne l'index de la première occurrence.

```
Dim myIndex As Integer = Array.IndexOf(myArray, myString)
```

Retourne -1 si l'élément n'est pas trouvé.

LastIndexOf fait une recherche à partir de la fin.

Ici la recherche est linéaire : on compare l'élément recherché avec le premier puis le deuxième, puis le troisième élément…C'est long , mais le tableau n'a pas besoin d'être trié.

On a probablement intérêt à trier le tableau et à faire un Binarrysearch (Cela se dit, mais je ne l'ai pas vérifié).

**Ubound**

Retourne le plus grand indice disponible pour la dimension indiquée d'un tableau.

```
Dim Indice, MonTableau(10, 15, 20)
Indice = UBound(MonTableau, 1) ' Retourne 10. (1 indique la première dimension du tableau)
```

GetUpperBound même fonction.

```
Indice = MonTableau.GetUpperBound(0) '( 0 pour première dimension!!) Retourne 10.
```

Lbound existe (plus petit indice), mais est inutile, car toujours égal à 0.

**Length** retourne un entier qui représente le nombre d'éléments total dans le tableau.

Pour un tableau à une dimension Length-1 retourne l'indice du dernier élément.

Cela est souvent utilisé pour parcourir tous les éléments du tableau :

```
Dim t(10) As String

 Dim i As Integer

 For i = 0 To t.Length-1

   t(i)=…

 Next t
```

On remarque que dans un tableau multidimension Length n'est pas égale à Ubound.

GetLength(x) retourne un entier qui représente le nombre d'éléments dans la dimension x.

**GetValue** et **SetValue** permettent de connaitre ou de modifier la valeur d'un élément du tableau :

Mois.GetValue(0) est équivalent à Mois(0)

Dans un tableau à 2 dimensions, comment modifier l'élément (0,3) :

```
myArray.SetValue("fox", 0, 3)
```

C'est équivalent à myArray(0,3)="ox"

**ArraySegment** permet de définir un segment, une plage dans une Array.(framework 2).

```
Dim myArrSegMid As New ArraySegment(Of String)(myArray, 2, 5) 'ici le segment débute au second
 élément et contient 5 éléments.
```

(Si on modifie un élément de myArrSegMid cela modifie myArray, car le segment définit une plage du tableau et non un nouveau tableau).

Sur des tableaux, les actions à effectuer sont principalement:

Rechercher un élément.

Trier le tableau.

Insérer un élément.

Enlever un élément.

On a déjà évoqué cela, mais pour étudier le détail de ces algorithmes voir le chapitre 'Travail sur les tableaux et collections'.

**Pour les super pro** (débutant passe ton chemin), on peut utiliser des méthodes génériques.

Exemple recherche dans un tableau de Short nommé monTab l'élément 2.

index= Array.indexOf (Of Short)(monTab, 2) est hyper plus rapide que

index= Array.indexOf (monTab, 2), car la première version avec génériс est directement optimisée pour les Short.

Il est est de même pour Binarysearch et Sort.

Cela est valable pour les types 'valeur' (peu d'intérêts pour les strings par exemple).

## V-H-2 - Fonctions avancées sur les tableaux

Débutant s'abstenir

La méthode **Resize** permet de modifier le nombre d'éléments du tableau sans perdre le contenu :

```
Dim array As T(10)
Dim newSize As Integer=2

Array.Resize(array, newSize)
```

La méthode **Reverse** permet d'inverser les éléments d'un tableau.

```
Dim t(10) As Integer
        t(1) = 2
        Array.Reverse(t)
```

On peut aussi spécifier l'indice de début et le nombre d'éléments à inverser.

```
Dim t(10) As Integer
        t(1) = 2
        Array.Reverse(t, 2, 2)
```

À partir du Framework 2 les Arrays ont donc de nouvelles méthodes.

- **Exists**
- Le tableau contient-il des éléments qui correspondent aux conditions définies par un prédicat ?
- **TrueForAll**
- Chaque élément dans le tableau correspond-il aux conditions définies par un prédicat ?
- **Find**
- Recherche un élément qui correspond aux conditions définies par le prédicat et retourne la première occurrence.
- **FindLast**
- Idem pour la dernière occurrence.
- **FindAll**
- Récupère tous les éléments qui correspondent aux conditions définies par le prédicat.
- **ConvertAll**
- Chaque élément est passé individuellement à un Converter, et les éléments convertis sont enregistrés dans le nouveau tableau.

La syntaxe est de la forme **Array.Find(Tableau, AdresseOf Predicat)**

Un Predicat est une Sub qui retourne True si une condition est remplie.

Exemple fourni par Microsoft : on a un tableau contenant le nom d'animaux préhistoriques, le prédicat retourne True si le nom de l'animal se termine par 'saurus'. On veut savoir si la condition est remplie sur la liste au moins une fois (Exists), si tous les éléments remplissent la condition (TrueForAll), quel élément remplit la condition(Find), le premier, le dernier (FindLast), on veut récupérer dans un nouveau tableau tous les éléments qui remplissent la condition.

```
    Dim dinosaurs() As String = { "Compsognathus", _
              "Amargasaurus",   "Oviraptor",      "Velociraptor", _
              "Deinonychus",    "Dilophosaurus",  "Gallimimus", _
              "Triceratops" }

          Console.WriteLine()
          For Each dinosaur As String In dinosaurs
              Console.WriteLine(dinosaur)
          Next

          Console.WriteLine(vbLf & _
              "Array.Exists(dinosaurs, AddressOf EndsWithSaurus): {0}", _
              Array.Exists(dinosaurs, AddressOf EndsWithSaurus))

          Console.WriteLine(vbLf & _
              "Array.TrueForAll(dinosaurs, AddressOf EndsWithSaurus: {0}", _
              Array.TrueForAll(dinosaurs, AddressOf EndsWithSaurus))

          Console.WriteLine(vbLf & _
              "Array.Find(dinosaurs, AddressOf EndsWithSaurus): {0}", _
              Array.Find(dinosaurs, AddressOf EndsWithSaurus))

          Console.WriteLine(vbLf & _
              "Array.FindLast(dinosaurs, AddressOf EndsWithSaurus): {0}", _
              Array.FindLast(dinosaurs, AddressOf EndsWithSaurus))

          Console.WriteLine(vbLf & _
              "Array.FindAll(dinosaurs, AddressOf EndsWithSaurus):")
          Dim subArray() As String = _
              Array.FindAll(dinosaurs, AddressOf EndsWithSaurus)

          For Each dinosaur As String In subArray
              Console.WriteLine(dinosaur)
          Next

      End Sub


      Private Shared Function EndsWithSaurus(ByVal s As String) _
          As Boolean
      'Retourne True si la fin du mot se termine par "saurus"
          If (s.Length > 5) AndAlso _
              (s.Substring(s.Length - 6).ToLower() = "saurus") Then
              Return True
          Else
              Return False
          End If
      End Function
```

Résultat affiché :

```
'Compsognathus
'Amargasaurus
'Oviraptor
'Velociraptor
'Deinonychus
'Dilophosaurus
'Gallimimus
'Triceratops
'
'Array.Exists(dinosaurs, AddressOf EndsWithSaurus): True
```

```
'
'Array.TrueForAll(dinosaurs, AddressOf EndsWithSaurus: False
'
'Array.Find(dinosaurs, AddressOf EndsWithSaurus): Amargasaurus
'
'Array.FindLast(dinosaurs, AddressOf EndsWithSaurus): Dilophosaurus
'
'Array.FindAll(dinosaurs, AddressOf EndsWithSaurus):
'Amargasaurus
'Dilophosaurus
```

Pour Array.ConvertAll, elle retourne un tableau dont chaque élément qui vient d'un premier tableau a été modifié par une fonction.
Ici on va créer une fonction qui mettre en majuscules.

```
Private Sub Button1_Click() Handles Button1.Click
        Dim dinosaurs() As String = {"Compsognathus", _
            "Amargasaurus", "Oviraptor", "Velociraptor", _
            "Deinonychus", "Dilophosaurus", "Gallimimus", _
            "Triceratops"}
        Dim dinosaurs2() As String


        dinosaurs2 = Array.ConvertAll(dinosaurs, New Converter(Of String, String)(AddressOf
 MettreEnMajuscules))

End Sub

Private Shared Function MettreEnMajuscules(ByVal e As String) _
        As String
        Return e.ToUpper
    End Function
```

On peut aussi utiliser les expressions lambda multilignes :

```
Dim nums() As Integer = {1, 2, 3, 4, 5}

    nums = Array.FindAll(nums, Function(n)

                                   Console.WriteLine("testing " & n)
                                   Return n > 2

                               End Function)
```

**ForEach**
Exécute une action spécifiée sur chaque élément du tableau spécifié.
Syntaxe de la forme :
Array.ForEach(MyArray, Action)

Exemple : on a un tableau d'Integer, on veut afficher ces nombres et leurs carrés.
On va créer une sub ShowCarré qui reçoit un Integer et affiche le carré.
Il faut ensuite créer une 'Action', un delegate qui pointe sur la Sub.
Enfin, utiliser Array.ForEach.

```
Sub Demo
   Dim nums() = {2, 3, 5, 4}
  'Créer un delegate pour la méthode ShowCarré
    Dim action As New Action(Of Integer)(AddressOf ShowCarré)

    Array.ForEach(nums, action)

End Sub


    Private Shared Sub ShowCarré(ByVal val As Integer)
```

```
        Console.WriteLine("{0:d} Carré = {1:d}", val, val * val)
    End Sub
```

On peut utiliser une expression lambda, voir le chapitre plus loin.

```
        'Tableau
        Dim nums() = {2, 3, 5, 4}

        'Expression lamda multiligne
        'Pour chaque élément du tableau afficher  'nombre: x'
        Array.ForEach(nums, Sub(n)
                                Console.Write("Nombre: ")
                                Console.WriteLine(n)
                            End Sub)
'ou
'Expression lambda simple: affiche la série de nombres
    Array.ForEach(nums, Sub(n) Console.WriteLine(n))
```

## V-H-3 - Exemple courant d'utilisation des tableaux

Exemple détaillé

Créer un tableau de 6 éléments, mettre dans chaque élément du tableau le carré de son indice, afficher le contenu du tableau.

Cela montre l'intérêt d'utiliser une boucle pour balayer tous les éléments d'un tableau. Première boucle pour remplir le tableau, seconde boucle pour afficher.(Une boucle For …Next est ici utilisée, on verra cela plus loin.)

```
      Dim arr(5) As Integer

      Dim i As Integer

      For i = 0 To arr.GetUpperBound(0)' GetUpperBound(0) retourne 5

         arr(i) = i * i

      Next i


     For i = 0 To arr.GetUpperBound(0)

         Console.WriteLine("arr(" & i & ") = " & arr(i))

      Next i
```

Faire une boucle allant de 0 au dernier élément du tableau (For i=0 to …)

Dans chaque élément du tableau, mettre le carré de son indice (arr(i)=i*i )

Nouvelle boucle pour afficher les noms des différents éléments et leur contenu. (Console.WriteLine() affiche sur la console le nom de l'élément et son contenu)

Le programme génère la sortie suivante :

```
arr(0) = 0

arr(1) = 1

arr(2) = 4

arr(3) = 9
```

```
arr(4) = 16

arr(5) = 25
```

Exemple de recherche dans un tableau :

Dans un tableau de String rechercher dans quel élément et à quelle position se trouve la string "MN".

```
Dim Tableau() As String = {"ABCDEFG", "HIJKLMNOP"}
Dim AChercher As String = "MN"
Dim i As Integer
Dim position As Integer
For i = 0 To Tableau.Length - 1 'on parcourt chaque élément du tableau
    position = Tableau(i).IndexOf(AChercher) 'dans l'élément  du tableau on cherche la sous-
chaine
    If position >= 0 Then Exit For
Next i
```

Exemple de tri de 2 tableaux :

On crée un tableau de clés et un tableau des valeurs, à chaque clé est liée une valeur.

On trie à partir du tableau des clés myKeys , le tableau myValues est modifié pour 'suivre' le tri des clés. La Sub PrintKeysAndValues affiche les résultats.

```
Public Shared Sub Main()

' ****************Création des tableaux.
'Tableau des clé
Dim myKeys() As String = {"red", "GREEN", "YELLOW", "BLUE", "purple", "black", "orange"}
'tableau des éléments
Dim myValues() As String =
 {"strawberries", "PEARS", "LIMES", "BERRIES", "grapes", "olives", "cantaloup"}



'Affichage du tableau non trié

Console.WriteLine("Tableau non trié:")

PrintKeysAndValues(myKeys, myValues)


' Tri les éléments 1 à 3 puis affichage.

Array.Sort(myKeys, myValues, 1, 3)

Console.WriteLine("Après tri d'une partie du tableau:")

PrintKeysAndValues(myKeys, myValues)


    ' Tri la totalité du tableau.

Array.Sort(myKeys, myValues)

Console.WriteLine("Après tri de la totalité du tableau:")

PrintKeysAndValues(myKeys, myValues)

End Sub 'Fin de Main



    ' Routine affichant dans la console les clés et valeurs
```

```
Public Shared Sub PrintKeysAndValues(ByVal myKeys() As [String], ByVal myValues() As [String])

Dim i As Integer

For i = 0 To myKeys.Length - 1

    Console.WriteLine(" {0,-10}: {1}", myKeys(i), myValues(i))

Next i

Console.WriteLine()

End Sub 'PrintKeysAndValues
```

Création de tableau avec CreatInstance.

```
' Créons un tableau d'entier (Int32) comprenant 5 éléments.
Dim myArray As Array = Array.CreateInstance(GetType(Int32), 5)
Dim i As Integer
For i = myArray.GetLowerBound(0) To myArray.GetUpperBound(0)
    myArray.SetValue(i + 1, i)
Next i
```

Merci Microsoft pour les exemples.

## V-I - Les 'Collections'

Une alternative aux tableaux est l'usage de Collection.

**Les Collections permettent de regrouper des données. Les collections sont très utilisées dans la programmation 'Objet'.**

Une collection fonctionne plutôt comme un groupe d'éléments dans laquelle il est possible d'ajouter ou d'enlever un élément à n'importe quel endroit sans avoir à se préoccuper de la taille de la collection ni où se trouve l'élément.

Le nombre d'éléments n'est pas défini au départ comme dans un tableau. Dans une collection, il n'y a aucun élément au départ, puis il n'y a que les éléments que l'on a ajoutés.

Les éléments sont repérés grâce à un index ou avec une Clé unique.

Les items affichés dans une ListBox donnent une idée concrète de ce qu'est une collection.

### V-I-1 - Exemple simpliste

Soit la collection Col, au départ elle est vide.

J'ajoute des éléments (ou items) à cette collection.

Col.Add ("Toto")

Voici la collection :

| Toto |
|---|

La collection a maintenant 1 élément (on dit un **Item**).

Je fais maintenant :

Col.Add("Lulu")

Col.Add("Titi")

| Toto |
|---|
| Lulu |
| Titi |

La collection a 3 éléments maintenant, l'élément (on dit Item) 0, 1, 2.

Je fais :

Col.Remove(1) enlève le deuxième élément. (Attention on compte les éléments à partir de l'élément 0).

| Toto |
|---|
| Titi |

La collection n'a plus que 2 éléments maintenant.

On voit que le nombre d'éléments n'est pas connu à l'avance, il varie en fonction des éléments ajoutés (ou retirés)

Un élément est repéré par son indice.

Col.Item(1) contient "Titi" (le second Item de la collection)

Remarque:

J'ai pris une collection de 'Base 0': le premier élément à l'indice 0, c'est habituel dans les classes du Framework; il existe aussi des collections (celles venant de Visual Basic) de Base 1.

## V-I-2 - Classification des collections

Il est intéressant de classer les collections par fonction.

Il y a les **List**, comme dans l'exemple simpliste. On a un Index pour repérer les éléments. (Pas de clé).

| Toto |
|---|
| Lulu |
| Titi |

Il y a les **Dictionnaires**, chaque élément à une clé, on parle de Collection Clé-Valeur. On utilise la clé pour retrouver une valeur.

| Clé | Valeur |
|---|---|
| 69 | Rhone |
| 75 | Paris |
| 83 | Var |
| 1 | Ain |

Certaines collections combinent List et Dictionnaire, d'autres sont triées automatiquement.

Enfin il y a des collections particulières: les Piles, Queue, HashSet, SortedSet…

Certaines collections peuvent contenir des objets, d'autres des Strings ou des Bytes…

Certaines collections utilisent, elles, les **génériques :** elles sont faites pour contenir des génériques c'est-à-dire ce que l'on veut. Quand on utilise la Collection, on indique le type.

Du coup la collection est fortement typée : elle ne peut contenir qu'un type de donnée.

Exemple : List(Of String) est une List ne pouvant contenir que des Strings.

Voici les principales collections :

- **Les Listes :**ArrayList, List(Of…) VB 2005
- **Les Dictionnaires :**HashTable, Dictionnary
- **Les Listes-Dictionnaires :**SortedList,DictionnaryList
- **Les Queue et les Queue (Of…)**
- **Les Piles: Les Stack et les Stack (Of…)**
- **Les Listes chaînées Les LinkedList(Of….) VB 2005**
- **Gestion des ensembles: Les HashSet VB 2008**
- **Collections travaillant sur les Bits :**BitArray, BitVector32
- **Collections triées :** SortedList, SortedDictionnary SortedSet du framework 4
- **Autres : ObservableCollection**

## V-I-3 - ArrayList

Fait partie de System.Collections. C'est une Classe .Net. Il faut donc ajouter en haut du module :

```
Imports System.Collections
```

C'est une 'Liste' d'objets, d'**Item.** La ArrayList est une collection particulière. On peut y mettre des objets : chaines, nombres… rien n'empêche que le premier élément soit un entier, le second une chaine… . Il n'y a pas de clé.



*Attention le premier élément, le premier **Item**, est ici l'élément 0 (l'index va de 0 à count-1) ; c'est du .NET!!*

Exemple :

```
Dim L As New ArrayList()      'On crée une collection ArrayList

Dim L As ArrayList = ArrayList.Repeat("A", 5)
```

```
'On crée une ArrayList de 5 éléments contenant chacun  "A" (on répète "A")

L.Add("Bonjour")              'On ajoute un élément à la collection

MsgBox(L(0))                  'On affiche le premier élément
```

**L.Add()** permet d'ajouter un élément, on affiche le premier élément **L(0).**

On pourra aussi écrire **L.Item(0)** pour pointer le premier élément, en effet les éléments sont L.Item(0), L.Item(1), L.Item(2)…

```
MsgBox(L.Count.ToString)      'On affiche le nombre d'éléments.
```



*Attention c'est le nombre d'éléments. S'il y a 3 éléments dans la ArrayList ce sont les éléments d'index 0,1,2.*

```
L.Remove("Bonjour")           'On enlève l'élément de la liste qui contient "Bonjour"

L.RemoveAt(0)                 'On enlève l'élément 0 de la liste

L.Sort()                      'Trie la collection

L.Clear()                     'Efface tous les éléments

L.Contains (élément)         ' Retourne True si la liste contient élément.
```

**Insert** permet d'insérer à un index spécifié :

```
L.Insert( position, Ainserrer)
```

InsertRange insère une ArrayList dans une autre ArrayList.

```
L.Containts (élément)  ' Retourne True si la liste contient 'élément'.
```

Recherche d'un élément dans une collection NON TRIÉE avec **IndexOf :**

```
Dim l As New ArrayList

Dim i As Integer

l.Add("toto")

l.Add("lulu")

i = l.IndexOf("lulu")

MsgBox(i.ToString)    'Affiche 1 qui est l'index de "lulu"
```

On rappelle qu'il existe aussi **LastIndexOf** qui démarre par la fin et une surcharge permettant de débuter la recherche à partir d'un indice donné. Comment rechercher "lulu" à partir du 3e élément).

```
i = l.IndexOf(3,"lulu")
```

Recherche d'un élément dans une collection TRIÉE avec **BinarySearch :**

```
Dim l As New ArrayList

Dim i As Integer
```

```
l.Add("toto")

l.Add("lulu")

l.Sort()'Il est nécessaire que le tableau soit trié

i = l.BinarySearch("lulu")

MsgBox(i.ToString) 'affiche 1
```

**Pour parcourir une collection, 3 méthodes :**

*-Avec l'index de l'item*

```
For i=0 to L.Count-1

      A=L.Item(i)

Next i
```

N.B. Comme vu plus haut, on utilise Count pour trouver le nombre d'éléments, aussi la boucle doit balayer de 0 à count-1. Enfin bien se souvenir que A est un Objet, il faudra le convertir pour l'utiliser :

```
Dim s As String= CType(A,String)
```

*-Avec For Each*

```
Dim o As Objet

For Each o in L

      À=o

Next
```

Attention, A est un objet. De plus on verra que dans une boucle For Each, on ne peut pas modifier la collection.

*-Avec l'objet IEnumerator (débutant passe ton chemin)*

On crée un objet C de type IEnumerator pour parcourir la collection, cet objet a 3 propriétés :

MoveNext qui avance d'un élément dans la collection. S'il ne peut plus avancer (s'il est déjà après le dernier) il retourne False

Reset qui place l'élément courant au début, avant le premier élément (Comme au départ)

Current désigne l'élément courant.

Exemple montrant la seule manière de faire pour parcourir la collection :

```
Dim L As New ListArray

Dim C As IEnumerator= L.GetEnumerator()

While C.MoveNext())

     A=C.Current

End While
```

*Attention, si Option Explicit=On*

Les éléments de la ListArray étant des objets, on ne peut pas les affecter à une variable String par exemple, il faut écrire :

```
Str = CType(L(0), String)     'on convertit (on cast) l'objet en String.
```

Remarque :

```
L.Add(Nothing)     'est accepté: on ajoute un élément vide
```

## V-I-4 - List (Of)

À partir de 2005 on a des collections que l'on peut typer, c'est-à-dire qu'elles ne pourront contenir qu'un type de donnée, que des String, des entiers, des instances de telle classe… On parle de collections génériques. Le terme **Of** permet de définir le type de la collection.

Nécessite :

```
Imports System.Collections.Generic
```

Créons une liste ne contenant que des 'Decimal'.

```
Dim lst As New List (Of Decimal)
```

Exemple: créons une collection de String List(Of String): Elle est typée, car elle ne peut contenir que des 'String'.

```
Dim lst As New List(Of String)
```

Il s'agit d'une List avec Index.

lst(0) est le premier élément.

ou lst.item(0)

On ajoute une String :

```
lst.Add("toto")
```

Elle devient le dernier élément de la liste.

Comment affecter cet élément à une String ?

```
Dim S As String = lst.Item(0)
```

L'item est bien typé : même avec 'Option Strict=on' pas besoin de CType.

Nombre d'éléments de la list :

```
lst.Count
```

On peut à partir de vb 2010 'remplir' simplement une collection grâce à 'From' :

```
Dim names As New List(Of String) From {"Christa", "Brian", "Tim"}
```

Noter bien le New.

On peut aussi remplir avec un tableau ou ajouter une List par un AddRange :

```
Dim input() As String = { "Brachiosaurus", _
                                  "Amargasaurus", _
                                  "Mamenchisaurus" }

Dim Animals As New List(Of String)(input)

'Ajouter une list à une autre liste avec AddRange
Animals.AddRange(2, Animals)
```

La liste contient-elle "toto"?

```
Dim present As Boolean =lst.Contains("toto")
```

Present = True si la liste contient "toto".

Insérer un élément à une position donnée :

```
lst.Insert(2, "lulu")
```

Supprimer un élément à une position donnée :

```
lst.Remove("lulu")     'supprime le premier élément contenant "lulu"
lst.RemoveAt(3)        'supprime le 4e élément
lst.RemoveRange(3,2)   'supprime du 4e élément au 5e élément
```

Parcourir tous les éléments et les afficher :

```
For Each element As String In lst

    Console.WriteLine(element)

Next
```

Rechercher un élément dans la liste :

```
lst.IndexOf("lulu") 'retourne l'index de l'élément qui contient "lulu"

lst.IndexOf("lulu", 2,7) recherche à partir de l'élément 2 et sur 7 éléments.
```

Il existe aussi LastIndexOf.

Sur une list triée on utilise BinaryScearch, voir ArrayList, c'est pareil.

On peut copier une List ou partie de List dans un tableau :

```
'Avec CopyTo
Dim array(14) As String 'tableau
lst.CopyTo(array)        'copier la list dans le tableau
lst.CopyTo(array, 6)     'copier 6 éléments de la list dans le tableau
lst.CopyTo(2, array, 12, 3)  'index de départ dans list, tableau, index de départ dans array,
 nombre

'Avec GetRange
Dim output() As String = lst.GetRange(2, 3).ToArray()
```

On voit que List (Of) possède toutes les méthodes des ArrayList, mais en plus il existe des méthodes propres aux collections génériques (à partir du Framework 2) :

- **Exists**
- List contient-il des éléments qui correspondent aux conditions définies par un prédicat ?
- **TrueForAll**
- Chaque élément dans List correspond-il aux conditions définies par un prédicat ?
- **Find**
- Recherche un élément qui correspond aux conditions définies par le prédicat et retourne la première occurrence.
- **FindLast**
- Idem pour la dernière occurrence.
- **FindAll**
- Récupère tous les éléments qui correspondent aux conditions définies par le prédicat.
- **ConvertAll**
- Chaque élément est passé individuellement à un Converter, et les éléments convertis sont enregistrés dans la nouvelle collection.
- **RemoveAll**
- Efface les éléments qui correspondent au Predicat

La syntaxe est de la forme **ListeResultat= List.Find(Liste, AdresseOf Predicat)**

Un Predicat est une Fonction qui retourne True si une condition est remplie.

Exemples

Exemple 1
Avec FindAll
J'ai une list 'Animals', je veux mettre dans 'listResult' tous les éléments de Animals qui se terminent par 'us'.

On crée une liste (listResult) qui grâce à FindAll se chargera des éléments de Animals qui répondent à une condition :

```
'List de String contenant des noms d'animaux
Dim Animals As New List(Of String) From {"Compsognathus", _
            "Amargasaurus", "Oviraptor", "Velociraptor", _
            "Deinonychus", "Dilophosaurus", "Gallimimus", _
            "Triceratops"}

Dim listResult As List(Of String) = Animals.FindAll(AddressOf SeTermineParUS)
```

En argument de FindAll on a l'adresse d'une fonction: ici la fonction 'SeTermineParUS'. Pour chaque élément de Animals si SeTermineParUS retourne True, l'élément correspondant est passé dans listResult.

Voici la fonction de test, le Predicat.

```
Private Shared Function SeTermineParUS (ByVal s As String) As Boolean

If (s.Length > 2) AndAlso  (s.Substring(s.Length - 2).ToLower() = "lu") Then
     Return True
Else
    Return False
End If

End Function
```

Exemple 2
Avec ConvertAll on obtient à partir d'une première liste, une seconde liste ou chaque élément à été convertit par une fonction.
Ici on a une liste d'animaux , on va obtenir une seconde liste avec des noms courts (4 caractères).

```
'List de String contenant des noms d'animaux
Dim Animals As New List(Of String) From {"Compsognathus", _
            "Amargasaurus", "Oviraptor", "Velociraptor", _
            "Deinonychus", "Dilophosaurus", "Gallimimus", _
            "Triceratops"}
'Seconde list de String
Dim Animals2 As New List(Of String)

'Remplir Animals2 avec tous les éléments de Animals après passage par le converteur RaccourcirNom

Animals2 = Animals.ConvertAll(New Converter(Of String, String)(AddressOf RaccourcirNom))


        'Afficher la seconde list
        For Each dinosaur As String In Animals2
            Console.WriteLine(dinosaur)
        Next

   'Fonction  RaccourcirNom
    Private Shared Function RaccourcirNom(ByVal x As String) As String
       Return x.Substring(0, 4)
    End Function
```

### Trier une List

Soit une liste (Of String) de noms d'animaux, on veut trier par ordre alphabétique:
On utilise Sort.

```
Dim Animals As New List(Of String) From {"Compsognathus", _
            "Amargasaurus", "Oviraptor", "Velociraptor", _
            "Deinonychus", "Dilophosaurus", "Gallimimus", _
            "Triceratops"}



        Animals.Sort()
```

Mais on peut indiquer une Fonction qui va définir la manière de trier : ici je veux trier en fonction de la longueur des String.
Je vais indiquer l'adresse de ma fonction de comparaison comme argument de Sort:

```
Dim Animals As New List(Of String) From {"Compsognathus", _
            "Amargasaurus", "Oviraptor", "Velociraptor", _
            "Deinonychus", "Dilophosaurus", "Gallimimus", _
            "Triceratops"}



        Animals.Sort(AddressOf CompareByLength)


'Avec la fonction:
 Private Shared Function CompareByLength( _
          ByVal x As String, ByVal y As String) As Integer
                 Return x.Length.CompareTo(y.Length)
End Function
```

## V-I-5 - HashTable

C'est un '**Dictionnaire**' qui comporte des **couples clé-élément**, des **paires clé-valeur**.

Ces couples sont de type Objet-Objet.

| Clé | Valeur |
|---|---|
| 69 | Rhone |
| 75 | Paris |
| 83 | Var |
| 1 | Ain |

La clé toujours unique permet de retrouver la valeur, La clé ne doit pas être vide non plus.

Créons une Hashtable :

```
Dim H As New Hashtable
```

**H.Add(Clé,Valeur)** Ajoute un élément.

**H.Item(Clé)** Retourne l'élément correspondant à une clé.

**H.ContainsKey(Clé)** Retourne True si la Clé est dans la table.

**H.ContainsValues(Valeur)** Retourne True si la valeur est dans la table.

**H.Clear** Efface tous les éléments.

**H.Remove(Clé)** Supprime l'élément ayant une clé spécifiée.

Les collections **H.Values** et **H.Keys** contiennent les valeurs et les clés.

Exemple :

```
'  Creation d'une Hashtable.
Dim myHT As New Hashtable()

' Mettre des éléments dans la HashTable
myHT.Add("un", "premier")
myHT.Add("deux", "second")
myHT.Add("trois", "troisième")
myHT.Add("quatre", "quatrième")


'Recherche la valeur correspondant à la clé "trois"

myReponse=myHT.Item("trois")     'Retourne "troisième"


'Parcourir la HashTable

'Création d'un IDictionaryEnumerator
Dim myEnumerator As IDictionaryEnumerator = myHT.GetEnumerator()

While myEnumerator.MoveNext()

'Afficher clé et valeur
    MsgBox( myEnumerator.Key+ myEnumerator.Value)
End While
```

Attention on n'utilise pas de numéro d'index, mais uniquement la clé.

En interne, pour chaque élément, la clé est 'hachée' pour créer un code de hashage qui sert à pointer l'élément et sa valeur (voir le chapitre sur l'algorithme), ce procédé accélère la recherche de la valeur à partir de la clé.

## V-I-6 - Dictionnary (Of)

À partir de VB 2005, il y a cette collection de type **Dictionnaire (Clé-Valeur),** mais elle utilise les **génériques**.

La clé doit être unique (pas de doublon de clé).

La récupération d'une valeur à partir de sa clé est très rapide.(Utilisation d'un hachage.)

**Of** permet de choisir le type de la clé et celui des valeurs.

Créons un Dictionnary avec des clés de type String et des valeurs de type String.

```
Dim Dic As New Dictionary(Of String, String)
```

' Ajout d'élément

```
    Dic.Add("txt", "notepad.exe")

     Dic.Add("bmp", "paint.exe")
```

Depuis vb 2010 on peut ajouter rapidement des éléments :

```
Dim days = New Dictionary(Of Integer, String) From
    {{0, "Sunday"}, {1, "Monday"}}
```

' Ajout d'élément en vérifiant avant si la clé n'existe pas

```
  If Not Dic.ContainsKey("ht") Then

        Dic.Add("ht", "hypertrm.exe")

    End If
```

' Modifier la valeur correspondant à la clé 'doc'

```
Dic("doc") = "winword.exe"
```

'Parcours du Dictionary (le type de clé/Value est KeyValuePair).

```
For Each kvp As KeyValuePair(Of String, String) In Dic

     Console.WriteLine("Key = {0}, Value = {1}",  kvp.Key, kvp.Value)

Next kvp
```

' Récupérer une valeur correspondant à une clé

```
Dim tt As String = Dic("rtf")
```

'TryGetValue permet de rechercher une valeur correspondant à une clé, retourne False si la clé n'existe pas (sans déclencher d'erreur).

```
Dim value As String = ""

If Dic.TryGetValue("tif", value) Then

     Console.WriteLine("For key = ""tif"", value = {0}.", value)
```

```
 Else

     Console.WriteLine("Key = ""tif"" non trouvée.")

End If

Dic…ContainsKey("ht") permet aussi de tester si une clé existe.
```

'Enlever un élément

```
Dic.Remove("doc")
```

## V-I-7 - SortedList SortedList (Of)et SortedSet

**SortedList**
Combine List et Dictionnaire avec un tri automatique.

Il permet l'accès aux valeurs par l'intermédiaire des **clés** associées ou des **index**.

C'est un hybride de HashTable et de Array.

On ajoute un élément par **mySL.Add(Clé,Valeur).**

La séquence d'index est basée sur la séquence de tri. Quand un élément est ajouté, il est inséré dans l'ordre de tri adéquat, et l'indexation s'ajuste en conséquence. Le tri est donc automatique.

**On peut donc lire une valeur par sa Clé ou son Index**

- Quand la clé d'un élément permet d'accéder à celui-ci à l'aide de la propriété d'indexeur Item, l'élément se comporte comme Hashtable.

```
      mySL.Item(CLE) 'retourne la valeur correspondant à la clé CLE
```

**On peut donc lire une valeur par sa Clé ou son Index:**

- Quand l'index d'un élément permet d'accéder à celui-ci à l'aide de GetByIndex ou de SetByIndex, l'élément se comporte comme Array (tableau avec un Index).

```
mySL.GetKey(3) 'retourne la Clé qui est dans l'élément d'index 3

mySL.GetByIndex(3) 'retourne la valeur qui est dans l'élément d'index 3
```

**On peut donc lire une valeur par sa Clé ou son Index :**

SortedList maintient en interne deux tableaux, un tableau pour les clés et un autre pour les valeurs associées.

Index de base 0 : le premier élément est 0.

Exemple :

```
Dim mySL As New SortedList()
mySL.Add("1", "Hello")
mySL.Add("2", "World")
mySL.Add("3", "!")


Console.WriteLine(" Count: {0}", mySL.Count)    'Nombre d'éléments
Console.WriteLine(" Capacity: {0}", mySL.Capacity)
'nombre d'éléments possible,automatique, on n'a pas à s'en occuper.
```

```
Dim i As Integer
For i = 0 To mySl.Count - 1
    Console.WriteLine( myList.GetKey(i)& myList.GetByIndex(i))
     'affiche les éléments de la collection
     ' par index croissant.
Next i
```

Les **SortedList(Of…)** sont des SortedList génériques avec Clé et valeur ,triées sur la clé. Ressemble à SortedDictionary, mais occupe moins de mémoire et est moins rapide pour les insertions/suppressions.

Les **SortedDictionnary(Of…)** sont des collections génériques avec Clé et valeur ,trié sur la clé.

**Les SortedSet** (Framework 4)
Collections génériques. Un SortedSet maintient un ordre trié à mesure que les éléments sont insérés et supprimés sans que les performances en soient affectées.Les éléments dupliqués ne sont pas autorisés.

Il est possible de créer un comparer qui sera utilisé par le tri.

```
Dim mediaFiles1 As SortedSet(Of String) = New SortedSet(Of String)(New ByFileExtension)

 'on crée un comparer:
  Public Class ByFileExtension
        Implements IComparer(Of String)
……
End Class
```

## V-I-8 - Queue

Collection d'objets de type FIFO (First In, First Out)

Premier arrivé premier servi.

C'est la queue devant un cinéma, le premier arrivé, prend son billet le premier.

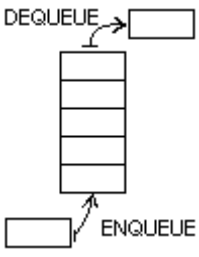
Les objets (String, Integer… ) stockés dans Queue sont insérés à une extrémité et supprimés à l'autre.

Le nombre d'éléments de la queue est géré automatiquement.

**DeQueue** supprime et retourne l'objet de début de liste.

**EnQueue** ajoute un objet en fin de liste.

**Peek** retourne l'objet de début sans le supprimer.




```
Dim myQ As New Queue()
myQ.Enqueue("One")
myQ.Enqueue("Two")
```

```
myQ.Enqueue("Tree")

Console.WriteLine ( myQ.Count) 'Affiche le nombre d'éléments.

Console.WriteLine (myQ.Dequeue())
```

Affiche le premier sorti en le sortant. "one" dans notre exemple. S'il n'y a plus d'élément cela lève une exception (une erreur) il faut donc gérer l'exception ou contrôler le nombre d'éléments avec la propriété Count.

```
If MyQ.Count>0 then

    myQ.Dequeue…

End If
```

```
Console.WriteLine (myQ.Peek())
```

Affiche le premier élément sans l'enlever de la Queue

```
myQ.Clear()
```

Efface tous les éléments de la queue.

Les **Queue(Of…)** sont des Queue mais avec un générique.

## V-I-9 - Stack

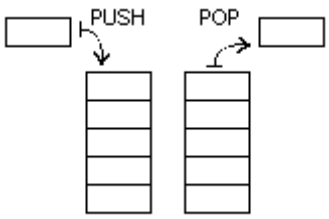
Collection d'objets de type pile ( ou stack) LIFO (Last In, First Out)

Dernier entré, premier sorti.

Ce type de stack (pile) est très utilisé en interne par les programmes informatiques: on stocke dans une stack les adresses de retour des procédures appelées, au retour on récupère l'adresse du dessus.

**Push** insère un objet en haut de la pile

**Pop** enlève et retourne un objet en haut de la pile




On peut utiliser une pile dans un programme pour gérer le déplacement de l'utilisateur dans un arbre, les éléments en cours sont stockés par Push, pour remonter en chemin inverse, on Pop.



*Attention le premier élément est ici l'élément 1 (élément d'index 1 à count)*

Exemple :

```
Dim MaPile As New Stack()

Dim Str As String

'Ajouter des éléments à la pile

MaPile.Push ("A")

MaPile.Push ("B")

MaPile.Push ("C")

'Récupérer un objet de la pile:

 Srt =MaPile.Pop()
```

Str est maintenant égal à "C"



*Attention, si Option Explicit=On, les éléments de la pile étant des objets, on ne peut pas les affecter à une variable String, il faut écrire:*

```
Str = CType(MaPile.Pop(), String)     'on convertit (cast) l'objet en String
```

Si la pile est vide et que l'on 'Pop', une exception non gérée du type 'System.InvalidOperationException' se produit. (une erreur se produit et cela plante!!), là aussi vérifier que MaPile.Count (qui indique le nombre d'éléments dans la pile) n'est pas égale à 0 avant de 'Poper'.

```
Mapile.Clear()      'Supprime tous les objets.
```

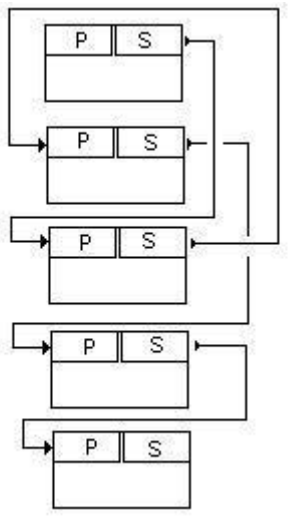
Les **Stack(Of…)** sont des track mais avec un générique.

## V-I-10 - Les LinkedList (Of)

Ce sont des **Listes chainées de générique**, chaque élément comportant une propriété **Value**(qui contient la valeur de l'élément), **Next** et **Previous**. À partir d'un élément, on peut connaitre le suivant ou le précédent.

Voir le chapitre sur les algorithmes qui explique la notion de liste chainée.

Schémas d'une liste chainée :



```
Imports System.Collections.Generic

' Création d'une list.

Dim words() As String = {"the", "fox", "jumped", "over", "the", "dog"}

' Création d'une Linkedlist.

Dim lk As New LinkedList(Of String)(words)

 Ajouter le mot 'today' au début.

 lk.AddFirst("today")

'Effacer le dernier élément.

 lk.RemoveLast()

'Les éléments sont des LinkedListNode, on peut en chercher un avec Find FindFirst, FindLast.

Dim current As LinkedListNode(Of String) = lk.FindLast("the")


'À partir de l'élément courant, on peut ajouter avant ou après.

lk.AddAfter(current, "old")  'il y a aussi AddBefore

'À partir de l'élément courant, on peut parcourir la linkedList

Dim element As LinkedListNode(Of String) = current.Previous  'il y a aussi Next

'On peut déplacer un élément

lk.AddBefore(current, element)

'On peut voir le contenu d'un LinkedListNode

current.Value

current.Previous.Value

'on peut voir la valeur du premier ou du dernier élément:

lk.First.Value

lk.Last.Value

'Il existe aussi

Containst, Count
```

## V-I-11 - HashSet (Of)

Travailler sur les ensembles.

Il s'agit d'une collection de génériques sans ordre qui contient des éléments uniques. HashSet possède comme toutes les collections Add, Remove et Contains… et fournit plusieurs opérations d'ensembles (notamment l'union, l'intersection et la différence symétrique) ce qui permet de prendre en charge la plupart des opérations mathématiques qui sont généralement réalisées sur des ensembles (sens mathématique du terme).

```
Dim hs As New HashSet(Of String)

'Ajout d'éléments:

hs.Add("toto")

hs.Add("lulu")
```

```
hs.Add("titi"
```

La méthode Add renvoie True ou False pour indiquer si elle a fonctionné (s'il n'y avait pas déjà dans la HashSet l'élément que l'on veut ajouter).

```
Dim caMarche As Boolean = hs.Add("toto")  'retourne False
```

```
hs.Count 'donne le nombre d'éléments.
```

On peut effacer un élément :

```
hs.Remove("lulu")
```

On peut effacer sous condition.

Exemple : effacer tous les éléments contenant un "t":

```
hs.RemoveWhere( Adress Of Test)
```

'La fonction Test reçoit chaque string de la table et retourne un booléen qui indique si la condition est remplie ce qui déclenche le Remove.

```
Private Shared Function Test(ByVal s As String) As Boolean

    Return (Instr(s,"t")<>0)

End Function
```

On peut ajouter la collection hs2 à hs grâce à UnionWith :

```
hs.UnionWith(hs2)
```

Les éléments doublons (qui existent déjà dans hs) ne sont pas ajoutés.

Cela correspond à un **And**.

On peut rechercher les éléments communs à hs2 et à hs grâce à IntersectWith :

```
hs.IntersectWith(hs2)
```

hs contient maintenant les éléments qui étaient présents dans hs et hs2

Cela correspond à un **Or**.

On supprime tous les éléments de hs qui sont aussi contenus dans la collection passée en paramètre (hs2) avec ExceptWith.

```
hs.ExceptWith(hs2)
```

hs contient maintenant les éléments qui n'étaient pas présents dans hs et hs2.

On peut rechercher les éléments contenus dans hs2 et dans hs, mais pas dans les 2 grâce à SymmetricExceptWith :

```
hs.SymmetricExceptWith(hs2)
```

hs contient maintenant les éléments qui étaient présents dans hs ou hs2, mais pas les deux.

On peut rechercher si hs2 est un sous-ensemble de hs grâce à IsSubsetOf :

```
Dim b As Boolean= hs.IsSubsetOf(hs2)
```

b est égal à True si hs est un sous-ensemble de hs2 (tous les éléments de hs sont dans hs2).

Il existe aussi :

IsProperSubstOf qui retourne True si hs est un sous-ensemble de hs2 et si hs différent de hs2 (sous ensemble strict).

On peut rechercher si hs2 est un surensemble de hs grâce à IsSupersetOf :

```
Dim b As Boolean= hs.IsSupersetOf(hs2)
```

b est égal à True si hs est un sur ensemble de hs2 (tous les éléments de hs2 sont dans hs).

Il existe aussi :

IsProperSupersetOf qui retourne True si hs est un sur ensemble de hs2 et si hs est différent de hs2 (sur ensemble strict).

## V-I-12 - BitArray

Crée une collection de booléens (codés sur un bit). La valeur de chaque élément est True ou False.

```
'Creation de BitArray.
Dim myBA As New BitArray(5)     'BitArray de 5 bits

Dim myBA As New BitArray(5, False) 'BitArray de 5 bits à False

Dim myBA() As Boolean = {True, True, False, False, False}
Dim myBA As New BitArray(myBytes) 'on crée un  tableau de Booleans que l'on met dans le BitArray
```

Le premier élément est l'élément 0.

On peut mettre tous les bits à True avec SetAll :

```
myBA.SetAll(True)
```

' Mettre le dernier Bit à False avec Set.

```
myBA.Set(myBA.Count - 1, False)
```

'Obtenir la valeur du second Bit.

```
myBA.Get(1)
```

myBA(1) ou myBA.Item(1) donnent aussi la valeur du second Bit.

On peut effectuer des opérations logiques entre 2 BitArray (Or, Xor, And Not).

Exemple pour Or :

```
myBA1.Or(myBA2)
```

Count et Length donnent le nombre d'éléments, mais Count est en lecture seule, Length permet, lui, de modifier le nombre d'éléments.

**La Collection BitVector32:**

Ne permet de travailler que sur 32 bits, mais est plus rapide.

Il faut avoir ajouté: Imports System.Collection.Specialized.

## V-I-13 - StringCollection

L'espace **System.Collections.Specialized** fournit ces nouveaux types de collections très spécifiques, elles ne sont faites que pour un seul type.

**StringCollection** ne peut contenir que des chaines (cela devrait aller plus vite)

```
' Créer une StringCollection.
Dim myCol As New StringCollection()

'Créer un tableau de String, l'ajouter( en fin) à la collection.
Dim myArr() As [String] = {"rouge", "vert", "orange", "vert",)
myCol.AddRange(myArr)

'Ajouter un élément à la fin de la collection
myCol.Add("marron")

'Insérer un élément à l'index 3
myCol.Insert(3, "bleue")

'Enlever un élément
myCol.Remove("orange")


' chercher et enlever tous les éléments "vert"
Dim i As Integer = myCol.IndexOf("vert")
While i > - 1
myCol.RemoveAt(i)
i = myCol.IndexOf("vert")
End While

' La collection contient-elle "jaune"?
If myCol.Contains("jaune") Then…

' Copie la collection dans un tableau.
Dim myArr2(myCol.Count) As [String]
myCol.CopyTo(myArr2, 0)

' Efface toutes les strings de la Collection.
myCol.Clear()


'Afficher la liste des Strings

Dim myEnumerator As System.Collections.IEnumerator = myCol.GetEnumerator()

While myEnumerator.MoveNext()
Console.WriteLine(" {0}", myEnumerator.Current)
End While
'C'est un peu complexe!! on y reviendra.
```

*Attention le premier élément est ici l'élément 0 (l'index va de 0 à count-1), c'est du .NET !!*

## V-I-14 - ObservableCollections, SortedSet(Of T)

Pour mémoire on se souvient qu'il existait un Type **Collection** en VB6, de Base 1, à oublier.

Par contre, à partir de VB 2008 existent les collections **ObservableCollections** qui peuvent être 'Bindées' (attachées) à des Objets visuels (comme une List ou une Grid WPF) et qui permettent la mise à jour automatique du contrôle quand on modifie la collection.

Enfin, à partir de VB 2010 existe **SortedSet(Of T)**. Un SortedSet(Of T) maintient un ordre trié à mesure que les éléments sont insérés et supprimés sans que les performances en soient affectées.Les éléments dupliqués ne sont pas autorisés.

## V-I-15 - Généralisation de la notion de collection

Certains objets ont une liste de données, d'items, Vb les organise en Collections.

Une collection peut donc faire partie des propriétés d'un objet.

Exemple

On verra plus loin qu'un contrôle nommé TextBox peut contenir du texte, ce contrôle à une collection nommée .lines qui contient les lignes de texte (s'il y en a plusieurs)

Si le texte contient 3 lignes, elles seront dans la collection 'lines'.

```
Texbox1.lines(0)     'remarquer, ici le premier élément est 0!!

Textbox1.lines(1)

Textbox1.lines(2)
```

L'indice des éléments va de 0 à count-1

Autres exemples

Les contrôles ListBox possèdent une collection 'Items' dans laquelle sont placés tous les éléments contenus dans la liste. Pour ajouter un élément, on utilise la méthode Add de la collection Items:

```
ListBox.Items.Add( )
```

Un tas d'objets possèdent des collections.

Encore plus: chaque formulaire possède une Collection 'Controls'. Il s'agit d'une collection qui contient tous les contrôles de ce formulaire.

## V-I-16 - Pourquoi le premier élément est-il 0 ou 1 ?



*Le .NET Framework normalise les collections comme étant des collections de base zéro (ArrayList par exemple). Visual Basic fournit des collections de base 1 (Comme Collection qu'il ne faut plus utiliser).*

## V-I-17 - Exemples sur les collections

Créer une ArrayList, une queue, ajouter la queue à la ArrayList, chercher un élément, insérer un élément.

Les collections font partie de l'espace de noms Systeme.Collections

```
Imports System.Collections


' Créer une ArrayList.
Dim myAL As New ArrayList()
myAL.Insert(0, "un")
myAL.Insert(1, "deux")

' Créer une Queue.
Dim myQueue As New Queue()
myQueue.Enqueue("trois")
myQueue.Enqueue("quatre")


' Copies la Queue dans ArrayList à l'index 1.
myAL.InsertRange(1, myQueue)


' Chercher "deux" et ajouter "moins de deux" avant .
myAL.Insert(myAL.IndexOf("deux"), "moins de deux")


' Ajouter "!!!" à la fin.
myAL.Insert(myAL.Count, "!!!")
```

## V-I-18 - Lexique anglais=>français

Array = tableau, table.

length= longueur.

Key= clé.

Remove (to)= enlever.

Stack= tas.

## V-J - Les 'Structures'

En règle générale, une structure est utilisée comme conteneur pour un petit jeu de variables.
Permet de regrouper des données de types différents.

(En Vb6 il y avait les types définis par l'utilisateur, ils sont remplacés par les structures.)

*Les structures sont intéressantes quand vous voulez utiliser des variables contenant plusieurs informations de différent type. Les structures sont surtout utilisées dans la programmation non 'objet'(En programmation objet, on utilisera plutôt les Collections).*

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 (au niveau Module ou Class, pas dans une procédure).

```
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 déclarée comme une adresse, elle contient donc :

un numéro qui est dans 'MonAdresse.Numero' ;

un nom de rue qui est dans 'MonAdresse.Rue' ;

un nom de ville qui est 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 (et cela va plus vite).

```
With MonAdresse

    .Rue= "Grande rue"

    .Ville= "Lyon"

End With
```

With est utilisable sur tous les objets.

Il est possible de travailler sur un **tableau de structures :**

```
Dim Adresses(99) as Adresse    'Permet de travailler sur un tableau de 100 adresses

Adresses(33).Rue="Place de la mairie"
```

On peut utiliser une variable de type structure comme paramètre d'une fonction :

```
Sub AfficheAdresse( ByVal  Une Adresse As Adresse)

… Imprimer l'adresse

End sub
```

Pour imprimer l'adresse 33, on écrira AfficheAdresse ( Adresse(33)).

## V-J-1 - Tableau dans une structure

Attention quand dans une structure il y a un tableau, il faut l'initialiser:

on veut définir une structure dans laquelle il y a 25 données DriveNumber.

On aurait tendance à écrire :

```
Public Type DriveInfo

   DriveNumber(25) As Integer 'FAUX

   DriveType As String

End Type
```

Mais cela ne fonctionne pas !!

En Visual Basic .NET il y a 2 méthodes pour utiliser un tableau dans une structure :

1-Méthode par initialize

Une structure peut comporter une méthode 'Initialize' qui sera exécutée quand on déclare une variable de type structure.

Ici, on a créé une méthode Initialize qui redimensionne le tableau interne à la structure.

```
Public Structure DriveInfo

   Dim DriveNumber() As Short

   'Noter que le nombre d'éléments a disparu.

   Dim DriveType As String

   'maintenant on instance les 25 éléments.

   Public Sub Initialize()

      ReDim DriveNumber(25)

   End Sub

End Structure
```

Exemple de routine utilisant la structure.

```
Function AddDrive(ByRef Number As Short, ByRef DriveLabel As String) As Object

   Dim Drives As DriveInfo
```

```
    Drives.Initialize()
    Drives.DriveNumber(0) = 123
    Drives.DriveType = "Fixed"
End Function
```

2-Autre manière de faire

Après la déclaration de la variable, on 'Redimensionne' le tableau.

```
Public Structure DriveInfo
    Dim DriveNumber() As Short
    Dim DriveType As String
End Structure

Function AddDrive(ByRef Number As Short, ByRef DriveLabel As String) As Object
    Dim Drives As DriveInfo
    Redim Drives.DriveNumber(25)
    Drives.DriveNumber(3)=12
    Drives.DriveType = "Fixed"
End Function
```

Si on utilise 100 variables Drives, il faut 'Redim' ou 'Initialize' le tableau pour chaque variable !!

```
Dim Drives (100) As DriveInfo

For i as Integer =0 to 100
Drives (i).Initialize                    'Dur dur!!
Next i
```

En plus si Dim Drives (100) est en tête d'un module, il faut mettre la boucle dans une procédure.

## V-J-2 - Allons plus loin

Une structure hérite de System.ValueType

### V-J-2-a - Les structures sont des types 'valeur'

Une variable d'un type structure contient directement les données de la structure, alors qu'une variable d'un type classe contient une référence aux données, ces dernières étant connues sous le nom d'objet.

Cela a de l'importance: si je crée une variable avec une structure, que je copie cette variable dans une seconde, le fait de modifier la première variable ne modifie pas la seconde.

Prenons l'exemple donné par Microsoft :

```
Structure Point
   Public x, y As Integer
   Public Sub New(x As Integer, y As Integer)
      Me.x = x
      Me.y = y
   End Sub
End Structure
```

On définit une structure Point et on définit une méthode 'New' permettant de saisir les valeurs :

'Public Sub New' est un constructeur.

Pour saisir les valeurs de x et y ont peut utiliser :

```
Dim a As Point
a.x=10
a.y=10
```

ou utiliser le constructeur :

```
Dim a As New Point(10,10)
```

En partant de la déclaration ci-dessus, le fragment de code suivant affiche la valeur 10 :

```
Dim a = new Point(10, 10)
Dim b = a
a.x = 100
Console.WriteLine(b.x)     'b est donc bien différent de a
```

L'assignation de a à b crée une copie de la valeur, et b n'est donc pas affecté par l'assignation à a.x. Si, en revanche, Point avait été déclaré comme une classe, la sortie aurait été 100 puisque a et b auraient référencé le même objet.

Enfin, les structures n'étant pas des types 'référence', il est impossible que les valeurs d'un type structure soient nulles ( elles sont égales à 0 après la création).

## V-J-2-b - Les structures peuvent contenir plein de choses

On a vu qu'elles peuvent contenir :

- des variables de différent type ;

- des tableaux ;

- des méthodes : on a vu l'exemple de Initialize et de New.

Mais aussi :

- des objets ;

- d'autres structures. ;

- des procédures ;

- des propriétés.

Exemple donné dans l'aide (et modifié par moi)

Débutant : à relire peut-être ultérieurement quand vous saurez utiliser les Classes.

Cet exemple définit une structure Employee contenant une procédure CalculBonus et une propriété Eligible.

```
Public Structure Employee
Public FirstName As String
Public LastName As String
' Friend members, accessible partout dans le programme.
Friend EmployeeNumber As Integer
Friend WorkPhone As Long
' Private members, accessible seulement dans la structure.
Private HomePhone As Long
Private Level As Integer
Public Salary As Double
Public Bonus As Double
  ' Procedure .
  Friend Sub CalculateBonus(ByVal Rate As Single)
   Bonus = Salary * CDbl(Rate)
  End Sub
' Property pour retourner l'éligibilité d'un employé.
  Friend ReadOnly Property Eligible() As Boolean
    Get
      Return Level >= 25
    End Get
  End Property
End Structure
```

Utilisons cette structure :

```
Dim ep As Employee    'Déclaration d'une variable Employee

ep.Salary = 100       'On saisit le salaire

ep.CalculateBonus(20) 'On calcule le bonus

TextBox1.Text = ep.Bonus.ToString    'On affiche le bonus
```

Cela ressemble aux Classes !! Non ?

## V-J-2-c - Portée

Vous pouvez spécifier l'accessibilité de la structure à l'aide des mots-clés : Public, Protected, Friend ou Private ou garder la valeur par défaut, Public. Vous pouvez déclarer chaque membre en spécifiant une accessibilité. Si vous utilisez l'instruction Dim sans mot-clé, l'accessibilité prend la valeur par défaut, Public.

```
Private Mastructure

    Public i As Integer

    …

End Structute
```

En conclusion les structures sont maintenant très puissantes et peuvent contenir autant de choses que les Classes , on verra cela plus loin. Mais les structures sont référencées 'par valeur' alors que les Classes le sont 'par référence'.

# V-K - Type valeur ou référence

Résumons la notion très importante de variable 'par valeur' ou 'par référence'.

Un type de données est un **type valeur** s'il contient des données dans l'espace qui lui est alloué en mémoire. Un **type référence** contient un pointeur vers un autre emplacement en mémoire contenant les données.(dixit Microsoft)

## V-K-1 - La variable 'par Valeur'

Contient réellement une valeur.

Prenons pour exemple une variable de type 'Long'.

```
Dim L As Long

L= 1456
```

'L' occupe 8 octets nécessaires pour coder un long, ici L a une valeur de 1456, donc dans ces 8 octets il est codé 1456.

**Sont des variables par 'Valeur' :**

- les Integer, les Long les Short ;
- les Single, Double, Decimal ;
- les Booleans, Char, Date ;
- les Structures ;
- les énumérations.

## V-K-2 - La variable 'par Référence'

Elles ne contiennent pas la valeur de l'objet, mais son adresse en mémoire, sa référence.

```
Dim O As Object
```

O contient l'adresse de l'objet codée sur 4 octets.

**Sont des variables par référence :**

- les objets ;
- les strings ;
- les tableaux ;
- les classes.

## V-K-3 - Influence sur l'Affectation

Posons le problème.

Travaillons sur A et B, 2 variables ayant la même 'nature'.

A existant déjà, faisonsc :

```
Dim B=A
```

**Puis modifions la valeur de A, cela modifie-t-il B ?**

Les variables par Valeur ou par Référence ne réagissent pas de la même manière.

Si le type de variable est par valeur (valable pour les entiers, les Long… les structures…), chaque variable ayant sa valeur, B n'est pas modifié.

Si le type de variable est par référence (valable pour les tableaux, les objets, les string…), chaque variable est définie par sa référence (son lieu physique); faire A=B entraine que A et B ont même référence: ils 'pointent' sur le même endroit. Si on modifie A, B est modifié, car il pointe au même endroit.

Voyons des exemples

Même si on affecte une variable par valeur à une autre, les deux variables restent différentes : elles conservent leur propre espace de stockage :

```
Dim L As Long

Dim P As Long

L=0

L=P      'on affecte P à L

P=4      'on modifie P
```

=> L=0 P=4 Modifier P n'a pas modifié L.

Par contre si on affecte une variable par référence à une autre, elle pointe toutes les 2 sur le même endroit mémoire: si j'en modifie une, cela modifie l'autre.

```
'Créons une Classe contenant un entier (Exemple à relire quand vous aurez étudié les Classes)

Class Class1
   Public Value As Integer = 0
End Class

Dim C1 As New Class1()
Dim C2 As Class1 =C1     'on crée C2, on affecte C1 à C2
C2.Value = 123           'on modifie C2
```

=> C1.Value=123 C2.Value=123 Modifier C2 a modifié C1, car elles pointent sur le même endroit mémoire.

## V-K-4 - Copie d'objet By Ref: exemple des Tableaux

Exemple sur les tableaux qui sont 'Par référence':

```
Dim A(3) As String

A(1) = "a"

Dim B(3) As String

B(1) = "b"

B = A
```

```
À(1) = "c"

Label1.Text() = B(1)     'Affiche 'c'
```

Voyons le détail.

**B = A**

Un tableau est 'par référence' et le fait de faire A=B donne la même adresse mémoire aux 2 tableaux, aussi , modifier l'un modifie l'autre. C'est ce qui se passe dans notre exemple.

**Copie élément par élément.**

Si on a déclaré 2 tableaux distincts, B(2)= À(2) affecte un élément d'un tableau à un élément d'un autre tableau, cela ne modifie que la valeur d'un élément et n'affecte pas le tableau. Aussi si on veut faire une copie 'indépendante' d'un tableau, il faut le déclarer puis avec une boucle copier chaque élément du tableau dans le nouveau.

**B= A.Clone**

B= A.Clone copie le tableau A dans B en conservant 2 tableaux distincts. Ensuite, modifier un élément du tableau ne modifie pas l'autre.

## V-K-5 - Le cas particulier des 'String'.

Elles sont 'Par référence'.



*Attention : par contre :*

```
Dim A As String

À = "a"

Dim B As String

B = "b"

B = A

À = "c"

Label1.Text() = B     'Affiche 'a'
```

Bien que les Strings soit par référence, B=À affecte simplement la valeur de A à B, si on modifie ultérieurement A, B n'est pas modifié. (Idem pour clone et copy !!) Pour une string qui est 'par référence', il parait donc impossible de la dupliquer, elle se comporte comme une variable par valeur !!

Pourquoi les String sont 'par référence' et se comportent comme si elles étaient 'par valeur'??

L'opérateur d'affectation "=" de deux strings "A=B" a simplement été défini de manière restrictive pour les strings. Les créateurs de vb .net lui ont permis uniquement une copie de la valeur de la string et non de la référence.
Il vaut mieux ne pas permettre l'affectation de la même référence pointant sur le même objet c'est dangereux pour les débutants et cela serait totalement incompatible avec les versions précédentes… Ensuite, parce que la copie de la valeur d'une string dans une autre est une opération extrêmement courante chez les programmeurs. Ce qui n'est pas le cas de l'affectation de la même référence pointant sur le même objet.

On dit que les String sont 'Immutable' (comme System.Nullable ?).

En conclusion, rien de choquant dans le fait qu'un type string se comporte comme un type par valeur : car c'est juste la définition de l'opérateur d'affectation "=" qui a été redéfinie, et c'est tout. Tout ce qui concerne l'implémentation du type string est strictement comme tous les types par référence. (Merci Sabri pour cette explication).

## V-K-6 - Déclaration avec New ?

En théorie, il faut utiliser New quand on déclare une variable 'par référence'.

Il faut écrire :

```
Dim L As Long    'un long est 'par valeur'

Dim F As New Button    'un bouton est un objet 'par référence'
```

En fait

```
Dim L As New Long  'est accepté

Dim O As Object    'est accepté, mais on a une référence vide.

Dim S As String    'est accepté.
```

Pour les Classes ou les objets graphiques, il faut par contre bien taper New pour créer l'objet :

```
Dim F As New Button
```

Si on tape Dim F As Button on crée une référence vide, mais pas d'objet Button.

## V-K-7 - Valeur après déclaration

Apres création (avant initialisation) une variable numérique 'par Valeur' contient 0,

```
Dim L As Long    'L contient 0
```

Par contre une String (par référence) qui a été créée par Dim et non initialisée contient 'Nothing'.

```
Dim S As String    'S contient Nothing: il ne pointe sur aucun objet.
```

On peut tester par **If IsNothing( O )** then… ou **If O Is Nothing**…

Pour les tableaux, bien que le tableau soit 'par Référence', c'est le type de variable utilisé dans le tableau qui décide de la valeur des éléments après déclaration.

```
Dim T(3) As Long    '=>T(0)=0
```

## V-K-8 - Comparaison

1-Une variable par Valeur peut être comparée à une autre par "=",

```
Dim L As Long=12

Dim P As Long=24
```

```
If L=P Then…
```

2-Par contre une variable par référence peut être comparée à une autre par "Is".

```
Dim O As Object

Dim Q As Object

If O Is Q then…
```

NB: pour les String '=' et 'Is' peuvent être utilisés.

3-Equals peut être utilisé pour comparer les 2 types de données :

```
Obj1.Equals(Obj2)  'Retourne True si Obj1 et Obj2 ont le même pointeur.
```

ou

```
N1.Equals(N2)     'Retourne True si la valeur de N1= la valeur de N2
```

**Pour les types 'référence', l'égalité est définie comme une égalité d'objets, c'est-à-dire si les références renvoient ou non au même espace mémoire. Pour les types valeur, l'égalité est définie comme une égalité au niveau du bit , autrement dit si la valeur est la même.**

## V-K-9 - IsReference

Il existe une instruction permettant de voir si une variable est de type 'Par référence'.

Cet exemple utilise la fonction **IsReference** pour vérifier si plusieurs variables font référence à des types référence.

```
Dim R as Boolean
Dim MyArray(3) As Boolean
Dim MyString As String
Dim MyObject As Object
Dim MyNumber As Integer

R = IsReference(MyArray) '  R= True. Tableau
R = IsReference(MyString) ' R= True. String
R = IsReference(MyObject) ' R= True. Objet
R = IsReference(MyNumber) ' R= False. Entier
```

## V-L - Variable 'Object' et autre type

Il existe un autre type de variable: le type 'Object'.

## V-L-1 - Le Type 'Object'

Parfois on ne sait pas ce que va contenir une variable : un Integer ? Un String ? Un Single ?

Pour résoudre ce problème, on utilise une variable de type 'Object'.

Cela remplace le type 'Variant' de VB6.

```
Dim myObjet As Object
```

Ensuite :

```
myObjet=12
```

est accepté, et myObjet sera considéré comme un type Integer.

```
myObjet=12.6
```

est accepté, et myObjet sera considéré comme un type Single.

```
myObjet="Visual Basic"
```

est accepté aussi, et myObjet sera considéré comme un type String.

Les 3 affectations myObjet= peuvent se suivre sans planter, l'objet contenant successivement un type Integer, Single et String.

On rappelle qu'une variable objet est une variable 'Par référence'.

On peut, suite au dernier exemple, récupérer l'objet et le transformer en String.

```
Dim maString As String

maString= CType(myObjet, String)
```

**Comment savoir quel type de variable contient la variable 'Objet' ?**

Si on fait **myObjet.GetType.ToString** cela retourne 'System.String' indiquant que myObjet contient bien une String.

myObjet.GetType.Name retourne 'String'

Pour tester si myObjet est une String, il y a une autre manière avec **TypeOf Is :**

```
If TypeOf myObjet Is String Then…


End if
```



*Attention, TypeOf Is retourne True aussi pour les Classes d'objet parent.*

```
Dim monlabel As New Label

If TypeOf monlabel Is Control Then ' est vérifié, car label dérive de control
```

monlabel est bien un Label, mais c'est aussi un Control. (On verra que tous les objet visuel comme Label dérive de la classe Control).

## V-L-1-a - Comment utiliser les propriétés d'un objet ?

Comment utiliser les membres des variables String qui sont dans un objet ?

Exemple : mettre une string dans une variable Objet, connaitre la longueur de la String.

- - Si Option strict=Off (On force VB à ne pas être trop strict !!! On verra cela plus loin)

```
Dim myObjet As New Object

myObjet="VB"

MessageBox.Show(myObjet.length)   'affiche 2
```

- - Si Option strict=On (On force VB à ne rien tolérer)

MessageBox.Show(myObjet.length) déclenche une erreur, car length n'est pas une propriété des Object.

Il faut convertir l'objet en String et là, on peut utiliser 'Length' : il faut écrire simplement :(Merci le forum de developpez.com)

```
Dim myObjet As New Object

myObjet="VB"

MessageBox.Show(DirectCast(myObjet, String).Length.ToString)
```

DirectCase transforme un type de variable en un autre, DirectCase peu'tolérant', car la variable qui reçoit doit être du bon type.

Une autre méthode consiste à transformer par CType le contenu de l'objet vers une variable String, puis à afficher la longueur de cette variable String.

```
Dim myObjet As New Object

myObjet="VB"

Dim  myString  As String

myString = CType(myObjet, String)

MessageBox.Show(myString.Length.ToString)
```

Au départ, VB ne sait pas quel type de variable sera dans l'objet, on ne connait donc pas les propriétés de la variable; la recherche de la propriété se fait à l'exécution, c'est plus long, de plus les contrôles et vérifications se font à l'exécution. , cela se nomme une liaison tardive (à éviter).

On évitera donc d'utiliser si possible des variables 'Object'.

Utilisez plutôt des variables typées (des variables String, Integer…) au départ; quand on les utilise, les contrôles et appels sont vérifiés dés le départ, on appelle cela une **liaison anticipée ou précoce**.

## V-L-1-b - Comparaison d'objets

**Is** permet de savoir si 2 variables object se rapportent à la même instance.

```
Dim o1 As New Objet = monObjet

Dim o2 As Objet

o2= o1

If o1 Is o2 Then…
```

## V-L-1-c - Nothing

Après :

```
Dim myObjet As Object
```

myObjet contient Nothing, c'est-à-dire 'Rien': pas de référence à une instance.

Après avoir utilisé myObjet=12

On peut faire myObjet=Nothing.

Lorsque vous assignez Nothing à une variable objet, cette dernière ne fait plus référence à une instance d'objet, elle ne pointe sur rien.

Si la variable avait fait référence à une instance au préalable, l'assignation de Nothing à la variable ne met pas fin à l'instance. L'instance se termine, et les ressources mémoire et système qui lui sont associés sont libérés uniquement lorsque le garbage collector (qui fait le ménage) détecte l'absence de toute référence active restante.

On peut tester si un Objet contient Nothing avec .IsNothing

## V-L-2 - Les variables d'autres types

On verra que l'interface utilisateur est composée de contrôles, ces contrôles étant des objets, et bien, on peut déclarer une variable de type 'control' : bouton, ou textbox ou formulaire.

```
Dim myButton As New Button    'crée une variable myButton de type Button
```

Ensuite on peut utiliser les membres de la classe Button.

```
myButton.BackColor
```

Autre exemple : créons un TextBox :

```
Dim myTextBox As New TextBox    'crée une variable myTextBox de type TextBox
```

## V-L-3 - Utilisez donc des variables le plus typées possible

Éviter les 'Object'. Utilisez donc des variables le plus typées possible.

Si une variable doit contenir des boutons, créer une variable de type 'Button'.

Si une variable doit être utilisée pour contenir diverses choses : Button, ListBox… plutôt que la déclarer en Objet, il est préférable de la déclarer en System.Windows.Forms.Control

```
Dim fistControl As New System.Windows.Forms.Control

fistControl= New Button
```

Plus la variable sera typée plus le code sera rapide, solide, plus on évitera les erreurs de programmation.

## V-L-4 - Attention quand on met un objet dans une variable objet

Si je mets un Button dans une variable Object.

```
Dim MyObjet As Object

MyObjet = Button1
```

MyObjet donne accès aux **propriétés des Object** (Equals, GetType, ToString…); pour utiliser les propriétés de Button (comme Text par exemple, il faut d'abord transformer l'objet en Button en écrivant : CType(MyObjet, Button).

Par exemple, pour mettre le texte du button contenu dans MyObjet dans la variable MyTexte, il faut écrire :

```
Dim MyTexte As String = CType(MyObjet, Button).Text
```

## V-M - Variable booléenne

*Mr Georges Boole 1805-1864*

Il existe un autre type de variable : le type 'Boolien 'ou 'Booléen'(Boolean).

### V-M-1 - Introduction

L'algèbre de Boole est la partie des mathématiques, de la logique de l'électronique et de l'informatique qui s'intéresse aux opérations et aux fonctions sur les variables logiques. En logique propositionnelle, une expression est soit vraie soit fausse. (le vrai (1) et le faux (0)).

**Georges Boole** (1815-1864), physicien anglais définit en 1847 une algèbre qui est applicable au raisonnement logique, qui traite des fonctions à variables binaires (deux valeurs). Mais il ne s'applique pas aux systèmes à plus de deux états d'équilibre.

Une variable booléenne, ou logique, ou binaire ne prend que deux valeurs (elle est généralement stockée sous la forme d'un bit).

Vers la fin des années 30, Claude Shannon démontra qu'à l'aide d'interrupteurs fermés pour "vrai" et ouverts pour "faux" il était possible d'effectuer des opérations logiques en associant le nombre 1 pour "vrai" et 0 pour "faux".

Ce codage de l'information est nommé base binaire. C'est avec ce codage que fonctionnent les ordinateurs. Il consiste à utiliser deux états (représentés par les chiffres 0 et 1) pour coder les informations.

Il permet d'étudier les circuits logiques.

### V-M-2 - Les booléens

On a parfois besoin de savoir si une assertion est vraie ou Fausse (**True** ou **False**).

Pour stocker une information de ce type, on utilise une variable de type booléen. Une variable de ce type ne peut contenir que True ou False.

Le terme booléen vient de "l'algèbre de Boole", cette algèbre ne travaille que sur les valeurs 1 ou 0 (True ou False)

Soit myBoolean une variable booléenne :

```
Dim myBoolean As Boolean
```

On peut écrire myBoolean = True

On peut aussi tester cette variable :

```
If myBoolean = False Then…
```

L'expression après If est évaluée, si elle est vraie 'Then' se produit.

Autre exemple montrant comment le raisonnement informatique est 'logique' :

```
If maValeur=2 Then …

End If
```

L'expression 'maValeur=2' est évaluée, si maValeur est effectivement égal à 2, l'expression prend la valeur True; dans ce cas le programme se poursuit après Then.

si maValeur est différent de 2, maValeur=2 est évaluée et prend la valeur False; dans ce cas le programme se poursuit après End If.

Un booléen peut donc prendre deux états (vrai/faux, oui/non, 1/0, etc.). Il s'agit donc d'une "valeur binaire". Techniquement, un booléen peut être représenté par un seul bit (binary digit = chiffre binaire).

Dans les langages n'ayant pas de variables booliennes, on se servait souvent d'un entier, avec pour convention que la valeur 0 représente "faux", tandis que toute valeur non nulle représente "vrai". En VB6 vrai était égale à -1. En VB.Net vrai = 1. Mais on s'en fiche, car :

*un booléen est un booléen, en VB.Net on utilise donc True ou False comme seules valeurs pour un Booléen.*

Il n'empêche que si on utilise une expression, un nombre et qu'on l'évalue comme si c'était un booléen (c'est pas bien !!), la valeur 0 représente "False", tandis que toute valeur non nulle représente "True".

## V-M-3 - Les conditions

On a vu que quand il faut faire un choix (comme dans 'If condition Then') il faut une condition qui est une expression booléenne. (avec While, Do Loop aussi).

Exemple :

```
If Condition Then    'Si 'condition' est vraie faire…

  …

End if
```

```
Do Until condition    'Boucler jusqu'à ce que 'condition'
      …
Loop

While Condition        'Tant que 'condition' boucler
…
End While
```

Pour écrire une condition, on utilise les opérateurs :

```
= égal
> supérieur à
 < inférieur à
>= supérieur ou égal
<= inférieur ou égal
<> Différent de
```

L'évaluation d'une condition donne True (Vrai) ou False (Faux), car on l'a dit c'est une expression booléenne.

Exemple :

```
      Dim valeur1 As Integer=2
      Dim valeur2 As Integer=3
      If valeur1=valeur2 Then
        …
      End if
```

valeur1 étant différent de valeur2, la condition 'valeur1=valeur2' prend la valeur False et le programme passe à la ligne après End If).

Ici le signe = n'indique pas une affectation, mais 'égale' dans une expression à évaluer.

On peut combiner les opérateurs et mettre des parenthèses :

```
If (valeurC <> valeurD )AND (valeurD =2)
```

## V-M-4 - Les opérateurs logiques

Si on a plusieurs expressions logiques, on peut les combiner avec des opérateurs logiques.

Si A et B sont des expressions booléennes :

```
À And B      retourne True si A et B sont vrais

À Or B       retourne True si une des 2 est vrai

À Xor B      retourne True si une et une seule est vrai

Not A        retourne True si A est faux et vice versa

IsNot A      À partir de VB 2005
```

On entend par expression booléenne le résultat de l'évaluation d'une condition:

c=d retourne True si c égal d et False si c différent de d.

Exemple

Si A différent de B… peut s'écrire If Not(A=B) Then…

Si A compris entre 2 et 5 peut s'écrire If A>=2 And A<=5 Then…

**Comment faire une bascule**

Il faut écrire A= Not A

A chaque fois que l'on effectue cette instruction A bascule à True s'il était à False et vice versa.

Les opérateurs **AndAlso** et **OrElse** sont plus rapides, car ils n'évaluent pas la seconde expression si ce n'est pas nécessaire.

Parfois les expressions sont complexes et on peut les simplifier en utilisant des transformations:

| Originale | Transformation |
|---|---|
| Not A And Not B | Not (À Or B) |
| Not A And B | Not (A Or Not B) |
| À And Not B | Not (Not A Or B) |
| À And B | Not (Not A Or Not B) |
| Not A Or Not B | Not (À And B) |
| Not A Or B | Not (A And Not B) |
| À Or Not B | Not (Not A And B) |
| À Or B | Not (Not A And Not B) |

Exemple pratique

Si A compris entre 2 et 5 peut s'écrire If A>=2 And A<=5 Then…

ou If Not (A<2 Or A>5) Then…

Une remarque

Avec une expression booléenne, on peut écrire :

```
Dim a As Boolean= True

If a = True Then…
```

ou

```
If a Then…
```

Exemple :

```
If (x=15)=True  Then…

ou If x=15 Then…
```

Donc, avec une expression booléenne, et uniquement avec une expression booléenne, il est possible de se passer du = True après un If, car de toute façon, l'expression est évaluée.

**IsNot** à partir de VB 2005
If Not (Objet1 Is Nothing ) Then…

devient

If Objet1 IsNot Nothing Then

Voir aussi le chapitre sur l'algèbre de Boole.

## V-N - Soyons strict et explicite (et Compare et Infer ?)

VB peut être tolérant ou pas

Option Strict=On et Option Explicit=On le rend totalement intolérant, et c'est tant mieux !! Voyons cela.

## V-N-1 - Notion de conversion Explicite et Implicite

La conversion Explicite: est permet de forcer la conversion d'un type de données vers un autre type à l'aide de mots-clés.

Exemple :

```
Dim d As Double = 2.65

Dim i As Integer

 i=CType(d,Integer)   'conversion d'une valeur double en Integer
```

Il existe aussi la conversion implicite effectuée automatiquement sans syntaxe particulière et de manière transparente.

VB peut le permettre (si Option Explicit Off dans la configuration).

Exemple :

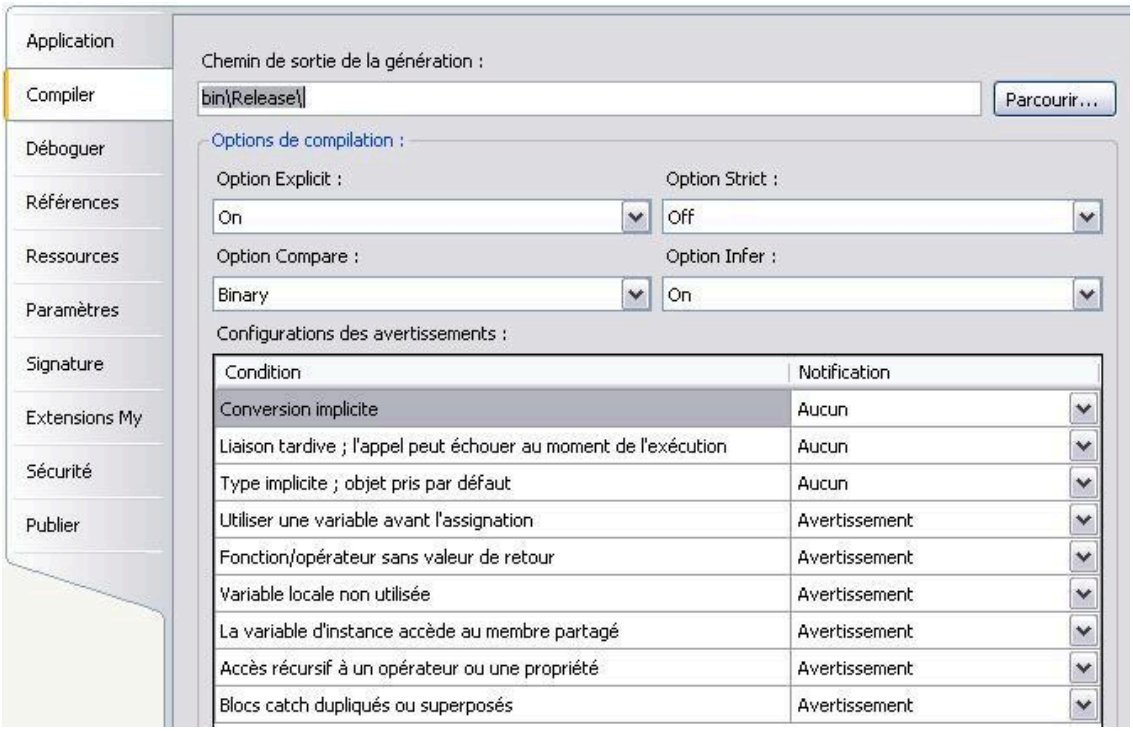
```
Dim d As Double = 2.65

Dim i As Integer

 i=d  'Pour affecter à i le Double d, Vb a transformé le double d en Integer.
```

## V-N-2 - Comment modifier une option ?

Menu Projet puis 'Propriétés de …'.

Onglet 'Compiler'

En VB 2008 :




Là on peut modifier les options de compilation.

## V-N-3 - Option Strict

### V-N-3-a - Conversions implicites

Avec Option Strict=On VB refuse les conversions implicites qui pourraient entrainer des pertes de données.

VB est naturellement très arrangeant (trop sympa !!) quand il est configuré avec Option Strict Off.

Par défaut il transforme, quand c'est possible, et si nécessaire un type de variable en un autre type.

Si je passe un nombre qui est en double précision (Double) dans une variable en simple précision (Single), VB accepte, au risque de perdre de la précision (s'il y a un très grand nombre de chiffres significatifs).

Ainsi :

```
Dim D As Double

Dim S As Single

D=0.123456789

S=D

MessageBox.Show(s) ' affiche 0,1234568
'le 9 est perdu, car un single à 7 chiffres significatifs.
```

Cela peut être ennuyeux si c'est des calculs d'astronomie !! et le programmeur ne s'en rend pas forcément compte !!

Pour éviter cela il faut activer l'OPTION STRICT à ON (elle est par défaut à Off).

Menu Projet > Propriétés de Nom de projet.

Page de propriétés de Langage VB.

Propriétés communes, génération.

En face de Option Strict, mettre On

Maintenant seules les conversions effectuées explicitement seront autorisées.

S=D est souligné dans le code pour signaler une conversion interdite.

(Par contre D=S est accepté, car on passe d'une variable à une variable plus précise)

Il faudra maintenant, pour notre exemple, écrire :

```
S= CType(D,Single)
```

Cela entraine une conversion de la valeur Double en Single; s'il y a perte de précision, elle se produit quand même, MAIS le programmeur SAIT qu'il y a conversion, il prendra ou pas EN CONNAISSANCE DE CAUSE le risque.

Avec Option Strict le langage VB.Net devient bien moins tolérant :

Écrire un programme avec Option Strict à Off, ça passe, mettre Option Strict à On un tas d'instruction coince!! même certains exemples Microsoft !! Car sans s'en rendre compte on passe d'un type de variable à l'autre sans arrêt !!

## V-N-3-b - Conversions String-numérique

Avec Option Strict=On VB refuse les conversions String-numériques implicites.

Avec Option Strict=Off

```
Dim n As Integer=12

MessageBox(n)
```

Affiche 12 : le contenu de l'entier 'n' a été transformé automatiquement en String pour être affiché.

Avec Option Strict=On

```
Dim n As Integer=12

MessageBox(n)
```

plante.

Il faut transformer explicitement n en String et écrire :

```
MessageBox(n.ToString)
```

C'est pour cela qu'il y a des '.ToString' partout !!

## V-N-3-c - Liaisons tardives

Avec Option Strict=On VB refuse les liaisons tardives :

```
Dim V As Object
V="VB"
MessageBox.Show(V.Length) 'est refusé
```

MessageBox.Show(V.Length) est refusé

Il faut écrire

```
MessageBox.Show(CType(V, String).Length.ToString)
```

Du fait que les membres utilisés avec une variable Object ne sont pas définis à l'écriture du programme (on ne sait même pas quel type de variable sera dans l'objet, on n'en connait donc pas les membres), la recherche du membre se fait à l'exécution, c'est plus long, de plus les contrôles et vérifications se font à l'exécution.

Cela se nomme une **liaison tardive**, à éviter donc.

Utilisez plutôt des variables typées (des variables String , Integer…) au départ, quand on les utilise, les contrôles et appels sont vérifiés dés le départ, on appelle cela une **liaison anticipée ou précoce**.

'Option Strict Off' permet n'importe quoi. C'est du mauvais Basic .

'Option Strict On' oblige à une grande rigueur.

## V-N-3-d - VB rapide ?

Avec Option Strict=On VB est plus rapide.

La vérification est effectuée lors de la compilation, à l'exécution il y a moins de contrôle de type.

## V-N-4 - Option Explicit

Pour la déclaration des variables, nous avions dit que toute variable utilisée devait être déclarée.

Par défaut c'est vrai.

Ouvrir Menu Projet > Propriétés de Nom de projet.

Page de propriétés de Langage VB.

Onglet 'Compiler' en VB 2008.

En face de Option Explicit, il y a On

On pourrait (c'est fortement déconseillé) mettre cette option à Off.

Cela ne rend plus obligatoire la déclaration des variables.

MaVariable=10 sans déclaration préalable est acceptée.

Cela présente certains inconvénients : si on fait une faute de frappe en tapant le nom d'une variable, VB accepte le nouveau nom et crée une nouvelle variable objet distinct.

```
Dim MaVariable          'MaVariable avec un b

MaVariabble=10          'Faute de frappe(bb)
```

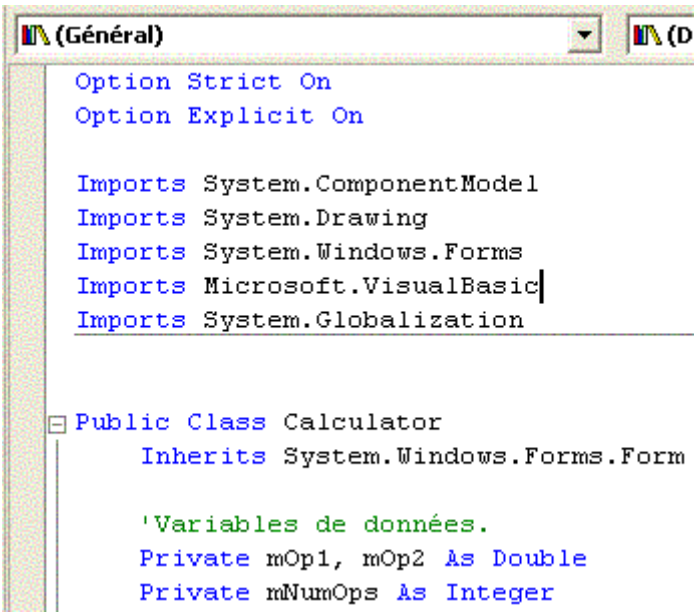
Je crois avoir mis 10 dans Mavariable. En fait j'ai mis 10 dans une nouvelle variable nommée MaVariabble

Mavariable à toujours une valeur=0

Donc, c'est clair et sans appel : laisser Option Explicit à On, ce qui oblige à déclarer toutes les variables avant de les utiliser Dans ce cas si vous tapez le nom d'une variable non déclarée, elle est soulignée en bleue.

## V-N-5 - Option strict et Explicit dans un module

On peut aussi indiquer dans un module les options, ces instructions doivent être tapées avant toutes les autres.



## V-N-6 - Option Compare

**Option Compare Binany** permet de comparer des chaines de caractères en fonction de leur code Unicode (le numéro du caractère).

**Option Compare Text** permet de comparer des chaines de caractères en fonction du CultureInfo qui prend en compte, pour chaque langue, la signification, la sémantique du caractère.

Exemple : Comparons 2 caractères, on affiche True s'ils sont égaux.

```
Console.WriteLine("a" = "A")
```

Donne True si Option Compare Text car sémantiquement parlant c'est le même caractère, du moins il a la même signification.

Donne False si Option Compare Binary, car le code Unicode de "a" et de "A" n'est pas le même.

**Avec Option Compare Binary**

Les caractères sont classés dans un ordre croissant (l'ordre de leur code Unicode)

Voyons l'ordre des certains caractères particuliers :

" " +,-./ 0123456789 :;ABCDEF abcdef èéê

On constate que l'ordre est espace puis quelques caractères spéciaux, les chiffres, les majuscules puis les minuscules, les accentués.(voir le tableau d'Unicode)

Ainsi "B" est inférieur à "a".

En utilisant Option Compare Binary, la plage [A-E] correspond à A, B, C, D et E.

**Avec Option Compare Text:**

Les caractères sont classés dans un ordre qui reflète plus la réalité d'un texte:

Tous les types de a: À, a, À, à, puis tous les types de b: B, b…

Avec Option Compare Text, [A-E] correspond à A, a, À, à, B, b, C, c, D, d, E et e. La plage ne correspond pas à Ê ou ê parce que les caractères accentués viennent après les caractères non accentués dans l'ordre de tri.

Ainsi "B" est supérieur à "a".

## V-N-7 - Option Infer

Option Infer apparait en VB 2008. Débutant passe ton chemin.

'Option Infer On' permet de se passer de donner le type d'une variable quand on la déclare. Lors de l'utilisation de la variable, elle prendra le type nécessaire. Ainsi, si on met une String dans la variable cela devient une variable String.

Par défaut on a Option Infer Off.

Exemple :

```
Option Infer On

Module Test
      Sub Main()
            ' Le type de x est ' Integer'
            Dim x = 10

            ' Le type de y est 'String'
            Dim y = "abc"
      End Sub
End Module
```

Son utilité se retrouve dans l'usage de base de données et surtout de Linq qui permet d'interroger les bases de données.

Éviter Option Infer On pour du code habituel.

## V-O - Les constantes, les énumérations

### V-O-1 - Constantes

Comme les variables, elles ont un **nom** et un **type**, mais leurs valeurs sont '**constantes**'.

On les déclare par le mot **Const**, on peut les initialiser en même temps avec =

Exemple :

```
Const NOMDUPROGRAMME= "LDF"    'constante chaine de caractères.

Const NOMBREDECASE As Integer = 12              'constante Integer
```

Ensuite on peut utiliser la constante :

```
For k= 0 To NOMBREDECASE

…

Next k
```

Si on utilise: For k=0 To 12, à la lecture c'est moins clair.

Si on écrit : NOMBREDECASE=13 cela déclenche une erreur !!

Habituellement, les constantes sont créées en début de programme.

*Il est conseillé par convention d'écrire le nom des constantes en majuscules.*

#### V-O-1-a - Intérêts des constantes ?

- Améliore la lisibilité et évite d'utiliser des constantes littérales.

Il faut éviter :

```
For i=0 To 100    'À quoi correspond 100?

…

Next i
```

Il faut écrire :

```
Const NBMAXPATIENT As Integer= 100

For i=0 To NBMAXPATIENT

…

Next i
```

- Modifications du code facilitées.
  Si une constante doit être modifiée ultérieurement, il suffit en mode conception de modifier sa valeur ce qui modifie sa valeur dans l'ensemble du code de l'application.
  Const NBMAXPATIENT As Integer= 200 'Si j'introduis une modification de valeur

```
For i=0 To NBMAXPATIENT            'Toutes les boucles utilisant NBMAXPATIENT seront à
 jour

Next i
```

- Amélioration la vitesse.

```
Const NBMAXPATIENT As Integer= 100

Dim nombre= NBMAXPATIENT
```

est plus rapide que :

```
Dim nbpatient As Integer= 100

Dim nombre= nbpatient
```

Car le compilateur code directement nombre=20 dans le premier cas.

On rappelle que seuls les types primitifs peuvent avoir des constantes (Byte, Boolean, Short, Integer, Long, Single, Double, Decimal, Date, Char, String).

## V-O-1-b - Constantes prédéfinies de VB

Les constantes de Visual Basic sont toujours là :

```
vbOk    'retourné par une MessageBox quand l'utilisateur a cliqué sur Ok.

vbBack

vbCancel

vbCrLf     'caractère numéro 13  puis numéro 10 = saut à la ligne.
```

## V-O-1-c - True False

On rappelle que True et False sont des valeurs booléennes faisant partie intégrante de VB.

Pour les anciens de VB6 ne plus utiliser -1 et 0 (d'ailleurs c'est maintenant 1 et 0).

Mais, en plus, dans Visual Basic .NET, la plupart des constantes sont remplacées par des énumérations dans le .NET Framework.

(Voir plus bas.)

*Utiliser largement ces constantes fournies par VB, cela améliore la lisibilité et la maintenance.*

## V-O-2 - Énumération

Les énumérations sont utilisées lorsque l'on a un jeu de constantes liées logiquement.

Un bloc **Enum** permet de créer une liste (une énumération) de constantes :

```
Enum TypeFichier

    DOC

    RTF

    TEXTE

End Enum
```

**Les constantes ainsi créées sont :**

- TypeFichier.DOC ;
- TypeFichier.RTF ;
- TypeFichier.TEXTE.

Le bloc Enum doit être dans l'entête du module (en haut).

C'est bien pratique, car en écrivant le code, dès que je tape 'TypeFichier.' la liste (DOC RTF TEXTE) apparait.

Ensuite, on peut utiliser dans le programme les constantes créées par exemple :

```
fichierEnCours= TypeFichier.DOC
```

On peut ensuite tester par exemple :

```
If fichierEnCours= TypeFichier.RTF then
```

Il est conseillé, par convention, d'écrire le nom des énumérations en minuscules avec la première lettre en majuscule.

Ce qui suit concernant les énumérations est un peu plus complexe.

**Chaque constante littérale de l'énumération a une valeur par défaut.**

Par défaut

```
TypeFichier.Doc =0

TypeFichier.RTF =1
```

```
TypeFichier.TEXTE=2

…
```

La première valeur est 0.

Si on ne spécifie rien, les valeurs sont des Integers.

Parfois le nom utilisé dans l'énumération (la constante littérale) est suffisant en soi et on n'utilise pas la valeur : dans un programme gérant des fichiers, une variable prendra la valeur TypeFichier.Doc pour indiquer qu'on travaille sur les fichiers .DOC. Peu importe la valeur de la constante.

Mais d'autres fois il faut que chaque constante de l'énumération possède une valeur particulière.

Je peux imposer une valeur à chaque constante de l'énumération :

```
Enum TypeFichier

    DOC=2

    RTF=4

    TEXTE=8

End Enum
```

Cela évite d'écrire fichierEnCours= 15 (en retenant que 15=fichier doc, 30= fichier rtf…)

Je peux même donner plusieurs valeurs avec And et Or à condition d'utiliser l'attribut Flags.

```
<Flags()> Enum TypeFichier

    DOC=2

    RTF=4

    TEXTE=8

    TOUS= DOC AND RTF AND TEXTE

End Enum
```

L'attribut **Flags()** indique que les valeurs sont codées en bits, ce qui permet les combinaisons de valeurs. (pour 2 le second bit est à 1, pour 4 le troisième bit est à 1, pour 8, le quatrième bit est à 1…) (voir chapitre sur l'algèbre de Boole).

Voici un exemple avec une Énumération 'Repas' ou chaque bit indique la présence d'un plat.
La propriété HasFlag permet de tester si un bit est égal à 1.

```
<Flags()> Public Enum Repas As Integer
    None = 0
    Entree = 1
    Poisson = 2
    Viande = 4
    Dessert = 8
    Boisson = 16
    Digestif = 32
End Enum


  Dim myRepas As Repas = Repas.Entree Or Repas.Dessert Or
                                    Repas.Boisson

        'Affiche myRepas
```

```
        Console.WriteLine(myRepas.ToString)
        'Teste si myRepas contient Dessert
        Console.WriteLine(myRepas.HasFlag(Repas.Dessert))

        'Sortie
        'Entree, Dessert, Boisson
        'True
```

Les énumérations sont des types qui héritent de System.Enum et qui représentent symboliquement un ensemble de valeurs. Par défaut ses valeurs sont des 'Integer', mais on peut spécifier d'autres types: Byte, Short, Integer ou Long. L'exemple suivant déclare une énumération dont le type sous-jacent est Long :

```
Enum Color As Long
 Red
 Green
 Blue
End Enum
```

Habituellement, on utilise les énumérations dans le code, comme des constantes.

Exemple :

```
Enum TypeFichier

    DOC=2

    RTF=4

    TEXTE=8

End Enum

' affecter à une variable:

Dim monFichier As TypeFichier = TypeFichier.RTF
```

On remarque qu'on crée une variable de type énumération dans laquelle on ne peut mettre qu'une énumération (en fait un Integer).

```
' affichage d'une valeur

Console.Out.WriteLine("Numéro type du fichier=" & monFichier)
```

'monFichier' affiche: 4

```
Console.Out.WriteLine("Type du fichier=" & monFichier.ToString)
```

On utilise 'monFichier.ToString' qui affiche : RTF

```
' test avec la constante de l'énumération

If (monFichier = TypeFichier.RTF) Then

    Console.Out.WriteLine("C'est du RTF")

End If
```

Affiche :"C'est du RTF"

Mais parfois on a besoin de récupérer la liste des éléments d'une énumération.

**Comment relire la liste des énumérations ?**

Il faut utiliser une méthode statique (ne nécessitant pas d'instanciation) GetValues pour obtenir toutes les constantes littérales ou valeurs d'un type énuméré que l'on passe en paramètre.

```
' liste des mentions littérales (Constantes)

For Each t As TypeFichier In [Enum].GetValues(monFichier.GetType)

    Console.Out.WriteLine(t.ToString)

Next

' liste des mentions entières (Valeurs)

For Each t As Integer In [Enum].GetValues(monFichier.GetType)

    Console.Out.WriteLine(t)

Next
```

Affiche :

```
DOC

RTF

TEXTE

2

4

8
```

**GetValues**, quand on lui donne le type de l'énumération retourne la liste des éléments de l'énumération; c'est pratique pour remplir une ListBox avec une énumération :

```
ListBox1.DataSource = [Enum].GetValues(GetType(TypeFichier))
```

Si on affecte un élément d'une énumération à une variable Integer, on récupère la valeur, si on utilise ToString on récupère la constante littérale.

```
Dim n As Integer

n = TypeFichier.RTF

Console.Out.WriteLine(n.ToString)

Dim st As String

st = TypeFichier.RTF.ToString

Console.Out.WriteLine(st)

Affiche
```

```
2

RTF
```

### Comment récupérer dans une énumération une constante à partir de sa valeur ou une valeur à partir de la constante ?

Ici il faut instancier :

```
Dim s As Type = GetType(TypeFichier)

Console.Out.WriteLine(CType([Enum].GetName(s, 15), String))

Console.Out.WriteLine(CType([Enum].Parse(s, "DOC"), String))
```

Affiche :

```
DOC

2
```

## V-O-3 - Les énumérations VB.NET

Noter que VB.Net contient, comme on l'a vu, un tas de constantes classées à l'aide d' Enum.

### V-O-3-a - ControlChars

Cette énumération contient les caractères de contrôle.

ControlChars.CrLf égale à Chr$(13)+Chr$(10) qui sert à sauter à la ligne dans une chaine de caractères.

Si on affiche "VISUAL" & ControlChars.CrLf & "BASIC"

On obtient à l'écran :

```
VISUAL

BASIC
```

ControlChars.Tab Chr$(9) = caractère de tabulation

ControlChars.NullChar Aucun caractère

ControlChars.Nothing chaine vide

ControlChars.Back

Taper ControlChars. Et comme d'habitude vous obtiendrez la liste des constantes.

### V-O-3-b - Couleurs

On peut aussi utiliser l'énumération des couleurs définies par le Framework

```
System.Drawing.Color.Blue    'Pour le bleu
```

ou en simplifiant (si Imports System.Drawing a été écrit)

```
Color.Chocolate

Color.Black

...
```

## V-O-3-c - Math

Si Import System.Math est présent en haut du module,

**PI** contient 3,14…

**E** contient la base log naturel

## V-O-3-d - Touche du clavier dans le Framework

Il est parfois nécessaire de savoir si une touche précise a été tapée par l'utilisateur au clavier, pour cela il faut connaitre les touches, mais pas besoin de se souvenir du code des touches, il suffit de taper Keys, et la liste des touches s'affiche. Cliquer sur le nom de la touche recherchée et vous obtenez la constante correspondant à la touche :

```
Keys.Right    'Désigne le code de la touche '->'

Keys.D8       'Désigne le code de la touche '8'

Keys.Delete   'Désigne le code de la touche 'Suppr'

Keys.D   'Désigne le code de la touche 'D'

Keys.Shift   'Désigne le code de la touche 'Majuscule'

Keys.SnapShot   'Désigne le code de la touche 'Impression écran'
```

## V-O-3-e - Autre exemple

Quand on ferme une MessageBox (une fenêtre qui affiche un message), cela retourne une valeur qui contient :

```
MsgBoxResult.Yes

MsgBoxResult.No    ou

MsgBoxResult.Cancel
```

En fonction du bouton qu'a utilisé l'utilisateur pour sortir de la fenêtre MessageBox (appuie sur les boutons Oui, Non, Cancel).

# V-P - Les opérateurs

*+-*/And OrMod&*

Pour travailler sur les variables, on utilise des opérateurs (addition, soustraction…).

## V-P-1 - Addition : +

Dans le cas de variables numériques.

```
    Dim A,B, C As Integer
                B=2
                C=3
```

```
A=B+C
```

si B=2 et C=3 => A=5

On peut écrire :

A=A+1

Dans ce cas, on affecte à la variable A son ancienne valeur +1, si A=2 au départ, A=3 ensuite.

A+=1 est équivalent à A=A+1

Cela incrémente la variable A.

On peut utiliser '+' pour ajouter une string à une autre, il est préférable d'utiliser '&'.

## V-P-2 - Soustraction : -

B=C-D

A-=1 est équivalent à A=A-1

## V-P-3 - Multiplication : *

C'est une étoile : *

B= C*D

## V-P-4 - Division : /

On remarque que ":" n'est pas l'opérateur de division. (Ce signe sert de séparateur quand plusieurs instructions sont sur la même ligne.)

Retourne le quotient complet qui conserve le reste dans la partie fractionnaire.

B=C/D

Si C=10 et D=3 B=3.33333333333333

La division de 2 Single retourne un Single.
La division de 2 Doubles retourne un Double.
La division de 2 Decimal retourne un Decimal.
Voir en bas de page, des informations complémentaires, car
La division de 2 entiers (Integer…) retourne un Double.

## V-P-5 - Division entière : \

Si A=10\3 => A=3 'on perd le reste

Voir en bas de page, des informations complémentaires, car "\"sur 2 Integer retourne un Integer.

## V-P-6 - Puissance : ^

```
À=B^3          'À=B*B*B
```

## V-P-7 - Modulo : Mod

C'est le reste de la division par un nombre :

10 Mod 3 donne 1

Exemple A est-il multiple de 3 ?

Si A Mod 3 = 0 , A est un multiple de 3

```
If A Mod 3 = 0 then…
```

## V-P-8 - Concaténation : &

C'est une mise bout à bout des chaines de caractères.

Si

A= "VISUAL"

B= " "

C= "BASIC"

D=A & B & C donne D="VISUAL BASIC"

Le signe + peut aussi être utilisé, mais il est plutôt réservé aux additions de variables numériques.

&= permet aussi la concaténation A&=B est équivalent à A= À&B

## V-P-9 - Priorités

*L'ordre des calculs se fait en fonction de la priorité des opérateurs.*

**S'il y a plusieurs opérateurs, '^' a la priorité la plus forte puis * et / puis + et -**

Cela veut dire que VB effectue les élévations à puissance puis les multiplications et divisions puis les additions et soustractions.

Pour être complet, voyons les priorités par ordre décroissant :

```
^ élévation à la puissance

- négation unaire

/ et * multiplication et division

\ division entière
```

```
mod  modulo

+ et -  addition et soustraction.
```

Exemple 2+3^3 donne 29, car VB effectue (3^3)+2 et non pas 125 (2+3)^3

**S'il y a plusieurs opérateurs de même priorité, l'ordre des calculs se fait de gauche à droite.**

***Pour éviter toute faute d'interprétation, utiliser des parenthèses.***

2+(3^3) lève toute ambiguïté.

## V-P-10 - Comparaison

```
= égal

 > supérieur à

 < inférieur à

>= supérieur ou égal

<= inférieur ou égal

<> Différent de
```

Le résultat d'une comparaison est True (Vrai) ou False (Faux)

Exemple :

```
      Dim A As Integer=2

      Dim B As Integer=3

      If A=B then

        …

      End If
```

À étant différent de B, A=B prend la valeur False et le programme passe à la ligne en dessous de End If (pas après then).

Ici le signe = n'indique pas une affectation, mais une expression à évaluer.

Ici aussi on peut combiner les opérateurs et mettre des parenthèses :

```
R= (C<>D)AND (D=2)    'Si C différent de D  et si D égal 2
```

**Comparaison de chaines de caractères**

Les chaines de caractères sont comparées en fonction du tri alphabétique.

Par défaut, 'Option Compare Binary' est activé, ce qui fait que l'ordre des caractères est en relation avec leur code Unicode (voir chapitre sur les Options).

```
' À<B<C&#8230;&#8230;<Y<Z<a<b<c&#8230;&#8230;y<z<à<é…

      Dim A As String="A"

      Dim B As String="Z"

      If A<B then…
```

À est bien inférieur à B, donc A<B prend la valeur True et le programme saute après Then.

La casse (majuscules ou minuscule) est différenciée.

Si on veut comparer sans tenir compte du fait que c'est en majuscules ou minuscules, il faut d'abord transformer les 2 chaines en minuscules par exemple.

On veut comparer A= "aaa" et B= "AAA"

Normalement A est différent de B :

A=B retourne False

Par contre A.ToLower=B.ToLower retourne True (Vraie)

En utilisant 'Option Compare Text' en début de module, on ne différencie plus la casse: "A" devient égal à "a".

## V-P-11 - Logique : Not And Or ElseOr Xor

### V-P-11-a - Si A et B sont des expressions booléennes

**A And B** retourne True si A et B sont vrais

**A Or B** retourne True si une des 2 est vrai

**A Xor B** retourne True si une et une seule est vrai

**Not A** retourne True si A était faux et vice versa

On entend par expression booléenne le résultat de l'évaluation d'une condition:

A=B retourne True si A=B et False si A différent de B.

**Exemple**

Si A différent de B… peut s'écrire **IF Not(A=B)…**

Si A compris entre 2 et 5 peut s'écrire **If A>=2 And A<=5…**

**Comment faire une bascule**

Il faut écrire **A= Not A**

À chaque fois que l'on effectue cette instruction A bascule à True s'il était à False et vice versa.

## V-P-11-b - Si A et B sont des nombres (Integer par exemple)

L'opération est effectuée sur chaque bit.

A = 7 'en décimal ( 0111 en binaire)

B = 12 'en décimal( 1100 en binaire)

Que donne A And B ?

On effectue l'opération bit à bit.

Pour le premier bit de chaque nombre 0 And 1 = 0.

Pour le second bit de chaque nombre 1 And 1 = 1…

Cela donne 0100.

A And B = 4 'en décimal( 0100 en binaire)

Autre exemple

A And 1 indique si le bit le moins significatif (le plus à droite) est a 1

Exemple : si A = 7 'en décimal ( 0111 en binaire) A And 1 retourne 1

## V-P-11-c - Les opérateurs And, Or et Xor sont évalués en fonction du type des opérandes

### V-P-11-c-i - Pour le type Boolean

Une opération And logique est effectuée sur les deux opérandes.

Une opération Or logique est effectuée sur les deux opérandes.

Une opération Or exclusif logique est effectuée sur les deux opérandes.

### V-P-11-c-ii - Pour les types Byte, Short, Integer, Long et tous les types énumérés

L'opération spécifiée est réalisée sur chaque bit de la représentation binaire des deux opérandes

And : Le bit de résultat est 1 si les deux bits sont 1. Sinon, le résultat est 0.

Or : Le bit de résultat est 1 si l'un des deux bits est 1. Sinon, le résultat est 0.

Xor : Le bit de résultat est 1 si l'un des deux bits est 1, mais pas les deux. Sinon, le bit de résultat est 0 (c'est-à-dire 1 Xor 0 = 1, 1 Xor 1 = 0).

Les opérateurs AndAlso et OrElse sont uniquement définis sur le type booléen, ils sont plus rapides, car ils n'évaluent pas la seconde expression si ce n'est pas nécessaire.

Il n'est pas judicieux d'effectuer des opérations logiques sur des Single, Decimal, Double (nombre avec virgule).

## V-P-12 - Déplacement de bits

Les opérateurs binaires << et >> effectuent des opérations de déplacement de bits. Ces opérateurs sont définis pour les types Byte, Short, Integer et Long.

L'opérateur << décale à gauche les bits du premier opérande du nombre de positions spécifié. Les bits de poids fort situés en dehors de la plage du type de résultat sont éliminés, et les positions libérées par les bits de poids faible sont remplies par des zéros.

L'opérateur >> décale à droite les bits du premier opérande du nombre de positions spécifié. Les bits de poids faible sont éliminés et, si l'opérande de gauche est positif, les positions libérées par les bits de poids fort sont mises à zéro ; s'il est négatif, elles sont mises à un. Si l'opérande de gauche est de type Byte, les bits de poids fort disponibles sont remplis par des zéros.

À quoi cela sert ?

Exemple décaler à gauche un Byte revient à faire une multiplication par 2.

3 en décimal= 11

Je décale à gauche, j'obtiens 110 , c'est 6 en décimal.

## V-P-13 - Remarque 1 : Allons plus loin avec / et \

La division de 2 Single avec "/" retourne un Single.
La division de 2 Decimal avec "/" retourne un décimal.
mais
La division de 2 entiers avec "/" retourne un double.

Avec Option Strict=Off

```
Dim i As Integer = 4

Dim j As Integer = 2

Dim k As Integer = i / j 'est accepté
'car i/j donne un double transformé en Integer 'automatiquement'.
```

Avec Option Strict=On, il faut écrire :

```
Dim i As Integer = 4

Dim j As Integer = 2

Dim k As Integer = CType(i / j, Integer) 'on est obligé de caster le double en Integer.
```

Mais "\" retourne un Integer si on divise 2 entiers.
Pour diviser 2 entiers, utiliser donc "\".

```
Dim i As Integer = 4

Dim j As Integer = 2

Dim k As Integer = i \ j 'est accepté
'même si  Option Strict=On
```

## V-P-14 - Remarque 2 : Division par zéro

**La division par zéro est impossible mathématiquement.**

- Si on divise un double (ou un Single ) par zéro, on obtient **NaN :** nombre non défini ou **PositiveInfinity** ou **NegativeInfitiny** selon le dividende.
- Pour les entiers, Integer , Byte… une division par zéro déclenche une erreur ( on dit une exception) **DivideByZeroException** ou **OverflowException** en vb 2010.

En pratique les choses ne sont pas si évidentes, voyons des exemples :

```
Dim i As Integer = 4

Dim j As Integer = 0

TextBox1.Text = (i/j).ToString 'Affiche " +Infini"
```

Le résultat de l'opération (i/j) qui est un Double prend la valeur infini et est directement affiché.

Par contre :

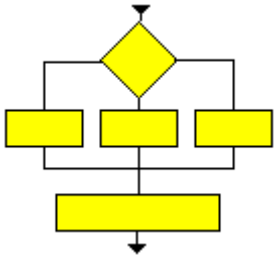
```
Dim i As Integer = 4

Dim j As Integer = 0

Dim k As Integer = CType(i / j, Integer) 'Erreur
```

Retourne une exception (une erreur) Overflow.Exception, car le résultat de l'opération est l'infini donc plus grand que MaxValue des Integers.

Il faut donc, si on risque d'avoir la valeur zéro, faire un contrôle avant la division :

```
If j<>0 Then

    k=i/j

End If
```

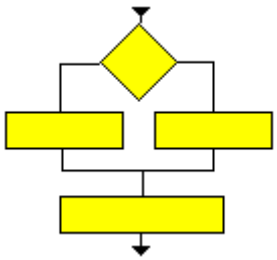
## V-Q - Les structures de contrôle : Choix et boucles



Elles permettent de gérer le déroulement du code.

## V-Q-1 - If Then

Permet de créer une **structure décisionnelle :**



If Condition Then

End if

Si la Condition est vraie alors…

Une instruction (ou un bloc d'instructions) peut être exécutée si une condition est vraie.

Exemple :

```
If A=B then

    MsgBox("A=B")

End If
```

Si A = B alors on exécute le bloc de code entre Then et End If, il affiche dans une fenêtre MessageBox "A=B"

Noter que, si on le désire, on peut écrire sur la même ligne après Then (Pas besoin de End If).

```
If A=B Then MsgBox("A=B")
```

**On peut tester une condition fausse** et dans ce cas utiliser **Not**.

```
If Not A=B Then MsgBox("A différent de B")
```

Si A et B sont différents (Not A=B signifie NON égaux) afficher "A différent de B".

(On aurait pu écrire If A<>B Then…)

Il peut y avoir des **opérateurs logiques dans la condition :**

```
If A=B And C=D then…    'Si A égal B et si C égal D
```

Autre exemple :

```
Dim n As String ="33" ' on crée une String , on y met les caractères "33"

If Not IsNumeric(n) then

            MsgBox ("n n'est pas un nombre")

            Exit Sub

End if
```

Si n n'est pas numérique alors afficher dans une boite de dialogue: "n n'est pas un nombre" puis quitter la procédure (Exit Sub).

Noter bien que comme il y a plusieurs instructions après Then on crée un bloc d'instruction de plusieurs lignes entre Then et End If.

### Simplification d'écriture

Au lieu de

```
If Condition = True Then

End if
```

On peut écrire:

```
If Condition Then

End if
```

Condition étant de toute manière évaluée pour voir si elle est égale à True.

On peut aussi utiliser la structure '**Si…Alors…Sinon**' :

```
If condition then

… 'effectué si condition vraie

…

Else

      …'effectué si condition fausse

      …

End if
```

Exemple :

```
If A=B then

    MsgBox("A=B")

Else

    MsgBox("A différend de B")

End If
```

Des structures If Then peuvent être **imbriquées :**

```
If…

      If…

      …

      Else

            If…

     …

End if
```

```
      End if

End If
```

Pour bien repérer les différents niveaux, utiliser les tabulations et décaler le 'If then' et son code au même niveau.

Pour vérifier s'il n'y a pas d'erreur, compter les 'If', il doit y en avoir autant que des 'End If'. VB souligne le 'If' si il n'y a pas de 'End if'.

**Dernière syntaxe avec 'ElseIf' :**

```
If Condition1 Then

    …

ElseIf condition2 Then

    …

ElseIf condition3 Then

    …

end if
```

Si condition1…

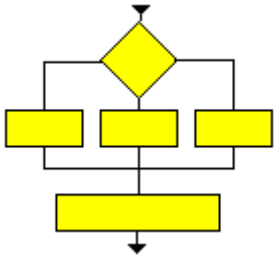
Sinon si condition2

Sinon si condition3

Fin Si

## V-Q-2 - Select Case

Créer une structure décisionnelle permettant d'exécuter un grand nombre de blocs de code différents en fonction de la valeur d'une expression :



```
Select Case expression

Case valeur1

      'code effectué si expression=valeur1


Case valeur2

      'code effectué si expression=valeur2


Case valeur3
```

```
      'code effectué si expression=valeur3

…

Case Else

      'code effectué dans tous les autres cas


End Select
```



*Attention si expression=valeur1 le code entre Case Valeur1 et Case valeur2 (et uniquement celui-là) est effectué, puis l'exécution saute après End Select.*

Exemple d'un code affichant le jour de la semaine :

J est un entier contenant le numéro d'ordre du jour

```
Select Case N

Case 1                        'Si N=1

      MsgBox "Lundi"          'Afficher 'Lundi'


Case 2

      MsgBox "Mardi"


Case 3

      MsgBox "Mercredi"

…

…

Case Else

      MsgBox "Nombre pas entre 1 et 7"

End select
```

Nous venons d'utiliser une expression simple après chaque Case, mais on peut utiliser des expressions plus complexes.

Plusieurs clauses d'expression peuvent être séparées par des **virgules**.

```
Select Case N

Case 8,9,10

      'Effectuer le code  si N=8 ou N=9 ou N=10
```

Le mot-clé **To** permet de définir les limites d'une plage de valeurs correspondantes pour N.

```
Select Case N

Case 8 To 20
      'Effectuer le code  si N est dans la plage 8 à 20
```

```
End Select
```

Le mot-clé **Is** associé à un opérateur de comparaison (=, <>, <, <=, > ou >=) permet de spécifier une restriction sur les valeurs correspondantes de l'expression. Si le mot-clé Is n'est pas indiqué, il est automatiquement inséré.

```
Select Case N

Case Is >= 5

      'Effectuer le code  si N supérieur ou égal à 5.

End Select
```

Vous pouvez utiliser plusieurs expressions ou plages dans chaque clause Case (séparées par des virgules). Par exemple, la ligne suivante est valide :

```
Case 1 To 4, 7 To 9, 11, 13, Is > MaxNumber
```

Vous pouvez aussi indiquer des plages et des expressions multiples pour des chaines de caractères. Dans l'exemple suivant, Case correspond aux chaines qui sont absolument identiques à "aaa", aux chaines comprises entre «ccc» et «ddd» dans l'ordre alphabétique, ainsi qu'à la valeur de Var :

```
Case "aaa", "ccc" To "ddd", Var
```

Pour les 'Pro':

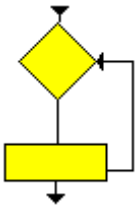
Les "Case" peuvent contenir n'importe quelle expression. Aussi il est possible de tester dans les conditions, non pas les valeurs d'une même variable, mais diverses fonctions totalement indépendantes ou comme ici des fonctions travaillant toutes sur une même variable. C'est un usage méconnu du Select Case qui clarifie l'écriture et qui évite de multiples If Then ou Goto.

Ici une routine reçoit une String contenant un nom de fichier, elle teste si le nom n'est pas vide puis si le fichier existe…

```
Sub TestFichier (File As String)

Select Case true

        Case len(File) = 0

            msgbox "Pas de nom de fichier"

        Case Not Exit(File)

            errorState=File.NotExist

        Case Not Open(File)

            errorState=File.NotOpen

        Case Not Overwrite(File)

            errorState=File.NotOverwrite

        Case else

        'placer les exceptions ici

End Select

End Sub
```

## V-Q-3 - For Next

Permet de faire des boucles.



Les boucles sont très utilisées pour parcourir une plage de valeur qui permet par exemple de parcourir tous les éléments d'un tableau ou pour effectuer de manière itérative un calcul.

Le nombre de boucles va être déterminé par une variable qui sert de compteur : la variable de boucle.

**Le nombre d'exécutions est déterminé au départ de la boucle,** car le compteur a une valeur de départ, une valeur d'arrêt.

Pour variable allant de 'début' à 'fin'

Boucler

donne en VB

```
For variable=début To fin

…

Next variable
```

Exemple :

```
Dim i as Integer

For i=1 to 10

MsgBox (i.toString)

Next i
```

En langage courant : pour i allant de 1 à 10, afficher la valeur de i dans une MessageBox.

La variable compteur va prendre successivement les valeurs 1 puis 2 puis 3…… jusqu'à 10 et effectuer à chaque fois le code qui est entre For et Next.

Si on décompose :

i=1 Affiche "1", arrivé à Next, remonte à For, i =2 , affiche "2"……

… i=10 , affiche "10" poursuit après Next.

En effet i augmente d'une unité à chaque 'tour'.

Il peut y avoir un pas (**Step**), le compteur s'incrémente de la valeur du pas à chaque boucle.

```
Dim i as Integer

For i=1 to 10 Step 2

MsgBox i.toString

 Next i
```

Affiche 1 puis 3 puis 5 puis 7 puis 9



*Attention si la valeur de sortie de boucle est inférieure à celle d'entrée il faut indiquer un pas négatif.*

```
Dim i as integer

For i=10 to 1 Step -2

MsgBox (i.toString)

 Next i
```

Affiche 10 puis 8 puis 6 puis 4 puis 2

Bien sûr on peut utiliser des expressions calculées :

```
For i=À to B*10 Step X-2

    MsgBox i.toString

Next i
```

La variable boucle peut être déclarée après For, dans ce cas cette variable n'existe que dans la boucle :

```
For K As Integer = 1 To 10
    …
Next K
```

On peut quitter prématurément la boucle avec **Exit For**.

```
For K As Integer = 1 To 10
    …
    If A=2 Then Exit For

Next K
```

Dans ce cas la boucle s'arrête de tourner si A=2, on poursuit après Next.

Remarque

Le nom de la variable de boucle est facultatif après Next :

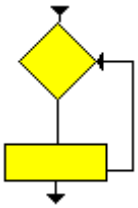
```
For K As Integer = 1 To 10
    …
Next
```

est correct.

Depuis la version 2005 il existe aussi **Continue For** qui permet de sauter au prochain Next et de poursuivre la boucle.

## V-Q-4 - Do Loop

Permet aussi de faire des boucles, mais **sans que le nombre de boucles (d'itérations) soit déterminé au départ**.



La boucle suivante tourne sans fin :

```
Do
Loop
```

Il faut une condition d'arrêt qui détermine la sortie de la boucle.

On doit mettre **Until** (Jusqu'à ce que) ou **While** (Tant que) avant la condition d'arrêt pour sortir de la boucle.

**On peut mettre la condition après Do :**

```
Do Until condition

      Code

Loop

 'Boucler jusqu'à ce que condition soit vraie.
```

Si condition est fausse, effectuer le code, boucler et recommencer le code jusqu'à ce que condition soit égale à True.

Attention, avant de débuter la boucle, la condition doit être fausse sinon la boucle ne sera jamais exécutée…

À chaque boucle la condition est évaluée.

Exemple pour chercher un mot dans une liste :

```
Lire Premier Mot
Do Until MotCherché=MotPointé

      Pointer Mot suivant

Loop
```

On peut aussi utiliser While (Tant que) :

```
Lire Premier mot
Do While MotCherché<>MotPointé

      Pointer Mot suivant

Loop
```

Tant que le mot cherché est différent du mot pointé, boucler.

**La condition peut être mise en fin de boucle**, cela permet d'effectuer au moins une fois le code. Cela évite aussi d'avoir à démarrer le processus avant la boucle, dans notre exemple cela permet d'éviter de lire le premier mot avant la boucle.

Les mots sont dans un tableau Mot(), premier élément Mot(0).

```
IndexMot=-1

Do

     IndexMot=IndexMot+1

Loop While MotCherché<>Mot(IndexMot)
```

Il faudrait en plus boucler jusqu'à la fin du tableau et pas plus.

Il y a **Exit Do** pour sortir de la boucle, il existe aussi **Continue Do** qui permet de sauter au prochain Loop et de poursuivre la boucle.

**Exemple complet :** Imposer la saisie d'un nombre négatif à l'utilisateur.

On utilise InPutBox qui ouvre une fenêtre et attend une réponse.

```
Dim Reponse as Single

Do

      Reponse=InPutBox("Entrer un nombre négatif.")

Loop  While Reponse>=0
```

Si le nombre n'est pas négatif, la boucle fonctionne et la boite InPutBox s'ouvre de nouveau. Si le nombre est négatif, on sort de la boucle.

### Comment créer une boucle qui tourne sans fin ?

```
Do

   …

Loop  While True
```

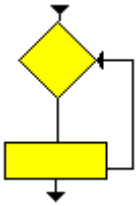
### Autre exemple

Créer une boucle affichant successivement dans une MessageBox les chiffres de 1 à 10.

```
Dim i As Integer = 0

Do

    i = i + 1           'incrémente i de 1

   MsgBox(i.ToString)   'affiche la valeur de i dans une messageBox

Loop Until i = 10       'sort de la boucle quand i=10
```

## V-Q-5 - While End While

**Permet une boucle qui tourne tant qu'une condition est vraie.**



Principe :

```
Tant que Condition
….
Fin Tant que
```

En VB :

```
While Condition

…

End While
```

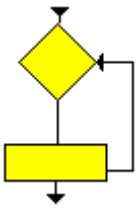
Exemple : on incrémente un compteur, on sort quand il est égal à 20.

```
Dim Counter As Integer = 0
While Counter < 20 ' Test la valeur du compteur.
   Counter += 1 ' Incrémente le compteur.
End While
```

Il y a **Exit While** pour sortir de la boucle, il existe aussi **Continue While** qui permet de sauter au prochain End While et de poursuivre la boucle.

## V-Q-6 - For Each

C'est une variante de la boucle For, mais elle **permet de parcourir les objets d'une collection**. Elle n'utilise pas l'indice.



Prenons un contrôle ListBox il a une collection Items qui contient tous les éléments de la ListBox

ListBox.item(0) contient la première ligne

ListBox.item(1) contient la seconde ligne

ListBox.item(2)…contient la troisième.

Parcourir tous les éléments de la ListBox et les mettre dans une variable V s'écrirait :

```
Dim mystring As String

Dim item as Object
```

```
For Each item in ListBox.Items

      mystring=mystring+item

Next
```

La variable de boucle peut être déclarée après For :

```
Dim mystring As String

For Each item As Objet in ListBox.items

      mystring=mystring+item

Next
```

Au lieu de déclarer Item comme un objet, on aurait pu le déclarer comme un ListBox.Item.

Cette notion de collection est beaucoup plus large que je le pensais.

Ici pour tester chaque caractère dans une String, et voir s'il est égal à "i", on peut utiliser For Each :

```
Dim chaine As String = "aeiou"

Dim c As String

For Each car As String In chaine

    If, car= "i" Then…

Next
```

**Attention, dans une boucle For Each, on peut parcourir la collection, mais on ne peut pas modifier un élément de la collection.**

## V-Q-7 - Switch

Switch est utilisé avec des couples d'arguments, si le premier est vrai, le second est retourné.

Réponse=Switch( Expression1, Reponse1, Expression2, Reponse2)

Si Expression2 est vrai Reponse2 est retourné.

```
Monnaie= Microsoft.VisualBasic.Switch(Pays = "USA", "Dollar", _
Pays = "FRANCE", "Euro", Pays = "Angleterre", "Livre")
```

Si Pays="FRANCE", cette expression est vrai, le second objet du couple est retourné.

Retourne Euro

## V-Q-8 - IIF

IIf est utilisé avec 3 arguments.

Si le premier argument est vrai, le second est retourné.

Si le premier argument est faux c'est le troisième qui est retourné.

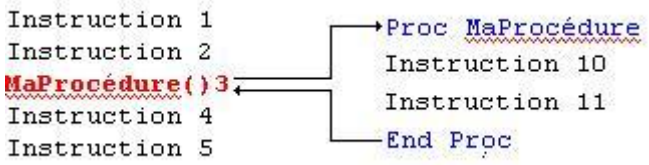
```
Reponse = IIf( Nombre > 0, "Positif", "Négatif ou 0")
```

Comme dans Switch on peut utiliser des procédures comme argument.

## V-R - Les procédures et leurs paramètres

On se souvient qu'en programmation procédurale on découpe les problèmes en fonctions: les Sub et les Function.

Quand on appelle une procédure (un sous-programme, une routine), le logiciel 'saute' au sous-programme, il effectue celui-ci puis revient effectuer ce qui est sous l'appel.




En VB les procédures sont des **Sub** ou des **Function**.

**- Les procédures Sub**

Elles débutent par le mot Sub et se terminent par End Sub. Elles ont des paramètres, mais ne 'retournent' rien.

Exemple: une sub qui retourne la somme de 2 nombres :

```
Sub Addition (a , b, result)
result= a+b
End Sub
```

Pour l'utiliser :

```
Dim a, b, result As Integer
a=2
b=3
Addition (a ,b ,result)
```

**- Les procédures Function**

Si on a besoin que la procédure retourne un résultat (un seul), on utilise une Fonction.

Elles débutent par Function et se terminent par End Function.

On peut fournir aux procédures des paramètres qui sont envoyés à la fonction.

Exemple :

```
Function Carré ( v as Single) As Single

    Return v*v
```

```
End Function
```

Cela crée une fonction qui se nomme 'Carré' , on peut lui envoyer un paramètre (elle accepte un Single dans v).

Cette fonction retourne un Single (indiqué par Function Carre() *As Single*) qui est le carré du paramètre fourni.

Pour l'utiliser :

```
Dim resultat As Single

resultat= Carré(2)        'resultat est alors égal à 4
```

On appelle la fonction carré avec le paramètre 2, elle retourne 4.

Les paramètres peuvent être des variables :

```
Dim resultat as Single

Dim valeur as Single=3

resultat= Carré(valeur)
```

Remarque : ici, on a un paramètre nommé 'valeur', on appelle la fonction Carré avec ce paramètre. Dans la fonction Carré on retrouve ce paramètre, il se nomme 'v' : Ce paramètre est passé à la fonction, mais il a un nom différent dans la fonction.

*On conseille, quand le nom d'une procédure est composé de plusieurs mots, de mettre la première lettre de chaque mot en majuscules.*

Exemple :

```
MyCalcul()
```

## V-R-1 - Les parenthèses

Rappel, même s'il n'y a pas de paramètre, mettre des () lors de l'appel de procédure.

```
MaRoutine()
```

## V-R-2 - Par Valeur, Par Référence

Il y a 2 manières d'envoyer des paramètres.

**Par valeur :** (By Val)c'est la valeur (le contenu de la variable) qui est envoyée.

(La variable est copiée dans une autre partie de la mémoire pour être utilisée par la routine appelée.)

**Par référence :** (By Ref) c'est l'adresse (le lieu physique où se trouve la variable) qui est envoyée. Si la Sub modifie la variable, cette modification sera visible dans la procédure appelante après le retour.

Exemple de procédures :

```
Sub MaProcedure (ByRef x as Long, ByVal y As Long)
```

```
End Sub
```

Chaque paramètre est ByRef ou ByVal suivi du nom du paramètre dans la procédure puis de son type.

Si j'appelle cette procédure à partir d'une procédure nommée Main() :

```
Sub Main()
Dim A, B As Long
         MaProcedure (A, B)

End sub
```

C'est l'adresse de A qui est envoyée et la valeur contenue dans la variable B. Elles se retrouvent dans les variables x et y de la procédure MaProcedure. Noter que le type de la variable fournie en paramètre dans Main et le type de la variable dans 'MaProcedure' doit être le même (ici un Long).

Si dans cette dernière je modifie x, A est modifié dans la Sub Main (puisque x et A pointe sur le même endroit). Si dans Maprocedure je modifie y, B n'est pas modifié.

Exemple permettant de bien différencier By Val et By Ref.

**Exemple avec ByVal :**

```
Sub MaProcedure

Dim A As Integer                                    'On créer une variable A

A=1                                                 'On met 1 dans A

Call MaProcedure(  A  ) 'On appelle la procédure MaProcedure  en envoyant le paramètre A

Label1.Text= A.ToString 'On affiche la valeur de A

End Sub


Sub MaProcedure ( ByVal Variable As Integer )     'La procédure reçoit la valeur de A donc  1

                                                    ' et la met dans 'Variable'

    Variable=Variable+1                              'Variable=2, mais A=1

End Sub
```

Après l'appel de la procédure A=1, Labe1 affiche '1', car bien que dans MaProcedure Variable =2 , cela n'a pas modifié A.

**Exemple avec ByRef :**

```
Sub MaProcedure

Dim A As Integer                                    'On créer une variable A

A=1                                                 'On met 1 dans A

Call MaProcedure(  A  ) 'On appelle la procédure MaProcedure en envoyant le paramètre A

Label1.Text= A.ToString 'On affiche la valeur de A

End Sub
```

```
Sub MaProcedure ( ByRef Variable As Integer )
'La procédure reçoit l'adresse de A ; Variable et A ont donc même adresse

                                                'Variable et A contiennent 1

    Variable=Variable+1                         'Variable=2, mais A=2 aussi

End Sub
```

Après l'appel de la procédure A=2, Labe1 affiche '2', car la procédure reçoit l'adresse de A ; Variable et A ont donc même adresse; si je modifie variable cela modifie A.

Compris !!



*L'avantage de passer un argument ByRef est que la procédure peut retourner une valeur au code qui a appelé la procédure en modifiant la valeur de la variable qui a été passée en argument.*

*L'avantage de passer un argument ByVal est que la variable de la routine est 'protégée' dans le code qui a appelé la procédure; elle ne peut pas être modifiée par la procédure qui reçoit le paramètre.*

## V-R-3 - Par Défaut, que se passe-t-il ?

Si on n'indique pas By Val ou By Ref…



*ATTENTION : par défaut les paramètres sont transmis PAR VALEUR.*

Pour la clarté du code et pour éviter toute ambiguïté, spécifier ByRef ou ByVal, c'est plus lisible, plus clair.

Taper Sub MaProcedure (ByRef x as Long, ByVal x As Long).

Plutôt que Sub MaProcedure ( x as Long, x As Long).

## V-R-4 - Optional

Un paramètre ou argument peut être Optional, c'est-à-dire facultatif.

Indique que cet argument n'est pas requis lorsque la procédure est appelée. Si ce mot-clé est utilisé, tous les arguments suivants doivent aussi être facultatifs et déclarés à l'aide du mot-clé Optional. Chaque déclaration d'argument facultative doit indiquer une valeur par défaut qui sera utilisée dans la routine s'il n'y a pas de paramètre.

```
Sub MaRoutine (Optional X As Integer=0)
```

## V-R-5 - Tableau de paramètres

Il est possible d'envoyer un tableau comme paramètre.

Exemple :

```
Dim Reponses(10) As Integer

'Appel de la Sub
Affiche( Reponses())
```

La Sub 'Affiche' débute par :

```
Sub Affiche ( ByVal R() As Integer )

End Sub
```

## V-R-6 - ParamArray

Parfois il faut envoyer des paramètres de même type, mais dont on ne connait pas le nombre, dans ce cas on utilise **ParamArray** (Liste de paramètres).

Exemple d'une Function nommée 'Somme' qui calcule la somme de X Integer :

```
Function Somme ( ByVal ParamArray Valeurs() as Integer) As Integer

    Dim i as Integer

    Dim Total as Integer

    For i=0 to Valeurs.Length-1

        Total += Valeurs(i)

    Next i

    Return  Total

End Sub
```

Pour appeler cette fonction :

```
Dim LeTotal As Integer

LeTotal= Somme (2, 5, 6, 8 ,5)
```

À noter que le paramètre ParamArray doit être le dernier des paramètres, c'est obligatoirement un paramètre ByVal et comme on l'a dit, tous les paramètres sont de même type.

## V-R-7 - Portée des procédures

Le terme Sub ou Function peut être précédé d'une indication de portée, la procédure sera-t-elle visible uniquement dans le module où elle se trouve ou partout ?

La procédure peut être **Private**. Dans ce cas on ne peut l'appeler qu'à partir du module qui la contient.

Les procédures événements, d'une Form sont, par exemple, Private :

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
 Handles MyBase.Load

End Sub
```

La procédure peut être **Public**. Dans ce cas on pourra l'appeler à partir de la totalité du programme.

La Sub 'Calcul' qui est par exemple dans un module, peut être appelée de partout.

```
Public Sub Calcul

End Sub
```

## V-R-8 - Nommage

**Sub , Fonctions**

Utilisez la 'case Pascal' pour les noms de routine (la première lettre de chaque mot est une majuscule).

Exemple : CalculTotal()

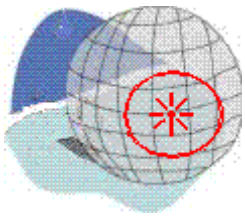
Évitez d'employer des noms difficiles pouvant être interprétés de manière subjective, notamment Analyse() pour une routine par exemple.

Utilisez les verbe/nom pour une routine : CalculTotal().

**Pour les noms de paramètres**, utilisez la 'case Camel' selon laquelle la première lettre des mots est une majuscule, sauf pour le premier mot.

Exemple : typeName.

## V-S - Portée des variables



Quand on déclare une variable, jusqu'où est-elle visible ?

Quand une variable est visible, on peut l'utiliser. Il est important souvent de rendre une variable visible dans une procédure, mais pas les autres, dans un seul module ou dans la totalité du programme. Comment faire cela ?

## V-S-1 - Dans les procédures

On déclare une variable avec **Dim**.

Si on déclare une variable dans une procédure (une Sub ou une Function), elle est visible uniquement dans cette procédure, c'est une variable locale :

```
Sub MaProcedure (ByRef X As Integer)

    Dim Y As Integer

    …

End sub
```

Y est déclaré en début de procédure, on pourra travailler avec Y dans la procédure jusqu'à End Sub.

Dans une autre procédure Y ne sera pas visible (l'utilisation de Y déclencherait une erreur).

Après End Sub Y n'existe plus, son contenu est perdu.

Il en est de même pour X qui est déclaré sur la ligne Sub (X reçoit la valeur envoyée comme paramètre).

Une autre procédure pourra déclarer et utiliser une variable Y, mais, même si elle a le même nom cela ne sera pas la même : chaque variable Y est uniquement visible dans sa procédure.

**Variable statique**

Si à la place de Dim on utilise **Static**, la variable est dite 'Statique' : à la sortie de la procédure, la variable et sa valeur continue d'exister et on garde sa valeur en mémoire, lors des appels suivants de la procédure, on retrouve la valeur de la variable.

Exemple :

```
Sub compteur

    Dim A as integer

    Static B as integer

    A +=1

    B +=1

End sub
```

À chaque appel de cette procédure A prend la valeur, 0 puis 1 puis disparait.

B prend les valeurs 0, puis 1, puis 2… (incrémentation à chaque appel).

## V-S-2 - Dans un bloc d'instructions

Si vous déclarez une variable dans un bloc, elle ne sera visible que dans ce bloc :

```
Do

Dim Compteur As integer

 Compteur +=1

…

Loop
```

La variable Compteur existe uniquement entre Do et Loop.

Cela est aussi valable pour les blocs If.

Exemple :

```
If A=0 Then

    Dim risk As String = "Haut"

Else
```

```
    Dim risk As String = "Bas"

End If

Console.WriteLine("Le risque est " & Risk)
```

Dans la dernière ligne Risk est soulignée comme contenant une erreur, car il est considéré comme non déclaré. En effet les 2 déclarations Dim risk sont dans les blocs 'If…Else' et 'Else…End If'.

**Attention quand même à la position de la variable locale, il peut y avoir des pièges.**

Voyons ce code :

```
Dim i, j,  As Integer

For i = 1 To 10
    For J = 1 To 3
        Dim k As Integer
        K+ =1
        'Imprimer k
    Next
Next
```

On souhaite que K=1 puis 2 ,3 ,1 ,2 ,3… FAUX !!

Cela donne :1 ,2 ,3 ,4 ,5 ,6 ,7… 30 !!!??? pas normal

En remplaçant par Dim k As Integer = 0 cela donne des 1 ,1 ,1 ,1 ,1 , 'normal

Et en intercalant la déclaration de k entre les 2 For, cela marche comme on le souhaite:1, 2 ,3 ,1 ,2 ,3.

```
Dim i, j,  As Integer

For i = 1 To 10

    Dim k As Integer
    For J = 1 To 3
        K+ =1
        'Imprimer k
    Next
Next
```

## V-S-3 - Dans la section déclaration d'un Module

Dans la section déclaration d'un module (en haut du module juste après la ligne 'Module'), on utilise à la place de Dim.

**Private**, dans ce cas la variable est propre au module, elle est visible dans toutes les procédures du module, pas dans les autres modules.

**Public**, dans ce cas la variable est accessible dans la totalité du programme.

```
Module Module1

    Public A as String

    …

 Sub MaRoutine

 End Sub
```

```
End Module
```

A est accessible partout dans la totalité du programme.

(On parle dans ce cas de variable dite 'Globale'.)

## V-S-4 - Dans la section déclaration d'une fenêtre, d'un formulaire

Dans la section déclaration d'un formulaire (en haut du formulaire juste après la ligne 'Inherits').

**Private:** indique dans ce cas que la variable est propre au formulaire, elle est visible dans toutes les procédures du formulaire, pas dans les autres modules ou formulaires.

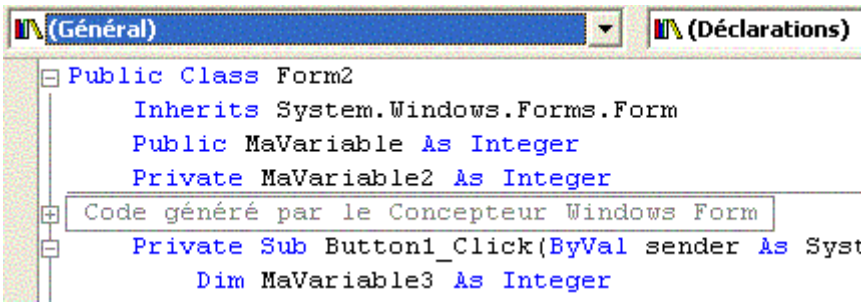
**Public :** indique de même une variable UNIQUEMENT visible dans le formulaire.

Elle est visible hors du formulaire à condition de la préfixer avec le nom du formulaire (Class1.A).

```
Class Class1

   Inherits System.Windows.Forms

    Public A as String

    …

 Sub MaRoutine

 End Sub

End Class
```

Exemple

On peut atteindre la zone de déclaration en déroulant le menu de droite.




Dans l'exemple ci-dessus :

MaVariable est visible dans le formulaire et hors du formulaire à condition d'utiliser Form1.MaVariable ;

MaVariable2 est visible dans le formulaire ;

MaVariable3 n'est visible que dans la procédure Button1_Click.

## V-S-5 - En pratique

Pour se repérer et se souvenir quelle est la portée d'une variable, on utilise une lettre en début du nom de la variable (notation avec un préfixe dite 'hongroise') :

g_MaVariable sera public (g comme global) ;

m_Variable2 sera accessible au niveau du module.

Dans un module standard, on peut mettre 'Public' toutes les variables que l'on veut rendre accessibles à tout le programme. Ce sont les variables (et constantes) générales utilisées dans la totalité de l'application : état du programme, utilisateur en cours…Pour des raisons que nous verrons plus loin, il faut éviter ce type de variable publique dite 'globale'.

Dans chaque formulaire on met dans la section déclarations, les variables du module : état du formulaire. Variable permettant l'affichage…

Dans chaque procédure les variables locales, compteur de boucle…

Pour les variables locales, on peut donc utiliser un même nom dans différentes procédures, par exemple, on nomme souvent I les variables de boucle dans toutes les procédures, par contre



*il faut éviter de donner un même nom à des variables dont la portée se recoupe.*

VB l'accepte et utilise la variable la plus proche, celle du bloc ou du module…mais c'est dangereux et générateur de bugs.



*De manière générale, utiliser des variables avec une portée la plus réduite possible.*

## V-S-6 - En général

Que ce soit pour les variables, procédures ou Classes :

-les variables qui sont **Dim** sont accessibles dans une procédure ;

- celles qui sont **Public** sont accessibles dans la totalité du programme ;

- celles qui sont **Private** ne sont accessibles qu'à l'intérieur même du module.

il y a en plus :

- celles qui sont **Protected** sont similaires aux Private, mais dans le cas des classes, elles ont une particularité en cas d'héritage ;

- celles qui sont **Friend** ne sont accessibles qu'à l'intérieur du projet, et pas par des éléments extérieurs au projet en cours.

# V-T - Les nombres aléatoires

Comment obtenir un nombre aléatoire ?

## V-T-1 - Avec la classe 'Random' du Framework

Il existe une classe (faisant partie de System ) nommée Random.

La méthode **Next**() retourne un Integer positif entre 0 et 2 147 483 647

La méthode **NextDouble()** retourne un Double entre 0 et 1.

La méthode **NextBytes()** retourne un Byte (octet)

On peut surcharger ces méthodes pour définir des bornes.

Exemple

J'instancie un objet à partir de la classe.

```
Dim Al As New Random
```

L'objet **Al** est initialisé avec une valeur probablement liée au temps, à l'horloge interne, aussi l'initialisation est 'aléatoire'.

Pour obtenir un nombre (un double) entre 0 et 1 (toujours inférieur à 1), j'écris :

```
MonNombrealeatoire=Al.NextDouble
```

Ensuite chaque NextDouble génère le nombre aléatoire suivant (à partir d'une formule).

Noter bien que dans ce qui précède, si on fait plusieurs fois Dim Al As New Random , le nombre obtenu par NextDouble n'est jamais le même.

Par contre, si on fait :

```
Dim Al As New Random(1)

MonNombrealeatoire=Al.NextDouble

MonNombrealeatoire=Al.NextDouble
```

On obtient toujours :

'0.248668584157093'

'0.110743977181029'

On obtient donc la même série, car on a imposé avec Random(1) une valeur de départ qui est fonction de (1) et non du temps.

Pour obtenir un nombre aléatoire entre 0 et 10, on utilise Next :

```
Dim Al As New Random()

MonNombrealeatoire=Al.Next(10)

MonNombrealeatoire=Al.Next(10)
```

On obtient la série: 2, 1, 4, 7, 6, 4, 3, 9

On travaille sur des 'Integer'.

Pour obtenir un nombre aléatoire entre 5 et 10 (mais < à 10), on utilise Next :

```
Dim Al As New Random()

MonNombrealeatoire=Al.Next(5,10)

MonNombrealeatoire=Al.Next(5,10)
```

La série comportera les nombres integers 5, 6, 7, 8, 9 (pas 10).

Pour remplir un tableau d'octets avec des octets aléatoires, on utilise NextBytes :

```
Dim rnd As New Random()
Dim b(10) As Byte
rnd.NextBytes(b)
```

## V-T-2 - Avec les instructions Rnd() et Randomize() de Visual Basic.Net

On peut utiliser les instructions VB. Ne faut-il pas mieux utiliser le Framework ?

**Rnd()** fournit un nombre aléatoire entre 0 et 1 (sans jamais atteindre 1):valeur inférieure à 1, mais supérieure ou égale à zéro; ce nombre aléatoire est un Single.

En fait ,si on fait des Rnd() successifs, le nombre aléatoire précédemment généré est utilisé pour le calcul du nombre aléatoire suivant (avec une formule mathématique complexe), ce qui fait que la suite de nombres aléatoires est toujours la même et qu'elle est périodique (au bout d'un grand nombre de tirages, on recommence la même suite).

**Randomize()** initialise le générateur de nombres aléatoires. Si on ne donne pas d'argument, Randomize utilise la valeur de l'horloge interne pour initialiser; cette valeur est due au hasard, aussi le Rnd qui suit va être dû au hasard.

Si on n'utilisait pas Randomize() avant Rnd(), la fonction Rnd() fournirait toujours les mêmes nombres aléatoires dans le même ordre.

En résumé

Rnd , s'il n'y a pas d'argument, fournit une suite de nombre pseudo aléatoire (le suivant étant calculé à partir du précédent), la suite est toujours la même, seule le premier change et est initialisé par Randomize qui est basé soit sur l'horloge système (et qui a priori initialise à une valeur toujours différente) s'il n'y a pas d'argument soit sur un argument.

Pour obtenir plusieurs fois les mêmes séries de nombres, utilisez Randomize avec un argument numérique puis appelez Rnd() avec un argument négatif.

**Simuler un jeu de lancer de dé**

Comment obtenir un nombre entier entre un et six au hasard ?

```
Dim MyValue As Integer

Randomize   ' Initialise le générateur de nombre aléatoire.

MyValue = CInt(Int((6 * Rnd()) + 1)) ' Génère un nombre aléatoire entre 1 et 6.
```

Rnd() fournissant un nombre aléatoire entre 0 et 1, je le multiplie par 6 et j'ajoute 1 pour qu'il soit entre 1 et 7 sans atteindre 7 (il peut être entre 1 et 6,999), je prends sa valeur entière : il est maintenant entre 1 et 6, enfin je le transforme en Integer.

## V-T-3 - En cryptographie avec le Framework

Pour remplir un tableau d'octets avec des octets aléatoires forts d'un point de vue cryptographique (pour générer un mot de passe par exemple), on utilise plutôt la classe RNGCryptoServiceProvider()

L'exemple suivant crée une séquence aléatoire de 100 octets de long et la stocke dans ra.

```
Imports System.Security.Cryptography

Dim ra() As Byte = New Byte(100) {}
Dim rng As New RNGCryptoServiceProvider()
rng.GetBytes(ra) ' les octets dans ra sont maintenant aléatoires.
```

Il existe aussi GetNonZeroBytes pour ne pas avoir d'octet=0.

## V-T-4 - Un peu de théorie

Un nombre aléatoire est obtenu par tirage au sort à égalité des chances, il est impossible de prévoir le tirage suivant.

Il existe des procédures physiques permettant de générer des nombres aléatoires : comptage de désintégration par compteur Geiger, analyse de bruit…

En informatique, on utilise les nombres pseudo aléatoires générés par des algorithmes.

L'implémentation actuelle de la classe Random est basée sur l'algorithme du générateur de nombres aléatoires soustractif de Donald E. Knuth. Pour plus d'information, consultez D. E. Knuth. "The Art of Computer Programming, volume 2 : Seminumerical Algorithms." Addision-Wesley, Reading, MA, deuxième édition, 1981.

Soit un nombre de départ x (nommé' graine'ou seed en anglais)

Le nombre est utilisé pour le calcul du nombre aléatoire suivant (avec une formule mathématique complexe), ce qui fait que la suite de nombre aléatoire est toujours la même pour une même graine et qu'elle est périodique.

La formule, dite générateur à congruence linéaire, pour trouver le nombre suivant, est de la forme :

$X_{n+1} = (aX_n+b) \bmod m$

$x_{n+1} = (1\ 664\ 525\ x_n + 1\ 013\ 904\ 223) \bmod 2^{32}$ (générateur de Knuth & Lewis)

Voir l'excellent article sur les nombres pseudo aléatoires:Article de P larbier :http://www.alrj.org/docs/algo/random.php

et l'excellent site de D. Müller: www.apprendre-en-ligne.net/random/index.html

On a vu que le générateur est périodique: au bout d'un certain nombre de tirages pseudo aléatoire, dès qu'un nombre apparait la seconde fois, on recommence la même série. En théorie, la période maximale serait m de mod m dans la formule soit 232.

Quelle est la période de la Classe Random en pratique?

Période: 81 672 063 avec Next (Integer)

Période: 562 183 903 avec NextDouble (Double)

C'est un ordre de grandeur, car en fonction de la graine (valeur de départ), la série et la période sont différentes (j'ai essayé !).

Tout l'art est de choisir la graine (le premier nombre) aléatoirement!!Cela est effectué par Randomize en VB ou l'instanciation d'un objet Random. Randomize utilise par exemple la valeur de l'horloge interne pour initialiser, cette valeur serait due au hasard

Amélioration

Comment générer des nombres plus aléatoires que les pseudo aléatoires ?

1- Solution créée par le programmeur

Le nombre aléatoire est la combinaison d'un nombre pseudo aléatoire et d'un nombre probablement aléatoire par exemple :

- Position de la souris

Temps entre 2 pressions sur des touches.

- Statistique d'activité du disque dur.

- Temps écoulé depuis… (Ce que fait randomize).

Exemple

Le nombre aléatoire est le produit d'un nombre pseudo aléatoire et du nombre de secondes écoulé depuis une date :

```
        Dim pAlea As Double     'Nombre pseudo aléatoire

        Dim second As Double    'Nombre de secondes depuis le 30/12/96

        Dim Alea  As Double     'Nombre aléatoire
```

```
        Randomize

        pAlea = Int((1000000 * Rnd) + 1)

        second = DateDiff("s", "12/30/96", Now)

        Alea =  second *pAlea
```

Il y a des biais, en particulier, si on utilise régulièrement cette routine, le nombre de secondes est régulièrement croissant. On pourrait améliorer en utilisant second Mod quelque chose.

2- Solution proposée par le Framework

Méthode utilisée dans la classe System.Security.Cryptography

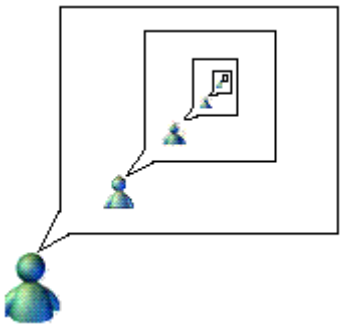
Le générateur doit être capable de produire des nombres aléatoires résistant à des attaques ou à des analyses statistiques qui permettraient de prédire la suite.

Les méthodes courantes pour générer des nombres aléatoires en cryptographie consistent à utiliser diverses sources disponibles sur un ordinateur : temps entre deux accès au disque, taille de la mémoire, mouvements du pointeur de la souris… et à faire passer le résultat dans une fonction de hachage cryptographique comme MD5 ou SHA-1 puis à utiliser cette valeur comme graine puis…

## V-U - La 'Récursivité'

La récursivité c'est quoi ?

Exemple trouvé sur developpeur.journaldunet.com :



"C'est l'histoire d'un petit garçon qui ne voulait pas dormir et dont la mère lui raconte l'histoire de la petite grenouille qui ne voulait pas dormir et dont la mère lui raconte l'histoire de l'ourson qui ne voulait pas dormir et dont la mère lui raconte l'histoire du bébé écureuil qui s'est endormi, et l'ourson s'endormit, et la petite grenouille s'endormit, et le petit garçon s'endormit."

Cette histoire, permet de mieux voir ce qui se produit lors de la récursivité : la procédure (le petit qui ne dort pas et à qui on raconte une histoire) appelle, la même procédure (le petit qui ne dort pas et à qui on raconte une histoire) qui appelle la même procédure… on passe au "niveau" suivant puis au suivant tant qu'on n'a pas atteint la condition d'arrêt (ici, l'endormissement). Celle-ci atteinte, la récursion se termine pour les autres niveaux en sens inverse en remontant.

*Une procédure est récursive si elle peut s'appeler elle-même.*

VB accepte les procédures récursives :

```
Sub Calcul()

…

    Calcul()

…

End Sub
```

On voit ici que la procédure Calcul() s'appelle elle même: la ligne centrale appelle de nouveau la procédure Calcul() avec nouveaux paramètres, nouvelles variables locales, à la sortie de la procédure (après End Sub), retour à la 'version' précédente de la procédure Calcul() ou on retrouve les variables de la précédente version.

*Une procédure non récursive appelle, elle, d'autres procédures.*

Pourquoi utiliser la récursivité ?

Une procédure récursive découpe le problème en morceaux plus petits et s'appelle elle-même pour résoudre chacun des plus petits morceaux, elle résout une petite partie du problème elle-même.

```
Sub Calcul(Gros)

        If…

           Résout petit problème

        Else

           Découpe

           Calcul(Petit)

        End If

End Sub
```

Ici 'Résout petit problème' s'appelle le point terminal ou le point d'arrêt, c'est la branche qu'une condition qui n'appelle pas de nouveau la fonction Calcul(). C'est indispensable.

Ou bien elle découpe le problème en plus petits morceaux et pour chaque morceau on appelle de nouveau la procédure :

```
Sub Calcul(Gros)

        If…

           Découpe

           Calcul(Petit)

        End If

End Sub
```

À un moment donné, la condition n'est pas remplie, cela correspond au point terminal.

On se rend compte qu'une bouche For Next peut être transformée en procédure récursive.

Exemple

Créons une procédure qui ajoute N éléments par ordre décroissant (ajoute l'élément N puis N-1 puis … 2 puis 1).

On l'appelle avec Calcul(10) par exemple :

avec For :

```
Function Calcul(N As Integer)

    Dim total As Integer

    For i= N to 1 Step-1

         total=total + i

     Next i

    Calcul=total

End Function


'Avec la récursivité:

Function Calcul(N As Integer)

    Dim total As Integer

    If N>0 Then

         total= N+ Calcul (N-1)

     End If

    Calcul= total

End Fonction
```

On l'appelle avec Calcul(10)

Mais la récursivité ne sert pas seulement à cela, elle sert à résoudre aussi des problèmes qui seraient extrêmement complexes en programmation non récursive.

## V-U-1 - Règles fondamentales d'une fonction récursive

**1-La récursivité doit s'arrêter à un moment donné.**

Il doit y avoir un point terminal (ou point d'arrêt).

Il doit y avoir dans la fonction récursive, une expression conditionnelle dont au moins un des cas conduit à une expression évaluable.

Il doit donc y avoir un chemin non récursif (chemin où la fonction ne s'appelle pas de nouveau).

Il doit y avoir un test qui survient obligatoirement et qui arrête le fonctionnement récursif sinon la fonction tourne sans fin (ou du moins, elle plante quand la pile est pleine).

**2- À aucun moment les paramètres appelant de nouveau la fonction ne doivent être les mêmes que l'appel précédent.**

Sinon cela tournera indéfiniment.

**3-Le nombre d'appels récursifs ne doit pas être très grand.**

Sous peine de 'StackOverflow' : la pile des appels qui stocke les adresses de retour de chaque appel récursif est pleine, elle dépasse ses capacités.

Certains ajoutent dans le code de la fonction récursive 'un compteur de sécurité' :

```
Sub Calcul(ByRef Compteur As Long)

If Compteur> LIMITE Then  Exit Sub

Compteur= Compteur+1

…

    Calcul(Compteur)

…

End Sub
```

Noter que le compteur est un paramètre ByRef, ce qui permet de toujours incrémenter la même variable.

Voir exemple sur les fractales.

**4-La fonction récursive ne doit pas déclarer un grand nombre de variables ou d'objets.**

Sous peine d'occuper une place trop importante en mémoire.

**5-Limiter les fonctions récursives à une seule procédure, éviter plusieurs fonctions récursives imbriquées.**

Sinon cela devient vite trop complexe.

**6- Chaque fois qu'elle est appelée de manière récursive (par elle-même, donc), un ou plusieurs des arguments qui lui sont transmis doivent se rapprocher de la condition d'arrêt.**

Sinon il n'y aura pas d'arrêt.

**7- La complexité du problème doit être réduite à chaque nouvel appel récursif.**

**8- Ne peut-on pas faire plus simple avec une boucle For Next ?**

Parfois une boucle simple remplace avantageusement une fonction récursive. Dans ce cas, utiliser la boucle !!

C'est le cas de la fonction factorielle !!

## V-U-2 - Exemple 1 : Inversion de chaines

Soit une chaine de caractères, on veut une fonction qui inverse cette chaine: dernier caractère en premier, avant-dernier en second…

Exemple: "abcd" retournera "dcba"

Principe de la fonction 'inverse' récursive:

La fonction 'inverse' met le dernier caractère au début et appelle la fonction 'inverse' avec comme paramètre la chaine moins le dernier caractère.

Exemple "abcd", on met "d" au début et rappelle la fonction inverse avec comme paramètre "abc".

Point d'arrêt : si la chaine est vide, plus d'appel récursif, on retourne une chaine vide.

```
Function inverse(ByVal st As String) As String

If st = "" Then

   inverse = ""

Else

   inverse = st.Substring(st.Length() - 1, 1) + inverse(st.Substring(0, st.Length() - 1))

End If

End Function
```

## V-U-3 - Exemple 2 : Calcul de 'Factorielle'

On rappelle que N! (factorielle N)= 1*2*3*…*(N-2)*(N-1)*N

Exemple Factorielle 3 =1*2*3 :

```
Dim R As Long

R=Factorielle(3)     'retournera 6
```

Cette fonction n'est pas fournie par VB, créons une fonction Factorielle SANS récursivité :

```
Function Factorielle (ByVal N as Long) As Long

    Dim i As Long

    Resultat=1

    For i= 1 to N

        Resultat=i* Resultat

    Next i

    Return Resultat

end Function
```

Cela crée une fonction recevant le paramètre N et retournant un long.

La boucle effectue bien 1*2*3…*N-1*N.

Factorielle avec 'Récursivité' :

Une autre manière de calculer une factorielle est d'utiliser la récursivité.

Comment faire ?

On sait que N!= N * (N-1) * (N-2)… 3 * 2 * 1

on remarque donc que Factorielle N!= N * Factorielle(N-1)

N!= N*(N-1)! : en sachant que 1!=1

Créons la fonction.

Si N=1 la fonction retourne 1 sinon elle retourne N* factorielle(N-1) :

```
Function Factorielle (ByVal N as Long) As Long

    If N=1 then

        Return 1

    Else

        Return N* Factorielle(N-1)

    End If

end Function
```

Dans la fonction Factorielle on appelle la fonction Factorielle, c'est bien récursif.

Pour N=4.

La fonction 'descend' et appelle chaque fois la factorielle du nombre inférieur.

La fonction Factorielle est appelée 4 fois :

Factorielle(4) appelle Factorielle(3) qui appelle Factorielle(2) qui appelle Factorielle(1)

Puis la fonction remonte en retournant le résultat de chaque factorielle.

Factorielle(1) retourne 1

Factorielle(2)retourne 2 '2*factorielle(1)

Factorielle(3)retourne 6 '3*factorielle(2)

Factorielle(4) retourne 24 '4*factorielle(3)

Vb gère cela avec une pile des appels. il met dans une pile les uns au-dessus des autres les appels, quand il remonte, il dépile de haut en bas (dernier rentré, premier sorti).



*Attention : la pile a une taille maximum, si N est trop grand, on déclenche une erreur de type StackOverflow.*

## V-U-4 - Exemple 3 : Calcul d'une expression avec parenthèses multiples

Comment calculer la valeur de la chaine "(4+2(2*8)-(5/(8+1)))"

Une partie du code nommée Calculsimple sait calculer une chaine de type "8+1" ou "4+2" sans parenthèses.

Il faut gérer les parenthèses : la sub découpe ce qui est entre parenthèses et s'appelle elle-même pour calculer ce qui est entre parenthèses.

La sub calcul fait 2 choses.

S'il y a des parenthèses : appelle Calcul() avec comme paramètre la chaine entre parenthèses puis remplace la chaine entre parenthèses par sa valeur.

S'il n'y a pas de parenthèses calcule l'expression simple (= - * /).

Voici l'algorithme :

```
Sub Calcul(Chaine As String) As String

    Si  Chaine contient  "("

        Decouper ValeurEntreParenthese

        Resultat=Calcul (ValeurEntreParenthese)        'Appel récursif

        Remplacer (ValeurEntreParenthese) par Resultat

    Sinon

        CalculSimple

    Fin Si

End Sub
```

## V-U-5 - Exemple 4 : PGCD

On rappelle que le PGCD est le 'Plus Grand Commun Diviseur'.

Soit a et b 2 nombres :

Si b divise a => PGCD=b. sinon, PGCD(a,b) = PGCD(b, a mod b)

```
Function PGCD(ByVal P As Long, ByVal Q As Long) As Long

If Q Mod P = 0 Then

Return P

Else

     Return PGCD(Q, P Mod Q)

End If
```

## V-U-6 - Exemple 5 : Tri récursif

**Tri récursif**

Le principe est que la fonction récursive scinde le tableau en 2 et pour chaque partie appelle de nouveau le tri récursif, la condition d'arrêt survient quand le dernier élément est < ou = au premier.

Dans un premier temps on range le tableau de telle sorte que tous les éléments inférieurs à l'élément d'indice pivot se trouvent placés à la gauche de celui-ci et donc tous les éléments supérieurs à sa droite. Ensuite on appelle à nouveau (récursivement) la procédure QuickSort pour chacun des deux sous-tableaux.

Cette méthode de tri récursif qui se nomme QuickSort est proportionnellement efficace au désordre du tableau à trier. Cette méthode mettra plus de temps (qu'une autre méthode) à trier un tableau qui est déjà en partie trié qu'un tableau rangé au hasard… Mais en cas de désordre intégral, c'est certainement la plus rapide.

```
Sub QuickSort(debut As Integer, fin As Integer)
Dim pivot, gauche, droite, temp  As Integer

  pivot  = debut
  gauche = debut
  droite = fin
  do
    if t(gauche) >= t(droite) then
    'échanger si nécessairet(droite) et t(gauche)
      temp = t(gauche)
      t(gauche) = t(droite)
      t(droite) = temp
      pivot = gauche + droite - pivot 'nouvelle position du pivot
                    'pivot est alors égal à droite ou à gauche, car pivot était avant égal
                    'à gauche ou à droite
    End If
    if pivot = gauche then droite=droite-1 else gauche=gauche+1
  loop until gauche = droite
  if debut < gauche - 1 then QuickSort(debut, gauche - 1) ' //appel récursif sur la partie gauche
  if fin > droite + 1 then QuickSort(droite + 1, fin)  'appel récursif sur la partie droite
End Sub
```

Comment l'utiliser

On crée un tableau Public d'integer contenant les valeurs à trier :

```
Public t() As Integer = {10, 2, 7, 4, 1, 3, 12, 6}
```

```
Dim i As Integer

Call QuickSort(0, 7) 'paramètre= premier et dernier élément du tableau
```

Affichage du tableau trié dans une texteBox1

```
For I = 0 To 7

TextBox1.Text = TextBox1.Text + ControlChars.CrLf + t(i).tostring

Next
```

## V-U-7 - Exemple 6 : Parcours de répertoires et de sous répertoires

On veut afficher dans une ListBox les noms des répertoires, sous-répertoires et fichiers.

On crée une routine AfficheTree qui affiche :

- le nom du répertoire courant ;

- le nom des fichiers du répertoire courant ;

- qui parcourt les sous-répertoires et pour chacun d'eux appelle AfficheTree :

```
Imports System.IO



Sub  AfficheTree ( ByVal myDir As String, ByVal Optional Niveau As Integer =0)
```

```
'Affiche le répertoire myDir
List1.Items.Add(New String (" ", niveau *2) & myDir)

'Affiche les fichiers
For Each fichier As String  In Directory.GetFiles( myDir)
    List1.Items.Add(New String (" ", niveau *2+2) & fichier)
Next

'Parcourt les sous-répertoires
For each sousRepertoire As String In Directory.GetDirectories( myDir)
    'Appel de manière récursive 'AfficheTree pour afficher le contenu des sous répertoires.
    AfficheTree (sousRepertoire, niveau+1)
Next

End Sub
```

## V-U-8 - Exemple 7 : Évaluation d'un nombre écrit en chiffres romains

On veut taper III et voir s'afficher 3.

Taper M et voir s'afficher 1000.

Taper XLVIII et voir s'afficher 48.

On remarque (je ne l'ai pas fait tout seul !!) que :

chaque caractère romain a une valeur (I=1, V=5, X=10, L=50, C=100, D=500, M=1000) ;

pour deux caractères, on compare leurs valeurs :

si le premier est plus petit, on le soustrait au second: IX = 10 - 1 = 9 ;

si le premier est plus grand, on l'ajoute au second: VI = 5 + 1 = 6.

Pour une suite de n caractères : en partant de la gauche, si le premier chiffre a une valeur inférieure au deuxième, alors on le soustrait de la valeur de tout le reste, sinon on l'additionne à la valeur de tout le reste…

Le programme va donc comparer la valeur des 2 caractères de gauche, il va ajouter (si la valeur du premier est plus grande) ou soustraire (si la valeur du premier est plus petite) la valeur du premier caractère à la valeur de la chaine raccourcie du premier caractère.

Exemple : pour XLVIII

```
X plus petit que L  donc  -10 +valeur de (LVIII)
```

```
L plus grand que V  donc  -10 +50 + valeur de (VIII)

V plus grand que I  donc  -10 +50 + 5 + valeur de (III)

….
```

Il faut créer une routine nommée valeur qui calcule la valeur d'un caractère.

Et la routine Eval qui calcule l'expression.

S'il y a un caractère dans la chaine c passée en paramètre, on retourne sa valeur, s'il y en a plusieurs, on compare les 2 premiers caractères, et on additionne ou soustrait à la valeur du premier caractère l' Eval (appel récursif) du reste de la chaine.

```
Function valeur(ByVal c As Char) As Integer

Select Case c

Case "M"c : valeur = 1000

Case "D"c : valeur = 500

Case "C"c : valeur = 100

Case "L"c : valeur = 50

Case "X"c : valeur = 10

Case "V"c : valeur = 5

Case "I"c : valeur = 1

End Select

End Function



Function eval(ByVal s As String) As Integer


Dim n As Integer


If s.Length = 1 Then

    eval = valeur(s.Chars(0))

Else

    n = valeur(s.Chars(0))

    If n < valeur(s.Chars(1)) Then n = -n

        eval = n + eval(s.Substring(1, s.Length - 1))

    End If

End Function
```

Si on veut tester : créer dans une form 2 texteBox : TextDecimal, TextRomain et un bouton Button1 :

```
Private Sub Button1_Click
```

```
    TextDecimal.Text = eval(TextRomain.Text).ToString

End Sub
```

## V-U-9 - Exemple 8 : Suite de Fibonacci

"Possédant initialement un couple de lapins, combien de couples obtient-on en douze mois si chaque couple engendre tous les mois un nouveau couple à compter du second mois de son existence ?"

**On suppose que :**

- le premier mois, il y a juste une paire de lapins ;
- les lapins ne sont pubères qu'à partir du deuxième mois ;
- chaque mois, toute paire susceptible de procréer engendre effectivement une nouvelle paire de lapins ;
- les lapins ne meurent jamais (donc la suite de Fibonacci est strictement croissante).

Sont notés en gras, les couples productifs.

En janvier : 1 couple
En février : **1** couple
En mars : 1 + **1** = 2 couples
En avril : 1 + **2** = 3 couples
En mai : 2 + **3** = 5 couples
En juin : 3 + **5** = 8 couples
En juillet : 5 + **8** = 13 couples
En août : 8 + **13** = 21 couples
En septembre : 13 + **21** = 34 couples
En octobre : 21 + **34** = 55 couples
En novembre : 34 + **55** = 89 couples
En décembre : 55 + **89** = 144 couples

Les réponses constituent les nombres de la suite de Fibonacci : 1 - 1 - 2 - 3 - 5 - 8 - 13 - 21 - …, dont chaque terme à partir du 3e est la somme des deux précédents.

```
Function Fibonacci(ByVal n As Integer)

' si n > 91, cela entraine un overflows sur les long.

Dim result As Long = 0


If n < = 2 Then

    result = 1

Else

    result = Fibonacci(n - 1) + Fibonacci(n - 2)

End If


Return result


End Function
```

Programme itératif correspondant :

```
Function Fibonacci2(ByVal n As Integer)

Dim u, v, w, i As Long


If n <= 0 Then Return 0

If n = 0 Then Return 1


u = 0

v = 1

For i = 2 To n

w = u + v

u = v

v = w

Next


Return v

End Function
```

## V-U-10 - Exemple 9 : Fractales

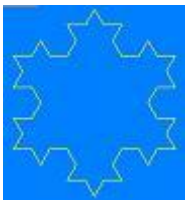
Calculs de fractales

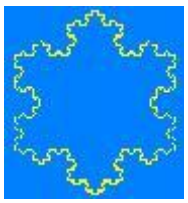
Les fonctions fractales sont des fonctions récursives théoriquement infinies…

Le Flocon de Koch, du nom du mathématicien suédois Helge Von Koch (1870-1924), est une courbe fermée, reproduisant un triangle équilatéral à des échelles de plus en plus petites. En répétant ce processus une infinité de fois, la courbe obtenue possède alors un périmètre infini, mais une aire limitée. Pour ce faire, chaque segment formant un triangle équilatéral est lui-même décomposé en un triangle équilatéral dont la base mesure un tiers du segment, centrée et confondue à ce segment.

On va donc créer une fonction récursive nommée 'flocon' qui décompose un segment en ajoutant un triangle puis qui pour chaque segment appelle la fonction 'flocon.

Comme on ne peut pas afficher des points infiniment petits, on va ajouter une condition d'arrêt qui est déclenchée par le nombre d'appels récursifs. Si la condition d'arrêt est remplie, on dessine le segment.





Voici la fonction récursive :

```
Private Sub Flocon(ByRef gh As Graphics, ByVal a As Point, ByRef b As Point, ByRef n As Integer)

'procédure récursive pour dessiner la fractale de Von Koch

Dim d, c, e As Point

Dim Couleur As Color = Color.Aqua


If n = 0 Then

    'Condition de sortie de la récursivité

    gh.DrawLine(New Pen(Color.Red), a.X, a.Y, b.X, b.Y)

Else

    'Appel récursif

    c = Tiers(a, b)

    d = Tiers(b, a)

    e = Sommet(c, d)

    Flocon(gh, a, c, n - 1)

    Flocon(gh, c, e, n - 1)

    Flocon(gh, e, d, n - 1)

    Flocon(gh, d, b, n - 1)

End If


End Sub
```

Pour que cela fonctionne, il faut les deux routines suivantes :

```
Private Function Sommet(ByRef a As Point, ByRef b As Point) As Point

'Calcule le sommet du triangle équilatéral

'dont la base est définie par 2 points

    Sommet.x = (b.x + a.x) / 2 + (b.y - a.y) * Racine3Sur2

    Sommet.y = (b.y + a.y) / 2 - (b.x - a.x) * Racine3Sur2

End Function




Private Function Tiers(ByRef a As Point, ByRef b As Point) As Point

'Calcul le premier tiers d'un segment

'Attention: Le résultat est orienté

    Tiers.x = (b.x - a.x) / 3 + a.x

    Tiers.y = (b.y - a.y) / 3 + a.y

End Function
```

Pour utiliser cet exemple, il faut dans un formulaire un PictureBox nommé PictureBox1 pour afficher la fractale, un TextBox1 permettant de saisir le nombre d'appels récursifs ( ne pas dépasser 9 en pratique), et un bouton command1 exécutant le calcul et l'affichage.

Mettre en haut du module :

```
Const Racine3Sur2 As Double = 0.866025404
```

```
Private Sub Command1_Click(ByVal eventSender As System.Object, ByVal eventArgs As
 System.EventArgs) Handles Command1.Click

Dim newBitmap As Bitmap = New Bitmap(400, 400)     'créons un BitMap

Dim g As Graphics = Graphics.FromImage(newBitmap) 'créons un Graphics et y mettre le BitMap

Dim t As Integer



'déclarons les 3 points du triangle initial

Dim a(2) As Point

Dim b(2) As Point



'donnons une valeur à 2 points, calculons le troisième

a(0).X = 164

a(0).Y = 10

b(1).X = 20

b(1).Y = 260

b(0) = Sommet(a(0), b(1))

a(1) = b(0)
```

```
a(2) = b(1)

b(2) = a(0)


'Pour chaque côté

For t = 0 To 2

  'Appelons la fonction récursive

    Flocon(g, a(t), b(t), Val(TextBox1.Text))

Next t


'Affichons

PictureBox1.Image = newBitmap


End Sub
```

Code issu d'un code VB6 situé sur CodeS-SourceS VBFrance.com écrit par D. Thuler et transcrit par moi en VB.Net.Merci David.

## V-U-11 - Autre

**Recherche de chemin dans un labyrinthe.**

Le principe est que la fonction récursive teste le déplacement à droite, à gauche, en haut, en bas. La condition d'arrêt se produit quand on a trouvé la sortie; les endroits où on est déjà passé sont notés.

L'article http://recursivite.developpez.com/ donne des exemples et des explications extraordinaires de programmes récursifs EN PASCAL. Si vous avez le courage de les traduire en VB , envoyez-les-moi !!

## V-V - Faut-il oublier le GoTo ?

*Et hop !! On saute…*

Le Goto c'est quoi ?

Faut-il utiliser le GoTo ?

### V-V-1 - Le 'Goto'

L'instruction GoTo permet de transférer le contrôle à l'emplacement d'une étiquette (on dit parfois un label) locale. Une étiquette permet de localiser un endroit du code Les étiquettes sont toujours terminées par un deux-points ":".

**Goto permet un saut non conditionnel :** aller à, sauter vers une étiquette :

```
…
GoTo FIN
A=2
B=A+2
FIN:
```

FIN: est une étiquette, un mot en début de ligne qui désigne un endroit du code; pour créer une étiquette, taper en début de ligne un mot suggestif de l'endroit, puis ajouter ":".

Le programme saute de la ligne GoTo FIN à l'étiquette FIN: puis se poursuit après FIN (Les instructions entre Goto FIN et FIN : ne sont pas exécutées :

Le GoTo est souvent utilisé avec une instruction If (pour rendre le saut conditionnel):

```
If A=0 Then GoTo FIN
…
FIN:
```

## V-V-2 - Pourquoi éviter le 'GoTo'

L'utilisation du GoTo est peu élégante et à éviter:

Cela casse la structure du code qui se déroule de haut en bas, le cheminement du programme est moins évident, moins visible; s'il y a plusieurs GoTo le programme devient vite illisible.

On parle dans ce cas de "code spaghetti"  ou de "code kangourou" (à cause des sauts). Dans la programmation structurée, il faut bannir le 'GoTo'. On le remplacera avantageusement par des If…Then et des boucles.

Exemple

On peut remplacer avantageusement la ligne :

```
If A=0 Then GoTo FIN
B=1
FIN:
```

Par :

```
If A<>0 Then
    B=1
End if
```

Autre exemple

Saisir à l'aide d'une InPutBox un nombre. S'il n'est pas entre 1 et 31, ressaisir.

Avec GoTo :

```
Dim réponse As String

Question:

réponse= InPutBox ("Donner un nombre entre 1 et 31")

If Val(Reponse)<1 Or Val(Reponse) >31 Then GoTo Question
```

Sans GoTo :

```
Dim réponse As String

While Val(Reponse)<1 Or Val(Reponse) >31

    réponse= InPutBox ("Donner un nombre entre 1 et 31")

Wend
```

## V-V-3 - Quand utiliser le 'GoTo'

Il est cependant parfois utilisé pour la clarté du code, car il permet de sortir de plusieurs blocs de code qui se suivent.

Exemple : détection d'erreur avant l'écriture sur un fichier :

```
If Not Exit(File) Then

    errorState=File.NotExist

    GoTo FIN

End if


If Not Open(File) Then

    errorState=File.NotOpen

    GoTo FIN

End if


If Not Overwrite(File) Then

    errorState=File.NotOverwrite

    GoTo FIN

End if

FIN:
```

C'est propre.

Là aussi on aurait pu utiliser des If imbriqués :

```
  If  Exit(File) Then
```

```
      If  Open(File) Then
        If  Overwrite(File) Then
            errorState=File.Ok
        Else
            errorState=File.NotOverwrite
        End if

      Else
            errorState=File.NotOpen
      End if

  Else
            errorState=File.NotExit
  End if
```

La solution avec les GoTo à l'avantage d'être plus lisible.

Pour info, il existe une autre solution élégante et originale utilisant un Select Case : les "Case" peuvent contenir n'importe quelle expression.

```
Select Case true
        Case len(File) = 0
            msgbox "Pas de nom de fichier"
        Case Not Exit(File)
            errorState=File.NotExist
        Case Not Open(File)
            errorState=File.NotOpen
        Case Not Overwrite(File)
            errorState=File.NotOverwrite
        Case else
        'placer les exceptions ici
End Select
```

# V-W - Les bases binaires, hexadécimales, l'algèbre de Boole

*Mr Georges Boole 1805-1864*

On étudie dans ce chapitre les différentes bases (binaire, hexadécimale…) en approfondissant. Ce chapitre n'est pas à lire de suite pour les débutants.

Voici ce que nous allons voir.

**Le Bit, poids d'un bit.**

**Conversion décimale binaire.**

**L'octet, Kilo, Méga, Téracotet**

**Opération: L'addition, la multiplication binaire, les nombres négatifs.**

**Table de vérité.**

**Fonction logique. Or And, Not, Xor…**

**Notation.**

**Ordre des évaluations.**

**Loi de composition.**

**Déplacement de bit.**

**Hexadécimale.**

**Intérêts en Visual Basic.**

**A notre disposition Boolean, Integer Byte…**

**Conversion binaire, décimale, hexadécimale.**

**Cas particulier: If A then**

**Les masques de bit**

**Cryptage par Xor**

**Travail sur les couleurs, graphiques…**

**Viewer hexadécimal.**

## V-W-1 - Introduction

L'**algèbre de Boole** est la partie des mathématiques, de la logique de l'électronique et de l'informatique qui s'intéresse aux opérations et aux fonctions sur les variables logiques. En logique propositionnelle, une expression est soit vraie soit fausse. (le vrai (1) et le faux (0)).

**Georges Boole** (1815-1864), physicien Anglais définit en 1847 une algèbre qui est applicable au raisonnement logique, qui traite des fonctions à variables binaires (deux valeurs). Mais il ne s'applique pas aux systèmes à plus de deux états d'équilibre.

**Une variable booléenne**, ou logique, ou binaire ne prend que deux valeurs (elle est généralement stockée sous la forme d'un bit).

Vers la fin des années 30, **Claude Shannon** démontra qu'à l'aide d'interrupteurs fermés pour "vrai" et ouverts pour "faux" il était possible d'effectuer des opérations logiques en associant le nombre 1 pour "vrai" et 0 pour "faux".

Ce codage de l'information est nommé **base binaire.** C'est avec ce codage que fonctionnent les ordinateurs. Il consiste à utiliser deux états (représentés par les chiffres 0 et 1) pour coder les informations.

Il permet d'étudier les circuits logiques.

## V-W-2 - Notions théoriques

***Notion de base:***

On utilise diverses bases dans la vie courante.

Pour les heures, minutes, secondes on utilise sans le savoir **une base 'soixante' :**

60 secondes = 1 minute

60 minutes = 1 heure

Si je compte : 1 s, 2s, 3s,…59s, 1mn, 1mn+1 s… 1mn+59s, 2 minutes… 59 mn+ 59s, 1h…

**En base décimale**, on a à notre disposition les caractères 1 2 3 4 5 6 7 8 9.

Si on veut représenter le nombre au-dessus de 9, comme on n'a plus de caractère, on recommence avec 1 mais en le décalant vers la gauche pour signifier qu'il s'agit d'une dizaine. On obtient '10'. Le nombre suivant est '11' puis "12'… Idem pour 100, 1000…

**En base binaire**, on n'a que le caractère 1 (et le zéro), 1 binaire= 1 décimal.

Si on veut écrire en binaire le nombre au-dessus de 1, comme on n'a plus de caractère, on procède de même en décalant vers la gauche le 1 :

10 binaire= 2 décimal.

Le nombre suivant est 11 binaire (3 en décimal).

Puis 100 (4 en décimal), car il faut de nouveau décaler.

**En base hexadécimale**, on a à notre disposition les caractères 1 2 3 4 5 6 7 8 9 A B C D E F.

Aussi 10 décimal = A en hexadécimal

…

15 décimal = F en hexadécimal

16 décimal = 10 en hexadécimal

Si on veut représenter le nombre au-dessus de 15, comme on n'a plus de caractères, on recommence avec 1 mais en le décalant vers la gauche pour signifier qu'il s'agit d'une 'seizaine'. On obtient '10' hexadécimal qui correspond à 16 décimal. Le nombre suivant est '11' puis "12' jusqu'a 1F puis 100… 1000…

***Utilisons l'analogie des allumettes et des paquets d'allumettes.***

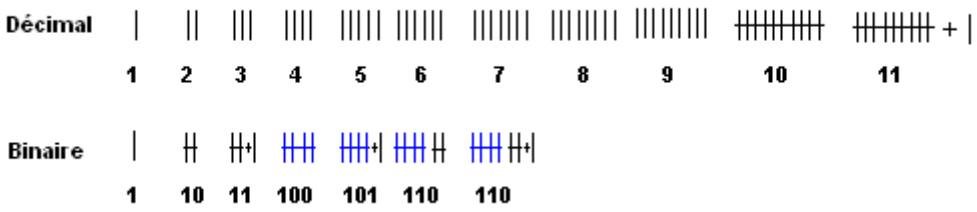
**En décimal** on a des paquets de 10 allumettes.

On compte 1 ,2 ,3…9, 1 paquet de 10, puis 1 paquet + 1 allumette…

On a des gros paquets de 100, des énormes paquets de 1000 allumettes…

**En binaire** on a des paquets de 2 allumettes et des gros paquets de 4 allumettes.

On compte 1 , 1 paquet de 2, puis 1 paquet + 1 allumette, puis 1 gros paquet de 4…




Donc pour compter en binaire : Binaire Décimal

1 allumette. 1 1

1 paquet de 2 10 2

1 paquet de 2 + 1 allumette 11 3

1 gros paquet de 4 100 4

1 gros paquet de 4 +1 allumette 101 5

1 gros paquet de 4 +1 paquet de 2 110 6

….

Le nombre d'allumettes qu'il y a dans un paquet se nomme **le poids**.

**En hexadécimal**, les paquets sont de 16 allumettes.

On compte 1, 2, 3 …jusqu'a 15 allumettes puis 1 paquet de 16 puis 1 paquet plus une allumette…

### *Base binaire*

### *Soyons maintenant un peu plus scientifique.*

Le bit

Le terme **bit** (b avec une minuscule) signifie "binary digit", c'est-à-dire 0 ou 1 en numérotation binaire. Il s'agit de la plus petite unité d'information manipulable par un ordinateur.

Physiquement cette information binaire correspond à :

* un signal électrique ou magnétique (pas de signal=0, au-delà d'un certain seuil de +5V, valeur 1) ;

* des trous ou pas de trous sur un CD ;

* l'état de bistables, c'est-à-dire des composants électroniques contenus dans les circuits intégrés qui ont deux états d'équilibre (état 1, état 0).

Avec un bit il est donc possible d'avoir deux états :

soit 1 ;

soit 0.

Grâce à 2 bits, il est possible d'obtenir quatre états différents (2*2) :

On peut avec 2 bits , avoir les valeurs: 0, 1, 10, 11 soit 0,1, 2, 3 en décimal :

| | | Décimal |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 2 |
| 1 | 1 | 3 |

Avec 3 bits, il est possible d'obtenir huit états différents (2*2*2) de 0 à 7 en décimal :

| En binaire 3 bits | En décimal |
|---|---|
| 000 | 0 |
| 001 | 1 |
| 010 | 2 |
| 011 | 3 |
| 100 | 4 |
| 101 | 5 |
| 110 | 6 |
| 111 | 7 |

Avec 4 bits, il est possible d'obtenir huit états différents (2*2*2*2) de 0 à 15 en décimal :

| En binaire 4 bits | En décimal |
|---|---|
| 0000 | 0 |
| 0001 | 1 |
| 0010 | 2 |
| 0011 | 3 |
| 0100 | 4 |
| 0101 | 5 |
| 0110 | 6 |
| 0111 | 7 |
| 1000 | 8 |
| 1001 | 9 |
| 1010 | 10 |
| 1011 | 11 |
| 1100 | 12 |
| 1101 | 13 |
| 1110 | 14 |
| 1111 | 15 |

Pour un groupe de n bits, il est possible de représenter 2n valeurs ( de 0 à 2n-1 ).

Avec 8 bits =256 valeurs.

Avec 16 bits=65536 valeurs.

Avec 32 bits=4294967296 valeurs.

Avec 64 bits=18446744073709551616 valeurs.

Avec 8 bits (un octet) on peut représenter un nombre qui peut avoir 256 valeurs différentes :

de 0 à 255

| binaire 8 bits | décimal |
|---|---|
| 00000000 | 0 |
| 00000001 | 1 |
| 00000010 | 2 |
| 00000011 | 3 |
| ..... | .. |
| ..... | .. |
| 11111110 | 254 |
| 11111111 | 255 |

Poids des bits:

Chaque bit a un **poids**, qui dépend de la position du bit en partant de la droite. Comme les dizaines, les centaines et les milliers pour un nombre décimal, le poids d'un bit croît d'une puissance de deux en allant de la droite vers la gauche :

| Poids | $2^7 = 128$ | $2^6 = 64$ | $2^5 = 32$ | $2^4 = 16$ | $2^3 = 8$ | $2^2 = 4$ | $2^1 = 2$ | $2^0 = 1$ |
|---|---|---|---|---|---|---|---|---|

Remarque : cela est valable pour toutes les bases.

Soit un nombre 'mno' en base b

Le premier chiffre à droite a la valeur : o x b1

Le deuxième chiffre à droite a la valeur : n x b2

Le troisième chiffre à droite a la valeur : m x b3

En allant de la droite vers la gauche le poids croît d'une puissance de b.

Le nombre total est la somme de toutes les valeurs :

o x b1 + n x b2 + m x b3

### Conversion

**Pour convertir un nombre binaire en nombre décimal**

Il faut multiplier la valeur de chaque bit par son poids, puis d'additionner les résultats.

Ainsi, le mot binaire 10101 vaut en décimal :

$2^4x1 + 2^3x0 + 2^2x1 + 2^1x0 + 2^0x1$

= 16x1 + 8x0 + 4x1 + 2x0 + 1x1

= 21

**Pour convertir un nombre décimal en nombre binaire**

*- Méthode des poids*

Soit 21 en décimal.

Il faut connaitre les poids (puissance de 2): 2, 4, 8, 16 ,32…

Trouver le premier poids (la première puissance de 2) inférieur au nombre décimal 21 : c'est 16

16 donne en binaire 10000

Faire 21-16 =5

Trouver le premier poids inférieur au nombre 5, c'est 4.

4 donne en binaire 100

Faire 5-4= 1

Quand on atteint 1 (ou 0) on s'arrête.

On additionne 10000

+ 100

+ 1

———

10101

*- Méthode des divisions par 2*

21 /2 = 10 reste 1

10 /2 = 5 reste 0

5 /2 = 2 reste 1

2 /2 = 1 reste 0

1 /2 = 0 reste 1

On prend les restes en commençant par la fin : 10101

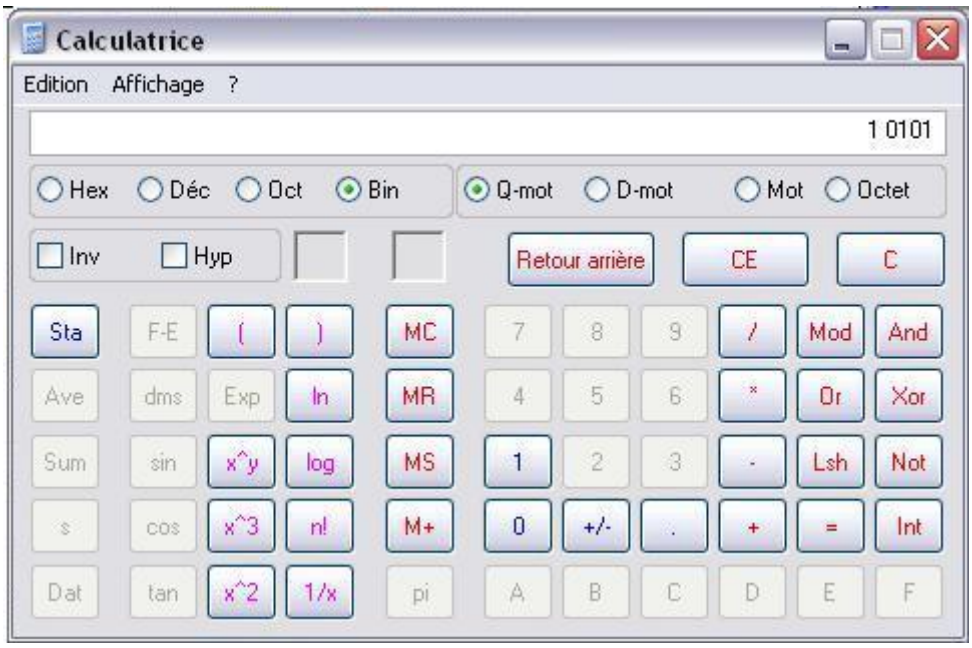
Y a-t-il une solution plus élégante ?

**La calculette Windows** permet les conversions.

1. Dans le menu **Affichage de la calculette**, cliquez sur **Scientifique.**

2. Tapez le nombre décimal que vous souhaitez convertir.

3. Cliquez sur le RadioButton 'Bin'.




Octet, Kilo méga Téra Octet.

**L'octet** (en anglais Byte ou B) est une unité d'information composée de 8 bits (256 valeurs possibles). Il permettait par exemple de stocker le code d'un caractère (une lettre ou un chiffre : 65 indiquant 'A' dans le code ASCII). Il y a quelques années les ordinateurs fonctionnaient avec des octets puis ils ont utilisé 16 bits, 32 bits et maintenant 64 bits. On voit que plus l'unité d'information contient de bits, plus elle pourra contenir des grands nombres.

**En informatique**, si 8 bits correspondent à un **octet** *(Byte)*, 16 bits est généralement appelé **mot** (en anglais *word*), 32 bits correspond à un **mot double** (en anglais double word, d'où l'appellation *dword*).

**En Visual Basic.Net**, les entiers (Integer) sont codés sur 32 bits, les Long sur 64 bits. Les valeurs sont signées (positive ou négative), un bit est donc utilisé pour le signe. Par contre UInteger est un entier non signé codé sur 32 bits pouvant donc prendre les valeurs 0 à 4 294 967 295.

**Ko, Mo, Go, To** ( *kB, MB, GB, TB en anglais*)

Pour indiquer la capacité d'une mémoire, on utilisait:

* Un kilooctet (Ko) = 210 octets = 1024 octets

* Un Mégaoctet (Mo) = 220 octets = 1024 ko = 1 048 576 octets

* Un Gigaoctet (Go) = 230 octets = 1024 Mo = 1 073 741 824 octets

* Un Téraoctet (To) = 240 octets = 1024 Go = 1 099 511 627 776 octets

Cela correspondait bien à des puissances de 2, de plus c'était en accord avec les circuits intégrés de mémoire qui avaient bien 1024 octets dans un Kilooctet.

Il parait que depuis 1998 l'IEC a décidé :

* Un kilooctet (Ko ou KB) = 1000 octets

* Un Mégaoctet (Mo ou MB) = 1000 Ko = 1 000 000 octets

* Un Gigaoctet (Go ou GB) = 1000 Mo = 1 000 000 000 octets

* Un Téraoctet (To) = 1000 Go = 1 000 000 000 000 octets

Hors de France, on utilise le nom de "byte" plutôt que le terme "octet". Cela donne les kilobyte, mégabyte, gigabyte et terabyte : (kB, MB, GB, TB avec un B majuscule)

Ne pas confondre **Byte B** (octet) et **bit** (bit ou binary digit).

Opérations

**Addition binaire**

L'addition en binaire se fait comme en décimale :

0+1 = 1

1+0 = 1

0+0 = 0

1+1 =10

Pour plusieurs digits, on additionne en commençant par les bits de droite. On a des retenues lorsque la somme de deux bits de même poids dépasse la valeur de l'unité la plus grande (dans le cas du binaire : 1), cette retenue est reportée sur le bit de poids plus à gauche…

C'est ce qui se passe avec 1+1= 10

Autre exemple

0 1 1 0 0

+ 0 1 1 1 0

- - - - - -

1 1 0 1 0

**Soustraction binaire**

La soustraction en binaire se fait comme cela :

100

- 010

_____

010

Mais pour les processeurs il est plus facile d'additionner le complément à 2.

Le complément à 1 c'est le fait d'inverser tous les bits du nombre sur toute sa longueur.

010 donne le complément à 1 suivant: 11111101

(si on travaille sur des octets, on inverse les huit bits)

Le complément à 2, c'est le complément à 1 +1: 11111101+1=11111110

Ajoutons 00000100 à 11111110 cela donne bien 00000010

(la dernière retenue tombe dans le vide)

**La table de multiplication** en binaire est très simple :

* 0x0=0

* 0x1=0

* 1x0=0

* 1x1=1

La multiplication se fait en formant un produit partiel pour chaque digit du multiplicateur (seuls les bits non nuls donneront un résultat non nul). Lorsque le bit du multiplicateur est nul, le produit partiel est nul, lorsqu'il vaut un, le produit partiel est constitué du multiplicande décalé du nombre de positions égal au poids du bit du multiplicateur.

Par exemple :

1 1 0 1 multiplicande

x 0 0 1 0 multiplicateur

- - - - - -

0 0 0 0

1 1 0 1

0 0 0 0

- - - - - -

1 1 0 1 0

On constate que pour multiplier un nombre par 2, il faut le décaler d'un bit à gauche.

10 binaire multiplié par 2 est égal à 100 (2 x 2=4 en décimal)

**Nombres négatifs**

On peut utiliser des **nombres non signés** (contenant une valeur absolue), dans un octet il peut y avoir 256 valeurs (0 à 255)

On peut utiliser des **nombres signés** (positif ou négatif), on 'code' les nombres négatifs en complément à 2 :

Le complément à 1 c'est le fait d'inverser tous les bits du nombre sur toute sa longueur.

010 donne le complément à 1 suivant: 11111101 sur un octet

(si on travaille sur des octets, on inverse les huit bits)

Le complément à 2, c'est le complément à 1 +1: 11111101+1=11111110

On se rend compte que le premier bit à gauche est à 1 pour les nombres négatifs. Dans ce cas on ne peut plus coder que 128 valeurs (sur 7 bits) pour un octet signé.

### Table de vérité

Une **table de vérité** est un tableau permettant de décrire toutes les possibilités de sorties en fonction des entrées. On place donc les variables d'entrées dans les colonnes de gauche en les faisant varier. La colonne (ou les colonnes si la fonction a plusieurs sorties) de droite décrit le résultat.

Exemple de table de vérité **de la multiplication.**

| Entrée | | Sortie |
|---|---|---|
| A | B | S |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

L'expression logique correspondante est S= A X B

On retrouve bien par exemple : si les 2 entrées sont 1 et 1 la sortie est 1 : en d'autres termes 1 X 1 =1

Exemple des tables de vérité **des fonctions logiques :**

| | Entrée | | Sortie |
|---|---|---|---|
| | A | B | S |
| OR | 0 | 0 | 0 |
| | 0 | 1 | 1 |
| | 1 | 0 | 1 |
| | 1 | 1 | 1 |
| AND | 0 | 0 | 0 |
| | 0 | 1 | 0 |
| | 1 | 0 | 0 |
| | 1 | 1 | 1 |
| XOR | 0 | 0 | 0 |
| | 0 | 1 | 1 |
| | 1 | 0 | 1 |
| | 1 | 1 | 0 |
| NOT | 0 | | 1 |
| | 1 | | 0 |

Pour la fonction And par exemple, l'expression logique correspondante est S= A AND B si les 2 entrées sont 1 et 1 la sortie est 1: en d'autres termes 1 And 1 =1

### Comment écrire une fonction logique à partir d'une table de vérité

Il est possible à partir de la table de vérité d'une fonction d'écrire l'expression algébrique de celle-ci.

Exemple 1:Soit la table de vérité suivante :

| Entrée | | Sortie |
|---|---|---|
| A | B | S |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

La sortie vaut 1 lorsque A vaut 1 et B vaut 0, l'expression logique de cette fonction est donc :

S=A AND NOT B

Exemple 2 : soit la table de vérité suivante :

| Entrée | | Sortie |
|---|---|---|
| A | B | S |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

La sortie vaut 1 lorsque A vaut 1 et B vaut 0 ou lorsque A et B sont à 0, l'expression logique de cette fonction est donc :

S=(A And Not B) Or (Not A And Not B)

En conclusion

On écrit donc les expressions pour chaque sortie à 1 (avec des And), on les combine avec des Or

Fonction logique:

**La loi AND, dite conjonction**

Elle est définie de la manière suivante : a And b est égal à 1 si et seulement si a est égal à 1 et b est égal à 1.

On peut construire la table.

| | Entrée | | Sortie |
|---|---|---|---|
| | A | B | S |
| AND | 0 | 0 | 0 |
| | 0 | 1 | 0 |
| | 1 | 0 | 0 |
| | 1 | 1 | 1 |

"l'un et l'autre"

**La loi OR, dite disjonction ou disjonction inclusive**

Elle est définie de la manière suivante : *a* Or *b* est égal à 1 si et seulement si *a* est égal à 1 **ou** *b* est égal à 1. (Notons que si *a* est égal à 1 et que *b* est égal à 1 aussi, alors *a* OU *b* est égal à 1.)

On peut construire la table:

| | Entrée | | Sortie |
|---|---|---|---|
| | A | B | S |
| OR | 0 | 0 | 0 |
| | 0 | 1 | 1 |
| | 1 | 0 | 1 |
| | 1 | 1 | 1 |

"l'un ou l'autre ou les deux"

**Le contraire, dite négation**

Le contraire de "a" égal à 1 si et seulement si *a* est égal à 0. Le contraire de a est noté Not a

| | Entrée | Sortie |
|---|---|---|
| | A | S |
| NOT | 0 | 1 |
| | 1 | 0 |

**La loi XOR, dite disjonction exclusive**

Elle est définie de la manière suivante : *a* Xor *b* est égal à 1 si et seulement si *a* est égal à 1 **ou** *b* est égal à 1 mais pas les deux. (notons que si *a* est égal à 1 et que *b* est égal à 1 aussi, alors *a* Xor *b* est égal à 0.)

On peut construire la table :

| | Entrée | | Sortie |
|---|---|---|---|
| | A | B | S |
| XOR | 0 | 0 | 0 |
| | 0 | 1 | 1 |
| | 1 | 0 | 1 |
| | 1 | 1 | 0 |

"l'un ou l'autre, mais pas les deux".

a Xor b équivalent à (a Or b) And Not( a And b)

a Xor b Xor b = a : si on applique sur a 2 fois Xor b, on retrouve a. C'est une propriété très utilisée.

**L'opérateur Équivalence**

L'équivalence (notée =) est vraie si les deux entrées ont la même valeur et faux sinon. Elle se compose comme suit :

a=b est égal à Not( a Or b) Or ( a And b)

On peut aussi dire que :

a=b est égal à Not (a Xor b)

| a | b | a = b |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Notons la notation utilisée dans les traités d'algèbre de Boole :

+ (addition) équivalent à Or

. (produit) équivalent à And

- au-dessus (négation) équivalent à Not

Ordre des évaluations

Les opérations seront soumises aux mêmes règles que les opérations "de tous les jours", **la fonction Not est prioritaire par rapport à And qui est prioritaire par rapport à la fonction Or, enfin on trouve Xor ;** on peut, pour s'aider, placer des parenthèses dans les opérations pour forcer l'ordre des opérations.

Exemple :

```
Dim a As Boolean = 0
    Dim b As Boolean = 1
    Dim c As Boolean  1

    On cherche a And b Or c

    a And b = 0
    (0 And 1 = 0)

    0 Or c = 1
    (O Or 1 = 1)
```

Le résultat est donc :

```
a And b Or c = 1
```

Les parenthèses modifient l'ordre des opérations : elles sont prioritaires sur tout :

```
a And (b Or c) = 0
```

En VB, étant donné que les opérateurs logiques et de bits ont une priorité inférieure à celle des opérateurs arithmétiques et relationnels, toutes les opérations au niveau du bit doivent être mises entre parenthèses afin de garantir une exécution précise.

**Sur un groupe de bit les opérations s'effectuent bit à bit**

Exemples

15 décimal 00001111

4 décimal 00000100

15 And 4 = 00000100 --->4 décimal

4 décimal 00000100

2 décimal 00000010

4 Or 2 = 00000110 --->6 décimal

Les lois de composition:

Ce sont des règles logiques qui permettent de simplifier l'écriture des expressions algébriques.

*Associativité :*

* (A And B)And C est équivalent à A And (B And C) et A And B And C

* (A Or B) Or C est équivalent à A Or (B Or C) et A Or B Or C

*Absoption :*

* A And (A Or B) est équivalent à A

* A Or A And B est équivalent à A

*Commutativité :*

* A And B est équivalent à B And A L'ordre est sans importance

* A Or B est équivalent à B Or A

*Distributivité :*

* A Or(B And C) est équivalent à (A Or B) And (A Or C)

* A And (B Or C) est équivalent à A And B Or A And C

*Mais aussi :*

* A Or A est équivalent à A

* À And A est équivalent à A

*Identité :*

* 1 And A est équivalent à A

* 0 Or A est équivalent à A

*Inversion :*

* A And Not A est équivalent à 0 'A ne peut pas être vrai et faux

* A Or Not A est équivalent à 1

*Nullité :*

* 0 And A est équivalent à 0

* 1 Or A est équivalent à 1

*Théorème de De Morgan*

Not (a Or b) est équivalent à Not a And Not b

Dans les deux cas, l'expression ne sera égale à 1 que si a et b sont = 0.

Not( a And b) équivalent à Not a Or Not b

Dans les deux cas, l'expression ne sera =1 que si a ou b sont =0.

**Les expressions complexes** peuvent donc être simplifiées en utilisant **des transformations :**

| Originale | Transformation |
|---|---|
| Not A And Not B | Not (A Or B) |
| Not A And B | Not (A Or Not B) |
| A And Not B | Not (Not A Or B) |
| A And B | Not (Not A Or Not B) |
| Not A Or Not B | Not (A And B) |
| Not A Or B | Not (A And Not B) |
| A Or Not B | Not (Not A And B) |
| A Or B | Not (Not A And Not B) |

**Il existe aussi plusieurs autres opérateurs qui n'ont pas d'équivalent en Visual Basic Net**

Ils existaient en VB6 !!

**Implication**

L'implication (notée IMP) qui n'existe pas en VB.Net s'écrit de la manière suivante :

a IMP b peut s'écrire en VB: Not(a) Or b

Cette opération n'est pas commutative a est une condition **suffisante** pour b, qui, elle, est une condition **nécessaire** pour a.

| IMP | | |
|---|---|---|
| **a** | **b** | **a IMP b** |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Inhibition**

L'inhibition (notée INH) n'existe pas en VB.Net, elle se compose comme suit :

a INH b peut s'écrire en VB: a And Not(b)

Cette opération n'est pas commutative.

| INH | | |
|---|---|---|
| a | b | a And Not b |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

### Déplacement de bit

Les opérateurs binaires << et >> effectuent des opérations de déplacement de bits.

**L'opérateur** << **décale à gauche les bits** du premier opérande du nombre de positions spécifié. Les bits de poids fort situés en dehors de la plage du type de résultat sont éliminés, et les positions libérées par les bits de poids faible sont remplies par des zéros.

**L'opérateur** >> **décale à droite les bits** du premier opérande du nombre de positions spécifié. Les bits de poids faible sont éliminés et, si l'opérande de gauche est positif, les positions libérées par les bits de poids fort sont mises à zéro ; s'il est négatif, elles sont mises à un. Si l'opérande de gauche est de type **Byte**, les bits de poids fort disponibles sont remplis par des zéros.

À quoi cela sert ?

Exemple décaler à gauche un Byte revient à faire une multiplication par 2.

3 en décimal= 11

Je décale à gauche, j'obtiens 110 , c'est 3*2=6 en décimal.

**Revenons sur la base hexadécimale**

**En hexadécimal**

On a 16 caractères : 0, 1, 2, 3 ,4 …8, 9, A, B, C, D, E, F.

Quand on compte et qu'on arrive à F (15 décimal), on passe à 10 (16 décimal) puis 11…

Voyons la correspondance décimale, hexadécimale, binaire :

| Base décimale | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Base hexadécimale | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| Base binaire | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |

Pour un nombre hexadécimal à plusieurs chiffres, le poids de chaque chiffre est :

| Poids en décimal : | $16^3$ | $16^2$ | $16^1$ | $16^0$ |
|---|---|---|---|---|
| Nombre hexadécimal: | 1000 | 0100 | 0010 | 0001 |

1C en base 16 c'est donc 10+C en hexadécimal = en décimal c'est $16^1$ + 12x $16^0$ = 16 + 12 = 28

Le nombre 28 (en base 10) vaut en base 16 : $1*16^1 + 12*16^0 = 1*16^1 + C*16^0$

c'est-à-dire 1C en base 16.

Le nombre FB4 (en base 16) vaut en base 10 : $F*16^2 + B*16^1 + 4*16^0$ = 3840 + 176 + 4 = 4020

À quoi sert la base hexadécimale ?

C'est une représentation plus imagée de la représentation binaire.

Pour convertir un octet en hexadécimale, on le partage en 2 groupes de 4 bits, qui correspondent chacun à un chiffre hexadécimal.

00101010 c'est un octet en binaire; impossible à retenir en binaire (en décimal on ne voit pas du tout ce qu'il représente en bits). Cet octet, on le coupe en 2, chaque demi-octet représente 4 bits dont la valeur est comprise entre 0 (0000 en binaire) et F (1111 en binaire, 15 en décimal)

00101010 en binaire= 2A en hexadécimal.

| 2 | A |
|---|---|
| 0010 | 1010 |

Il suffit de se souvenir des nombres de 1 à 15 en binaire pour se représenter rapidement 2A.

Autre exemple :

| D | 5 |
|---|---|
| 1101 | 0101 |

HFF = 255 décimal

HFFFF=65535 décimal

Notons que pour signifier qu'on a affaire à un nombre hexadécimal, on ajoute H devant.

**L'hexadécimal est donc une manière rapide et mnémotechnique de se représenter des nombres binaires.**

**Les viewers et éditeurs permettant de voir et modifier les octets contenus dans un fichier affichent les octets en hexadécimal (voir plus bas).**

## V-W-3 - Pratique en Visual Basic

**En Visual Basic.Net**, on a à notre disposition :

- les Boolean qui peuvent prendre les valeurs True ou False… ;

- les entiers : Byte contient 8 bits (non signé) pouvant prendre les valeurs 0 à 255 : il existe aussi en VB 2005, SByte contenant un octet signé dont les valeurs varient de moins 128 à plus 127 ;

- les Short 16 bits, les Integer sont codés sur 32 bits, les Long sur 64 bits. Ces valeurs sont signées (positives ou négatives), un bit est donc utilisé pour le signe.

Par contre en VB 2005, UInteger est un entier non signé codé sur 32 bits pouvant donc prendre les valeurs 0 à 4 294 967 295. Ushort et ULong existent aussi. (U comme Unsigned)

Il existe aussi les nombres en virgule flottante ou fixe (Single, Double, Decimal), ceux-là, on ne les utilisera pas pour travailler sur les bits.

**Littéral**

Un littéral est censé être en base décimale.

Dim A As Integer = 12 (12 est en base décimale)

On peut forcer un littéral a être un hexadécimal ou un octal : un nombre hexadécimal est noté avec le préfixe &H , exemple :

A=&HFF

(&O pour octal)

Il n'est pas possible de saisir un littéral en binaire.

Bien comprendre que, en interne, les entiers sont codés en binaire : c'est normal, la mémoire de l'ordinateur et les registres des processeurs sont composés d'octets de 8 bits de Dword de 16 bits et maintenant de 32 et 64 bits contenant des bits positionnés à 0 ou 1.

Dim A As Integer = 12

c'est **12** est en base décimale, c'est notre manière de l'utiliser.

mais c'est **0000000000001100** en mémoire physique, représentation binaire.

c'est **000C** en hexadécimal, mais dans une autre base plus pratique, "plus imagée".

C'est toujours le même nombre !!

**And Or Xor AndAlso, OrElse**

En VB, en plus de And, Or, Xor, existent AndAlso et OrElse qui testent la première valeur puis, si nécessaire, la seconde. Si la seconde valeur n'a pas à être évaluée, est ne le sera pas, c'est un gain de temps.

Pourquoi Xor n'a pas d'équivalent ?

Car pour évaluer Xor, il faut d'emblée utiliser les 2 valeurs.

**Comment réagit VB avec les booléens, les entiers ?**

* **Si A et B sont des expressions booléennes** (valeur True ou False uniquement) :

IF A>=2 And A<=5 Then…

**Les expressions booléennes sont évaluées** et on a comme résultat True ou False.

* **Si A et B sont des nombres entiers** (Integer= 32 bits, Long=64 bits, Short=16 bits, Byte=8 bits par exemple).

**L'opération est effectuée sur chaque bit de l'équivalent binaire, le résultat binaire est affiché en décimal.**

A = 7 'en décimal ( 0111 en binaire)

B = 12 'en décimal ( 1100 en binaire)

A And B = 4 'en décimal ( 0100 en binaire)

* **Si A et B sont des nombres en virgule flottante** (Single, Double par exemple), ils sont codés sous la forme de mantisse plus exposant, une opération logique devrait produire un résultat aberrant. Ils sont convertis en Long avant évaluation si Option Strict= Off :

Dim a As Single = 7

Dim b As Single = 12

Dim c As Boolean = True

MsgBox(a And b) 'affiche '4', car 7 et 12 sont transformés en Long (si Option Strict=Off)

si Option Strict=On Levée d'une exception.

**Un conseil : utilisez des Booléens quand vous voulez effectuer des opérations logiques, des entiers quand vous travaillez sur les bits.**

### Conversion Binaire, hexadécimale, décimal

L'affichage d'un entier se fait en décimal par défaut si on utilise la méthode ToString :

```
Dim a As Integer =&HFF   Littéral en hexadécimal

MsgBox (a.ToString)      Affiche  '255' décimal
```

On peut surcharger la méthode et afficher en hexadécimal :

```
Dim a As Integer = 255

MsgBox(a.ToString("X"))  Affiche 'FF'
```

On peut afficher en hexadécimal et décider le nombre de digits affichés :

```
Dim a As Integer = 255

MsgBox(a.ToString("X4")) Affiche '00FF'
```

En utilisant la classe Convert, on peut même afficher en base binaire, octale, décimale, hexadécimale.

Convert.ToString(Int64, base) Convertit la valeur d'un entier signé 64 bits en sa représentation String équivalente dans une base 'base' spécifiée (base 2, 8, 10, 16).

Dim d As Long = 3

Dim r As String

r= Convert.ToString(d, 2)) 'convertit la valeur Long de "d" en base 2

d= 3 donne r="11" binaire

Si on avait eu une String "3" on l'aurait convertie en Long grâce à CLng(d)

Convert.ToInt64(s, base) 'convertit en type int64(long en base 10) la valeur de la String 's' à partir d'une base 'base'.

Dim d As String = "111"

Dim r As Long

r= Convert.ToInt64(d, 2)) 'convertit d (string représentant un binaire) en Long

cela donne r=7

Enfin dans l'espace Visual Basic l'instruction Hex donne la représentation hexadécimale d'un nombre , ( Oct existe aussi)

str=Hex(456) retourne 1C8

### Conversion String en Byte

Conversion String en Bytes :

```
Dim str As String= "…"
Dim encoding As New System.Text.ASCIIEncoding()
Dim Bytes() As Byte()= encoding.GetBytes(str)
```

Conversion Bytes en String :

```
Dim Bytes As Byte() = ….
Dim str As String
Dim enc As New System.Text.ASCIIEncoding()
str = enc.GetString(Bytes)
```

### Enregistrement d'un tableau de Bytes

*1 - En mémoire : (dans un mémoryStream)*

```
Imports System.IO

…

' Creation d'un tableau de Byte.
Dim dataArray(1000) As Byte

' On le remplit avec des nombres aléatoires.
Dim randomGenerator As New Random
randomGenerator.NextBytes(dataArray)
```

Écriture

```
Dim binWriter As New BinaryWriter(New MemoryStream())

' Écrire les données dans la mémoire.

binWriter.Write(dataArray)
```

Lecture

```
' Créer un reader en utilisant le stream du Writer
Dim binReader As New BinaryReader(binWriter.BaseStream)

' mettre la position au début du stream.
```

```
binReader.BaseStream.Position = 0

' Relecture dans verifyArray.
Dim verifyArray() As Byte = binReader.ReadBytes(dataArray.Length)
```

*2 - Dans un fichier :*

```
…

Dim fs As New FileStream(FILE_NAME, FileMode.CreateNew)

Dim w As New BinaryWriter(fs)
w.Write(datArray)

w.Close()
fs.close
```

Le Framework 2 permet une écriture encore plus simple pour lire écrire les octets d'un fichier:

Lire et mettre dans un tableau les octets d'un fichier ? (Framework 2)

Dim myBytes() As Byte =File.ReadAllBytes("c:\monText.txt")

Il existe aussi WriteAllByte.

**Précision sur 'If a Then'**

**Avec des valeurs numériques** si

* a=0, a est évalué comme False le programme se poursuit après Then;

* si a est différent de 0 (1, -1, 45…) a est considéré comme True et le programme se poursuit après Then.

Donc

```
Dim a As Integer =15

If a  Then…
```

C'est une mauvaise méthode !! Il vaut mieux écrire :

```
Dim a As Integer =15

If a <>0 Then…
```

Avec une expression booléenne par contre, on peut écrire :

```
Dim a As Boolean= True

If a = True Then…

    'ou

If a Then…
```

Exemple :

```
If (x=15)=True  Then…

    'ou
```

```
If x=15 Then…
```

Avec une expression booléenne, et uniquement avec une expression booléenne, il est possible de se passer du = True après un If, car de toute façon, l'expression est évaluée.

### Masque de bit

**Or permet d'écrire un bit à 1**

Soit un entier, on veut forcer un des bits à 1 , la solution est de faire un Or avec un entier ayant ce seul bit à 1.

Exemple : Mettre le deuxième bit de 00000100 (4 en décimal) à 1

Il faut faire un Or avec 00000010.

Le poids du deuxième bit est 2, c'est le 'mask bit'.

4 Or 2 = 6

00000100 Or 00000010 = 00000110

En faisant Or 2, on a bien mis le deuxième bit à 1.

Le masque est toujours une puissance de 2.

Or 8 met le quatrième bit à 1

**And permet de tester un bit**

A And 1 indique si le bit le moins significatif (le plus à droite) est a 1

Exemple: si A = 7 'en décimal ( 0111 en binaire) A And 1 retourne 1

À And 8 indique si le quatrième bit est a 1 (8 est le poids du quatrième bit).

Exemple: si A = 7 'en décimal ( 0111 en binaire) A And 8 retourne 0

8 c'est le 'mask bit' (00001000) du quatrième bit.

A And 8 peut ensuite être évalué comme une expression booléenne:

If A and 8 Then ' Si le quatrième bit de A est à 1 alors,

**And permet aussi de forcer à 0 une partie des bits et de ne conserver que la valeur de certains bits**

Soit une couleur codée sur 24 bits, les 8 bits à droite codent la composante bleue, Je veux conserver uniquement ces 8 bits de droite (l'octet de droite) :

myColor And &HFF conserve le premier octet, mais met les 2 autres à 0 :

MyColor=0010 0100 1000 0010 0010 0110

And 0000 0000 0000 0000 0000 1111

= 0000 0000 0000 0000 0000 0110

Le masque correspond aux bits à conserver.

**Xor permet de forcer un bit à 0**

Pour mettre le 4e bit à 0

Il faut faire un Xor avec 00001000.

Le poids du deuxième bit est 2, c'est le 'mask bit'.

12 Xor 8 = 4

00001100 Or 00001000 = 00000100

**Exemple pratique**

**Comment stocker plusieurs valeurs dans une variable en utilisant un masque.**

Souvent, plutôt que de coder une information par octet, on peut coder une information par bit et ainsi coder 8 informations par octet.

Le paramètre d'une fonction est, par exemple, composé d'un entier ou chaque bit à une signification.

Exemple fictif :

00000001 le premier bit à 1 signifie gras (1 en décimal)

00000010 le deuxième bit à 1 signifie l'italique (2 en décimal)

00000100 le troisième bit à 1 signifie le soulignage. (4 en décimal)

Si je veux envoyer les paramètres gras et souligné, j'enverrais le paramètre 1 Or 4 qui correspond a 00000101. Les bits 1 et 3 sont bien à 1.

On note bien que chaque paramètre doit être une puissance de 2.

C'est plus clair de créer une énumération :

```
<Flags()> Enum, Car

    Normal=0

    Gras= 2

    Italique= 4

    Souligne= 8

End Enum
```

Si je veux envoyer les paramètres gras et souligné, j'enverrai le paramètre Car.Gras Or Car.Souligne

<Flags()> indique qu'on travaille bien sur des bits.

Souvent les valeurs sont proposées par VB, par exemple quand on utilise MsgBox, le deuxième paramètre qui indique le style peut comporter plusieurs indications séparées par des Or :

```
reponse= MsgBox(msg, MsgBoxStyle.DefaultButton2 Or  MsgBoxStyle.Critical Or MsgBoxStyle.YesNo,
 Title)
```

Pour lire un bit en retour d'une fonction, on utilisera And :

```
Si reponse And Car.Italique =1
```

c'est que le second bit de réponse est à 1.

Bien que ce soit une opération sur les bits on écrit souvent :

```
If reponse And Car.Italique Then….
```

### Cryptage simple par Xor

La technique la plus simple est d'appliquer un "OU exclusif" (XOR) entre le texte à chiffrer et la clé.

Pour obtenir le message crypté on effectue Message Xor Cle (si la clé fait x octets on effectue le Xor entre le premier octet du message et le premier de la clé, puis le deuxième… quand on arrive à x+1 caractère du message, on recommence au premier caractère de la clé).

Comme Message Xor Cle Xor Cle =Message, pour déchiffrer le message codé, il suffit de faire de nouveau un Xor avec la clé.

La clé est donc la même pour coder et décoder, on appelle cela une clé symétrique.

Bien sûr, si on utilise un texte comme clé et comme message, c'est le code du caractère qui est utilisé.

### Travail sur les couleurs

Les valeurs RVB (couleurs) sont stockées dans trois octets de 8 bits, conduisant à une couleur à 24 bits, chaque octet correspondant respectivement au rouge, au vert et au bleu.

rrrr rrrr vvvv vvvv bbbb bbbb

**Comment récupérer la composante rouge verte ou bleue ?**

```
Dim myColor As Integer = 15963245
'Un Integer a 32 bits , les 24 premiers sont utilisés.

Dim R, V, B As Byte
```

Pour le rouge :

```
R = myColor >> 16
```

On décale de 16 bits vers la droite: 0000 0000 0000 0000 rrrr rrrr

Pour le vert :

```
V = (myColor And &HFF00) >> 8
```

On fait un And &HFF00 ce qui met le premier et le troisième octet à 0 0000 0000 vvvv vvvv 0000 0000

On décale de 8 bits vers la droite: 0000 0000 0000 0000 vvvv vvvv

Pour le bleu :

```
B = (myColor And &HFF)
```

On fait un And &HFF ce qui met le premier et le second octet à 0

0000 0000 0000 0000 bbbb bbbb

(En Vb on peut faire plus simple)

### Travail sur les graphiques

Un mode souvent utilisé pour la réalisation d'interfaces est **le mode XOR**. Ce mode permet d'effacer facilement un cadre de sélection en le redessinant une seconde fois à la même position.

Si l'on a un écran noir et blanc pour lequel 1 = noir et 0 = blanc et que l'on affiche une forme en noir, chaque pixel appartenant à la forme est inversé à l'écran puisque 1 xor p = not p. Donc si l'on dessine la forme deux fois, chaque pixel est inversé deux fois et revient donc dans son état initial.

Par contre, sur un écran couleur, les résultats sont imprévisibles. Si le noir est représenté par la valeur de pixel 1111 et que l'on dessine en xor sur un pixel de valeur 1001, le résultat est un pixel de valeur 1111 xor 1001 = 0110. La couleur résultante est alors imprévisible : on obtient un effet "technicolor".

En VB on a d'autres fonctions sur les graphiques.

## V-W-4 - Viewer hexadécimal

Comment voir le contenu d'un fichier en hexadécimal ?

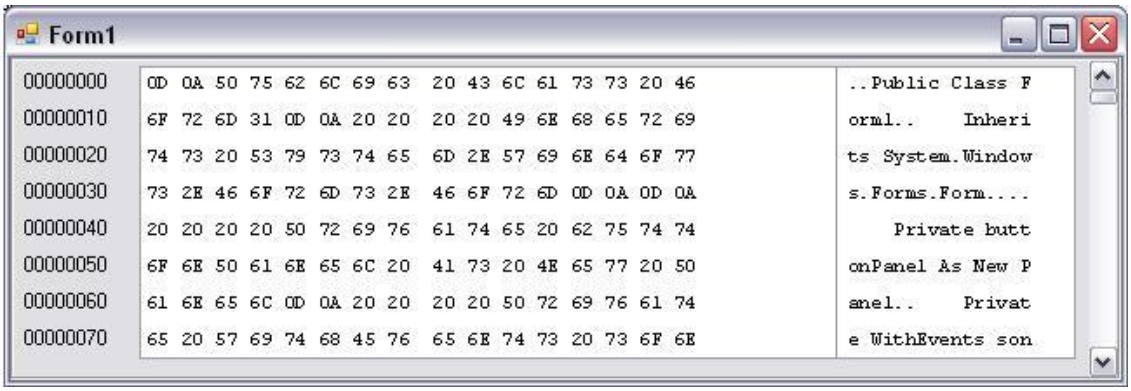
C'est très simple et VB 2005 :

on utilise un composant ByteViewer ;

charger la référence System.design.Dll ;

puis entrer le code dans Form_Load :

```
Private Sub Form1_Load() Dim viewer As New System.ComponentModel.Design.ByteViewer()

Dim viewer As New System.ComponentModel.Design.ByteViewer

Me.Controls.Add(viewer)

viewer.Dock = DockStyle.Fill

Dim ofd As New OpenFileDialog 'Choix d'un fichier

If ofd.ShowDialog() = Windows.Forms.DialogResult.OK Then viewer.SetFile(ofd.FileName)

End Sub
```




Si vous avez déjà un tableau de bytes, utilisez sa méthode SetBytes.

Vous pouvez même choisir son mode d'affichage (Ansi, Unicode, Hexadump ou automatique) avec sa méthode SetDisplayMode.

**Second exemple**

Ouvrir un fichier image .jpg le charger dans un tableau de Bytes et l'afficher :

```
Dim viewer As New System.ComponentModel.Design.ByteViewer

Me.Controls.Add(viewer)

viewer.Dock = DockStyle.Fill

Dim ofd As New OpenFileDialog

ofd.ShowDialog()

Dim img As Image = Image.FromFile(ofd.FileName)

Dim mstImage As IO.MemoryStream = New IO.MemoryStream

img.Save(mstImage, System.Drawing.Imaging.ImageFormat.Jpeg)

Dim bytImage As Byte() = mstImage.GetBuffer

viewer.SetBytes(bytImage)
```

Fait à partir d'un **article de c2i** en C#.

## V-W-5 - Éditeur hexadécimal

On peut écrire en VB.Net un éditeur hexadécimal de fichier (lecture du fichier, visualisation en hexa et ASCII, modification d'un octet.

Voir le lien suivant :

**Editeur hexadécimal ULTRA EDIT de fichier par VBsorcier**

Pour que le source marche, ne pas oublier de générer puis mettre les fichiers vb dans MyProjet et les fichiers ressources dans le répertoire de ressources.

## V-X - Les génériques

Super complexe ? Non !!

## V-X-1 - Définition

À partir de VB 2005, on peut utiliser les génériques.

Un type générique (Generic) permet de créer une Classe ou une procédure, ayant des Data Types non définis au départ.

En d'autres termes, les paramètres et variables n'ont pas de type: ce ne sont pas des Strings, des Integers… Ce sont des génériques. Quand on utilise la Classe ou la procédure, on indique le type.

Les génériques nous permettent de définir un comportement ou un algorithme commun sur les types ou un sous-ensemble de types .Net. Ils sont un moyen de mutualiser un comportement.

Par exemple, je vais écrire une routine de calcul avec des génériques, elle sera utilisable avec des Integers, des Single…

**Exemple de Fonction utilisant un 'generic'.**

Permettant d'en comprendre l'intérêt.

Créons une sub nommée Swap (elle sert à intervertir 2 variables) fonctionnant pour tous les types de données :

```
Private Sub Swap(Of ItemType) (ByRef v1 As ItemType, ByRef v2 As ItemType)

Dim temp As ItemType

temp = v1

v1 = v2

v2 = temp

End Sub
```

Notons que en plus des 2 paramètres v1 et v2 à 'swapper' ,"Of ItemType" indique le type de donnée qui doit être utilisé.

Si on a 2 entiers à swapper, il faut appeler la fonction Swap comme cela :

```
Swap(Of Integer)(v1, v2)
```

Si ce sont des Strings :

```
Swap(Of String)(v1, v2)
```

Le JIT compile la fonction Swap comme si elle avait été écrite pour des Strings.

Sans les génériques j'aurais fait plusieurs routines de code pour chaque Type. Or en utilisant les génériques cette redondance peut être évitée.

**Exemple de Classe utilisant un 'generic'.** À revoir quand vous connaitrez les classes.

De la même manière, on peut créer une Classe entièrement générique :

```
Public Class SomeClass(Of ItemType)

Private internalVar as ItemType    ' variable generic
```

```
Public Function SomeMethod(ByVal value As ItemType) As ItemType

'Fonction acceptant un generic comme paramètre
…
End Function

End Class
```

**Exemple de Collection utilisant un 'generic'**

On peut créer une collection générique (System.Collections.Generic) et lui imposer un type.

Exemple : créons une collection de String : List(Of String).

```
Imports System.Collections.Generic.


Dim l As New List(Of String)

l.Add("toto")    'On ajoute une string

Dim S As String = l.Item(0) ' l'item est bien typé : même avec 'Option Strict=on'
'pas besoin de CType.
```

Habituellement les collections contiennent des objets; ici c'est une collection de String.

Je ne peux y mettre que des String (sinon cela provoque une erreur).

Comme par définition c'est des string, il n'y a pas de conversion String=>Objet et Objet=>String (pas de boxing/unboxing)

On peut aussi créer des Stack(Of…) Queue(Of…), Dictionnary(Of…) SortedList(Of…)…

## V-X-2 - Intérêts des génériques ?

Pourquoi ne pas utiliser des types 'Object' à la place des génériques ?

Les génériques sont **fortement typés**. Si on crée une collection générique de Long, on ne peut utiliser que des Long : c'est mieux, cela évite les erreurs, les conversions de type.

Ils sont plus rapides que l'usage des objets.

S'il y a erreur, elle se produit probablement à la compilation et pas à l'exécution.

Cela permet d'utiliser l'intellisense.

**Comparaison ArrayList (non générique) et List (Of) générique.** Si on utilise une ArrayList qui est une liste non générique, on peut sans problèmes ajouter un Integer puis une String : cela n'est pas logique et possiblement une erreur.
De plus quand on travaille sur cette liste ou qu'on parcourt cette liste il y a des opérations de Cast et de boxing/unboxing sans arrêts: on stocke des Integer, String dans des objets.
Par contre si on utilise une List générique typée, pas de Cast ni de boxing (rapidité++) et réduction du nombre d'erreurs possibles: une List de String ne peut contenir que des String.

## V-X-3 - Usage des génériques

**On peut utiliser des méthodes génériques pour travailler sur les tableaux.**

Exemple recherche dans un tableau de short nommé monTab l'élément 2

```
index= Array.indexOf (Of Short)(monTab, 2)
```

est hyper plus rapide que

index= Array.indexOf (monTab, 2), car la première version avec génériс est directement optimisée pour les Short.

Il est de même pour Binarysearch et Sort.

Cela est valable pour les types 'valeur' (peu d'intérêts pour les strings par exemple).

**Collections génériques**

On peut créer une collection générique (System.Collections.Generic) et lui imposer un type.

Exemple : créons une collection de String (**List(Of String)**): Elle est typée, car elle ne peut contenir que des 'String'.

```
Dim l As New List(Of String)

Il s'agit d'une List avec Index.

l.Add("toto")     'On ajoute une string

Dim S As String = l.Item(0) ' l'item est bien typé : même avec 'Option Strict=on'
'pas besoin de CType.
```

Il y a aussi de nouveaux types de collections génériques :

- les **Dictionnary(Of…)** avec Clé et valeur ;

- les **SortedDictionnary(Of…)** avec Clé et valeur triés ;

- les **LinkedList(Of…)** Liste Chainée, chaque élément comportant une propriété Value, Next et Previous ;

- les **SortedList(Of…)**… ;

- les **Stack(Of…)** ;

- les **Queue(Of…)**.

On peut aussi créer des collections 'composées'.

```
Dim genericColl As New Dictionary(Of String, String)
genericColl.Add("PremiereClé", item1)
```

## V-Y - Linq

'Language-Integrated Query' (**LINQ**), veut dire "langage de requête intégré".

On l'utilise dans VB à partir de VB2008 (Framework 3.5).

## V-Y-1 - Définition, mise en place

C'est un langage de requêtes (permettant d'interroger une source de données) directement dans le code Visual Basic et à l'aide de mots-clés familiers (issues du SQL, le langage d'interrogation des bases de données).

Cette source de données peut être une Base de données (Linq To SQL et Linq To DataSet )un fichier XML (Link To XML), mais aussi une **collection**, un **tableau**, une chaine **de caractères**.

On parle dans ce dernier cas de '**Linq To Objects**'. Si un objet prend en charge l'interface IEnumerable ou IEnumerable (Of), le fournisseur LINQ to Objects vous permet de l'interroger.

LINQ (dixit Microsoft) offre trois principaux avantages par rapport aux boucles for Each traditionnelles.

Les requêtes

- Elles sont plus concises et lisibles, surtout lors du filtrage de plusieurs conditions.

- Elles fournissent des fonctions puissantes de filtrage, de classement et de regroupement avec un minimum de code d'application.

- Elles peuvent être appliquées à d'autres sources de données avec peu ou pas de changement.

Pour que LINQ soit pris en compte, il faut :

utiliser **VB 2008** et le **framework 3.5**.

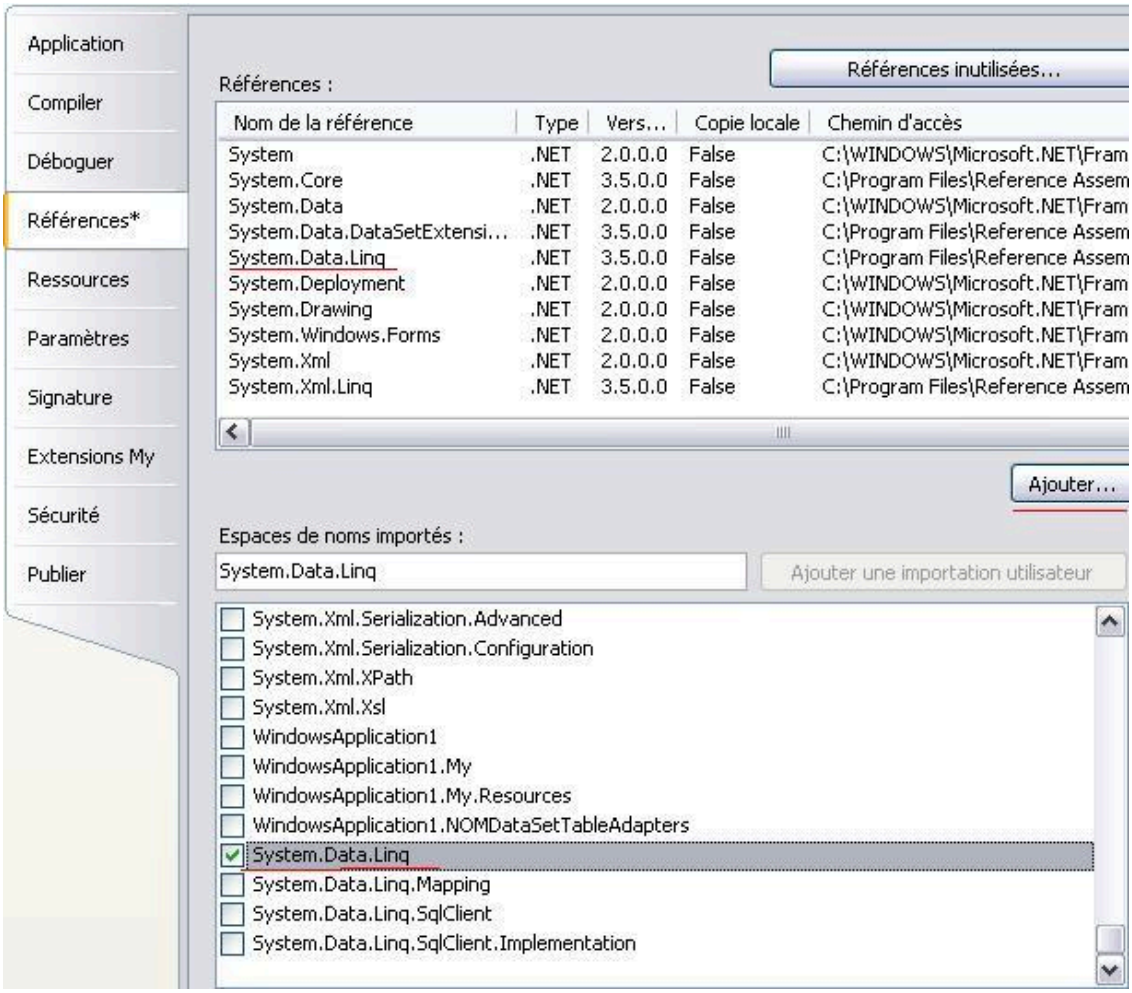
Dans les propriétés, onglet compile, il faut que **Option Infer=On**

Il faut ajouter **System.Data.Linq**.

Si vous créez un nouveau projet dans VB 2008, toutes les conditions sont effectives par défaut, si vous modifiez un ancien projet, il faut rajouter certaines références.

Dans l'Explorateur de solutions (Projet, Propriétés…), cliquez sur Références, puis cliquez sur Ajouter une référence.

Cliquez sur .NET, sur l'assembly System.Data.Linq, puis sur OK, cela ajoute la référence.




Il faut ajouter l'espace de nom.

Dans l'explorateur de solution, cocher Systel.Data.Link comme ci-dessus
ou ajouter les directives suivantes en haut du Module1 : Imports System.Data.Linq

## V-Y-2 - Principe d'une requête Linq

À titre d'exemple simpliste, on a des données dans MyData et chaque donnée a les champs 'Nom', "Prenom", "Ville"…
Comment chercher les enregistrements ayant comme nom "toto" ?

```
Dim Resultat = From Element In MyData _

Where Element.Nom = "Toto" _

Select Element
```

On crée une **variable de requête** (ici ' Dim Resultat') qui sera chargée de contenir la requête (et pas les résultats),

puis l'expression de requête composée de :

**From :** dans quoi chercher ? Dans quel élément ?

**In :** quelle source de données ? Dans MyData ;

**Where :** précise les conditions à appliquer, c'est le 'filtre' ;

**Select :** précise les éléments à extraire qui vont apparaitre dans 'Resultat'.

Remarquons que Dim From In Where Select doivent être sur une seule unique et même ligne; pour la lisibilité, on écrit sur plusieurs lignes en ajoutant des continuateurs de lignes "_".

Remarquons aussi qu'initialement on connait MyData et on sait que chaque élément de MyData a un champ 'Nom', c'est tout !! On utilise dans la requête les nouvelles variables 'Resultat' et 'Element' sans avoir à déclarer leurs types (on aurait pu le faire). 'Element' est une variable de portée déduite comme élément de MyData.

Ce fonctionnement particulier de LINQ est possible grâce à l'inférence de type et aux types anonymes (voir plus bas).

Et pour afficher les noms dans une ListBox :

```
For Each P In Resultat

   ListBox1.Items.Add(P.NOM )

Next
```

Ici la requête contenue dans la variable de requête 'Resultat' est exécutée pour 'alimenter' la boucle 'For Each'. On remarque donc que l'**exécution est différée**.

On peut 'forcer' l'**exécution immédiate** en mettant la requête entre parenthèses et en utilisant une propriété (.Count , .ToArray, .ToList ) :

```
' Execution immédiate avec ToList.
Dim ListPersonneAyantPrenomToto = (From Element In MyData _
Where Element.Nom = "Toto" _
Select Element).ToList()
' On retrouve la liste des éléments de MyData ayant le Prenom='Toto"

' Execution immédiate avec Count.
Dim NombrePersonneAyantPrenomToto = (From Element In MyData _
Where Element.Nom = "Toto").Count()
' On a compté le nombre d'éléments ayant  pour Prenom="Toto".
'NombrePersonneAyantPrenomToto contient le résultat
```

On peut aussi utiliser .ToList ou .ToArray en mode différé :

```
' Execution différée .
Dim Resultat = From Element In MyData _
Where Element.Nom = "Toto" _
Select Element
' . . .
Dim Tableau = Resultat.ToArray()
```

**Order By** permet de trier les résultats.

```
Dim Resultat = From Element In MyData _

 Order By Element.Price Descending, Element.Nom _

 Select Element.Nom, Element.Price
```

Ici on trie par prix décroissant, puis à prix égal sur le nom croissant.

Remarquons qu'on sélectionne seulement 2 'colonnes'.

Il est possible d'avoir plusieurs sources, dans ce cas chaque bloc .In est séparé par une virgule :

```
Dim queryResults = From cust In customers, ord In orders _
                   Where cust.CustomerID = ord.CustomerID _
                   Select cust, ord
```

La clause Where peut contenir des conditions complexes avec des AND des OR…

```
Dim custs = From cust In db.Customers _
            Where cust.Country = "France" _
                And (cust.CompanyName.StartsWith("F") _
                Or cust.CompanyName.StartsWith("V")) _
            Order By cust.CompanyName _
            Select cust.CompanyName, cust.City

DataGridView1.DataSource = custs
```

## V-Y-3 - Link et les tableaux d'Integers

Un tableau peut être interrogé par Linq.

Exemple : rechercher les nombres pairs dans un tableau d'Integer :

```
' La Data source: c'est un tableau d'Integer

Dim numbers() As Integer = {0, 1, 2, 3, 4, 5, 6}



' Création de la requête.

'Pour chaque élément num dans la source

'Si l'élément num est tel que num Mod 2=0 (condition pour qu'il soit pair)

'Selectionner num et le mettre dans réponses

 Dim réponses = From num In numbers _

                Where num Mod 2 = 0  _

                Select num


' Exécution de la requête.

' On utilise les réponses

For Each number In réponses

    Console.Write(number & " ")

Next
```

Cela affiche sur la console (menu Affichage puis Sortie) : 0 2 4 6

On peut vouloir compter uniquement les nombres pairs :

```
Dim nombredepair = (From num In numbers _

                    Where num Mod 2 = 0 _

                    Select num).Count()

Console.Write(nombrepair) 'pour afficher 4
```

On remarque que dans le premier exemple (Select num) l'exécution de la requête est effectuée au cours de la boucle For Each (exécution différée par rapport à la création) alors que dans le second exemple (count), l'exécution est immédiate.

## V-Y-4 - Link et les chaines de caractères

Soit une chaine de caractères MyString, rechercher les caractères qui sont des nombres.

```
' Un string

Dim MyString As String = "ABCjkjhkhs666KMOOP"

' Select les caractères qui sont des nombres
Dim Query = From ch In MyString _
Where ch.IsDigit(ch) _
Select ch


' Exécution de la requête
For Each c As Char In Query
  Console.Write(c & " ")
Next

' Combien y a-t-il de nombres?
Dim count As Integer = Query.Count()
Console.WriteLine("Count = " & count)
```

On remarque qu'il n'est pas nécessaire de réexecuter la requête.

**Autre syntaxe**
Sélectionner tous les caractères avant '6'

```
Dim Query2 = MyString.TakeWhile(Function(c) c <> "6")

' Execute the second query
For Each ch In Query2
 Console.Write(ch)
Next
```

Ici on a utilisé **TakeWhile** qui sélectionne les caractères jusqu'à 6. (les sélectionne une seule fois). On a utilisé une expression lambda (voir le chapitre sur les expressions lambdas).

```
Dim Query2 = MyString.Except("6")
```

Ici on a utilisé **Except** qui sélectionne les caractères sauf 6.

## V-Y-5 - Link et les mots d'une chaine de caractères

Rechercher combien de fois une String contient le mot 'Basic' :

```
Dim text As String = "Ceci est un cours Visual Basic" & _
" pour les débutants et les autres"

Dim searchTerm As String = "Basic"

' Conversion de la String en Tableau de mots:.
Dim dataSource As String() = text.Split(New Char() {" ", ",", ".", ";", ":"}, _
StringSplitOptions.RemoveEmptyEntries)

' Création et exécution de la requête
' Utiliser ToLower pour  trouver "Basic " et "Basic"
```

```
Dim Query = From word In dataSource _
Where word.ToLowerInvariant() = searchTerm.ToLowerInvariant() _
Select word

' Compter les 'Basic'.
Dim count As Integer = Query.Count()
Console.WriteLine(count )
```

## V-Y-6 - Link pour rechercher la différence entre deux listes de noms

Rechercher dans la String nom1, les noms qui ne sont pas aussi dans nom2.

```
' Soit 2 tableaux de Sting
Dim nom1 As String() =  {"Philippe", "Paul"}
Dim nom2 As String() = {"Paul", "Jean"}



' Créer la requête.
Dim difference = nom1.Except(nom2)



' Executer
For Each name As String In difference
 Console.WriteLine(name)
Next
```

Affiche 'Philippe'

## V-Y-7 - Link et les contrôles

Comment obtenir la liste des contrôles actifs dans un formulaire ?

```
Dim ControlsEnabled = _
            From c In Me.Controls _
            Where CType(c, Control).Enabled _
            Select CType(c, Control)
```

On se rend bien compte que, ici, Linq est une alternative à For Each.

## V-Y-8 - Inférence de Type et type anonyme

Débutant, tu peux sauter !!

On a vu que Linq fonctionnait d'une manière un peu particulière. Pour mieux comprendre cela, il faut savoir qu'à partir de VB 2008 on peut utiliser l'inférence de type et les types anonymes.

**Inférence de Type**

Il faut pour cela que Option Infer =On (Off par défaut)

Passer par le menu 'Projet' puis 'Propriétés de…', onglet 'Compiler'

En plus des options Explicit, Compare, Strict, on peut modifier Option Infer.

**L'inférence est la capacité de déduire le type d'une variable par analyse des types fournis en entrées ainsi que des opérations effectuées sur ceux-ci. C'est donc le compilateur qui déduit le type de la variable.**

```
' Type explicite pour une String
Dim Myname1 As String = "Rouge"

' Exploitation de l'inférence de type
Dim Myname2 = "Vert"
```

Le passage du curseur de la souris sur Myname2 vous permet de découvrir que celui-ci est bien un type "String".

On avait dit qu'il fallait travailler avec Option Implicit = True et là on ne déclare même pas le type !!! En fait l'inférence existe afin de supporter par exemple les types anonymes ou encore LINQ.

Il existe des cas pour lesquels l'inférence de type ne se produit pas. Pour les instructions Dim locales, l'inférence de type survient uniquement lorsqu'il y a une assignation sur la ligne de déclaration. Par conséquent, pour les assignations effectuées hors de la déclaration de la variable, le compilateur supposera que le type est Object. Object est également toujours déduit comme type des membres de niveau classe, si bien que l'inférence de type ne s'applique pas aux fonctions, sous-routines, propriétés, champs de classe/structure, etc. Lorsque Option Explicit est Off, une variable locale peut être utilisée dans le code sans déclaration explicite. La variable est supposée être dans ce cas de type Object et tous les appels sont liés tardivement. L'inférence de type ne survient pas sur les variables définies implicitement.

**Type anonyme**

Habituellement, on peut déclarer Mycustomer, une instance de la classe Customer et renseigner une propriété .Name.

```
Dim MyCustomer = New Customer With {.Name = "Philippe"}
```

Grâce au type anonyme, on peut écrire :

```
Dim AnomyneCustomer = New With {.Name = "Philippe"}
```

Remarque= avant New il doit y avoir '=' et pas As.

Cela créer une nouvelle classe anonyme (sans nom) possédant une propriété .Name.

Les types anonymes sont surtout utilisés avec Linq.

Exemple :

```
Dim namePriceQuery = From prod In products _

                     Select prod.Name, prod.Price
```

Si products est une liste d'objets avec plein de propriétés, namePriceQuery est une collection de type anonyme qui possède 2 propriétés : .Name et .Price .

## V-Z - Les 'Region', compilation conditionnelle, 'Attributs'

**Dans le code on peut ajouter des choses qui ne sont pas du code VB, mais plutôt des directives pour l'affichage, le compilateur ou le Runtime:**

## V-Z-1 - Les Régions

Pour une meilleure visibilité, il est possible de créer des 'régions' de code. Une région peut être déroulée ou contractée.

Une région peut être déroulée: le code entre #Region et #End Region est visible (pour modifier le code par exemple)

-

**#Region "Routine de Tri"**

```
Sub QuickSort(ByVal debut As Integer, ByVal fin As Integer)

Dim pivot, gauche, droite, temp As Integer

Do

…

Loop Until gauche = droite

End Sub

#End Region
```

Si on clique sur le petit carré (avant #region), cela contracte la région et masque le code, on voit seulement un petit carré avec un plus et le nom de la région.

+ Routine de Tri

Cela permet de masquer une procédure en totalité.

Attention, cela ne permet pas de masquer seulement une partie du code, mais la procédure entière.

Exemple

En VB 2003, dans une Classe de formulaire, il existe une région nommée 'Code généré par le Concepteur Windows Form' qui contient le code créant les contrôles du formulaire. Ce code est habituellement caché dans une 'région' fermée.

## V-Z-2 - La Compilation conditionnelle

La compilation conditionnelle contrôle si les séquences de lignes sont traduites en code réel. Certaines lignes peuvent être ignorées pendant le processus de compilation.

Les instructions de compilation conditionnelle sont précédées de #

On utilise :

```
#if … then

#else

#end if
```

Exemple :

```
#const Demo = True     'créer une constante conditionnelle
```

```
Class MaClasse
#if Demo then
  Sub F()
#else
  Sub G()
#end if

End Class
```

La compilation produit le résultat suivant :

```
Class C
Sub F()
End Class
```

Il suffit de changer la valeur de la constante pour compiler des parties différentes de code.

Noter que #const Demo crée une constante privée accessible uniquement dans le fichier.

En VB 2005 on peut définir une constante au niveau projet avec /define

```
/define const Demo=True
```

## V-Z-3 - Les Attributs

Les attributs peuvent être utilisés pour **décrire votre code** au runtime (fournir des informations supplémentaires) ou **modifier le comportement de l'application au moment de l'exécution.** Le Framework fournit de nombreux attributs, mais vous pouvez également créer vos propres attributs personnalisés.

Les attributs sont entre < et > en VisualBasic.

Les attributs peuvent modifier le comportement des propriétés, méthodes, classes, assemblys. Ils couvrent différents aspects comme la compilation, la sécurité, les services Web…

**Exemple: <Obsolete > Avec une procédure.**

Déclarons une fonction Add comme obsolète, en plus, le compilateur affiche le message: 'Sera enlevé à la prochaine version'.

On utilise donc <Obsolete > ou le nom complet de l'attribut: <System.ObsoleteAttribut>

```
<Obsolete("Sera enlevé à la prochaine version ")> Function Add(a as Integer,
 b as Integer) as Integer
Add = a + b - c
End Function
```

**Exemple:<Browsable> avec un composant.**

Dans un composant, je crée une Propertie nommée 'Valide', je ne veux pas qu'elle apparaisse dans la fenêtre 'propriétés' du composant; je veux qu'elle soit accessible uniquement par code :

```
Imports System.ComponentModel    'Classe chargée du comportement des composants.



<Browsable(False)> Property Valide() As Integer
```

**Exemple:<ToolBoxBitMap> avec un composant.**

Quand on crée un composant, on désire parfois avoir une icône propre à ce composant dans la boite à outils:

```
<ToolBoxBitMap("C:MonIcone")> Public Class MaClasse
```

**Exemple:<Serializable> avec une Classe.**

Quand on crée une classe, on a parfois besoin qu'elle soit sérializable :

```
<Serializable()> Public Class TestSimpleObject

Public member1 As Integer
Public member2 As String
Public member3 As String
Public member4 As Double

'Un member qui ne doit pas être sérialisé.
<NonSerialized()> Public member5 As String
```

Il est possible de faire un tas de choses avec les attributs, mais cela devient vite très complexe.

## V-AA - Traiter les erreurs

Il y a plusieurs types d'erreurs.

- Les erreurs de syntaxe.

- Les erreurs d'exécution.

- Les erreurs de logique.


Voir la vidéo : **au format 'Flash'**> ou **au format 'Avi'** en Visual Basic 2005.

## V-AA-1 - Les erreurs de syntaxe

On peut aussi les nommer 'les erreurs du compilateur', elles se produisent lorsque le compilateur Visual Basic rencontre un code non reconnaissable, erreur de saisie ou méconnaissance du langage. Comme les erreurs du compilateur empêchent un programme de s'exécuter, vous devez être averti de ces erreurs avant de tenter d'exécuter votre programme, autrement dit durant la saisie du code.

Elles surviennent donc en mode conception quand on tape le code.

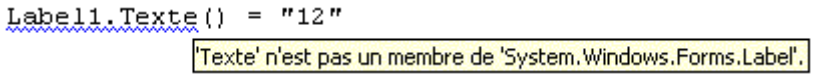
Exemples :

```
A+1=B            'Erreur dans l'affectation
```

```
f.ShowDialogue   'Faute de frappe, il fallait taper ShowDialog

2 For… et un seul Next

Dim i As Integer: Label.Text= i  'Affectation d'un Integer à une propriété text qui attend une
 String.

……
```

Dans ces cas VB souligne en ondulé bleu le code. Il faut mettre le curseur sur le mot souligné, l'explication de l'erreur apparait.

Exemple : Propriété Text d'un label mal orthographiée :




Il faut les corriger immédiatement en tapant le bon code (ici 'Text').

En bas il y a aussi une fenêtre; "liste des erreurs" :


Elle affiche tous les problèmes; pour atteindre le code correspondant à une de ces erreurs, double-cliquez sur une des lignes de la liste.

En VB 2005 un bouton avec point d'exclamation permet d'ouvrir une fenêtre proposant le moyen de corriger l'erreur :


Ici on met dans la propriété text d'un label un Integer, alors qu'il faut mettre une String (Option Strict est probablement égal à On); Vb montre la correction : CStr(i) convertit i en String.

Si vous exécutez le programme dans l'IDE alors qu' il y a un problème, VB demande si on veut exécuter la dernière génération réussie :


Si vous tapez 'oui' VB exécute la dernière version qui a été générée correctement, mais PAS le code source actuel qui contient des erreurs !!

## V-AA-2 - Les erreurs d'exécution

Elles surviennent en mode Run dans l'IDE ou lors de l'utilisation de l'exécutable:
**Une instruction ne peut pas être effectuée.**

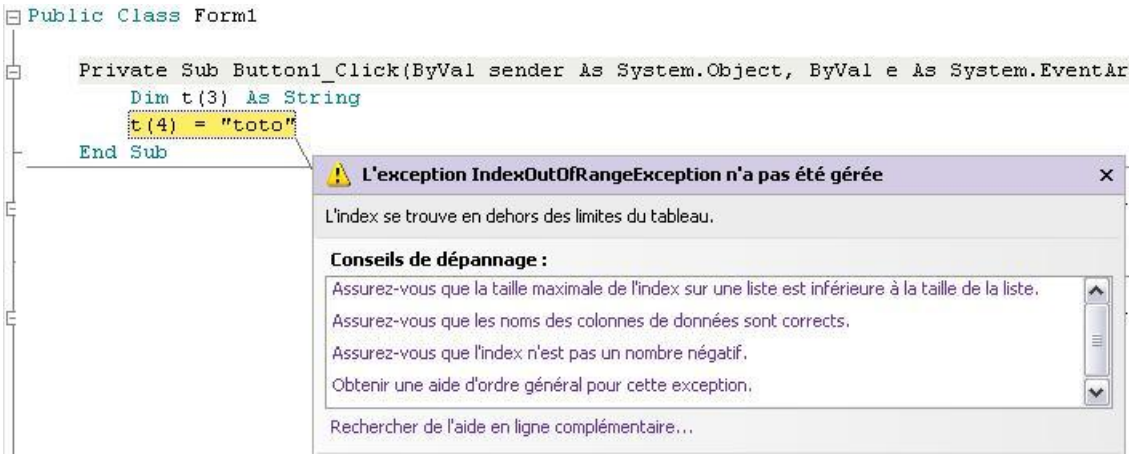
Quand on utilise l'exécutable: le logiciel s'arrête brutalement, c'est très gênant !!

Pour l'utilisateur c'est un 'BUG' 

Il y a 'levée d'une **exception**', voila ce que cela donne dans l'IDE.

Exemple : je tente d'accéder à un élément d'un tableau qui n'existe pas (l'indice est trop grand cela entraine une exception 'OutOfRange').

En cours de test, dans l'IDE, s'il y a une exception, le logiciel s'arrête, l'instruction qui a planté apparait en jaune et VB donne une explication.




**L'erreur est :**

- soit une erreur de conception.
  Exemple
  Ouvrir un fichier qui n'existe pas ( on aurait dû vérifier qu'il existe avant de l'ouvrir!).
  Division par zéro.
  Utiliser un index d'élément de tableau supérieur au nombre d'éléments.
  Envoyer un mauvais paramètre à une fonction ;
- soit une erreur de l'utilisateur.
  Exemple : on lui demande de taper un chiffre, il tape une lettre ou rien puis valide.

Il faut toujours vérifier ce que fait l'utilisateur et prévoir toutes les possibilités.
Exemple: si je demande à l'utilisateur de taper un nombre entre 1 et 10, il faut :
vérifier qu'il a tapé quelque chose ;
que c'est bien un chiffre (pas des lettres) ;
que le chiffre est bien entre 1 et 10 ;
sinon il faudra reposer la question.

**A-Capter les erreurs avec Try Catch Finally**

Plutôt que de laisser le logiciel 'planter', je vais anticiper et essayer de capter l'erreur au niveau des lignes de code qui peuvent la provoquer.

Avant l'instruction supposée provoquer une erreur indiquez : Essayer l'instruction (**Try**), si une erreur se produit Intercepter l'erreur (**Catch**) puis poursuivre (après **Finally**).

```
Try

    'Instruction susceptible de provoquer une erreur.

Catch

    'Traitement de l'erreur

Finally

    'Code toujours exécuté

End Try
```

Il faut pour que cela fonctionne avoir tapé au préalable **Imports System.IO**

Il est possible d'utiliser Catch pour récupérer l'objet '**Exception**' qui est généré par l'erreur.

```
Catch ex As Exception
```

Cet objet Exception à des propriétés.

**Message** qui contient le descriptif de l'erreur.

**Source** qui contient l'objet qui a provoqué l'erreur…

ex. Message contient donc le message de l'erreur.

Cet objet généraliste **Exception** (de l'espace IO) a aussi des classes dérivées :

-StackOverFlowException ;

-FileNotFoundException ;

-EndOfStreamException ;

-FileLoadException ;

-PathTooLongException.

Enfin une exception peut provenir de l'espace System : ArgumentExceptions ; ArithmeticException.

-DivideByZeroException…

Il est possible d'écrire plusieurs instructions Catch avec pour chacune le type de l'erreur à intercepter. (Faisant partie de la classe Exceptions.)

Exemple

On ouvre un fichier par StreamReader, comment intercepter les exceptions suivantes ?

Répertoire non valide

Fichier non valide

Autre.

```
Try
    sr= New StreamerReader (NomFichier)
Catch ex As DirectoryNotFoundException
    MsgBox("Répertoire invalide")
Catch ex As FileNotFoundException
    MsgBox("Fichier invalide")
Catch ex As Exception
    MsgBox(ex.Message)
End Try
```

Noter que le dernier Catch intercepte toutes les autres exceptions.

On peut encore affiner la gestion par le mot-clé **When** qui permet une condition.

```
Catch ex As FileNotFoundException
             When ex.Message.IndexOf ("Mon Fichier.txt") >0
                MsgBox ("Impossible d'ouvrir Mon Fichier.txt")
```

Si le texte "Mon Fichier.txt" est dans le message, affichez que c'est lui qui ne peut pas être ouvert.

**Exit Try** permet de sortir prématurément. Quitte immédiatement le bloc Try ou Catch dans lequel il est. L'exécution continue avec le bloc Finally s'il y en a un, ou avec l'instruction qui suit End Try.

**B-Capter les erreurs avec On error :**

On peut aussi utiliser la méthode Visual Basic:

**On Error Goto** permet en cas d'erreur de sauter à une étiquette (un emplacement dans le code) emplacement ou une portion de code traite l'erreur.

On peut lire le numéro de l'erreur qui s'est produite, ce numéro est dans **Err.Number**.

**Err.Description** contient le texte décrivant l'erreur. Err.Source donne le nom de l'objet ou de l'application qui a créé l'erreur.

Quand l'erreur est corrigée, on peut revenir de nouveau à la ligne qui a provoqué l'erreur grâce à **Resume** ou poursuivre à la ligne suivante grâce à Resume Next

Exemple :

```
On Error GoTo RoutinedErreur 'Si une erreur se produit se rendre à 'RoutineErreur'
Dim x As Integer = 33
Dim y As Integer = 0
Dim z As Integer
z = x / y ' Crée une division par 0 !!

RoutinedErreur: ' La Routine d'erreur est ici (remarquer  ':' indiquant une etiquette).
Select Case Err.Number ' On regarde le numéro de l'erreur.
Case 6 ' Cas : Division par zéro interdite
    y = 1 ' corrige l'erreur.
Case Else
    ' autres erreurs……
End Select
Resume ' Retour à la ligne qui a provoqué l'erreur.
```

Pour arrêter la gestion des erreurs, il faut utiliser :

```
On Error Goto 0
```

Parfois on utilise une gestion hyper simplifiée des erreurs.

Si une instruction 'plante', la sauter et passer à l'instruction suivante, pour cela on utilise:

**On Error Resume Next**

Exemple : On veut effacer un fichier

```
On Error Resume Next

Kill (MonFichier)

On Error goto 0
```

Ainsi, si le fichier n'existe pas, cela ne plante pas (on aurait pu aussi vérifier qu'il existe avant de l'effacer !!).

On Error Gosub n'existe plus.

*On Error est moins performant que Try Catch et surtout il ralentit le code+++ : si nécessaire utiliser Try Catch.*

**En résumé: pour éviter les erreurs d'exécution, il est donc possible :**

- d'écrire du code gérant le problème, contrôlant les actions de l'utilisateur…

Exemple : on demande à l'utilisateur de saisir un nombre dans TextBox1 puis de cliquer sur Button3.

Si l'utilisateur a tapé une lettre au lieu d'un chiffre, le prévenir.

```
Private Sub Button3_Click

If String.IsNullOrEmpty(TextBox1.Text) Then 'on teste si l'utilisateur a tapé quelque chose
```

```
 MsgBox("Tapez quelque chose")

Else

 If Not IsNumeric(TextBox1.Text) Then 'on teste si l'utilisateur a tapé du numérique

    MsgBox("Tapez un chiffre")

 End If

End If

End Sub
```

- une autre possibilité est de capter l'erreur.

Exemple : on demande à l'utilisateur de saisir un nombre dans TextBox1 puis de cliquer sur Button3.

Convertir le texte tapé en Integer, on sait que si la conversion est impossible (pas de texte tapé ou texte non numérique) une exception **invalidCastException** sera levée et le programme 'plantera'. On écrit donc avant l'instruction CType un Try pour capter l'erreur.

Tester s'il y a une erreur, la capter.

```
Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
 Button3.Click

Dim i As Integer

Try

    i = CType(TextBox1.Text, Integer)

Catch

    MsgBox("saisir un nombre")

End Try

End Sub
```

## V-AA-3 - Les erreurs de logique

Le programme fonctionne, pas d'erreurs apparentes, mais les résultats sont erronés, faux.

Il faut faire des tests dans les conditions réelles avec des données courantes, mais aussi avec des données remarquables (limites supérieures, inférieures, cas particuliers…) pour voir si les résultats sont cohérents et exacts.

Une fois l'erreur trouvée, il faut en déterminer la cause et la corriger.

Ou bien elle est évidente à la lecture du code ou bien elle n'est pas évidente et c'est l'horreur.

Dans ce dernier cas, il faut analyser le fonctionnement du programme pas à pas, instruction par instruction en surveillant la valeur des variables.(voir la rubrique débogage )

Les erreurs les plus communes sont :

utilisation d'un mauvais nom de variable (la déclaration obligatoire des variables évite cela) ;
erreur dans la portée d'une variable ;
erreur dans le passage de paramètres (Attention au By Val et By Ref) ;
erreur dans la conception de l'algorithme.
…

Quelques règles permettent de les éviter : voir Règles de bonne programmation.

## V-AA-4 - Les Tests

Il faut donc toujours tester le fonctionnement du programme de multiples fois.

**On fera des :**

- tests unitaires : qui testeront les procédures, les classes une à une sans tester la totalité du programme ;
- tests de composants et d'intégration : qui testeront plusieurs procédures ou classes fonctionnant ensemble ;
- tests de régression : c'est la répétition des tests précédents afin de voir si une modification ou un ajout n'entraine pas de nouvelles erreurs qui n'existaient pas ;
- tests système : test sur le logiciel dans sa version finale.

Les tests détecteront les erreurs, le débogage permettra de trouver la cause et de corriger l'erreur.

*Il faut avoir une armée de Bêta-testeurs.*

## V-AB - Travailler sur les dates, les heures, sur le temps

Il est probablement nécessaire de lire le chapitre VI sur les Classes avant de lire celui-ci.

Il existe un type de variable '**DateTime**' pour gérer les dates et heures, comment l'utiliser ?

Nous verrons aussi comment utiliser les **Timers** pour déclencher des événements à intervalle régulier. Enfin comment perdre du temps ?

Une variable **DateTime** contient une date plus l'heure.

Elle occupe 8 octets.(64 bits.)

Elle peut contenir une date comprise entre le 1er janvier de l'année 01 et le 31 décembre 9999 et une heure comprise entre 0:00:00 (minuit) et 23:59:59.

En fait ce qui est codé dans la variable DateTime est le nombre de graduations (une graduation= 100 nanosecondes) écoulées à compter de minuit, le 1er janvier de l'année 1 jusqu'a la date codée.

N. B. DateTime fait partie d'une Classe .Net, il existe aussi un type nommé Date qui fait partie de Visual Basic, qui est équivalent à DateTime.

Préférez DateTime qui est une Classe Net.

## V-AB-1 - Définir une date, une heure

A - Pour définir une valeur DateTime en utilisant un littéral: elle doit être placée entre des signes (#) et son format doit être de type m/d/yyyy, par exemple #5/31/1998#.(contrairement à ce que je pensais, même si le format date dans la configuration de la langue de l'ordinateur est de type français).

```
Dim  dateNaissance As DateTime

dateNaissance= #12/02/1951#
'attention  mois/jour/année
```

B - Autre manière de saisir une date, une heure :

```
Dim dateNaissance As New System.DateTime(1996, 6, 3, 22, 15, 0)

'Année, mois, jour, heure, minute, seconde, et éventuellement millisecondes)
```

Ici on a utilisé le constructeur.

C -Troisième méthode

On peut saisir une date dans une string puis convertir :

```
DateNaissance = CDate("02/12/1951")
```

CDate convertit donc une chaine en dateTime. On peut aussi utiliser Ctype :

```
Dim dateNaissance As Date = CType("01/12/2005", Date)
```

**IsDate** (objet) permet de vérifier si objet est convertible en date.

IsDate retourne True si l'expression est de type Date ou est une chaine convertible en type Date ; sinon, elle retourne False.

Cela permet de vérifier, après une saisie d'une string par exemple, si l'utilisateur a bien tapé des chiffres valides et même si la date est valide ("31/02/1945" n'est pas valide).

Bizarrerie= "12/2005" est considéré comme une date valide et équivalente à "01/12/2005"!! Pas de vérification des 2 '/'.

```
If IsDate( MyString) Then…
```

Exemple de Microsoft :

```
Dim MyDate, YourDate As DateTime
Dim NoDate As String
Dim D As Boolean
MyDate = CDate("12 Février, 1969")
YourDate = #2/12/1969#
NoDate = "Hello"
D = IsDate(MyDate) ' Retourne True.
D = IsDate(YourDate) ' Retourne True.
D = IsDate(NoDate) ' Retourne False.
```

## V-AB-2 - Afficher une date, une heure

Pour afficher les dates et heures simplement, il suffit d'utiliser .ToString

```
MsgBox(DateNaissance.ToString)     'Affichera  02/12/1951 11:00:00
```

Le format utilisé est le format d'affichage des dates de l'ordinateur (en fonction du pays, en France c'est le format fr).

ToString peut comporter des arguments qui formatent l'affichage.

Voici quelques codes de formatage :

```
d         affiche le jour                           2

dd        affiche le jour sur 2 chiffres            02

ddd       affiche le jour abrégé                    Dim.

dddd      affiche le jour complet                   Dimanche

M         affiche le mois                           12

MM        affiche le mois sur 2 chiffres            12

MMM       affiche le mois abrégé                    déc

MMMM      affiche le mois complet                   décembre

y, yy, yyyy affiche 1 à 2 chiffres, deux chiffres ou quatre chiffres     51, 51, 1951

H       affiche l'heure  sur un ou deux chiffres (format 24h)

HH      affiche l'heure sur 2 chiffres

h et hh font de même, mais avec un format 12 h.

t, tt  affiche l'heure en format 12h  plus A ou P (pour matin, après midi)

m, mm, s, ss, f, ff font de même pour les minutes, secondes et millisecondes.

: et / sont les séparateurs heure et date.
```

Exemple :

```
MsgBox(DateNaissance.ToString("dddd d MMMM yyyy"))     'Affichera  Dimanche 2 décembre 1951

MsgBox(DateNaissance.ToString("hh:mm")     'Affichera  11:00

MsgBox(DateNaissance.ToString("dd/MM/yy")     'Affichera  02/12/51

MsgBox(DateNaissance.ToString("%h)
'Affichera  11   le caractère % est utilisé quand on affiche une seule donnée.
```

On peut enfin utiliser les méthodes de la classe DateTime !!

```
DateNaissance.ToLongDateString        'dimanche 02 décembre 1951

DateNaissance.ToShortDateString       '02/12/1951

DateNaissance.ToLongTimeString        '11:00:00

DateNaissance.ToShortTimeString       '11:00
```

## V-AB-3 - Variable "temps"

Un **TimeSpan** est une unité de temps (un intervalle de temps) exprimée en jours, heures, minutes, secondes.

Un TimeSpan initialisé avec 1.0e+13 graduations représente "11.13:46:40", ce qui correspond à 11 jours, 13 heures, 46 minutes et 40 secondes.

On peut initialiser un TimeSpan avec des graduations :

```
Dim instance As New TimeSpan(ticks)
```

ou avec des heures, minutes, secondes, millisecondes.

```
Dim instance As New TimeSpan(h, m ,s ,ms)
```

On peut aussi l'initialiser avec un certain nombre de jours, d'heures, de secondes. Exemple avec 4 jours.

```
Dim value As Double= 4
Dim returnValue As TimeSpan

returnValue = TimeSpan.FromDays(value)
```

L'espace de noms System.DateTime. contient une multitude de membres.

## V-AB-4 - Add, Substrat

On peut ajouter ou soustraire un TimeSpan à un DateTime, on obtient un DateTime.

En clair on peut ajouter à une date une durée, on obtient une date.

```
' Quel sera la date  dans 36 jours?.
Dim today As System.DateTime
Dim duration As System.TimeSpan
Dim answer As System.DateTime

today = System.DateTime.Now
duration = New System.TimeSpan(36, 0, 0, 0)
answer = today.Add(duration)
```

On peut ajouter ou soustraire 2 dates, on obtient une TimeSpan

```
Dim diff1 As System.TimeSpan
diff1 = date2.Subtract(date1)
```

## V-AB-5 - AddDay, AddMouths, AddHours, AddSeconds, AddMiliseconds

Permet d'ajouter des jours, des mois, des heures, des secondes, ou des millisecondes à une date, on obtient une date.

```
Answer=today.AddDay(36)
```

## V-AB-6 - Year, Mouth, Day, Hour, Minute, Seconde, Millisecond

Permettent d'extraire l'année, le mois, le jour, l'heure, les minutes, les secondes, les millisecondes d'une date :

```
I=DateNaissance.Year     ' => I=1951
```

```
I=System.DateTime.Now.Day    'donne le jour d'aujourd'hui (1 à 31)
```

(DatePart permet aussi d'extraire plein d'informations d'une date: jour , mois, année, jour de la semaine…).

## V-AB-7 - DayOfWeek, DayOfYear, DayInMonth

Retourne le jour de la semaine (0 pour dimanche à 6 pour samedi) :

```
I=DateNaissance.DayOfWeek    'I=0,  car le 02/12/1951 est un dimanche.
```

DayOfYear existe aussi.

dateNaissance.IsDaylightSavingTim indique si on est dans l'heure d'été pour le fuseau

Date.DayInMonth donne le nombre de jours dans le mois spécifié :

```
Date.DaysInMonth(1951, 12)
```

## V-AB-8 - Now, ToDay, TimeOfDay

Now est la date et l'heure du système. (Là, maintenant.)

ToDay est la date du système avec l'heure à 0.

TimeOfDay est l'heure actuelle.

My.Computer.Clock permet de récupérer l'heure courante ainsi que le nombre de millisecondes écoulées depuis le démarrage.

```
MsgBox(My.Computer.Clock.LocalTime.ToString) 'Affiche date et heure
```

## V-AB-9 - Ticks

Donne le nombre de graduations d'un DateTime.

AddTicks peut être utilisé.

## V-AB-10 - Année bissextile, jours fériés

Pour cela, utiliser IsLeapYear :

```
MsgBox(DateTime.IsLeapYear(2005)) 'Affiche False
```

## V-AB-11 - Comparaison de DateTime

On utilise Compare: DateTime.Compare(t1, t2) retourne 0 si t1=t2, une valeur positive si t1>t2 négative si t1<t2.

```
Dim t1 As New DateTime(100)
Dim t2 As New DateTime(20)

If DateTime.Compare(t1, t2) > 0 Then
    Console.WriteLine("t1 > t2")
```

```
End If
If DateTime.Compare(t1, t2) = 0 Then
    Console.WriteLine("t1 = t2")
End If
If DateTime.Compare(t1, t2) < 0 Then
    Console.WriteLine("t1 < t2")
End If
```

On peut aussi utiliser la méthode op_Equality de l'espace de noms pour voir si 2 dates sont égales :

```
areEqual = System.DateTime.op_Equality(april19, otherDate)
```

Il existe aussi op_GreaterThan et beaucoup d'autres.

## V-AB-12 - Calcul de la différence entre deux dates

On utilise DateDiff, il faut fournir en paramètre :

- l'intervalle de temps à utiliser comme unité de la différence entre Date1 et Date2.
  DateInterval.Day pour obtenir le nombre de jours entre les 2 dates.
  DateInterval.Year pour obtenir le nombre d'années entre les 2 dates ;
  …
- Date1 ;
- Date2.

Exemple

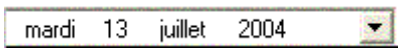
Afficher le nombre de jours entre une date donnée et la date du jour.

```
Dim DateS, Msg As String ' Declare les  variables.
Dim DateD As DateTime
DateS = InputBox("Entrer une date") 'Saisir une date : on récupère une string
DateD = CDate(DateS)                'Conversion de la string en DateTime
Msg = "Nombre de jour:"& DateDiff(DateInterval.Day, Now, DateD) 'différence en jours
MsgBox (Msg)
```

## V-AB-13 - Comment saisir rapidement une date dans un programme ?

En ajoutant à une fenêtre un contrôle **DateTimePicker**.

En mode Run , il apparait une zone rectangulaire avec la date système dedans :




Si l'utilisateur clique sur la flèche déroulante, il apparait une fenêtre calendrier.


Il suffit pour l'utilisateur de cliquer sur la bonne date.

Le programmeur récupère la date dans DateTimePicker1.value.

Il existe, bien sûr, de multiples propriétés et plusieurs événements, le plus remarquable étant: ValueChanged.

MonthCalendar est un contrôle similaire, mais qui reste toujours ouvert.

De plus grâce à CalendarDimension on peut afficher plusieurs mois.

## V-AB-14 - Fuseau horaire

**TimeZone représente le fuseau horaire actuel, celui défini dans le panneau de configuration**.

Vous pouvez utiliser la classe TimeZone pour récupérer des informations à propos du fuseau horaire actuel et convertir l'heure locale en temps universel (UTC, Universal Time Coordinated) ou vice versa.

Le fuseau actuel est dans **TimeZone.CurrentTimeZone**, son nom est dans la propriété **.StandartName**; pour convertir en temps universel : **.ToUniversalTime(currentDate)**.

```
Imports System.Globalization
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
 Button1.Click

        Const dataFmt As String = "{0,-30}{1}"
        Const timeFmt As String = "{0,-30}{1:dd-MM-yyyy HH:mm}"

        Console.WriteLine("Exemple TimeZone" & vbCrLf)

        ' Prendre le time zone local , la datetime de maintenant.
        'l'année actuelle
        Dim localZone As TimeZone = TimeZone.CurrentTimeZone
        Dim currentDate As DateTime = DateTime.Now
        Dim currentYear As Integer = currentDate.Year

        ' Affiche le fuseau local
        ' Affiche le fuseau de l'heure d'été.
        Console.WriteLine(dataFmt, "Nom du fuseau local:", _
            localZone.StandardName)
        Console.WriteLine(dataFmt, "Nom du fuseau de l'heure d'été:", _
            localZone.DaylightName)

        ' Affiche date et heure courantes
        'Affiche si l'heure d'été est en cours
        Console.WriteLine(vbCrLf & timeFmt, _
            "Date et heure courante:", currentDate)
        Console.WriteLine(dataFmt, "Heure d'été?", _
```

```
            localZone.IsDaylightSavingTime(currentDate))

        ' Prendre le temps universel (UTC) et l'offset
        Dim currentUTC As DateTime = _
            localZone.ToUniversalTime(currentDate)
        Dim currentOffset As TimeSpan = _
            localZone.GetUtcOffset(currentDate)

        Console.WriteLine(timeFmt, "Date et heure universelle:", _
            currentUTC)
        Console.WriteLine(dataFmt, "Différence local-UTC:", currentOffset)

        ' Prendre l'heure d'été.
        Dim daylight As DaylightTime = _
            localZone.GetDaylightChanges(currentYear)

        ' Affiche début et fin heure d'été puis le delta.
        Console.WriteLine(vbCrLf & _
            "Année de l'heure d'été {0}:", currentYear)
        Console.WriteLine("{0:dd-MM-yyyy HH:mm} à " & _
            "{1:dd-MM-yyyy HH:mm}, delta: {2}", _
            daylight.Start, daylight.End, daylight.Delta)
    End Sub


End Class
```

Affiche :

```
Exemple TimeZone

Nom du fuseau local:          Paris, Madrid
Nom du fuseau de l'heure d'été:Paris, Madrid

Date et heure courante:       23-05-2009 14:51
Heure d'été?                  True
Date et heure universelle:    23-05-2009 12:51
Différence local-UTC:         02:00:00

Année de l'heure d'été 2009:
29-03-2009 02:00 à 25-10-2009 03:00, delta: 01:00:00
```

Vous ne pouvez pas utiliser la classe TimeZone pour représenter des fuseaux horaires autres que ceux de la zone locale ou pour gérer des conversions de date et heure d'un fuseau horaire en un autre. Pour cela, utilisez la classe **TimeZoneInfo** (permet de travailler sur n'importe quel fuseau horaire). Exemple : dans un TimeZoneInfo, on va mettre le TimeZoneInfo local :

```
Dim localZone As TimeZoneInfo = TimeZoneInfo.Local
Console.WriteLine("Local Time Zone ID: {0}", localZone.Id)
Console.WriteLine("   Display Name is: {0}.", localZone.DisplayName)
Console.WriteLine("   Standard name is: {0}.", localZone.StandardName)
Console.WriteLine("   Daylight saving name is: {0}.", localZone.DaylightName)
```

**TimeZoneInfo.Utc** Obtient un objet TimeZoneInfo qui représente la zone de temps universel (UTC, Universal Time Coordinated).

On peut convertir un DateTime d'un fuseau à un autre :

```
Dim dateTime As DateTime
Dim destinationTimeZone As TimeZoneInfo
Dim returnValue As DateTime

returnValue = TimeZoneInfo.ConvertTime(dateTime, _
    destinationTimeZone)
```

La nouvelle classe **DateTimeOffset** permet de mieux gérer les applications qui utilisent les zones dates et heures.

Elle associe un DateTime à un Offset (différence entre heure locale et temps universel).

## V-AB-15 - Les Timers

Pour déclencher un événement à intervalle régulier, il faut utiliser les minuteries ou '**Timer**'.

Prendre le contrôle Timer dans la Boite à outils, l'ajouter à la fenêtre. Il apparait en bas sous la fenêtre dans la barre d'état des composants.

Il n'apparait pas à l'utilisateur dans la fenêtre en mode Run.

Il est très simple à utiliser.

La propriété **Interval** contient la périodicité de l'événement **Ticks**, événement qui se déclenche régulièrement.

Interval est en millisecondes. Pour Interval=500 l'événement Ticks se déclenche toutes les 1/2 secondes.

**Start** et **Stop** déclenche et arrête la minuterie (De même Enabled active ou non ).

Exemple

Faire clignoter un label toutes les 1/2 secondes.

Créer le label1

Ajouter un Timer1 (qui se place en bas sous la fenêtre).

```
Private Sub Form3_Load(… )

    Timer1.Interval = 500

    Timer1.Start()

End Sub


Private Sub Timer1_Tick(…)

    Label1.Visible = Not (Label1.Visible)

End Sub
```

Un événement Timer_Tick se produit toutes les 1/2 secondes et inverse la valeur de la propriété visible du label. (Si elle était égale à True, elle devient égale à False et vice versa.)

**Mais attention: Timer a des restrictions de taille**

- Si votre application ou une autre demande beaucoup au système (boucles longues, calculs complexes, accès intensifs à un périphérique, un réseau ou un port, par exemple), les événements de minuterie peuvent être moins fréquents que spécifiés dans la propriété Interval. Il n'est pas garanti que l'intervalle s'écoule dans le temps exact !!
- L'intervalle peut être compris entre 1 et 64 767 millisecondes : l'intervalle le plus long ne dépasse pas de beaucoup la minute (64,8 secondes).
- Le système génère 18 graduations à la seconde (même si la valeur de la propriété Interval est mesurée en millisecondes, la véritable précision d'un intervalle ne dépassera pas un dix-huitième de seconde).

Donc pour faire clignoter un label : OUI

Pour compter précisément un intervalle de temps : NON

Mais il y a d'autres méthodes.

## V-AB-16 - Perdre du temps

Parfois on a besoin de perdre du temps.

Exemple ne rien faire pendant 3 secondes puis poursuivre…

- Il est exclu de faire des boucles vides :

```
For i=0 to 100000    ' le temps écoulé est variable en fonction des machines…

Next i
```

- Autre méthode : on boucle tant que l'heure courante est inférieure à l'heure du départ+3s

```
Dim t As DateTime=DateTime.Now

Do While DateTime.Now <t.AddSeconds(3)

Loop
```

  Mais cela accapare le processeur.
- On peut utiliser un Timer et vérifier dans la procédure Tick si le temps est écoulé (avec les restrictions que l'on connait).
- On peut utiliser **Thread.Sleep** (qui met le processus en cours en sommeil).

```
System.Threading.Thread.Sleep(3000)
```

Le temps de sommeil du thread est en millisecondes : 3000 correspond à 3 secondes.

## V-AB-17 - Chronométrer

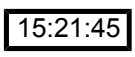
Parfois on a besoin de chronométrer un événement.

Voir la rubrique Chronométrer.

L'exemple sur l'horloge est aussi didactique.

## V-AB-18 - Exemple: Horloge numérique

Très Simple : comment créer une horloge numérique ?




Dans la fenêtre Form1.

Mettre un Timer (Timer1)

Mettre un label (Label1)

Ajouter le code :

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles MyBase.Load

    Timer1.Interval = 1000    'Timer1_Tick sera déclenché toutes les secondes.

    Timer1.Start()            'On démarre le Timer

End Sub

Private Sub Timer1_Tick(ByVal sender As Object, ByVal e As System.EventArgs) Handles Timer1.Tick

Label1.Text = Now.ToLongTimeString    'Affiche l'heure format long.

End Sub
```

Simple !! non !!

## V-AC - Lire et écrire dans les fichiers (séquentiels ou aléatoires)

Il est probablement nécessaire de lire le chapitre VI sur les Classes avant de lire celui-ci.

Comment lire et écrire dans des fichiers du texte, des octets, du XML du Rtf ?

### V-AC-1 - Généralités et rappels

Le mot '**fichier**' est a prendre au sens informatique: ce n'est pas un ensemble de fiches, mais plutôt un ensemble d'octets. Un fichier peut être un programme (Extension .EXE), du texte (Extension .TXT ou .DOC… ), une image (Extension .BMP .GIF .JPG…), une base de données (.MDB…) du son, de la vidéo…

Pour travailler avec du texte, des octets, des données très simples (sans nécessité d'index, de classement…), on utilise les méthodes décrites dans ce chapitre: travail direct dans les fichiers séquentiels, aléatoires, binaires. Mais dès que les informations sont plus structurées, il faut utiliser les bases de données (il y a plusieurs chapitres plus loin traitant des bases de données).

Un fichier a un nom: 'Image.GIF', une extension: '.GIF' qui en indique généralement le type de contenu , des attributs (Longueur, Date de création, de modification, Fichier en lecture seule ou non…).

On voit cela dans l'explorer Windows :

| Nom | Taille | Type | Date de modification |
|---|---|---|---|
| 2035.gif | 10 Ko | Image GIF | 25/11/2002 18:55 |
| acceuil.gif | 1 Ko | Image GIF | 27/12/2001 23:42 |
| ampoule.gif | 2 Ko | Image GIF | 11/04/2002 22:34 |

Un fichier est composé d'enregistrements qui sont des 'paquets' de données, suivant le type de fichier un enregistrement peut correspondre à une ligne, un octet, un groupe d'octets…

Un fichier peut être vu comme contenant du texte, du XML, des octets.

Comment utiliser les fichiers ? Voici le plan de cet article.

A- Il est conseillé de travailler avec les Classes du Framework.

Avec la Classe FileInfo, on obtient des renseignements sur le fichier.

Pour lire écrire dans un fichier (en dehors des bases de données), il y a plusieurs méthodes:

Avec la Classe System.Io on a a notre disposition StreamReader StreamWriter BinaryReader BinaryWriter FileStream:

Pour lire ou écrire dans un fichier, il faut l'ouvrir (Open), lire ou écrire en utilisant un flux de données (Stream) puis le refermer (Close).

Le Stream (flux, torrent, courant) est une notion générale, c'est donc un flux de données provenant ou allant vers un fichier, un port, une connexion TCP/IP…

L'accès est séquentiel: les données sont traitées du début à la fin du fichier.

B- Il existe toujours la méthode classique du FileOpen.

On ouvre le fichier en mode séquentiel, aléatoire, binaire, on lit X enregistrements, on referme le fichier.

C- Avec certains objets, on gère automatiquement les lectures écritures sur disque.

Comme avec le RichTextBox par exemple.

**En résumé, pour travailler sur les fichiers, on dispose :**

- de l'espace de noms System.IO avec les Classes et objets .NET ;
- des instructions VisualBasic traditionnelles: FileOpen WriteLine… ;
- des instructions du FSO (FileObjetSystem) pour la compatibilité avec les langages de script.

Les 2 derniers font appel au premier, donc pourquoi ne pas utiliser directement les Classes .NET ?

## V-AC-2 - Classe FileInfo et File, Stream du Framework

Pour travailler sur les fichiers, il faut au préalable taper :

Imports System.IO

La classe **File** est utilisée pour travailler sur un ensemble de fichiers ou un fichier (sans instanciation préalable : ce sont des méthodes statiques), la Classe **FileInfo** donne des renseignements sur un fichier particulier (il faut instancier au préalable un objet FileInfo).

La Classe **File** possède les méthodes suivantes.

```
Exists            Teste si le fichier existe.

Create            Crée le fichier

Copy              Copie le fichier

Delete            Efface le fichier

GetAttributes , SetAttributes     Lire ou écrire les attributs.

GetCreationTime , GetLastAccessTime , GetLastWriteTime et les Set… correspondant.

Move              Déplacement de fichier
```

```
Replace           Framework 2

ReadAllText, WriteAllText   Framework 2 lire ou écrire un texte dans un fichier

ReadAllLines, WriteAllLines   Framework 2 lire ou écrire des lignes dans un fichier

ReadAllBytes, WriteAllBytes   Framework 2 lire ou écrire des octets dans un fichier
```

Toutes les méthodes Open (pour un FileStream) OpenRead, OpenWrite, OpenText.

Exemple

**Un fichier existe-t-il? Afficher True s'il existe :**

```
                        Label1.Text = File.Exists("vessaggi.gif").ToString
```

Exists est bien une 'méthode de Classe': pas besoin d'instancier quoi que ce soit.

**Déplacer un fichier de c: vers d:**

```
    File.Move("c:\monText.txt", "d:\monText.txt")
```

**Copier un fichier de c: vers d:**

```
    File.Copy("c:\monText.txt", "d:\monText.txt", True)
```

Le dernier argument facultatif (framework 2) permet de remplacer le fichier cible s'il existe.

**Sauvegarde un fichier et le remplace (Framework 2)**

```
    File.Copy("c:\monText.txt", "c:\newText.txt",, "c:\newText.bak " True)
```

Sauvegarde monText.txt dans un .bak , puis copie NewText.txt dans monText.txt; True permet de remplacer la cible si elle existe.

**Efface un fichier**

```
File.Delete("d:\monText.txt")
```

**Lire la totalité d'un fichier texte? (Framework 2)**

```
Dim myText As String =File.ReadAllText("c:\monText.txt")
```

File.WriteAllText("c:\monText.txt", myText) 'pour réécrire le texte dans un autre fichier.

La méthode AppendAllText existe aussi.

**Lire un fichier texte et mettre dans un tableau les lignes d'un fichier texte ? (Framework 2)**

```
Dim myLines() As String =File.ReadAllLines("c:\monText.txt")
```

**Lire et mettre dans un tableau les octets d'un fichier (Framework 2)**

```
    Dim myBytes() As Byte =File.ReadAllBytes("c:\monText.txt")
```

**Un fichier est-il en lecture seule ?**

```
    If File.GetAttributes("c:\monText.txt") And FileAttributes.ReadOnly Then…
```

La Classe **FileInfo** possède les propriétés suivantes :

```
Name            Nom du fichier (sans chemin)

FullName        Nom complet avec chemin

Extension       Extension   (.txt par exemple)

Length          Longueur du  fichier.

Directory       Répertoire parent

DirectoryName   Répertoire où se trouve le fichier

Exists          Existe?

LastAccessTime  Date du dernier accès, LastWriteTime existe aussi.

Attributes      Attributs
```

Pour voir les attributs d'un fichier, il faut faire un AND entre Attributes et une valeur de l'énumération FileAttributes ( Archive, Compressed, Directory, Encrypted, Hidden, Normal, ReadOnly, System, Temporaly).

Pour tester ReadOnly par exemple :

```
Dim sNom As String = "c:\monfichier.txt"

Dim Fi As FileInfo      'On déclare un FileInfo

Fi=New FileInfo( sNom)  'On instancie ce FileInfo avec comme paramètre le nom du fichier
```

Ensuite :

```
    Fi.Attributes And FileAttributes.ReadOnly    'Retourne True si le fichier est ReadOnly
```

Et aussi :

**Fi.Name** retourne "monfichier.txt"

**Fi.FullName** retourne "c:\monfichier.txt"

**Fi.Name.Substring(0, Fi.Name.LastIndexOf("."))** retourne "monfichier" : pas de chemin ni d'extension.

Et les méthodes suivantes :

**Create, Delete, MoveTo**

**AppendTex, CopyTo Open, OpenRead, OpenWrite, OpenText…**

On voit que toutes les informations sont accessibles.

Exemple

Pour un fichier, afficher successivement le nom, le nom avec répertoire, le répertoire, la longueur, la date de dernière écriture et si le fichier est en ReadOnly.

```
Dim sNom As String = "c:\monfichier.txt"
```

```
Dim Fi As FileInfo  'On déclare un FileInfo

Fi=New FileInfo( sNom)  'on instance ce FileInfo avec comme paramètre le nom du fichier

    MsgBox("Nom="& Fi.Name)

    MsgBox("Nom complet ="& Fi.FullName)

    MsgBox("Répertoire="& Fi.DirectoryName)

    MsgBox("Longueur="& Fi.Length.ToString)

    MsgBox("Date dernière modification="& Fi.LastWriteTime.ToShortDateString)

    MsgBox("ReadOnly="& (Fi.Attributes And FileAttributes.ReadOnly).ToString)
```

## V-AC-3 - Classe My.Computer.FileSystem

À partir de VS 2005 il y a en plus la classe My.Computer.FileSystem qui simplifie énormément les choses.

Les méthodes CopyFile, DeleteFile, FileExits permettent de copier, effacer un fichier ou de voir s'il existe. Il existe aussi RenameFile et MoveFile.

Exemple

**Afficher dans une MsgBox True si 'c:\config.sys' existe.**

```
MsgBox(My.Computer.FileSystem.FileExists("c:\config.sys").ToString)
```

Exemple

**Afficher la liste des fichiers qui sont sous c:**; ici on utilise **GetFiles** qui retourne une collection des fichiers.Count contient le nombre de fichiers, item () les noms.

```
Dim i As Integer

For i = 0 To My.Computer.FileSystem.Getfiles("c:\").Count - 1

ListBox1.Items.Add(My.Computer.FileSystem.GetFiles("c:\").Item(i))

Next i
```

**Un fichier existe-t-il et est-il ouvert et utilisé par une autre application ?**

```
If My.Computer.FileSystem.FileExists("c:\monText.txt") Then

Try

   'on tente d'ouvrir un stream sur le fichier, s'il est déjà utilisé, cela déclenche une erreur.

   Dim fs As IO.FileStream = My.Computer.FileSystem.GetFileInfo("c:
\monText.txt").Open(IO.FileMode.Open, _
   IO.FileAccess.Read)

   fs.Close()

Catch ex As Exception

  MsgBox("Le fichier  est déjà ouvert")

End Try
```

```
End If
```

## V-AC-4 - Utiliser les "Stream" du Framework

Le **Stream** (flux, torrent, courant) est une notion générale, c'est donc **un flux de données** provenant ou allant vers un fichier, un port, une connexion TCP/IP…

Ici on utilise un Stream pour lire ou écrire dans un fichier.

L'accès est séquentiel: les données sont traitées du début à la fin du fichier.

Pour écrire dans un fichier texte :

**il faut instancier un objet de la classe StreamWriter. Puis on écrit avec Write ou WriteLine (ajoute un saut de ligne). Enfin on ferme avec Close.**

On peut instancier avec le constructeur de la classe StreamWriter et avec New, ou par la Classe File.

```
Dim SW As New StreamWriter ("MonFichier.txt") ' crée le fichier ou, si existe déjà, écrase
```

Il existe une surcharge permettant de ne pas écraser, mais d'ajouter à la fin du fichier :

```
Dim SW As New StreamWriter ("MonFichier.txt", True) ' crée ou si existe ajoute
```

Avec la classe File :

```
Dim SW As  StreamWriter=File.CreateText ("MonFichier.txt") ' crée ou si existe écrase

Dim SW As StreamWriter = File.AppendText("MonFichier.txt") ' crée ou si existe ajoute
```

Ensuite pour écrire 2 lignes :

```
SW.WriteLine ("Bonjour")

SW.WriteLine ("Monsieur")
```

Enfin on ferme :

```
SW.Close()
```

Pour lire dans un fichier Texte :

**il faut instancier un objet de la classe StreamReader. Puis on lit avec Read (un nombre d'octets) ReadLine (une ligne) ReadToEnd (de la position courante jusqu'à la fin). Enfin on ferme avec Close.**

Avec le constructeur de la Classe Stream Reader :

```
Dim SR As New StreamReader ("MonFichier.txt")
```

Avec la Classe File :

```
Dim SR As  StreamReader=File.OpenText ("MonFichier.txt") '
```

Comment lire chaque ligne du fichier et s'arrêter à la fin ?

En effet on ne sait pas habituellement combien le fichier contient de ligne, si le fichier contient 2 lignes il faut en lire 2 et s'arrêter sinon on tente de lire après la fin du fichier et cela déclenche une erreur.

**Trois solutions :**

1 Utiliser ReadToEnd qui lit en bloc jusqu'à la fin ;
2 Avant ReadLine mettre un Try : quand l'erreur 'fin de fichier' survient elle est interceptée par Catch qui sort du cycle de lecture et ferme le fichier ;
3 Utiliser Peek qui lit dans le fichier un caractère, mais sans modifier la position courante de lecture.

La particularité de Peek est de retourner -1 s'il n'y a plus de caractère à lire sans déclencher d'erreur, d'exception.

La troisième solution est la plus générale et la plus élégante :

```
Do Until SR.Peek=-1

     Ligne=SR.ReadLine()

Loop

Enfin on ferme:

SR.Close()
```

**Notion de 'Buffer', utilisation de Flush**

En fait quand on écrit des informations sur le disque, le logiciel travaille sur un buffer ou mémoire tampon qui est en mémoire vive. Si on écrit des lignes dans le fichier, elles sont 'écrites' dans le buffer en mémoire vive. Quand le buffer est plein (ou que l'on ferme le fichier) l'enregistrement du contenu du buffer est effectué effectivement sur le disque.

Ce procédé est général à l'écriture et à la lecture de fichier, mais totalement transparente, car le programmeur ne se préoccupe pas des buffers.

Parfois, par contre, même si on a enregistré peu d'informations, on veut être sûr qu'elle est sur le disque, il faut donc forcer l'enregistrement sur disque même si le buffer n'est pas plein, on utilise alors la méthode Flush.

```
   SW.Flush()
```

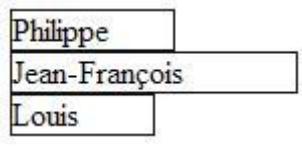
Le fait de fermer un fichier par Close, appelle automatiquement Flush() ce qui enregistre des données du buffer.

## V-AC-5 - Utiliser "FileOpen" du VisualBasic

Bonjour les anciens.

Visual Basic fournit trois types d'accès au fichier :

- **l'accès séquentiel** pour lire et écrire des fichiers 'texte' de manière continue, chaque donnée est enregistrée successivement du début à la fin; les enregistrements n'ont pas la même longueur, ils sont séparés par un séparateur (des virgules ou des retours à la ligne).



On ne peut qu'écrire le premier enregistrement puis le second, le troisième, le quatrième…

Pour lire, c'est pareil: on ouvre, on lit le premier, le second, le troisième, le quatrième…

Pour lire le troisième enregistrement, il faut lire avant les 2 premiers ;

- **l'accès aléatoire (Random)**, (on le nomme parfois accès direct) pour lire et écrire des fichiers texte ou binaire constitués d'enregistrements de longueur fixe; on peut avoir directement accès à un enregistrement à partir de son numéro.

| |
|---|
| Philippe<br>1 place de la gare |
| Jean<br>35 rue du cloître |
| Pierre<br>14 impasse du musée |
| Louis<br>sdf |

Les enregistrements ont une longueur fixe: il faut prévoir!! si on décide de 20 caractères pour le prénom, on ne pourra pas en mettre 21, le 21e sera tronqué, à l'inverse l'enregistrement de 15 caractères sera complété par des blancs.

Il n'y a pas de séparateur entre les enregistrements.

Les enregistrements peuvent être constitués d'un ensemble de variables: une structure, ici prénom et adresse.

Ensuite on peut lire directement n'importe quel enregistrement, le second enregistrement par exemple, ou écrire sur le 3e.(on comprend que, connaissant la longueur d'un enregistrement qui est fixe, l'ordinateur peut calculer la position d'un enregistrement quelconque ;

- **l'accès binaire**, pour lire et écrire dans tous les fichiers, on lit ou écrit un nombre d'octets désiré à une position désirée… C'est comme l'accès direct, on peut lire le 36e octet…

| | | | | |
|---|---|---|---|---|
| B | B | B | B | B |

En pratique

Les fichiers séquentiels sont bien pratiques pour charger une série de lignes (toujours la même) dans une ListBox par exemple.

Faut-il utiliser les fichiers séquentiels ou random (à accès aléatoire, à accès direct) pour créer par exemple un petit carnet d'adresses?

**Il y a deux manières de faire :**

- créer un fichier random et lire ou écrire dans un enregistrement pour lire ou modifier une adresse ;
- créer un fichier séquentiel. À l'ouverture du logiciel, lire séquentiellement toutes les adresses et les mettre dans un tableau (de structure). Pour lire ou modifier une adresse: lire ou modifier un élément du tableau. En sortant du programme enregistrer tous les éléments du tableau séquentiellement.(Enregistrer dans un nouveau fichier, effacer l'ancien, renommer le nouveau avec le nom de l'ancien).

Bien sûr s'il y a de nombreux éléments dans une adresse, un grand nombre d'adresses, il faut utiliser une base de données.

***Attention :*** *si on ouvre un fichier en écriture et qu'il n'existe pas sur le disque, il est créé.*

*Si on ouvre un fichier en lecture et qu'il n'existe pas, une exception est déclenchée (une erreur). On utilisait cela pour voir si un fichier existait: on l'ouvrait, s'il n'y avait pas d'erreur c'est qu'il existait. Mais maintenant il y a plus simple pour voir si un fichier existe (File.Exists).*

*Si on ouvre un fichier et que celui-ci est déjà ouvert par un autre programme, il se déclenche généralement une erreur (sauf si on l'ouvre en Binaire, c'était le cas en VB6, c'est à vérifier en VB.NET).*

Pour ouvrir un fichier, on utilise **FileOpen**.

FileOpen (FileNumber, FileName, Mode, Access, Share, RecordLength)

Paramètres de FileOpen

**FileNumber**

À tout fichier est affecté un numéro unique, c'est ce numéro que l'on utilisera pour indiquer sur quel fichier pratiquer une opération… Utilisez la fonction FreeFile pour obtenir le prochain numéro de fichier disponible.

**FileName**

Obligatoire. Expression de type String spécifiant un nom de fichier. Peut comprendre un nom de répertoire ou de dossier, et un nom de lecteur.

**Mode**

Obligatoire. Énumération OpenMode spécifiant le mode d'accès au fichier : Append, Binary, Input (séquentiel en lecture), Output (séquentiel en écriture) ou Random (accès aléatoire).

**Access**

Facultatif. mot-clé spécifiant les opérations autorisées sur le fichier ouvert : Read, Write ou ReadWrite. Par défaut, la valeur est OpenAccess.ReadWrite.

**Share**

Facultatif. Spécifiant si un autre programme peut avoir en même temps accès au même fichier : Shared (permet l'accès aux autres programmes), Lock Read (interdit l'accès en lecture), Lock Write (interdit l'accès en écriture) et Lock Read Write (interdit totalement l'accès). Le processus OpenShare.Lock Read Write est paramétré par défaut.

**RecordLength**

Facultatif. Nombre inférieur ou égal à 32 767 (octets). Pour les fichiers ouverts en mode Random, cette valeur représente la longueur de l'enregistrement. Pour les fichiers séquentiels, elle représente le nombre de caractères contenus dans la mémoire tampon.

Pour écrire dans un fichier, on utilise :

**Print**, **Write**, **WriteLine** dans les fichiers séquentiels ;

**FilePut** dans les fichiers aléatoires.

Pour lire dans un fichier, on utilise :

**Input**, **LineInput** dans les fichiers séquentiels ;

**FileGet** dans les fichiers aléatoires.

Pour fermer le fichier, on utilise **FileClose()**.

**Numéro de fichier**

Pour repérer chaque fichier, on lui donne un numéro unique (de type Integer).

La fonction **FreeFile** retourne le premier numéro libre.

```
Dim No as Integer

No= Freefile()
```

Ensuite on peut utiliser No pour repérer le fichier sur lequel on travaille.

```
FileOpen( No, "MonFichier", OpenMode.Output)

Print(No,"toto")

FileClose (No)
```

## V-AC-5-a - Fichier séquentiel en VB

Vous devez spécifier si vous voulez lire (entrer) des caractères issus du fichier (mode **Input**), écrire (sortir) des caractères vers le fichier (mode **Output**) ou ajouter des caractères au fichier (mode **Append**).

Ouvrir le fichier 'MonFichier' en mode séquentiel pour y écrire :

```
Dim No as integer

No= Freefile

FileOpen( No, "MonFichier", OpenMode.Output)
```

Pour écrire dans le fichier séquentiel : on utilise Write ou WriteLine Print ou PrintLine.

- La fonction **Print** écrit dans le fichier sans aucun caractère de séparation.

```
Print(1,"toto")

Print(1,"tata")

Print(1, 1.2)
```

Donne le fichier 'tototata1.2'

- La fonction **Write** insère des points-virgules entre les éléments et des guillemets de part et d'autre des chaines au moment de leur écriture dans le fichier, les valeurs booléennes et les variables DateTime sont écrites sans problèmes.

```
Write(1,"toto")

Write(1,"tata")

Write(1, 1.2)
```

Donne le fichier '"toto";"tata";1.2"



*Attention s'il y a des points-virgules dans les chaines, elles seront considérées comme séparateurs !! ce qui entraine des erreurs à la lecture. Il faut mettre la chaine entre "" ou bien remplacer le point-virgule par un caractère non utilisé (# par exemple) avant de l'enregistrer puis après la lecture remplacer '#' par ';'*

- Il faut utiliser **Input** pour relire ces données (Input utilise aussi le point-virgule comme séparateur.
- La fonction **WriteLine** insère un caractère de passage à la ligne, c'est-à-dire un retour chariot+ saut de ligne (Chr(13) + Chr(10)),On lira les données par LineInput.

```
WriteLine(1,"toto")

WriteLine(1,"tata")

WriteLine(1, 1.2)
```

Donne le fichier:

"toto"

"tata"

1.2

Il faut utiliser **LineInput** pour relire ces données, car il lit jusqu'au retour Chariot, saut de ligne.

Toutes les données écrites dans le fichier à l'aide de la fonction Print respectent les conventions internationales ; autrement dit, les données sont mises en forme à l'aide du séparateur décimal approprié. Si l'utilisateur souhaite produire des données en vue d'une utilisation par plusieurs paramètres régionaux, il convient d'utiliser la fonction Write

**EOF (NuméroFichier)** veut dire 'End Of File', (Fin de Fichier) il prend la valeur True si on est à la fin du fichier et qu'il n'y a plus rien à lire.

**LOF (NuméroFichier)** veut dire 'Length Of File', il retourne la longueur du fichier.

Exemple: Lire chaque ligne d'un fichier texte.

```
Dim Line As String
FileOpen(1, "MonFichier.txt", OpenMode.Input) ' Ouvre en lecture.
While Not EOF(1) ' Boucler jusqu'à la fin du fichier

Line = LineInput(1) ' Lire chaque ligne
Debug.WriteLine(Line) ' Afficher chaque ligne sur la console.


End While
FileClose(1) ' Fermer.
```

Ici on a utilisé une boucle While… End While qui tourne tant que EOF est Faux. Quand on est à la fin du fichier EOF (End of File)devient égal à True et on sort de la boucle.

## V-AC-5-b - Fichier à accès aléatoire en VB

**On ouvre le fichier avec FileOpen et le mode OpenMode.Random, ensuite on peut écrire un enregistrement grâce à FilePut() ou en lire un grâce à FileGet(). On peut se positionner sur un enregistrement précis (le 2e, le 15e) avec Seek.**

Le premier enregistrement est l'enregistrement numéro 1.

Exemple : Fichier des adresses.

Créer une structure Adresse, on utilise <VBFixedString( )> pour fixer la longueur.

```
Public Structure Adresse

   <VBFixedString(20)>Dim Nom        As String

   <VBFixedString(20)>Dim Rue        As String

   <VBFixedString(20)>Dim Ville      As String

End Structure



'Ouvrir le fichier, comme il n'existe pas, cela entraine sa création
Dim FileNum As Integer, RecLength As Long,  UneAdresse As Adresse
' Calcul de la longueur de l'enregistrement
RecLength = Len(UneAdresse)
' Récupérer le premier numéro de fichier libre.
FileNum = FreeFile
' Ouvrir le fichier.
FileOpen(FileNum, "MONFICHIER.DAT", OpenMode.Random, , , RecLength)
```

Pour écrire des données sur le second enregistrement par exemple :

```
UneAdresse.Nom = "Philippe"

UneAdresse.Rue = "Grande rue"

UneAdresse.Ville = "Lyon"

FilePut(FileNum, UneAdresse,2 )
```

Dans cette ligne de code, 'FileNum' contient le numéro utilisé par la fonction FileOpen pour ouvrir le fichier, 2 (un Long) est le numéro de l'enregistrement ou est copié la variable 'UneAdresse' et 'UneAdresse' est une variable déclarée en tant que type Adresse défini par l'utilisateur. Cela écrase l'enregistrement 2 s'il contenait quelque chose.

Pour écrire à la fin du fichier, ajouter un enregistrement, il faut connaitre le nombre d'enregistrements et écrire l'enregistrement suivant.

```
Dim last as long  'noter que le numéro d'enregistrement est un long
```

Pour connaitre le nombre d'enregistrements, il faut diviser la longueur du fichier par la longueur d'un enregistrement.

```
last = FileLen("MONFICHIER.DAT") / RecLength
```

On ajoute 1 pour créer un nouvel enregistrement.

```
FilePut(FileNum, UneAdresse,last+1 )
```

Pour lire un enregistrement (le premier par exemple) :

```
FileGet(FileNum, UneAdresse, 1)
```

Attention Option Strict doit être à false .

Si option Strict est à True, la ligne qui précède génère une erreur, car le second argument attendu ne peut pas être une variable 'structure'. Pour que le second argument de FileGet (une adresse) soit converti dans une variable Structure automatiquement Option Strict doit donc être à false. (Il doit bien y avoir un moyen de travailler avec Option Strict On et de convertir explicitement, mais je ne l'ai pas trouvé)

Remarque: si le fichier contient 4 enregistrements, on peut écrire le 10e enregistrement, VB ajoute entre le 4e et le 10e, 5 enregistrements vides. On peut lire un enregistrement qui n'existe pas, cela ne déclenche pas d'erreur.

Le numéro d'enregistrement peut être omis dans ce cas c'est l'enregistrement courant qui est utilisé.

On positionne l'enregistrement courant avec **Seek**.

Exemple : Lire le 8e enregistrement :

```
    Seek(FileNum,8)
    FileGet(FileNum,Une Adresse)
```

Suppression d'enregistrements

Vous pouvez supprimer le contenu d'un enregistrement en effaçant ses champs (enregistrer à la même position des variables vides), mais l'enregistrement existe toujours dans le fichier.

**Pour supprimer totalement un enregistrement :**

- créez un nouveau fichier ;
- copiez tous les enregistrements valides du fichier d'origine dans le nouveau fichier (pas ceux qui sont vides) ;
- fermez le fichier d'origine et utilisez la fonction Kill pour le supprimer ;
- utilisez la fonction Rename pour renommer le nouveau fichier en lui attribuant le nom du fichier d'origine.

## V-AC-5-c - Fichier binaire en VB

Dans les fichiers binaires, on travaille sur les octets.

La syntaxe est la même que pour les fichiers Random, sauf qu'on travaille sur la position d'un octet et non sur un numéro d'enregistrement.

Pour ouvrir un fichier binaire :

**FileOpen**(FileNumber, FileName, OpenMode.Binary)

**FileGet** et **FilePut** permettent de lire ou d'écrire des octets .

```
FileOpen(iFr, ReadString, OpenMode.Binary)
MyString = New String(" "c, 15)        'Créer une chaine de 15 espaces
FileGet(iFr, MyString)                 ' Lire 15 caractères dans MyString
```

```
FileClose(iFr)
MsgBox(MyString)
```

Le fait de créer une variable de 15 caractères et de l'utiliser dans FileGet permet de lire 15 caractères.

## V-AC-6 - Utilisation du Contrôle RichTextBox

Un contrôle RichTextBox est un contrôle qui contient du texte enrichi qui peut être modifié. On rappelle que du texte présent dans un contrôle RichTextBox peut être enregistré ou lu très simplement avec les méthodes .SaveFile et .LoadFile.

Le texte peut être du texte brut ou du RTF.

Comment enregistrer cetexte dans un fichier sur disque ?

```
richTextBox1.SaveFile(FileName, RichTextBoxStreamType.PlainText)
```

Si on remplace .PlainText par .RichText c'est le texte enrichi et non le texte brut qui est enregistré

Pour lire un fichier, il faut employer **.LoadFile** avec la même syntaxe.

Simple, non !!!

## V-AC-7 - Lire ou écrire des octets ou du XML

BinaryWriter et BinaryReader permettent d'écrire ou de lire des données binaires.

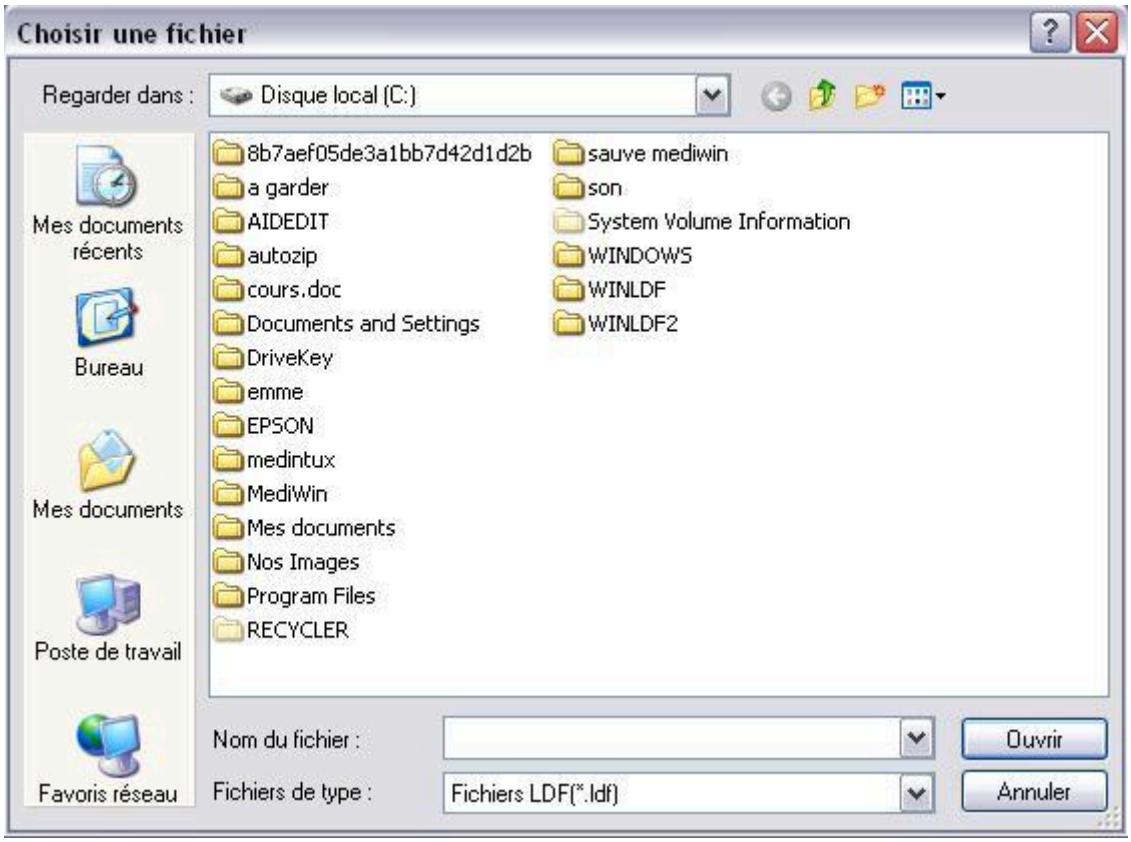
XMLTextWriter et XMLTextReader écrivent et lisent du Xml.

Pour enregistrer un tableau, un objet, Vb.Net propose aussi la Sérialization (voir ce chapitre).

## V-AC-8 - Boite de dialogue pour choix de fichier

Tapez :

```
Dim dialogOpen As New OpenFileDialog


With DialogOpen
.InitialDirectory = "C:"            'répertoire sur lequel s'ouvrira la boite
.Title = "Choisir un fichier"          'titre de la barre
.Filter = "Fichiers LDF(*.ldf)|*.ldf"   'filtre, seuls les fichiers LDF apparaitront
.ShowDialog()                           ' on ouvre la boite de dialogue enfin
        'Retour après la fermeture de la boite de dialogue
If Err.Number = 32755 Then Exit Sub     'le bouton 'annuler' a-t-il été cliqué ?
If Len(.FileName) = 0 Then  Exit Sub     'aucun choix
Dim sFile As String = .FileName           'nom du fichier choisi ( avec extension)
End With
```




Vous avez le nom du fichier à ouvrir, vous devez écrire le code pour l'ouvrir avec un Open…

**SaveFileDialog** existe aussi.

## V-AC-9 - Stream et fichier aléatoire avec structure

Comment enregistrer, lire, effacer des données (qui sont dans une variable structurée) dans un fichier binaire:

Sans utiliser de FileOpen FilePut, FileGet, mais en utilisant plutôt des FileStream (un BinaryReader et un BinaryWriter). On reste dans le Framework .Net.

Par Bruno Chappe.

Débutant s'abstenir.

Cette syntaxe est entièrement écrite en VB .NET 2005, et n'utilise que des objets avec méthodes et propriétés issues de VB .NET 2005.

On crée une Structure 'personne', une Class 'myBinaryReader', une Class 'myBinaryWriter' permettant de lire et d'enregistrer des 'personne'.

'System.IO doit être importé dans l'entête de votre module.

**ÉTAPE N°1 :** Créer la structure et les classes binaryReader et binaryWriter personnalisées.

```
'Créer la structure avec son constructeur spécifique

Structure personne
```

```
Public pNom As String

Public pPrenom As String

Public pAge As Integer

Public pMarie As Boolean

Sub New(ByVal myNom As String, ByVal myPrenom As String, ByVal myAge As Integer, ByVal
 myMarie As Boolean)

Me.pNom = myNom

Me.pPrenom = myPrenom

Me.pAge = myAge

Me.pMarie = myMarie

End Sub

End Structure



'Créer une classe de binarywriter personnalisée à partir de la classe binarywriter
' native de vb

Class myBinarywriter

Inherits System.IO.BinaryWriter

Sub New ( ByVal st1 As System.IO.Stream )

MyBase.New(st1)

End Sub

'PadRight est utilisé pour faire des chaines de caractères de longueur  fixes

'C'est indispensable pour pouvoir calculer la longueur des enregistrements
' par la suite et avoir des enregistrements qui ont tous la même longueur

Overloads Sub write( ByVal e As personne )

MyBase.Write(e.pNom.PadRight(15))

MyBase.Write(e.pPrenom.PadRight(15))

MyBase.Write(e.pAge)

MyBase.Write(e.pMarie)

End Sub

End Class



'Créer une classe de binaryreader personnalisée à partir de la classe binaryreader
' native de vb

Class myBinaryreader

Inherits System.IO.BinaryReader

Sub New ( ByVal st2 As System.IO.Stream )

MyBase.New(st2)

End Sub
```

```
Function readpersonne() As personne

Dim pNom As String = MyBase.ReadString

Dim pPrenom As String = MyBase.ReadString

Dim pAge As Integer = MyBase.ReadInt32

Dim pMarie As Boolean = MyBase.ReadBoolean

readpersonne = New personne(pNom, pPrenom, pAge, pMarie)

End Function

End Class
```

**ÉTAPE N° 2 :** Utilisation des classes personnalisées.

***Écrire un enregistrement

DEBUT

```
'Variable string stockant le chemin d'accès au fichier

Dim myFile as String

'Si l'on veut écrire directement dans un enregistrement existant
' (plutôt que d'écrire à la suite du fichier)

'il faut au préalable récupérer le rang de l'enregistrement ou l'on veut commencer à écrire

'et multiplier ce rang par la longueur d'un enregistrement (en octets).
'Cette opération nous donne la position du premier octet à écrire dans le fichier.

'Pour mémoire, la longueur d'un enregistrement en octet est égale à la taille du fichier en
 octet

'divisé par le nombre d'enregistrements (voir Annexe de ce document pour les explications)

Dim position As Integer

'Déclarer une variable de type 'personne' et assigner les valeurs voulues à ses champs

'Il est possible de travailler avec un tableau de structure, cela revient au même, mais

'est moins indispensable qu'avant vu que l'on peut aller écrire et lire directement dans le
 fichier

Dim maPersonne As personne

With maPersonne

.pNom = 'valeur du champ

.pPrenom = 'valeur du champ

.pAge = 'valeur du champ

.pMarie = 'valeur du champ

End With

'Déclare le flux et le writer qui vont nous permettre d'écrire notre structure

'Bien faire attention aux propriétés FileMode et FileAccess en fonction des opérations

'que vous voulez effectuer

'Si vous voulez écrire l'enregistrement à la suite des autres :
```

```
Dim fs As FileStream = File.Open(myFile, FileMode.Append, FileAccess.Write, FileShare.None)

'Si vous voulez réécrite/modifier un enregistrement :

Dim fs As FileStream = File.Open(myFile, FileMode.Open, FileAccess.Write, FileShare.None)

'Dans les 2 cas, remarquez ici que nous instancions un objet issu de la classe
' que nous avons créée plus haut
'et non pas un objet issu des classes natives de vb

Dim bw As New myBinarywriter(fs)

'Nécessaire que si vous souhaitez écrire à un endroit précis dans le fichier

'Où position indique la position de début d'écriture dans le flux (fs) et SeekOrigin.Begin
' indiquant à partir de quel endroit du flux commencer la recherche de position

fs.Seek(position, SeekOrigin.Begin)

'Dans tous les cas, l'instruction ci-dessous écrit l'intégralité de notre structure
' sous forme binaire dans le fichier

bw.write(maPersonne)

'SURTOUT BIEN PENSER À FERMER LES FLUX ET WRITER

bw().Close

fs().Close
```

FIN

***Lire un enregistrement

DEBUT

```
'Variable string stockant le chemin d'accès au fichier

Dim myFile as String

'Déclarer la variable qui va recevoir les informations issues du fichier

Dim maPersonne As personne

'Instancier un objet flux et un objet binaryReader

'Remarquez le FileAcces.Read pour la lecture

Dim fs As FileStream = File.Open(myFile, FileMode.Open, FileAccess.Read, FileShare.None)

'Objet instancié à partir de notre classe personnalisée myBinaryreader

Dim br As New myBinaryreader(fs)

'Si besoin d'aller lire un enregistrement précis

fs.Seek(position, SeekOrigin.Begin) 'Mêmes paramètres que pour l'écriture

'Sinon, vous ferez probablement un traitement en boucle de tous les enregistrements du fichier

Do Until fs.Position = fs.Length 'Faire jusqu'à ce que la position
' dans le flux soit égale à la longueur totale du flux

myPersonne = br.readpersonne

'Ici traitez les informations récupérées, par exemple pour afficher dans un listBox

Loop
```

```
'SURTOUT BIEN PENSER À FERMER LES FLUX ET READER

br().Close

fs().Close
```

FIN

***Supprimer un enregistrement

Au préalable il faut marquer l'enregistrement à effacer en utilisant la procédure décrite plus haut pour modifier un enregistrement. Pour cela on peut faire soit comme dans le TP en utilisant un champ "supprimé"de notre structure que l'on marque à True soit on écrit un enregistrement vide par-dessus l'enregistrement que l'on veut effacer (en affectant "Nothing" au premier champ de l'enregistrement par exemple). En résumé, n'importe quoi qui nous permet de repérer au moment de la réécriture du fichier, que cet enregistrement ne doit pas être recopié.

Commencer par créer une copie du fichier original (ne pas oublier de faire des tests sur l'existence du fichier).

```
Dim fi As New FileInfo(myFile)

Dim newFile As String = myFile & &#8220;.bck&#8221;

fi.CopyTo(newFile) 'Créé newFile et copie myFile dedans

'Ensuite on ouvre les 2 flux (1 en lecture et 1 en écriture) et les 2 binary (-reader & -writer)

Dim fsR As FileStream = File.Open(newFile, FileMode.Open, FileAccess.Read, FileShare.None)

'Ici le paramètre FileMode.Create va faire que le fichier myFile va être recréé
' par-dessus l'ancien fichier et l'écraser

Dim fsW As FileStream = File.Open(myFile, FileMode.Create, FileAccess.Write, FileShare.None)

'Après l'instruction ci-dessus on a donc un fichier myFile qui existe, mais qui est vide
' (sans avoir besoin de le supprimer d'abord et de le recréer ensuite),
' prêt à recevoir les données du fichier newFile.

Dim br As New myBinaryreader(fsR)

Dim bw As New myBinarywriter(fsW)

'On dimensionne une variable du type de notre structure

Dim maPersonne As personne

'Faire une boucle sur le fichier en lecture

Do Until fsR.Position = fsR.Length

maPersonne = br.readpersonne

'Si marquage de maPersonne n'indique pas qu'il faut effacer l'enregistrement alors :

bw.write(maPersonne)

End If

Loop

'Cette boucle va donc lire chaque enregistrement dans newFile, l'affecter dans
' une variable structurée maPersonne et si cette variable ne contient pas
' d'indicateur de suppression, l'écrire dans myFile.
' À la fin de la boucle myFile contient donc tous les enregistrements de newFile sauf ceux
 marqués à supprimer.

'On n'oublie pas de fermer les flux
```

```
br.Close()

bw.Close()

fsR.Close()

fsW.Close()

'Et de supprimer le fichier newFile qui ne nous sert plus à rien

File.Delete(newFile)
```

**ANNEXE :** Comment calculer la valeur de la variable "position" utilisée dans la méthode seek de notre FileStream.

Nous avons besoin de 3 éléments : la longueur totale du fichier en octets, la taille d'un enregistrement en octets, le numéro de l'enregistrement que l'on souhaite modifier/remplacer/effacer.

Longueur totale du fichier en octets :

```
Dim fi as New FileInfo(myFile)

Dim longueurFichier As Long

longueurFichier = fi.length 'Renvoie un entier long représentant la taille totale en octet
```

Taille d'un enregistrement en octets

```
Dim nbEnreg As Integer 'Au préalable il faut récupérer le nombre d'enregistrements,
' par exemple lors d'une première lecture du fichier séquentielle,
' en incrémentant un compteur par programmation.

Dim tailleEnreg as Integer

tailleEnreg = longueurFichier / nbEnreg
```

Pointer directement dans le fichier à l'octet voulu pour l'écriture.

Autrement dit, calculer la variable "position" pour pouvoir l'utiliser dans la méthode seek.

Il suffit de multiplier le numéro de l'enregistrement à écrire (le premier enregistrement doit avoir l'indice 0) par la taille d'un enregistrement. Par exemple, si j'ai un fichier de 120 octets avec 12 enregistrements (de 0 à 11), cela me fait des enregistrements de 10 octets de long. Si je veux aller modifier le 4e enregistrement, ma variable position sera égale à 40 (4*10). Il ne me reste plus qu'à déplacer le pointeur dans mon flux avant d'écrire ma variable, en utilisant :

```
fs.Seek(position, SeekOrigin.Begin)
```

Remarque: Le code proposé dans l'exemple permet de créer un enregistrement de 37 octets par "ligne" pour autant que l'on n'utilise pas de caractères accentués. Il faut ajouter 1 octet pour chaque caractère accentué utilisé dans les zones de texte, et ce, malgré l'usage de la fonction PadRight(15).

L'usage de Seek tel que décrit dans le même chapitre en devient impossible.

Comment corriger cela ?

# V-AD - Travailler sur les répertoires et fichiers

Il est probablement nécessaire de lire le chapitre VI sur les Classes avant de lire celui-ci.

Comment créer, copier effacer des répertoires (ou dossiers) ?

Avec les classes **DirectoryInfo** et **Directory**.

Avec la classe **Path**.

Avec la classe **Environment**.

Avec **My.Computer.FileSystem** en VS 2005.

Avec les Classes de **VisualBasic**.

Comment créer une boite de dialogue 'choix de répertoire' en VB2005 ?

## V-AD-1 - Classe DirectoryInfo et la Classe Directory du Framework

Pour travailler sur les dossiers (ou répertoires), il faut au préalable taper :

**Imports System.IO**

La classe **Directory** est utilisée pour travailler sur un ensemble de dossiers, la Classe DirectoryInfo donne des renseignements sur un dossier particulier (après instanciation ).

La Classe Directory possède les méthodes suivantes.

| | |
|---|---|
| **Exists** | Teste si le dossier existe. |
| **CreateDirectory** | Crée le dossier |
| **Delete** | Efface le dossier |
| **Move** | Déplacement de dossier |
| **GetCurrentDirectory** | Retourne le dossier de travail de l'application en cours |
| **SetCurrentDirectory** | Définit le dossier de travail de l'application. |
| **GetDirectoryRoot** | Retourne le dossier racine du chemin spécifié. |
| **GetDirectories** | Retourne le tableau des sous-dossiers du dossier spécifié. |
| **GetFiles** | Retourne les fichiers du dossier spécifié. |
| **GetFilesSystemEntries** | Retourne fichier et sous dossier avec possibilité d'un filtre. |
| **GetLogicalDrives** | Retourne les disques |
| **GetParent** | Retourne le dossier parent du dossier spécifié. |

La Classe Directory est statique: on l'utilise directement.

Exemple

**Afficher dans une ListBox les sous-dossiers (répertoires) du répertoire de l'application :**

```
Dim SousDos() As String= Directory.GetDirectories(Directory.GetCurrentDirectory)

Dim Dossier As String

For Each Dossier In SousDos

    List1.Items.Add(Dossier)

Next
```

**Afficher dans une ListBox les sous-dossiers et fichiers.**

On utilise ici la récursivité. Pour chaque sous-répertoire, on appelle la routine elle-même.

```
Imports System.IO


Sub  AfficheTree ( ByVal myDir As String, ByVal Optional Niveau As Integer =0)


'Affiche le répertoire myDir

List1.Items.Add(New String (" ", niveau *2) & myDir)


'Affiche les fichiers

For Each fichier As String  In Directory.GetFiles( myDir)

    List1.Items.Add(New String (" ", niveau *2+2) & fichier)

Next


'Parcourt les sous-répertoires

For each sousRepertoire As String In Directory.GetDirectories( myDir)

    'Appel de manière récursive 'AfficheTree pour afficher le contenu des sous-répertoires.

    AfficheTree (sousRepertoire, niveau+1)

Next


End Sub
```

La variable niveau permet de pratiquer une indentation :New String (" ", niveau*2) produit une chaine d'espace de longueur niveau *2.

On appelle cette routine avec AfficheTree (c:\myprogramme", 0) 'Pour tester éviter"c:\", car c'est très très long!!! on le fait tous pour tester!!

Directory.GetFiles et Directory.GetDirectories acceptent un argument supplémentaire qui fait office de filtre.

Directory.GetFiles( myDir, "*.txt") 'pour ne voir que les fichiers .txt.

**Afficher dans une ListBox les exécutables d'un répertoire et de ses sous-répertoires.**

Ici on utilise un argument supplémentaire qui permet de rechercher dans les sous répertoires.

```
For Each file As String In Directory.GetFiles("c:\windows", "*.exe",
 System.IO.SearchOption.AllDirectries ))

    List1.Items.Add (file)

Next
```

Génial , non ? Quelle économie de code !!

En plus il y a des méthodes permettant de retourner **dans une collection IEnumerable la liste des fichiers (Directory.EnumerateFiles) ou la liste des répertoires** (Directory.EnumerateDirectories) d'un chemin.
Exemple pour récupérer la liste des fichiers et l'afficher dans un ListBox :

```
        ' récupérer la liste des fichiers dans un répertoire
        Dim files1 As IEnumerable = _
            Directory.EnumerateFiles("c:\Article_dvp\global", "*", SearchOption.AllDirectories)
       'Mettre la liste dans une ListBox
        For Each f As String In files1
            ListBox1.Items.Add(f)
        Next
```

**Afficher les disques présents dans une ListBox :**

```
Imports System.IO


For Each disque As String In Directory.GetLogicalDrives()

    List1.Items.Add (Disque)

Next
```

**Afficher dans une ListBox les fichiers .jpg d'un répertoire :**

```
Dim dirInfo As New System.IO.DirectoryInfo ("C:\Nos Images\sicile")

Dim file As System.IO.FileInfo

Dim files() As System.IO.FileInfo = dirInfo.GetFiles("*.jpg")

If (files IsNot Nothing) Then

For Each file In files

    ListBox1.Items.Add(file.FullName)

Next

End If
```

**Changer le répertoire courant, effacer un sous-répertoire :**

```
Directory.SetCurrentDirectory ("c:\mydirectory") 'change le répertoire courant

Directory.Delete (c:\otherdirectory") 'efface ce répertoire s'il est vide
```

```
Directory.Delete (c:\otherdirectory", True) 'efface ce répertoire ses fichiers et sous-
répertoires.
```

Ah !! nostalgique du DEL *.*

Il y a d'autres méthodes pour obtenir des infos des répertoires ou des fichiers et les modifier: GetCreationTime, GetLastAccesTime, GetLastWtriteTime et les Set… correspondants.
Exemple permettant de voir la date de création d'un fichier :

```
Dim d As String = Directory.GetCreationTime("c:\Article_dvp\global\Thumbs.db")
```

### La Classe DirectoryInfo possède les propriétés suivantes :

```
Name            Nom du dossier (sans extension)

Full Name       Chemin et nom du dossier

Exists

Parent         Dossier parent

Root            Racine du dossier
```

La Classe DirectoryInfo n'est pas statique : il faut instancier un dossier avant de l'utiliser.

Il y a aussi les méthodes suivantes :

```
Create, Delete, MoveTo

CreateSubdirectory

GetDirectories    Retourne les sous-dossiers

GetFiles          Retourne des fichiers

GetFileSystemInfos
```

Exemple

### Afficher le répertoire parent d'un dossier :

```
Dim D As DirectoryInfo

D= New DirectoryInfo( MonDossier)

MsgBox(D.Parent.ToString)
```

### Créer un répertoire :

```
Dim D As DirectoryInfo

D= New DirectoryInfo( MonDossier)

D.CreateSubdirectory("monsousdossier")
```

### Effacer un répertoire et ses sous-répertoires :

```
Dim D As DirectoryInfo

D= New DirectoryInfo( MonDossier)

D.Delete(True)
```

## V-AD-2 - Classe Path

La Classe statique **Path** a des méthodes simplifiant la manipulation des répertoires.

Exemple :

```
Si  C= "C:\Windows\MonFichier.txt"

Path.GetDirectoryName(C)  'retourne "C:\Windows"

Path.GetFileName(C) retourne "Monfichier.txt"

Path.GetExtension(C) retourne ".txt"

Path.GetFileNameWithoutExtension(C) retourne "MonFichier"

Path.PathRoot(C) retourne "c:\"
```

Il y a aussi les méthodes **GetFullPath ChangeExtension, Combine, HasExtension…**

**GetFullPath**: Transforme un chemin relatif en chemin absolu à l'aide du répertoire courant.

Path.GetFullPath("monAppli.exe")) retourne "C:\MonRep\monAppli.exe" si le répertoire courant est "C:\MonRep"

**Combine**: combine bout à bout un chemin et un nom de fichier

Path.Combine("C:\MonRep", "monAppli.exe")) retourne "C:\MonRep\monAppli.exe"

## V-AD-3 - Classe DriveInfo

Nouveauté en VB 2005, la Classe DriveInfo :

Pour un disque particulier, il faut instancier un DriveInfo avec la lettre du drive, ensuite, on a accès à toutes les propriétés du lecteur.

```
Dim di As New DriveInfo ("c:")

di.Name retourne le nom  du lecteur ( "c:" ici)

VolumeLabel  Nom (label) du lecteur (en lecture écriture)

DriveType ( Fixed, Removal, CDRom, Ram, Networl, Unknown)

DriveFormat (NTFS, Fat32)

TotalSize, TotalFreeSpace, AvailableFreeSpace
```

DriveInfo.GetDrives retourne tous les disques installés

```
For Each di As DriveInfo in DriveInfo.GetDrives()

 If di.IsReady Then 'il parait qu'il faut bien tester s'il est ready!!

    MsgBox (di.VolumeLabel)

 End if


Next
```

# V-AD-4 - Classe Environment

Donne des informations concernant l'environnement et la plateforme en cours ainsi que des moyens pour les manipuler. Par exemple: les arguments de la ligne de commande, le code de sortie, les paramètres des variables d'environnement, le contenu de la pile des appels, le temps écoulé depuis le dernier démarrage du système ou le numéro de version du Common Language Runtime, mais aussi certains répertoires.

```
Environment.CurrentDirectory  'donne le répertoire courant : ou le processus en cours démarre.

Environment.MachineName       'Obtient le nom NetBIOS de l'ordinateur local.

Environment.OsVersion 'Obtient un Identificateur et le numéro de version de la plateforme en
 cours.

Environment.SystemDirectory   'Obtient le chemin qualifié complet du répertoire du système

Environment.UserName          'Obtient le nom d'utilisateur de la personne qui a lancé le thread
 en cours.
```

La fonction GetFolderPath avec un argument faisant partie de l'énumération SpecialFolder retourne le répertoire d'un tas de choses.

### Exemple : Quel est le répertoire Système ?

```
Environment.GetFolderPath(Environment.SpecialFolder.System)
```

En vb 2010 on trouve les répertoires :
Cookies ;
CDBurning ;
Desktop ;
Favories ;
History ;
Programs ;
MyMusic ;
MyPicture ;
Recent ;
SendTo ;
System ;
Templates ;
Personal (Mydocuments) ;
ProgramFiles ;
UserProfile ;
CommonDocuments, CommonMusic, CommonPictures, CommonVideos ;
MyVideos ;
Ressources ;
Windows.

### Comment récupérer le nom des disques ?

```
Dim drives As String() = Environment.GetLogicalDrives()
```

### Comment récupérer la ligne de commande ?

```
Dim arguments As String() = Environment.GetCommandLineArgs()
```

ou avec My :

```
'Afficher dans une liste Box les arguments de la ligne de commande
```

```
ListBox1.DataSource = My.Application.CommandLineArgs.ToArray
```

## V-AD-5 - Classe My.Computer.FileSystem en VS 2005

En VS 2005 la classe My.Computer.FileSystem simplifie énormément les choses :

les méthodes **CopyDirectory, CreateDirectory, DeleteDirectory, DirectoryExits** permettent de copier, créer, effacer un répertoire ou de voir s'il existe. Il existe aussi RenameDirectory et MoveDirectory.

Exemple

**Afficher dans une MsgBox True si le répertoire 'c:\MyApplication\' existe.**

```
MsgBox(My.Computer.FileSystem.DirectoryExists("c:\MyApllication\").ToString)
```

**Copier un répertoire dans un autre :**

```
My.Computer.FileSystem.CopyDirectory("c:\a\", "c:\b\")
```

**Afficher la liste des répertoires qui sont sous c:\; ici on utilise GetDirectories qui retourne une collection des répertoires.(count contient le nombre des répertoires, item () les noms.**

```
Dim i As Integer

For i = 0 To My.Computer.FileSystem.GetDirectories("c:\").Count - 1

ListBox1.Items.Add(My.Computer.FileSystem.GetDirectories("c:\").Item(i))

Next i
```

SpecialDirectories permet de connaitre certains répertoires spéciaux comme Programs, My Documents, My Music…

Exemple

```
MsgBox(My.Computer.FileSystem.SpecialDirectories.CurrentUserApplicationData)
```

My.Computer.FileSystem.Drives est une collection contenant les disques présents.

**On peut rechercher les fichiers qui contiennent un certain texte et afficher leurs noms dans une listBox.**

Grâce à My.Computer.FileSystem.FindInFiles (Répertoire, texteàchercher, respectdelacasse, tyderecherche)

```
Dim value As System.Collections.ObjectModel.ReadOnlyCollection(Of String) =
 My.Computer.FileSystem.FindInFiles
_("c:\", "Open", False, FileIO.SearchOption.SearchTopLevelOnly)

For Each name As String In value

ListBox1.Items.Add(name)

Next
```

## V-AD-6 - Les méthodes de l'espace Visual Basic

**CurDir()** retourne le chemin d'accès en cours.

```
MyPath = CurDir()
```

```
MyPath = CurDir("C"c)
```

### Dir()

Retourne une chaine représentant le nom d'un fichier, d'un répertoire ou d'un dossier qui correspond à un modèle ou un attribut de fichier spécifié ou à l'étiquette de volume d'un lecteur.

```
'vérifier si un fichier existe:

' Retourne "WIN.INI" s'il existe.
MyFile = Dir("C:\WINDOWS\WIN.INI")

' Retourne le fichier spécifié par l'extension .
MyFile = Dir("C:\WINDOWS\*.INI")

'Un nouveau Dir retourne le fichier suivant
MyFile = Dir()

' On peut surcharger avec un attribut qui sert de filtre .
MyFile = Dir("*.TXT", vbHidden) ' affiche les fichiers cachés

' Recherche les sous-répertoires.
MyPath = "c:\" ' Set the path.
MyName = Dir(MyPath, vbDirectory)
```

**ChDrive** change le lecteur actif. La fonction lève une exception si le lecteur n'existe pas.

```
ChDrive("D")
```

**MkDir** crée un répertoire ou un dossier. Si aucun lecteur n'est spécifié, le nouveau répertoire ou dossier est créé sur le lecteur actif.

```
MkDir("C:\MYDIR")
```

**RmDir** efface un répertoire ou un dossier existant.

```
' Vérifier que le répertoire est vide sinon effacer les fichiers avec Kill.
RmDir ("MYDIR")
```

**ChDir** change le répertoire par défaut, mais pas le lecteur par défaut.

```
ChDir("D:\TMP")
```

L'exécution de changements relatifs de répertoire s'effectue à l'aide de "…", comme suit :
ChDir("…") ' Remonte au répertoire parent.

### FileCopy
Copier un fichier.

```
FileCopy(SourceFile, DestinationFile)
```

### Rename
Renommer un fichier, un répertoire ou un dossier.

```
Rename (OldName, NewName)
```

**FileLen** donne la longueur du fichier, SetAttr et GetAttr pour modifier ou lire les attributs du fichier.

```
Result = GetAttr(FName)
```

**Result** est une combinaison des attributs. Pour déterminer les attributs, utilisez l'opérateur And pour effectuer une comparaison d'opérations de bits entre la valeur retournée par la fonction GetAttr et la valeur de l'attribut. Si le résultat est différent de zéro, cet attribut est défini pour le fichier désigné. Par exemple, la valeur de retour de l'expression And suivante est zéro si l'attribut Archive n'est pas défini :

```
Result = GetAttr(FName) And vbArchive
```

## V-AD-7 - Boite de dialogue 'Choix de répertoire' en VB2005

Il faut instancier un **FolderBrowserDialog**, indiquer le répertoire de départ (RootFolder), le texte de la barre (Description) et l'ouvrir avec ShowDialog.

Le répertoire sélectionné par l'utilisateur se trouve dans **SelectedPath**.

```
Dim fB As New FolderBrowserDialog

fB.RootFolder = Environment.SpecialFolder.Desktop

fB.Description = "Sélectionnez un répertoire"

fB.ShowDialog()

If fB.SelectedPath = String.Empty Then

  MsgBox("Pas de sélection")

Else

  MsgBox(fB.SelectedPath)

End If

fB.Dispose()
```


## V-AD-8 - Parcours de répertoires et de sous répertoires

Parcours de répertoires et sous-répertoires

On veut afficher dans une ListBox les noms des répertoires, sous-répertoires et fichiers en utilisant la récursivité.

On crée une routine AfficheTree qui affiche :

- le nom du répertoire courant ;

- le nom des fichiers du répertoire courant ;

- qui parcourt les sous-répertoires et pour chacun d'eux appelle AfficheTree.

```
Imports System.IO


Sub  AfficheTree ( ByVal myDir As String, ByVal Optional Niveau As Integer =0)


'Affiche le répertoire myDir
List1.Items.Add(New String (" ", niveau *2) & myDir)


'Affiche les fichiers
For Each fichier As String  In Directory.GetFiles( myDir)
    List1.Items.Add(New String (" ", niveau *2+2) & fichier)
Next


'Parcourt les sous-répertoires
For each sousRepertoire As String In Directory.GetDirectories( myDir)
    'Appel de manière récursive 'AfficheTree pour afficher le contenu des sous répertoires.
    AfficheTree (sousRepertoire, niveau+1)
Next


End Sub
```

## V-AD-9 - Fichiers et répertoires avec Linq

Lire le nom des fichiers du répertoire courant avec Linq.(VB 2008)

```
Dim myFiles= From Files in My.Computer.fyleSystem.GetFile(CurDir)
Select Files
```

## V-AE - Afficher correctement du texte

A Remarque sur le rafraîchissement de l'affichage.

B Comment afficher du texte, du numérique suivant le format désiré ?
C Comment utiliser les 'CultureInfo' ?

## V-AE-1 - Remarque sur la mise à jour de l'affichage

La mise à jour de l'affichage d'un Label (comme les autres contrôles d'ailleurs) est effectuée **en fin de Sub.**

Si on écrit :

```
Dim i As Integer

For i = 0 To 100

    Label1.Text = i.ToString

Next i
```

La variable i prend les valeurs 1 à 100, mais à l'affichage rien ne se passe pendant la boucle, VB affiche uniquement 100 à la fin.
Cela provient du fait qu'il y a une hiérarchie dans l'exécution des tâches; on a l'impression que l'affichage à une priorité faible et qu'il est effectué en fin de Sub quand la totalité du code a été exécuté.
Si on désire voir les chiffres défiler avec affichage de 0 puis 1 puis 2… il faut rafraîchir l'affichage à chaque boucle avec la méthode **Refresh() :**

```
Dim i As Integer

For i = 0 To 100

    Label1.Text = i.ToString: Label1.Refresh()

Next i
```

Une alternative est de mettre un **Application.DoEvents()** qui donne à Windows le temps de traiter les messages et de rafraîchir l'affichage.

## V-AE-2 - Afficher du texte

On a vu que pour afficher du texte il fallait l'affecter à la propriété 'Text' d'un label ou d'un textBox (ou pour des tests l'afficher sur la fenêtre 'console').

Pas de problème pour afficher des chaines de caractères, par contre, pour les valeurs numériques, il faut d'abord les transformer en 'String' et les formater (définir les séparateurs, le nombre de décimales…).

Ce n'est pas à proprement parler une conversion, mais plutôt une mise en forme.

### V-AE-2-a - ToString

On a déjà vu que pour afficher une variable numérique, il fallait la transformer en 'String' de la manière suivant :

**MyDouble.ToString**

Exemple : pour afficher dans un TextBox la valeur contenue dans la variable MyDouble:
**MyTextBox.Text=MyDouble.ToString**

ToString utilise le séparateur de la culture en cours (',' si vous être en culture française, '.' si vous êtes en culture anglaise).

Mais ToString peut être surchargé par un paramètre appelé chaine de format. Cette chaine de format peut être standard ou personnalisée.

Chaine de format standard

Cette chaine est de la forme 'Axx' ou A donne le type de format et xx le nombre de chiffres après la virgule. Le format est défini par la 'culture' en cours ( française, anglaise…)sur le thread courant.

```
Imports System
Imports System.Globalization
Imports System.Threading

Module Module1
Sub Main()

Thread.CurrentThread.CurrentCulture = New CultureInfo("en-us")'changement de culture
Dim UnDouble As Double = 123456789

Console.WriteLine("Cet exemple est  en-US culture:")
Console.WriteLine(UnDouble.ToString("C"))    'format monétaire (C) affiche $123,456,789.00
Console.WriteLine(UnDouble.ToString("E"))    'format scientifique (E) affiche     1.234568E+008
Console.WriteLine(UnDouble.ToString("P"))    'format %   (P)            affiche
 12,345,678,900.00%
Console.WriteLine(UnDouble.ToString("N"))    'format nombre (N)    affiche      123,456,789.00
Console.WriteLine(UnDouble.ToString("N4"))   'format nombre (N) 4 chiffres après la virgule,
 affiche  123,456,789.0000
Console.WriteLine(UnDouble.ToString("F"))    'format virgule fixe (F) affiche       123456789.00

End Sub
End Module
```

La 'culture' en cours est utilisée; ainsi en français le format 'N' utilise le séparateur décimal ','.

Autre exemple

**S=(1.2).ToString("C")** retourne en CurrentCulture Français (par défaut sur mon ordinateur) :1,2€

Il existe aussi D pour décimal, G pour général X pour hexadécimal.

- Chaine de format personnalisé

On peut créer de toute pièce un format, on utilise pour cela les caractères suivants :

0 indique un espace réservé de 0

Chaque '0' est réservé à un chiffre. Affiche un chiffre ou un zéro. Si le nombre contient moins de chiffres que de zéros, affiche des zéros non significatifs. Si le nombre contient davantage de chiffres à droite du séparateur décimal qu'il n'y a de zéros à droite du séparateur décimal dans l'expression de format, arrondit le nombre à autant de positions décimales qu'il y a de zéros. Si le nombre contient davantage de chiffres à gauche du séparateur décimal qu'il n'y a de zéros à gauche du séparateur décimal dans l'expression de format, affiche les chiffres supplémentaires sans modification.

# indique un espace réservé de chiffre.

Chaque '#' est réservé à un chiffre. Affiche un chiffre ou rien. Affiche un chiffre si l'expression a un chiffre dans la position où le caractère # apparait dans la chaine de format ; sinon, n'affiche rien dans cette position.

Ce symbole fonctionne comme l'espace réservé au 0, sauf que les zéros non significatifs et à droite ne s'affichent pas si le nombre contient moins de chiffres qu'il n'y a de caractères # de part et d'autre du séparateur décimal dans l'expression de format.

. (point) indique l'emplacement du séparateur décimal (celui affiché sera celui du pays )

Vous devriez donc utiliser le point comme espace réservé à la décimale, même si vos paramètres régionaux utilisent la virgule à cette fin. La chaine mise en forme apparaitra dans le format correct pour les paramètres régionaux.

, (virgule) indique l'emplacement du séparateur de millier.

Séparateur de milliers. Il sépare les milliers des centaines dans un nombre de quatre chiffres ou plus à gauche du séparateur décimal.

"Littéral" la chaine sera affichée telle quelle.

% affichera en pour cent.

Multiplie l'expression par 100. Le caractère du pourcentage (%) est inséré

E0 affiche en notation scientifique.

: et / sont séparateur d'heure et de date.

; est le séparateur de section : on peut donner 3 formats (un pour les positifs, un pour les négatifs, un pour zéro) séparés par ;

Exemples

Chaine de format '0000', avec le nombre 145 cela affiche '0145'

Chaine de format '####', avec le nombre 145 cela affiche '145'

Chaine de format '000.00', avec le nombre 45.2 cela affiche '045.20'

Chaine de format '#,#', avec le nombre 12345678 cela affiche '12,345,678'

Chaine de format '#,,' avec le nombre 12345678 cela affiche '12'

La chaine de formatage' #,##0.00 ' veut dire obligatoirement 2 chiffres après le séparateur décimal et un avant.

Si on affiche avec ce format :

1.1 cela donne 1,10

.5 cela donne 0,50

4563 cela donne 4 563,00

Exemple :

```
Dim N As Double = 19.95

Dim MyString As String = N.ToString("$#,##0.00;($#,##0.00);Zero")
```

```
' En page  U.S. English culture, MyString ="$19.95".
' En page  Française , MyString ="19,95€".
```

Exemple 2 :

```
Dim UnEntier As Integer = 42
MyString = UnEntier.ToString( "Mon nombre " + ControlChars.Lf + "= #" )
```

Affiche :
Mon nombre
= 42

Pour mémoire on a aussi d'autres manières de transformer un numérique en String :

```
Dim num As Integer = CType( chaine, String)
```

## V-AE-2-b - Str() de Microsoft.VisualBasic est toujours accepté

Il permet de transformer une variable numérique et String, qui peut ensuite être affichée.

```
MyString=Str( LeNombre)
Label1.Text=MyString
```

Pas de formatage et le séparateur décimal est le point…

## V-AE-2-c - String.Format du Framework

Permet de combiner des informations littérales à afficher sans modification et des zones formatées.

**Les arguments de String.Format se décomposent en 2 parties séparées d'une virgule.**

- Chaine de formatage entre guillemets: Exemple "{0} + {1} = {2}" : les numéros indiquent l'ordre des valeurs.
- Valeurs à afficher dans l'ordre, la première étant d'indice zéro. Exemple= A, B, A+B

Exemple :

```
Si A=3 et B=5
MsgBox(String.Format("{0} + {1} = {2}",A, B, A+B)) affiche  3+5=8
```

Autre exemple :

```
Dim MonNom As String = "Phil"
String.Format("Nom = {0}, heure = {hh}", MonNom, DateTime.Now)
```

Le texte fixe est "Nom =" et ", heure =", les éléments de format sont "{0}" et "{hh}" et la liste de valeurs est MonNom et DateTime.Now.

Cela affiche: Nom = Phil Heure= 10

**Là aussi on peut utiliser les formats**

- Prédéfinis : ils utilisent là aussi les paramètres régionaux. Ils utilisent C, D, E, F,G,N,P,R,X comme ToString.

```
MsgBox(String.Format("{0:C}",-456.45))    Affiche -456,45€
```

```
MsgBox(String.Format("{0:D8}", 456))    Affiche 00000456    Décimal 8 chiffres

MsgBox(String.Format("{0:P}", 0.14))    Affiche   14%    Pour cent

MsgBox(String.Format("{0:X}", 65535))    Affiche  FFFF    Hexadécimal
```

- Personnalisés : avec des # et des 0 :

```
MsgBox(String.Format("{0:##,##0.00}", 6553.23))
```

La fonction **Format** de Microsoft.VisualBasic (pas la classe String.Format) fournit des fonctions similaires, mais les arguments sont dans l'ordre inverse (valeur, chaine de formatage) et il n'y a pas de numéro d'ordre et de{} !! C'est pratique pour afficher une seule valeur, mais c'est quand même à éviter.

```
MyStr = Format(5459.4, "##,##0.00") ' Returns "5,459.40".
MyStr = Format(334.9, "###0.00") ' Returns "334.90".
MyStr = Format(5, "0.00%") ' Returns "500.00%"
```

## V-AE-3 - CultureInfo

On se rend compte que l'affichage est dépendant de la **CurrentCulture du Thread en cours**.

Exemple

Si la CurrentCulture est la 'CultureInfo Us' et que j'affiche avec le format 'C' (monétaire) cela affiche un $ avant, si je suis en 'CurrentCulture Français' cela affiche un € après.

Par défaut la CultureInfo est celle définie dans Windows, probablement 'fr-FR' sur votre ordinateur.
fr signifie français et FR signifie 'région France'; il existe fr-CH (Suisse), fr-BE (Belgique).
Dans en-US, en signifie anglais et US signifie USA.

On peut modifier la CurrentCulture par code (voir exemple plus haut).

```
Imports System.Threading

Thread.CurrentThread.CurrentCulture = New Globalization.CultureInfo("en-us")'passage en culture
 US culture
```

On peut utiliser l'objet **My**.

```
MsgBox(My.Application.Culture.ToString) 'affiche 'fr-FR'

(My.Application.ChangeCulture 'permettra de changer la culture )
```

On peut aussi modifier le cultureInfo uniquement sur une instruction ToString ou Format :

```
Dim d As Double=12.35
Dim s As String= d.ToString( New CultureInfo("en-US")
```

En français par défaut :

le séparateur de décimal numérique est le '.'

Exemple : 1.20

Le séparateur décimal monétaire est la ','

Exemple : 1,20€

## V-AF - Méthode d'extension, Lambda expression

Ce sont des nouveautés de VB 2008, débutant passe ton chemin.

**Méthodes d'extension**

Permet d'ajouter des fonctionnalités à un Type (sans devoir faire une Classe dérivée).

Exemple

Soit le Type 'String', je veux y ajouter une méthode Print qui affichera la String sur la console :

```
Imports System.Runtime.CompilerServices

Module StringExtensions

    <Extension()> _
    Public Sub Print(ByVal aString As String)
        Console.WriteLine(aString)
    End Sub

End Module
```

C'est le "ByVal aString As String" qui indique que c'est une extension sur les 'String'.

Comment utiliser la méthode Print ?

```
Imports ConsoleApplication2.StringExtensions

Module Module1

    Sub Main()

        Dim exemple As String = "Bonjour"
        ' Appel de  l'extension method Print.
        exemple.Print()

        ' Appel de la méthode d'instance  'ToUpper'.
        exemple.ToUpper()
        exemple.ToUpper.Print()

    End Sub

End Module
```

Si on veut ajouter un paramètre à la méthode Print, il faut l'ajouter au premier paramètre qui lui indique le DataType.

```
<Extension()> _
Public Sub PrintPonctuation(ByVal aString As String,  ByVal punc As String)
    Console.WriteLine(aString & punc)
End Sub
```

Ensuite pour l'utiliser :

```
Dim exemple As String = "Exemple"
exemple.PrintPonctuation(".")
```

### Lambda Expression

Une expression lambda est une fonction permettant de calculer et retourner une valeur unique. Exemple: Créons une expression lambda qui incrémente un Integer. Création de la fonction :

```
Dim ajoute1 = Function(num As Integer) num + 1
```

Utilisation de la fonction dans la même sub :

```
Console.WriteLine(ajoute1(5))   Affiche 6.
```

On dit que la fonction lambda 'ajoute1( num As Integer)' conduit à num+1.

On peut déclarer et utiliser la fonction en même temps :

```
Console.WriteLine((Function(num As Integer) num + 1)(5))
```

Dans ce cas il n'y a pas de nom de fonction.

Attention

On n'a pas de 'End Function'(dans les expressions lambda à une ligne) ni de 'Return' ni de 'As', on ne peut pas utiliser les génériques.

Si on veut déclarer l'expression lambda dans la tête du module afin d'avoir un accès public, c'est plus complexe :

```
Class Window1

Delegate Function ajoute(ByVal num As Integer) As Integer

Public ajoute1 As ajoute = Function(num) num + 1



Private Sub Button_Click(ByVal sender As System.Object, ByVal e As
 System.Windows.RoutedEventArgs) _
 Handles Button.Click

    MsgBox(ajoute1(3).ToString)

End Sub


End Class
```

Voyons comment on peut passer à une Sub une fonction lambda en argument. On crée une fonction 'testResult' qui a pour argument une valeur et une fonction lambda. Cette fonction affiche "Success" ou "Failure" en fonction de la valeur True ou False retournée par la fonction lambda qui a reçu la valeur. Pour utiliser cette sub on l'appelle avec comme argument la valeur à tester et la fonction Lambda.

```
Module Module1
 Sub Main()
'On appelle une fonction en envoyant une valeur et une fonction lambda.
' La ligne affiche "Success", car 4 est pair.
testResult(4, Function(num) num Mod 2 = 0)
' La ligne affiche "Failure", car 5 n'est pas > 10.
testResult(5, Function(num) num > 10)
 End Sub

' Sub testResult a 2 arguments, 'value' un Integer  et 'fun' la fonction lambda
' On teste la fonction lambda 'fun(value)'
```

```
' en fonction du résultat True ou False on affiche "Success" ou "Failure"
Sub testResult(ByVal value As Integer, ByVal fun As Func(Of Integer, Boolean))
If fun(value) Then
 Console.WriteLine("Success")
Else
 Console.WriteLine("Failure")
End If
End Sub
End Module
```

En vb 2010 on peut créer une **expression lambda sur plusieurs lignes**, on ajoute dans ce cas un 'End Function'.

```
Dim paireoupas = Function(x)
                     If (x Mod 2=0) Then
                      Return "paire"
                     Else
                      Return "Impair"
                    End If
                End Function

' Affiche 2.
Console.WriteLine(paireoupas(1))
```

En plus, on peut créer une Sub lambda.

## V-AG - L'espace de noms 'My'

Ce chapitre est placé ici, car il ne concerne pas l'interface, mais il sera plutôt lu dans un second temps.

Cet espace de noms comporte des objets qui sont des **chemins d'accès simplifiés à de nombreux domaines** touchant l'application, l'ordinateur, les ressources, les imprimantes…

My qui est extrêmement pratique est présent à partir de VB 2005 et uniquement dans VB (pas dans C#).

My : le SUPER RACCOURCI.

## V-AG-1 - My.Application

**My.Application** permet d'accéder rapidement aux propriétés de l'application en cours.

Vous pouvez ainsi récupérer des informations sur l'assembly, la culture (langue), de l'application.

Vous pouvez aussi avec My.Apllication.Info récupérer des informations sur le répertoire de travail le titre le copyrhigt de l'application.

```
'Culture
MsgBox(My.Application.Culture.ToString) ' culture du thread en cours affiche 'fr-FR'
My.Application.ChangeCulture (it-IT) 'permettra de changer la culture

'Formulaire
'My.Application.OpenForms qui retourne la collection des formulaires ouverts.
'Exemple: rajouter le texte 'ouvert' à la barre  des formulaires ouverts:
For Each F As System.Windows.Forms.Form In My.Application.OpenForms
F.Text += "[ouvert]"
Next

'modifier la durée de l'affichage de l'écran Splash
My.Application.MinimumSplashScreenDisplayTime = 2000 '<= À rajouter dans Application_New
'voir chapitre sur écran Splash

'Afficher dans une liste Box les arguments de la ligne de commande
```

```
ListBox1.DataSource = My.Application.CommandLineArgs.ToArray

'Afficher les infos de l'application
MsgBox(My.Application.Info.DirectoryPath) 'affiche le nom du répertoire ou est l'exécutable.
MsgBox(My.Application.Info.Title) 'affiche le titre de l'application, de l'exécutable.
MsgBox(My.Application.Info.Version.ToString) 'affiche le nom du répertoire ou est l'exécutable.
MsgBox(My.Application.Info.ProductName) 'affiche le nom de produit de l'application.
```

## V-AG-2 - My.Computer

My.Computer
permet d'accéder aux propriétés de l'ordinateur, du hardware.
Aux ressources logicielles et/ou matérielles de l'ordinateur.
**My.Computer.Audio :** permet de jouer des fichiers wav, ainsi que les sons systèmes de Windows.

```
My.Computer.Audio.Play("c:\mysound.wav")
My.Computer.Audio.Play("c:\mysound.wav", AudioPlayMode.BackgroundLoop) 'joue en boucle
My.Computer.Audio.Stop  'stop la boucle
```

**My.Computer.Clipboard :** permet de récupérer des informations sur le contenu du presse-papier, de récupérer et de définir son contenu.

```
If My.Computer.Clipboard.ContainsImage Then
    PictureBox1.Image = My.Computer.Clipboard.GetImage
ElseIf My.Computer.Clipboard.ContainsText Then
    TextBox1.Text = My.Computer.Clipboard.GetText
End If
```

**My.Computer.Clock :** permet de récupérer l'heure courante ainsi que le nombre de millisecondes écoulées depuis le démarrage.

```
MsgBox(My.Computer.Clock.LocalTime.ToString) 'Affiche date et heure
```

**My.Computer.FileSystem**: permet de trouver des répertoires et d'effectuer les opérations d'entrées/sorties standards.

On peut ainsi **lire le chemin des répertoires habituels :**

```
    MsgBox(My.Computer.FileSystem.CurrentDirectory)  'répertoire courant
MsgBox(My.Computer.FileSystem.SpecialDirectories.MyDocuments) 'répertoire documents
MsgBox(My.Computer.FileSystem.SpecialDirectories.CurrentUserApplicationData)'rep données
 utilisateur en cours
MsgBox(My.Computer.FileSystem.SpecialDirectories.AllUsersApplicationData)'rep données utilisateur
 en cours
MsgBox(My.Computer.FileSystem.SpecialDirectories.Programs) 'répertoire des programmes
```

Il y a aussi Destock, MyMusic, MyPictures, ProgramsFiles.

Récupérer le nom du premier disque :

```
MsgBox(My.Computer.FileSystem.Drives.Item(0).ToString)'affiche 'A:'
```

La collection Drives contient des items indiquant les disques.
GetDrives et GetDriveInfo permettent de récupérer une collection de disques présents, et des informations sur les disques.

Un répertoire existe-t-il ?

```
MsgBox(My.Computer.FileSystem.DirectoryExists("c:\").ToString) 'affiche True si c:\ existe.
```

De même pour un fichier :

```
MsgBox(My.Computer.FileSystem.FileExists("c:\myfiel.txt").ToString) 'affiche True si c:
\myfile.txt existe.
```

**Possibilité de copier, créer, effacer, déplacer répertoires ou fichiers**

Il y a :
CopyDirectory
DeleteDirectory
RenameDirectory
MoveDirectory
CreateDirectory
et
CopyFile
DeleteFile
RenameFile
MoveFile

Exemple : copie d'un répertoire :

```
My.Computer.FileSystem.CopyDirectory(sourcedirectory, destinationdirectory)
```

On peut ajouter 2 paramètres, pour afficher une boite de dialogue qui indique la progression de la copie et pour générer une interruption si l'utilisateur annule le processus (il faut dans ce cas que le code soit dans un Try Catch).

Voir les sous-répertoires :

```
My.Computeur.FileSystem.GetDirectories("c:\").item(0) 'permet de voir le premier sous répertoire.
```

Afficher dans ListBox1 tous les répertoires qui sont dans C :

```
ListBox1.DataSource = My.Computer.FileSystem.GetDirectories("c:\")
```

GetFiles fait de même avec les fichiers et GetDrives pour les disques.

Mettre le contenu d'un fichier texte dans une variable :

```
Dim LeTexte As String = My.Computer.FileSystem.ReadAllText("c:\devicetable.log")
```

(Il existe aussi WriteAllText, ReadAllBytes et WriteAllBytes)

**FindInFiles** permet de retrouver les fichiers contenant un texte.

```
Dim list As System.Collections.ObjectModel.ReadOnlyCollection(Of String)
liste = My.Computer.FileSystem.FindInFiles("C/", "Chaine à chercher", True,
 FileIO.SearchOption.SearchAllSubDirectories)
```

**My.Computer.Info :** Obtient des informations concernant l'ordinateur et le système d'exploitation (mémoire vive libre, nom de l'os, version de l'os, etc.).

```
MsgBox(My.Computer.Info.TotalPhysicalMemory.ToString) 'affiche la mémoire physique
```

Il y a aussi AvailablePhysicalMemory, TotalPhysicalMemory, TotalVirtualMemory OSVersion, OSFullName…

**My.Computer.Keyboard :** permet de tester l'état des touches CTRL, ALT, etc., et de simuler l'appui de touches grâce à la méthode Sendkeys.

```
MsgBox(My.Computer.Keyboard.AltKeyDown.ToString)' teste si la touche Alt est enfoncée.

My.Computer.Keyboard.SendKeys("a")' simule l'appui d'une touche.
```

**My.Computer.Mouse :** permet de récupérer des informations sur la souris (présence de la souris, présence de molette, boutons inversés, etc.)

```
MsgBox(My.Computer.Mouse.WheelExists.ToString) 'affiche True s'il y a une molette.
```

**My.Computer.Name :** récupère le nom de l'ordinateur.

```
MsgBox(My.Computer.Name.ToString)
```

**My.Computer.Network :** permet de télécharger et d'uploader des fichiers, de vérifier si l'ordinateur est connecté à Internet, d'effectuer des pings, et de récupérer les événements lors des connexions et déconnexions.

Charger un fichier à partir du réseau :

My.Computer.Network.DownloadFile(AdresseCompleteFichierACharger, DestinationFileNane)

```
With My.Computer.Network
    If .IsAvailable And .Ping(txtIpAdress.text) Then
        .UploadFile("c:\filetupload.ext", txtIpAdress.Text)
    End If
End With
```

**My.Computer.Ports :** permet de récupérer la liste des ports séries, et de les ouvrir.

**My.Computer.Printers :** permet de récupérer la liste des imprimantes installées et de définir l'imprimante par défaut. (absent dans la version bêta)

**My.Computer.Registry :** permet de manipuler la base de registre facilement.

**My.Computer.Screen :** permet de récupérer les informations concernant les écrans installés.

```
    MsgBox (My.Computer.Screen.Bounds.Height.ToString) 'voir la hauteur de l'écran
```

## V-AG-3 - My.User

**My.User** permet de récupérer les informations sur l'utilisateur courant.

My.User.Identity.Name

My.User.IsInRole("Administrators") 'contient l'administrateur

## V-AG-4 - My.Ressources

**My.Ressources** permet de manipuler et récupérer très facilement les ressources incorporées à l'assembly.

Mettre une ressource image dans le plan BackGround d'un formulaire :

```
Me.BackgroundImage = My.Resources.Form1Background
```

Mettre une ressource image dans le plan BackGround d'un bouton :

```
MyButton.BackGroundImage= MonProgramme.My.Ressources.Ressources.button_Blue
```

Mettre une ressource icône comme icône d'un formulaire :

```
Me.Icon = My.Resources.MyIcon
```

Jouer un son qui est dans les ressources :

```
My.Computer.Audio.Play(My.Resources.Form1Greeting, AudioPlayMode.Background)
```

## V-AG-5 - My.Setting

**My.Setting :**fichiers de configuration.

En VB2005 vous pouvez créer des paramètres de type Color, Font, Point, Size :

```
My.Settings.MyFont= New Font (Me.Font, FontStyle.Italics)
```

Ce paramètre sera enregistré automatiquement lors de la fermeture de l'application (voir chapitre sur la configuration).

Quand vous exécuterez ensuite le programme, vous pourrez récupérer le paramètre :

```
TextBox1.Font= My.Settings.MyFont
```

## V-AG-6 - My.Forms

**My.Forms**: Donne accès à tous les formulaires. Si un formulaire se nomme HelpForm, pour l'afficher :

```
My.Forms.HelpForm.Show()
```

On peut étendre et personnaliser l'espace de noms My, mais c'est un peu complexe.

## V-AH - Son, musique, batteries

Le plus simple : faire entendre un bip :

```
Beep()
```

Pour faire entendre un fichier wav, on utilise un SoundPlayer.

```
'On instancie un nouveau SoundPlayer, on utilise un son système ici
Dim MySoundPlayer = New System.Media.SoundPlayer("C:\Windows\Media\chord.wav")

'On joue le son:
MySoundPlayer.Play()
```

La méthode Play joue le son de manière asynchrone (avec un autre Thread): le code se poursuit même pendant l'émission d'un son long. PlaySync joue le son de manière synchrone.

On peut écouter le son en boucle avec PlayLoop dans ce cas il faut l'arrêter avec Stop.

```
Public Class Form1
```

```
    Public MySoundPlayer As System.Media.SoundPlayer = New System.Media.SoundPlayer("C:\Windows
\Media\chord.wav")

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
 Button1.Click


        MySoundPlayer.PlayLooping()

    End Sub

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

        MySoundPlayer.Stop()

    End Sub
End Class
```

On peut indiquer où se trouve le fichier son avec **SoundLocation** (chemin ou Url Internet), le charger avec **Load()**(synchrone avec le même thread) ou **LoadAsync** (Asynchrone avec un autre thread) puis le jouer avec **Play()**.

Les sons system (Asterisk, Beep, Exclamation, Hand et Question) peuvent être joués encore plus simplement avec SystemSounds :

```
    System.Media.SystemSounds.Asterisk.Play()
```

### Information sur les batteries d'un portable :

```
Dim BatteryLife As Integer 'contient la durée de vie restante de la batterie en %
Dim BatteryLifeRestant As Integer 'contient la durée de vie restante de la batterie en sec

BatteryLife = SystemInformation.PowerStatus.BatteryLifePercent * 100 'récupérations du
 pourcentage, 255 si inconnue
BatteryLifeRestant = SystemInformation.PowerStatus.BatteryLifeRemaining 'récupération de la durée
 en sec,-1 si inconnue

'si le secteur est branché alors 'LifeRemaining' = -1
'si batterie seule alors 'LifeRemaining' = x secondes

'mais problème parfois, car la valeur de 'PowerLineStatus' est rafraîchie plus souvent par le
 framework que la valeur de 'LifeRemaining'
```

# VI - Classes


Voir la vidéo : **au format 'Flash'**> ou **au format 'Avi'** en Visual Basic 2005.

La vidéo (identique à celle du chapitre sur les objets) contient :

1. Objets, classes :

2. Références, espaces de noms.

## VI-A - Espace de noms, Classes, Objet

### VI-A-1 - Classes

Nous avons vu qu'on utilise des objets. Des objets 'visuels' pour les formulaires, boutons… ou des objets comme les variables, les collections, des objets mathématiques…

Il existe des '**types**' d'objets qui définissent les caractéristiques communes des objets. Ces types se nomment les Classes.

Exemple

La Classe System.Windows.Forms contient les objets 'Forms' et les 'Control' (formulaire= fenêtre et contrôles).

On rappelle que se sont ces classes que l'on utilise pour instancier (créer) un objet, une instance.

```
Dim B As New Form ' crée un formulaire( une instance de formulaire)
'à partir de la Classe Form (Fenêtre).
```

B hérite de toutes les caractéristiques de la Classe Form.

Les classes sont regroupées en bibliothèques sous la dénomination '**Espace de noms**' (NameSpace).

Un **Framework** est un ensemble d'espace de noms et de classes.

### VI-A-2 - Essayons de comprendre

**Pour utiliser un objet en VB (fourni par le Framework par exemple), il faut :**

A que la Dll correspondante soit chargée dans le projet. (La Dll c'est un fichier exécutable '.dll' qui contient le code nécessaire; le Framework en comporte plusieurs). En VB.NET on appelle cela la 'Référence' (dans 'Explorateur de solutions'). Exemple de Dll :

```
System.dll
```

B que l'espace de noms soit importé : une Dll contient des espaces de noms dans lesquels se trouvent les Classes. Pour utiliser une Classe, il faut inclure l'espace de noms correspondant dans le programme donc il faut l'importer à partir de la Dll. On va par exemple importer l'espace de noms 'System.Windows.Forms' (il est dans System.dll et contient les Classes 'Form' et 'Control') :

```
Imports System.Windows.Forms
```

C on peut maintenant utiliser les Classes contenues dans cet espace de noms et créer un objet. Par exemple on va créer une fenêtre avec la Classe Form contenue dans System.Windows.Forms.

```
Dim Form1 As New Form
```

**Pour utiliser un objet en VB (fourni par le Framework par exemple), il faut :**

Form1 est donc un objet formulaire qui hérite de tous les membres (Propriétés, méthodes) de la Classe Form, on peut donc utiliser une méthode de cet objet.

```
Form1.Show()     'pour faire apparaitre la fenêtre
```

ou une propriété :

```
Form1.BackColor=Color.Red 'pour modifier la couleur de fond
```

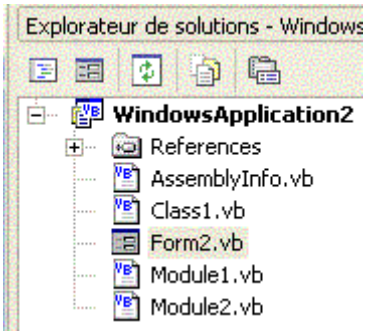
*Les Classes les plus courantes sont déjà chargées et disponibles, ce qui simplifie un peu les choses.*

## VI-A-3 - Détails en VB 2003

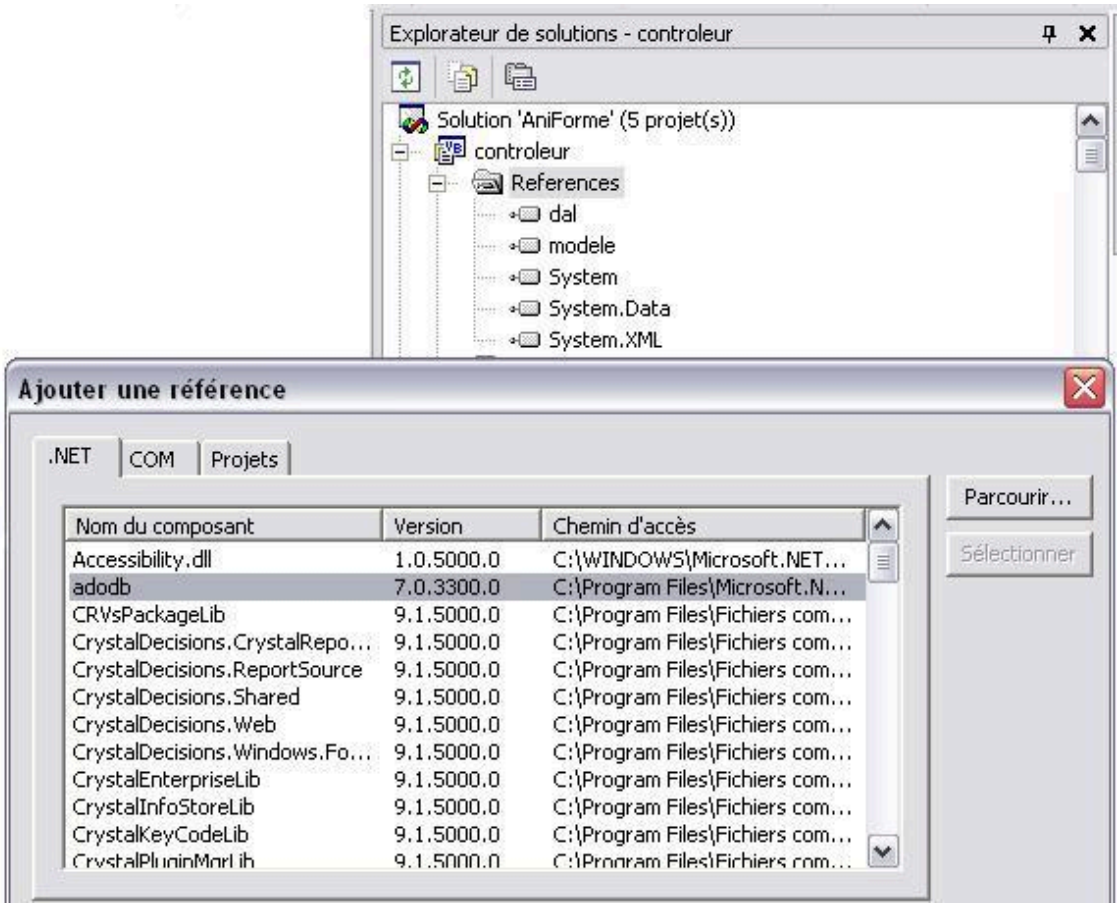
### VI-A-3-a - Les références

Pour qu'une classe soit utilisée, il faut que le composant correspondant (la DLL) soit chargé. On a vu que par défaut quelques composants du Framework (System.dll…) et le CLR (mscorlib.dll) étaient chargés.

Dans 'Explorateur de solutions' :




Double-cliquez la rubrique références pour voir les DLL (références déjà chargées).




Si vous souhaitez utiliser un autre composant dans votre application et qu'il n'est pas chargé, il faut ajouter la référence de ce composant. Dans la fenêtre de l'explorateur de solutions, cliquez le bouton droit de la souris puis cliquez sur 'Ajouter une référence'. La boite de dialogue 'Ajouter une référence de Visual Studio .NET' propose trois options :

- .NET - Répertorie tous les composants .NET Framework pouvant être référencés. (Certains sont déjà chargés comme System…) et les composants extérieurs NET ;
- COM - Répertorie tous les composants COM pouvant être référencés (ceux qui fonctionnaient en VB6) ;
- Projets - Répertorie tous les composants réutilisables créés dans des projets locaux.

## VI-A-3-b - Importation d'espace de noms

Certains espaces de noms ne sont pas chargés, l'espace de noms Math n'est pas chargé par exemple. (Bien que la référence, la dll qui se nomme System soit présente dans le projet.)

Si je veux utiliser Round pour arrondir un nombre il faut d'abord importer l'espace de noms 'Math'.

Pour cela il faut taper en haut de la fenêtre (au-dessus de public Class).

```
Imports System.Math
```

Ensuite, on peut écrire :

```
Label1.Text = (Round(1.2)).ToString 'qui affiche 1.
```

Si l'Import n'a pas été fait, **System.Math.Round(1.2)** est accepté aussi. (À condition que la Dll soit chargée.)

Noter bien que dans notre exemple, comme Math fait partie de System, la référence (la Dll correspondante) est déjà chargée.

Autre exemple : si on veut utiliser les fichiers, il faut importer **System.IO**.

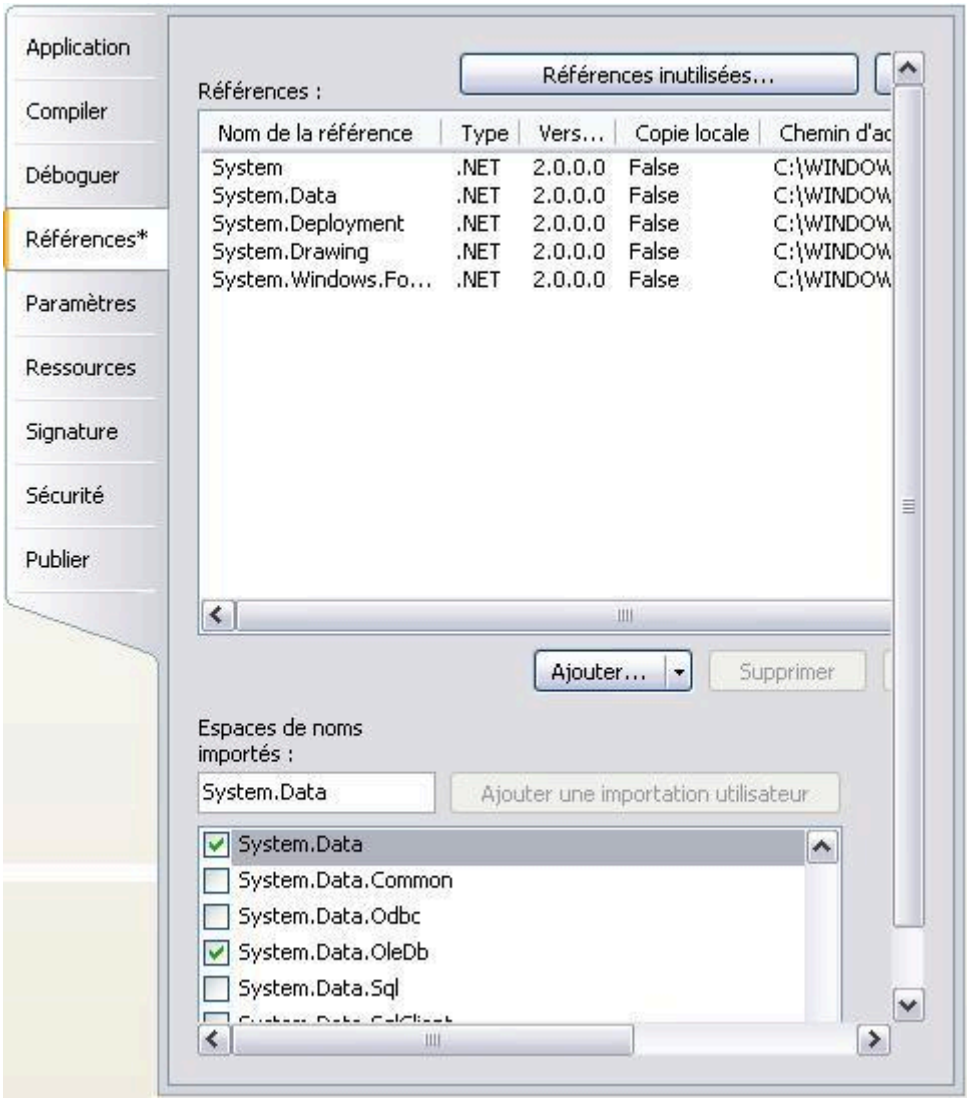
## VI-A-4 - Détails en VB 2005 2008 2010

### VI-A-4-a - Les références

Pour qu'une classe soit utilisée, il faut que le composant correspondant (la DLL) soit chargé. On a vu que par défaut quelques composants du Framework (System.dll…) et le CLR ( mscorlib.dll) étaient chargés.

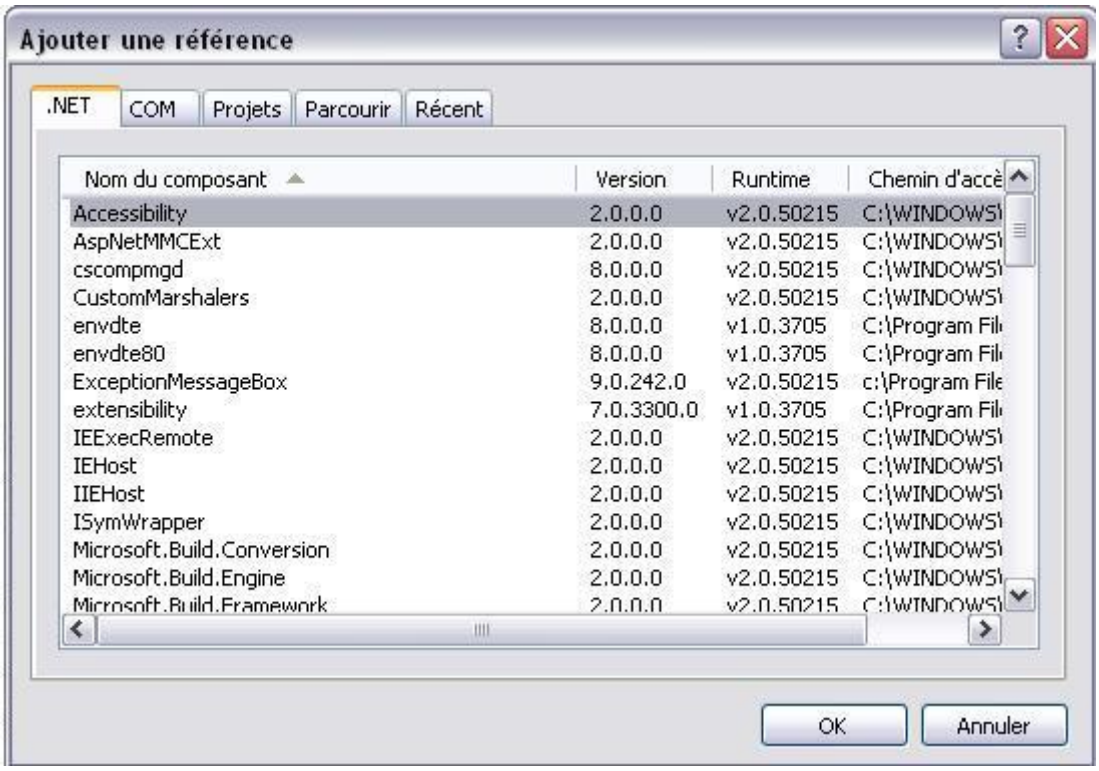
Dans 'Explorateur de solutions' double-cliquez sur 'My Projet', ou dans le menu Projet, cliquez sur propriétés de…

Puis choisir l'onglet 'Références'




On voit la liste des références chargées dans le projet : ici System, System.Data…

Si vous souhaitez utiliser un autre composant dans votre application et qu'il n'est pas chargé, il faut ajouter la référence de ce composant. Cliquez le bouton 'Ajouter…'. La boite de dialogue 'Ajouter une référence' s'ouvre :




**Elle propose :**

- .NET - Répertorie tous les composants .NET Framework pouvant être référencés. (Certains sont déjà chargés comme System…) et les composants extérieurs NET ;
- COM - Répertorie tous les composants COM pouvant être référencés (ceux qui fonctionnaient en VB6) :
- Projets - Répertorie tous les composants réutilisables créés dans des projets locaux. (D'autres programmes VB) ;
- Parcourir permet de rechercher ; Récent liste les DLL récemment chargées.

En cliquant sur une référence puis sur 'OK', cela charge la référence.

## VI-A-4-b - Importation d'espace de noms

Certains espaces de noms ne sont pas chargés, l'espace de noms Math n'est pas chargé par exemple. (Bien que la référence, la dll qui se nomme System soit présente dans le projet.)

Si je veux utiliser Round pour arrondir un nombre il faut d'abord importer l'espace de noms 'Math'.

Pour cela il faut taper en haut de la fenêtre (au-dessus de public Class).

```
Imports System.Math
```

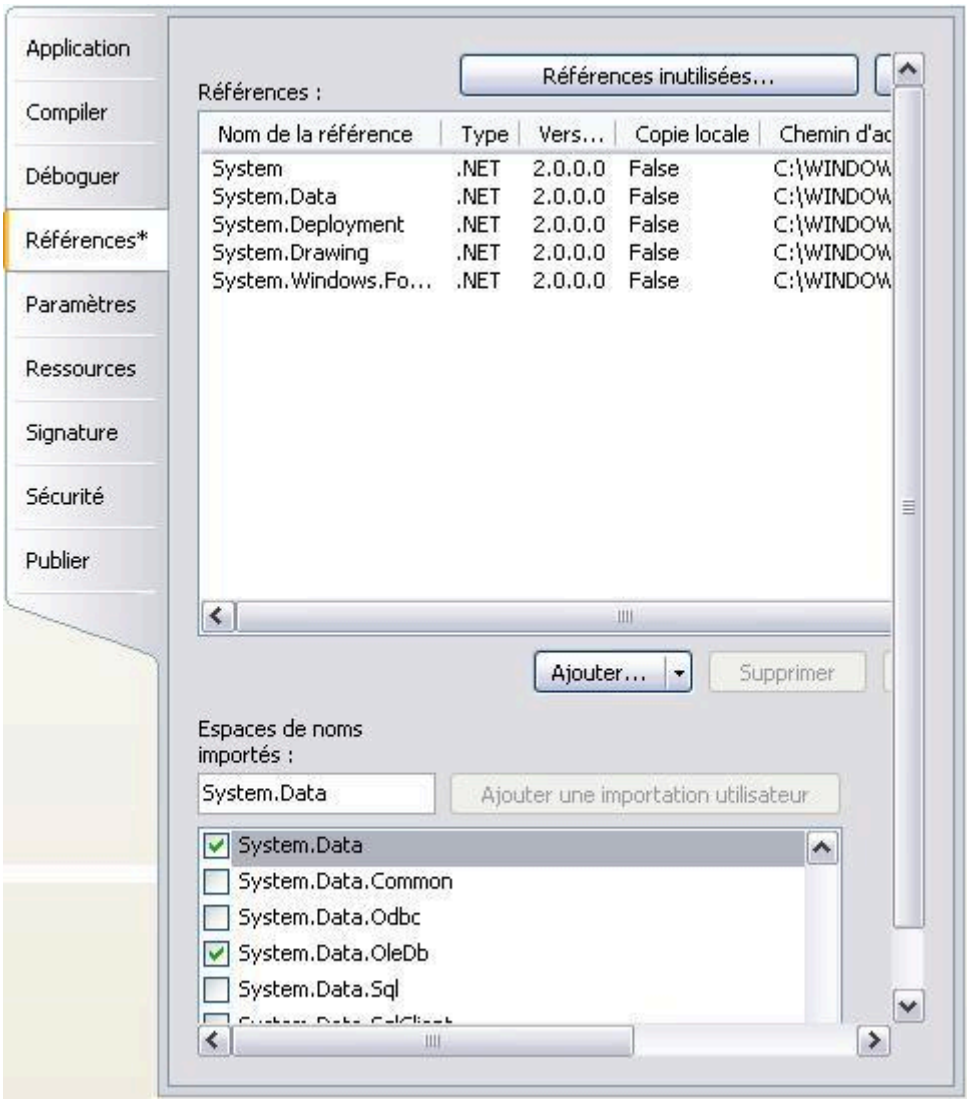
Ensuite, on peut écrire :

```
Label1.Text = (Round(1.2)).ToString 'qui affiche 1.
```

Si l'Import n'a pas été fait, System.Math.Round(1.2) est accepté aussi. (À condition que la Dll soit chargée.)

Noter bien que dans notre exemple, comme Math fait partie de System, la référence (la DLL correspondante) est déjà chargée.

On peut aussi importer les espaces de noms directement depuis l'onglet références (après avoir cliqué sur 'My Projet') :




Il suffit de choisir la dll en haut pour voir l'afficher les espaces de noms contenus dans cette dll en bas et de cocher ceux que l'on veut charger.

## VI-A-4-c - Portée de l'espace de noms

Si un seul espace de noms est spécifié (Imports System), tous les membres à nom unique de cet espace de noms sont présents. Si un espace de noms et le nom d'un élément de l'espace de noms sont spécifiés (Import System.Math), seuls les membres de cet élément sont disponibles sans qualification.

Exemple :

Imports System

permet d'utiliser System.ArgumentException, mais pas Systeme.Math.round.

Pour utiliser Round il faut Importer System.Math.

## VI-A-4-d - Propriété ambiguë

Certaines propriétés sont communes à plusieurs classes, il peut y a avoir ambiguïté et il faut utiliser dans ce cas la syntaxe complète : 'EspacedeNom.Classe'.

C'est le cas pour Left qui est une propriété de Microsoft.VisualBasic.String, mais aussi une propriété des contrôles.

```
MonControle.Left=250  'est accepté

Chaine= Left(C,2)     'pose des problèmes.
```

Pour lever l'ambiguïté, il faut écrire Microsoft.VisualBasic.Left(C,i) par exemple quand on utilise Left pour manipuler des chaines (car Left fait partie de l'espace Microsoft.VisualBasic).

```
Chaine= Microsoft.VisualBasic.Left(C,2) 'est accepté.
```

On voit là tout l'intérêt des espaces de noms qui permettent d'avoir plusieurs éléments (classe, propriétés…) de même nom, mais dans des espaces de noms différents.

## VI-A-4-e - Alias

Parfois pour simplifier l'écriture ou pour éviter des ambiguïtés on peut utiliser des Alias.

Imports STR= Microsoft.VisualBasic.Strings importe l'espace de noms String, mais le désigne sous le nom de STR (STR est un Alias), STR est utilisé ensuite :

```
Chaine=STR.Left(C,i)
```

*En résumé*

*Les Classes sont dans des espaces de noms qui sont dans des Dll (références).*

*OU*

*Les 'Dll' contiennent des 'Espaces de noms' contenant des 'Classes'.*

*Les références (Dll) permettent de charger des composants, des Classes du Framework ou d'autres classes.*

*L'instruction 'Imports' permet d'importer des espaces de noms venant de ses références.*

*Cela donne accès dans le programme à des classes. On pourra donc instancier des objets grâce à ces Classes puis utiliser des méthodes.*

Noter que dans les Classes, il existe une structure arborescente.

La première Classe (en haut) est System.

Dessous il y a entre autres System.WindowsForms.

Dessous System.Windows.Forms.Control.

Enfin System.Windows.Forms.Control.Name par exemple.

## VI-A-4-f - Héritage

Les classes héritent des membres (propriétés, méthodes) dont elles sont issues.

Exemple

Un contrôle Button hérite de System.Windows.Forms.Control et de toutes ses propriétés, car tous les composants avec représentation visuelle héritent de 'Control'.

Et bien les propriétés Name, Left, Right, Visible, Enabled qui sont des membres de Control deviennent aussi des membres de Button.

## VI-A-4-g - Membre d'instance et membre partagé

Un objet a des membres (propriétés et méthodes) :

- un membre peut être accessible directement sur une instance : on appelle cela **un membre d'instance**.
  Exemple
  Dim A As String crée une variable A, en fait une instance, un objet à partir de la Classe String.
  Ensuite on peut utiliser :

```
B=A.Trim(" ") 'ici Trim enlève les blancs en début et fin de chaine.
```

  Trim est bien une méthode qui s'applique à une instance (A) ;
- un autre membre peut être accessible par les **méthodes de la classe** (en non pas de l'instance) :
  *La Classe de sa nature, de son type.*
  Dans la classe 'String' j'utilise la méthode Compare pour comparer 2 chaines.
  c=String.Compare(a,b)
  J'utilise 2 paramètres, mais j'appelle la méthode directement à partir de la Classe String.
  *La Classe de l'opération à effectuer.*
  Dans la Classe Math j'utilise la méthode Abs (Valeur absolue) :
  c= Math.Abs(-12)

On appelle cela une **méthode partagée (shared),** car on l'utilise directement à partir du nom de la classe sans avoir à instancier.

On remarque que la Classe String a des membres d'instance et aussi des membres partagés.

La syntaxe est parfois bizarre, mais obéit donc à une certaine logique.

## VI-A-4-h - Classes statiques ou non

Ici on parle des classes et non des membres.

Certaines Classes sont dites **statiques,** car elles existent d'emblée et on peut travailler dessus sans les instancier.

Exemple

La Classe Directory (répertoire) :

```
Directory.GetCurrentDirectory    'est utilisable directement pour obtenir le répertoire courant.
```

Par contre avec une **Classe non statique** il faut instancier l'objet que l'on va utiliser.

Pour la classe DirectoryInfo (information sur le répertoire), on doit instancier avant usage un DirectoryInfo particulier :

```
Dim D As DirectoryInfo

D= New DirectoryInfo( MonDossier)
```

C'est un peu théorique, mais on verra au fur et à mesure des exemples pratiques de cela.

# VI-B - Les différentes Classes, le Framework

## VI-B-1 - Les différentes 'Classes'

Il existe trois types de Classes.

### VI-B-1-a - Les classes du Framework fournies par Microsoft avec VB

**Il existe ainsi des classes :**

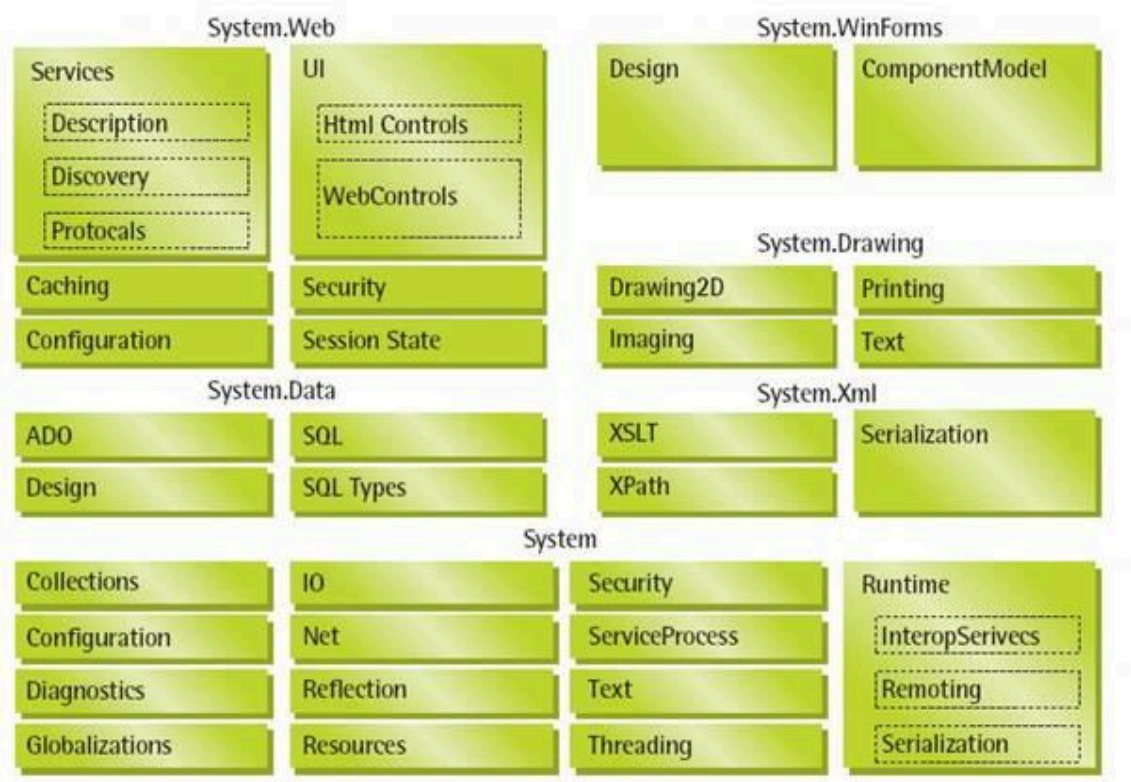
- pour les formulaires Windows (WindowsForms) ;
- pour le Web (WebForms) ;
- pour l'accès aux données ;
- pour les réseaux ;
- pour la sécurité ;
- etc.

Elles sont communes à VB, C#, C…

Le Framework est fourni avec VB, Vista, Window 7.

Il y a les Frameworks 1, 2, 3, 3.5 (correspondant à VB 2003, 2005, 2008).

Exemple du vieux Framework 1 :




Il y a aussi l'instruction **VisualBasic**.

Elle est propre au VisualBasic.

*Quand on crée un nouveau projet, les Classes le plus souvent utilisées sont automatiquement chargées dans le projet.*

## VI-B-1-b - Les classes fournies par des tiers

On peut ajouter des références (Dll) permettant d'ajouter des classes nouvelles, cela permet d'ajouter de nouvelles fonctionnalités à VB. Exemple : les pilotes de base de données, les contrôles qui n'existent pas dans VB et que vous achetez…

## VI-B-1-c - Les Classes écrites par le programmeur

Le programmeur peut les créer de toute pièce dans les modules de Classe (on verra cela plus loin).

En VB, on peut créer une classe, ses propriétés, méthodes… et l'utiliser dans le corps du programme.

Vous pouvez vous aussi créer une Classe, la compiler, puis dans un autre projet référencer la Dll de la classe que vous avez créée et l'utiliser dans le nouveau projet.

## VI-B-2 - Dans Visual Basic.Net

**Dans VB.Net il y a donc possibilité de travailler avec :**

- les **Classes du Framework**, leurs propriétés, leurs méthodes. Les Forms, Controls, les classes des variables (String, Int32…) sont disponibles par défaut (voir annexe 1) ;
- les **instructions VB.Net** de Microsoft.VisualBasic disponibles par défaut. Ce sont des instructions VB bien connues (Left, Int, IsNumeric… ) (voir annexe 2) ;
- à oublier : les instructions de **la bibliothèque de compatibilité VB6**. il faut dans ce cas importer Microsoft.VisualBasic. Compatibility et Microsoft.VisualBasic. Compatibility.Data, ces instructions sont là pour aider à la conversion, elles permettent d'utiliser des fonctions qui n'existent plus en VB.Net (comme les chaines fixes par exemple), il faut les éviter impérativement, car ce n'est pas du VB.Net et elles disparaitront probablement dans les futures versions.

Exemple

Pour la manipulation des nombres :

'Int' 'Randomize' et 'Rnd' font partie de VB.Net ;

'Round' fait partie de la classe Math donc du Framework ;

'Imp' fait partie de la bibliothèque de compatibilité VB6.

Parfois certaines fonctions font double-emploi et ont des équivalents dans les 2 ou 3 catégories.

Lesquelles utiliser ?

Les Classes sont souvent plus riches avec multiples surcharges et sont communes à tous les langages utilisant le Framework .Net. Si vous voulez passer au C#, les classes sont les mêmes.

Les instructions VB.Net sont propres à VB , c'est du VB et du .Net.

Par contre, les instructions devant de la compatibilité VB6 sont à éviter absolument. Seront-elles conservées dans les futures versions de VB.NET ?

## VI-B-3 - Lors de la création d'un nouveau projet

Sont automatiquement chargés :

une partie du Framework (System.dll) et le CLR la couche qui fait tourner le programme (**mscorlib.dll**).

Sont donc à disposition :

- quelques espaces de noms contenant des classes du Framework :

**System :** espace de noms racine pour les types de données et le type objet,

**System.data :** Classe contenant l'accès aux bases de données; ADO.Net en particulier ,

**System.drawing, System.Xml** c'est clair !!

**System.Windows :** Classes de l'interface utilisateur.

Ce dernier contient les Controls

et aussi:

**Microsoft.VisualBasic** qui contient la bibliothèque qui permet d'utiliser les instructions VB (MsgBox, IsNumeric, Chr, Asc…)

Comme ces Classes sont chargées au départ cela permet d'emblée de créer des feuilles, des contrôles…(qui sont dans les WindowsForms et les Controls). Cela permet aussi d'utiliser les instructions VB.

Si on a besoin d'autres classes, il faut les importer :

**Imports System.Math** par exemple pour utiliser les fonctions mathématiques ;

**Imports System.IO** pour les fichiers séquentiels, aléatoires… ;

**Imports System.Collections** pour les Collections… ;

**Imports System.Net** pour le réseau… ;

…

Éviter d'importer Microsoft.VisualBasic.Compatibility qui apporte la compatibilité VB6.

## VI-B-4 - Framework 1, 2, 3, 3.5, 4.

Un Framework est donc un ensemble de Classes.

Le **framework 1.0** est utilisé par VB 2003.

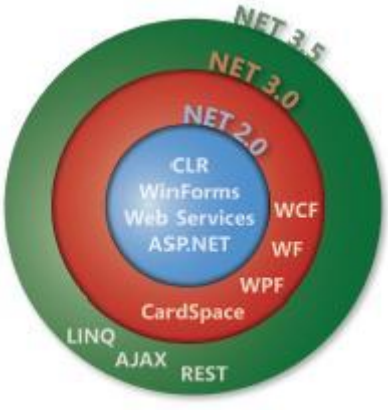
Le **framework 2.0** est utilisé par VB 2005.

Il contient des classes supplémentaires.

Le **framework 3.0** peut être utilisé par VB 2005.

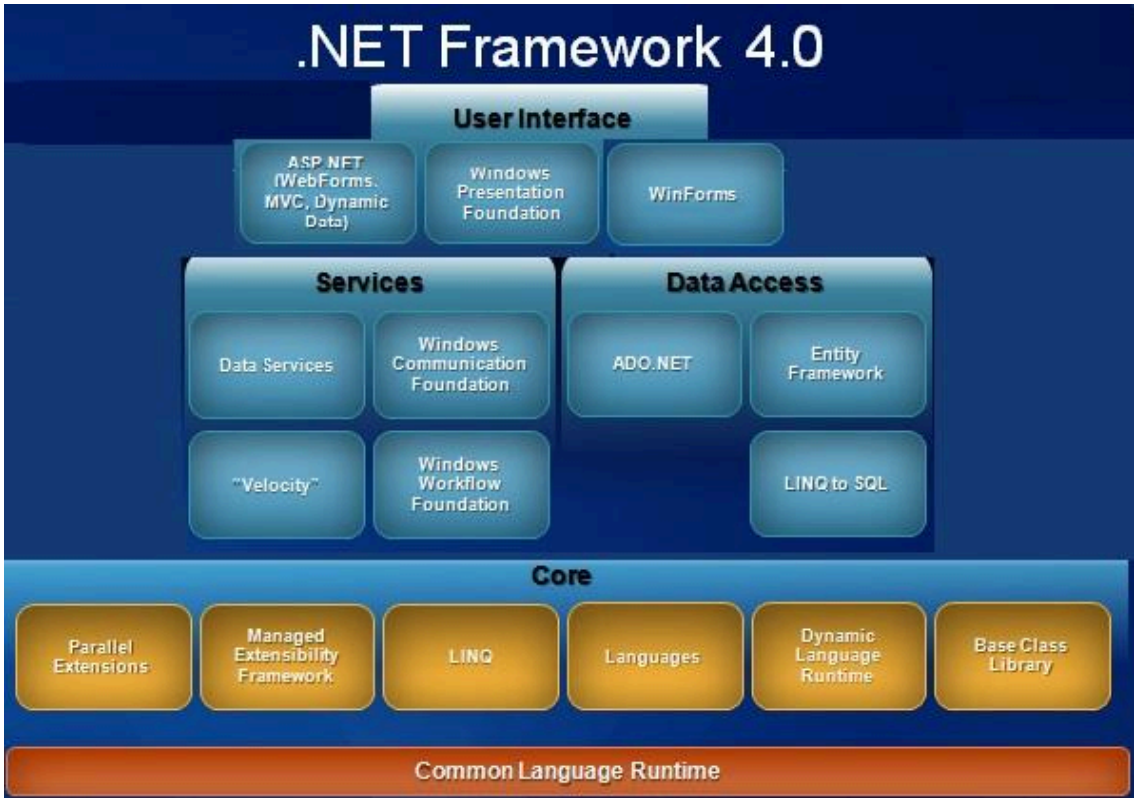
Le framework 3.0 est composé du framework 2.0 auquel s'ajoutent WCF (Windows Communication Foundation), WWF (Windows Workflow Foundation), WPF (Windows Presentation Foundation) et infocard pour l'authentification des utilisateurs.
Windows Presentation foundation permet de gérer l' affichage des fenêtres. Celui-ci est basé sur direct x (version 10).

Le **Framework 3.5** (fin 2007) utilisé par VB2008. C'est le Framework 3 auquel s'ajoute AJAX, LINQ et REST.



Versions cumulées du .NET Framework

Le **framework 4.0** peut être utilisé par VB 2010.




Sous Windows 98, XP, il faut installer le framework (avant d'utiliser l'environnement VisualBasic ou un exécutable VB).

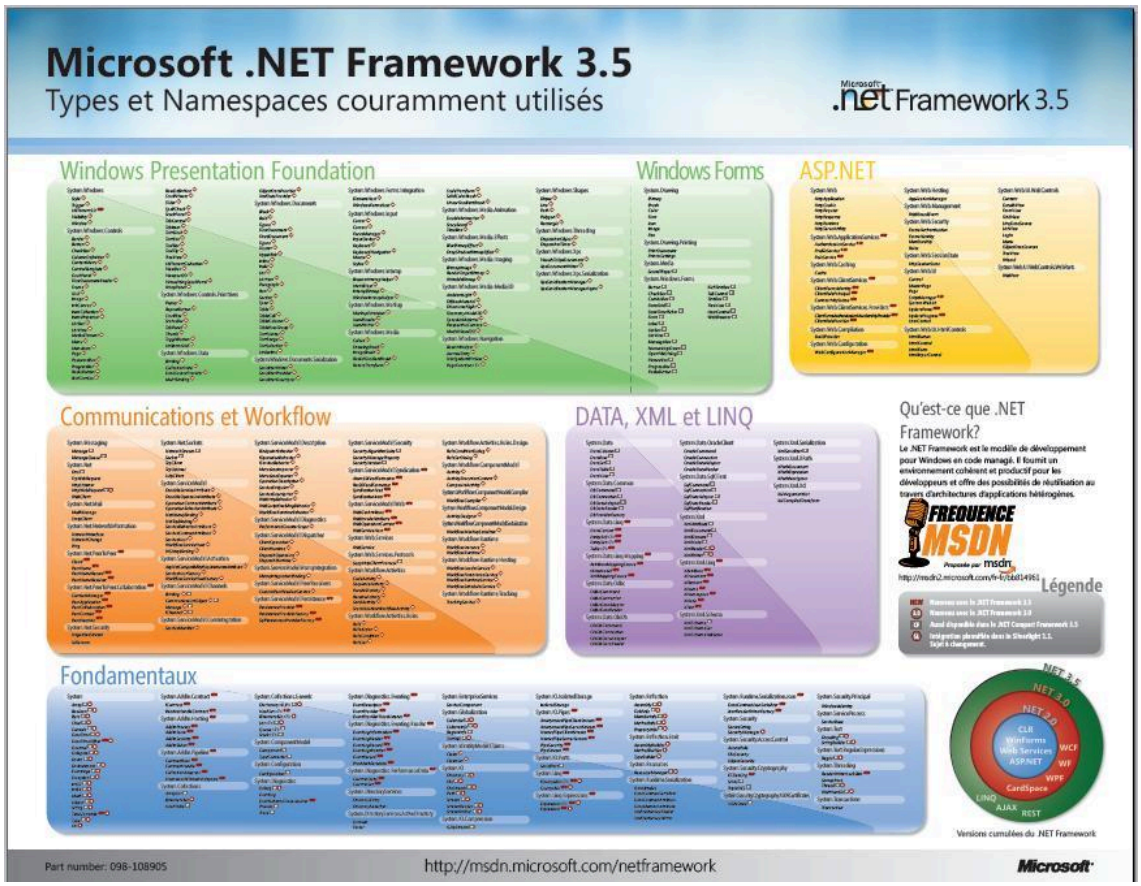
Sous Windows Vista et Windows 7 les Frameworks sont installés nativement(Framework 1, 1.1, 2, 3, 3.5 pour Windows 7).

VB 2003 utilise le Framework 1.0.

VB 2005 utilise le Framework 2.0. ou 3.0.

VB 2008 permet de choisir et d'utiliser le Framework 1.0, 2.0, 3.0, 3.5.

Voici le Framework 3.5 :




VB 2010 permet de choisir et d'utiliser le Framework 2.0, 3.0, 3.5, 4.

## VI-C - Le CLR

On rappelle que ce qui fait tourner le Framework c'est le CLR (**Common Language RunTime**), de la version 2 à la version 3.5 du Framework, c'était toujours la version 2.0.50727 du CLR qui est utilisée.
Avec le Framework 4 c'est la version 4 !! du CLR qui est utilisée.

## VI-D - Procédures événement, surcharge de méthode

### VI-D-1 - Événement et procédure événement

On a vu qu'un objet pouvait avoir un événement.

L'objet 'Button1' a un événement Button1.Click.

Comment faire pour que l'action de cliquer sur le bouton1 exécute une procédure ?

Il faut l'indiquer grâce à Handles.

Si on veut que le clic sur Button1 exécute la procédure, la Sub 'MaRoutine', il faut écrire :

```
Sub Maroutine() Handles Button1.Click

End Sub
```

Ainsi quand l'utilisateur clique sur le bouton, la procédure Maroutine est exécutée.

VB, qui est vraiment très sympa, écrit automatiquement le code pour vous.

En mode 'Design', créez un bouton, double-cliquez dessus, vous vous retrouvez dans la procédure :

```
Private Sub Button1_Click() Handles Button1.Click

End Sub
```

Par défaut il a nommé la procédure Button1_Click (mais elle aurait pu se nommer autrement).

Dans la réalité, il a ajouté 2 paramètres :

**Sender** qui contient le nom de l'objet qui a déclenché l'événement (c'est un objet) ;

**e** un objet qui contient les arguments de cet événement. (de type EventAgrs, mais pas toujours).

Cela donne :

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)  _
Handles Button1.Click

End Sub
```

Pour les forts et pour être complet, dans Form1.InitializeComponent (c'est une routine où il y a le code créant les contrôles), il y a la ligne :

**Friend WithEvents Button1 As System.Windows.Forms.Button** qui indique que le bouton a des événements (WithEvents).

En conclusion

VB écrit automatiquement les procédures événement. On verra plus loin qu'il est possible de créer nous-mêmes nos objets, événements et procédures événement.

## VI-D-2 - Différentes méthodes pour gérer les événements

### Avec WithEvents

On vient de voir cette méthode utilisée automatiquement quand on ajoute un objet visuel dans l'interface :
**WithEvents** indique qu'il faut gérer les événements.

```
Friend WithEvents Button1 As System.Windows.Forms.Button
```

Cela permet ensuite d'écrire une Sub qui est exécutée quand l'événement se produit.
Le mot **Handles** indique l'événement qui exécute la Sub.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)  _
Handles Button1.Click

End Sub
```

L'inconvénient de cette méthode est que la gestion de l'événement ne peut pas être modifiée.

### Avec AddHandler

On crée un gestionnaire d'événement.
On crée un bouton, une Sub BoutonClick, avec **AddHandler** on indique que quand l'événement Bouton.Click se déclenche, il faut sauter à l'adresse de la routine BoutonClick.

```
Dim Button1 As New Button
AddHandler Button1.Click, addressOf BoutonClick


Sub BoutonClick…
End Sub
```

Cette méthode est dynamique : dans le code, en cours d'exécution, on peut ajouter un objet, géré les événements.
On peut même annuler la gestion d'événement grâce à RemoveHandler :

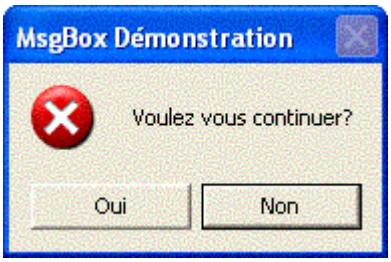
```
RemoveHandler Button1.Click, addressOf BoutonClick
```

## VI-D-3 - Surcharge de Méthode

Quand on utilise une méthode avec des paramètres, il y a parfois possibilité d'utiliser, avec la même méthode, un nombre différent de paramètres ou des paramètres de natures différentes : on dit que **la méthode est surchargée**.

Chaque manière d'écrire les paramètres s'appelle **une signature**.

Exemple :

Voici une fenêtre MessageBox : 


Pour ouvrir une fenêtre MessageBox, il y a 12 signatures, en voici 2 :

```
Reponse= MessageBox.show(TexteAAfficher,Titre, TypeBouton etIcone, BoutonParDéfaut)
```

Ici on donne 4 paramètres.

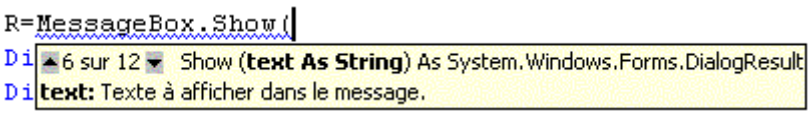
```
Reponse= MessageBox.show(TexteAAfficher)
```

Ici 1 seul paramètre.

On voit qu'on peut appeler la méthode MessageBox.Show avec un nombre différent de paramètres.

Comme on ne peut pas connaitre toutes les signatures, VB nous aide.

Si on tape R= MessageBox.show( VB affiche dans un cadre une signature, de petites flèches permettent de faire défiler toutes les autres signatures :




## VI-E - L'écriture 'Compact'

Habituellement on écrit une instruction par ligne.

Mais :

on peut créer une variable et l'initialiser en même temps.

Au lieu de :

```
Dim s As String

s = "Philippe. Jean . Toto"
```

on peut écrire :

```
Dim s As String= "Philippe. Jean . Toto"
```

Rien n'empêche d'initialiser avec une expression complexe :

```
Dim t As Single =  (3* Int(u))-2)
```

Autre exemple

Ici par exemple on découpe la string s en utilisant des séparateurs et on met les noms dans un tableau.

Version non compacte :

```
Dim s As String= "Philippe. Jean . Toto"

Dim sep() As Char={" "c, ","c, "."c}

Dim nom() As String 'Tableau des noms

nom = s.Split ( sep, 100, StringSplitOptions.RemoveEmptyEntries )
```

Mais on peut compacter le code :

```
Dim nom() As String = "Philippe. Jean . Toto".Split(New Char() {" "c, ","c, "."c, ":"c}, 100, _
StringSplitOptions.RemoveEmptyEntries)
```

On crée la variable nom, on l'initialise en même temps avec une méthode, les paramètres étant directement créés au niveau de l'appel de la méthode.

Même chose avec des fonctions.

Le paramètre d'une fonction peut être le résultat d'une fonction.

Soit une fonction qui se nomme 'Carré', on peut lui envoyer un paramètre (elle accepte un Single), elle retourne le carré du paramètre :

```
Function Carré ( v as Single) as Single

    Return v*v

End Function
```

Soit une fonction qui se nomme 'Inverse', on peut lui envoyer un paramètre (elle accepte un Single), elle retourne le paramètre après avoir inversé le signe :

```
Function Inverse ( v as Single) as Single

    Return -v

End Function
```

Si on veut prendre le carré de 2 puis l'inverse et enfin la partie entière du résultat :

```
Dim resultat As Single

Dim resultatintermediaire1 As Single

Dim resultatintermediaire2 As Single

resultatintermediaire1= Carré(2)

resultatintermediaire2 = Inverse (resultatintermediaire1)

resultat= Int(resultatintermediaire2)
```

ou de manière compacte :

```
Dim resultat As Single = Int(Inverse(Carré(2)))
```



*L'écriture compacte du code est plus ramassée, moins 'lourde', avec moins de variables intermédiaires, mais **on perd en lisibilité** parfois.*

```
Dim resultat As Single = Int(Inverse(Carré(2))) 'Cet exemple est compact, mais lisible.

Dim nom() As String = S.Split(New Char() {" "c, ","c, "."c, ":"c}, 100, _
StringSplitOptions.RemoveEmptyEntries)
'C'est moins lisible
```

## VI-F - Notion de flux ou 'Stream'

Le Stream (flux, torrent, courant) est une notion générale, c'est un flux de données provenant ou allant vers un fichier, un port, une connexion TCP/IP…

Ici on utilise un Stream pour lire ou écrire dans un fichier.

L'accès est séquentiel : les données sont traitées du début à la fin du fichier.

Pour écrire dans un fichier texte.

Il faut instancier un objet de la classe StreamWriter. On écrit avec Write ou WriteLine (ajoute un saut de ligne). Enfin on ferme avec Close.

On peut instancier avec le constructeur de la classe StreamWriter et avec New, ou par la Classe File.

```
Dim SW As New StreamWriter ("MonFichier.txt") ' crée ou si existe écrase
```

Il existe une surcharge permettant de ne pas écraser, mais d'ajouter à la fin du fichier :

```
Dim SW As New StreamWriter ("MonFichier.txt", True) ' crée ou si existe ajoute
```

Avec la classe File :

```
Dim SW As  StreamWriter=File.CreateText ("MonFichier.txt") ' crée ou si existe écrase

Dim SW As StreamWriter = File.AppendText("MonFichier.txt") ' crée ou si existe ajoute
```

Ensuite pour écrire 2 lignes :

```
SW.WriteLine ("Bonjour")

SW.WriteLine ("Monsieur")
```

Enfin on ferme :

```
SW.Close()
```

Pour lire dans un fichier Texte

Il faut instancier un objet de la classe StreamReader. On lit avec Read (un nombre d'octets) ReadLine (une ligne) ReadToEnd (de la position courante jusqu'à la fin). Enfin on ferme avec Close.

Avec le constructeur de la Classe Stream Reader :

```
Dim SR As New StreamReader ("MonFichier.txt")
```

Avec la Classe File :

```
Dim SR As  StreamReader=File.OpenText ("MonFichier.txt") '
```

Comment lire chaque ligne du fichier et s'arrêter à la fin ?

En effet on ne sait pas habituellement combien le fichier contient de lignes, si le fichier contient 2 lignes il faut en lire 2 et s'arrêter sinon on tente de lire après la fin du fichier et cela déclenche une erreur.

**Trois solutions :**

1 Utiliser ReadToEnd qui lit en bloc jusqu'à la fin ;
2 Avant ReadLine mettre un Try : quand l'erreur 'fin de fichier' survient elle est interceptée par Catch qui sort du cycle de lecture et ferme le fichier ;
3 Utiliser Peek qui lit dans le fichier un caractère, mais sans modifier la position courante de lecture. La particularité de Peek est de retourner -1 s'il n'y a plus de caractère à lire sans déclencher d'erreur, d'exception.

La troisième solution est la plus générale et la plus élégante :

```
Do Until SR.Peek=-1

     Ligne=SR.ReadLine()

Loop
```

Enfin on ferme :

```
SR.Close()
```

# VII - Exemples de code, exercices

On prendra des exemples de routine très simples ne contenant que du code.

## VII-A - Petites routines sur les chaines de caractères

### VII-A-1 - Une string 'Nom' contient un nom, mettre si nécessaire la première lettre en majuscule

1 En utilisant les Classes String et Char :

```
Dim Nom As String = "philippe"

' Si le premier caractère est minuscule

If Char.IsLower(Nom.Chars(0)) Then

' Le transformer en majuscule et afficher

MessageBox.Show(Char.ToUpper(Nom.Chars(0)) + Nom.Substring(1, Nom.Length - 1))

End If
```

On regarde si le premier caractère de la chaine Nom.Chars(0) est minuscule (.IsLower).

Si oui on transforme ce premier caractère en majuscule (.ToUpper) et on ajoute la sous-chaine allant du second au dernier caractère.

2 En utilisant les instructions VB :

```
Dim Nom As String = "philippe"

Nom = UCase(Microsoft.VisualBasic.Left(Nom, 1)) & Mid(Nom, 2)
```

On prend le premier caractère de gauche : Left(Nom,1), on le transforme en majuscule (Ucase), on ajoute la chaine commençant par le second caractère et allant jusqu'à la fin.

### VII-A-2 - Comment voir si un caractère est une voyelle

```
Dim Voy As String = "aeiouy"

Dim C As String = "p"

If Instr( Voy, C)<>0 then

…

End If
```

Ici on regarde si la String C est contenue dans la String Voy (qui contient toutes les voyelles), si cela retourne 0 c'est que la String C n'y est pas.

### VII-A-3 - Comment éliminer une combinaison bien précise de caractères en début de chaine

Exemple : éliminer une balise html de type </b …/b0> et son contenu dans une chaine nommée Ch :

```
' Le premier caractère est-il '<', recherche avec StartsWith
'(en plus on élimine les espaces avec Trim())
```

```
If Ch.Trim().StartsWith("<") Then

' Rechercher le caractère '>', le premier
Dim lastLocation As Integer = Ch.IndexOf(">")

If lastLocation >= 0 Then
' éliminer la chaine entre '<' et '>'
    Ch = Ch.Substring((lastLocation + 1))
End If
End If
```

## VII-A-4 - Vous avez une chaine de caractères : comment afficher le premier caractère, puis les 2 premiers, puis 3… ?

Dans un formulaire (une fenêtre), il y a un TextBox1( zone de texte avec sa propriété Multiline=True)

```
Dim C As String = "DUBONET"

Dim Tx As String

Dim i As Integer

'Avec VisualBasic
For i = 1 To Microsoft.VisualBasic.Len(C)

    Tx += Microsoft.VisualBasic.Left(C, i) + ControlChars.CrLf

Next i

TextBox1.Text = Tx


'Avec le Framework c'est mieux
For i = 1 To C.Length

    Tx += C.Substring(0, i) + ControlChars.CrLf


Next i

TextBox1.Text = Tx
```

Mettre ce code dans Form_Load puis lancer le programme.

Affiche :

```
D

DU

DUB

DUBO

DUBON

DUBONE

DUBONET
```

On remarque : Tx est une string permettant de stocker temporairement la string à afficher. À chaque boucle on ajoute la nouvelle string (Tx += est équivalent à Tx=Tx+…) et un caractère de retour à la ligne.

Left fait partie de l'espace de noms Microsoft.VisualBasic.

## VII-A-5 - Vous avez deux chaines de caractères : comment savoir si la seconde est une anagramme de la première ?

Pour les nuls, une anagramme c'est les mêmes lettres dans un ordre différent.

Il faut mettre les 2 chaines dans un tableau de caractères, trier les 2 tableaux, les remettre dans des strings et les comparer.

```
'créons 2 string

Dim maString1 As String = "stressed"

Dim maString2 As String = "desserts"


'On passe les strings dans des tableaux

Dim myChar1 As Char =mastring1.ToCharArray

Dim myChar2 As Char =mastring2.ToCharArray


'On trie les tableaux

Array.Sort( myChar1)

Array.Sort( myChar2)

'On passe les caractères dans des strings

Dim MyStringTrie1 As New String (myChar1)

Dim MyStringTrie2 As New String (myChar2)

'On compare les 2 Strings, si elles sont égales cela retourne 0, l'expression 0=0 est True,
'on la mettre dans un Boolean

Dim anagramme As boolean =(String.Compare (MyStringTrie1 ,MyStringTrie2 )=0)
```

## VII-A-6 - Compter combien de fois un mot apparait dans un texte

Calculer le nombre d'occurrences (compteur) d'une sous-chaine (monMot) dans une String (monTexte).

On rappelle que .IndexOf permet de chercher une sous-chaine dans une chaine (il retourne 0 si la chaine n'est pas présente ou la position de la chaine).

```
Dim monTexte As String ="jfkjf……"

Dim monMot As String= "lulu"


Dim compteur As Integer =-1

Dim index As Integer =-1


Do

  compteur+= 1

  index= monTexte.IndexOf (monMot, index +1)
```

```
Loop Until index <0
```

On initialise le compteur à -1, car la boucle Do Loop est systématiquement effectuée une fois et incrémente le compteur une fois de trop.

On initialise l'index à -1, car dans la boucle Do Loop on utilise Index+1, sinon, si la chaine cherchée débute au 1er caractère elle n'est pas comptée.

## VII-B - Petits programmes de mathématiques

**On prendra des exemples de routines mathématiques simples :**

- calcul de l'hypoténuse d'un triangle rectangle ;
- somme de N entiers ;
- afficher les tables de multiplication ;
- valeur maximum d'un tableau ;
- calcul de factorielle (avec ou sans récursivité) ;
- un nombre est-il premier ? ;
- décomposition en nombres premiers ;
- diviseurs d'un nombre.

### VII-B-1 - Calcul de l'hypoténuse d'un triangle rectangle

On crée pour cela une fonction, on envoie 2 paramètres de type Single : les 2 côtés du triangle, la fonction retourne l'hypoténuse.

```
Function Hypotenuse (ByVal Side1 As Single, ByVal Side2 As Single) As Single
    Return Sqrt((Side1 ^ 2) + (Side2 ^ 2))
End Function
```

Pour les nuls, on rappelle que le carré de l'hypoténuse est égal à la somme des carrés des 2 autres côtés.

### VII-B-2 - Somme de N entiers

Calculer par exemple pour Nombre=20 la Somme=1+2+3+4…+18+19+20

```
Dim Somme As Integer     'Variable somme

Dim Nombre As Integer=20

Dim i As Integer         'Variable de boucle



For i=0 To Nombre

    Somme += i

Next i
```

On rappelle que Somme += i est équivalent à Somme =Somme + i

Pour afficher le résultat, s'il existe une TextBox :

```
TextBox1.Text = Cstr(Somme)     'Somme est transformé en String puis affecté à la propriété Text
 du TextBox
```

## VII-B-3 - Afficher les tables de multiplication

On fait 2 boucles :

- celle avec i (qui décide de la table: table des 1, des 2…).

On affiche 'table des' puis valeur de i ;

- celle avec j (allant de 1 à 10 pour chaque table)

Pour chaque ligne, on affiche la valeur de i puis ' X ' puis la valeur de j puis ' = ' puis la valeur de i fois j.

ControlChars.CrlF permet un saut à la ligne .

À chaque fois que l'on a quelque chose à afficher, on l'ajoute à la variable String T.

À la fin on affecte T à la propriété Text d'un TextBox pour rendre visible les tables.

```
Dim i As Integer

Dim j As Integer

Dim T As String


For i = 1 To 10

    T += ControlChars.CrLf

    T += "Table des " & i & ControlChars.CrLf

    For j = 1 To 10

        T += i.ToString & " X " & j.ToString & "=" & i * j & ControlChars.CrLf

    Next j

Next i

TextBox1.Text = T
```

Affiche :

```
Table des 1

1 X 1 =1

1 X 2 =2

….
```

## VII-B-4 - Trouver la valeur la plus élevée d'un tableau d'entiers

Pour cela on crée une Fonction, on l'appelle en donnant en paramètre le tableau d'entiers, la fonction retourne l'entier le plus grand.

```
Function FindMax(ByVal a() As Integer) As Integer

Dim fin As Integer = UBound(a)
```

```
Dim valeurMax As Integer = a(0)

Dim i As Integer

For i = 0 To fin

If a(i) > valeurMax Then valeurMax = a(i)

Next i

Return valeurMax

End Function
```

Une boucle compare chaque valeur du tableau a() avec valeurMax, si l'élément du tableau est plus grand que valeurMax, valeurMax prend la valeur de l'élément.

## VII-B-5 - Factorielle

On rappelle que N! (factorielle N)= 1*2*3*…*(N-2)*(N-1)*N

Exemple Factorielle 3 =1*2*3

```
Dim R As Long

R=Factorielle(3)     'retournera 6
```

Cette fonction n'est pas fournie par VB, créons une fonction 'Factorielle' :

```
Function Factorielle (ByVal N as Long) As Long

    Dim i As Long

    Resultat=1

    For i= 1 to N

        Resultat=i* Resultat

    Next i

    Return Resultat

end Function
```

Cela crée une fonction recevant le paramètre N et retournant un long.

Une boucle effectue bien 1*2*3…*N-1*N.

## VII-B-6 - Factorielle avec 'récursivité'

Une autre manière de calculer une factorielle est d'utiliser la récursivité.

Une procédure est récursive si elle peut s'appeler elle-même.

VB gère la récursivité.

Comment faire pour les factorielles ?

On sait que Factorielle N= N * Factorielle(N-1)

N!= N*(N-1)! : en sachant que 1!=1.

Créons la fonction :

```
Function Factorielle (ByVal N as Long) As Long

    If N=1 then

        Return 1

    Else

        Return N* Factorielle(N-1)

    End If

end Function
```

Dans la fonction factorielle, on appelle la fonction Factorielle, c'est bien récursif.

Pour N=4, la fonction Factorielle est appelée 4 fois : Factorielle (4) puis Factorielle(3) puis Factorielle(2) puis Factorielle (1)

```
Factorielle (1) retourne 1

Factorielle (2)retourne  2    '2*factorielle(1)

Factorielle (3)retourne  6    '3*factorielle(2)

Factorielle (4) retourne 24   '4*factorielle(3)
```

Vb gère cela avec une pile des appels. Il met dans une pile les uns au-dessus des autres les appels, quand il remonte, il dépile de haut en bas (dernier rentré, premier sorti).



*Attention : la pile a une taille maximum, si N est trop grand, on déclenche une erreur de type StackOverflow.*

## VII-B-7 - Un nombre est-il premier ?

Un nombre premier est seulement divisible par 1 et lui-même.

Pour voir si N est premier on regarde successivement si ce nombre est divisible par 2 puis 3 puis 4… jusqu'a N-1

Un nombre est divisible par un autre si la division donne un entier.

N. B. Comment voir si le nombre A est entier ?

Pour ma part, j'utilise la méthode suivante : À est entier si A=Int(A).

Une autre méthode pour voir si N est divisible par B : il est divisible si N Mod(B)=0.

Avec une boucle For Next :

```
Dim IsPremier As Boolean

Dim j, k As Long
```

```
IsPremier = True
j = 2
For k = 2 To N
     If (N Mod k = 0) And (k <> N) Then
           IsPremier = False
           j = n
     End If
     j = j + 1
Next
```

En sortie de boucle si IsPremier= true , le nombre N est premier.

Avec un Do Loop :

```
Dim IsPremier As Boolean
Dim N As Double=59 'nombre à étudier
Dim I As Double
I=2:  IsPremier=True
Do
    If N/I= Int(N/I) then
        IsPremier=False
    Else
        i += 1
    End if
Loop While IsPremier=True And I<N
```

Pour 59 IsPremier sera égal à True.

On peut améliorer la routine en remarquant :

si un nombre n'est pas premier il admet 2 diviseurs dont un est inférieur à racine N.

**On peut donc :**

- vérifier que le nombre n'est pas pair puis ;
- vérifier s'il est divisible par les nombres allant de 3…jusqu'à racine de N en ne tenant compte que des nombres impairs.

Remarque pour ceux qui veulent tester le code

Pour utiliser la routine sur les nombres premiers, il faut créer une petite interface : dans un formulaire créer un bouton nommé 'Button1' et une TextBox nommée 'TextBox1', enfin mettre dans la routine Button1_Click le code ci-dessous.

Quand on lance le programme, on saisit un nombre dans le textbox, puis on clique sur le bouton, cela affiche True ou False dans une MessageBox, si le nombre est premier ou non :

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles Button1.Click

Dim IsPremier As Boolean

Dim N As Double

Dim I As Double

N = Val(TextBox1.Text)

I = 2 : IsPremier = True

Do

If N / I = Int(N / I) Then

IsPremier = False

Else

I += 1

End If

Loop While IsPremier = True And I < N

MsgBox(IsPremier.ToString)

End Sub
```

## VII-B-8 - Décomposition en nombres premiers

```
Dim nbr As Integer = 1000 'nombre à étudier

Dim div As Integer = 2


Dim t As String = ""

While div <= nbr

While nbr / div = Int(nbr / div)

t = t & div & " x "

nbr = nbr / div

End While

div = div + 1

End While

TextBox1.Text = t
```

## VII-B-9 - Diviseurs d'un nombre

C'est une fonction qui retourne une collection de type List contenant tous les diviseurs d'un nombre nommé Number.

Si Number Mod i est égal à zéro c'est que i est un diviseur de Number, on l'ajoute à la liste.

```
Public Function Diviseur (ByVal Number As Integer) As List(Of Integer)

Dim Diviseur As New List(Of Integer)

For i As Integer = 1 To Number

      If Number Mod i = 0 Then Diviseur.Add(i)

Next

End Function
```

# VII-C - Travail sur les tableaux et collections (tri, recherche, insertion, effacement d'éléments)

## VII-C-1 - Utiliser les tableaux

Le travail sur des tableaux est très intéressant. On étudiera plusieurs routines en comparant la méthode classique et les fonctions du Framework. Puis on utilisera les collections pour faire la même chose.

### VII-C-1-a - Trier un tableau

Routine de Tri

Pour trier un tableau de chaines de caractères, il faut comparer 2 chaines contiguës, si la première est supérieure (c'est-à-dire après l'autre sur le plan alphabétique) on inverse les 2 chaines, sinon on n'inverse pas. Puis on recommence sur 2 autres chaines en balayant le tableau jusqu'à ce qu'il soit trié.

Tout l'art des routines de tri est de faire le moins de comparaisons possible pour trier le plus vite possible.

Voyons une des routines les plus rapides, le Bubble Sort (ou tri à bulle). On le nomme ainsi, car l'élément le plus petit monte progressivement au fur et à mesure jusqu'au début du tableau comme une bulle. (Parfois c'est l'élément le plus grand qui descend !!)

Une boucle interne balaye le tableau et compare 2 éléments contigus et les inverse si nécessaire. Une boucle externe fait tourner la boucle interne N fois.

La boucle interne fait descendre l'élément le plus grand vers la fin du tableau, la boucle externe répète l'opération N fois permettant le tri de tous les éléments.

```
Private Sub Button1_Click

'Création du tableau et de son contenu non trié.

Dim T(5) As String 'élément de 0 à 4

T(0) = "toto"

T(1) = "tata"

T(2) = "tutu"

T(3) = "lolo"

T(4) = "lulu"


'Création des variables

Dim N As Integer = 5  'Nombre d'éléments
```

```
Dim i, j As Integer
Dim Temp As String    'Variable temporaire

'Routine de tri
For i = 0 To N - 1    'Boucle externe
For j = 0 To N - 1   'Boucle interne
If T(j) > T(j + 1) Then
  Temp = T(j) : T(j) = T(j + 1) : T(j + 1) = Temp  'Inverser si pas dans le bon ordre
End If
Next j
Next i

' Pour afficher le tableau trié dans un textbox
Dim tt As String= ""
For i = 0 To N
tt = tt + T(i) + ControlChars.CrLf
Next i
TextBox1.Text = tt
End Sub
```

Remarque : pour inverser le contenu de 2 variables, on doit écrire :

Temp=T(j): T(j)=T(j+1):T(j+1)=Temp (L'instruction qui faisait cela en VB6 et qui se nommait Swap n'existe plus.)

Cette routine trie bien le tableau, mais n'est pas optimisée : il n'est pas nécessaire que la boucle interne tourne de 0 à N-1 à chaque fois, car après une boucle, le dernier élément, le plus grand, est à sa place. Pour i=0 la boucle interne tourne jusqu'à N-1, pour i=1 jusqu'à N-2…

Cela donne :

```
For i=0 To N-1
    For j=0 To N-i-1
        If T(j)>T(j+1) then
            Temp=T(j): T(j)=T(j+1):T(j+1)=Temp
        End if
    Next j
 Next i
```

Il existe d'autres méthodes encore plus rapides (Méthode de Shell et Shell-Metzner), il existe aussi le QuickSort très performant pour les tableaux non triés, voir chapitre sur la récursivité.

Mais il y a plus simple : tri avec la méthode SORT du Framework.

Pour un tableau unidimensionnel.

```
Dim Animals(2) As String
Animals(0) = "lion"
Animals(1) = "girafe"
Animals(2) = "loup"
Array.Sort(Animals)
```

Et le tableau est trié !!

On rappelle que l'on ne peut pas trier un tableau multidimensionnel, mais il y a des ruses (voir rubrique : tableau).

Les Collections peuvent être triées automatiquement aussi.

## VII-C-1-b - Rechercher un élément dans un tableau

Routine de recherche

On a un tableau de string, on veut chercher où se trouve (en quelle position) une string.

Pour une liste non triée, on n'a pas d'autres choix que de comparer la string cherchée à chaque élément du tableau, on utilisera donc une boucle :

```
N=4  'tableau de 5 éléments.

Dim T(N) As String  'élément de 0 à 4

T(0)="vert"

T(1)="bleu"

T(2)="rouge"

T(3)="jaune"

T(4)="blanc"

Dim i  As Integer    'Variable de boucle

Dim AChercher As String= "rouge" 'String à chercher

For i=0 To N

      If T(i)=AChercher then

            Exit For

      End if

Next i

'i contient 2
```

Pour une liste triée (suite ordonnée), on peut utiliser la méthode de recherche dichotomique. On compare l'élément recherché à l'élément du milieu du tableau, cela permet de savoir dans quelle moitié se situe l'élément recherché. De nouveau on compare à l'élément recherché à l'élément du milieu de la bonne moitié…jusqu'à trouver. Pour cela on utilise les variables Inf et Sup qui sont les bornes inférieure et supérieure de la zone de recherche et la variable Milieu. On compare l'élément recherché à l'élément du tableau d'indice milieu, s'ils sont égaux on a trouvé, on sort, s'ils sont différents on modifie Inf et Sup pour pointer la bonne plage puis on donne à Milieu la valeur du milieu de la nouvelle plage et on recommence.

```
Dim N As Integer

Dim T(N) As String  'élément de 0 à 4

Dim Inf, Sup, Milieu As Integer '

Dim Reponse As Integer 'contient le numéro de l'élément

                       'ou -1 si élément non trouvé

Dim i  As Integer    'Variable de boucle

Dim AChercher As String= "c" 'String à chercher


N=4  'tableau de 5 éléments.

T(0)="a"

T(1)="b"

T(2)="c"

T(3)="d"

T(4)="e"

Inf=0: Sup=N

Do

    if inf>Sup then Reponse=-1: Exit Do

    Milieu= INT((Inf+Sup)/2)

    If Achercher=T(Milieu) then Reponse=Milieu:Exit Do

    If Achercher<T(Milieu) then Sup=Milieu-1

    If Achercher>T(Milieu) then Inf=Milieu+1

Loop

'Reponse =2
```

La recherche dichotomique est rapide, car il y a moins de comparaisons.

Recherche avec les facilités du Framework.

Mais comme d'habitude VB.Net possède des propriétés permettant de rechercher dans un tableau trié ou non, et cela, sans avoir à écrire de routine.

Binarysearch recherche un élément dans un tableau trié unidimensionnel (algorithme de comparaison binaire performant sur tableau trié : probablement une recherche dichotomique).

Exemple :

```
I=Array.BinarySearch(Mois, "Février")

IndexOf
```

Recherche un objet spécifié dans un tableau unidimensionnel (trié ou non), retourne l'index de la première occurrence.

```
Dim myIndex As Integer = Array.IndexOf(myArray, myString)
```

Retourne -1 si l'élément n'est pas trouvé.

LastIndexOf fait une recherche à partir de la fin.

## VII-C-1-c - Effacer, insérer un élément dans un tableau

### 1- Éliminer un élément

Avec une routine

Soit un tableau de 4 String T(0), T(1), T(2),T(3).

| Toto |
|---|
| Tata |
| Tonton |
| Titi |

On veut éliminer l'élément "Tata", le second élément, il faut passer le troisième dans le second, le quatrième dans le troisième… et effacer le dernier élément du tableau.

```
Dim N As Integer

Dim T(N) As String  'création d'un tableau de String.

Dim i As Integer

For i= 1 To N-1

    T(i)=T(i+1)

Next i

T(N-1)=""           'ne pas oublier de modifier le dernier élément
```

On obtient :

| Toto |
|---|
| Tonton |
| Titi |

On remarque que la boucle démarre au niveau de l'élément à enlever et pas à la fin du tableau.

Avec les facilités du Framework

On peut décaler les éléments d'un tableau avec la méthode Copy de la Classe Array.

Il suffit de copier le tableau sur le même tableau, mais décalé d'un élément.

```
Array.Copy(T, 2, T, 1, T.Length - 2)

T(T.Length-1)=""
```

On utilise ici une surcharge de Copy :

```
Array.Copy(Tor, indexOrigine, td, indexDestitation, nombredElements)
```

Tor : tableau d'origine, Td : tableau destination.

### 2- Insérer un élément

Avec une routine

Soit un tableau de 4 String T(0), T(1), T(2),T(3).

| Toto |
|---|
| Tonton |
| Titi |

On veut insérer l'élément "Tata" en seconde position, après "Toto"; il faut d'abord décaler les éléments vers la fin du tableau. Attention : il faut le faire en commençant par la fin du tableau : il faut passer l'avant-dernier élément en dernière position puis l'avant-avant-dernier en avant-dernière position.

```
Dim N As Integer

Dim T(N) As String  'création d'un tableau de String.

Dim i As Integer

For i= N-1 To 1 Step -1

    T(i)=T(i-1)

Next i

T(1)="Tata"           'ne pas oublier d'ajouter
```

On obtient :

| Toto |
|---|
| Tata |
| Tonton |
| Titi |

On remarque que la boucle doit commencer à la fin du tableau et remonter.

Avec les facilités du Framework

On peut déplacer les éléments avec la méthode Copy de Array.

On copie le tableau sur le même tableau, mais décalé d'un élément.

```
Array.Copy(T, 1, T, 2, T.Length - 2)

T(1)="Tata"
```

On utilise ici une surcharge de Copy :

```
Array.Copy(Tor, indexOrigine, td, indexDestitation, nombredElements)
```

Tor: tableau d'origine; Td : tableau destination.

## VII-C-2 - Utiliser les Collections

Relire le chapitre sur les collections.

On rappelle que nombre d'éléments dans une collection n'est pas défini au départ comme dans un tableau. Dans une collection il n'y a que les éléments que l'on a ajoutés.

Les éléments sont repérés grâce à un index, mais attention, si vous ajoutez un élément, cela décale l'index des éléments qui suivent.

```
Dim L As New ArrayList()      'On crée une collection ArrayList

L.Add("toto")              'On ajoute un élément à la collection

MsgBox(L(0))                  'On affiche le premier élément
```

On pourra aussi écrire L.Item(0) pour pointer le premier élément.

```
      MsgBox(L.Count.ToString)      'On affiche le nombre d'éléments.
```



*Attention c'est le nombre d'éléments. S'il y a 3 éléments dans la ArrayList ce sont les éléments d'index 0,1,2.*

## VII-C-2-a - Trier une collection

```
L.Sort()                      'Trie la collection
```

## VII-C-2-b - Rechercher un élément dans une collection

```
L.Containts (élément)         ' Retourne True si la liste contient élément.
```

Recherche d'un élément dans une collection NON TRIÉE avec IndexOf :

```
Dim l As New ArrayList

Dim i As Integer

l.Add("toto")

l.Add("lulu")

i = l.IndexOf("lulu")

MsgBox(i.ToString)     'Affiche 1
```

On rappelle qu'il existe aussi LastIndexOf qui démarre par la fin et une surcharge permettant de débuter la recherche à partir d'un indice donné.

Recherche d'un élément dans une collection TRIÉE avec BinarySearch :

```
Dim l As New ArrayList

Dim i As Integer

l.Add("toto")
```

```
l.Add("lulu")

l.Sort()'Il est nécessaire que le tableau soit trié

i = l.BinarySearch("lulu")

MsgBox(i.ToString)
```

## VII-C-2-c - Effacer, insérer un élément dans une collection

```
   L.Remove("toto")          'On enlève un élément de la liste

      L.RemoveAt(0)                'On enlève l'élément 0 de la liste
```

Insert permet d'insérer à un index spécifié.

```
      L.Insert( position, Ainserrer)
```

## VII-C-3 - Différences tableau/collection

Un tableau peut avoir plusieurs dimensions, cela permet plusieurs indices.

Soit un tableau T(X,2) de String permettant de stocker en mémoire des patients (Nom, prénom, adresse…).

Nom Prénom Adresse

| | | |
|---|---|---|
| Dupont | Pierre | 32 rue du… |
| Dubout | Jean | 12 Place… |
| … | | |

…

Le premier indice est l'indice patient (ligne), le second indice indique la colonne.

Ainsi le prénom du patient 1 est T(1,1)

Un tableau peut avoir X dimensions, tous les éléments sont du même type.

```
Dim t(45,45,45) As Strings
```

On peut définir une structure

```
Public Structure Adresse

   Dim Numero     As Integer

   Dim Rue        As String

   Dim Ville      As String

End Structure
```

Il est ensuite possible de travailler sur un tableau de variable 'structure'.

```
Dim Adresses(99) as Adresse    'Permet de travailler sur un tableau de 100 adresses

Adresses(33).Rue="Place de la mairie"
```

Un tableau a un nombre défini d'éléments quand on le déclare, plus les dimensions sont grandes, plus il occupe de place, même si certains de ses éléments sont vides. Une collection ne contient que les éléments qu'on a mis dedans.

Une collection n'a qu'une 'dimension' : pour chaque indice on n'a qu'un seul élément (un seul Objet). On peut créer une collection de String ou d'Integer.

On verra plus loin qu'une collection comme ArrayList est une collection d'objets. (Collection d'objet 'Patient' par exemple, chaque objet 'Patient' ayant un nom, un prénom…)

Le travail sur les tableaux est beaucoup plus rapide que sur les collections.

Par exemple , pour stocker 1000 string dans un tableau ou une collection, l'usage du tableau est 60 % plus rapide.

En programmation procédurale, on utilise les tableaux par habitude, en programmation objet on utilise bien les Collections.

## VII-C-4 - Utilisation particulière des tableaux

Utiliser un tableau plutôt qu'une lourde routine permet parfois de résoudre un problème de programmation.

Le principe est : plutôt que d'utiliser une multitude de tests pour trouver une valeur, il est parfois préférable de mettre judicieusement les valeurs dans un tableau et ensuite de lire la bonne 'case'.

1 - Exemple 1

J'ai besoin de savoir le nom du jour de la semaine en fonction de son numéro.

1 doit retourner "Lundi"
2 doit retourner "Mardi"
3 doit retourner "Mercredi"
…

Avec If Then

Ce n'est pas bien élégant, en plus il faut appeler la routine à chaque fois qu'on veut le nom.

```
Dim J As Integer  ' Contient le numéro du jour

Dim Nom As String

If J= 1 Then

        Nom="Lundi"

ElseIf 2 Then

        Nom="Mardi"

….

End If
```

Avec Select Case

On peut faire un Select Case, c'est pas bien élégant, en plus il faut appeler la routine à chaque fois qu'on veut le nom.

```
SelectCase J
```

```
    Case 1

        Nom="Lundi"

    Case 2

        Nom="Mardi"

    …

End Select
```

Avec un tableau

C'est plus élégant de créer un tableau.

```
Dim Nom as String ={"Lundi", "Mardi", "Mercredi"……"Dimanche"}

Lenom= Nom (J-1)
```

C'est plus commode et le plus compact.

Notez que le premier élément du tableau étant l'élément 0 , il faut utiliser J-1.

Avec le Framework

On utilise pour cela une énumération :

```
Enum NomJour
    Lundi
    Mardi
    Mercredi
    Jeudi
    Vendredi
    Samedi
    Dimanche
End Enum
```

puis

```
Dim s As Type = GetType(NomJour) 'on instance s

Nom = [Enum].GetName(s, J)     'Paramètre: une instance de l'énumération et J; retourne le nom.
```

2 - Exemple 2

J'ai besoin de connaitre le risque de faire un accident vasculaire en fonction du sexe et de la tranche d'âge.

10 à 20 ans , Homme doit retourner 0
…
60 à 70 ans et Femme doit retourner 20
…

Avec If Then

C'est pas bien élégant, en plus il faut appeler la routine à chaque fois.

```
Dim S As Integer  ' Contient le sexe 0 masculin, 1 féminin
```

```
Dim A As Integer  ' Contient la tranche d'âge
Dim Risque As Integer
If S= 1 Then
     If A=1 Then
        Risque=0
     ElseIf A=2 Then
        Risque=3
    …
    End If
Else
    If A=1 Then
        Risque=0
     ElseIf A=2 Then
        Risque=3
    …
    End If
End If
```

Avec un tableau

C'est plus élégant de créer un tableau à 2 dimensions.

Les colonnes indiquent le sexe, les lignes, la tranche d'âge.

Sexe Masculin Sexe Féminin

| | |
|---|---|
| 0 | 0 |
| 1 | 30 |
| … | … |

```
Dim Risques(,) as Integer ={{0, 0}, {1, 2}, {0, 0}, {2, 3}}
Risque= Risques(S, A)
```

# VII-D - Calculs financiers simples

## VII-D-1 - Conversion francs=>euros

Si un objet coûte 100F, cela fait combien d'euros ?

```
Dim Valeur As Double=100
Dim Resultat As Double
Resultat =Math.Round((Valeur / 6.55957), 2)
```

On divise par 6.55957 puis on arrondit à 2 chiffres après la virgule.

## VII-D-2 - Cout d'augmentation de la vie

Si un objet de 100€ augmente de 3 % par an, combien coûtera -t-il dans 10 ans.

```
Dim Prix As Decimal=100
Dim Taux As Decimal=3

Dim Periode As Integer=10
Dim i As Integer
For i= 1 to Periode
 Prix=Prix+(Prix*Taux/100)

Next i
```

On peut remplacer les 3 dernières lignes par :

```
Prix=Prix*(1+Taux/100)^Periode
```

Noter que l'on utilise des variables de type décimales, c'est une bonne habitude pour faire des calculs financiers (pas d'erreurs d'arrondis).

## VII-D-3 - Remboursement d'un prêt

Quel est le remboursement mensuel d'un prêt d'une somme S durant une durée D (en année) à un taux annuel T ?

R=S x T / 1-(1+T)^-D (ici avec T en % mensuel et D en mois

```
Dim R, S , D, T  As Decimal

S=5000  '5000€

D=15    'Sur 15 ans

T=4     '4% par an

T=T/12/100 'Taux au mois

D=D*12     'Durée en mois

  R=S*T/(1-(T+1)^(-D))'Formule connue par tous bon comptable!!
```

Si on voulait afficher le résultat dans un label (on verra cela plus loin)

```
Label1.text= R.ToString("C")
```

Ici le résultat est transformé en chaine de caractères (grâce à ToString) au format monétaire ("C"), on obtient '36,98€' que l'on met dans le label pour l'afficher.

Ultérieurement on verra un exemple plus complet utilisant les fonctions financières de VB.

# VII-E - Contrôle des connaissances

Voici des exercices sur les notions que vous devez absolument maîtriser en langage Visual Basic.

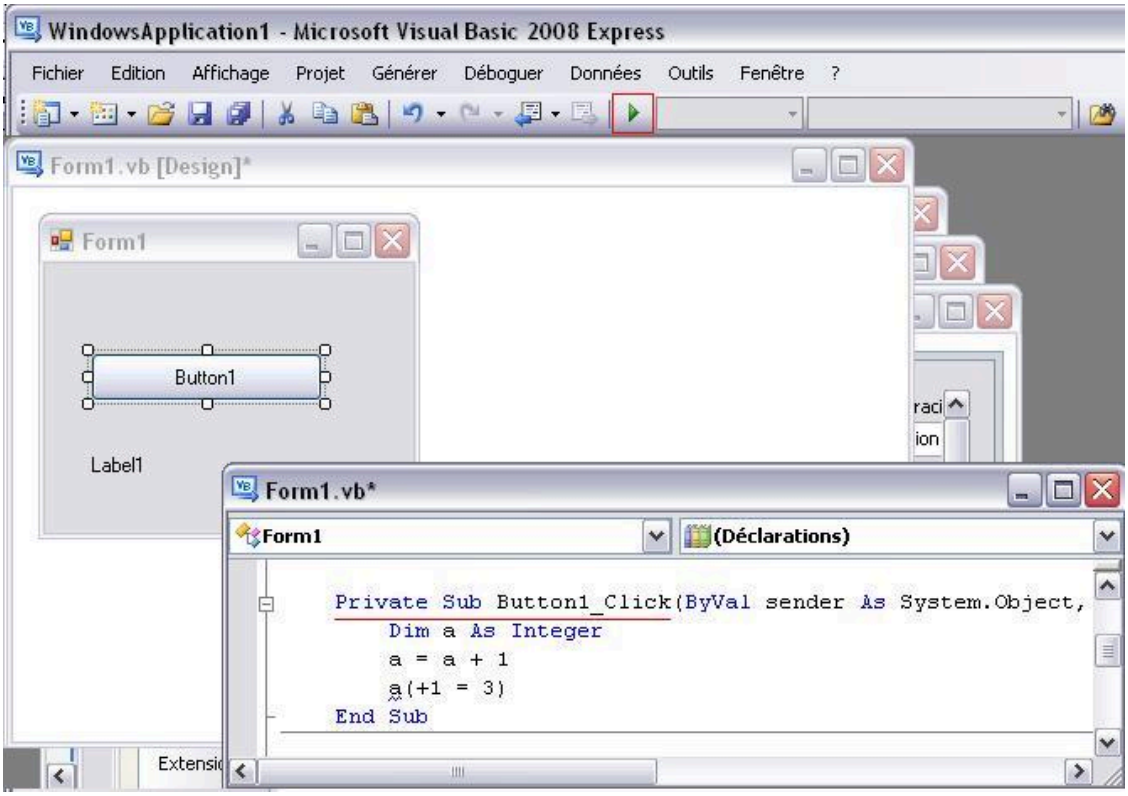
Pour faire les exercices, écrire votre code dans VB et le tester:

Créer une application (menu 'Fichier', 'Nouveau', 'Projet' cliquer sur l'icône 'Application Windows Forms' puis 'OK'.

Dans le formulaire 'Form1', mettre un Bouton 'Button1' et un label nommé 'Label1'.(Pour ajouter un objet sur le formulaire, on clique sur l'objet à gauche dans les 'Outils', puis on clique sur le formulaire , on déplace et on lâche le bouton de la souris).

Double-cliquer sur le bouton 'Button1', la procédure Private Button1_Click(…) apparait.

C'est cette procédure qui sera exécutée lorsque l'utilisateur cliquera sur le bouton.




Mettre votre code (la réponse aux questions) dans cette procédure, entre :

```
Private Button1_Click(…)

End Sub
```

Si vous tapez une erreur, elle est soulignée en ondulé dès que vous quittez la ligne (comme le 'a' de l'exemple).

Cela permet de la corriger. Toutes les réponses sont dans le cours.

Pour exécuter le code cliquer sur la flèche verte en haut, cela démarre le programme.

Dans la fenêtre Form1 qui s'ouvre, cliquer sur le bouton 'Buttom1', cela exécute votre code.

Cliquer sur le carré pour arrêter le programme.

Ces exercices sont effectués avec option Strict= On.

Je donne une solution, mais souvent il y a plusieurs solutions possibles.

## VII-E-1 - Exercices sur les variables

**Questions**

**1.1 Écrire le code créant une variable nommée 'myNumber' devant contenir un entier, donnez-lui la valeur 12. Nommez en termes Visual Basic les différents éléments et étapes.**

**1.2 Voici des déclarations de variables, indiquez les noms de variables corrects et ceux qui ne seront pas acceptés.**

```
        Dim 2a As Integer

        Dim maPremiereVariableDeBoucleInterne As Integer

        Dim nom Utilisateur As String

        Dim MonNom  As String
```

**1.3 Quel type de variable utiliser pour :**

**- faire des calculs financiers ?**

**- mettre un texte de 500 caractères ?**

**- une variable de boucle allant de 0 à 100 et qui soit la plus rapide possible ?**

**1.4 Déclarer une constante qui se nomme myName et lui donner la valeur "lulu".**

**Réponses**

**1.1 Écrire le code créant une variable nommée 'myNumber' devant contenir un entier, donnez-lui la valeur 12. Nommez en termes Visual Basic les différents éléments et étapes.**

```
Dim myNumber As Integer =12
```

On aurait pu aussi écrire :

```
Dim myNumber As Integer

    myNumber = 12
```

On déclare la variable myNumber, elle est de 'type' Integer. On l'initialise avec la valeur 12. '12' est un littéral.

**1.2 Voici des déclarations de variables, indiquer les noms corrects et ceux qui ne seront pas acceptés.**

Dim 2a As Integer Erreur: un nom de variable ne doit pas commencer par un chiffre.

VB souligne 2a et indique, si on met le curseur sur '2a', 'Identificateur attendu', car 2a n'est pas un identificateur (un nom de variable) valide.

Dim maPremiereVariableDeBoucleInterne As Integer Correcte: le nom d'une variable peut être très long.

Dim nom Utilisateur As Integer Erreur: un nom de variable ne doit pas contenir d'espace.

On aurait pu écrire: nom_Utilisateur, car le caractère '_' peut être utilisé.

Dim MonNom As String Correcte et accepté, mais on aurait pu écrire 'monNom' pour suivre les règles de bonnes écritures qui consistent à mettre en majuscule la première lettre de chaque mot sauf pour le premier mot.

**1.3 Quel type de variable utiliser pour :**

Faire des calculs financiers ? Les Decimal.

Mettre un texte de 500 caractères ? Une String.

Un variable de boucle allant de 0 à 100 et qui soit le plus rapide possible ? Un Integer.

**1.4 Déclarer une constante qui se nomme myName et lui donner la valeur "lulu"**

```
Const MyName As String ="lulu"
```

## VII-E-2 - Exercices sur les Strings et Char

**Questions**

**2.1 Créer une variable 's' de type String contenant "45.12", éliminer les espaces de début et de fin, remplacer le point par une virgule s'il y a un point.**

**2.2 Créer une variable s de type String, l'initialiser avec "Philippe", afficher dans une Messagebox la longueur de la chaine.**

**2.3 Créer une variable ch de type Char, y mettre le caractère "2", afficher sur la console True ou False si ch est un chiffre, une lettre.**

**2.4 Créer une variable s de type String, l'initialiser avec "Philippe Dubout". Tester s'il y a un espace dedans, si oui mettre les 2 mots dans un tableau (avec l'instruction Split). Mettre les 2 mots en majuscules. Mettre les 3 premières lettres du premier mot dans une nouvelle variable 'm' puis l'afficher dans une MessageBox. Si le second mot se termine par 'BOUT' afficher 'Se termine par bout'.**

**Réponses**

**2.1 Créer une variable 's' de type String contenant "45.12", éliminer les espaces de début et de fin, remplacer le point par une virgule s'il y a un point.**

```
Dim s As String= "   45.12  "

s= s.Trim(" ")

If s.Contains(".") Then

    s= s.Replace(".","," )

End If
```

Attention avec Option Strict= On, s=s.Trim("") n'est pas accepté (car la chaine de caractères" "n'est pas castée en char). Il faut écrire s=s.Trim(" "c) ou s=s.Trim(CChar(" ")).

**2.2 Créer une variable s de type String, l'initialiser avec "Philippe", afficher dans une MessageBox la longueur de la chaine**

```
Dim s As String ="Philippe"

MsgBox(s.Length.ToString)
```

s.length retournant un entier, il faut le transformer en chaine de caractères (grâce à .ToString) pour l'afficher.

**2.3 Créer une variable ch de type Char, y mettre le caractère "2", afficher sur la console True ou False si ch est un chiffre, une lettre.**

```
Dim ch As Char
ch = "2"c
Console.WriteLine(Char.IsDigit(ch))      ' Output: "True"'  Est un chiffre
Console.WriteLine(Char.IsLetter(ch))     ' Output: "False"' n'est pas  une lettre
```

**2.4 Créer une variable s de type String, l'initialiser avec "Philippe Dubout". Tester s'il y a un espace dedans, si oui mettre les 2 mots dans un tableau (avec l'instruction Split). Mettre les 2 mots en majuscules. Mettre les 3 premières lettres du premier mot dans une nouvelle variable 'm' puis l'afficher dans une MessageBox. Si le second mot se termine par 'BOUT' afficher 'Se termine par bout'.**

```
Dim s As String ="Philippe Dubout"

Dim m As String

If  s.Contains( " ") Then

   Dim mot() As String=s.Split(" "c)

     mot(0)= mot(0).ToUpper

     mot(1)= mot(1).ToUpper

     m = mot(0).Substring(0, 3)

    MsgBox (m)

    If  s.EndsWith ("BOUT")

        MsgBox ("Se termine par BOUT")

     End If

End If
```

Remarquer que Split fonctionne avec comme séparateur des caractères et non des String (d'où le ""c et non le" ").

Notons aussi que, comme avec SubString, une chaine commence par le caractère numéro 0.

## VII-E-3 - Exercices sur les nombres

**Questions**

**3.1 Créez une variable 'i' de type Integer, initialisez-la avec la valeur 2, Incrémentez i (ajouter 1).**

**3.2 Créez une variable 's' de type virgule flottante simple précision, initialisez-la avec la valeur 12,7561 , créez une variable 's1' qui devra contenir la partie entière de 's' créez une variable s2 qui devra contenir 's' arrondi à 2 décimales après la virgule (Comme pour les valeurs monétaires). Que contiendra s1 et s2 à la fin ?**

**3.3 Créez un nombre virgule flottante double précision nommé 'x', initialisez-la avec 123456,45. Calculez le cube de x (x puissance 3), la racine $4^e$ de x :**

## Réponses

### 3.1 Créez une variable 'i' de type Integer, initialisez-la avec la valeur 2, Incrémentez i (ajouter 1).

```
Dim i As Integer=2

i=i+1
```

ou i += 1

### 3.2 Créez une variable 's' de type virgule flottante simple précision, initialisez-la avec la valeur 12,7561 , créez une variable 's1' qui devra contenir la partie entière de 's' créez une variable s2 qui devra contenir 's' arrondi à 2 décimales après la virgule (Comme pour les valeurs monétaires). Que contiendra s1 et s2 à la fin ?

```
Dim s As Single

Dim s1 As Single

Dim s2 As Single

 s =12.7561

s1 =Math.Truncate(s)

s2 =Math.Round(s, 2)
```

s1 sera égal à 12 (partie entière).

s2 sera égal à 12.76 (arrondi à l'entier le plus proche).

On remarque qu'on a déclaré toutes les variables au début (c'est une bonne manière de faire), plutôt qu'au moment où on en a besoin.

On a bien écrit s =12.4561 et pas s =12,4561, car le séparateur décimal pour les littéraux est le point.

Au lieu d'écrire s1 =Math.Truncate(s), on écrit souvent s1 =Int(s) en utilisant une instruction Visual Basic.

Question : pourquoi Math. avant Truncate ? C'est pour indiquer l'espace de noms System.Math qui contient Trunccate ; on aurait pu aussi écrire en tête de module Imports System.Math et ensuite s1 =Truncate(s) aurait été accepté.

### 3.3 Créer un nombre virgule flottante double précision nommé 'x', l'initialiser avec 123456,45. Calculer le cube de x (x puissance 3), la racine 4$^{e}$ de x :

```
Dim x As Double

Dim x1 As Double

Dim x2 As Double

 x =123456.45

x1= Math.Pow(x, 3)

x2= Math.Pow(X, 1/4)
```

Prendre la racine énième d'un nombre revient à le mettre à la puissance 1/N.

donc racine 4$^{e}$ de X : x2= Math.Pow(X, 1/4)

# VII-E-4 - Exercices nombres-String

## Questions

**4.1 Créer une variable 'x' de type Integer, pour toute valeur de x, afficher dans un label 'Label1' "Le cube de 12 est 1728" (exemple si x=12).**

**4.2 Demander dans une InputBox à l'utilisateur de taper un nombre entier. Multiplier ce nombre par 2, afficher le résultat dans une MessageBox (avec l'instruction VB MsgBox puis avec la Classe MesssageBox du Framework).**

**4.3 Même chose que l'exercice 4.2, mais afficher uniquement le résultat si l'utilisateur a bien tapé dans la InputBox une valeur numérique. Si la saisie n'est pas numérique une MessageBox doit indiquer 'Erreur de saisie'.**

**4.4 Même chose que l'exercice 4.2, mais en demandant de taper un nombre avec 2 chiffres après la virgule (calcul sur des Single). Gérer le fait que l'utilisateur peut se tromper et ne pas employer le bon séparateur décimal (en France s'il tape un point au lieu de la virgule par exemple). Utiliser pour la conversion String vers Single une instruction de conversion spécifique (pas CType).**

## Réponses

**4.1 Créer une variable 'x' de type Integer, pour toute valeur de x afficher dans un label 'Label1' "Le cube de 12 est 1728" (exemple si x=12).**

```
Dim X As Integer

x=12

Label1.text= "Le cube de " & X.ToString & "  est " & (Math.Pow(X , 3)).ToString
```

**4.2 Demander dans une InputBox à l'utilisateur de taper un nombre entier. Multiplier ce nombre par 2, afficher le résultat dans une MessageBox (avec l'instruction VB MsgBox puis avec la Classe MesssageBox du Framework).**

```
Dim s as String

Dim i as Integer

s= InputBox ("Test", "Taper un nombre entier")

i= CType(S, Integer)

i=i*2

MsgBox (i.ToString)
```

ou

MessageBox.Show(i.ToString)

L'InputBox retourne une String, il faut la transformer en Integer, effectuer le calcul puis la retransformer en String pour l'afficher.

**4.3 Même chose que l'exercice 4.2, mais afficher uniquement le résultat si l'utilisateur a bien tapé dans la InputBox une valeur numérique. Si la saisie n'est pas numérique, une MessageBox doit indiquer 'Erreur de saisie'.**

```
Dim s as String

Dim i as Integer

s= InputBox ("Test", "Taper un nombre entier")

if  IsNumeric (s) Then

    i= CType(S, Integer)

    i=i*2

    MsgBox (i.ToString)

Else

     MsgBox ("Erreur de saisie")

End If
```

Noter que le code entre If et Else et entre Else et End If est décalé à droite par l'ajout d'espaces ou de Tab, ce qui permet une meilleure lecture du code.

**4.4 Même chose que l'exercice 4.2, mais en demandant de taper un nombre avec 2 chiffres après la virgule (calcul sur des Single). Gérer le fait que l'utilisateur peut se tromper et ne pas employer le bon séparateur décimal (en France s'il tape un point au lieu de la virgule par exemple). Utiliser pour la conversion String vers Single une instruction de conversion spécifique (pas CType).**

```
Dim s as String

Dim i as Single

s= InputBox ("Test", "Taper un nombre avec 2 chiffres après la virgule")

s=s.Replace (".",",")

i= CSng(S)

i=i*2

MsgBox (i.ToString)
```

Comme le séparateur décimal, sur un ordinateur français (Culture Fr) est le ',', on remplace les points par des virgules avant de convertir en Single.

On remarque que le résultat est affiché avec une virgule, car ToString utilise le séparateur de la culture en cours.

## VII-E-5 - Exercices sur les boucles

### Questions

**5.1 Écrire une boucle qui affiche les nombres pairs de 2 à 100 dans le label 'label1'. Affiche 2 puis 4, 6 ,8,….100.**

**5.2 Écrire une boucle qui affiche les nombres allant d'un nombre demandé à l'utilisateur et descendant de ce nombre jusqu'à 1 mais n'affichant pas le nombre 4 (si l'utilisateur tape 8 cela affichera: 8 puis 7 ,6 ,5 ,3 ,2 ,1).**

**5.3 Afficher le plus grand nombre possible dont le carré est inférieur à 1000. En d'autres termes, écrire une boucle qui affiche dans label1 les nombres croissants 1,2 ,3, 4…tant ce que le nombre au carré est inférieur à 1000. Utiliser While pour cette boucle et une variable 'Counter'.**

**5.4 Chercher l'erreur dans ce code qui affiche dans une boite de message les résultats de la table de multiplication de 1 à 9 :**

```
Dim i, j as Integer

For i=1 to 9

 For j=1 To 9

MsgBox (i*j.ToString)

 Next i

Next j
```

**5.5 Demander par une InputBox à l'utilisateur de taper un chiffre entre 1 et 12, vérifier que ce chiffre est bien compris entre 1 et 12 ; si ce n'est pas le cas, reposer la question (utiliser une boucle Do… loop pour boucler en cas de mauvaise réponse).**

Si l'utilisateur tape sur 'Annuler' dans la InputBox cela retourne une chaine vide et cela plante. Comment gérer cela ?

**5.6 Faire une boucle avec Do Loop, tournant de 1 à 100 et additionnant à une variable 'somme' à chaque tour la variable de boucle.**

**Réponses**

**5.1 Écrire une boucle qui affiche les nombres pairs de 2 à 100 dans le label 'Label1'. Affiche 2 puis 4, 6 ,8, ….100.**

```
Dim i as Integer

For i=2 to 100 Step 2

    Label1.Text= i.ToString

    Label1.Refresh

Next i
```

Ici, comme on connait les valeurs de début et de fin, on utilise une boucle For Next.

Step permet de 'boucler' de 2 en 2.

Ne pas oublier Label1.Refresh qui force l'affichage pour chaque tour de la boucle. Sans cela l'affichage serait mis à jour uniquement en fin de procédure.

Noter que le code entre For et Next est décalé à droite par l'ajout d'espaces ou de Tab, ce qui permet une meilleure lecture du code.

**5.2 Écrire une boucle qui affiche les nombres allant d'un nombre demandé à l'utilisateur et descendant de ce nombre jusqu'à 1 mais n'affichant pas le nombre 4 (si l'utilisateur tape 8 cela affichera : 8 puis 7 ,6 ,5 ,3 ,2 ,1).**

```
Dim i as Integer

Dim sfin As String

Dim fin As Integer

sfin= InputBox ("Donner un nombre entier")

fin= CInt(sFin) 'on transforme sfin, une String saisie par l'utilisateur en fin , un Integer.
```

```
For i= fin to 1 Step -1

    if i <>4 Then Label1.Text= i.ToString

    Label1.Refresh

Next i
```

Ici on ne connait pas la valeur de la fin de la boucle, mais elle sera saisie par l'utilisateur et on peut la mettre dans la variable nommée 'fin'. La boucle c'est une boucle descendante, il faut donc un pas négatif : Step -1. On pourrait aussi vérifier par un If fin>1 Then que la valeur de fin n'est pas inférieur ou égale à 1.

**5.3 Afficher le plus grand nombre possible dont le carré est inférieur à 1000. En d'autres termes, écrire une boucle qui affiche dans label1 les nombres croissants 1,2 ,3, 4…tant ce que le nombre au carré est inférieur à 1000. Utiliser While pour cette boucle et une variable 'Counter'.**

A priori, on ne connait pas la valeur de fin de boucle, donc on va utiliser une boucle While plutôt que For Next.

Il faut gérer soi-même la variable de boucle et l'incrémenter, mais en fin de boucle.

```
Dim Counter As Integer = 0
While Counter* Counter < 1000  ' Test la valeur du compteur.
   label1.Text= Counter.ToString

    Counter += 1 ' Incrémente le compteur.
End While
```

**5.4 Chercher l'erreur dans ce code qui affiche dans une boite de message les résultats de la table de multiplication de 1 à 9 :**

```
Dim i, j as Integer

For i=1 to 9

 For j=1 To 9

MsgBox (i*j.ToString)

 Next i

Next j
```

Réponse : erreur sur les variables dans les 2 Next : si le premier For utilise la variable de boucle i, c'est le dernier Next qui doit indiquer la variable i : La boucle interne doit tourner DANS la boucle externe. Le bon code est :

```
Dim i, j as Integer

For i=1 to 9

 For j=1 To 9

MsgBox (i*j.ToString)

 Next j

Next i
```

**5.5 Demander par une InputBox à l'utilisateur de taper un chiffre entre 1 et 12, vérifier que ce chiffre est bien compris entre 1 et 12, si ce n'est pas le cas, reposer la question (utiliser une boucle Do… loop pour boucler en cas de mauvaise réponse) :**

```
Dim rep As String

Dim r As Integer

Do

    rep=InputBox ("Tapez un chiffre entre 1 et 12")

      r= CType(rep,Integer)


Loop Until r>0 And r<13
```

Si l'utilisateur tape sur 'Annuler' dans la InputBox cela retourne une chaine vide et cela plante. Comment gérer cela ?

```
Dim rep As String

Dim r As Integer

Do

    rep=InputBox ("Tapez un chiffre entre 1 et 12")

     If rep = "" Then Exit Do

     r= CType(rep,Integer)


Loop Until r>0 And r<13
```

Exit Do permet de sortir de la boucle Do Loop.

**5.6 Faire une boucle avec Do Loop, tournant de 1 à 100 et additionnant à une variable 'somme' à chaque tour la variable de boucle.**

```
Dim i As Integer = 0

Dim somme As Integer

Do

 i=i+1

 Somme = somme + i

Loop Until i = 100 ' sort de la boucle quand i=100

MsgBox(somme.ToString)
```

## VII-E-6 - Exercice sur les structures et tableaux

### Questions

**6.1 Créer une Structure 'DVD' contenant un 'Numero' (un Integer ), un 'Titre' (une String), un 'Auteur' (une String). Déclarer un tableau structuré de 10 DVD. Indiquer que le dernier DVD doit avoir comme nom 'Red House' et comme auteur 'Clapton'. Afficher dans une MessageBox le titre du premier DVD. Rechercher, à l'aide d'une boucle, les DVD dont l'auteur est 'Clapton' et afficher dans une MessageBox leurs titres.**

**6.2 Déclarer un tableau t de 100 Integer. Le remplir avec un nombre aléatoire compris entre 1 et 100, le trier par ordre croissant.**

**Pour visualiser les éléments du tableau trié, demander à l'utilisateur dans une InputBox un numéro d'élément puis afficher dans une MessageBox la valeur de l'élément. Créer une boucle pour redemander sans cesse un numéro d'élément. Arrêter si l'utilisateur clique sur 'Annuler' dans la InputBox.**

**Réponses**

**6.1 Créer une Structure 'DVD' contenant un 'Numero' (un Integer ), un 'Titre' (une String), un 'Auteur' (une String). Déclarer un tableau structuré de 10 DVD. Indiquer que le dernier DVD doit avoir comme nom 'Red House' et comme auteur 'Clapton'. Afficher dans une MessageBox le titre du premier DVD. Rechercher, à l'aide d'une boucle, les DVD dont l'auteur est 'Clapton' et afficher dans une MessageBox leurs titres.**

Il faut déclarer la structure, mais en haut du module, sous Public Class Form1, pas dans la procédure.

```
Public Class Form1

Public Structure dvd

   Dim Numero     As Integer

   Dim Titre        As String

   Dim Auteur       As String

End Structure
```

Puis dans une procédure il faut déclarer le tableau :

```
Private Button1_Click (…)

Dim i As Integer  'variable de boucle

Dim MesDvd(10) As dvd 'déclaration du tableau.

MesDvd(9).Titre ="Red House"

MesDvd(9).Auteur= "Clapton"

End Sub

MsgBox (MesDvd(0).Titre)

For i=0 to 9

   If MesDvd(i).Auteur= "Clapton" Then

        MsgBox (MesDvd(i).Titre))

   Next i

End Class
```

On remarque que le tableau de 10 éléments va de MesDvd(0) à MesDvd(9).

Au lieu d'écrire For i=0 to 9 on aurait pu écrire For i=0 to MesDvd.Length-1

(MesDvd.Length étant le nombre d'éléments dans MesDvd, MesDvd.Length-1 est l'index du dernier élément).

**6.2 Déclarer un tableau t de 100 Integer. Le remplir avec un nombre aléatoire compris entre 1 et 100, le trier par ordre croissant. Pour visualiser les éléments du tableau trié, demander à l'utilisateur dans une InputBox un numéro d'élément puis afficher dans une MessageBox la valeur de l'élément. Créer une boucle pour redemander sans cesse un numéro d'élément. Arrêter si l'utilisateur clique sur 'Annuler' dans la InputBox.**

```
Dim t(100) As Integer

Dim i, r As Integer

Dim rep As String

Randomize()

For i = 0 To t.Length - 1

t(i) = CType((Int(Rnd() * 100)) + 1, Integer)

Next i

Array.Sort(t)

Do

  rep = InputBox("Voir &#8218;l'élément numéro?")

  If rep = "" Then Exit Do

  r = CType(rep, Integer)

  MsgBox(t(r).ToString)

Loop
```

## VII-E-7 - Exercice sur les collections

### Questions

Si Vb n'accepte pas de créer une collection, c'est que l'espace de noms correspondant n'est pas importé. Il faut écrire tout en haut du module (au-dessus de Public Class) **Imports System.Collections**.

**7.1 Créer une collection de type ListArray nommée 'L', ajouter "Dupont", "Durand", "Dubout" à la collection. Afficher dans une MessageBox le premier élément de la collection puis le dernier. Enlever le second. Si la collection contient "Dubout", ajouter "Toto" à la position où est "Dubout". Créer une boucle qui affiche tous les éléments de la liste.**

**7.2 Créer une collection nommée 'lst' de génériques List(Of) et contenant des entiers Long, ajouter 12, 24, 32. Afficher dans une MessageBox le troisième élément. Créer une boucle pour afficher dans une MessageBox successivement tous les éléments de la liste.**

### Réponses:

**7.1 Créer une collection de type ListArray nommée 'L', ajouter "Dupont", "Durand", "Dubout" à la collection. Afficher dans une MessageBox le premier élément de la collection puis le dernier. Enlever le second. Si la collection contient "Dubout", ajouter "Toto" à la position où est "Dubout". Créer une boucle qui affiche tous les éléments de la liste.**

```
Dim L As New ArrayList()            'On crée une collection ArrayList

L.Add("Dupont")                     'On ajoute un élément à la collection

L.Add("Dubout")                     'On ajoute un élément à la collection

L.Add("Durand")                     'On ajoute un élément à la collection

MsgBox(L(0))                        'On affiche le premier élément

MsgBox(L(L.Count-1))                'On affiche le dernier élément
```

```
'S'il y a 3 éléments dans la ArrayList ce sont les éléments d'index 0,1,2.

L.RemoveAt(1)                       'On enlève l'élément d'index 1 de la liste

If L.Contains ("Dubout") Then

        L.Insert( L.IndexOf("Dubout"), "Toto")

End If



Dim Element As Object

For Each Element in L

    MsgBox( Element )

Next
```

Bien se souvenir qu'une ListArray contient des Objets.

Attention Element étant un objet, si je veux l'afficher par exemple, il faut le 'caster' en String.

Comme on est en option Strict, il ne faut pas écrire L(0).Item, mais L(0)

**7.2 Créer une collection nommée 'lst' de génériques List(Of) et contenant des entiers Long, ajouter 12, 24, 32. Afficher dans une MessageBox le troisième élément. Créer une boucle pour afficher dans une MessageBox successivement tous les éléments de la liste.**

```
Dim lst As New List(Of Long)

lst.Add 12

lst.Add 24

lst.Add 32

MsgBox (lst(2))  'affiche 32

For Each l As Long In lst

MsgBox(l)

Next
```

## VII-E-8 - Exercices sur les fonctions et paramètres

**Questions**

**8.1 Quand employer une 'Function' plutôt qu'une Sub ?**

**8.2 Créer le squelette d'une Sub nommée 'Calcul' recevant 2 paramètres : une String et un Integer (nommés dans la Sub 'Nom' et 'Id'), paramètres passés 'Par Valeur'.**

**Comment utiliser cette Sub dans une autre Sub. Expliquer ce qu'est un paramètre par valeur ?**

**8.3 Créer une Function nommée 'IsPaire' recevant 1 paramètre Integer (nommé dans la Sub 'Nombre'), paramètre passé 'Par Valeur' et retournant un Boolean qui a la valeur True si nombre est pair. Écrire une**

**procédure appelant cette fonction et afficher dans une MessageBox "Le nombre est pair" ou "Le nombre est impair" suivant le cas.**

### Réponses

#### 8.1 Quand employer une 'Function' plutôt qu'une Sub ?

Quand une procédure doit retourner une seule valeur.

#### 8.2 Créer le squelette d'une Sub nommée 'Calcul' recevant 2 paramètres : une String et un Entier (nommés dans la Sub 'Nom' et 'Id'), paramètres passés 'Par Valeur'.

```
Sub Calcul (ByVal Nom As String, ByVal Id As Integer)

End Sub
```

**Comment utiliser cette Sub dans une autre Sub.**

```
Calcul ("Titi",2)
```

ou

```
Dim n As String= "Titi"

Dim i As Integer=2

Calcul (n,i)
```

**Expliquer ce qu'est un paramètre par valeur (ByVal)**

C'est la valeur qui est envoyée et non la référence (l'adresse en mémoire).

Dans l'exemple ci-dessus, c'est "titi" qui est envoyé en premier paramètre et pas l'adresse de "titi". Si dans la Sub je fais Nom="Toto", dans la procédure appelante, n sera toujours égal à "Titi". Si on avait passé Nom en 'ByRef' n aurait été modifié.

#### 8.3 Créer une Function nommée 'IsPaire' recevant 1 paramètre Integer (nommé dans la Sub 'Nombre), paramètre passé 'Par Valeur' et retournant un Boolean qui a la valeur True si nombre est pair.

```
Function IsPaire(ByVal Nombre As Integer) As Boolean

If Nombre Mod (2) = 0 Then

   Return True

Else

   Return False

End If

End Function
```

**Écrire une procédure appelant cette fonction en donnant un nombre et afficher dans une MessageBox "Le nombre est pair" ou "Le nombre est impair" suivant le cas.**

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
 Button1.Click
```

```
If IsPaire(4) Then

  MsgBox("le nombre est pair")

Else

  MsgBox("le nombre est impair")

End If

End Sub
```

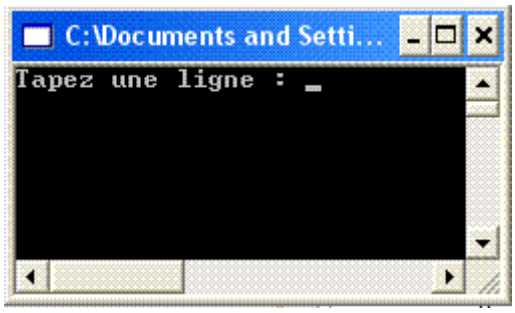
# VIII - Interfaces utilisateur

## VIII-A - Différentes interfaces utilisateur: Console, Windows Forms, WPF

**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.

Où est-il possible de saisir des données au clavier, où seront affichés les résultats ? Quand on crée une application VB, on a le choix.

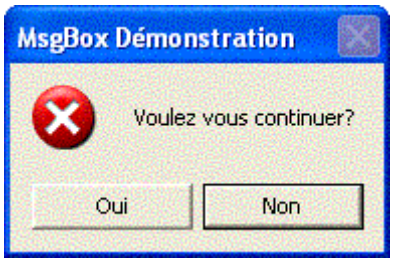
**- Faire une application purement 'Console' :**saisir et afficher sur la console qui est une simple fenêtre de type DOS. Cette méthode utilisant la console, est peu évoluée, archaïque. Pour une Application Console: Menu 'Projet'-> 'Propriétés de …' Combo 'Type de sortie' : Application Console. Dans ce cas la console correspond à une fenêtre type DOS. Et là, on peut entrer des infos à partir du clavier, modifier les couleurs, la position du curseur… On peut écrire du texte sur la console.




Voici un programme purement 'Console'

```
Module Module1

    Sub Main()
        Console.SetWindowSize(65, 35) 'Dimension de la fenêtre
        Console.Title = "test"        'Titre
        Console.BackgroundColor = ConsoleColor.Blue'Couleur du fond
        Console.Clear()  'Applique la couleur du fond
        Console.ForegroundColor = ConsoleColor.Red 'Couleur du texte
        Console.SetCursorPosition(10, 20) 'Position du curseur
        Console.Write("Tapez un texte") 'ecrire sur la console
        Dim ligne As String = Console.In.ReadLine() 'récupérer le texte tapé, le mettre dans
 'ligne'
        Console.Beep  'emettre un Beep
        Console.Beep(800, 1000) 'emettre un son de 800Herz pendant 1000 MilliSecondes
        Console.Read()      'lire à partir de la console
        'Cela permet de figer la fenêtre (sinon elle s'efface instantanément)
        'et de lire jusqu'à ce qu'on tape une touche
    End Sub

End Module
```

**- Faire une application 'WindowsForms' :** créer des fenêtres (ou formulaires Windows Forms ) basées sur GDI, y mettre des boutons, des textbox… pour saisir et afficher. C'est l'interface habituelle des programmes Windows, nous l'étudierons en détail dans les chapitre "Windows Forms" et suivant. Une application Windows (option par défaut), c'est celle que nous utiliserons dans le reste du cours. Menu 'Projet'-> 'Propriétés de …' Combo 'Type de sortie' :Application Windows. On utilise des fenêtres Windows : pour créer l'application.




Dans le cas des Windows Forms, en plus on peut aussi écrire sur la 'console'. La console correspond à la fenêtre de 'Sortie': Menu Affichage>Autres fenêtres>Sortie(Ctrl Alt O). On utilise habituellement cette fenêtre de sortie pour la mise au point et pour afficher des informations de débogage.
Mais là, on ne peut qu'écrire sur la console :

```
Console.write (A+1)
```

**3- À partir de Visual Basic 2008, on peut aussi utiliser les WPF.**

WPF utilise lui un moteur de rendu vectoriel et des accélérations matérielles (Direct X) pour afficher. Cela permet d'afficher de la 2D, de la 3D, des animations, de plus l'affichage étant vectoriel, il n'y a pas de dépendance avec les dimensions de l'écran. Mais c'est un peu complexe pour les débutants.


**4- Programme sans interface.**

Rarement, on n'a pas besoin d'interface, on lance un programme qui effectue une tâche puis se ferme sans avoir besoin de recevoir ou de donner des informations à l'utilisateur. C'est possible en VB.

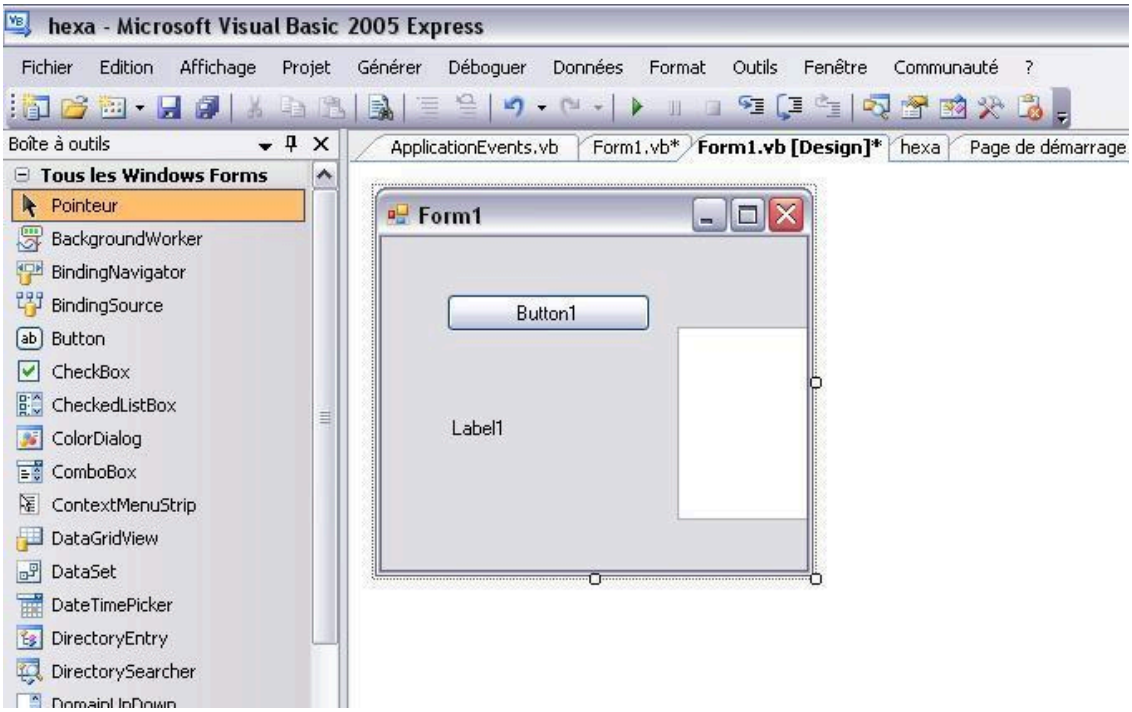
## VIII-B - Interface utilisateur Windows Forms et 'Control'

L'interface utilisateur (IU) correspond aux fenêtres et contrôles que voit l'utilisateur.

Elle assure le dialogue entre l'homme et la machine. C'est l'Interface Homme Machine (IHM). Sous Windows, c'est l'interface graphique.

# VIII-B-1 - En pratique, comment créer l'interface utilisateur ?

On a vu que le développeur dessine cette interface en mode conception (Design) dans l'IDE (Environnement de développement) :




Exemple

On crée une interface utilisateur avec une fenêtre, un bouton, un label (affichant du texte).

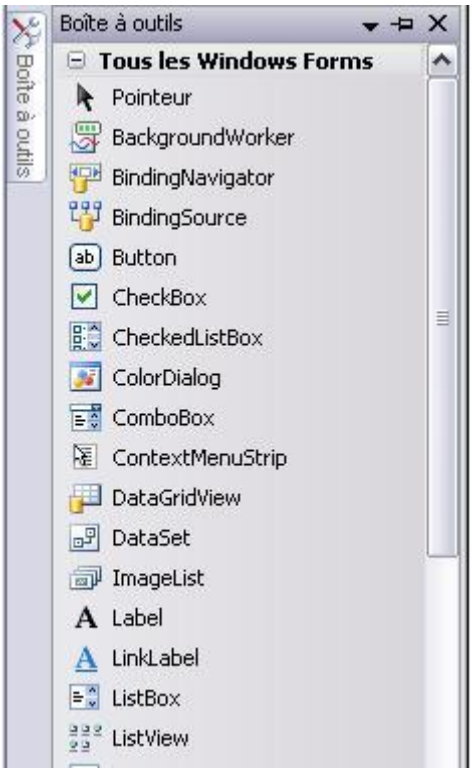
Quand on crée un nouveau projet, il y a création d'un Form1 automatiquement.

Comment rajouter une fenêtre ?

Menu Projet, Ajouter un formulaire Windows, cliquer sur WindowsForm, une fenêtre 'Form1' apparait. On a bien créé une fenêtre avec la classe WindowsForms.Form (en fait on a créé une Classe 'Form1', on verra cela plus loin).

Comment ajouter un bouton ?

Cliquer sur 'Boite à Outils' à gauche, bouton WindowsForms, puis bouton 'Button',cliquer dans Form1, déplacer le curseur sans lâcher le bouton, puis lâcher le bouton : un bouton apparait.




***Comment ajouter un label ?***

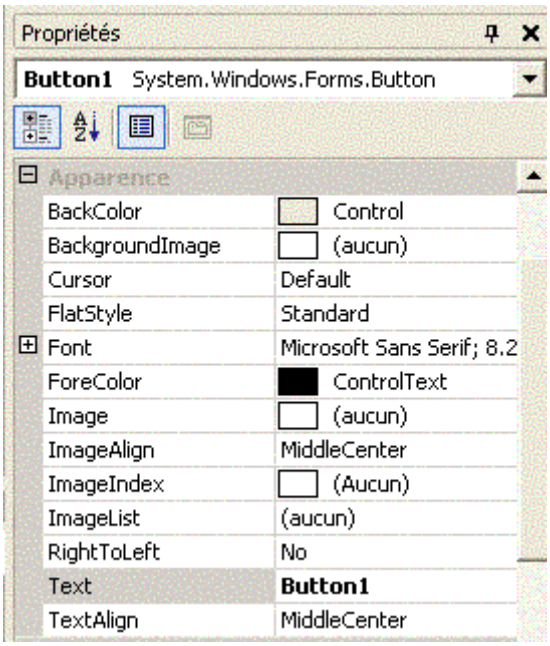
Un label est un contrôle qui permet d'afficher un texte.

Comme pour le bouton cliquer sur 'Boite à Outils' à gauche, bouton WindowsForms, bouton 'Label' et mettre un contrôle label sur la fenêtre.

On obtient dans la fenêtre principale :

***Modifier les propriétés des l'objet (Bouton, label…) en mode Design.***

Il suffit ensuite de modifier les propriétés de l'objet pointé (celui qui est entouré de petits carrés) pour lui donner l'aspect désiré. Les propriétés sont accessibles dans la fenêtre de propriétés de droite.




Noter que pour modifier la taille des objets, on peut le faire très facilement à la souris en cliquant sur les petits carrés entourant l'objet et en tirant les bords. (On peut interdire les modifications de taille et de position des contrôles par le menu Format puis verrouiller les contrôles une fois que leurs tailles et positions est bien définies.)

***Modifier les propriétés des objets (Bouton, label…) par code.***

Dans le code des procédures, les propriétés des objets sont aussi accessibles.

```
Button1.Text="OK"    'Permet par exemple de modifier la propriété Text d'un bouton.
```

## VIII-B-2 - La Classe 'Control'

Les contrôles dits visuels (Button, TextBox ListBox…) héritent tous de la classe **Control** qui hérite elle-même de la classe **System.WindowsForms**.

Autrement dit, les Button, Label, TextBox… sont des 'Control'.

Tous ces objets ont des propriétés communes héritées de la classe Control.

Nous allons voir les plus utilisées.

**Name :** il s'agit du nom de l'objet tel qu'il est géré par l'application.

Par défaut, VB baptise tous les objets que vous créez de noms génériques, comme Form1, Form2, Form3 pour les fenêtres, List1, List2 pour les listes…

Cette propriété est accessible en mode conception uniquement.

*Il est vivement conseillé de renommer les objets que vous venez de créer afin de donner des noms plus évocateurs.*

Le bouton sur lequel est écrit "OK" sera nommé BoutonOK ou ButtonOk if you are anglophile.

La liste qui affiche les utilisateurs sera nommée ListUtilisateurs.

Il est conseillé de commencer le nom de l'objet par un mot évoquant sa nature :

BoutonOk ou BtOk ou ButtonOk, btnOk c'est comme vous voulez.

Microsoft conseille :
**btn** pour les Boutons ;
**lst** pour les ListBox
**chk** pour les CheckBox ;
**cbo** pour les combos ;
**dlg** pour les DialogBox ;
**frm** pour les Form ;
**lbl** pour les labels ;
**txt** pour les Textbox ;
**tb** pour les Toolsbar ;
**rb** pour les radiobutton ;
**mm** pour les menus ;
**tmr** pour les timers.

**Text :** il s'agit du texte qui est associé à l'objet.

Dans le cas d'une fenêtre, c'est le texte qui apparait dans la barre de titre en haut.

Pour un TextBox ou un Label, c'est évidemment le texte qui est affiché.

On peut modifier cette propriété en mode conception ou dans le code.

Exemple : avec du code, comment faire pour que le bouton ButtonOk porte l'inscription 'OK'

```
ButtonOk.Text= "OK"
```

**Enabled :** accessible

Indique si un contrôle peut répondre à une interaction utilisateur.

La propriété Enabled permet l'activation ou la désactivation des contrôles au moment de l'exécution. Vous pouvez désactiver les contrôles ne s'appliquant pas à l'état actuel de l'application. Vous pouvez également désactiver un contrôle pour interdire son utilisation. Par exemple, un bouton peut être désactivé pour empêcher l'utilisateur de cliquer dessus. Si un contrôle est désactivé, il ne peut pas être sélectionné. Un contrôle désactivé est généralement **gris**.

Exemple : désactiver le ButtonOk

```
ButtonOk.Enabled=False
```

**Visible :**

Indique si un contrôle est visible ou non.

```
ButtonOk.Visible=False 'fait disparaitre le bouton.
```



*Attention pour rendre visible une fenêtre on utilise la méthode .Show.*

Exemple :


Button1.Enabled = True (il est accessible )

Button2.Enabled = False (il est grisé, l'utilisateur ne peut pas cliquer dessus)

Button3.Visible = False (on ne voit pas ce bouton 3)

Button4 a le focus

Button5.Text= "OK"

**Font :** permet le choix de la police de caractères affichée dans l'objet.

```
TextBox1.Font = New Font("Courier New", 12, FontStyle.Italic)
```

Pour qu'un TextBox utilise la font "Courier New" de taille 12 en italique.

**BackColor ForeColor :** couleur du fond, couleur de l'avant-plan.

Pour un bouton Forecolor correspond au cadre et aux caractères.

```
ButtonOk.ForeColor=  System.Drawing.Color.Blue
```

**Tag**

Permet de stocker une valeur ou un texte lié à l'objet. Chaque objet a un Tag qui peut contenir un objet.

On l'utilise souvent comme un Flag lié à l'objet.

Par exemple : une liste peut contenir la liste des CD ou des DVD ou des K7, quand je charge la liste des CD, je rajoute List1.Tag="CD" cela permet ultérieurement de voir ce qu'il y a dans la liste.

**Locked**

Si locked est égal à True, le contrôle ne peut pas être déplacé ou redimensionné dans l'IDE (à partir de VB2005). Dans ce cas apparait un petit cadenas.


**Size, MinimunSize et MaximumSize**

Donne les dimensions du control (largeur, hauteur) ou les dimensions minimum et maximum que l'on peut utiliser pour redimensionner le control lorsqu'il est redimensionnable.
On ne peut pas donner la largeur et hauteur directement, il faut instancier un nouvel objet Size :

```
Button.Size = New Size(100, 100) 'Les dimensions sont contenues dans un objet Size que l'on crée.
```

Width et Height : donne la largeur et hauteur.

```
Button.Width = 150  'ici on donne directement la valeur de la largeur en pixels
```

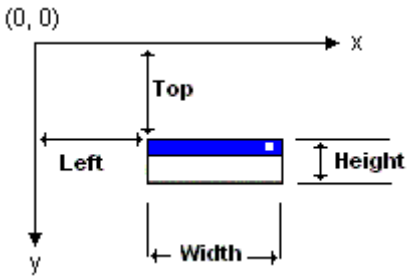
### Location

Donne les coordonnées de la position du coin supérieur gauche du contrôle.
On peut aussi utiliser Location.X et Location.Y (en pixels toujours).
L'origine (coordonnées 0,0) est en haut à gauche.

```
Button1.Location = New Point(100, 100)

'ou pour simplement modifier X:
Button.Location.X = 32
```

Top et Left donnent aussi les coordonnées de la position du coin supérieur gauche (mais en VB 2008, ils n'apparaissent pas dans la fenêtre des propriétés).

```
Button.Left = 32
```




**SetBounds** permet de modifier x, y, width, height.

```
Button.SetBounds(100, 100, 45, 45)
```

**Modifiers** concerne la visibilité de l'objet au niveau programmation.

**Private :** le contrôle est visible dans le formulaire uniquement.

**Public :** visible dans toute la solution.

**Friend :** visible dans tout le programme et les assemblages liés. (Valeur par défaut.)

**Protected :** visible dans le formulaire et les sous-formulaires.

### AutoSize

S'il est égal à True, le contrôle se redimensionne automatiquement en fonction du contenu. Pour un bouton, par exemple la largeur va se modifier en fonction de la longueur du texte qui est dans le bouton, la hauteur aussi en fonction de AutoSizeMode.

Il y a bien d'autres propriétés.

Un contrôle peut en contenir d'autres qui en contiennent d'autres.

Exemple : un formulaire contient un cadre qui contient des boutons.

La propriété **Parent** indique le contrôle conteneur : celui qui contient le contrôle en cours. **FindForm** retourne le formulaire conteneur.

**HasChildren** indique si le contrôle en cours contient d'autres contrôles, dans ce cas, ils sont dans la collection **Controls**.

Exemple

Si un Button1 est dans un formulaire nommé Form1 et dont la barre de titre contient 'Mon formulaire'

Button1.Parent retourne "NomProgramme.Form1 Text: Mon formulaire"

FindForm retourne la même chose ici.

## VIII-B-3 - Événements liés aux objets avec représentation visuelle

On a vu que les objets de l'interface utilisateur ont des procédures déclenchées par les événements de cet objet.

**2 exemples :**

1 Quand l'utilisateur clique sur un bouton OK , la procédure BtOkClick s'effectue ;
2 Quand l'état (coché ou non coché) d'une case à cocher nommée Co change, la procédure Co.CheckedChanged est activée.

La syntaxe complète de la procédure est :

```
Private Sub BtOkClick (ByVal sender As System.Objet, ByVal e As System.EventArgs)  Handles
 BtOk.Clic

End Sub
```

Détaillons.

**Private Sub**

La procédure événement est privée donc accessible uniquement dans le module.

**BtOkClick**

Après le nom Sub, il y a un nom de procédure. Ce nom est construit avec le nom du contrôle et l'événement, mais cela aurait pu être n'importe quel nom, car ce n'est pas ce nom qui indique ce qui déclenche la procédure.

**Sender**

indique le contrôle ayant déclenché l'événement. C'est un Objet.

sender.Name contient par exemple le nom du contrôle ayant déclenché l'événement.

**e**

C'est une variable de type SystemEventArgs qui permet de connaitre l'événement qui a déclenché la procédure.

**Handles**

Indique quels objet et événement ont déclenché la procédure. (On verra qu'il peut y en avoir plusieurs.)

Handles BtOk.Clic indique que c'est l'événement Click sur l'objet BtOk qui déclenche la procédure.

On voit que quand on crée un objet, ses procédures événement sont automatiquement créées.

On se rend compte que dans une procédure événement on peut modifier (en mode conception) ou lire (en mode Run) quel objet et quel événement a déclenché la procédure. On peut même indiquer plusieurs objets liés à cette procédure.

Certains événements sont communs à tous les contrôles :

```
Clic

DoubleClick

GotFocus     Prise du focus

LostFocus    Perte du focus

KeyUp        Remontée d'une touche clavier

KeyPress     Pression d'une touche clavier

KeyDown      Appuie sur une touche clavier

MouseUp      Lâcher le bouton gauche de la souris

MouseDown    Appui sur le bouton gauche de la souris

MouseMove   La souris passe sur le contrôle.

Resize      Modification de la taille (pour un formulaire°
```

Il y a toujours des méthodes **Changed** déclenchées par un changement d'état : CheckedChanged pour une case à cocher, TextChanged pour un contrôle texte.

Pour ne pas alourdir les exemples, nous écrivons souvent une version simplifiée de l'entête de la procédure.

Nous écrivons:

```
Private Sub BtOkClick ()
```

au lieu de :

```
Private Sub BtOkClick (ByVal sender As System.Objet, ByVal e As System.EventArgs)  Handles
 BtOk.Clic
```

**Attention, au Type des paramètres.**

Sender est un 'Object'.

Exemple : dans Form_Load d'une form, le paramètre sender est bien le formulaire qui a déclenché le Load, mais son type est 'Objet'. Et sender.ForeColor n'est pas accepté, car Forecolor n'est pas une propriété d'un objet.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles MyBase.Load

Couleur=sender…ForeColor '<=  ERREUR

End Sub
```

Pour pouvoir utiliser ForeColor, il faut caster (transformer) l'objet en Form.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles MyBase.Load

Couleur=CType(sender, Form).ForeColor

End Sub
```

## VIII-B-4 - En résumé

Le programmeur dessine les fenêtres et contrôles.

Il peut modifier les propriétés des objets dessinés :

par la fenêtre de propriétés (en mode conception) ;

par du code (des instructions) dans les procédures.

## VIII-C - Les fenêtres ou 'Formulaires'

Elles correspondent aux fenêtres ou 'formulaires'. Ce sont les 'Windows Forms'.

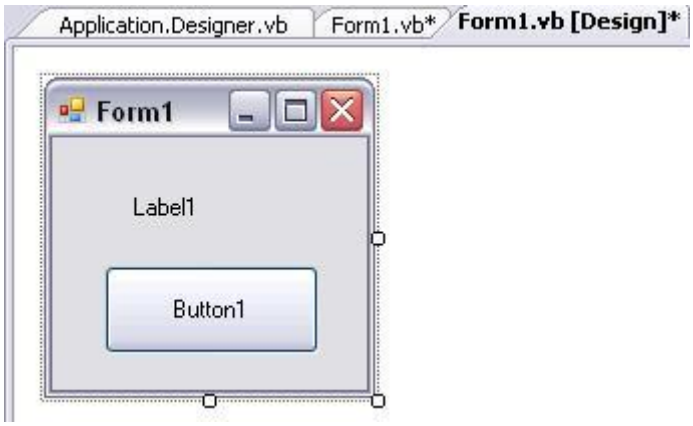
## VIII-C-1 - Créer une fenêtre en mode conception

Quand on ouvre un nouveau projet, il y a déjà une Form1. On peut avoir besoin d'en ajouter une autre au projet.

Menu Projet, Ajouter un formulaire Windows, cliquer sur WindowsForm, une fenêtre 'Form1' apparait. On a bien créé une fenêtre avec la classe WindowsForms.

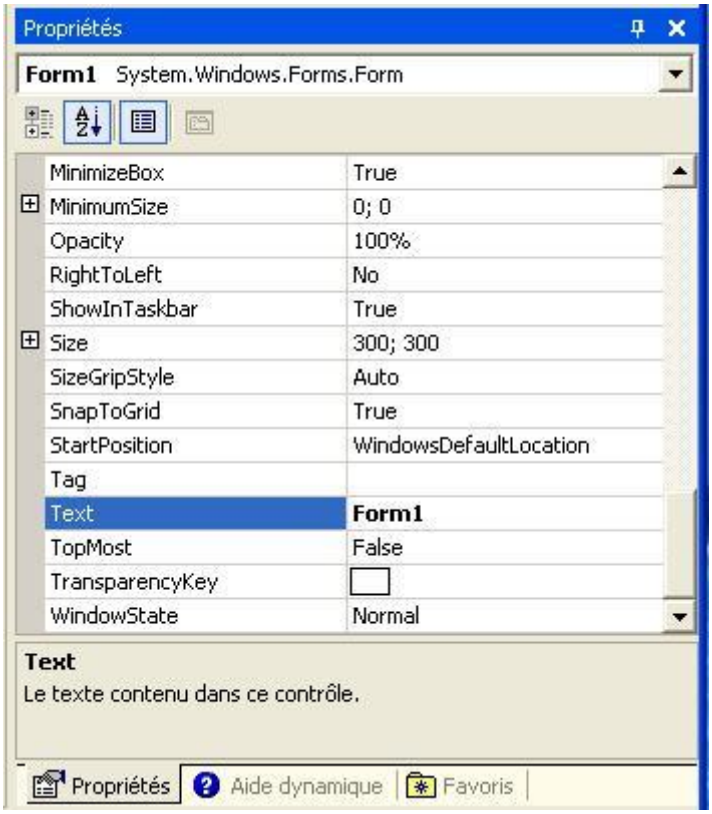
Toute l'interface se trouve sur des fenêtres.

En VB.net on parle de **formulaire**.




## VIII-C-2 - Propriétés

Bien sûr, la fenêtre possède des propriétés qui peuvent être modifiées en mode design dans la fenêtre 'Propriétés' à droite :




Pour modifier la propriété Text par exemple, on clique sur 'Form1' en face de Text, on efface 'Form1' et on tape le nouveau texte.

On peut aussi modifier les propriétés par une ligne de code :

```
Form1.Text = "Nouveau texte"
```

Voyons les choses plus en détail.

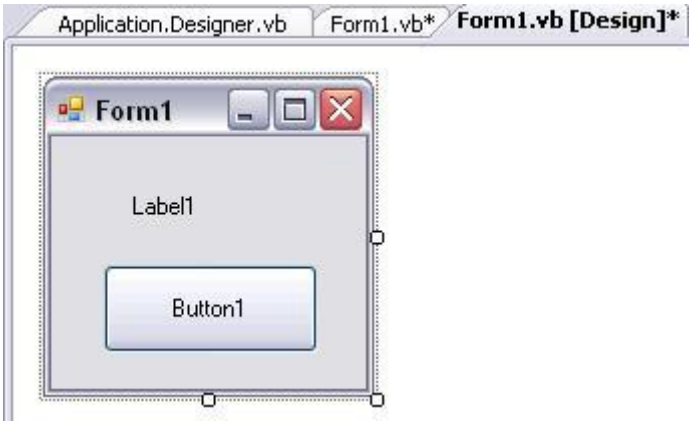
**Name : n**om du formulaire. C'est le nom qui désignera l'objet dans le code.

Donner un nom explicite : FrmDemarrage par exemple.

Dès qu'une fenêtre est créée, on modifie immédiatement ses propriétés en mode conception, dans la fenêtre de propriétés, pour lui donner l'aspect que l'on désire.

**Text :** c'est le texte qui apparaitra dans la barre de titre en haut.

Text peut être modifié par le code : Form1.Text= "Exemple"




**Icon :** propriété qui permet d'associer à la Form un fichier icône. Cette icône s'affiche dans la barre de titre, tout en haut à gauche. Si la Form est la Form par défaut du projet, c'est également cette icône qui symbolisera votre application dans Windows.


Comment mettre l'icône 'euro.ico' dans la barre de titre, par code ?

```
Dim MyIcon As Drawing.Icon = New System.Drawing.Icon("C:\euro.ico")
 Me.ShowIcon =True
 Me.Icon = MyIcon
```

**Comment créer une icône ?**

Dans l'IDE de VB (pas en VB Express !!).

Menu Fichier>Nouveau>Fichier cliquez sur Icon , Vb ouvre une fenêtre Icon1 (dans l'éditeur d'images de Visual Studio.Net). Cela permet de créer ou modifier une icône (Fichier>Ouvrir>Fichier pour modifier).

Comment enregistrer ? Clic droit dans l'onglet 'Icon1' ouvre un menu contextuel permettant d'enregistrer votre Icône.

On peut aussi utiliser des logiciels gratuits pour faire des icônes (Photofiltre est très bien).

**Cursor :** propriété qui permet d'associer à la Form un curseur.


Pour mettre une croix en guise de curseur :

```
Me.Cursor = Cursors.Cross
```

C'est valable dans tous les contrôles.

**WindowState**

Donne l'état de la fenêtre : plein écran (FormWindowState.Maximized), normale (FormWindowState.Normal), dans la barre de tâches (FormWindowState.Minimized).

Exemple : mettre une fenêtre en plein écran avec du code.

```
Me.WindowState =FormWindowState.Maximized
```

(Quand on tape Me.WindowsState= , Vb donne la liste, l'énumération).

Remarque : 'Me' indique l'instance ou le code qui est en train de s'exécuter, dans ce cas Me indique le formulaire en cours.

**ControlBox**

Si cette propriété a comme valeur False, les boutons de contrôle situés à droite de la barre de la fenêtre n'apparaissent pas.

**MaximizeBox**

Si cette propriété a comme valeur False, le bouton de contrôle 'Plein écran' situé à droite de la barre de la fenêtre n'apparait pas.

**MinimizeBox**

Si cette propriété a comme valeur False, le bouton de contrôle 'Minimize' situé à droite de la barre de la fenêtre n'apparait pas.

En résumé :

| ControlBox | MaximizeBox | MinimizeBox | | Result |
|---|---|---|---|---|
| False | | |  | Pas de bouton |
| True | True | True |  | La form peut être minimized ou maximized |
| True | True | False |  | La form peut être maximized mais pas minimized |
| True | False | True |  | La form peut être minimized mais pas maximized |
| True | False | False |  | La form peut être fermée uniquement. |

### FormBorderStyle

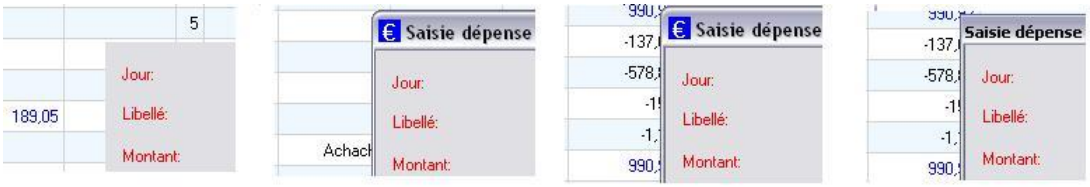
Permet de choisir le type des bords de la fenêtre : sans bord (None), bord simple (FixedSingle) ne permettant pas à l'utilisateur de modifier la taille de la fenêtre, bord permettant la modification de la taille de la fenêtre (Sizable)… Si on a 'None', la barre de titre n'apparait pas.

En VB 2005 il y a aussi Fixed3D qui ajoute un petit effet 3d de profondeur sur le tour, FixedDialog et aussi FixedToolWindows et SizableToolWindows (ces 2 dernières cachent les boutons Maximize et Minimize et l'icône).

Exemple de code :

```
Me.FormBorderStyle =FormBorderStyle.Sizable
```

Cela donne:




None FixedSingle Sizable SizableToolWindows.

### StartPosition

Permet de choisir la position de la fenêtre lors de son ouverture.

Fenêtre au centre de l'écran ? À la position qui existait lors de la conception :

```
Me.StartPosition =FormStartPosition.CenterScreen
```

**Location :** coordonnés du point supérieur gauche par rapport au conteneur (on utilise plutôt StartPosition).

```
    Me.Location = New Point(100, 100) 'c'est un objet 'point' (et non les coordonnés)
```

### Size, MinimunSize et MaximumSize

Donne les dimensions de la Form (largeur, hauteur) ou les dimensions minimum et maximum que l'on peut utiliser pour redimensionner une fenêtre lorsqu'elle est redimensionnable.

**Me.Size = New Size(100, 100)** 'Les dimensions sont contenues dans un objet Size que l'on crée.

Width et Height : donne la largeur et hauteur.

On parle ici des dimensions extérieures du formulaire (Barre de titre et bordures comprises).

Pour avoir les dimensions internes utilisables (sans barre de titre et bordures) il faut utiliser ClientSize.

On remarque que si on modifie ClientSize, cela modifie Size, car la hauteur de la barre de titre n'est pas modifiable.

**Opacity**

Allant de 0 % (0) à 100 % (1), permet de créer un formulaire plus ou moins transparent.

Pour 0 il est transparent, pour 100 il est totalement opaque (normal)

```
Me.Opacity= 50
```

**TransparencyKey** indique une couleur qui va être transparente : si j'indique le jaune, les parties jaunes du formulaire n'apparaitront pas en jaune, mais on verra par transparence ce qu'il y a derrière le formulaire.

**Locked :** propriété qui si elle a la valeur False interdit le déplacement et le redimensionnement de la Form dans le designer.

**Me.Locked= True** interdit le déplacement et les modifications de taille du formulaire dans le designer.

Est présent dans tous les contrôles.

**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.

**La collection Controls** contient les contrôles qui sont dans le formulaire (boutons, listes…).

On peut parcourir la collection, pour voir de quel type est le contrôle :

```
Dim i As Integer
For i = 0 To Me.Controls.Count - 1

… Me.Controls.Item(i).GetType….

Next i
```

Pour un bouton GetType retourne System.Windows.Forms.Button

Le nom du bouton est :

```
Me.Controls.Item(i).Name
```

On verra plus loin que pour ajouter, par code, un contrôle à une form, il faut l'ajouter à la collection Controls :

```
Me.Controls.Add(MyButton)
```

**Exemple complet :**

```
Me.FormBorderStyle= Sizable
```

```
Me.ControlBox=False

Me.Size = New Size(100, 100)

Me.StartPosition = FormStartPosition.CenterScreen

Me.Opacity= 0.75
Me.Text = "Exemple"
```

Donne au milieu de l'écran, la fenêtre :


(Dans notre exemple, on ne s'occupe pas pour le moment du bouton et du label "Bonjour").

## VIII-C-3 - Ouvrir un formulaire

On vient de dessiner une Form1 et une Form2 c'est donc les Class 'Form1 et 'Form2' (les moules) que l'on a dessiné.

Si dans une routine de la Form1 on veut ouvrir une seconde fenêtre de type Form2, il faut :

créer un Objet fenêtre (formulaire) avec le moule Form2 :

```
            Dim f As New Form2()
```

La nouvelle instance f de la Class 'form2' est un objet formulaire.

Pour la faire apparaitre j'utilise la méthode : .ShowDialog.

```
            f.ShowDialog()
```

La fenêtre f est modale, car on a utilisé ShowDialog : quand elle est ouverte, on ne peut pas aller dans une autre fenêtre de l'application avant de sortir de celle-là. (À titre d'exemple les fenêtres MessageBox sont toujours Modales).

Utiliser .Show pour ouvrir une feuille non modale.

```
f.Show()
```



*Attention : une fenêtre est un objet et est 'visible' suivant les règles habituelles des objets.*

Si on instancie une fenêtre à partir d'une procédure, elle sera visible dans cette procédure. Si elle est 'Public' et instanciée dans un module standard, elle sera visible partout.

**f.Activate** donne le focus à un formulaire visible.

## VIII-C-4 - Fermer un formulaire

On utilise Close.

Exemple

**Me.Close** ferme le formulaire courant : le formulaire n'existe plus, il disparait.

Par contre :

**Me.Hide** fait disparaitre le formulaire, il n'est plus visible, mais est toujours présent (Visible passe à False).

## VIII-C-5 - Événements

Au cours de l'exécution

Quand le formulaire est chargé, la procédure **Form1_Load()** est exécutée.

On pourra donc y mettre le code initialisant la feuille.

(Form_Load se produit AVANT l'affichage du formulaire.)

```
Private Sub Form1_Load (…)

    ' Code initialisant les variables, chargeant les listbox…

End Sub
```

Le formulaire est affiché.

**Form1_Activated()** est exécuté ensuite, car la feuille deviendra active.

**Form1.GotFocus()** est enfin exécuté puisque la fenêtre prend le focus (un contrôle qui a le focus est celui qui reçoit les événements clavier, souris… Sa barre de titre n'est plus grisée).

**Form1.Enter ()** est exécuté lorsque l'utilisateur entre dans la fenêtre.

Dès qu'une propriété change de valeur, un événement '**PropriétéChanged**' se déclenche :

**Form1.BackColorChanged** se déclenche par exemple quand la couleur du fond change.

**Form1.Resized** se déclenche quand on modifie la taille de la fenêtre (c'est intéressant pour interdire certaines dimensions).

**Form1.Leave** survient quand il y a perte du focus.

Bien sûr il existe aussi **Form1_Desactivate** quand la fenêtre perd le focus et n'est plus active.

Si l'utilisateur ferme la fenêtre :

**Form1.Closing** se produit avant la fermeture de la fenêtre (on peut annuler cette fermeture en donnant à la variable Cancel la valeur True).

**Form1.Closed** se produit lorsque la fenêtre est fermée.

Il y en a beaucoup d'autres par exemple les événements qui surviennent quand on utilise la souris (MouveUp, MouseDown, MouseMove) ou le clavier (KeyUp, KeyDown, KeyPress) sur la fenêtre.

Exemple pratique

- Comment voir 'bonjour' dans un textbox à l'ouverture du formulaire ?

```
Private Sub Form1_Load()

TextBox1.Text= "bonjour"

End Sub
```

- Comment afficher le formulaire Form1 PUIS une message box affichant "Hello" ?

Si on tape :

```
Private Sub Form1_Load(

MsgBox("hello")

End Sub
```

La message box est affichée Avant la fenêtre !!, car le formulaire s'affiche après Form-Load !!

Il faut écrire :

```
Private Sub Form1_Load
Me.Show() 'affiche la form1

MsgBox("hello")' affiche la messagebox

End Sub
```

Form1 est bien affichée avant la MessageBox.

Si on met le msgbox dans form_ gotfocus ou form_activated, cela boucle, car à la sortie de la msgbox le gotfocus et le activated sont effectués de nouveau.

- Comment utiliser le paramètre sender.

Dans Form_Load par exemple sender.ForeColor n'est pas accepté, car sender est un objet.

Pour pouvoir utiliser ForeColor, il faut caster l'objet en Form.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles MyBase.Load

Dim Couleur As New Color =CType(sender, Form).ForeColor

End Sub
```

## VIII-C-6 - Méthodes

On a déjà vu que pour faire apparaitre une fenêtre il faut utiliser .ShowDialog (pour qu'elle soit modale) ou .Show (pour non modale).

Sachant que **Me** indique l'instance où le code est en train de s'exécuter :

**Me.Close()** ferme le formulaire.

**Me.Activate()** l'active s'il est visible.

**Me.Show()** affiche un formulaire invisible.

**Me.Hide()** rend la fenêtre invisible.

## VIII-C-7 - Form et code généré par vb

On se rend compte que quand on dessine une fenêtre Form1 par exemple, VB crée une nouvelle classe 'Class Form1' (un 'moule' à formulaire !!)

```
Public Class Form1

End Class
```

Quand on tape **Dim f As New Form1()**, on crée une instance de la Class Form1. (Un véritable Objet 'formulaire').

Pour les surdoués

*En VB.Net 2005 ou 2008*

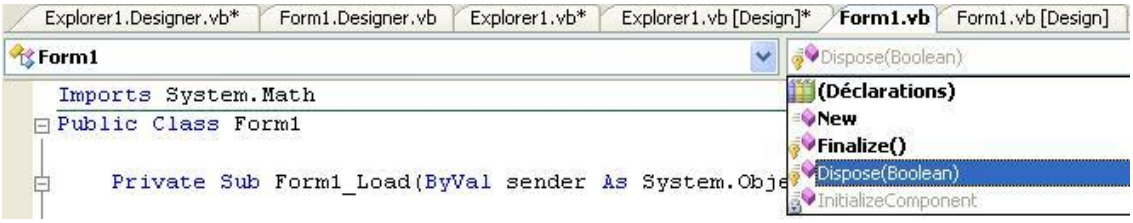
L'onglet Form1.vb contient :

```
Public Class Form1
Public Sub New()

'  Cet appel est requis par le Concepteur Windows Form.

InitializeComponent()    '<=======Appel d'une routine qui 'initialise les composants de la form

' ' Ajoutez une initialisation quelconque après l'appel InitializeComponent().

End Sub
End Class
```

On voit que la Class Form1 contient une routine nommée Sub New.

C'est Sub New qui est exécuté quand on instancie la form (on parle de 'constructeur'. La classe Form1 contient donc un constructeur (c'est la Sub New) qui appelle une routine nommée **InitialiseComponent**.

Pour voir InitializeComponent (le code généré par VB ,celui qui crée le formulaire), il faut, en haut sur la liste déroulante avoir 'Form1', dans ce cas en déroulant la liste à droite, on voit:




Si on clique sur InitializeComponent, l'onglet Form1.Designer.vb apparait.

On a ainsi accès à InitialiseComponent et à Dispose qui sont dans une classe Partielle de Form1.
(À partir de VB 2005, une Classe peut être 'découpée' en Classes partielles).

C'est ici qu'il est indiqué que la Class hérite de System.Windows.Forms.Form et que les contrôles sont crées :

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

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        Try
            If disposing AndAlso components IsNot Nothing Then
                components.Dispose()
            End If
        Finally
            MyBase.Dispose(disposing)
        End Try
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Me.Button1 = New System.Windows.Forms.Button
        Me.Label1 = New System.Windows.Forms.Label
        Me.SuspendLayout()
        '
        'Button1
        '
        Me.Button1.Location = New System.Drawing.Point(26, 64)
        Me.Button1.Name = "Button1"
        Me.Button1.Size = New System.Drawing.Size(107, 44)
        Me.Button1.TabIndex = 0
        Me.Button1.Text = "Button1"
        Me.Button1.UseVisualStyleBackColor = True
```

En conclusion: il faut comprendre qu'à un formulaire (fenêtre) et aux contrôles qui sont dans ce formulaire correspond du code généré par Vb. Ce code (sur lequel vous n'intervenez habituellement pas) permet de créer le formulaire et les contrôles.

## VIII-C-8 - Formulaire d'avant plan, Barre de tâches

Un formulaire d'avant-plan est un formulaire qui reste en avant-plan, devant les autres formulaires, même si on ne l'utilise pas.

Pour définir au moment de la conception un formulaire en tant que formulaire d'avant-plan d'une application :

- dans la fenêtre Propriétés, attribuez à la propriété **TopMost** la valeur true.

Pour définir par code un formulaire en tant que formulaire d'avant-plan d'une application :

- dans une procédure, attribuez à la propriété TopMost la valeur true.

```
    Me.TopMost = True
```

Par contre, **BringToFront** ramène la fenêtre au premier plan temporairement : si l'utilisateur clique sur une autre fenêtre, elle passera au premier plan…

**Me.ShowInTaskBar= True** permet d'afficher l'icône du formulaire dans la barre de tâches (sinon quand on minimise le formulaire, il passe dans une toute petite fenêtre en bas de l'écran).

Dans le chapitre X on apprendra à créer plusieurs formulaires et à les faire communiquer.

## VIII-C-9 - Formulaire non rectangulaire

Pour créer une fenêtre en forme d'ellipse, on affecte à la Region de la fenêtre un GraphicsPath dans lequel il y a une ellipse.
Il faut le faire dans le constructeur de la Form (Form1, New) et ne pas oublier d'importer Drawing2D.

```
Imports System.Drawing.Drawing2D
Public Class Form1

    Public Sub New()

        ' Cet appel est requis par le concepteur.
        InitializeComponent()
        Dim Gr As New GraphicsPath()
        Gr.AddEllipse(0, 0, 250, 350)
        Me.Region = New Region(Gr)


    End Sub
End Class
```

On peut aussi créer une forme de fenêtre à partir de Points :

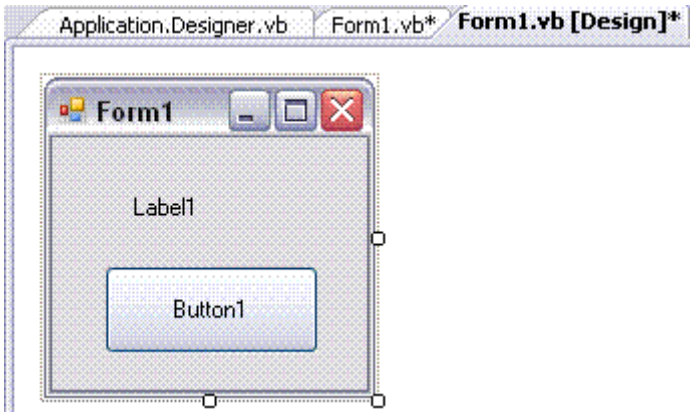
```
Dim Gr As New GraphicsPath()
Dim Points() As New Point{ New Point(10,10), New Point(20,30)……}
Gr.AddLines( Points
Me.Region = New Region(Gr)
```

## VIII-D - Les 'Boutons'

Ils sont omniprésents dans les 'formulaires'.

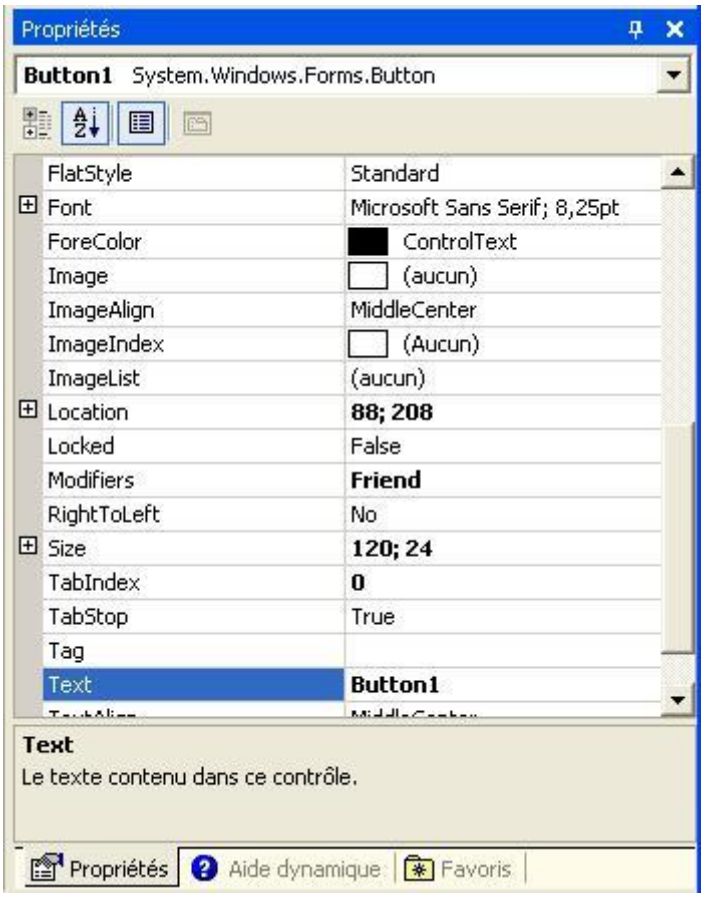
## VIII-D-1 - Créer un bouton

Cliquer sur 'Boite à Outils' à gauche , bouton Windows Forms, puis bouton 'Button',cliquer dans Form1, déplacer le curseur sans lâcher le bouton, puis lâcher le bouton : un bouton apparait.




## VIII-D-2 - Modifier ses propriétés

On peut modifier les propriétés dans la fenêtre des propriétés en bas à droite :




On peut aussi modifier les propriétés par du code.

**Name** est utilisé pour lui donner un nom explicite (BoutonOk BoutonCancel)

**FlatStyle** donne un aspect au bouton (Flat, standard, System, pop Up).

En VB 2003 :


En vb2005 :


System utilise le thème d'affichage de Windows que vous avez choisi dans le panneau de configuration. (Thème Windows XP, personnel…)

Quand on utilise Flat on peut choisir dans FlatStyle l'épaisseur du bord (BorderSize) et sa couleur (BorderColor), 3 et rouge dans notre premier bouton.
MouseDownBackColor et MouseOverBackColor indiquent la couleur d'arrière-plan du bouton quand on clique ou quand on survole le bouton.

Enfin on peut choisir la position du texte avec TextAlign. Il a la valeur TopLeft dans le dernier bouton.

**Text** contient le texte a afficher sur le bouton. ForeColor correspond à la couleur de ce texte (BackColor étant la couleur du fond).

Exemple

button1.Text="OK" affiche 'Ok' dans le bouton.

Si on y inclut un "&" la lettre qui suit sera soulignée et servira de raccourci clavier.

Button.Text= "&OK" donne sur le bouton OK et crée le raccourci clavier 'Ctrl O' qui est l'équivalent d'un clic sur le bouton.

**TextAlign** permet de positionner le texte dans le bouton.

**Image** contient le nom de l'image à afficher sur le bouton (si on veut afficher une image, on le fait en mode Design. Noter que quand on distribue l'application, il n'y a pas besoin de fournir le fichier contenant l'image avec l'application). AlignImage permet de positionner l'image sur le bouton.

On peut aussi puiser une image dans une ImageList grâce à la propriété ImageList et ImageIndex, on peut ainsi changer d'image.

Si le Flatstyle a la valeur 'System' l'image n'apparait pas.

En VB 2008, l'image est chargée dans les ressources. Les formats acceptés sont : bmp, gif, jpg, wpf, png.

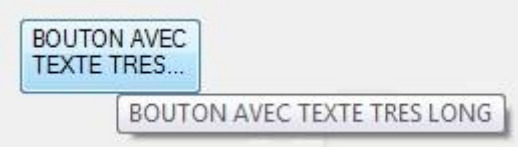
La propriété BackGroundImage permet de mettre une image de fond.

**TextImageRelation** permet de définir les relations entre le texte et l'image : avant, après, dessus (Overlay).

**Font** contient la police de caractères, sa taille, son enrichissement (gras, italique…)

**AutoEllipsis** permet de prendre en charge un texte trop long : il le tronque et ajoute '…', si le curseur de l'utilisateur passe sur le bouton, on voit un ToolTip qui affiche la totalité du texte :

```
Buttom1.AutoEllipsis= True
```




Si **AutoSize**=True, cela ajuste les dimensions du bouton à la longueur du texte : à éviter !

Truc : quand vous travaillez sur de très petits boutons, changer la propriété Font et choisir une petite taille de caractères (8 par exemple).

Habituellement, on modifie l'aspect du bouton dans le designer, mais on peut le faire aussi par code.
Exemple en VB 2008 :

```
With Button1
.Text= "OK"

.Image= MonProgramme.My.Resources.Resources.BtOk

.TextImageRelation=TextBeforeImage

.ImageAlign= MiddleRigth

.TextAlign= MiddleLeft
End With
```

**Padding** permet de positionner le contenu dans le contrôle : remplissage et marges, il est utilisable dans de nombreux contrôles, ici dans un bouton c'est le texte qui est déplacé.

```
Button1.Padding = New Padding(30, 10, 10, 10)
```

## VIII-D-3 - Utiliser les événements

L'événement principalement utilisé est Click() : quand l'utilisateur clique sur le bouton la procédure

```
Private Sub Button_Click(…)

End Sub
```

est traitée.

Cette procédure contient le code qui doit être exécuté lorsque l'utilisateur clique sur le bouton.

Le bouton peut être sélectionné grâce à un clic de souris, à la touche ENTRÉE ou à la BARRE d'espacement si le bouton a le focus.

## VIII-D-4 - Créer un bouton OK ou Cancel

Parfois, il faut permettre aux utilisateurs de sélectionner un bouton en appuyant sur la touche ENTRÉE même si le bouton n'a pas le focus.

Exemple: Il y a sur la fenêtre un bouton "OK" qui doit être enfoncé quand l'utilisateur tape 'Enter' au clavier, c'est le bouton qui 'valide' le questionnaire ( et qui le ferme souvent).

Comment faire ?

Définissez la propriété AcceptButton de la Form en lui donnant le nom du bouton.

Cela permet au formulaire d'avoir le comportement d'une boite de dialogue.

La propriété CancelButton de la Form permet de la même manière de créer un bouton 'Annuler' ( qui répond à la touche 'Echap' (ESC).

## VIII-D-5 - Couleur transparente dans les images des boutons

On a vu qu'on pouvait mettre une image dans un bouton, il faut pour cela donner à la propriété Image le nom du fichier contenant l'image, ceci en mode Design.

Mais l'image est souvent dans un carré et on voudrait ne pas voir le fond (rendre la couleur du fond transparente)

Voici l'image  , je voudrais ne pas afficher le 'jaune' afin de voir ce qu'il y a derrière et donner l'aspect suivant 

Dans Visual Basic 6.0, la propriété MaskColor était utilisée pour définir une couleur qui devait devenir transparente, permettant ainsi l'affichage d'une image d'arrière-plan.

Dans Visual Basic Net, il n'existe pas d'équivalent direct de la propriété MaskColor!! Mais il y a 2 ruses pour arriver à ses fins :

- faire une image GIF avec une couleur 'transparent' autour. Puis la mettre dans le bouton ;

- charger l'image dans le contrôle puis forcer une couleur à devenir transparente.

Dans le "Code généré par le Concepteur Windows Form" après la définition du bouton ou dans Form_Load ajouter :

```
Dim g As New System.Drawing.Bitmap(Button1.Image)

g.MakeTransparent(System.Drawing.Color.Yellow)

Button1.Image = g
```

On récupère le Bitmap de l'image du bouton , on indique que le jaune doit être transparent, on remet le BitMap.

Bien sûr il y a intérêt à choisir une couleur (toujours la même) qui tranche pour les fonds de dessin et ne pas l'utiliser dans le dessin lui-même.

## VIII-D-6 - Utilisation avancée : créer de magnifiques boutons à partir de VB2005

On peut créer des boutons avec ses propres images.

Exemple donné par Microsoft : 

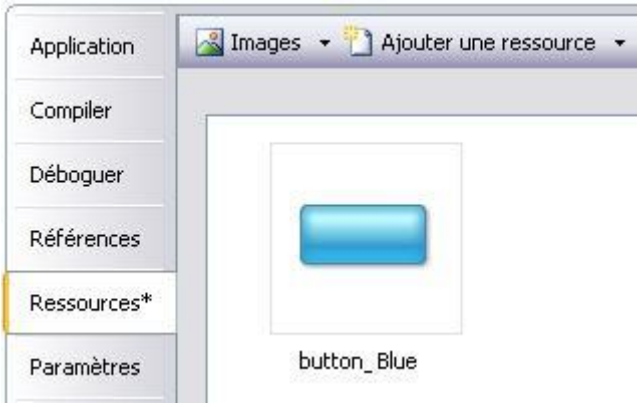

Pour faire cela, il faut créer le dessin, le mettre en fond et paramétrer correctement le bouton.

1-Mettre le dessin dans les ressources

Aller dans les ressources (ensemble d'éléments: images, textes, sons… qui seront incorporés dans le programme).

(Pour cela double-cliquez sur MyProjet dans l'explorateur de solution ou menu 'Projet'=>'Propriétés de…', Onglet 'Ressources'.)

Déroulez la liste à gauche pour y mettre 'Images' puis cliquez sur 'ajouter une ressource', on vous demande le nom de la ressource (tapez par exemple 'button_blue'), vous vous trouvez dans Paint, dessinez (ou collez) l'image de votre bouton. Puis menu 'Fichier'=>'Enregistrer' : le dessin apparait dans les ressources. Fermez Paint.




2-Mettre ce dessin comme fond du bouton :

MyButton.BackGroundImage= MonProgramme.My.Ressources.Ressources.button_Blue
(ou modifier cette propriété BackGroundImage dans la fenêtre de propriétés à droite)

2-Modifier les propriétés du bouton dans la fenêtre de propriétés.

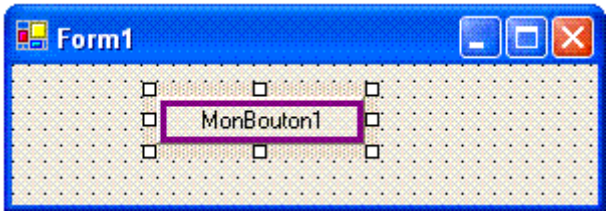
En effet l'image de la ressource n'a pas la même taille que le bouton, de plus le dessin du bouton standard apparait !!

```
MyButton.BackGroundImageLayout= Stretch  'met le dessin à la taille du bouton.

MyButton.FlatStyle= Flat  'fait disparaitre des éléments du bouton standard.

MyButton.BackColor= Transparent  'Efface le fond du bouton standard.

MyButton.Text= "Importer Image'  'met le texte.
```

Super, c'est beau !!

On peut aussi en VB 2003 ou VB2005 personnaliser ses boutons.

Un exemple :




Voir dans le chapitre sur les classes, 'Créer un composant', c'est un peu complexe !!

## VIII-D-7 - Utilisation avancée : création d'un bouton par code

L'exemple suivant crée un Button nommé Button1 sur lequel on voit "OK", on modifie certaines de ses propriétés et on l'ajoute à Form.

```
Private Sub InitializeMonButton()
Dim button1 As New Button1()
button1.Text="OK"
' Ajouter le bouton à la Form
Controls.Add(button1)
End Sub
```

Il faut par code créer aussi les événements liés à ce bouton : dans ce cas il faut déclarer le bouton plutôt avec la syntaxe contenant WithEvents et en haut du module.

```
Private WithEvents Button1 As New Button
```

(Dans ce cas on ne remet pas la ligne Dim button1 dans la Sub InitializeMonButton.)

Puis écrire la sub événement.

```
Sub OnClique ( sender As Objet, EvArg As EventArgs) Handles Button1

End Sub
```

Ainsi VB sait que pour un événement sur le Button1, il faut déclencher la Sub OnClique.

(On reviendra sur cela.)

## VIII-E - Les TextBox

Les contrôles permettant de saisir du texte sont:

les TextBox ;

les RichTextBox ;

les MaskedTextBox (VB2005 Framework2).

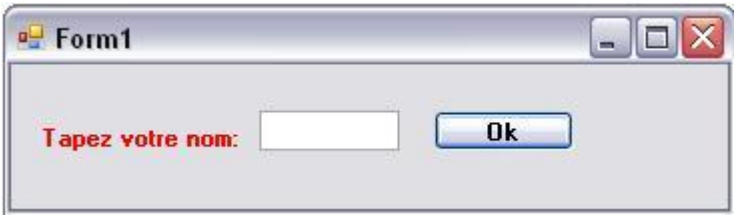
## VIII-E-1 - Les contrôles TextBox

Contrôle qui contient du texte qui peut être modifié par l'utilisateur du programme.

C'est la propriété **Text** qui contient le texte qui a été tapé par l'utilisateur.

**Exemple hyper simple : comment demander son nom à l'utilisateur ?**

Il faut créer un label dont la propriété Text contient "Tapez votre nom :", suivi d'un TextBox nommé txtNom avec une propriété Text="" (Ce qui fait que la TextBox est vide), enfin un bouton nommé btOk dont la propriété Text="OK".Cela donne :




**txtNom.Select()** dans Form_Load donne le focus à la TextBox.

Une fois que l'utilisateur a tapé son nom, il clique sur le bouton 'OK' :

Dans btOk_Click il y a :

```
Dim nom As String

nom= txtNom.Text
```

La variable Nom contient bien maintenant le nom de l'utilisateur.

Un TextBox correspond à un mini éditeur de texte. (Mais sans enrichissement: sans gras, ni italique… du moins pour être exact, l'enrichissement affecte la totalité du texte et pas seulement une partie.) La police de caractères affectant la totalité du texte peut simplement être modifiée par la propriété Font. La couleur du texte peut être modifiée par ForeColor, mais la totalité du texte aura la même couleur.


La propriété **.Text** permet aussi de modifier le texte visible dans le contrôle.

```
TextBox1.Text= "Bonjour" 'Affiche 'Bonjour' dans le contrôle.
```

## VIII-E-1-a - Propriétés

Il y a de multiples propriétés, signalons :

**ReadOnly**. Quand il a la valeur True, on ne peut pas modifier le texte ;

**Multiline :** autorise ou non l'écriture sur plusieurs lignes ;
**Scrollbars :** fait figurer une barre de défilement horizontale ou verticale (ou les deux).

```
TextBox1.Multiline = True
TextBox1.ScrollBars = ScrollBars.Both 'ou None, Horizontal, Vertical
```

Dans ce TextBox multiligne **WordWrap**=True force le passage à la ligne des mots.

**MaxLength :** limite le nombre de caractères qu'il est possible de saisir.

```
TextBox1.MaxLength= 3 'limite la saisie à 3 caractères.
```

Sa valeur par défaut est 32 767.
TextBox1.MaxLength= 0 'ne limite pas la saisie et permet de taper un très long texte.

**TextLength** donne la longueur du texte.

Un TextBox permet de saisir des mots de passe en affichant à la place des caractères tapés des ronds ou un caractère particulier.

```
'Affiche des ronds (caractère system)
 TextBox1.UseSystemPasswordChar = True

 'Affiche des étoiles
TextBox1.PasswordChar = "*"c
```

En mode MultiLine **la collection Lines** contient dans chacun de ses éléments une des lignes affichées dans le contrôle :

TexBox1.Lines(0) contient la première ligne ;

TexBox1.Lines(1) la seconde…

**Les TextBox contiennent une méthode Undo : annulation de la dernière modification.**

La propriété **CanUndo** du TextBox doit être à True.
Il existe aussi ClearUndo qui efface les dernières modifications dans le tampon.

Ensuite pour modifier :

```
If textBox1.CanUndo = True Then
textBox1.Undo()
' Vider le buffer Undo.
textBox1.ClearUndo()
End If
```

**Ajouter au texte**

On peut ajouter du texte au texte déjà présent dans le TextBox :

```
textBox2.AppendText(MonText)
```

C'est équivalent à textBox2.Text=textBox2.Text+MonText

**Pour effacer le texte :**

TextBox1.Clear().

**Événements liés aux TextBox** :

**KeyDown** survient quand on appuie sur la touche ;

**KeyPress** quand la touche est enfoncée ;

**KeyUp** quand on relâche la touche.

Ils surviennent dans cet ordre.

**KeyPress** permet de récupérer la touche tapée dans e.KeyChar (le code Unicode mais pas les touches de fonction de direction comme CTRL,F1, F2, flèche haut…).

**KeyDown** et **KeyUp** permettent de voir ce qui a été tapé physiquement, le code clavier (dans **e.KeyCode**) et aussi si MAJ ALT CTRL ont été pressés (dans e.Shift e.Alt…) :

```
Private Sub TextBox1_KeyUp(ByVal sender As Object, ByVal e As System.Windows.Forms.KeyEventArgs)
 _
 Handles TextBox1.KeyUp
If e.Shift And e.KeyCode= Keys.A Then
    MsgBox ("La lettre 'A' a été tapée")
End If
End Sub
```

On peut récupérer la touche pressée dans KeyPress et depuis VB 2005, on peut modifier e.KeyChar.

Voir exemple plus bas.

**Comment récupérer la totalité du texte qui est dans le TextBox ?**

T= textBox1.Text

**Comment mettre les lignes saisies par l'utilisateur dans un tableau ?**

```
Dim tempArray() as String

tempArray = textBox1.Lines     'On utilise la collection Lines
```

Comment récupérer la première ligne ?

```
T= textBox1.Lines(0)
```

Si une partie du texte est **sélectionnée** par l'utilisateur, on peut la récupérer par :

T= TexBox1.SelectedText

**Pour sélectionner une portion de texte, on utilise :**

```
TextBox1.SelectionStart=3 'position de départ

TextBox1.SelectionLength=4 'nombre de caractères sélectionnés
```

On peut aussi écrire :

TextBox1.Select(3,4)

puis

TexBox1.SelectedText="toto" 'remplace la sélection par 'toto'

TextBox1.DeselectAll() permet de désélectionner.

TextBox1.SelectAll() permet de tout sélectionner.

**Comment utiliser le presse-papier ?**

TextBox1.Cut() 'Coupe le texte sélectionné.
TextBox1.Copy() 'Copie le texte sélectionné.
TextBox1.Paste() 'Colle le texte du presse-papier dans la sélection.

**Comment positionner le curseur après le troisième caractère ?**

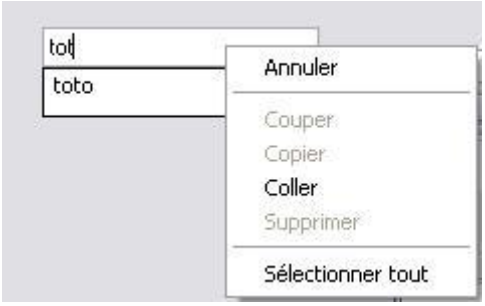
En donnant à la propriété SelectionStart la valeur 3

TextBox1.SelectionStart=3

SelectionLength doit avoir la valeur 0

**AcceptsTab** et **AcceptsReturn** permettent d'accepter dans le TextBox l'utilisation de la touche de tabulation et de celle d'entrée au lieu d'agir sur le formulaire (passage au contrôle suivant et validation du formulaire).

**VB 2005 (le Framework 2) apporte de nouvelles facilités.**
La propriété **ShortCutsEnabled = True** permet à l'utilisateur d'ouvrir un menu contextuel avec le clic droit, ce menu permet les annuler, couper, copier, coller, supprimer, sélectionner tout, ce qui peut aussi être fait avec les raccourcis clavier Shift/Inser Ctrl/Inser…




Il est aussi possible d'aider la saisie de mot.
Exemple : je tape 'tot' le textBox propose 'toto'.

Pour cela dans la fenêtre de propriétés :
- AutoCompletedMode doit prendre la valeur Append ou Suggest à la place de None ;
- AutoCompletedSource indique où chercher les lignes de 'suggestion' ;
- AutoCompletedCustomSource est une liste de lignes utilisées.

Ci-dessus, dans l'exemple :
- AutoCompletedMode= Suggest affiche le mot dessous, Append l'aurait affiché dans le textBox ;
- AutoCompletedSource= CustomSource indique d'utiliser les mots de AutoCompletedCustomSource ;
- AutoCompletedCustomSource contient la liste des mots (pour les rentrer cliquez sur les '…').

## VIII-E-1-b - Validation de saisie

C'est un problème très fréquent : l'utilisateur du logiciel doit saisir un certain nombre ou type de caractères (que des chiffres par exemple) ; il peut parfois y avoir des caractères interdits ; un nombre précis de caractères à saisir…
On va comprendre comment voir quel caractère a été tapé au clavier puis comment l'annuler, le remplacer.

On se souvient que la propriété **MaxLength** limite le nombre de caractères qu'il est possible de saisir.

```
TextBox1.MaxLength= 3 'limite la saisie à 3 caractères.
```

**Différentes manières de récupérer ce qui a été tapé**

L'événement **TextBox1_TextChanged** se produit dès qu'il y a une modification dans le textbox (frappe d'un caractère, mais aussi modification de la propriété Text), dans la procédure TextBox1_TextChanged on peut utiliser TextBox.text qui contient la totalité du texte.

Quand on tape un caractère au clavier, **les événements suivants se déclenchent** (dans l'ordre) :

TextBox1_KeyDown() quand on appuie sur la touche ;

TextBox1_KeyPress() ;

TextBox1_KeyUp() quand on relâche la touche.

TextBox1_KeyDown() et TextBox1_KeyUp() retournent le paramètre e dont la propriété **e.KeyCode** permet de récupérer la touche qui a été tapée.
Pour tester un caractère, on utilise l'énumération Keys.

TextBox1_KeyPress() retourne le paramètre e, sa propriété **e.KeyChar** permet de récupérer le caractère (en Unicode) qui a été tapé.

KeyChar est en lecture/écriture depuis Vb 2005. Mais KeyCode est en lecture seule.

**Comment interdire la frappe de certains caractères dans une TextBox ?**

Exemple 1

Empêcher de saisir le caractère 'A' :

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal e As
 System.Windows.Forms.KeyPressEventArgs) Handles TextBox1.KeyPress
        If e.KeyChar = "A" Then e.KeyChar = CType("", Char)
End Sub
```

On aurait pu aussi écrire :

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal e As
 System.Windows.Forms.KeyPressEventArgs) Handles TextBox1.KeyPress
        If e.KeyChar = "A" Then e.Handler = True
End Sub
```

e.Handler=True indique que l'événement KeyPress a été géré, il n'est donc plus géré et le caractère n'est pas affiché. Ne marche que dans KeyPress.

Exemple 2
Empêcher la saisie de plein de caractères différents :

```
Private Sub TextBox1_KeyPress
    If Not "<>=/*-+".Contains(e.KeyChar) Then e.KeyChar = Nothing
End Sub
```

Exemple 3
Ne permettre de saisir que des chiffres.

Pour cela, il faut utiliser l'événement KeyPress du textBox qui retourne un objet e de type KeyPressEventArgs. e.KeyChar contient le caractère pressé. Pour annuler la frappe (dans notre exemple si le caractère n'est pas un chiffre) il faut faire e.handled=True qui annule la frappe.

IsNumeric permet de tester si le caractère est numérique.

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal e As
 System.Windows.Forms.KeyPressEventArgs)
-Handles TextBox1.KeyPress

If IsNumeric(e.KeyChar) Then
```

```
    e.Handled = False

Else

    e.Handled = True

End If

End Sub
```

Exemple 4
Ne saisir que des majuscules.

À partir de VB 2005 **TextBox1.CharacterCasing = CharacterCasing.Upper** permet de ne saisir que des majuscules.

(Numeric qui ne permet de saisir que du numérique fonctionne uniquement en C#!!)

**Il existe une énumération nommée 'ControlChars'** qui contient quelques touches non imprimables (effacement, entrée…)

Exemple : interdire l'effacement :

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal e As
 System.Windows.Forms.KeyPressEventArgs) Handles TextBox1.KeyPress
        If e.KeyChar = ControlChars.Back Then e.KeyChar = ControlChars.NullChar
    End Sub
```

**Il existe une autre manière simple et élégante de tester si un caractère est numérique, une lettre, un signe de ponctuation, un blanc…**

Pour cela on utilise les propriétés Is… de la classe Char :
Exemple: Éliminer les chiffres :

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal e As
 System.Windows.Forms.KeyPressEventArgs) Handles TextBox1.KeyPress
        If Char.IsNumber(e.KeyChar) = True Then e.Handled = True
End Sub
```

On peut ainsi tester si la touche est :
IsNumber, IsLetter, IsDigit (chiffre), IsControl, IsLower, IsUpper, IsPunctuation, IsSeparator, IsSymbol, IsWhiteSpace, IsLetterOrDigit

**Y a-t-il un moyen de modifier le caractère tapé ?**
Exemple : remplacer une ',' par un '.'
Une solution est de modifier directement le texte :

Exemple : si l'utilisateur tape ',' afficher '.' à la place.

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal e As
 System.Windows.Forms.KeyPressEventArgs) _
Handles TextBox1.KeyPress

Dim pos As Integer

pos = TextBox1.SelectionStart 'on mémorise la position du curseur

If e.KeyChar = "," Then

e.Handled = True 'on ne valide pas le caractère ',' qui n'apparaitra pas.
```

```
TextBox1.Text = TextBox1.Text.Insert(pos, ".")'on insère un '.'

TextBox1.SelectionStart = pos + 1 'on avance le curseur d'un caractère

End If

End Sub
```

Depuis VB 2005 (Framework 2) y a-t-il un moyen plus simple de modifier le caractère tapé ?

Dans la Sub KeyPress e.KeyChar est enfin en lecture-écriture.
On intercepte donc la touche frappée, si nécessaire on la modifie avant qu'elle apparaisse dans le TextBox :

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal e As
 System.Windows.Forms.KeyPressEventArgs)
_Handles TextBox3.KeyPress

    If e.KeyChar = "," Then e.KeyChar = CType(".", Char)

End Sub
```

**Si on veut voir des touches n'ayant pas de caractères Unicode**, il faut utiliser **KeyUp** ou **KeyDown** qui retourne e.KeyCode (code de la touche et non le code Unicode du caractère).

```
Private Sub TextBox1_KeyUp(ByVal sender As Object, ByVal e As
 System.Windows.Forms.KeyPressEventArgs) _
Handles TextBox3.KeyUp
If e.KeyCode = Keys.A Then… (permet de voir si l'utilisateur a tapé sur la touche A.)

End If
```

Notez bien que c'est le code de la touche (et pas du caractère), on peut tester Keys.Escape Keys.Back… avec l'énumération Keys.

Aussi on peut aussi tester :

e.Alt permet de savoir si la touche Alt est enfoncée ;

e.Shift permet de savoir si la touche Shift est enfoncée ;

e.Ctrl permet de savoir si la touche Contrôle est enfoncée.

Comme KeyCode est en ReadOnly, on ne peut pas le modifier ; si on veut l'annuler, il faut passer par KeyPress et son paramètre e.Handler :

```
Public Class Form1
    'Flag indiquant si la touche doit être supprimée
    Dim IsSuppress As Boolean

    Private Sub TextBox1_KeyDown(ByVal sender As Object, ByVal e As
 System.Windows.Forms.KeyEventArgs) Handles TextBox1.KeyDown
        'Dans KeyDown qui survient avant KeyPress
        If e.KeyCode = Keys.Back Then
            'Si la touche est 'Back'(effacement) IsSuppress=True
            IsSuppress = True
        Else
            IsSuppress = False
        End If

    End Sub

    Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal e As
 System.Windows.Forms.KeyPressEventArgs) Handles TextBox1.KeyPress
```

```
        'Si la touche doit être supprimée, on la supprime
        If IsSuppress = True Then e.Handled = True
    End Sub
```

### Compter combien de fois on a tapé certains caractères ?

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal e As
 System.Windows.Forms.KeyPressEventArgs) _
Handles TextBox1.KeyPress

Select Case e.KeyChar
' Compte les  backspaces.
Case ControlChars.Back
    Nombrebackspace = Nombrebackspace + 1
' Compte les  'ENTER' .
Case ControlChars.Lf
    Nombrereturn = Nombrereturn + 1
' Compte les  ESC .
Case Convert.ToChar(27)
    NombreEsc = NombreEsc + 1
' Compte les autres.
Case Else
    keyPressCount = keyPressCount + 1
End Select
End Sub
```

Petite parenthèse

Pour savoir si un caractère a un code Unicode précis il y a 2 méthodes :

if e.KeyChar=Convert.ToChar(27) then

ou

if AscW(e.Keychar)=27 then

### On peut vérifier en quittant le contrôle la validité du texte saisi

Dans ce cas on utilise la Sub 'Validated' qui est effectuée lorsque l'on va quitter le contrôle TextBox.
Exemple : enlever les blancs pour vérifier que le texte contient uniquement les chiffres 0 à 9 et la virgule.
Si le texte n'est pas valide, on efface tout.
On peut coder toutes les opérations, mais pourquoi ne pas utiliser un Regex ?

```
Private Sub TextBox1_Validated(ByVal sender As Object, ByVal e As System.EventArgs) Handles
 TextBox1.Validated
        Dim Contenu As String = TextBox1.Text
        '*** Enlever les blancs
        Contenu = System.Text.RegularExpressions.Regex.Replace(Contenu, "[\s]*", "")
        '*** N'accepter que les entiers ou les décimaux avec 3 chiffres derrière la virgule,
 positifs
        Dim pattern As String
        pattern = "^(([0-9]+)|(([0-9]+)(,)([0-9]{0,3}))?)$"
        If Contenu = System.Text.RegularExpressions.Regex.Match(Contenu,
 pattern).ToString AndAlso Contenu <> "" Then
            TextBox1.Text = Contenu
        Else
            Contenu = ""
            TextBox1.Text = ""
            TextBox1.Focus()
        End If
```

En quittant le contrôle, on veut vérifier si le texte saisi correspond à une **date valide.**

On pourrait , au cours de la saisie (KeyPress) n'accepter que les chiffres et "/" puis à la sortie du contrôle (dans Validated) tester si c'est un format Date grâce à un Regex.
Il y a plus simple : on caste le texte en date, si cela 'plante', le texte ne contient pas une date valide.

```
Private Sub TextBox1_Validating(ByVal sender As Object, ByVal e As
 System.ComponentModel.CancelEventArgs) Handles TextBox1.Validating
        Dim sDate As String = CType(sender, TextBox).Text
        Try
            'On tente de convertir la String saisie en Date
            If sDate <> "" Then sDate = CDate(sDate) 'si la conversion déclenche une exception ce
 n'est pas une date
        Catch
            e.Cancel = True ' on invalide la date
            CType(sender, TextBox).Text = "" 'on remet aussi le textbox=""
        End Try
        MsgBox(sDate)
End Sub
```

e.Cancel = True invalide la saisie : le focus reste dans la textbox.

## VIII-E-2 - Le contrôle RichTextBox

Si vous êtes débutant passez à la rubrique suivante, vous reviendrez plus tard à la gestion du code RTF.

**Rich Text** veut dire 'Texte enrichi'.

Le contrôle RichTextBox permet d'afficher, d'entrer et de manipuler du texte mis en forme. Il effectue les mêmes tâches que le contrôle TextBox, mais il peut également afficher différentes polices de caractères, des couleurs et des liens, charger du texte et des images incorporées à partir d'un fichier, ainsi que rechercher des caractères spécifiques. Ce contrôle manipule du texte au format RTF (le contrôle WPF correspondant n'utilise plus le RTF).

*Le contrôle RichTextBox a les possibilités d'un traitement de texte comme Word.*


**Qu'est-ce que RTF ?**

Le format du texte que l'on peut mettre dans une RichTextBox est le format RTF (Rich Text Format = Format de Texte Enrichi).

Voir dans les annexes 'Format RTF' pour le détail.

Pour utiliser les fonctionnalités du RichTextBox il faut utiliser la propriété .Rtf.

Quand j'affecte un texte à la propriété .Text il est affiché tel quel, sans tenir compte de l'enrichissement.

Quand j'affecte un texte à la propriété .Rtf du contrôle pour l'afficher, s'il contient des enrichissements au format RTF, l'enrichissement est affiché :

**Comment afficher un texte enrichi ?**

```
RichTextBox1.RTF= T         'T étant le texte enrichi
```

**Exemple complet :**

```
"{\rtf1\ansi

 { \colortbl

\red0\green0\blue0;

\red255\green0\blue0;

\red0\green255\blue0;}

{\fonttbl

{\fo\froman Symbol;}

{\f1\fswiss Arial;}

}

Ce qui suit est en \f1 \cf1 \i Arial Rouge Italique \f0 \cf0 \i0

}"
```

Cela donne:


N. B. Si vous voulez copier-coller l'exemple pour l'essayer, enlevez les sauts à la ligne.

**Comment modifier l'aspect du texte qui a été sélectionné ?**

On n'est plus dans le cas où on affiche d'emblée la totalité du texte, mais dans le cas où l'utilisateur veut modifier son texte qui est déjà dans le contrôle.

Exemple : l'utilisateur sélectionne une portion du texte dans le contrôle puis clique sur un bouton nommé 'Rouge' pour mettre la sélection en rouge.

Dans BoutonRouge_Click() écrire :

```
RichTextBox1.SelectionColor = System.Drawing.Color.Red
```

De même pour modifier la police, la hauteur de la police, l'aspect gras ou non :

```
RichTextBox1.SelectionFont = New Font("Tahoma", 12, FontStyle.Bold)
```

**Enfin le texte peut être enregistré dans un fichier :**

```
richTextBox1.SaveFile(FileName, RichTextBoxStreamType.RichText)
```

Si on remplace .RichText par .PlainText c'est le texte brut et non le texte enrichi qui est enregistré.

Pour lire un fichier, il faut employer .LoadFile avec la même syntaxe.

**Comment faire une recherche dans le texte ?**

La fonction **Find** permet de rechercher une chaine de caractères dans le texte :

richTextBox1.Find(searchText, searchStart, searchEnd, RichTextBoxFinds.MatchCase)

La méthode retourne l'emplacement d'index du premier caractère du texte recherché et met en surbrillance ce dernier ; sinon, elle retourne la valeur -1.

Il peut y avoir des liens hypertextes et on peut interdire la modification enfin il y a les méthodes Undo et Redo sur les dernières modifications. Comme pour les textbox il y a une collection Lines() qui contient chacune des lignes.

Si **MaxLength**=0 , la limite du texte est de 64 k caractères.

**Comment imprimer ce que contient la RichTextBox ?**

Rien de simple…

Microsoft fournit le code d'un contrôle nommé ExtendedRichTextBox qui hérite de RichTextBox, mais qui contient en plus une méthode **Print**.

Dur, dur pour le trouver sur le NET !!

Il est sur le site CodeSource:http://www.codeproject.com/KB/vb/WordProcessingPackage.aspx

Exemple en anglais du code d'un traitement de texte (Word processor) utilisant un ExtendedRichTextBox qui permet l'impression.

Le code en VB du contrôle ExtendedRichTextBox est dans l'exemple.

La Solution RichTextEditor contient le code du contrôle ExtendedRichTextBox et le code de l'application RichTextEditor qui utilise le contrôle.

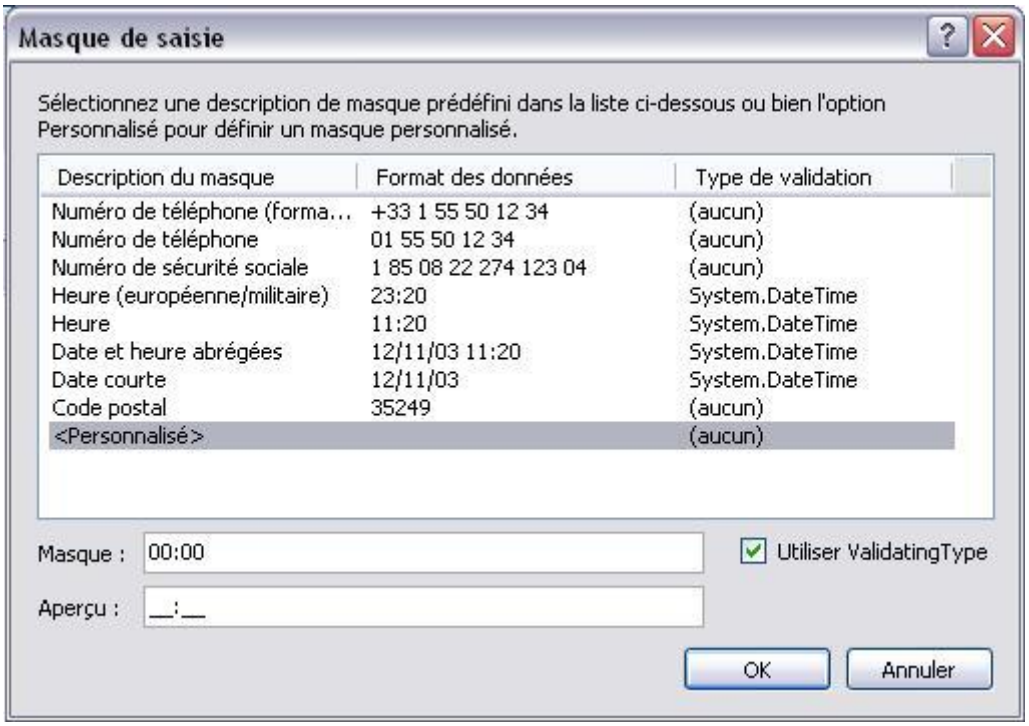
Il faut charger le projet RichTextEditor , le générer (pour que le contrôle soit 'compilé' ), fermer puis rouvrir le projet.

## VIII-E-3 - Le contrôle MaskedTextBox (VB Framework 2)

Permettant d'utiliser un masque pour la saisie de caractères. Le masque indique quels caractères interdire ou permettre.

La propriété Mask permet d'indiquer le masque de saisie.

On peut la modifier en mode 'Design' à partir de la fenêtre 'Propriétés' :




On voit bien dans la fenêtre ci-dessus : le masque '00:00' permet de saisir 2 groupes de 2 chiffres. L'utilisateur ne voit que ce qu'il y a dans l'aperçu '__:__' et est obligé de taper 2 chiffres puis 2 chiffres.

On peut utiliser des masques de la liste (Heure, date, code postal…) ou créer un masque personnalisé.

On peut aussi modifier le masque par code :

maskedTextBox1.Mask = "LL"

Pour le masque personnalisé, on utilise :

```
0 Chiffre requis (lettres refusées)
9 Chiffre ou espace optionnel. (lettres refusées)
# Chiffre ou espace optionnel. (+) (-) sont permis.
L Lettre requise. (chiffres refusés)
? Lettre requise optionnelle. (chiffres refusés)
& Caractère requis.(Chiffres et lettres permises)
C Caractère, requis optionnel.(Chiffres et lettres permises %*& permis)
A Alphanumérique requis opt.(Chiffres et lettres permises %*& refusés)
. Point Decimal; celui de la culture.
, Séparateur de millier; celui de la culture.
: Séparateur de temps; celui de la culture.
/ Séparateur de date; celui de la culture.
$ Symbole monétaire; celui de la culture.
< Convertir les caractères qui suivent en minuscules.
>  Convertir les caractères qui suivent en majuscules.
| Stop la conversion minuscules ou majuscules.
\ Escape. Le caractère qui suit devient un littéral.&#8220;\\&#8221; affichera '\'.
Autres caractères Littéraux. Affichés tels quels
```

Exemple

"00/00/0000" permet de saisir une date.

"LLL" permet de saisir trois lettres (pas des chiffres).

/ $ , : sont dépendant de la culture en cours:

Si le Mask="0$" il apparaitra "_€" en culture française.

On peut modifier cela par FormatProvider property.

MaskCompleted indique si la saisie est conforme.

```
Dim returnValue As Boolean
returnValue = maskedTextBox1.MaskCompleted
```

MaskedTextBox1.text permet de lire le contenu du texte.

L'événement le plus souvent utilisé est TextChanged

```
Private Sub MaTextBox_TextChanged(sender As Object, _
  e As EventArgs) Handles MaTextBox.TextChanged
End Sub
```

## VIII-F - Les 'Labels'

Il y a 2 sortes de Label :

- les 'Label' ;

- les 'LinkLabel'.

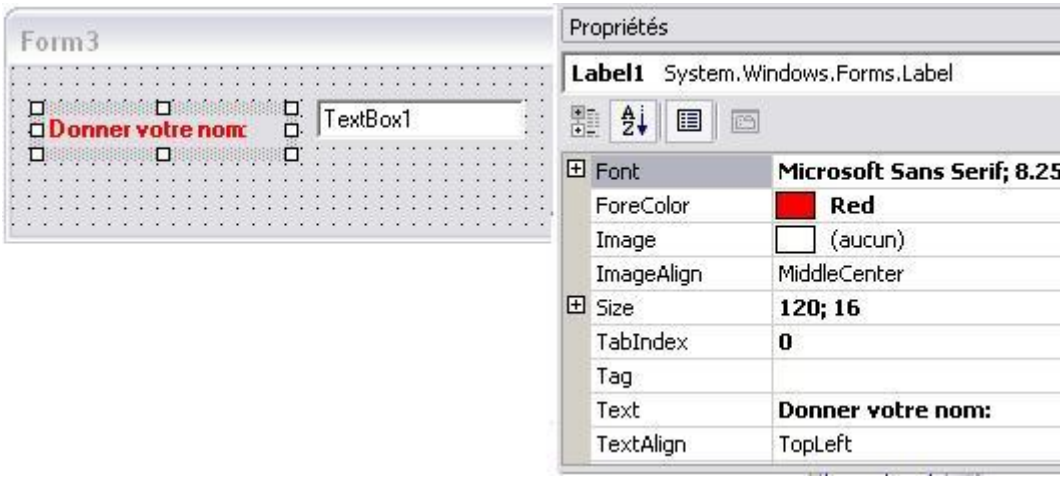
### VIII-F-1 - Les labels

On en a déjà utilisé pour afficher du texte non modifiable par l'utilisateur (ce n'est donc pas une zone de saisie pour l'utilisateur du logiciel).

Les contrôles Label sont généralement utilisés pour fournir un texte descriptif à un contrôle. Vous pouvez par exemple utiliser un contrôle Label pour ajouter un texte descriptif à un contrôle TextBox. Ceci a pour but d'informer l'utilisateur du type de donnée attendu dans le contrôle.

Exemple hyper simple : comment indiquer à l'utilisateur qu'il faut saisir son nom ?

Devant le TextBox permettant la saisie du nom, on ajoute un 'Label' qui affiche 'Donner votre nom:'.




La légende qui s'affiche dans l'étiquette est contenue dans la propriété **Text** du label.

Après avoir déposé le 'Label' sur le formulaire, on peut modifier le texte affiché à partir de la fenêtre de propriétés, en passant par la propriété 'Text'.

On peut aussi mettre la propriété ForeColor à Red pour que le texte du label soit en rouge.

**Pour modifier le texte du label1** par du code :

Label1.Text="Donner votre Prénom"

La propriété **Alignement** vous permet de définir l'alignement du texte dans l'étiquette (centré, à droite, à gauche), BorderStyle permet de mettre une bordure (un tour) ou non…

```
Label1.Text="Label avec une bordure et un fond jaune"

Label1.BorderStyle=BorderStyle.FixedSingle

Label1.ForeColor=Color.Red

Label2.BackColor=Color.Yellow
```

donne:


Remarque : la plupart du temps les labels sont modifiés en mode design, directement dans la fenêtre des propriétés.

Il est également possible d'y afficher une image avec la propriété **.Image**

La propriété **AutoSize** =True, autorise le contrôle à se redimensionner pour afficher la totalité du texte.

**Remarque sur la mise à jour de l'affichage**

La mise à jour de l'affichage du Label (comme les autres contrôles d'ailleurs) est effectuée en fin de Sub.

Si on écrit :

```
Dim i As Integer

For i = 0 To 100
```

```
    Label1.Text = i.ToString

Next i
```

La variable i prend les valeurs 1 à 100, mais à l'affichage rien ne se passe pendant la boucle, VB affiche uniquement 100 à la fin; si on désire voir les chiffres défiler avec affichage de 0 puis 1 puis 2… il faut rafraîchir l'affichage à chaque boucle avec la méthode Refresh() :

```
Dim i As Integer

For i = 0 To 100

    Label1.Text = i.ToString: Label1.Refresh()

Next i
```

Une alternative est de mettre un **Application.DoEvents()** qui donne à Windows le temps de traiter les messages et de rafraîchir l'affichage.

## VIII-F-2 - Les LinkLabel

Permettent de créer un lien sur un label.

Text Indique le texte qui apparait.

LinkArea définit la zone de texte qui agira comme un lien ; dans la fenêtre de propriété taper 11 ;4 (on verra que c'est plus simple que de le faire par code).

Les 4 caractères à partir du 11e seront le lien, ils seront soulignés. Ne pas oublier comme toujours que le premier caractère est le caractère 0.

L'événement LinkClicked est déclenché quand l'utilisateur clique sur le lien. Dans cette procédure on peut permettre le saut vers un site Internet par exemple ou toute autre action.

Exemple :

```
LinkLabel1.text= "Visitez le site LDF"

LinkLabel1.LinkArea = New System.Windows.Forms.LinkArea(11, 4)
```

Pourquoi faire simple !!

Cela affiche un lien hypertexte sur le mot 'site'.

Si l'utilisateur clique sur le mot 'site', la procédure suivante est déclenchée :

```
Private Sub LinkLabel1.LinkClicked&#8230;

End Sub
```

Il est possible de modifier la couleur du lien pour indiquer qu'il a été utilisé :

Si VisitedLinkColor contient une couleur e.visited=True modifie la couleur.

(e est l'élément qui a envoyé l'événement, j'en modifie la propriété Visited.)

On peut y inclure une action quelconque, en particulier un saut vers un site Web :

System.diagnostics.Process.Start("http://plasserre.developpez.com/")

'correspond au code qui ouvre un browser Internet (Internet Explorer …) et qui charge la page dont l'adresse est indiquée.

La collection Links permet d'afficher plusieurs liens dans un même texte, mais cela devient vite très compliqué.

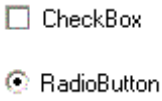
## VIII-G - Les cases à cocher

Il y a 2 sortes de cases à cocher :

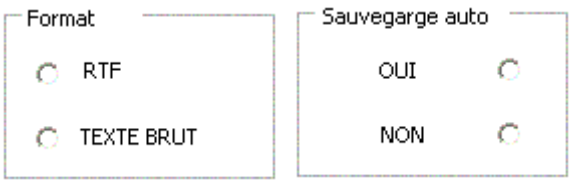
- les CheckBox ;

- les RadioButton.

- Les "cases à cocher" (CheckBox) : elles sont carrées, et indépendantes les unes des autres, si l'utilisateur coche une case, cela n'a pas d'influence sur les autres cases du formulaire, qu'elles soient regroupées dans un cadre pour faire plus joli ou non.
- Les "boutons radio" (RadioButton) : ils sont ronds et font toujours partie d'un groupe (ils sont dans une fenêtre ou dessinés dans un objet GroupBox). Ce groupe est indispensable, car au sein d'un groupe de RadioButton, un seul bouton peut être coché à la fois : si l'utilisateur en coche un, les autres se décochent.




Il faut regrouper les radios boutons dans des 'GroupBox' par exemple pour rendre les groupes indépendants :




Ici si je clique sur le bouton 'OUI' à droite, cela décoche 'NON', mais n'a pas d'influence sur le cadre Format

La propriété Text, bien sûr, permet d'afficher le libellé à côté du bouton, on peut aussi mettre une image avec la propriété Image. CheckAlign permet de mettre la case à cocher à droite ou à gauche du texte, TextAlign permet d'aligner le texte.

Exemple pour le bouton en haut à droite.

```
RadioButton3.Text="OUI"

RadioButton3.TextAlign= MiddleCenter 'Middle=hauteur, center = horizontale

RadioButton3.CheckAlign=MiddleRight
```

La propriété la plus intéressante de ces cases est celle qui nous permet de savoir si elle est cochée ou non et de modifier son état. Cette propriété s'appelle Checked. Sa valeur change de False à True si la case est cochée.

RadioButton.**Checked**=True 'Coche le bouton :

```
If RadioButton.Checked=True  Then ' Teste si le bouton est coché.

End If
```

La procédure RadioButton.CheckedChange() permet d'intercepter le changement d'état d'un bouton.

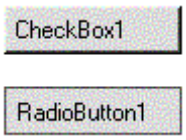
Voici la procédure :

```
Private Sub RadioButton1_CheckedChanged(ByVal sender As System.Object, ByVal e As
 System.EventArgs)
_Handles RadioButton1.CheckedChanged

End Sub
```

Pour le CheckButton **ThreeState**=True permet de définir 3 états au lieu de 2 (coché, indéterminé=grisé, non coché).

CheckedState indique dans ce cas un des 3 états (Checked, Unchecked, Indeterminate) (alors que Checked n'en indique que deux).

**Appearance** peut aussi donner une apparence de bouton à la case à cocher. Il est enfoncé ou pas en fonction de la valeur de Checked.




Ici les 2 boutons ont une Appearance=Button , celui du haut n'est pas coché, l'autre est coché (enfoncé).

Autocheck = True par défaut : quand on clique, l'état change automatiquement. Si AutoCheck= False il faut gérer soi-même le changement d'état avec Checked.

## VIII-H - Les 'Listes'

Il y a 6 sortes de contrôle affichant des listes :

les ListBox ;

les CheckedListBox ;

les Combos ;

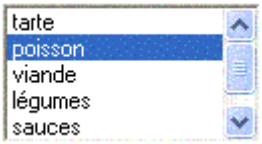
les ListView ;

les DomainUpDown et NumericUpDown ;

les TreeView.

## VIII-H-1 - Les 'ListBox'

Le contrôle ListBox affiche une liste d'éléments (d'objets) dans laquelle l'utilisateur peut faire un ou plusieurs choix.




La liste contient "tarte", "poisson", "viande", "légumes", "sauces".

Ici l'élément "poisson" est sélectionné, la ligne correspondante est en bleu.

La listBox contient une collection d'"Item".

Elle est vide au départ.

Si j'ajoute un élément à la ListBox, cela ajoute un élément à la collection Items.

ListBox1.Items est une collection contenant tous les éléments (les objets) chargés dans la liste.

La propriété Items.Count indique le nombre d'éléments contenus dans la liste. Attention le premier élément est toujours l'élément 0, aussi le nombre d'éléments est égal au numéro de l'élément le plus haut plus un.

La barre de défilement verticale s'affiche si la ListBox contient plus d'éléments qu'il y en a de visibles.
Si **ListBox.ScrollAlwaysVisible= True**, la barre de défilement verticale sera toujours affichée.

### VIII-H-1-a - Pour ajouter ou supprimer des éléments dans un contrôle ListBox

Utilisez la méthode Items.Add, Items.Insert, Items.Clear ou Items.Remove. En mode conception, vous pouvez également utiliser la propriété Items.

### VIII-H-1-b - Vider la ListBox

ListBox1.Items.Clear()

### VIII-H-1-c - Ajouter un ou des éléments

ListBox1.Items.Add("poisson")

Ajouter '4' :

ListBox1.Items.Add(4.Tostring)

ou

ListBox1.Items.Add(4) 'accepté, car les items sont des objets.

Insérer 'lulu en 4e position :

ListBox1.Items.Insert(4, "lulu")

Pour ajouter plusieurs éléments en une seule fois (et éviter le scintillement de l'affichage) on utilise **Items.AddRange :**

```
'Ajout d'un tableau
Dim t() As String {"Franc", "Norgege", Thaïlande" }
Me.ListBox1.Items.AddRange(t)

'ou
 Me.ListBox1.Items.AddRange(New Object() _
            {"France", "Norvege", "Thaïlande", _
            "Australie", "Italie", "Espagne"})
```

**Les listBox acceptent des objets**, elles utilisent ToString sur l'objet pour afficher ; elles affichent généralement ce qu'il y a dans la propriété 'Text' de l'objet.

## VIII-H-1-d - Charger dans une ListBox1 les nombres de 1 à 100

```
Dim i As Integer
For i = 1 To 100

    ListBox1.Items.Add(i.ToString)

Next i
```

## VIII-H-1-e - Comment enlever des éléments ?

```
' Enlever l'élément d'index 5:
ListBox1.Items.RemoveAt(5)


' Enlever l'élément sélectionné:
ListBox1.Items.Remove(ListBox1.SelectedItem)


' Enlever l'élément "Tokyo":
ListBox1.Items.Remove("Tokyo")
```

## VIII-H-1-f - Comment lire l'élément 3 ?

```
Dim t As String

t=ListBox1.Items(3).ToString
```

(En Option=Strict il est nécessaire de transformer l'objet Items(3) en String avec .ToString)

## VIII-H-1-g - Comment rechercher l'élément qui contient une chaine de caractères ?

```
Dim x As Integer

x=List1.FindString("pa")
'retourne le numéro du premier élément commençant par 'pa'.

x=List1.FindString("pa",12)
'retourne le numéro de l'élément commençant par 'pa' en cherchant à partir de l'élément numéro
 12.

x=List1.FindStringExact("papier")
'permet de rechercher l'élément correspondant exactement à la chaine.
```

## VIII-H-1-h - Comment sélectionner un élément par code ?

```
ListBox1.SetSelected(x, True) 'la ligne contenant x devient bleue
```

## VIII-H-1-i - L'utilisateur double-clique sur l'un des éléments, comment récupérer son numéro ?

Grâce à SelectedIndex :

```
Private Sub ListBox_DoubleClick.

    N=ListBox1.SelectedIndex

End If
```

N contient le numéro de l'élément sélectionné. Attention comme d'habitude, si je sélectionne la troisième ligne c'est en faite l'élément numéro 2.

SelectedIndex retourne donc un entier correspondant à l'élément sélectionné dans la zone de liste. Si aucun élément n'est sélectionné, la valeur de la propriété SelectedIndex est égale à -1.

La propriété **SelectedItem** retourne l'élément sélectionné ("poisson" dans l'exemple si dessus).

## VIII-H-1-j - Et la multisélection, quels éléments ont été sélectionnés ?

La propriété SelectionMode indique le nombre d'éléments pouvant être sélectionnés en même temps.

Lorsque plusieurs éléments sont sélectionnés, la valeur de la propriété SelectedIndex correspond au rang du premier élément sélectionné dans la liste. Les collections SelectedItems et SelectedIndices contiennent les éléments et les numéros d'index sélectionnés.

## VIII-H-1-k - On peut 'charger' une ListBox automatiquement avec un tableau en utilisant Datasource

```
Dim LaList() As String = {"one", "two", "three"}
ListBox1.DataSource = LaList
```

On peut aussi utiliser AddRange :

```
Dim Ite(9) As System.Object
Dim i As Integer
For i = 0 To 9
    Ite(i) = "Item" & i
Next i


ListBox1.Items.AddRange(Ite)
```

On peut 'charger' une ListBox avec les éléments d'une énumération:**GetValues**, quand on lui donne le type de l'énumération retourne la liste des éléments de l'énumération.

```
'En tête de module.
 Enum Nom
  Pierre
  Paul
  Jean
  End Enum
```

```
ListBox1.DataSource = [Enum].GetValues(GetType(TypeFichier))
```

## VIII-H-1-l - Comment 'charger' une ListBox automatiquement à partir d'un fichier texte

```
ListBox1.Items.AddRange(System.IO.File.ReadAllLines("c:\list.txt"))
```

(le fichier list.txt est un fichier .txt créé avec NotePad par exemple et contenant les items séparés par des retours à la ligne).

Exemple de fichier :

"philippe

paul

jean

luc"

## VIII-H-1-m - Comment connaitre l'index de l'élément que l'on vient d'ajouter (et le sélectionner) ?

```
Dim x As Integer
x = List1.Items.Add("Hello")
List1.SelectedIndex = x
```

On utilise la valeur retournée (x dans notre exemple) par la méthode Add.

(NewIndex n'existe plus en VB.NET.)

## VIII-H-1-n - Comment affecter à chaque élément de la liste un numéro, une clé ?

Exemple : dans un programme, chaque utilisateur a un nom et un numéro ; je charge dans une ListBox la liste du nom des utilisateurs ; quand on clique sur la liste, je veux récupérer le numéro de l'utilisateur (pas l'index de l'élément).

Comment donc, à chaque élément de la Listbox, associer un numéro (différent de l'index).

- En VB6 on utilisait une propriété (ListBox.ItemData()) pour lier à chaque élément de la ListBox un nombre (une clé) ; cela n'existe plus en VB.Ne t!!

Il existe des fonctions de compatibilité VB6, mais il faut éviter de les utiliser :

VB6.SetItemData(ListBox1, 0, 123) 'pour lier à l'élément 0 la valeur 123.

- Une alternative, pas très élégante :

ajouter l'élément "toto"+ ControlsChar.Tab+ clé (clé n'est pas visible, car les caractères de tabulation l'ont affichée hors de la Listbox).

Quand l'utilisateur clique sur la ligne, on récupère la partie droite donc la clé.

On peut aussi utiliser un Listview avec 2 colonnes ; la seconde colonne servant à stocker le numéro.

- Une solution plus élégante :

on utilise le Tag du ListBox (le Tag est une propriété qui peut contenir un objet (un tableau par exemple ; chaque élément de ce tableau va contenir le numéro de l'utilisateur.)

```
Dim pos As Integer

ReDim ListBox1.Tag(100)  'Création du tableau dans le Tag



pos = ListBox1.Items.Add("Utilisateur1") 'On ajoute le nom de l'utilisateur1,
'pos est l'index de l'élément ajouté

ListBox1.Tag(pos) = 1  'On ajoute dans le Tag  le numéro de l'utilisateur

pos = ListBox1.Items.Add("Utilisateur2")

ListBox1.Tag(pos) = 2

pos = ListBox1.Items.Add("Utilisateur3")

ListBox1.Tag(pos) = 3



'Quand l'utilisateur double-clique dans la liste, on récupère le numéro correspondant

Private Sub ListBox1_DoubleClick()

    MsgBox(ListBox1.Tag(ListBox1.SelectedIndex))

End Sub
```

La contrainte est qu'il faut connaitre le nombre maximum d'éléments à charger et charger la liste dans l'ordre ; il faut être en Option Strict=Off (sinon il y a liaison tardive).

-Enfin on peut utiliser un tableau de Structure (ou d'objets définis par une Classe), ce tableau sera utilisé comme DataSource pour la ListBox, DisplayMember sera affiché, ValueMember contiendra l'Id.

```
Structure NomId
        Property Nom As String
        Property Id As Integer
End Structure

'…
    Dim noms(10) As NomId
        noms(0).Nom = "Paul"
        noms(0).Id = 1
        noms(1).Nom = "Philippe"
        noms(1).Id = 2
        noms(2).Nom = "Louis"
        noms(2).Id = 3

        ListBox1.DisplayMember = "Nom"
        ListBox1.ValueMember = "Id"
        ListBox1.DataSource = noms

'On récupère l'Id quand on double-clique sur une ligne de la ListBox

    Private Sub ListBox1_DoubleClick(ByVal sender As Object, ByVal e As System.EventArgs) Handles
 ListBox1.DoubleClick
        MsgBox(CType(ListBox1.SelectedItem, NomId).Id)
    End Sub
```

Attention dans la structure il faut utiliser des Property.

Quand on charge une ListBox directement avec une base de données, c'est le même principe.

Remarque

Lorsque la propriété MultiColumn a la valeur true, la liste s'affiche avec une barre de défilement horizontale. Lorsque la propriété ScrollAlwaysVisible a la valeur true, la barre de défilement s'affiche, quel que soit le nombre d'éléments.

## VIII-H-1-o - Comment, à partir des coordonnées de la souris, connaitre l'élément de la liste qui est survolé ?

Exemple : la souris survole ListBox2 , on a e.X et e.Y, coordonnées de l'écran, comment obtenir l'index.

On va d'abord transformer e.X et e.Y en coordonnées client (par rapport à la listBox)

ListBox2.PointToClient(New Point(e.X, e.Y)

Puis ListBox2.IndexFromPoint() va retourner l'index survolé.

```
Private Sub ListBox2_DragOver(ByVal sender As Object, ByVal e As
 System.Windows.Forms.DragEventArgs)
_Handles ListBox2.DragOver

IndexdInsertion = ListBox2.IndexFromPoint(ListBox2.PointToClient(New Point(e.X, e.Y)))

End Sub
```

## VIII-H-1-p - Trier les items de la ListBox

*Si la propriété Sorted = True, les items de la liste sont triés automatiquement.*

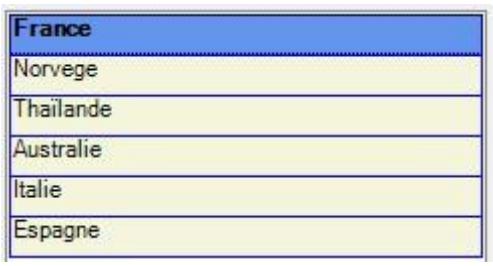
## VIII-H-1-q - Modifier l'affichage des Items dans une ListBox

Comment customiser sa ListBox ? (Mettre l'élément selectionné en gras avec fond bleu, modifier la hauteur des items.)

Si ListBox1.DrawMode = DrawMode.OwnerDrawVariable, ce n'est plus VB qui affiche automatiquement les Items. On devra écrire le code des Sub MeasureItem (gestion dimension des items) et DrawItem (dessine l'item). Attention, il faut tracer le cadre, écrire le texte…

```
Friend WithEvents ListBox1 As System.Windows.Forms.ListBox

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles MyBase.Load
        InitializeOwnerDrawnListBox()
    End Sub

    Private Sub InitializeOwnerDrawnListBox()
        Me.ListBox1 = New System.Windows.Forms.ListBox

        ' Position et dimension.
        ListBox1.Location = New Point(20, 20)
        ListBox1.Size = New Size(240, 150)

        ' Ajout de 6 items avec AddRange
        Me.ListBox1.Items.AddRange(New Object() _
```

```
            {"France", "Norvege", "Thaïlande", _
            "Australie", "Italie", "Espagne"})

        ' Pas de scrollbar.
        ListBox1.ScrollAlwaysVisible = False

        ' Mettre un bord.
        ListBox1.BorderStyle = BorderStyle.Fixed3D

        ' Mettre la propriété DrawMode = OwnerDrawVariable .
        ' On devra écrire le code des Sub MeasureItem et DrawItem
        ListBox1.DrawMode = DrawMode.OwnerDrawVariable
        'On ajoute le contrôle au formulaire
        Me.Controls.Add(Me.ListBox1)
    End Sub


    ' La Sub DrawItem affiche un Item
    Private Sub ListBox1_DrawItem(ByVal sender As Object, _
        ByVal e As DrawItemEventArgs) Handles ListBox1.DrawItem

        ' L' item est celui 'selected' si And sur 'State' et 'DrawItemState.Selected'= true.
        If (e.State And DrawItemState.Selected = DrawItemState.Selected) Then
            e.Graphics.FillRectangle(Brushes.CornflowerBlue, e.Bounds) ' remplir en bleue
            e.Graphics.DrawString(Me.ListBox1.Items(e.Index), New Font(Me.Font, FontStyle.Bold),
_
            Brushes.Black, e.Bounds.X, e.Bounds.Y) 'Afficher le texte et le mettre en gras
        Else
            ' Si non selected mettre en beige et pas en gras.
            e.Graphics.FillRectangle(Brushes.Beige, e.Bounds)
            e.Graphics.DrawString(Me.ListBox1.Items(e.Index), Me.Font, Brushes.Black, e.Bounds.X,
e.Bounds.Y)

        End If

        ' Rectangle bleu autour de chaque Item.
        e.Graphics.DrawRectangle(Pens.Blue, New Rectangle(e.Bounds.X, e.Bounds.Y,
e.Bounds.Width - 1, e.Bounds.Height))
        'On diminue la largeur du rectangle de 1 pixel, sinon le coté droit n'est pas visible

        ' Afficher le focus rectangle .
        e.DrawFocusRectangle()
    End Sub

    ' Sub MeasureItem.
    Private Sub ListBox1_MeasureItem(ByVal sender As Object, _
        ByVal e As MeasureItemEventArgs) Handles ListBox1.MeasureItem

        'Hauteur des items
        'On pourrait modifier la hauteur de chaque Item en fonction de ce qu'il contient
        e.ItemHeight = 20

    End Sub


End Class
```




Pour afficher des images dans les Items d'une ListBox voir 'Afficher des images dans un ListView', c'est le même principe.

## VIII-H-2 - Les CheckedListBox

C'est une Listbox, mais avec une case à cocher sur chaque ligne.


Attention : SelectedItems et SelectedIndices ne déterminent pas les éléments qui sont cochés, mais ceux qui sont en surbrillance.

La collection CheckedItems vous donne par contre les éléments cochés. La méthode GetItemChecked (avec comme argument le numéro d'index) détermine si l'élément est coché.

Exemple

Pour déterminer les éléments cochés dans un contrôle CheckedListBox :

tester chaque élément de la collection CheckedItems, en commençant par 0. Notez que cette méthode fournit le numéro que porte l'élément dans la liste des éléments cochés, et non dans la liste globale. Par conséquent, si le premier élément de la liste n'est pas coché alors que le deuxième l'est, le code ci-dessous affiche une chaine du type "Item coché 1 = Dans la liste : 2".

```
If CheckedListBox1.CheckedItems.Count <> 0 Then
   'S'il y a des éléments cochés une boucle balaye les éléments cochés
   '(collection CheckedItems) et affiche le numéro de l'élément DANS LA LISTE toutes lignes.
   Dim x As Integer
   Dim s As String = ""
   For x = 0 To CheckedListBox1.CheckedItems.Count - 1
      s = s & "Item coché " & (x+1).ToString & " = " & "Dans la liste :"&
 CheckedListBox1.CheckedItems(x).ToString _
      & ControlChars.CrLf
   Next x
   MessageBox.Show(s)
End If
```

On rappelle comme toujours que quand on parle du 3e élément cela correspond à l'index 2.

## VIII-H-3 - Les ComboBox

Les listes Combo (Liste combinée) possèdent deux caractéristiques essentielles par rapport aux ListBox.

Elles sont modifiables : c'est-à-dire que l'utilisateur a la possibilité d'entrer un élément qui ne figure pas au départ dans la liste. Cette caractéristique concerne donc les données proprement dites ; cela se traduit par la présence d'une zone de texte en haut de la liste.

Elles peuvent être déroulantes ou déjà déroulées : c'est-à-dire qu'on ne voit qu'un seul élément de la liste à la fois, et qu'il faut cliquer sur la flèche du côté pour "déplier" la liste, ou bien que la liste est déjà visible. C'est la propriété DropDownList qui gère cela.


La combo du bas a sa DropDownList=Simple

L'utilisateur peut donc cliquer dans la liste (ce qui met le texte cliqué dans la zone texte), ou taper un nouveau texte.

Items.Add (méthode) ajoute un élément à une liste.
Items.Clear (méthode) efface tous les éléments d'une liste.
Items.Count (propriété) renvoie le nombre d'éléments d'une liste.
Multiselect (propriété) permet la sélection multiple.
Item.Remove (méthode) supprime un élément de la liste.
Sorted (propriété) trie les éléments d'une liste.

Comment récupérer la zone texte quand elle change ?

Elle est dans la propriété Text.

On utilise l'événement TextChanged qui se déclenche quand le texte est modifié.

```
Private Sub ComboBox1_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs)
_Handles ComboBox1.TextChanged

        Label1.Text = ComboBox1.Text

End Sub
```

On peut 'charger' un combo (grâce à **DataSource**) avec les éléments d'une énumération : **GetValues**, quand on lui donne le type de l'énumération retourne la liste des éléments de l'énumération.

```
'En tête de module.
Enum Nom
  Pierre
  Paul
  Jean
End Enum



 Combo1.DataSource = [Enum].GetValues(GetType(TypeFichier))
```

On peut aussi charger un tableau :

```
Dim t() As String = {"Paul", "Pierre", "Jean"}
ComboBox1.DataSource = t
```

## VIII-H-4 - Le Contrôle ListView

De plus en plus puissant, le contrôle ListView permet d'afficher des listes multicolonnes, ou des listes avec icône ou case à cocher.

La propriété **View** permet de déterminer l'aspect général du contrôle, elle peut prendre les valeurs :

- Details permet une liste avec sous-éléments et titre de colonnes ;

- Liste utilise un ascenseur horizontal ;

- LargeIcon ;

- SmallIcon.

## VIII-H-4-a - ListView détails

Le volet de droite de l'explorateur donne une bonne idée d'une ListView détails.

Ajouter un ListView nommé ListView1.

Par programmation :

```
    ListView1.View= View.Details
```

Cela donne le mode détails (appelé mode Rapport).

Exemple : faire un tableau de 3 colonnes, mettre les nombres de 1 à 100 dans la première, leur carré dans la seconde, leur cube dans la troisième.

1-Comment remplir les entêtes de colonnes ?

En mode conception, dans la fenêtre propriété du ListView, il y a une propriété Columns, le fait de cliquer sur le bouton d'expansion (…) ouvre une fenêtre, cliquer sur 'Ajouter' permet d'ajouter une colonne ; la propriété Text permet de donner un libellé qui apparaitra en haut de la colonne. On peut ainsi nommer les 3 colonnes ("Nombre", "Carré", "Cube" dans notre exemple).

Par programmation on peut aussi créer des colonnes ; cela donne :

```
ListView1.Columns.Add ("Nombre", 60, HorizontalAlignment.Left)

ListView1.Columns.Add ("Carré", 60, HorizontalAlignment.Left)

ListView1.Columns.Add ("Cube", 60, HorizontalAlignment.Left)
```

Une autre manière est de créer une colonne et de l'ajouter :

```
Dim MyCol1 As ColumnHeader = New ColumnHeader

 MyCol1.Text = "Nombre"

 MyCol1.Width = 60

 MyCol1.TextAlign = HorizontalAlignment.Center

 ListView1.Columns.Add(MyCol1)
```

Si on créer 3 columns (MyCol1 , MyCol2, MyCol3), on peut les ajouter en une fois avec AddRange.

```
Dim cols() As ColumnHeader = {MyCol1, MyCol2, MyCol3}

 ListView1.Columns.AddRange(cols)
```

2-Comment remplir le tableau ?

Pour remplir le tableau, on pourrait, sur la ligne Items de la fenêtre des propriétés, cliquer sur … et rentrer les valeurs 'à la main'. On le fait le plus souvent par programmation.

Pour chaque ligne il faut créer un objet ListViewItem :

sa propriété Text contient le texte de la première cellule.

j'ajoute à cet objet des SubItems qui correspondent aux cellules suivantes.

Enfin j'ajoute le ListViewItem au contrôle ListView.

```
Dim i As Integer

For i = 1 To 100

    Dim LVI As New ListViewItem

    LVI.Text = i.ToString                  'première cellule

    LVI.SubItems.Add((i * i).ToString)     'seconde cellule

    LVI.SubItems.Add((i * i * i).ToString) 'troisième cellule

    ListView1.Items.Add(LVI)                 'ajout de la ligne

 Next i
```

Autre manière: ajouter la ligne ' 2 4 8'

```
Dim MyLine As ListViewItem = New ListViewItem("2")

Dim subLine As ListViewItem.ListViewSubItem = New ListViewItem.ListViewSubItem(MyLine, "4")

 MyLine.SubItems.Add(subLine)

 subLine = New ListViewItem.ListViewSubItem(MyLine, "8")

 MyLine.SubItems.Add(subLine)

 ListView1.Items.Add(MyLine)


 'Autre syntaxe
 Dim MyLine As ListViewItem= New ListViewItem( New String() {"Lasserre","Philippe", "1951"})
 ListView1.Items.Add (MyLine)
```

**Autre exemple complet :** 3 colonnes (Nom, Prénom, Date de naissance), ajouter une ligne.

```
'Le ListView1 existe
 ListView1.View= View.Details

 ListView1.Columns.Add("Nom", 60, HorizontalAlignment.Left)
 ListView1.Columns.Add("Prénom", 60, HorizontalAlignment.Left)
 ListView1.Columns.Add("Année naissance", 60, HorizontalAlignment.Left)

 Dim MyLine As ListViewItem = New ListViewItem(New String() {"Lasserre", "Philippe", "1951"})
 ListView1.Items.Add(MyLine)
```


On pourrait ajouter une image sur la ligne et même la couleur d'avant-plan, d'arrière-plan et la Font de la ligne :

```
Dim MyLine As ListViewItem = New ListViewItem(New String() {"Lasserre", "Philippe", "1951"}, 1,
 Color.Cyan, Color.Beige, Me.Font)
 ListView1.Items.Add(MyLine)
```

Pour l'image, on donne en argument l'index dans l'ImageList associé au ListView.

3-Comment relire une cellule, réécrire dans une cellule ?

Voyons d'abord comment on peut localiser une cellule :

| | | |
|---|---|---|
| ListView1.Items(0).Subiten | ListView1.Items(0).Subiten | ListView1.Items(0).Subitems(2).text |
| ListView1.Items(1).Subiten | ListView1.Items(1).Subiten | ListView1.Items(1).Subitems(2).text |
| … | … | … |

Les lignes sont contenues dans la collection Items.

La première ligne est ListView1.Items(0), la seconde ligne : ListView1.Items(1)…

Les cellules sont contenues dans la collection Items().SubItems.

La première cellule est ListView1.Items(0).SubItems(0) ou ListView1.Items(0).Text , la seconde ListView1.Items(0).SubItems(1).

Pour lire le texte de la seconde ligne seconde colonne :

Texte=ListView1.Items(1).SubItems(1).

Pour écrire dans une cellule :

```
ListView1.Items(1).SubItems(1).Text = "88"
```

De même pour les couleurs :

```
'Mettre le fond d'une case en rouge
ListView1.Items(2).UseItemStyleForSubItems= False
ListView1.Items(2).SubItem (3).BackColor= Colors.Red

'Mettre une ligne en vert
ListView1.Items(2).BackColor= Colors.Lime
```

Comment intercepter le numéro de la ligne qui a été cliquée par l'utilisateur (et l'afficher) ?

```
Private Sub Listview1_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
 Listview1.Click

        Label1.Text = Listview1.SelectedIndices(0).ToString
```

```
End Sub
```

Si la propriété MultiSelect est à False il y a, bien sûr, une seule ligne sélectionnée, sinon les lignes sélectionnées sont dans la collection SelectedIndices().

Si on veut récupérer le texte de la ligne sélectionnée, il faut utiliser :

```
ListView1.SelectedItems(0)
```

Comment effacer la ligne sélectionnée :

```
ListView1.Items.RemoveAt (ListView1.SelectedIndices(0))
'ou
ListView1.Items.Remove (ListView1.SelectedItems(0))
```

Si la propriété **GridLine** est à True, des lignes matérialisant les cases apparaissent.

Si la propriété **CheckedBox** est à True, des cases à cocher apparaissent.

**ListView1.LabelEdit** = True autorise la modification d'une cellule de la première colonne (pour modifier, il faut cliquer une fois (sélection) puis une seconde fois (modification). Pour les colonnes suivantes, il n'y a pas d'édit.

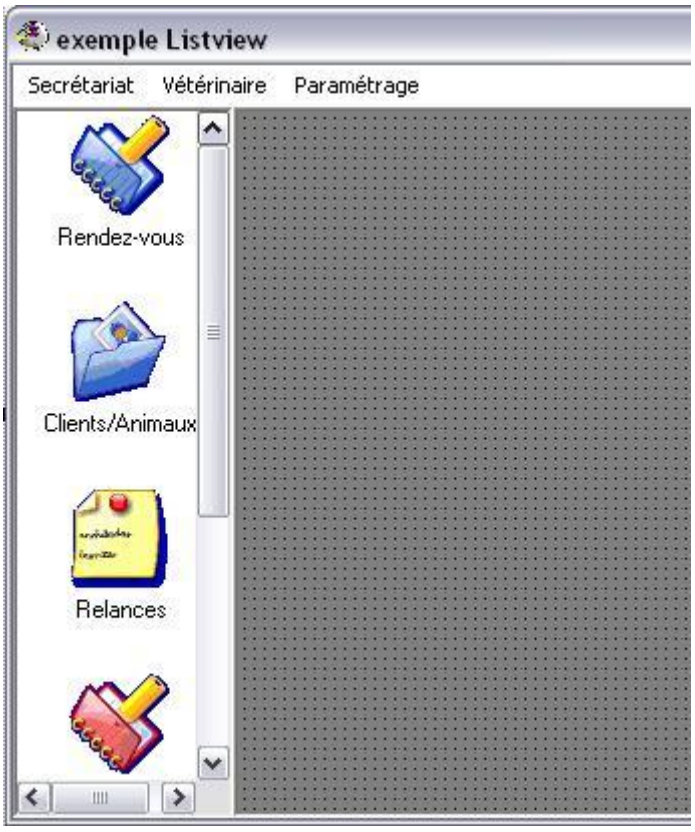
Attention : si la somme des colonnes est plus large que le contrôle, un ascenseur horizontal apparait !!

Pour ne pas voir cet ascenseur, ruser sur la largeur des colonnes (c'est le 2e paramètre de la méthode .Columns.Add)

Afficher des images dans les SubItems en VB 2005: voir annexe en bas de page

## VIII-H-4-b - Liste d'icônes

Permet de faire un menu d'icônes à gauche d'un formulaire, du plus bel effet :




La propriété View permet de déterminer l'aspect général du contrôle, ici on la met à la valeur LargeIcon.

Voyons certaines propriétés du Listview (nommé menu_graphique) :

```
With menu_graphique
 Dock= Left    'cela le 'colle' sur le rebord gauche.

 MultiSelected= False    'on n'utilise qu'un bouton à la fois.

 Activation= OnClick     'les icônes marchent comme des boutons

 LabelEdit= False        'impossible de modifier le texte sous les icônes

 LargeIconeList= NOMdeIMAGELIST
End With
```

En effet, il faut créer un contrôle ImageList y mettre les images du menu et indiquer au Listview le nom de l'ImageList dans la propriété LargeIconeList.

Dans la collection Items du Listview (fenêtre des propriétés), dans chaque élément, ImageIndex donne le numéro de l'image du imageList à mettre dans l'élément (ainsi que le texte qui est sous l'image).

Quand l'utilisateur clique sur une image, la procédure suivante se déclenche :

```
Private Sub menu_graphique_Click(ByVal sender As Object, ByVal e As System.EventArgs)
_Handles menu_graphique.Clic



Select Case menu_graphique.SelectedItems(0).Text

    Case "Rendez-vous"

    …
```

```
    Case "Quitter"

        Application.Exit()

End Select

End Sub
```

On voit que menu_graphique.SelectedItems(0).Text permet de savoir sur quel bouton l'utilisateur a cliqué et ainsi d'appeler la procédure correspondant au choix de l'utilisateur.

## VIII-H-5 - Le contrôle DomainUpDown

Le contrôle DomainUpDown permet d'afficher une liste occupant peu de place : on ne voit qu'une ligne, on se déplace avec les boutons up et down:


On charge la liste avec :

MondomainUpDown.Items.Add("une ligne")

Quand l'utilisateur change de ligne, cela déclenche SelectedItemChanged. Le texte sélectionné est dans SelectedItem.

La sub suivante affiche dans une messageBox l'index et le texte sélectionné.

```
Private Sub MondomainUpDown1_SelectedItemChanged _ (sender As System.Object, e As
 System.EventArgs)

MessageBox.Show(("Index sélectionné: " & MondomainUpDown1.SelectedIndex.ToString() & _
ControlChars.Cr & "Item sélectionné: " & MondomainUpDown1.SelectedItem.ToString()))

End Sub
```

Attention la liste contient des objets, il peut être nécessaire lorsqu'on utilise un des items de caster l'objet en string grâce à ToString.

Il existe aussi un contrôle NumericUpDown.

## VIII-H-6 - Le Contrôle TreeView

Le contrôle TreeView permet d'afficher des listes 'arborescentes' avec des nœuds.


Un arbre (Tree) est composé de nœuds (nodes) qui sont des objets.

Chaque nœud est composé d'une Image et d'un Text.

Les nœuds du premier niveau sont dans la collection Nodes du TreeView, ils sont repérés par un index unique.

MyTree.Nodes(0) est le premier nœud.

Avec l'image ci-dessus :

MyTree.Nodes(0).Text = "Paul" 'collection dont le premier élément est zéro

MyTree.Nodes(1).Text = "Luc" 'c'est bien le second élément du premier niveau

MyTexte= MyTree.Nodes(2).Text retourne une erreur: la collection ne comporte que les nœuds du premier niveau.

Chaque nœud a un parent (au-dessus), des nœuds enfants (au-dessous).

```
MyTree.Nodes(10).Children 'retourne le nombre d'enfants
```

Pour voir les enfants, il y a plusieurs méthodes :

```
MyTree.Nodes(0).Nodes(0).Text = "Odile" ' on utilise la collection Nodes du noeud Nodes(0)
MyTree.Nodes(0).FirstNode.Text = "Odile" ' on utilise FirstNode qui donne le premier enfant.
```

Il existe aussi LastNode.

NextNode, PrevNode Parent permettent, par rapport à un nœud (sur une instance de nœud et pas sur nodes()), de voir respectivement le nœud suivant, le précédent au même niveau ou le nœud parent.

MyTree.Nodes(0).FirstNode.NextNode permet dans les enfants de Nodes(0), de voir celui qui est après le premier.

On peut modifier l'image :

```
MyTree.Nodes(7).Image = "closed"
```

Nombre de nœuds :

```
MyTree.GetNodeCount(True) donne le nombre de noeuds (noeuds enfants compris, car l'argument est
 True).
MyTree.GetNodeCount(False) donne les noeuds de premier niveau
```

Pour ajouter des nœuds en mode Design, utiliser la propriété Nodes dans la fenêtre de propriété. Cela donne accès à une fenêtre qui permet de rajouter des nœuds au même niveau ou des nœuds enfants.

On peut aussi ajouter un nœud par code (au niveau de 'Paul' et 'Luc').

À partir d'un nœud, on travaille sur la collection nodes et la méthode Add de cette collection.

```
MyTree.Nodes.Add("Lucienne")
```

On peut ajouter un nœud sous le nœud sélectionné.

```
MyTree.SelectedNode.Nodes.Add("toto")
```

ou en développant:

```
Dim node As System.Windows.Forms.TreeNode
node = MyTree.SelectedNode
node.Nodes.Add("Nouveau noeud sous la sélection")
```

On peut enlever un nœud :

```
MyTree.Nodes.RemoveAt(0)
```

Un nœud peut être expanded or collaped on peut modifier l'état de l'arbre par CollapsedAll ou ExpandedAll, on peut travailler sur les nœuds visibles, voir ou non les '+' et les '-' grâce à la propriété ShowPlusMinus.

Le nœud sélectionné par l'utilisateur est dans SelectedNode

```
Dim node As System.Windows.Forms.TreeNode
node = MyTree.SelectedNode
```

MyTree.SelectedNode.Text retourne 'Tree Node: Paul' si on a cliqué sur le premier nœud.

Quand on est perdu, FullPath donne le chemin du nœud.

Comment savoir si l'utilisateur a sélectionné un nœud, ou double-cliqué sur un nœud ?

On utilise les procédures événements suivantes :

```
Private Sub MyTreeAfterSelect(ByVal sender As System.Object, ByVal e As
 System.Windows.Forms.TreeViewEventArgs)
_Handles MyTree.AfterSelect

End Sub

Private Sub MyTreeDoubleClick(ByVal sender As Object, ByVal e As System.EventArgs)
-Handles MyTree.DoubleClick

TextBox1.Text = TreeView1.SelectedNode.ToString

End Sub
```

Exemple complet : sur MyTree, rentrer les clients et leurs commandes.

(Exemple librement inspiré de Microsoft, un peu complexe.)

```
Dim node As TreeNode
node = MyTree.Nodes.Add("Noeud de niveau 1")
```

```
node.Nodes.Add("noeud de niveau 2")


' Créer une ArrayList pour Customer objects.
Private MyClientArray As New ArrayList()

Private Sub FillMyTreeView()
   ' On ajoute des éléments à l'ArrayList.
   Dim x As Integer
   For x = 0 To 999
      MyClientArray.Add(New MyClient("MyClient" + x.ToString()))
   Next x

   ' On ajoute des commandes (order) pour chaque client dans ArrayList.
   Dim MyClient1 As MyClient
   For Each MyClient1 In MyClientArray
      Dim y As Integer
      For y = 0 To 14
         MyClient1.MyClientOrders.Add(New Order("Order" + y.ToString()))
      Next y
   Next MyClient1

   ' on met d'un curseur d'attente.
   Cursor.Current = New Cursor("MyWait.cur")

   ' on gèle la mise à jour du contrôle.
   MyTree.BeginUpdate()

   ' on efface le précédent contenu.
   MyTree.Nodes.Clear()

   ' On ajoute des root TreeNode pour chaque MyClient object.
   Dim MyClient2 As MyClient
   For Each MyClient2 In MyClientArray
      MyTree.Nodes.Add(New TreeNode(MyClient2.MyClientName))

      ' on ajoute des TreeNode enfants pour chaque commande (Order) object .
      Dim order1 As Order
      For Each order1 In MyClient2.MyClientOrders
         MyTree.Nodes(MyClientArray.IndexOf(MyClient2)).Nodes.Add( _
    New TreeNode(MyClient2.MyClientName + "." + order1.OrderID))
      Next order1
   Next MyClient2

   ' On permet la mise à jour.
   MyTree.EndUpdate()

   ' On remet le curseur normal.
   Cursor.Current = System.Windows.Forms.Cursors.Default

End Sub
```

Exemple plus simple :

```
        'Il existe un contrôle Treview1 dans le formulaire

        Dim ImageList1 As New ImageList

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

        Me.TreeView1.ImageList = 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("205")
            .Nodes.Add("309")
        End With
        With noeud2
            .Text = "Renault"
            .ImageIndex = 1
            .Nodes.Add("Mégane")
            .Nodes.Add("Clio")
            .Nodes.Add("Laguna")
        End With

        With Me.TreeView1
            .Nodes.Add(noeud1)
            .Nodes.Add(noeud2)
        End With
    End Sub

Me.Controls.Add( TreeView1)
```

## VIII-H-7 - Annexe : Afficher des images dans un ListView

Débutant s'abstenir.

De plus en plus puissant: on a vu que dans le mode 'détails' d'un ListView, on avait des colonnes de SubItems. Comment afficher des images dans ces cellules ?

Il faut mettre la propriété du ListView OwnerDraw à True, cela permet au programmeur d'utiliser la Sub DrawSubItem qui se déclenche quand le subitem doit être dessiné et de dessiner soi-même dans la cellule en utilisant l'argument e qui correspond au bitmap de la cellule.

Dans cette sub DrawSubItem, par exemple, au lieu d'afficher un + dans la première colonne , j'affiche une image (avec e.Graphics.DrawImage). Si par contre je veux laisser s'afficher le texte normal, je dois écrire e.DrawDefault = True.

```
Private Sub ListView1_DrawSubItem(ByVal sender As Object, ByVal e As
 System.Windows.Forms.DrawListViewSubItemEventArgs) _
Handles ListView1.DrawSubItem

If e.ColumnIndex = 1 Then

If e.SubItem.Text = "+" Then

    e.Graphics.DrawImage(Image.FromFile("c:\b.bmp"), New Point(e.Bounds.X, e.Bounds.Y))

Else

    e.DrawDefault = True

End If

else

e.DrawDefault = True

end if

End Sub
```

Ne pas oublier d'afficher normalement les têtes de colonnes.

```
Private Sub ListView1_DrawColumnHeader(ByVal sender As Object, ByVal e  _
As System.Windows.Forms.DrawListViewColumnHeaderEventArgs) Handles ListView1.DrawColumnHeader

e.DrawDefault = True
```

```
End Sub
```

## VIII-I - Fenêtres toutes prêtes (MessageBox…)

Il existe :

- les MessageBox et MsgBox ;

- les InputBox ;

et les autres…

Ces fenêtres toutes faites facilitent le travail.

### VIII-I-1 - MessageBox du Framework

Ouvre une fenêtre qui affiche un message.

C'est une fonction qui affiche un message dans une boite de dialogue, attend que l'utilisateur clique sur un bouton (OK ou Oui-Non…), puis retourne si on le désire, le nom du bouton cliqué par l'utilisateur.

On utilise la méthode **Show** pour afficher la boite.

On doit fournir le texte à afficher, on peut aussi fournir le titre dans la barre, le type de bouton, le type d'icône et le bouton par défaut, une option, la présence d'un bouton d'aide.

Syntaxe :

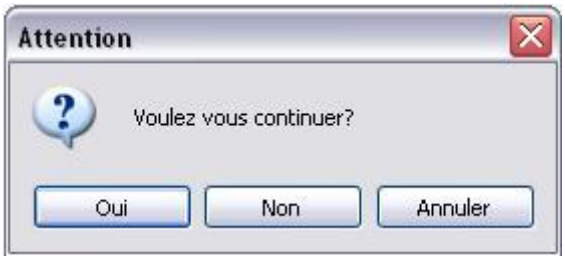
```
MessageBox.show(TexteAAfficher) ' Affiche une box avec le message et un bouton 'OK',
'pas de valeur de retour.
```

ou

```
Dim    Reponse As Integer = MessageBox.show(Texte,Titre, TypeBouton , Icone,
 BoutonParDéfaut, Option, Bouton aide)
```

Le bouton cliqué par l'utilisateur est retourné dans Reponse (de type DialogResult.Ok).

Exemple :




**Paramètres**

**TexteAAfficher**

Obligatoire. Expression String affichée comme message de la boite de dialogue (longueur maximale 1 024 caractères). N'oubliez pas d'insérer un retour chariot si le texte est long, cela crée 2 lignes.

**Titre**

Expression String affichée dans la barre de titre de la boite de dialogue. Si l'argument Titre est omis, le nom de l'application est placé dans la barre de titre.

**TypeBouton**

Expression numérique qui représente la somme des valeurs spécifiant

- le nombre et le type de boutons à afficher :

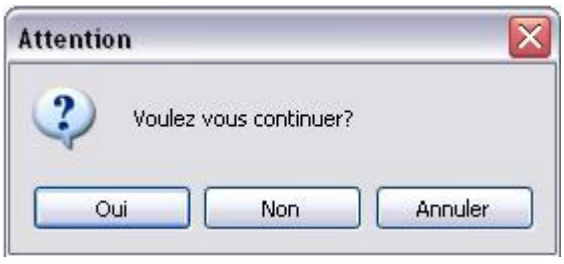
MessageBoxButtons.OKOnly Un seul bouton 'OK' ;

MessageBoxButtons.YesNo Deux boutons 'Oui' 'Non' ;

MessageBoxButtons.OkCancel 'OK' et 'Annuler' ;

MessageBoxButtons.AbortRetryIgnore 'Annule' 'Recommence' 'Ignore' ;

MessageBoxButtons.YesNoCancel, exemple :



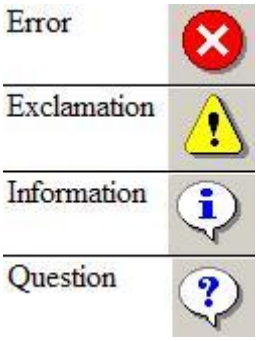

**Icons**

-le style d'icône à utiliser :

MessageBoxIcon.Error ;

MessageBoxIcon.Exclamation.

…




**L'identité du bouton par défaut**

MessageBoxDefaultButton.Button1.

MessageBoxDefaultButton.Button2.

Les options

MessageBoxOptions.RightAlign

**L'affiche d'un bouton d'aide**

True pour l'afficher.

Comme d'habitude, il suffit de taper MessageBox.Show( pour que VB proposer les paramètres.

**Retour de la fonction**

Retourne une constante de type DialogResult qui indique quel bouton a été pressé.

```
DialogResult.Yes

DialogResult.No

DialogResult.Cancel

DialogResult.Retry

DialogResult.OK
```

Exemple 1

Afficher un simple message :

```
MessageBox.Show("bonjour")
```

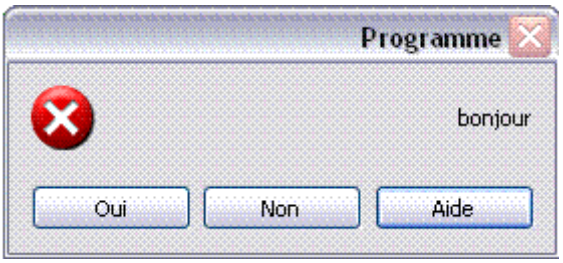
Affiche 

Exemple 2

Afficher les boutons Oui Non et un bouton d'aide, une icône d'erreur, tester si l'utilisateur a cliqué sur Oui :

```
Dim r As DialogResult  (cela marche aussi avec Integer)

r = MessageBox.Show("bonjour", "Programme", MessageBoxButtons.YesNo, MessageBoxIcon.Error,
_ MessageBoxDefaultButton.Button1, MessageBoxOptions.RightAlign, True)

If r = Windows.Forms.DialogResult.Yes Then

…

End If
```

Affiche :




Écriture plus compacte :

```
If MessageBox.Show("bonjour", "Programme", MessageBoxButtons.YesNo) = DialogResult.Yes then

…….

End If
```

## VIII-I-2 - MsgBox du Visual Basic

L'ancienne syntaxe Visual Basic avec MsgBox est conservée. (À éviter, préférer MessageBox.)

MessageBox.Show est du Framework Net, MsgBox est du VB.

```
Reponse= MsgBox(TexteAAfficher, Style, Titre)
```

Dans ce cas, il faut utiliser MsgBoxStyle et MsgBoxResult pour le retour. De plus les arguments ne sont pas dans le même ordre !!

- Pour le choix des boutons, MsgBoxStyle peut prendre les valeurs :

YesNo (cela affiche 2 boutons "Oui" et "Non"), OkCancel, AbortRetryIgnore…

Pour le choix du bouton par défaut MsgBoxStyle peut prendre les valeurs :

DefaultButton1, DefaultButton2…

Pour les icônes MsgBoxStyle peut prendre les valeurs :

| | | |
|---|---|---|
| Critical | 16 |  |
| Question | 32 |  |
| Exclamation | 48 |  |
| Information | 64 |  |

Il faut ajouter les styles les uns aux autres avec 'Or'.

Au retour on a les valeurs :

MsgBoxResult.Yes, MsgBoxResult.No, MsgBoxResult.Cancel…

Exemple simple fournissant un simple message à l'utilisateur :

MsgBox ("Bonjour") affiche une message box avec le message 'Bonjour', un bouton 'OK' et dans la barre de titre, le nom de l'application. Il n'y a pas de valeur de retour.

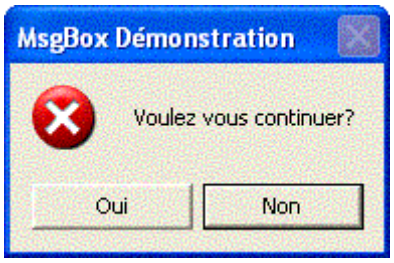
Exemple courant posant une question, l'utilisateur doit cliquer sur 'oui' ou 'non', on teste si c'est 'oui' :

```
If MsgBox("D'accord?", MsgBoxStyle.YesNo) = MsgBoxResult.Yes Then

End If
```

Exemple complet :

```
Dim msg As String

Dim title As String

Dim style As MsgBoxStyle

Dim response As MsgBoxResult

msg = "Voulez-vous continuer?"               ' Définition du message à afficher.

style = MsgBoxStyle.DefaultButton2 Or _

   MsgBoxStyle.Critical Or MsgBoxStyle.YesNo 'On affiche Oui Non

title = "MsgBox Démonstration"               ' Définition du  titre.

' Affiche la boite MsgBox.


response = MsgBox(msg, style, title)


If response = MsgBoxResult.Yes Then

   ' code si l'utilisateur a cliqué sur Oui

Else

   ' code si l'utilisateur a cliqué sur Non.

End If
```

Voilà ce que cela donne :




'On remarque que dans l'exemple, on crée des variables dans lesquelles on met le texte ou les constantes adéquates, avant d'appeler la fonction MsgBox. En condensé cela donne :

```
If MsgBox("Voulez-vous continuer ?", MsgBoxStyle.DefaultButton2 Or MsgBoxStyle.Critical Or
 MsgBoxStyle.YesNo,  _
"MsgBox Démonstration")= MsgBoxResult.Yes Then

End If
```

## VIII-I-3 - InputBox

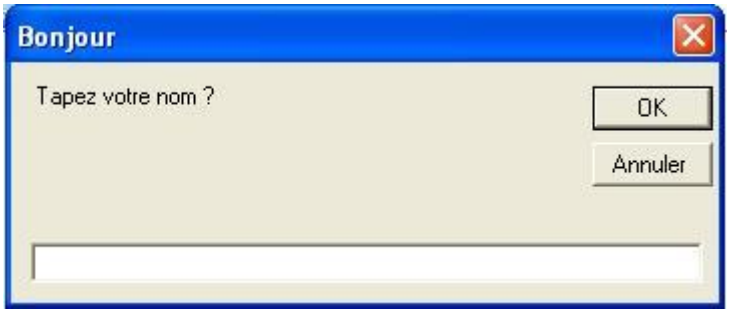
C'est une fonction qui permet d'ouvrir une fenêtre qui pose une question.

Elle retourne la réponse tapée par l'utilisateur.

Le retour est effectué dans une variable String.

```
Dim Nom As String

Nom = InputBox("Bonjour","Tapez votre nom ?")
```

Cela donne :




On pourrait rajouter un 3e argument=la réponse par défaut.

Si l'utilisateur clique sur le bouton 'Annuler', une chaine vide est retournée.

On a souvent besoin de contrôler si l'utilisateur a tapé quelque chose puis d'enlever les espaces :

```
Dim reponse As String 'on crée une variable qui contiendra la chaine tapée par l'utilisateur.

Do

    reponse= InputBox("Tapez un nom") 'saisir une chaine de caractères dans une InputBox

Loop Until String.IsNullOrEmpty(reponse)'si l'utilisateur n'a rien tapé, on boucle et on
 réaffiche l'inputbox
```

```
reponse=reponse.Trim(" ")     'enlève les espaces avant et après
```

## VIII-I-4 - OpenFileDialog

Comment afficher une boite de dialogue permettant de sélectionner un fichier (ou des fichiers) à ouvrir par exemple ?

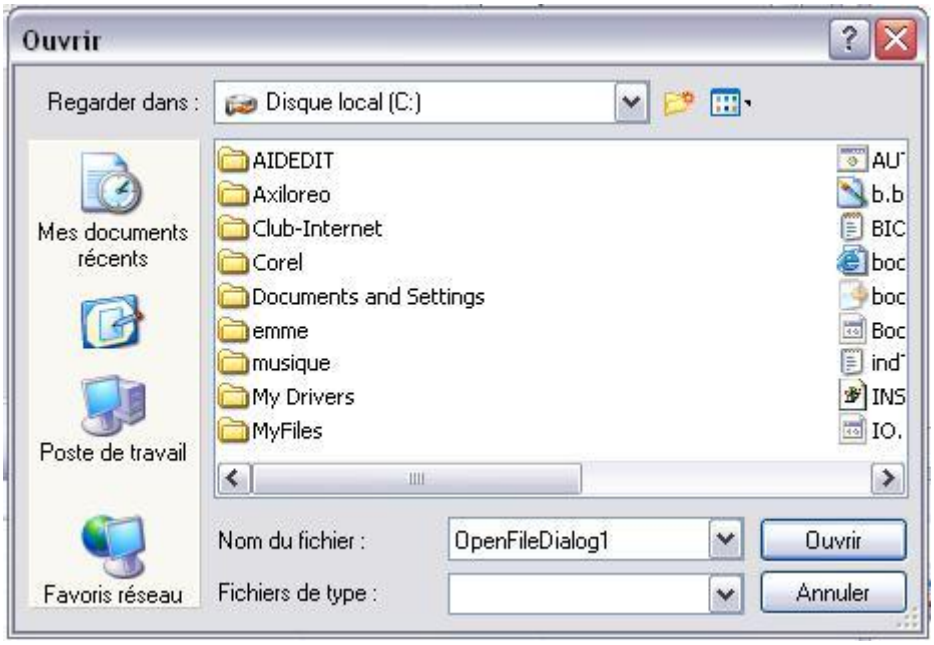
Dans la boite à Outils, cliquez sur OpenFileDialog puis cliquez sur la fenêtre en cours : un contrôle OpenFileDialog1 apparait sous la fenêtre.

Ouvre une boite de dialogue permettant de choisir un nom et un chemin de fichier, au programmeur d'écrire le code lisant les fichiers.

Dans le code, à l'endroit où doit s'ouvrir la fenêtre (dans l'événement Click d'un bouton nommé 'Ouvrir' par exemple), tapez :

```
Private Sub ButtonOuvrir_Click()

    OpenFileDialog1.ShowDialog()

End Sub
```

C'est suffisant pour créer une fenêtre montrant l'arborescence des fichiers et répertoires et pour que l'utilisateur choisisse un fichier :




Mais le plus souvent on a besoin que la boite de dialogue propose un type de fichier et un répertoire précis.

Par exemple, je veux ouvrir un fichier .TXT dans le répertoire c:\MesTextes.

Il faut dans ce cas, AVANT le ShowDialog renseigner certaines propriétés du contrôle OpenFileDialog1 :

```
With OpenFileDialog1

    .Title= "Ouvrir"         'Titre de la barre de titre

    .InitialDirectory = "c:\"    'répertoire de départ
```

```
    .Filter="Fichiers txt|*.txt" ' on travaille uniquement sur les .txt
                                's'il y a plusieurs filtres les séparer par ;
                                  FilterIndex  indique le filtre en cours
    .Multiselect=False          'sélectionner 1 seul fichier
    .CheckFileExists=True       'Message  si nom de fichier qui n'existe pas.
                                'Permet d'ouvrir uniquement un fichier qui existe; CheckPathExists
 peut aussi être utilisé.
    .ValidateNames=True          'n'accepte que les noms valides (win 32)
    .AddExtension=True          'ajoute une extension au nom s'il n'y en a pas
End With
```

Comment afficher la boite et vérifier si l'utilisateur a cliqué sur 'Ouvrir' ?

La méthode .ShowDialog peut retourner un élément de l'énumération DialogResult pour indiquer l'action de l'utilisateur sur la boite de dialogue :

```
DialogResult.OK    'si l'utilisateur a cliqué sur 'Ouvrir'
DialogResult.Cancel   'si l'utilisateur a cliqué sur 'Annuler'
```

Comment utiliser cela ?

```
If OpenFileDialog1.ShowDialog= DialogResult.Ok Then  'L'utilisateur a bien cliqué sur OK
End if
```

Maintenant, OpenFileDialog1.FileName contient le nom du fichier sélectionné (avec extension et chemin).

```
Path.GetFileName(OpenFileDialog1.FileName) 'donne le nom du fichier sans chemin.
```

En conclusion OpenFileDialog permet de sélectionner un nom de fichier en vue d'une ouverture, à vous ensuite d'ouvrir et de lire le fichier.

## VIII-I-5 - SaveFileDialog

Boite de dialogue fonctionnant de la même manière que OpenFileDialog, mais avec quelques propriétés spécifiques.

Ouvre une boite de dialogue permettant à l'utilisateur de choisir un nom et un chemin de fichier, au programmeur d'écrire le code enregistrant les fichiers.

```
SaveFileDialo1.CreatePrompt= True    ' Message de confirmation si
                                     'création d'un nouveau fichier
SaveFileDialo1.OverwritePrompt=True  'Message si le fichier existe déjà
                                     'évite l'effacement d'anciennes données
SaveFileDialo1.DefaultExt="txt"      'extension par défaut
```

On récupère aussi dans .FileName le nom du fichier si la propriété .ShowDialog a retourné DialogResult.Ok.

## VIII-I-6 - FolderBrowserDialog

Boite de dialogue 'Choix de répertoire' en VB2005

Il faut instancier un **FolderBrowserDialog**, indiquer le répertoire de départ (RootFolder), le texte de la barre (Description) et l'ouvrir avec ShowDialog.

Le répertoire sélectionné par l'utilisateur se trouve dans **SelectedPath**.

```
Dim fB As New FolderBrowserDialog

fB.RootFolder = Environment.SpecialFolder.Desktop

fB.Description = "Sélectionnez un répertoire"

fB.ShowDialog()

If fB.SelectedPath = String.Empty Then

  MsgBox("Pas de sélection")

Else

  MsgBox(fB.SelectedPath)

End If

fB.Dispose()
```

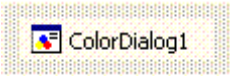

## VIII-I-7 - FontDialog

**Pour que l'utilisateur choisisse une police de caractères et modifie la police d'un contrôle List1.**

```
Dim myFontDialog As FontDialog
myFontDialog = New FontDialog()

   If myFontDialog.ShowDialog() = DialogResult.OK Then

      List1.Font = myFontDialog.Font
   End If
```

## VIII-I-8 - ColorDialog

Pour permettre à l'utilisateur de choisir une couleur, il faut mettre dans le formulaire une ColorDialog à partir de la boite à outils ; elle vient se placer sous le formulaire :




Il faut ensuite, par code, ouvrir cette ColorDialog.

La classe ColorDialog a une méthode ShowDialog, analogue à la méthode ShowDialog des classes OpenFileDialog et


Si l'utilisateur quitte la boite de dialogue en cliquant sur le bouton 'OK', la méthode ShowDialog retourne DialogResult.OK et la couleur choisie est dans la propriété Color de l'objet ColorDialog .

Exemple

L'utilisateur clique sur un bouton nommé 'CouleurButton' cela ouvre la ColorDialog, l'utilisateur clique sur une couleur puis sur 'OK', cela donne aux caractères de la TextBox 'Texte' la couleur choisie.

```
Private Sub CouleurButton_Click(ByVal sender As Object, ByVal e As System.EventArgs) _

Handles CouleurButton.Clic

' Ouverture de la dialogBox colordialog1

If colorDialog1.ShowDialog() = DialogResult.OK Then

' on met la couleur dans la propriété forecolor du TextBox

Texte.ForeColor = colorDialog1.Color

End If

End Sub
```

On peut modifier la boite à notre goût avant de l'ouvrir :

```
colorDialog1.SolidColorOnly= True  'Couleurs pures (unies) seulement
colorDialog1.AllowFullOpen= True  'Autorise le bouton des couleurs personnalisées (volet de
 droite)
colorDialog1.FullOpen= True  'Affiche les couleurs personnalisées
colorDialog1.Color= Color.Red  'Couleur présélectionnée
```

## VIII-I-9 - Créer une boite 'de dialogue' ou 'À propos de'

Si on a besoin dans un programme d'une boite de dialogue spécifique, il faut la créer soi-même dans un nouveau formulaire.
Elle doit respecter certaines règles.

Elle doit être 'modale' (le seul moyen d'en sortir est de la fermer) donc ouverte avec ShowDialog.
Elle est habituellement non redimensionnable (FormBorderStyle= FixedDialog).
Elle n'a pas de boutons dans la barre de titre (ControlBox, MinimizeBox, MaximizeBox= False).
La boite ne doit pas apparaitre dans la barre de tâches (ShowInTaskBar =False).
Il faut ajouter à la main la ou les zones permettant de saisir les informations demandées.
Elle doit contenir un bouton 'OK' et un bouton 'Annuler' (ou Oui/Non. Abort, Retry, Ignore).
Il faut générer ce qui se passe quand on clique sur ces boutons. Pour le bouton 'OK' il faut que cela ferme la fenêtre et retourne 'OK' à la fenêtre précédente.

1 - Si vous créez une boite de dialogue vous-même (nommée Form2), il faut créer un bouton dont le texte est 'OK' ; pour ce bouton mettre la propriété DialogResult= OK (dans la fenêtre de propriétés) ou par code :

```
Button1.DialogResult = DialogResult.OK
```

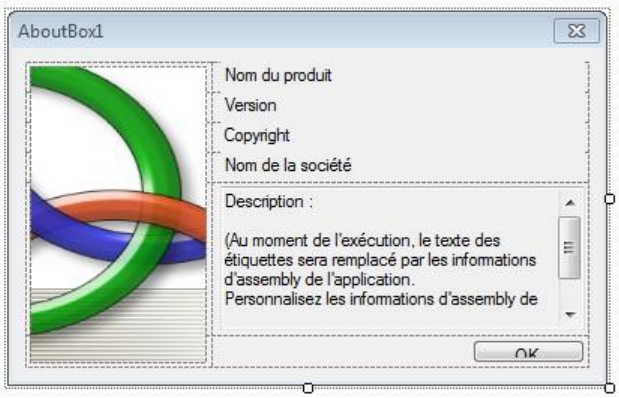
Dans ce cas, si l'utilisateur clique sur le bouton 'OK', la fenêtre modale est fermée et on retourne OK.
Voici le code qui ouvre cette fenêtre de dialogue :

```
Dim f2 As New Form2
        Dim Dia As DialogResult = f2.ShowDialog
        MsgBox(Dia.ToString) 'Affiche 'OK'
```

On peut de cette manière ajouter des boutons Cancel, Yes, No, Abort, Retry, Ignore…

Enfin si on veut pouvoir remplir ou récupérer le texte tapé dans un TextBox de dialogue, le plus simple est que le bouton soit 'Public' et non Private, ainsi il sera accessible à partir d'un autre formulaire.

2 - Vb aide à faire automatiquement et rapidement une DialogBox ou une AboutBox.
Exemple en vb 2010: Menu 'Projet' puis 'Ajouter un formulaire Windows'.
Puis cliquer sur 'Boite de dialogue' ou 'Boite à propos de' :


La boite 'À propos de' donne des informations sur le programme ; un bouton 'OK' permet simplement de la fermer.
La boite de dialogue utilise 'DialogResult' pour communiquer avec le formulaire qui ouvre la boite de dialogue :

```
'On ouvre le formulaire Dialog1
Dialog1.ShowDialog()

If Dialog1.DialogResult= DialogResult.Ok then

    'l'utilisateur a cliqué sur le bouton OK

End if
```

## VIII-J - Regroupement de contrôles 'Groupe de contrôles'

On peut regrouper des contrôles dans :

les GroupBox ;

les Panels ;

les PictureBox ;

les TabControl ;

les SplitContainer.(Framework 2 Vb2005) ;

les LayoutPanel.(Framework 2 Vb2005).

Comment se passer des groupes de contrôles de VB6 qui n'existent plus en VB.Net ?

Comment plusieurs contrôles peuvent-ils déclencher un même événement ?

Comment travailler sur plusieurs contrôles identiques ?

### VIII-J-1 - GroupBox et Panel

Il est possible de regrouper des contrôles dans un **container**, on peut par exemple regrouper plusieurs RadioButton.
Le container peut être un GroupBox ou un Panel.


Ci-dessus on a un **GroupBox** et un **Panel** avec AutoScroll =True et BorderStyle=Single

Pour l'utilisateur, le fait que toutes les options soient regroupées dans un panneau est un indice visuel logique (Tous les RadioButton permettront un choix dans une même catégorie de données). Au moment de la conception, tous les contrôles peuvent être déplacés facilement ; si vous déplacez le contrôle GroupBox ou Panel, tous les contrôles qu'il contient sont également déplacés. Les contrôles regroupés dans un panneau ou un GroupBox sont accessibles au moyen de la propriété Controls du panneau.

Le contrôle Panel est similaire au contrôle GroupBox; mais, **seul le contrôle Panel peut disposer de barres de défilement** et **seul le contrôle GroupBox peut afficher une légende**.

La légende de la GroupBox est définie par la propriété Text.

Pour faire apparaitre les barres de défilement dans le Pannel mettre AutoScroll =True et AutoScrollMinSize =100.

Dans un Panel, pour afficher des barres de défilement, donner à la propriété AutoScroll la valeur True… La propriété BorderStyle détermine si la zone est entourée d'une bordure invisible (None), d'une simple ligne (FixedSingle) ou d'une ligne ombrée (Fixed3D).

**Comment créer un contrôle Panel ?**

Faites glisser un contrôle Panel de l'onglet Windows Forms de la boite à outils jusqu'à un formulaire.

Ajoutez des contrôles au panneau en les déposant dans le panneau.

Si vous voulez mettre dans le panneau des contrôles existants, sélectionnez-les tous, coupez-les dans le Presse-papiers, sélectionnez le contrôle Panel et collez-les.

## VIII-J-2 - PictureBox

Le contrôle PictureBox peut afficher une image, mais peut aussi servir de conteneur à d'autres contrôles.

Retenons la notion de conteneur qui est le contrôle parent.

## VIII-J-3 - TabControl

Ce contrôle permet de créer des onglets comme dans un classeur, onglets entièrement gérés par VB. Chaque page peut contenir d'autres contrôles.

En mode conception, en passant par la propriété TabPages, on ajoute des onglets dont la propriété Text contient le texte à afficher en haut (ici : Page 1…). Il suffit ensuite de cliquer sur chaque onglet et d'y ajouter les contrôles.


En mode run les onglets fonctionnent automatiquement : cliquez sur Page 2 affiche la page correspondante (et déclenche l'événement Click de cet objet TabPage)…

La propriété **Alignment** permet de mettre les onglets en haut, en bas, à droite, à gauche.

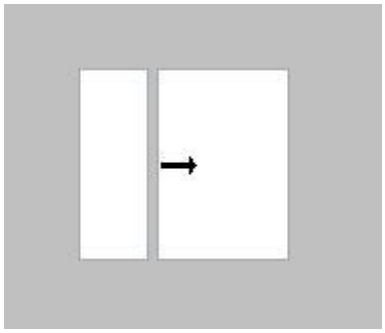
## VIII-J-4 - SplitContainer

À partir de VB 2005 Framework 2.

Permettant de créer facilement **une séparation déplaçable entre 2 zones**.


On met le SplitContainer, dans les 2 zones on met par exemple 2 textbox. Il faut mettre la propriété Dock de ces 2 textbox à Fill.

En mode Run, cela marche : si je déplace la zone de séparation centrale, cela agrandit un textbox et diminue le second.



**Margin** indique la largeur de la séparation.

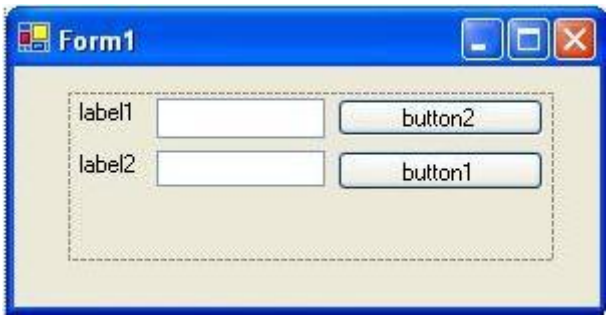
**Orientation** permet une séparation horizontale ou verticale.

## VIII-J-5 - LayoutPanel

À partir de VB 2005 Framework 2.

Permet de positionner les contrôles dans une Form en mode conception. On ne le voit pas en mode Run.

**FlowLayoutPanel :** place les contrôles à droite du précédent, passe 'à la ligne' si nécessaire, c'est génial pour créer plusieurs lignes de label, TextBox, Bouton :




**TableLayoutPanel :** on crée un tableau de panel, puis on met les contrôles dans les cellules :


## VIII-J-6 - Comment remplacer les groupes de contrôles de VB6 qui n'existent plus en VB.Net ?

En VB6 on pouvait créer un groupe de plusieurs contrôles identiques et les repérer par un Index.

Texte(0), Texte(1), Texte(2)…

Pour parcourir le groupe, on avait une boucle For Next sur l'index :

```
For i=1 To 10

Texte(i).text…

Next
```

de plus il n'y avait qu'une routine événement pour l'ensemble du groupe.

```
Sub Texte_Click (Index)
```

Cela n'existe plus en VB.Net !!!!!

Comment donc remplacer un groupe de contrôle en VB.Net ?

### VIII-J-6-a - Événement commun

Exemple : 3 cases à cocher permettent de colorer un label en vert rouge ou bleu. Plutôt que d'écrire une routine pour chaque case à cocher, je voudrais écrire une routine unique.

Comment gérer les événements ?

On peut donc écrire 3 routines complètes, une pour chaque case à cocher.

Il est aussi toujours possible dans chacune des 3 procédures CouleurX.checkedChanged de vérifier si la case est cochée et de modifier la couleur.

C'est plus élégant d'avoir une procédure unique qui, en fonction de la case à cocher qui a déclenché l'événement, change la couleur.

On désire donc parfois que l'événement de plusieurs contrôles différents soit dirigé sur une seule et même procédure.

Par contre par l'intermédiaire du Handles, il est possible d'associer plusieurs événements à une seule procédure :

```
Private Sub CouleurCheckedChanges (ByVal sender As System.Object, ByVal e As System.EventArgs) _

Handles CouleurVert.CheckedChanged, CouleurRouge.CheckedChanged, CouleurBleu.CheckedChanged



End Sub
```

Cette procédure est activée quand les cases à cocher CouleurVert CouleurBleu, CouleurRouge changent d'état.

À noter que Sender est le contrôle ayant déclenché l'événement et e l'événement correspondant.

Pour modifier la couleur, il faut ajouter dans la procédure :

```
Select Case sender

    Case CouleurRouge
    …
End Select
```

ou

```
If sender Is  CouleurRouge Then….
```

Pour savoir quel contrôle a déclenché l'événement commun, on peut aussi mettre dans la propriété 'Tag' de chaque contrôle un numéro (1, 2, 3…) et tester quel numéro est dans le Tag.

```
Private Sub Buttons_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
 Handles Button1.Click, Button2.Click, Button3.Click
        'sender et Tag sont des objets, il faut les transformer en Button et String
        Dim b As String = CType(CType(sender, Button).Tag, String)
        Select Case b
        Case "1"
        '…
        Case "2"
        '……
        End Select
    End Sub
```

## VIII-J-6-b - Comment travailler sur plusieurs contrôles ?

Si j'ai une interface utilisateur avec 20 Textbox, comment modifier la couleur de fond des 20 Textbox ?

(En VB6 on créait un groupe de contrôle, cela n'existe plus en VB.net !! On l'a dit !!)

### Solution 1

```
TextBox1.BackColor= Color.Red

TextBox2.BackColor= Color.Red

TextBox3.BackColor= Color.Red

…….  !!!!!!Bof
```

### Solution 2

Mettre les 20 TextBox dans un Panel (invisible) ; la collection Controls du Panel contient tous les contrôles contenus dans le panel, on utilise cette collection pour atteindre tous les textbox :

```
Dim i As Integer

For i=0 to Panel1.Controls.Count

    Panel1.Controls(i).BackColor=Color.Red

Next
```

Ici il faut que des TextBox dans le Panel, ce qui n'est pas toujours le cas.

Autre solution s'il n'y a que des TextBox :

```
For Each TB As TextBox in Panel1.Controls

    TB.BackColor= Color.Red

Next
```

S'il n'y a pas que des TextBox dans le Panel et qu'on ne veut modifier que les TextBox :

```
For Each TB As Control in Panel1.Controls

    If TypeOf (TB)= TextBox then

        TB.BackColor= Color.Red

    End if

Next
```

### Solution 3

Mettre les 20 TextBox dans un tableau :

```
Dim Textes(8) As Object

Textes(1) = TextBox1

Textes(2) = TextBox2

Dim i As Integer

For i = 1 To 2

Dim MyTexte As String = CType(Textes(i), TextBox).Text

Next
```

Comme c'est un tableau d'Object, pour utiliser la propriété 'Text', on est obligé de caster (convertir) l'élément du tableau en TextBox avant d'utiliser la propriété 'Text'.

C'est plus rusé d'utiliser un tableau de TextBox.

```
                    </paragraph>
Dim Textes(8) As TextBox

'puis dans le form_load
Textes(0) = TextBox0
Textes(1) = TextBox1
Textes(2) = TextBox2
…
'ensuite, on peut bien utiliser la syntaxe de VB 6.0

Dim i As integer
For i = 0 To 8
 Dim MyTexte As Integer= Textes(i).Text
…
Next
```

Noter qu'on a créé un tableau de TexBox ; pas besoin de convertir.

## VIII-K - Dimensions, position des contrôles

On peut dimensionner et positionner les contrôles et formulaires :

en mode conception ;

mais aussi avec du code.

Tous les contrôles héritent donc tous de la classe Windows.Forms.Control.

Les contrôles et formulaires ont tous des propriétés communes.

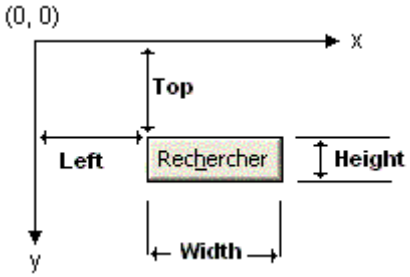
### VIII-K-1 - Unité de mesure

Pixel

L'unité de mesure est le 'Pixel' = Picture Elément (plus de twips comme en vb6).

Les coordonnées d'un contrôle se définissent à partir du coin supérieur gauche du conteneur (coin situé sous la barre de tâches dans le cas du formulaire).

Noter qu'à partir du coin supérieur gauche, l'axe des X va de gauche à droite, l'axe des Y de haut en bas.




### Le Point

Pour définir une paire de coordonnées on utilise un objet **Point** (ou System.Drawing.Point) contenant les coordonnées x et y du point :

```
Dim P As New Point(12,45)     'Ici 12 et 45 sont les coordonnées X et Y du point.
```

On peut utiliser P.x et P.y pour modifier une coordonnée.

### La taille

Pour définir la taille d'un contrôle on utilise un objet **Size** (ou System.Drawing.Size) contenant 2 integers indiquant habituellement largeur et hauteur :

```
Dim S As New Size(12,45)
Button1.Size= S
```

Parfois, on a besoin de définir une série de 4 chiffres (par exemple les 4 marges d'un contrôle) dans ce cas on utilise le type **Padding :**

```
Button1.Padding = New Padding(30, 10, 10, 10)
Button1.Margin = New Padding(30, 3, 30, 30)
```

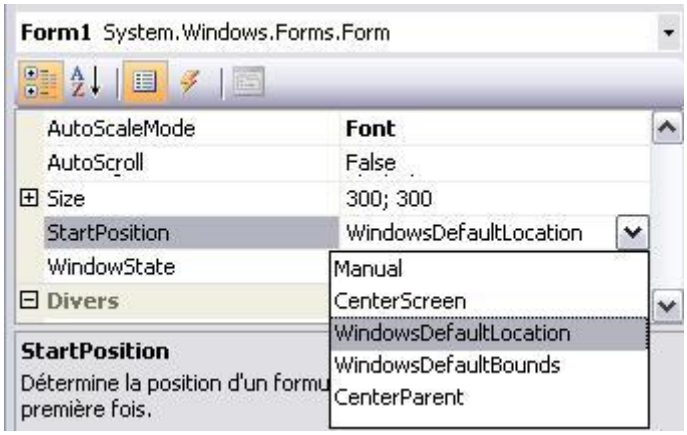
Enfin il existe **Rectangle**, qui contient les coordonnées d'un rectangle (Top, Left, Right, Bottom ou Size, Location) et les méthodes Containts(Point), Inflate (agrandir), Intersects(de 2 rectangles), Offset(déplacement), Union (de 2 rectangles).

```
Dim r As New Rectangle(10, 10, 20, 20)
' x, y, Width, Height
```

## VIII-K-2 - Position initiale dans l'écran d'un formulaire

On peut définir la position initiale, sur l'écran, d'un formulaire grâce à la propriété '**StartPosition**'.

Le formulaire peut apparaitre au centre de l'écran (CenterScreen), au centre du parent (CenterParent) ou à des coordonnées précises (Manual).




De plus la propriété **WindowState** permet de définir la taille du formulaire : normal, plein écran (Maximized) ou réduit dans la barre de tâches (Minimized).

## VIII-K-3 - Taille et position d'un formulaire ou d'un contrôle

On peut utiliser simplement :

**Left**, **Top** coordonnées du coin supérieur gauche et **Bottom**, **Right** inférieur droit ;

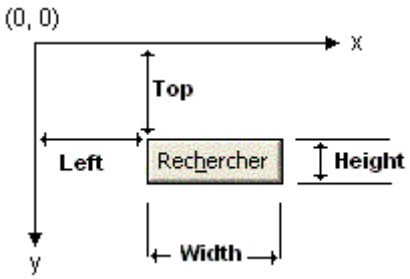
**Height**, **Width** pour la hauteur et la largeur du contrôle en pixels.

On peut utiliser aussi :

**Size :** hauteur, largeur.

**Location :** coordonnées X,Y du coin supérieur gauche du contrôle en pixels.

**SetBounds :** coordonnées X,Y , largeur, hauteur.




Voyons cela en détail.
Position du contrôle :

```
Button.left=188

Button.Top =300
```

Ou

```
Button.Location= New System.Drawing.Point(188,300)
```

Point() contient les coordonnées d'un point dans l'espace.

On remarque qu'il faut donner à la propriété Location un objet Point et non les coordonnées brutes. (En effet, Form1.Location =100, 100 n'est pas accepté.)

```
Form1.Location = New Point(100, 100) est équivalent à:

Form1.left=100: Form1.Top=100
```

Pour définir la taille :

```
Me.Size= New Size(150,150)
'ou
Me.Top= 150
```

On créer un objet Size que l'on affecte à la propriété size du contrôle.

On peut aussi donner la position et les dimensions du contrôle en une fois.

Exemple pour le formulaire courant (Me) :

```
Me.SetBounds (12, 15, 100, 100)    'X,Y, Width, Height.

Form1.DesktopLocation = new Point(100,100) 'Ici on donne les coordonnées par rapport à la barre
 de tâches.
```

*En mode conception il est bien plus simple de dimensionner les contrôles à la main dans la fenêtre Design.*

On peut aussi modifier les valeurs des propriétés dans la fenêtre de propriété, mais en VB 2008, sont uniquement visibles dans la fenêtre des propriétés : Location et Size.

**En conclusion, les contrôles sont positionnés en coordonnées absolues dans le formulaire. S'ils sont dans un contrôle parent (un GroupBox ou un Panel par exemple) les coordonnées se feront par rapport au contrôle parent.**

## VIII-K-4 - Redimensionnement de fenêtre par l'utilisateur

Pour que l'utilisateur puisse redimensionner la fenêtre (en cliquant sur les bords) il faut que la propriété **FormBorderStyle** de la fenêtre soit **Sizable**. (FixedSingle interdit le redimensionnement.)

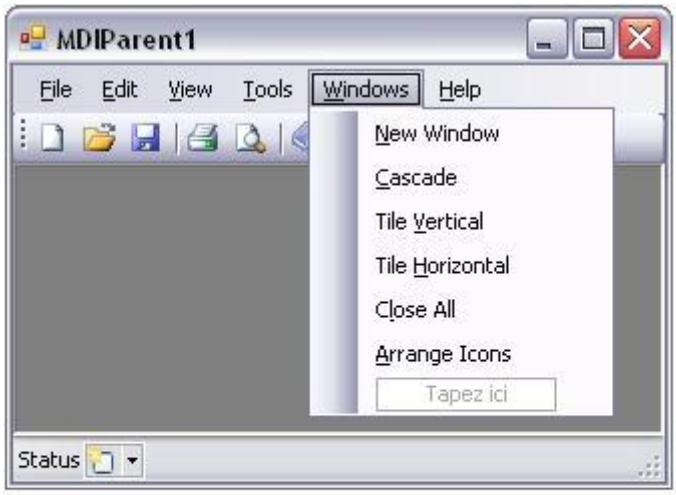
**ControlBox** permet d'afficher la boite de contrôles (bouton d'agrandissement, réduction, fermeture du formulaire) en haut à droite de la barre de tâches.

**MaximizedBox** et **MinimizedBox** permettent d'utiliser les boutons d'agrandissement ou de réduction du formulaire.

Exemple d'un formulaire ayant :

```
ControlBox =True
MinimizedBox =True
MaximizedBox =True
FormBorderStyle= Sizable
```

(les bords de la fenêtre ont 2 traits, ce qui permet le redimensionnement).





Après que l'utilisateur a modifié les dimensions du formulaire, on peut intervenir sur les dimensions du formulaire, pour cela on utilise l'événement **Form.Resize** qui est déclenché quand les dimensions du formulaire sont modifiées par l'utilisateur : dans Form.Resize on peut intervenir sur les dimensions du formulaire ou des contrôles.

Exemple : permettre à l'utilisateur de modifier la hauteur, mais imposer une largeur de formulaire de 200 pixels.

```
Private Sub Form1_Resize()

Me.Width = 200

End Sub
```

Noter que dans Form.Resize on peut récupérer les dimensions du formulaire avec Me.

Les propriétés **MaximmunSize** et **MinimunSize** imposent les dimensions maximales et minimales d'un formulaire ou d'un contrôle, ce qui permet de se passer du code qui précède.

Mais si l'utilisateur modifie la taille du formulaire qui contient les contrôles, la taille des contrôles ne suit pas.

Avant VB.net, il fallait dans l'événement Form_Resize, déclenché par la modification des dimensions de la fenêtre, écrire du code modifiant les dimensions et positions des contrôles afin qu'ils s'adaptent à la nouvelle fenêtre:

Exemple : La largeur d'une TextBox se modifie en fonction de la dimension du formulaire.

```
Private Sub Form1_Resize()

  TextBox.Width = Me.Width-50

End Sub
```

En VB.Net c'est plus facile avec **Dock** et **Anchor** qui 'fixent' le contrôle au conteneur (voir plus bas).

## VIII-K-5 - Déplacement

```
Form1.Left += 200    'déplace le formulaire de 200 pixels
```

```
'déplacement progressif d'un bouton de gauche à droite:

For i As Integer =0 to 100

    Button1.Left= i

Next i
```

## VIII-K-6 - Coordonnées souris

Certains événements relatifs à la souris comme MouseDown (appuyer sur le bouton) MouveUp (relâcher le bouton), MouseMove (déplacer le bouton) ont comme paramètre e qui contient les coordonnées souris, elles sont dans e.X et e.Y, ce sont bien les coordonnées DANS le contrôle (coordonnées 'client').

```
Private Sub ListBox2_MouseDown(ByVal sender As Object, ByVal e As
 System.Windows.Forms.DragEventArgs) _
Handles ListBox2.MouseDown

End Sub
```

Mais attention, dans les événements relatifs au Drag and Drop (DragOver par exemple) ce sont les coordonnées écran. Si je veux avoir des coordonnées relatives à l'objet graphique en cours, il faut les transformer à l'aide de **PointToClient** qui transforme un point écran en point client.

Exemple : La souris survole ListBox2, on a e.X et e.Y, coordonnées de l'écran, comment obtenir le Point par rapport à la ListView1 ?

On transforme e.X et e.Y en coordonnées client (par rapport à la listBox).
(e.Button retourne le bouton utilisé.)

```
Private Sub ListBox2_DragOver(ByVal sender As Object, ByVal e As
 System.Windows.Forms.DragEventArgs) _
Handles ListBox2.DragOver

MonImage.Location=ListBox2.PointToClient(New Point(e.X, e.Y))
```

En plus ListBox2.IndexFromPoint(ListBox2.PointToClient(New Point(e.X, e.Y)) retourne l'index survolé.

**PointToScreen** fait l'inverse (coordonnées listbox=> coordonnées écran).

## VIII-K-7 - Anchor

Permet d'**ancrer les bords.** Un bord ancré reste à égale distance du bord du conteneur quand le conteneur (la fenêtre habituellement) est redimensionné.

En mode conception, il suffit de cliquer sur '. . .' en face de Anchor pour voir s'ouvrir une fenêtre, cliquer sur les bords que vous voulez ancrer.

Par défaut les bords Top (haut) et left (gauche) sont ancrés.

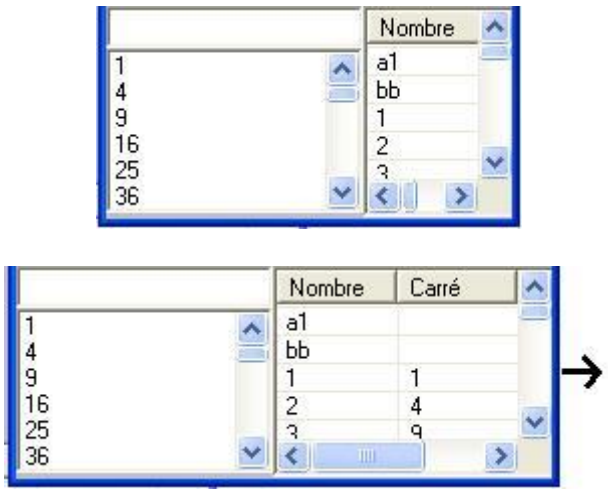
Expliquons !!

Left est ancré, si je déplace le bord droit de la fenêtre, le contrôle n'est pas déplacé, car la distance bord gauche de la fenêtre et bord gauche du contrôle est fixe. Par contre si je déplace le bord gauche de la fenêtre, le contrôle suit (ce qui parait évident !!!).

Exemple :

Prenons 2 contrôles dans une fenêtre, celui de gauche a Anchor =left et celui de droite à Anchor =left et right.

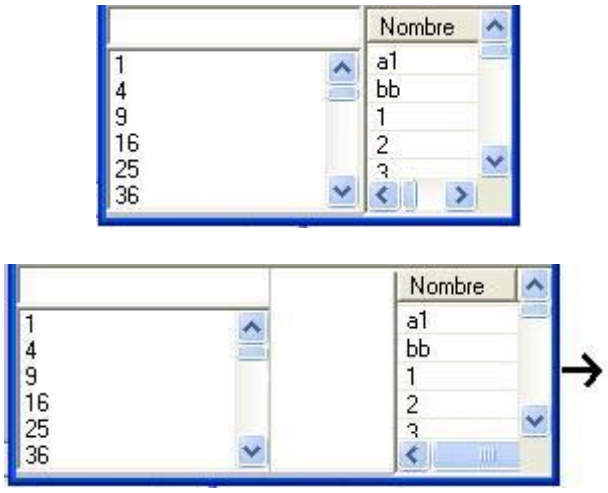
Si je déplace le bord droit (ou le gauche d'ailleurs), le contrôle droit est redimensionné, car la distance 'bord gauche du conteneur-bord gauche du contrôle droit' est fixe., les 2 contrôles restent côte à côte.




## VIII-K-8 - Dock

**Amarre aux bords.** La bordure spécifiée est ancrée directement au conteneur.

Exemple : le contrôle de droite est Dock=Right (Anchor=None)




Le bord droit du contrôle est accroché au bord droit du conteneur. Le contrôle droit est déplacé sans être redimensionné…

Il y a même possibilité d'amarrer aux 4 bords (Dock=Fill) pour remplir le conteneur, et de modifier la propriété DockPaddind du formulaire afin se s'éloigner légèrement des bords pour faire joli.

## VIII-K-9 - Spliter

Le contrôle Splitter sert à redimensionner des contrôles au moment de l'exécution par l'utilisateur.

Le contrôle Splitter est utilisé dans les applications dont les contrôles présentent des données de longueurs variables, comme l'Explorateur Windows.

Pour permettre à un utilisateur de redimensionner un contrôle ancré au moment de l'exécution, ancrer le contrôle à redimensionner au bord d'un conteneur, puis ancrez un contrôle Splitter sur le même côté de ce conteneur.

En VB.Net 2005 il existe aussi SplitContainer qui est plus pratique que Spliter et LayoutPanel voir 3-9.

## VIII-L - Main Menu, ContextMenu

Comment créer un menu principal en haut d'un formulaire ou un ContextMenu ?

Avec MainMenu et ContextMenu en VB 2003 ; on ne les utilise plus.

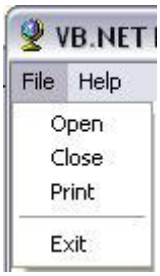
Avec **MenuTrip** et **ContextMenuStrip** à partir de VB 2005, c'est ceux qu'il faut utiliser.

### VIII-L-1 - MainMenu en Vb 2003

On peut ajouter un menu dans une fenêtre.

Beaucoup d'applications contiennent un menu.

Exemple de menu :

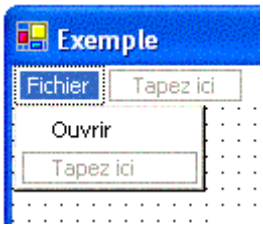



On remarque que le contenu des menus est standardisé afin que l'utilisateur s'y retrouve sans aide (l'utilisateur lit, à mon avis, rarement les aides !!).

Comment créer un menu ?

En allant dans la boite à outils, chercher un main menu et en le déposant sur la fenêtre : il apparait en dessous de la fenêtre.

Pour 'dessiner' le menu, il suffit de mettre le curseur sur le menu en haut de la fenêtre, où est écrit 'Taper ici' : tapez le texte du menu, ('Fichier' par exemple).




Il apparait automatiquement un 'Tapez Ici' pour les lignes dessous ou le menu suivant.

Les lignes du menu sont nommées automatiquement MenuItem1, MenuItem2…

Quand le curseur est sur une ligne du menu,la fenêtre de propriété donne les propriétés de la ligne :

la propriété ShortKey permet de créer un raccourci ;

la propriété Checked permet de cocher la ligne ;

la propriété Visible permet de faire apparaitre ou non une ligne.

Si vous double-cliquez sur la ligne du menu vous voyez apparaitre :

```
Private Sub MenuItem1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
_Handles MenuItem1.Click

End Sub
```

C'est la procédure événement liée à la ligne du menu.

Quand l'utilisateur clique sur une ligne du menu c'est le code contenu dans cette procédure qui est effectué.

## VIII-L-2 - Menu Contextuel Vb 2003

C'est un menu qui s'ouvre quand, le curseur de l'utilisateur est sur un objet, et qu'on clique sur le bouton droit de la souris.

En allant dans la boite à outils, chercher un Context menu, on le dépose sur la fenêtre : il apparait en dessous de la fenêtre.

Si on le sélectionne avec la souris, il apparait en haut et comme pour le menu principal, on peut ajouter des lignes.

Il faut ensuite affecter ce Context Menu à un contrôle ; pour cela donner à la propriété ContextMenu du contrôle le nom du ContextMenu.

```
TextBox1.ContextMenu= ContextMenu1
```

Si vous double-cliquez sur une ligne du menu vous voyez apparaitre les procédures événements correspondantes.

## VIII-L-3 - MenuStrip de Vb 2005

Remplace le MainMenu en VB 2005.


En allant dans la boite à outils, chercher un MenuStrip et en le déposant sur la fenêtre : il apparait en dessous de la fenêtre et la barre apparait en haut du formulaire.

On peut ajouter des menus, combobox et zone texte.

Pour remplir rapidement les menus, c'est comme en vb2003.

On peut mettre des images dans les menus.

Dans les propriétés Items permet d'avoir accès aux menus ou lignes et à toutes les propriétés des éléments (image…).

Chaque élément de la barre a sa procédure événement : pour le premier bouton par exemple :

```
Private Sub MenuStrip1_ItemClick(ByVal sender As System.Object, ByVal e As System.EventArgs)
_Handles ToolStripButton1.Click

End Sub
```

## VIII-L-4 - ContextMenuStrip de Vb 2005

Remplace le ContextMenu de Vb2003.

## VIII-M - Avoir le Focus

Nous allons étudier comment un objet de l'interface devient 'actif'.

Lorsqu'une fenêtre ou un contrôle est **actif** on dit qu'il a le **focus**.

Un contrôle qui a le focus est celui qui reçoit les événements clavier, souris…

Si une fenêtre prend le focus, sa barre de titre en haut prend la couleur active ; si c'est un contrôle texte, le curseur (le trait vertical) apparait, si c'est un bouton, il a un petit changement d'aspect qui indique que le focus est dessus.

## VIII-M-1 - Comment donner le focus à une fenêtre ?

Si une fenêtre est visible, la méthode **Activate** lui donne le focus.

```
Form1.Activate()
```

Dans ce cas l'événement Form1_Activated est effectué.

La méthode **Desactivate** est déclenchée quand la fenêtre perd le focus.

## VIII-M-2 - Comment donner le focus à un contrôle ?

Avec la méthode **Focus**.

```
TxtNom.Focus()
```

Avec la méthode **Select :**

```
TxtNom.Select()     'donne le focus à la zone de texte Txnom et met le curseur dedans.
```

On peut la surcharger et en plus sélectionner une portion du texte :

```
TxtNom.Select(2,3)    'donne le focus et sélectionne 3 caractères à partir du second.
```

ou forcer à ne rien sélectionner (second argument à 0).

On peut interdire à un contrôle le focus en donnant la valeur False à sa propriété **CanFocus**.

Aussi avant de donner le focus il est préférable de vérifier s'il peut le prendre :

```
If TxtNom.CanFocus then

    TxtNom.Focus()

End If
```

L'événement **GotFocus** se produit quand le contrôle prend le focus.

```
Private Sub TxtNom_GotFocus…

End Sub
```

**ActiveControl** est une propriété de la fenêtre qui indique le contrôle qui a le focus :

```
Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
 Button5.Click

        MsgBox(ActiveControl.Name) 'Affiche Button5

End Sub
```

## VIII-M-3 - Prise ou perte du focus

Lorsqu'un contrôle prend le focus, il se déclenche dans l'ordre :

**Enter**

Se produit quand l'utilisateur entre dans le contrôle.

**GotFocus**

Se produit quand le contrôle prend le focus.

**Leave**

Se produit quand le focus quitte le contrôle.

**Validating**

Se produit lorsque le contrôle est en cours de validation.

Il y a peut-être quelque chose à vérifier avant de quitter le contrôle. On va quitter le contrôle, ce n'est pas encore fait. Il faut vérifier avant. La validation c'est vous qui devez la faire !!! en écrivant du code.

Pour un bouton, par exemple, se produit lorsque l'on quitte le bouton, cela permet de contrôler la validité de certaines données et si nécessaire d'interdire de quitter le contrôle si certaines conditions ne sont pas remplies.

Exemple : ne pas quitter une textbox si l'utilisateur n'a rien tapé :

```
Private Sub TextBox1_Validating ((ByVal sender As Object, ByVal e As
 System.ComponentModel.CancelEventArgs)
_Handles TextBox1.Validating

If   TextBox1.Text=""  then

    e.Cancel = True    'Annuler la perte du focus: le focus reste sur TextBox11

End If

End Sub
```

**Validated**

Se produit lorsque le contrôle a terminé sa validation.

**LostFocus**

L'événement LostFocus se produit quand le contrôle perd le focus, mais attention :

lorsque vous changez le focus à l'aide du clavier (TAB, MAJ+TAB, etc.), en appelant les méthodes Select ou SelectNextControl, ou en définissant la propriété ContainerControl.ActiveControl au formulaire actuel, les événements Focus se produisent dans l'ordre suivant :

1 Enter ;
2 GotFocus ;
3 Leave ;
4 Validating ;
5 Validated ;
6 **LostFocus**.

Lorsque vous changez le focus avec la souris ou par l'appel à la méthode Focus, des événements Focus se produisent dans l'ordre suivant :

1 Enter ;
2 GotFocus ;
3 **LostFocus** ;
4 Leave ;
5 Validating ;
6 Validated.

Dans Validating, si la propriété Cancel du CancelEventArgs prend la valeur true, tous les événements qui se produiraient normalement après l'événement Validating sont supprimés.

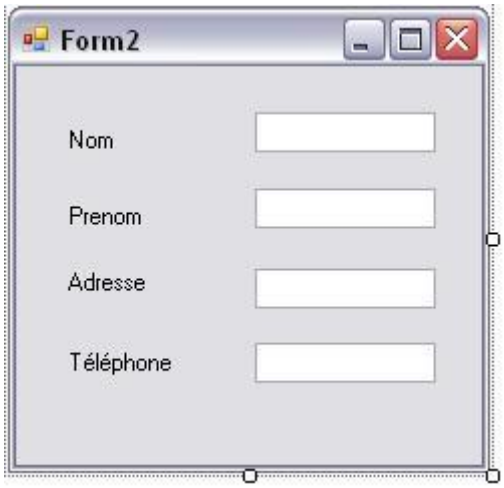
Si la propriété **CauseValidating** du contrôle a la valeur false, les événements Validating et Validated sont supprimés.

Les événements Enter et Leave sont supprimés dans les formulaires (fenêtres) Les événements équivalents dans la classe Form sont les événements Activated et Desactivate.

*Certains contrôles ne peuvent pas avoir le focus, comme les labels par exemple.*

## VIII-M-4 - La touche TAB pour passer d'un contrôle à l'autre

Dans une application où un utilisateur tape beaucoup de données dans de multiples contrôles, il passe souvent d'un contrôle (TextBox par exemple) au suivant avec la touche TAB.




Comment permettre cela ? Chaque contrôle à une propriété **TabIndex** qui s'incrémente automatiquement de 0 à 1, 2, 3… quand en cours de conception on ajoute des contrôles sur une fenêtre.

Lorsque l'utilisateur appuie sur TAB, le focus passe au contrôle qui a le TabIndex immédiatement supérieur.

On peut modifier le TabIndex des contrôles pour modifier l'ordre de tabulation.

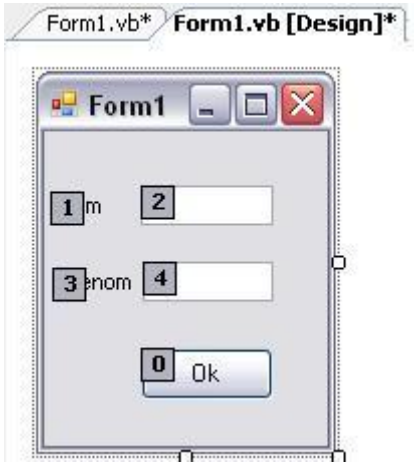
Quand **TabStop** a la propriété False (au niveau d'un contrôle) celui-ci est exclu de l'ordre de tabulation et le focus ne s'arrête pas dessus.

En VB 2005 on peut très rapidement modifier l'ordre de tabulation :

passer par le menu Affichage-> Ordre de tabulation.

En mode design apparait sur chaque contrôle un carré avec le numéro du TabIndex ; il suffit de cliquer successivement sur chaque carré dans l'ordre des tabulations croissantes pour mettre les tabulations dans le bon ordre.

Il faut pour finir repasser par le menu Affichage-> Ordre de tabulation.




## VIII-M-5 - Raccourcis clavier

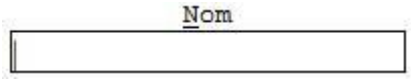
Dans beaucoup d'applications, certains contrôles ont un raccourci clavier.

Exemple : Nouveau est une ligne de menu. N étant souligné, ALT-N déclenche la ligne de menu, donne le focus au contrôle.

Comment faire cela : dans la propriété Text du contrôle, quand on tape le texte en mode conception, il faut mettre un '&' avant la lettre qui servira de raccourci clavier.

'&Nouveau' dans notre exemple affichera bien Nouveau et ALT N fonctionnera.

Pour une TextBox, la propriété text ne peut pas être utilisée, aussi il faut mettre devant la textBox un contrôle label (qui lui ne peut pas avoir le focus), si les TabIndex du label et du TextBox se suivent, le fait de faire le raccourci clavier du label donne le focus au TextBox.



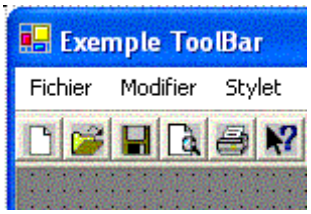

Exemple quand l'utilisateur tape Alt-N, le focus va dans le TextBox dessous.

## VIII-N - Barre de boutons, barre d'état

Comment mettre une barre de boutons en haut et une barre d'état en bas ?

## VIII-N-1 - La barre de boutons : ToolBar en VB 2003 (ne plus utiliser)

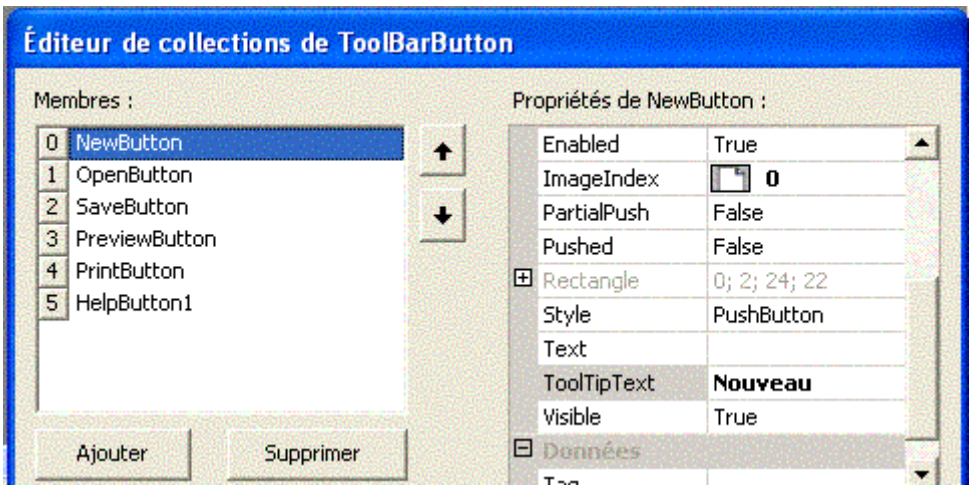
Voici un exemple classique, sous le menu il y a une barre de bouton : Nouveau, Ouvrir, Enregistrer, Chercher, Imprimer…




Allez chercher dans la boite à outils un contrôle ToolBar, il se place en haut, sous le menu. Mettre aussi un ImageList. Un contrôle ImageList est un contrôle qui stocke des images, chaque image étant chargée en mode conception et repérée par un numéro (0,1,2,3…).

Dans la ToolBar mettre dans la propriété ImageList le nom du contrôle ImageList qui contient les images des boutons.

Ouvrir la collection Buttons dans la fenêtre des propriétés de la ToolBar pour pouvoir ajouter ou modifier les boutons :




Vous pouvez ajouter ou enlever des boutons.

Chaque bouton a ses propriétés affichées à droite :

**Name :** nom du Bouton Exemple NewButton ;

**ImageIndex :** donne le numéro de l'image (contenue dans l'Imagelist) qui doit s'afficher dans le bouton ;

**ToolTipText :** donne le texte du ToolTip (carré d'aide qui apparait quand on est sur le bouton) Il faut aussi que la propriété ShowToolTip de la ToolBar soit à True.

L'événement déclenché par le clic de l'utilisateur sur un bouton est : ToolBar1_ButtonClick

L'argument e contient les arguments de l'événement clic de la ToolBar. e.Button contient le nom du bouton qui a déclenché l'événement. Pour chaque nom de bouton on appellera la procédure correspondante : NewDoc(), Open()…

Comme d'habitude il suffit de double-cliquer sur la ToolBar pour faire apparaitre ToolBar1_ButtonClick

Voici le code complet :

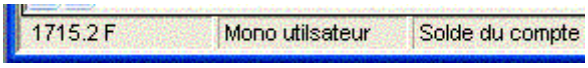
```
Private Sub ToolBar1_ButtonClick(ByVal Sender As System.Object,
_ByVal e As System.Windows.Forms.ToolBarButtonClickEventArgs)
_Handles toolBar1.ButtonClick

If e.Button Is NewButton Then
```

```
        NewDoc()
ElseIf e.Button Is OpenButton Then
        Open()
ElseIf e.Button Is SaveButton Then
        Save()
ElseIf e.Button Is PreviewButton Then
        PrintPreview()
…
End If
End Sub
```

*Le ToolBar a donc une collection de Buttons, de plus il n'y a qu'une procédure événement 'Click' unique propre à la ToolBar et pour tous les boutons.*

## VIII-N-2 - Contrôle StatusBar en VB 2003 (ne plus utiliser)

La barre d'état se trouve en bas de la fenêtre et affiche des informations relatives aux opérations en cours.




Ajouter un StatusBar au formulaire. Dans la fenêtre des propriétés du StatusBar, la collection Panels contient les zones d'affichage du StatusBar.

Dans le code, pour modifier le texte d'une zone faire :

```
StatusBar1.Panels(0).Text="1715.2F"
```

On remarque (c'est du Net) que le premier panel est panels(0).

## VIII-N-3 - ToolStrip en VB 2005

On peut créer une barre n'importe où dans le formulaire.

Exemple de barre de menu comprenant :

un bouton ;

un label ;

un bouton déroulant un menu ;

un comboBox ;

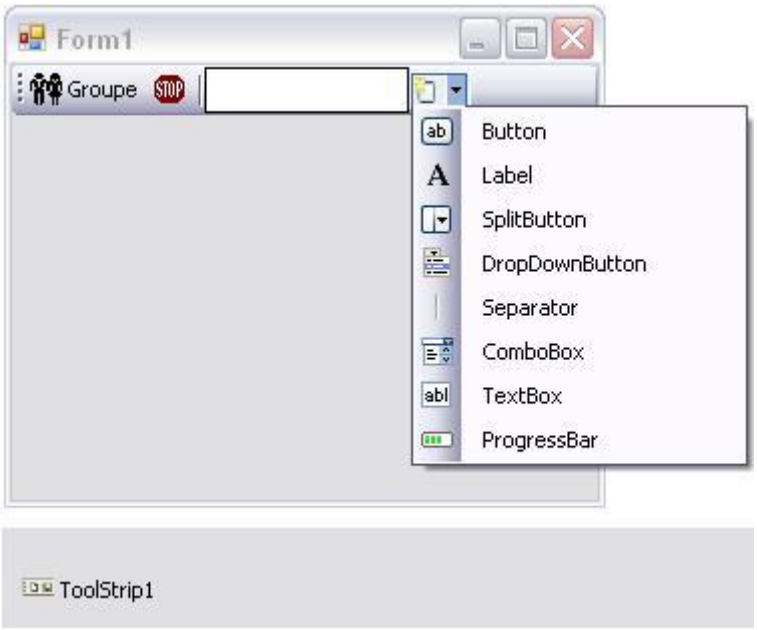
une zone texte ;

une barre de progression.


Images possibles dans les menus ; il peut y avoir des séparateurs.

Pour créer la barre ToolStrip, allez la chercher dans la boite à outils.

À la droite de la barre, en mode design, se trouve un menu déroulant qui permet de choisir l'élément à ajouter à la barre :




Cela ajoute des ToolStripLabel, ToolStripButton… (noter que ce sont des objets spécifiques aux ToolStrip). Chaque élément ajouté est un objet.

Ensuite, en cliquant sur chaque élément de la barre, on peut changer ses propriétés (qui apparaissent en bas à droite).

Ici le premier élément à gauche est un label ; j'en ai modifié la propriété (text='Groupe') et j'ai mis une image (il a accepté une icône) dans la propriété 'Image'. Le second élément est un bouton avec une image de stop.

Dans le code, on a accès aux propriétés de l'élément directement à partir de son nom :

```
ToolStripButton1.Text="OK"
```

Ou par l'intermédiaire de la barre ToolStrip qui a une collection d'items contenant tous les objets de la barre :

```
ToolStrip1.Items.Item (2).Text
```

À la place de l'index de l'élément dans la barre (ici 2), on peut mettre le nom d'élément.

Événement déclenché par un clic.

1- Comme d'habitude, en double-cliquant sur un élément (le second par exemple qui correspond à un bouton), on se retrouve dans la procédure exécutée quand l'utilisateur clique sur le bouton.

```
Private Sub ToolStripButton1_Click(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles ToolStripButton1.Click

End Sub
```



*Attention : chaque élément a sa propre procédure événement.*

Ainsi s'il y a un second bouton, il y aura une autre procédure Click :

```
Private Sub ToolStripButton2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
_Handles ToolStripButton2.Click

End Sub
```

2- On a aussi une procédure unique pour le clic sur la barre :

```
Private Sub ToolStrip1_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
 ToolStrip1.Click

End Sub
```

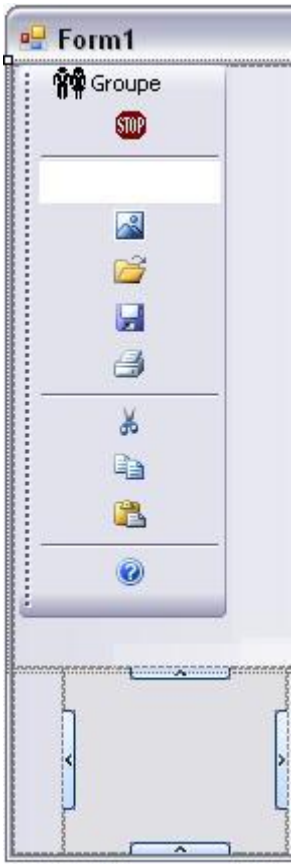
Mais le sender est le ToolStrip ; pour savoir dans la routine quel bouton a été cliqué, il faut modifier la sub en indiquant comme Handles le nom de tous les boutons de la barre, par exemple, on peut savoir quel bouton a été cliqué.

```
Private Sub ToolStrip1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
_Handles ToolStripButton1.Click,
_ToolStripButton2.Click, ToolStripButton3.Click

Dim Button As System.Windows.Forms.ToolStripItem = CType(sender,
 System.Windows.Forms.ToolStripItem)

  MsgBox(Button.Name) '

End Sub
```

La barre de boutons peut être mise horizontalement (grâce à la propriété **LayoutStyle**).

Avec l'aide du petit bouton permettant les tâches courantes sur le ToolStrip, on peut comme ici, incorporer instantanément tous les boutons standards (nouveau, ouvrir, enregistrer, imprimer, couper, copier, coller. Magique !!).

Avec l'aide du petit bouton permettant les tâches courantes sur le ToolStrip, on peut aussi mettre le ToolStrip dans un conteneur (ToolStripContainer), en mode design, on voit apparaitre dessous des outils permettant de modifier le conteneur ; si on clique sur un des côtés (dessous) on a accès aux propriétés du bord (mais la mise en œuvre n'est pas facile !!).




Il n'y a plus de 'groupe de bouton' avec un seul bouton enfoncé, ou du moins je n'ai pas trouvé. (On ne peut pas tout avoir!!) par contre, on peut 'enfoncer' ou non un bouton :

```
.ToolStripButton2.Checked = True
```

On n'a pas accès à cette propriété (et d'autres) en utilisant les Items du ToolSTrip.

Comment créer un bouton à bascule ?

Lorsqu'un utilisateur clique sur un bouton bascule, ce bouton apparait enfoncé et conserve cet aspect jusqu'à ce que l'utilisateur clique une nouvelle fois sur le bouton.

```
toolStripButton.CheckOnClick = True
toolStripButton.CheckedChanged AddressOf EventHandler(toolStripButton_CheckedChanged)
```

Merci Microsoft (non testé).

Le ToolStrip contient des objets (button, label…) qui ont chacun leur procédure événement.

## VIII-N-4 - StatuStrip en VB 2005

Pour créer une barre d'état en bas d'un formulaire ; remplace les StatusBar.

Peut contenir des ToolStripStatusLabel (Texte ou icône), des ToolStripProgressBar, ToolStripDropDownButton et ToolsStripSplitButton (combinaison d'un bouton standard et d'un bouton déroulant). Il n'y a plus de Panel.

# VIII-O - Les images

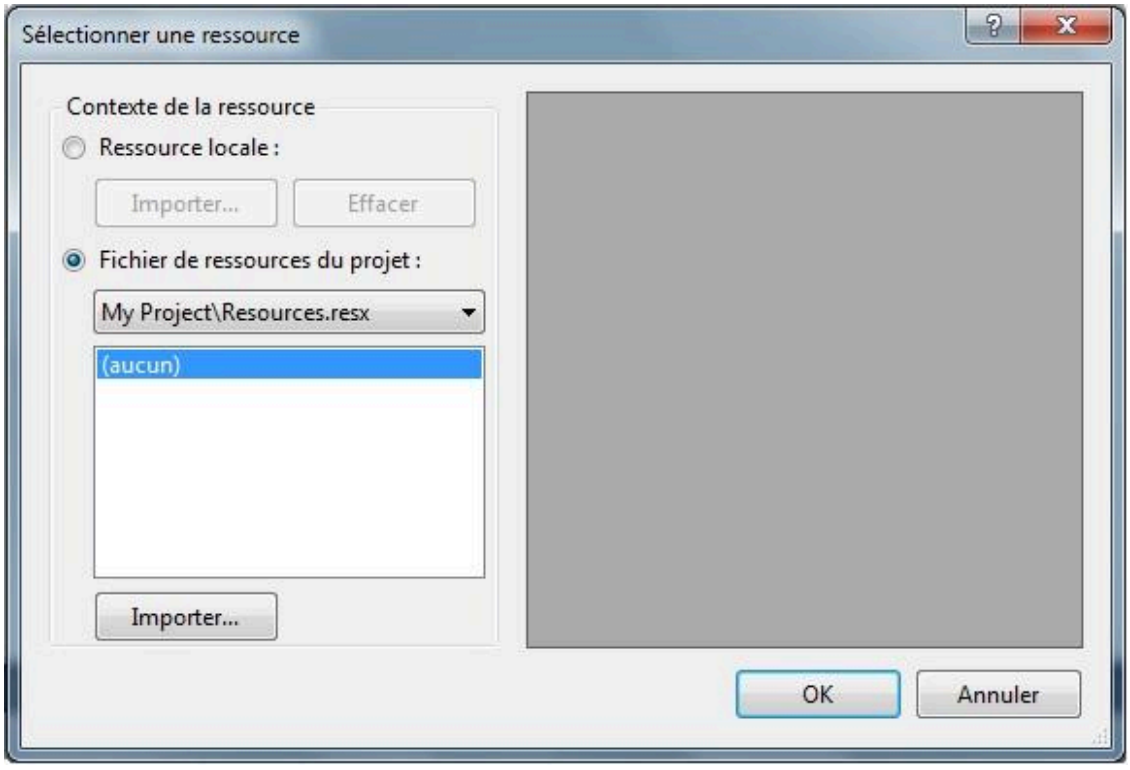
Comment afficher des images ?

## VIII-O-1 - Le contrôle 'PictureBox'

Le contrôle PictureBox sert à afficher des graphismes au format bitmap, GIF, JPEG, métafichier ou icône (Extension .BMP .GIF .JPG .WMF .ICO).

L'image à afficher est déterminée par la propriété **Image**, laquelle peut être définie au moment de l'exécution ou du design. La propriété SizeMode détermine la façon dont l'image et le contrôle se dimensionnent l'un par rapport à l'autre.

**On peut charger une image en mode conception** par la fenêtre 'propriétés' et la propriété 'Image' :




Là, on peut charger un fichier image ou une image qui est dans les ressources du programme.

On peut aussi charger une image par code :

```
PictureBox1.Image = Image.FromFile("vimage.gif")
```

La Classe Image (Shared) possède une fonction qui retourne une image à partir d'un fichier. On l'affecte ensuite à la propriété Image du PictureBox1.
(Il est aussi possible d'utiliser FromStream).

Ho! merveille, les GIF animés sont acceptés et s'animent sous VB.

**Comment effacer une image ?**

```
If Not (PictureBox1.Image Is Nothing) Then
    PictureBox1.Image.Dispose()
    PictureBox1.Image = Nothing
End If
```

Comment copier une image d'un PictureBox dans un autre PictureBox ?

```
PictureBox1.Image = PictureBox2.Image
```

La Classe PictureBox.Image a comme d'habitude des propriétés et des méthodes :
Clone (pour copier) ;
RawFormat (format du fichier) ;
Height et Width (hauteur et largeur de l'image en pixels) ;
VerticalResolution et HorizontalResolution (en pixels/pouces) ;
PhysicalDimension (retourne largeur et hauteur) ;
IsCanonicalPixelFormat et IsExtendedPixelFormat retournent True s'il y a respectivement 32 et 64 bits par pixels ;
PixelFormat (format du pixel) ;
…

La méthode **RotateFlip** permet par exemple d'effectuer une rotation de l'image ; quand on tape le code, VB donne automatiquement la liste des paramètres.

```
PictureBox1.Image.RotateFlip(RotateFlipType.Rotate90FlipX)
```

La méthode **Save** sauvegarde l'image dans un fichier.

```
PictureBox1.Image.Save("c:\image.bmp")
```

Bien noter que le nom de l'extension suffit à imposer le format de l'image sur le disque.

On peut charger une image .GIF puis la sauvegarder en .BMP

**PictureBox1.ErrorImage** donne le nom de l'image à afficher si le chargement d'une image échoue.
**PictureBox1.InitialImage** donne le nom de l'image à afficher pendant le chargement de l'image à afficher.

Il est possible de définir **une couleur comme 'transparente' :** voir la page sur les boutons.

**Comment charger une image à partir d'un fichier, mais sans 'bloquer' ce fichier.**

On a vu qu'on peut 'charger' une image par PictureBox1.Image = Image.FromFile("vimage.gif")

L'inconvénient de cette méthode est que tant que le programme est ouvert, le fichier correspondant sur le disque est utilisé et par conséquent il est impossible de travailler dessus. (Impossible d'effacer le fichier par exemple !!)

Une méthode pour libérer le fichier est d'utiliser un Stream (merci la Faq de Developpez.com) :

```
'En haut du module
```

```
Imports System.IO

' Créer le FileStream sur le fichier vimage.gif

Dim MyStream As FileStream = New FileStream("C:\vimage.gif&#8221;, FileMode.Open)

 ' affecter l'image à pictureBox1

pictureBox1.Image = Image.FromStream(MyStream)

' libérer les ressources

MyStream.Close

' supprimer le fichier vimage.gif

 File.Delete("C:\vimage.gif&#8221;)
```

**Comment placer l'image dans le PictureBox ?**

La propriété **SizeMode** impose comment l'image sera placée dans le contrôle PictureBox :

aucun : l'image est alignée en haut à gauche du contrôle. Elle peut être trop grande ou trop petite, mais rien ne change de taille ;

Stretch : l'image est automatiquement étirée afin que sa taille s'adapte à celle du contrôle qui la contient ;

Autosize : la taille du contrôle est modifiée pour faire la taille de l'image ;

CenterImage : l'image est centrée par rapport au contrôle.

## VIII-O-2 - La propriété 'Image' des contrôles

De nombreux contrôles Windows Forms peuvent afficher des images. L'image affichée peut être une icône symbolisant la fonction du contrôle ou une image d'arrière-plan ; par exemple, l'image d'une disquette sur un bouton indique généralement une commande d'enregistrement. L'icône peut également être une image d'arrière-plan conférant au contrôle une certaine apparence.

**Pour tous les contrôles affichant des images :**

- l'image peut être définie à l'aide des propriétés **Image** ou **BackgroundImage** directement en mode Design par la fenêtre des propriétés. Il faut sélectionner Image puis cliquez sur "…" et choisir un fichier contenant une image. Dans ce cas, une fois chargée, l'image fait partie intégrante du programme. (Il n'est pas utile de fournir le fichier .BMP ou .GIF avec l'application) ;
- lorsque le programme 'tourne' on peut aussi charger une Image. Le code affecte à la propriété Image ou BackgroundImage un objet de type System.Drawing.Image, en général, vous utiliserez la méthode **FromFile** de la classe Image pour charger une Image à partir d'un fichier. (Dans ce cas le fichier contenant l'image doit être fourni.)

Exemple pour un bouton :

```
button1.Image = Image.FromFile("C:\Graphics\MyBitmap.bmp")
' Aligne l'image.
button1.ImageAlign = ContentAlignment.MiddleRight
```

Exemple pour un label :

```
Dim Label1 As New Label()
Dim Image1 As Image
```

```
Image1 = Image.FromFile("c:\\MyImage.bmp")

' modifier la taille du label pour qu'il affiche l'image.

Label1.Size = Image1.Size

' Mettre l'image dans le label.

Label1.Image = Image1
```

Si on renseigne la propriété Image, on ne peut pas utiliser en même temps la propriété ImageList décrite ci-dessous.

Ces Images et BackGroundImage ont toutes les propriétés et méthodes des images vues dans les images des PictureBox.

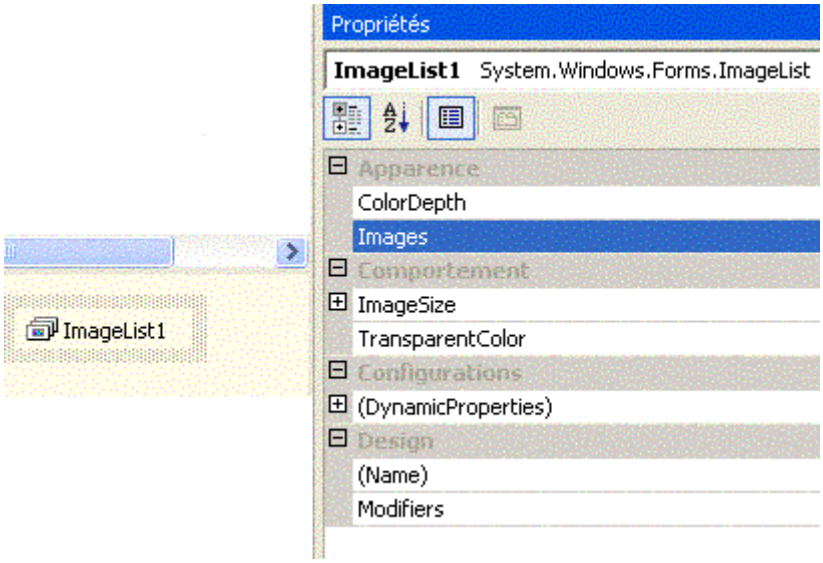
## VIII-O-3 - Le contrôle ImageList

Il sert de container à images, c'est une collection d'images. les images qu'il contient seront utilisées par d'autres contrôles (PictureBox, Listview, TreeView, Button….).

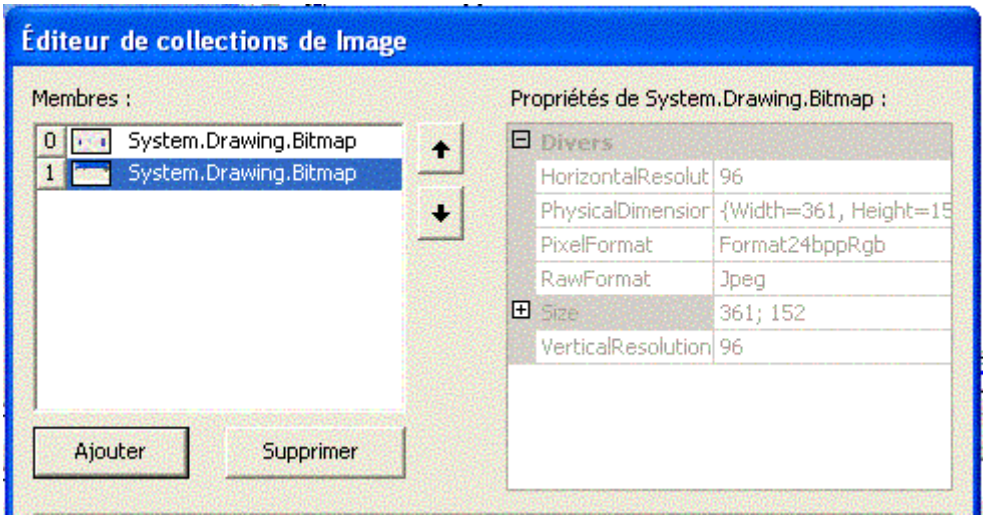
Il n'est pas visible en exécution.

En conception il apparait en bas sous la fenêtre. À droite figurent ses propriétés, en particulier, la collection Images qui contient les images et la propriété TransparentColor qui indique la couleur qui doit être transparent, c'est-à-dire non visible.

Il faut l'ajouter au formulaire, il apparait en dessous.




Si je clique sur le bouton '…' en face de Images, l'éditeur de collections d'image s'ouvre.




On peut ajouter des images avec le bouton 'Ajouter'.

L'ImageList est ainsi chargé.

Ensuite pour utiliser une image de l'ImageList dans un autre contrôle, il faut modifier les propriétés de cet autre contrôle (un bouton par exemple).

La propriété **ImageList** du bouton doit contenir le nom du contrôle imageList et **ImageIndex** du bouton doit contenir l'index de l'image dans l'ImageList.

```
btOk.ImageList = imagelist1

btOk.ImageIndex = 2
```

Un ImageList peut aussi être chargé par code :

```
imageList1.Images.Add(Image.FromFile(NomImage))
```

On ajoute à la collection Images une image venant d'un fichier nommé NomImage.

On peut surcharger la méthode Add en fournissant en plus la couleur transparente.

```
imageList1.Images.Add(Image.FromFile(imageToLoad), CouleurTransparente)
```

La taille des images peut aussi être modifiée par code :

```
imageList1.ImageSize = New Size(255, 255)
imageList1.TransparentColor = Color.White
```

## VIII-P - Couleurs et Font

# VIII-P-1 - Les couleurs

## VIII-P-1-a - Généralités

Une couleur est représentée par 3 octets correspondant aux composants de couleur rouge, vert et bleu. Chaque octet peut prendre la valeur 0 à 255 (ou 0 à FF en hexadécimal).

Si on utilise la notation hexadécimale, il faut mettre &H avant : &HFF correspond à 255.

Exemple : valeur des 3 composantes couleur et couleurs correspondantes :

| rouge | vert | bleu | |
|---|---|---|---|
| 0 | 0 | FF | |
| FF | 0 | 0 | |
| 80 | 0 | 80 | |
| 0 | 0 | 0 | |
| FF | FF | FF | |

En plus, dans certains cas, il y a une composante alpha qui indique la transparence. 255 indique que la couleur est opaque, 1 à 254 indique que la couleur est transparente.

Il y a une Classe **Color** dans SystemDrawing. On peut instancier un Objet Color :

```
Dim myColor As Color
```

On peut voir les composants de cette couleur avec :

myColor.A composante alpha ;

myColor.B composante bleue ;

myColor.R composante rouge ;

myColor.G composante verte.

On ne peut pas les modifier, car ces propriétés sont en ReadOnly !! Utiliser FromArg pour modifier.

## VIII-P-1-b - Énumération Color

Le plus simple est, pour modifier la couleur d'un objet par du code, d'utiliser l'énumération 'Color' qui contient le nom d'une couleur toute faite (en RGB sans composante Alpha):

Color.Back ;

Color.Fuchsia ;

Color.Chocolate ;

Color.Red …

Voici toutes les couleurs à votre disposition:


Elles font partie de **System.Drawing.**

Comme d'habitude, il suffit de taper Color. et la liste très longue des couleurs s'ouvre.

```
Bouton.BackColor=Color.Red     'modifie la couleur de fond du bouton
```


Ces couleurs semblent correspondre aux **couleurs 'Web'**. Il y a longtemps quand on avait des moniteurs affichant 256 couleurs, il existait une liste nommée 'web-safe colors ' contenant 216 couleurs qui étaient des couleurs 'sûres', c'est-à-dire affichables, sans utilisation de tramage (le tramage étant l'affichage d'une couleur en juxtaposant des pixels de couleurs différentes pour se rapprocher de la couleur manquante) ; mais maintenant cela est obsolète avec les moniteurs actuels. Il semble que les couleurs VB correspondent aux couleurs Web bien qu'il y ait plus de couleurs Web que de couleurs VB.

## VIII-P-1-c - Rouge, vert, bleu

Plus puissant :

```
Color.FromArgb
```

Crée une couleur à partir des valeurs des quatre composants ARVB (argb en anglais) 8 bits (alpha, rouge, vert et bleu).

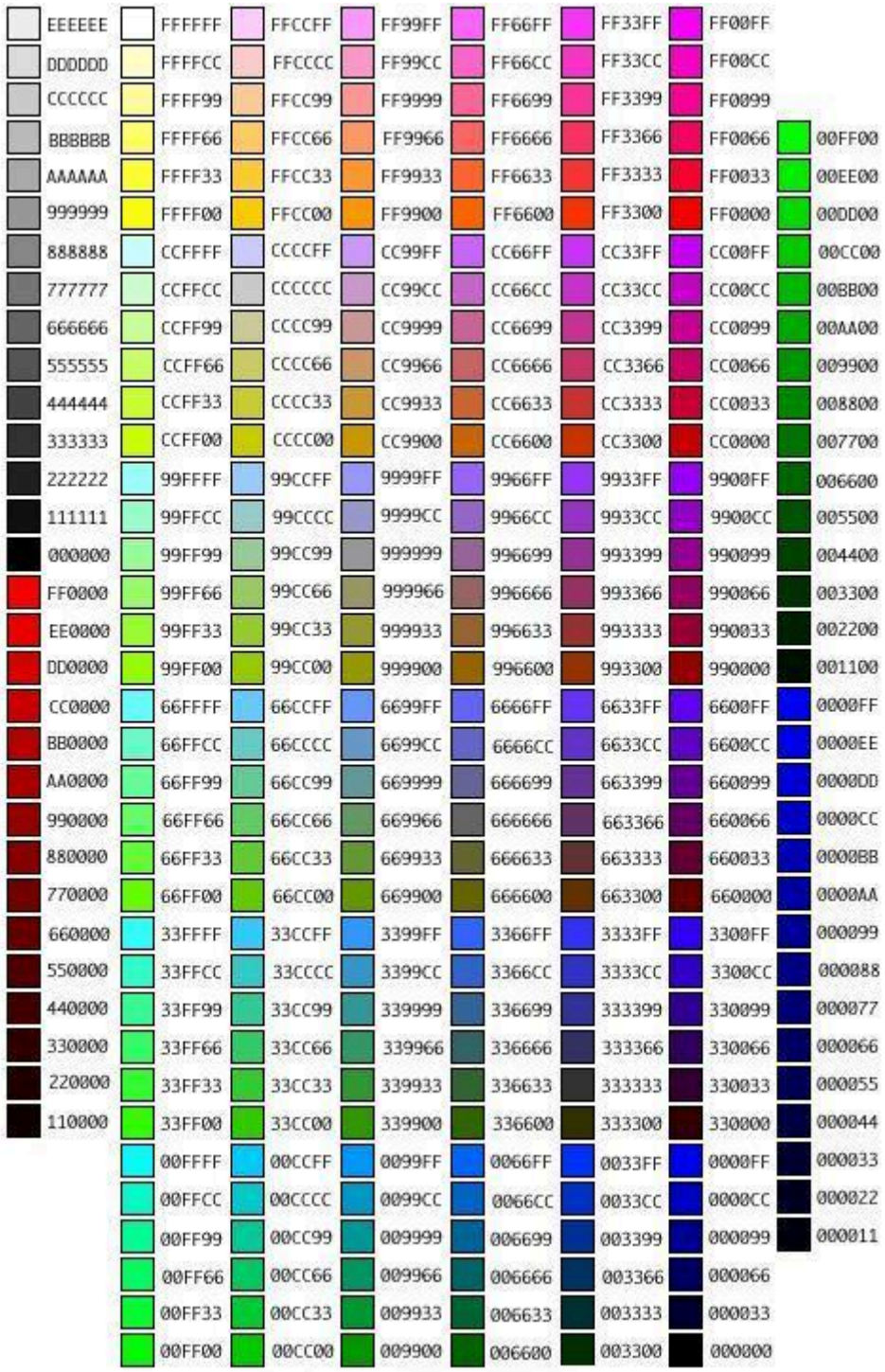
alpha indique la transparence. 255 indique que la couleur est opaque, 1 à 254 indique que la couleur est transparente.

L'octet le plus significatif, représenté par AA, correspond à la valeur du composant alpha. Les second, troisième et quatrième octets, représentés par RR, VV et BB, correspondent aux composants de couleur rouge, vert et bleu, respectivement. Chaque octet prend la valeur 0 à 255 ou 0 à FF en hexadécimal.

Le paramètre correspond à 4 X 8bits=32 bits= un Integer. Pour plus de clarté, on rentre généralement les données en hexadécimal :

```
Me.BackColor= Color.FromArgb(&H780000FF)     'correspond à un bleu transparent.
```

Voici les principales couleurs et le code hexadécimal correspondant :




Il y a des surcharges.

On peut passer chaque paramètre séparément :

```
Me.BackColor=Color.FromArgb(120, 0, 0, 255)
```

On peut aussi passer l'alpha et la couleur en second paramètre. Pour obtenir une couleur bleue à moitié transparente :

```
MaCouleur = Color.FromArgb(128,color.blue)
```

**Plus simple**

On peut définir une couleur avec la fonction RGB (red, green, blue), pas de composantes alpha ici.

```
Dim red As Color = RGB(255, 0, 0) ' fait partie de Microsoft.VisualBasic
```

## VIII-P-1-d - Couleurs 'System'

Ce sont les couleurs utilisées par Windows pour afficher la barre de titre, les fonds, couleur d'éléments actifs ou non. On peut modifier ces couleurs en passant par le panneau de configuration (option 'Affichage'). Toutes les applications les utilisent. On peut aussi les utiliser.

L'énumération KnownColor contient les couleurs système (couleur des barres, texte, fenêtre active…).

Mais pour utiliser une couleur system, il faut employer SystemColors.

```
Me.BackColor = SystemColors.ActiveBorder 'modifie la couleur de fond du formulaire en cours
```

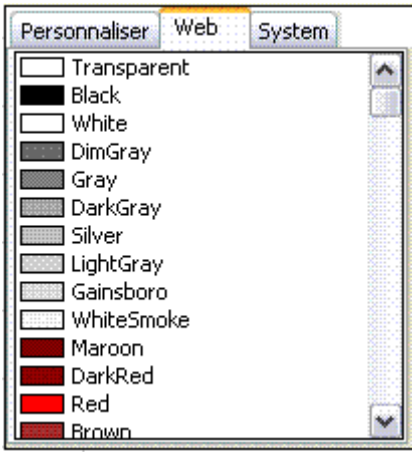
## VIII-P-1-e - Couleur dans les objets

Pour changer la couleur d'arrière-plan du contrôle (le fond), utilisez la propriété **BackColor**, la propriété d'avant plan est **Forecolor** (la couleur du texte dans un bouton par exemple).

Ici pour ce bouton, BackColor est égal à Color.Red et ForeColor est à Color.Black

Dans le code MyButton.BackColor = Color.Red

En mode Design (conception), on peut modifier la couleur directement en cliquant sur le bouton '…' en face de BackColor par exemple : la fenêtre de choix des couleurs apparait :




On a le choix entre toutes les couleurs possibles (65 535) par 'Personnaliser', les couleurs Web (celles de l'énumération Color) et couleurs system (System).

# VIII-P-1-f - Choix d'une couleur par l'utilisateur

Pour permettre à l'utilisateur de choisir une couleur, il faut mettre dans le formulaire une ColorDialog à partir de la boite à outils ; elle vient se placer sous le formulaire :


Il faut ensuite, par code, ouvrir cette ColorDialog.

La classe ColorDialog a une méthode ShowDialog, analogue à la méthode ShowDialog des classes OpenFileDialog et


Si l'utilisateur quitte la boite de dialogue en cliquant sur le bouton 'OK', la méthode ShowDialog retourne DialogResult.OK et la couleur choisie est dans la propriété Color de l'objet ColorDialog.

Exemple

L'utilisateur clique sur un bouton nommé 'CouleurButton' cela ouvre la ColorDialog, l'utilisateur clique sur une couleur puis sur 'OK', cela donne aux caractères de la TextBox 'Texte' la couleur choisie :

```
Private Sub CouleurButton_Click(ByVal sender As Object, ByVal e As System.EventArgs) _

Handles CouleurButton.Clic

' Ouverture de la dialogBox colordialog1

If colorDialog1.ShowDialog() = DialogResult.OK Then

' on met la couleur dans la propriété forecolor du TextBox

Texte.ForeColor = colorDialog1.Color

End If

End Sub
```

On peut modifier la boite à notre goût avant de l'ouvrir :

```
colorDialog1.SolidColorOnly= True  'Couleurs pures (unies) seulement
colorDialog1.AllowFullOpen= True  'Autorise le bouton des couleurs personnalisées (volet de
 droite)
colorDialog1.FullOpen= True  'Affiche les couleurs personnalisées
colorDialog1.Color= Color.Red  'Couleur présélectionnée
```

## VIII-P-2 - Police de caractères (ou Font)

Une 'Font' est définie par :

- un nom ;

- une taille ;

- un style (gras, italique, souligné…)

Pour modifier la police de caractères utilisée dans un contrôle, il faut lui assigner un objet '**Font**'

```
label1.Font = New System.Drawing.Font("Arial", 10)
```

On indique le nom de la font et la taille, on aurait pu ajouter un troisième argument : le style (gras, italique, souligné).

```
    label1.Font = New System.Drawing.Font(label1.Font, FontStyle.Bold Or FontStyle.Italic)
```

Ici c'est une autre signature Font, Style. On peut ajouter 2 autres paramètres (Unit et jeux de caractères).

Visual Basic .NET prend en charge les polices **TrueType** et **OpenType**.

Dans un contrôle, les propriétés de police sont automatiquement héritées du parent, sauf lorsqu'elles sont explicitement définies pour l'objet enfant. Par exemple, si vous avez deux contrôles d'étiquette dans un formulaire et que vous changiez les propriétés de police du formulaire en Arial, les polices du contrôle d'étiquette sont également changées en Arial. Si par la suite vous changez la police d'une étiquette en Times Roman, les modifications qui pourront être apportées à la police du formulaire ne remplaceront pas la police de l'étiquette.

**Pour lire les fonts installés,** utiliser l'espace de noms System.Drawing.FontFamily.

```
Dim ff As FontFamily
For Each ff In System.Drawing.FontFamily.Families
listBox1.Items.Add(ff.Name)
Next
```

**Pour que l'utilisateur modifie la police du contrôle List1.**

```
Dim myFontDialog As FontDialog
   myFontDialog = New FontDialog()

   If myFontDialog.ShowDialog() = DialogResult.OK Then

      List1.Font = myFontDialog.Font
   End If
```

Les polices True Type sont précédées d'un TT.

Les polices Open Type sont précédées d'un O.

On ouvre une fenêtre de choix de police, si une police est sélectionnée par l'utilisateur, on l'applique à List1.

**Pour mettre la font de la form en gras :**

```
If Not (Me.Font.Bold) Then
        Me.Font = New Font(Me.Font, FontStyle.Bold)
End If
```

Pour qu'un TextBox utilise la font "Courier New" de taille 12 en italique :

```
                        TextBox1.Font = New Font("Courier New", 12, FontStyle.Italic)
```

**Attention**

Ne pas utiliser dans votre programme des font 'exotiques' que vous trouvez très belle, mais qui ne seront pas présentes sur l'ordinateur des utilisateurs (elles seront dans ce cas substituées par d'autres avec un résultat aléatoire). Utilisez de l'Arial, il n'y aura pas de problèmes !!

**Police BitMap, 'True Type', 'Open Type'**

Avant les années 1990 il y avait des polices au format BitMap (image matricielle). Elles ne sont plus utilisées par VB.Net.

Depuis 1980 existe **True Type**, un format de police multiplateforme vectorielle, développé par Apple et vendu ensuite à Microsoft.

(Concurrent du format Type1 de PostScript d'Adobe.)

On utilise depuis 2001 **OpenType** qui est un nouveau format de police multiplateforme vectorielle, développé conjointement par Adobe et Microsoft. Adobe propose plusieurs milliers de polices OpenType.

Les deux principaux atouts du format OpenType résident dans sa compatibilité multiplateforme (un seul et même fichier de polices exploitable sur les postes de travail Macintosh et Windows) et sa prise en charge de jeux de caractères et de fonctions de présentation très étendus, qui offrent de meilleures capacités linguistiques et un contrôle typographique évolué.

Une police Open Type est basée sur le numéro de caractères Unicode et peut contenir jusqu'à 65 000 glyphes.

Le format OpenType est une extension du format TrueType SFNT qui gère également les données des polices Adobe® PostScript® et des fonctions typographiques inédites.

Les noms de fichier des polices OpenType contenant des données PostScript possèdent l'extension .otf, tandis que les polices OpenType de type TrueType portent l'extension .ttf.

**Notion de Font proportionnelle**

Il existe des fonts proportionnels, comme l'Arial, où les différents caractères n'ont pas la même largeur (le i est plus étroit que le P) c'est plus joli.

Philippe en Arial

Philippe en Courier New

Par contre dans les fonts non proportionnels, comme le 'Courier New', tous les caractères ont même largeur. C'est parfois plus pratique quand on veut afficher des lignes qui concordent sur le plan alignement vertical (et qu'on ne veut pas utiliser les tabulations).

## VIII-Q - Grille ou Grid

Qu'utiliser pour afficher dans une 'Grille' (Grid), un tableau (type tableur avec des lignes et des colonnes) ?

Ici on affiche du texte directement dans les cellules SANS utiliser de liaison avec une base de données. On parle de **grille 'indépendante'**.

Il y a :

MsFlexGrid de VB6 ;

LameGrid et SourceGrid Shareware VB.Net ;

DataGrid VB.Net 2003 ;

DataGridView VB.Net 2005.

### VIII-Q-1 - Contrôles Freeware à télécharger, c'est du '.Net'

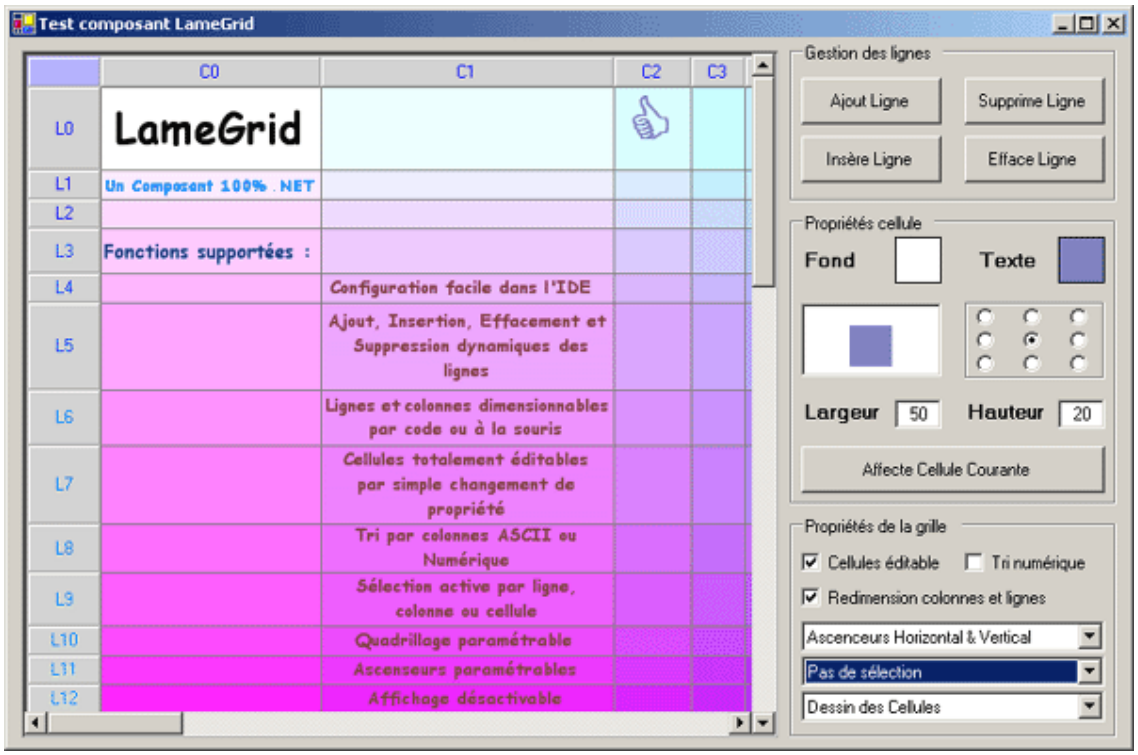
#### VIII-Q-1-a - 'LameGrid'en français +++++

Il existe un contrôle gratuit nommé lameGrid qui est du pur .Net et qui permet simplement d'afficher dans une grid.

On le trouve ici avec son mode d'emploi :

**https://kikos31.developpez.com/lamegrid/** par Christophe Holmes

C'est simple rapide, efficace. On le conseille.



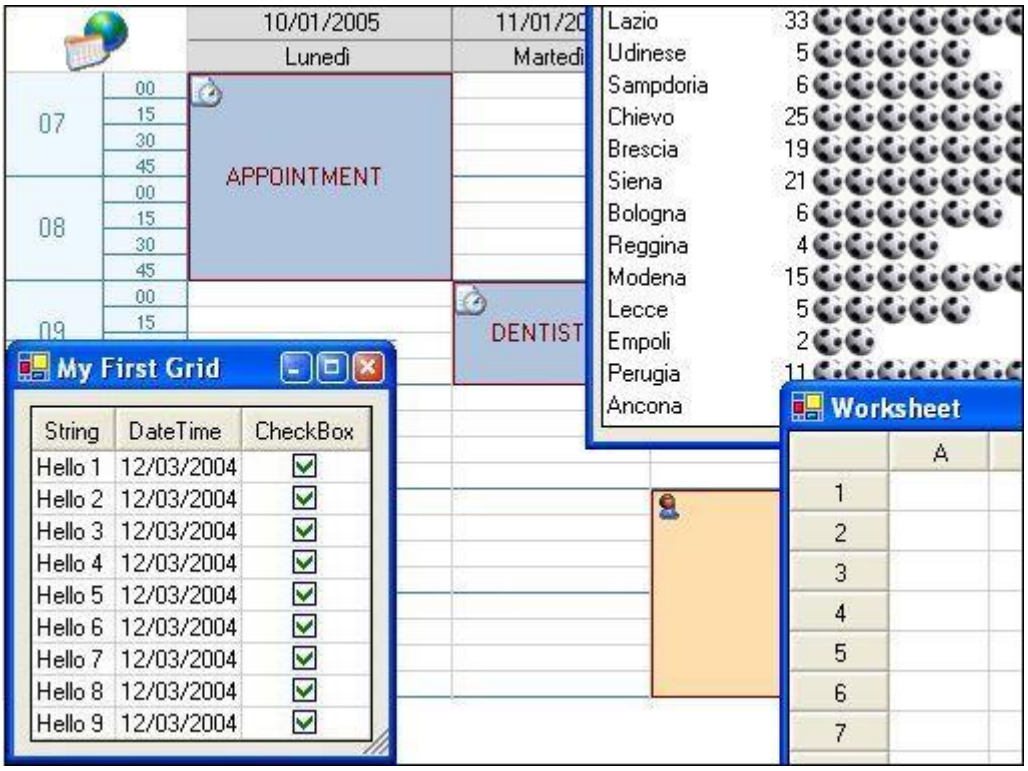

Son usage est simple :

```
Grille(1.2).Forecolor= MyColor

Grille(1.2).Font= MyFont

Grille(1.2).Texte="Lulu"
```

## VIII-Q-1-b - Autre

SourceGrid en anglais.

**http://www.devage.com/SourceGrid/SourceGrid2_EN.html**




## VIII-Q-2 - 'DataGridView' à partir de VB 2005

**C'est celui qu'il faut utiliser.**

Il remplace le DataGrid dans VB.Net 2005. Il est bien plus simple à utiliser surtout pour modifier directement la grille sans passer par un DataSet. (Contrôle indépendant.)

Exemple 1: on crée la Grid puis des colonnes de la grid ; on crée les lignes que l'on ajoute à la grille.

| Title | Artist | Album | Release Date | Track |
|---|---|---|---|---|
| Where Is My Mind? | Pixies | *Surfer Rosa* | 1988 | 7 |
| Scatterbrain. (As ... | Radiohead | *Hail to the Thief* | lundi 6 octobre 2003 | 13 |
| Revolution 9 | Beatles | *The Beatles [Whit...* | 11/22/1968 | 29 |
| One of These Days | Pink Floyd | *Meddle* | jeudi 11 novembre ... | 1 |
| Fools Rush In | Frank Sinatra | *Nice 'N' Easy* | 1960 | 6 |
| Dress | P J Harvey | *Dry* | 6/30/1992 | 3 |
| Can't Find My Mind | Cramps | *Psychedelic Jungle* | vendredi 1 mai 1981 | 9 |
| | | | | |

**MyDataGridView.ColumnCount = 5** indique le nombre de colonnes.

**MyDataGridView.Columns(0).Name = "Date"** met un texte dans le haut de la colonne.

**MyDataGridView.Rows.Add(t)** 'Ajout de ligne ; t est un tableau de 5 strings.

**MyDataGridView.CurrentCell** est la cellule courante (CurrentCellAdress contient les numéros de ligne et colonne).

**MyDataGridView.EditMode = DataGridViewEditMode.EditOnEnter** autorise de modifier les cellules.

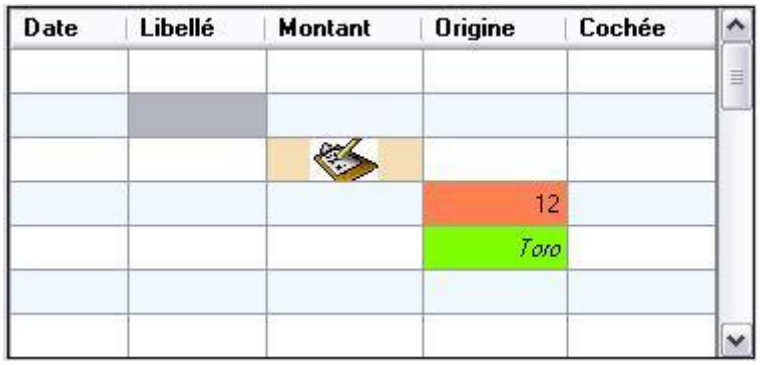
Exemple de Microsoft : afficher dans le contrôle MyDataGridView 5 colonnes (nommées date, piste, titre, artiste, album) et 6 lignes de chanson (exemple à partir d'un exemple de Microsoft).

```
'création de la grille

Private WithEvents MyDataGridView As New DataGridView

Me.Controls.Add(MyDataGridView)


'On met 5 colonnes

MyDataGridView.ColumnCount = 5


'On colore les entêtes, on met les fonts

With MyDataGridView.ColumnHeadersDefaultCellStyle

.BackColor = Color.Navy

.ForeColor = Color.White

.Font = New Font(MyDataGridView.Font, FontStyle.Bold)

End With


'on positionne la grille

With MyDataGridView

.Name = "MyDataGridView"

.Location = New Point(8, 8)

.Size = New Size(500, 250)

.AutoSizeRowsMode = DataGridViewAutoSizeRowsMode.DisplayedCellsExceptHeaders

.ColumnHeadersBorderStyle = DataGridViewHeaderBorderStyle.Single

.CellBorderStyle = DataGridViewCellBorderStyle.Single

.GridColor = Color.Black

.RowHeadersVisible = False


'On donne le nom des colonnes

.Columns(0).Name = "Date"

.Columns(1).Name = "Piste"

.Columns(2).Name = "Titre"

.Columns(3).Name = "Artiste"

.Columns(4).Name = "Album"

.Columns(4).DefaultCellStyle.Font = New Font(Me.MyDataGridView.DefaultCellStyle.Font,
 FontStyle.Italic)

.SelectionMode = DataGridViewSelectionMode.FullRowSelect

.MultiSelect = False
```

```
.Dock = DockStyle.Fill
End With



'Création d'un tableau de 5 strings pour chaque ligne
Dim row0 As String() = {"11/22/1968", "29", "Revolution 9", _
"Beatles", "The Beatles [White Album]"}
Dim row1 As String() = {"1960", "6", "Fools Rush In", _
"Frank Sinatra", "Nice 'N' Easy"}
Dim row2 As String() = {"11/11/1971", "1", "One of These Days", _
"Pink Floyd", "Meddle"}
Dim row3 As String() = {"1988", "7", "Where Is My Mind?", _
"Pixies", "Surfer Rosa"}
Dim row4 As String() = {"5/1981", "9", "Can't Find My Mind", _
"Cramps", "Psychedelic Jungle"}
Dim row5 As String() = {"6/10/2003", "13", _
"Scatterbrain. (As Dead As Leaves.)", _
"Radiohead", "Hail to the Thief"}
Dim row6 As String() = {"6/30/1992", "3", "Dress", "P J Harvey", "Dry"}
With Me.MyDataGridView.Rows
'Ajout de ligne
.Add(row0)
.Add(row1)
.Add(row2)
.Add(row3)
.Add(row4)
.Add(row5)
.Add(row6)
End With
With Me.MyDataGridView
'Ordre des colonnes
.Columns(0).DisplayIndex = 3
.Columns(1).DisplayIndex = 4
.Columns(2).DisplayIndex = 0
.Columns(3).DisplayIndex = 1
.Columns(4).DisplayIndex = 2
```

```
End With



'Ajouter une ligne
Me.MyDataGridView.Rows.Add()

'Enlever la ligne pointée
If Me.MyDataGridView.SelectedRows.Count > 0 AndAlso _
Not Me.MyDataGridView.SelectedRows(0).Index = _
Me.MyDataGridView.Rows.Count - 1 Then
Me.MyDataGridView.Rows.RemoveAt( _
Me.MyDataGridView.SelectedRows(0).Index)
End If

'Faire disparaitre toutes les lignes
Me.MyDataGridView.Rows.Clear()  'il ne reste plus que les entêtes de colonnes
```

Exemple 2 : on crée la grid avec des lignes et des colonnes puis on modifie les cellules.




```
'Mettre 5 colonnes et 50 lignes dans la grid
Grid.RowCount = 50
Grid.ColumnCount = 5

With Me.Grid



'Une ligne sur 2 en bleue
.RowsDefaultCellStyle.BackColor = Color.White
.AlternatingRowsDefaultCellStyle.BackColor = Color.AliceBlue
```

```
'Interdir la sélectionne de plusieurs cellules
.MultiSelect = False

'Empêche la saisie dans les cellules (en fait, le permet par programmation)
.EditMode = DataGridViewEditMode.EditProgrammatically
End With



'Gestion des entêtes de colonne
With Grid.ColumnHeadersDefaultCellStyle '
.BackColor = Color.Blue  'ça ne marche pas !!?? voir plus bas
.ForeColor = Color.Blue
.Font = New Font(Grid.Font, FontStyle.Bold)' en gras
End With

With Grid



'Empêche les modifications de lignes, colonnes, l'ajout, la suppression
.AllowUserToAddRows = False
.AllowUserToDeleteRows = False
.AllowUserToOrderColumns = False
.AllowUserToResizeColumns = False
.AllowUserToResizeRows = False



'Nomme les colonnes (en têtes)
.Columns(0).Name = "Date"
.Columns(1).Name = "Libellé"
.Columns(2).Name = "Montant"
.Columns(3).Name = "Origine"
.Columns(4).Name = "Cochée"
.RowHeadersVisible = False 'pas de première colonne d'en tête
.Columns(2).Width = 30 'modifie la largeur de la colonne 2
End With
```

La couleur des entêtes ne marche pas ? Il suffit de mettre la propriété **EnableHeadersVisualStyles à False** pour que le datagridview prenne en compte le style appliqué par code.

Pour avoir une 2 lignes d'entête :

```
.Columns(2).Name = "Montant" & ControlChars.CrLf & "en euros"

'(ColumnsHeaderHeightSizeMode est par défaut à AutoSize)



'On modifie la couleur de fond d'une cellule, on aligne au milieu, impose un format et affiche
 "12"

Grid.Item(3, 3).Style.BackColor = Color.Coral

Grid.Item(3, 3).Style.Alignment = DataGridViewContentAlignment.MiddleRight

Grid.Item(3, 3).Style.Format = "#####"

Grid.Item(3, 3).Value = 12



'On modifie la couleur de fond d'une cellule, on aligne au milieu,
' on met en italique et affiche "Toro"

Grid.Item(3, 4).Style.BackColor = Color.Chartreuse

Grid.Item(3, 4).Style.Alignment = DataGridViewContentAlignment.MiddleRight

Grid.Item(3, 4).Style.Font = New Font(Grid.Font, FontStyle.Italic)

Grid.Item(3, 4).Value = "Toro"


If Not Button1.Font.Style = FontStyle.Bold Then

Button1.Font = New Font(FontFamily.GenericSansSerif, _ 12.0F, FontStyle.Bold)

End If



'On force la cellule à accepter une image, on aligne au milieu, donne une couleur de fond
'et affiche une image à partir d'un fichier.

Grid.Item(2, 2) = New DataGridViewImageCell

Grid.Item(2, 2).Style.Alignment = DataGridViewContentAlignment.MiddleCenter

Grid.Item(2, 2).Style.BackColor = Color.Wheat

Grid.Item(2, 2).Value = New Bitmap("viconote.gif")




'On  autorise le redimensionnement  auto, marche pas?

Grid.AutoResizeColumns()



'Positionner la cellule courante, le curseur sur la cellule 1,1
```

```
Grid.Rows(1).Cells(1).Selected = True


'Connaitre la ligne et la colonne de la cellule  courante
Dim x As Integer = Grid.CurrentCellAddress.X
Dim y As Integer = Grid.CurrentCellAddress.Y




'Effacer le contenu de toutes les cellules de la grid
Grid.Rows.Clear()
Grid.RowCount = 50
Grid.ColumnCount = 5


'Modifier le ToolTipText (Petit rectangle jaune contenant un test qui apparait quand le curseur
 de
'la souris reste un moment sur une cellule)
Private Sub Grid_CellFormatting(ByVal sender As Object, ByVal e
_As System.Windows.Forms.DataGridViewCellFormattingEventArgs) Handles Grid.CellFormatting
Dim cell As DataGridViewCell = Grid(e.ColumnIndex, e.RowIndex)
cell.ToolTipText = "oui"
End Sub
```

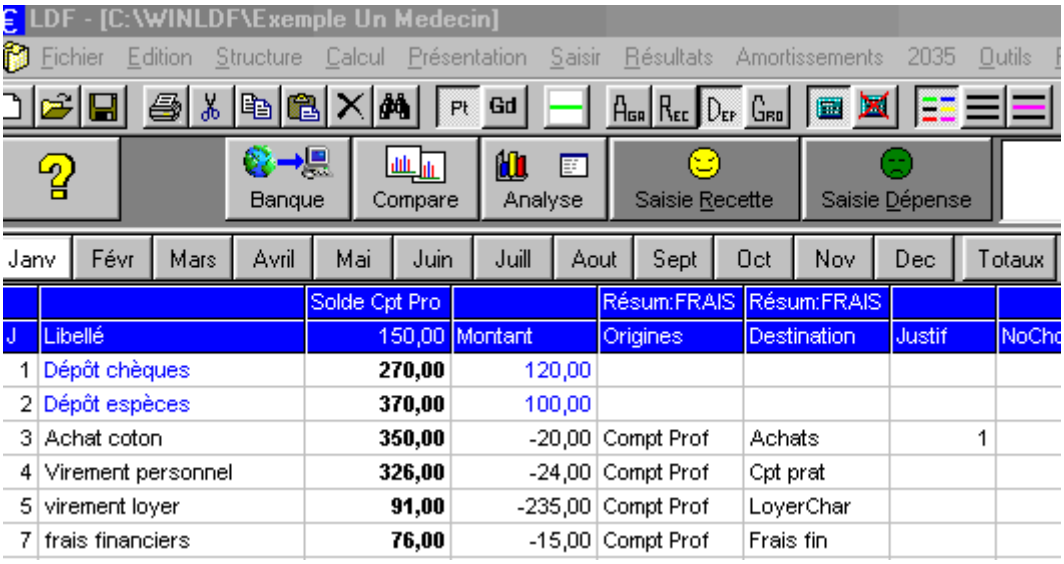
On rappelle que la première cellule en haut à gauche est la cellule '0,0'; on ne compte pas les entêtes.

## VIII-Q-3 - MsFlexGrid de VB6 et DataGrid de 2003 (pour mémoire)

Pour mémoire.
Microsoft fournissait avec VB6 l'activeX '**Microsoft Flexgrid 6**' qui permettait de satisfaire à la plupart des demandes.
Il est toujours possible d'utiliser cet activeX dans vos programmes, mais ce n'est plus du .net (c'est du non managé).

Il faut l'ajouter dans la boite à outils : bouton droit puis dans le menu 'Ajouter/Supprimer un composant' puis 'Parcourir', on ajoute MSFLXGRD.OCX qui est dans Windows/System32 (si vb6 installé, ou sinon le demander à quelqu'un qui a le CD VB6, mais il faut ensuite ouvrir le CD vb6 et cliquer sur c:\common\tools\vb\controls\vbctrls.reg).

Voilà ce qu'il permet de faire en VB6 : (Logiciel LDF de l'auteur).




| | | Solde Cpt Pro | | Résum:FRAIS | Résum:FRAIS | | |
|---|---|---|---|---|---|---|---|
| J | Libellé | 150,00 | Montant | Origines | Destination | Justif | NoCho |
| 1 | Dépôt chèques | **270,00** | 120,00 | | | | |
| 2 | Dépôt espèces | **370,00** | 100,00 | | | | |
| 3 | Achat coton | **350,00** | -20,00 | Compt Prof | Achats | 1 | |
| 4 | Virement personnel | **326,00** | -24,00 | Compt Prof | Cpt prat | | |
| 5 | virement loyer | **91,00** | -235,00 | Compt Prof | LoyerChar | | |
| 7 | frais financiers | **76,00** | -15,00 | Compt Prof | Frais fin | | |

Les propriétés **Cols** et **Rows** permettent de définir le nombre de colonnes et de lignes.

**FixedCols** et **FixedRows** permettent de déterminer les colonnes et lignes qui ne bougent pas (titres) ; BackColorFixed donne une couleur à ces lignes fixes.

Modifier la largeur d'une colonne :

```
Grid.ColWidth(i) =150

Pour modifier une cellule:

Grid.Row = 2         'Coordonnées de la cellule
Grid.Col = 3
Grid.CellFontBold = True     'Texte en gras
Grid.CellForeColor = Color.Red 'Couleur du texte

Grid.Text= Texte
```

ou

```
Grid.TextMatrix(2, 3) = Texte
```

.TextMatrix est beaucoup plus rapide que .Text, mais on n'a accès qu'au texte et pas à l'enrichissement.

Modifier la couleur de fond d'une cellule :

```
Grid.CellBackColor = Color.Red
```

Mettre une image dans une cellule :

```
Grid.CellPictureAlignment = flexAlignCenterCenter '4= gère l'alignement
Set Grid.CellPicture = ImageCoche.Picture    'Syntaxe VB6, le Set doit disparaitre en .Net
```

On peut gérer l'événement Grid_RrowColChanged quand l'utilisateur change de cellule. Il existe bien sur Grid_Click…

Pour accélérer l'affichage et éviter le scintillement, surtout s'il faut réafficher la totalité du tableau avec des couleurs et des images, il faut désactiver la mise à jour de l'affichage, afficher la page, réactiver. L'affichage devient instantané.

L'exemple suivant colore une ligne sur deux, c'est instantané.

```
Dim i As Integer

Dim j As Integer

Grid.Redraw = False
Grid.Clear

For i = 0 To NbMaxLigne Step 2
Grid.Row = i
For j = 0 To MaxColonne - 1
Grid.Col = j
Grid.CellBackColor = VERTCLAIR
Next j
Next i

Grid.Redraw = True
```

Il n'y a pas de gestion de **saisie dans les cellules**, il faut le faire 'à la main', Grid_KeyPress appelle une routine qui simule une saisie dans la grille avec un textbox qui prend les dimensions de la cellule.

Mettre dans un formulaire une grille MSFLEXGRID nommée Grid, une TextBox (avec borderSTyle =None) nommée TxtEdit.

Grid_KeyPress appelle une routine qui affiche le textbox (qui prend les dimensions de la cellule), l'utilisateur tape son texte dans le textbox, quand il sort, le textbox est effacé et le texte affiché dans la cellule de la grid.

AJOUTER DANS LES PROCÉDURES :

```
Private Sub Grid_DblClick()
If Txtedit.Visible = True Then Exit Sub 'evite une boucle
'edite
MSHFlexGridEdit Grid, Txtedit, 32 ' Simule un espace.
End Sub


Private Sub Grid_GotFocus()
If Txtedit.Visible = True Then
Grid = Txtedit
Txtedit.Visible = False
End If
End Sub


Private Sub Grid_KeyPress(KeyAscii As Integer)
MSHFlexGridEdit Grid, Txtedit, KeyAscii
End Sub



Private Sub Grid_RowColChange()
EditKeyCode Grid, Txtedit, 27, 0
End Sub


Private Sub Txtedit_KeyDown(KeyCode As Integer, Shift As Integer)
EditKeyCode Grid, Txtedit, KeyCode, Shift
End Sub
```

AJOUTER LES 3 routines :

```
Sub EditKeyCode(MSHFlexGrid As Control, Edt As Control, KeyCode As Integer, Shift As Integer)
' Traitement de contrôle d'édition standard.
Select Case KeyCode
Case 27 ' ÉCHAP : masque, renvoie le focus à MSHFlexGrid.
Edt.Visible = False
MSHFlexGrid.SetFocus
```

```
Case 13 ' ENTRÉE renvoie le focus à MSHFlexGrid.
Edt.Visible = False
MSHFlexGrid.SetFocus
MiseaJourLigne
Case 38 ' Haut.
MSHFlexGrid.SetFocus: DoEvents
Edt.Visible = False
MiseaJourLigne
If MSHFlexGrid.Row > MSHFlexGrid.FixedRows Then
MSHFlexGrid.Row = MSHFlexGrid.Row - 1
End If
Case 40 ' Bas.
MSHFlexGrid.SetFocus: DoEvents
Edt.Visible = False
MiseaJourLigne
If MSHFlexGrid.Row < MSHFlexGrid.Rows - 1 Then
MSHFlexGrid.Row = MSHFlexGrid.Row + 1
End If
Case 39 ' droit.
' MSHFlexGrid.SetFocus: DoEvents
' If MSHFlexGrid.Col < MSHFlexGrid.Cols Then
' MSHFlexGrid.Col = MSHFlexGrid.Col + 1
' End If
' Edt.Visible = False
' MiseAJourLigne
' Case 37 ' Gauche.
' MiseAJourLigne
' MSHFlexGrid.SetFocus: DoEvents
' If MSHFlexGrid.col > MSHFlexGrid.FixedCols - 1 Then
' MSHFlexGrid.col = MSHFlexGrid.col - 1
' End If
End Select
End Sub
```

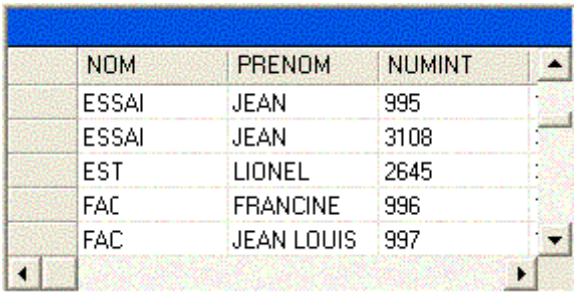
```
Sub MSHFlexGridEdit(MSHFlexGrid As Control, Edt As Control, KeyAscii As Integer)
' Utilise le caractère qui a été tapé.
Select Case KeyAscii
' Un espace signifie "modifier le texte en cours".
Case 0 To 32
Edt = Trim(MSHFlexGrid)
If Len(Edt) < 1 Then

Edt = Grid.Text
End If

Edt.SelStart = 1000
' Tout autre élément signifie "remplacer le ' texte en cours".
Case Else
Edt = Chr(KeyAscii)
Edt.SelStart = 1
End Select
' Affiche Edt au bon endroit.
Edt.Move MSHFlexGrid.Left + MSHFlexGrid.CellLeft, MSHFlexGrid.Top + MSHFlexGrid.CellTop,
                    _MSHFlexGrid.CellWidth - 8, MSHFlexGrid.CellHeight - 8
Edt.ForeColor = ROUGE
Edt.Visible = True
' Et laisse l'opération s'effectuer.
Edt.SetFocus
End Sub
```

```
Public Sub MiseaJourLigne()
'Met à jour la grid

Grid.text=Txtedit.text
End sub
```

## 'DataGrid' de VB 2003

Contrôle à éviter : utiliser 'DataGridView' à partir de VB 2005. C'est un des contrôles fournis avec VB.Net 2003 les plus puissants. Il est très adapté pour faire une liaison avec une base de données, mais pour l'utiliser simplement, dur, dur !!

Il est composé de lignes et de colonnes :

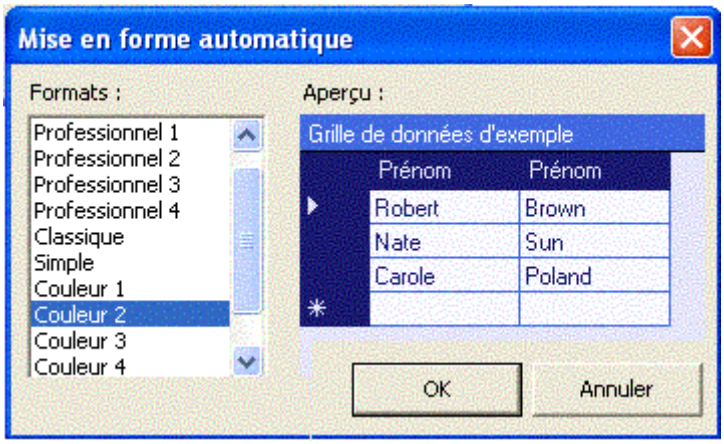



**Aspect du contrôle 'DataGrid'**

Mettre un 'DataGrid' dans le formulaire en cours en allant le chercher dans la boite à outils.

**On peut modifier l'aspect du DataGrid1 :**

- dans la fenêtre de propriété les propriétés ;
- en utilisant la mise en forme automatique (lien en bas de la fenêtre de propriétés).




**Pour travailler avec un DataGrid, on peut :**

- écrire directement dedans ;
- créer un DataSet (un DataSet c'est un objet qui a la structure d'une base de données, mais en local, il comporte des lignes, des colonnes… ). Ce DataSet sera ensuite lié au DataGrid par **DataGrid1.DataSource= MonDataSet**. Toute modification du Dataset sera ensuite répercutée automatiquement sur le DataGrid. Et toute modification du DataGrid sera répercutée sur le DataSet.

**Comment modifier le texte d'une cellule ?**

Pour modifier une cellule du Datagrid, il faut modifier le DataSet (pas le DataGrd) :

```
MonDataSet.Tables(0).Rows (0) (1)= "Montexte" '0 et 1 sont respectivement le numéro de ligne et
 de colonne.
```

**Comment lire le texte d'une cellule ?**

Lire ligne 1, colonne 1, l'afficher dans une TextBox :

```
TextBox1.Text = CType(DataGrid1(1, 1), String)

DataGrid1(1, 1) = TextBox1.Text
```

### Comment sélectionner une ligne ?

Il faut taper :

```
DataGrid1.Select(noligne) 'ligne est en bleue
DataGrid1.CurrentRowIndex = noLigne'selectionne vraiment
```

### Comment cacher une colonne ?

```
MonDataSet.Tables["Employees"].Columns["LastName"].ColumnMapping = MappingType.Hidden
```

### Comment améliorer la rapidité de l'affichage ?

Si on fait un grand nombre de modifications dans un DataGrid, le DataGrid est remis à jour à chaque modification, c'est long et souvent l'affichage clignote.

Pour éviter cela, il faut désactiver l'affichage par BeginUpdate, afficher toutes les modifications puis réactiver l'affichage par EndUpdate : la mise à jour se fait en une fois très rapidement.

```
Private Sub BeginEndUpdate()
' MyDataGridColumnStyle is a class derived from DataGridColumnStyle.
Dim dgc As MyDataGridColumnStyle
Dim dgCols As GridColumnStylesCollection
dgCols = DataGrid1.TableStyles(0).GridColumnStyles
For Each dgc In dgCols
dgc.BeginUpdate
Next

' Code to update not shown here.

For Each dgc In dgCols
dgc.EndUpdate
Next

End Sub
```

## VIII-R - ProgressBar

## VIII-R-1 - ProgressBar de VB 2003

Une progressBar permet de voir la progression d'une opération.

On donne une valeur aux propriétés **Minimum** et **Maximum**, la propriété **Value** permet de positionner la barre.

Souvent on utilise la ProgressBar différemment.

On donne une valeur aux propriétés Minimum et Maximum, on donne un pas (**Step**) ; la méthode **PerformStep()** augmente d'un pas.

Exemple de Microsoft

filenames() contient une liste de fichier à copier, à chaque fois qu'un fichier est copié, on avance la barre (qui se nomme MyBarre) :

```
Private Sub CopyAvecProgressBar(ByVal ParamArray filenames As String())

' Minimum à 1
MyBarre.Minimum = 1
' Maximum= nombre total de fichiers à copier.
MyBarre.Maximum = filenames.Length
' On initialise la ProgressBar.
MyBarre.Value = 1
' On indique le pas.
MyBarre.Step = 1

' Boucle de copie.
Dim x As Integer
for x = 1 To filenames.Length - 1
' Copier un fichier.
If CopyFile(filenames(x - 1)) = True Then
' Si la copie est OK incrémenter la ProgressBar.
      MyBarre.PerformStep()
End If
Next x
End Sub
```

## VIII-R-2 - ProgressBar de VB 2005

Fonctionne de la même manière.

```
MyBarre.Style = ProgressBarStyle.blocks    'indique d'avancer par bloc

MyBarre.Style = ProgressBarStyle.continuous 'indique d'avancer progressivement
```

On peut aussi, quand on ne connait pas la durée du processus, indiquer à la ProgressBar d'avancer de gauche à droite (comme lors de l'ouverture de Windows XP) :

```
MyBarre.Style = ProgressBarStyle.Marquee
```

## VIII-S - Créer des contrôles par code

**Dans le code, on peut créer soi-même de toutes pièces, des contrôles et leurs événements.**

## VIII-S-1 - Créer par code des contrôles

Dans le code d'une procédure, il est possible de créer de toute pièce un contrôle, mais attention, il faut tout faire !!

**Créons le bouton :**

```
Dim Button1 As New Button
```

Modifions ses propriétés :

```
Me.Button1.Location = New System.Drawing.Point(56, 144)

Me.Button1.Name = "Button1"

Me.Button1.Size = New System.Drawing.Size(104, 24)

Me.Button1.TabIndex = 0

Me.Button1.Text = "Button1"
```

Le bouton existe, mais il faut l'ajouter à la collection Controls de la fenêtre (cette collection contient tous les contrôles contenus dans la fenêtre) :

```
Me.Controls.Add(Button1)
```

## VIII-S-2 - Ajouter des événements

Le bouton existe, mais pour le moment, **il ne gère pas les événements.**

Il faut inscrire le bouton dans une méthode de gestion d'événements. En d'autres termes, Vb doit savoir quelle procédure événement doit être déclenchée quand un événement survient. Pour cela, il y a 2 méthodes :

* déclarer la variable avec le mot-clé WithEvents ce qui permet ensuite d'utiliser le Handles du contrôle dans la déclaration d'une Sub.

Déclaration **dans la partie déclaration du module**(en haut) (WithEvents n'est pas accepté dans une procédure) :

```
Private WithEvents Button1 As  New Button
```

Remarque Button1 est accessible dans la totalité du module.

Puis écrire la sub événement :

```
Sub OnClique ( sender As Object, EvArg As EventArgs) Handles Button1.Click

End Sub
```

Ainsi VB sait que pour l'événement Button1.Click, il faut déclencher la Sub OnClique.

Si on fait : Private WithEvents Button1 As Button (sans New) dans la partie déclaration puis DIM Button1 As New Button dans une procédure, la Sub OnClique ne fonctionne pas !!

C'est un problème de 'visibilité' donc bien faire Private WithEvents Button1 As New Button

Remarque : il pourrait y avoir plusieurs Handles sur une même sub, donc des événements différents sur des objets différents déclenchant la même procédure ;

* Utiliser AddHandler

Déclaration (possible dans une procédure) :

```
Dim Button1 As New Button
```

Puis écrire la gestion de l'événement.( L'événement Button1.click doit déclencher la procédure dont l'adresse est BouttonClique.)

```
AddHandler Button1.Click, AddressOf  BouttonClique
```

(Ne pas oublier la virgule avant AddressOf.)

Enfin on écrit la sub qui 'récupère l'événement :

```
Private Sub BouttonClique (sender As Object, evArgs As EventArgs)

End Sub
```

Ainsi VB sait que pour un événement du Button1, il faut déclencher la Sub ButtonClique

**Exemple avec AddHandler**

**Créons un TextBox nommé TB et une procédure déclenchée par KeyUp de ce TextBox.**

Dans une procédure (Button1_Click par exemple) : je crée un TextBox nommé TB, je le positionne, je mets dedans le texte 'ici une textbox'. Je l'ajoute aux Contrôles du formulaire.

Grâce à 'AddHandler', je lie l'événement Keyup de cet objet **TB** à la sub que j'ai créé : TextboxKeyup.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
 Button1.Click

    Dim TB As New System.Windows.Forms.TextBox

    TB.Location = New System.Drawing.Point(2, 2)

    TB.Text = "ici une textBox"

    Me.Controls.Add(TB)

    AddHandler TB.Keyup, AddressOf TextboxKeyup.

End sub


Sub TextboxKeyup.(ByVal sender As Object, ByVal e As KeyEventArgs)

…

End Sub
```

Si je crée un autre bouton TB2, j'ajoute de la même manière AddHandler TB2.Click, AddressOf TextboxKeyup2, ainsi chaque événement de chaque contrôle a ses propres routines événement et en cliquant sur le bouton TB2 on déclenche bien TextboxKeyup2.

Attention, la procédure TextboxKeyup doit recevoir impérativement les bons paramètres : un objet et un KeyEventArgs, car ce sont les paramètres retournés par un KeyUp.

**Autre exemple avec AddHandler, mais avec 2 boutons**

Il est possible de créer **plusieurs contrôles ayant la même procédure événement.**

Exemple : créons 2 boutons (BT1 et BT2) déclenchant une seule et même procédure (BoutonClique).

Dans ce cas, **comment savoir sur quel bouton l'utilisateur a cliqué ?**

En tête du module déclarons les boutons (ils sont **public**) :

```
Public BT1 As New System.Windows.Forms.Button

Public BT2 As New System.Windows.Forms.Button
```

Indiquons dans form_load par exemple la routine événement commune (BoutonClique) grâce à AddHandler.

```
Form_Load

    BT1.Location = New System.Drawing.Point(2, 2)

    BT1.Text = "Bouton 1"

    Me.Controls.Add(BT1)

    BT2.Location = New System.Drawing.Point(100, 100)

    BT2.Text = "Bouton 2"

    Me.Controls.Add(BT2)

    AddHandler BT1.Click, AddressOf BoutonClique

    AddHandler BT2.Click, AddressOf BoutonClique

End Sub
```

Si c'est le bouton 1 qui a été cliqué, afficher "button1" dans une TextBox :

```
Sub BoutonClique(ByVal sender As Object, ByVal e As EventArgs)

    If sender Is BT1 Then

      TextBox1.Text = "button 1"

    ElseIf sender Is BT2 Then

      TextBox1.Text = "button 2"

    End If

End Sub
```

La ruse est que déterminer quel objet (quel bouton) a déclenché l'événement, pour cela on utilise le premier paramètre, le sender :

```
If sender Is BT1 Then  ' Si le sender est le bouton1…
```

**Noter bien :**

- le mot-clé **Handles** permet d'associer un événement à une procédure **au moment de la conception.**

Le concepteur sait qu'une procédure doit gérer les événements (il peut y en avoir plusieurs) ;

- le mot-clé **Addhandler** permet d'associer un événement à une procédure **au moment de l'exécution**.

Ceci est utile dans un cadre producteur-consommateur d'événements. Un objet produit un événement qui doit informer d'autres objets ; au cours de l'exécution, on crée l'association entre l'événement et une procédure.

**Remarque importante**

Les Handler ne sont en fait libérés qu'au déchargement complet du programme (application.exit) et non à la fermeture de la fenêtre et des objets contenus dans celle-ci… Aussi, si vous ouvrez plusieurs fois un même formulaire possédant AddHandler sur un bouton, cela créera à chaque fois un Handler qui s'ajoute aux précédents et l'événement se déclenchera plusieurs fois lors de l'appui du bouton !!

Il faut donc utiliser RemoveHandler pour libérer le Handler. L'instruction s'utilise de la même façon que le AddHandler ! (Reprendre les lignes d'ajout du handler et remplacer AddHandler par RemoveHandler.)

## VIII-S-3 - Menu par code

Double-cliquez sur le composant **ContextMenu** dans la boite à outils pour l'ajouter sur le formulaire : cela crée un ContextMenu1.

Par code, on va le vider, puis ajouter des items (lignes) au menu, on indique le texte de l'item, mais aussi quelle routine déclencher lorsque l'utilisateur clique sur le menu contextuel :

```
' Vide le  context menu.
ContextMenu1.MenuItems.Clear()


' Ajoute une ligne 'Checked'.
ContextMenu1.MenuItems.Add("Ouvrir", New System.EventHandler(AddressOf Me.Ouvrir_Click))


' Ajoute une ligne 'Checked

ContextMenu1.MenuItems.Add("Fermer", New System.EventHandler(AddressOf Me.Fermer_Click))


' Test si le contrôle en cours est CheckBox, si oui ajout d'un item "Contrôler".
If ContextMenu1.SourceControl Is CheckBox1 Then
ContextMenu1.MenuItems.Add("Contrôler", New System.EventHandler(AddressOf Me.Controler_Click))
End If
```

Bien sûr, il faut écrire les Sub Ouvrir_Click() Fermer_Click Controler_Click.

En fonctionnement, l'utilisateur clique 'droit' sur un contrôle, le menu contextuel s'ouvre, il clique sur 'Ouvrir' ce qui exécute la routine Ouvrir_Click.

## VIII-T - Mise à jour et vitesse de l'affichage

**Mise à jour de l'affichage**

La mise à jour de l'affichage d'un Label (comme les autres contrôles d'ailleurs) est effectuée en fin de Sub.

Si on écrit :

```
Dim i As Integer

For i = 0 To 100

    Label1.Text = i.ToString

Next i
```

La variable i prend les valeurs 1 à 100, mais à l'affichage rien ne se passe pendant la boucle, VB affiche uniquement 100 à la fin ; si on désire voir les chiffres défiler avec affichage de 0 puis 1 puis 2… il faut rafraîchir l'affichage à chaque boucle avec la méthode Refresh() :

```
Dim i As Integer

For i = 0 To 100

    Label1.Text = i.ToString: Label1.Refresh()

Next i
```

Une alternative est de mettre un **Application.DoEvents()** qui donne à Windows le temps de traiter les messages et de rafraîchir l'affichage.

**Clignotement et lenteur d'affichage :** lorsqu'on modifie plusieurs propriétés visuelles d'un contrôle ou qu'on affiche dans une grille par exemple de nombreuses modifications, l'affichage est mis à jour après chaque modification: c'est long et cela clignote.
Pour remédier à cela, on suspend le moteur d'affichage, on fait toutes les modifications, on remet le moteur d'affichage. Les modifications visuelles sont instantanées.

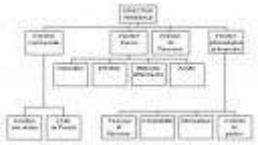
```
Me.SuspendLayout()
 …
 …
 …
Me.ResumeLayout(False)
```

Pour éviter le clignotement et accélérer l'affichage, on peut aussi utiliser BeginUpdate et EndUpdate sur un contrôle ListBox.

```
listBox1.BeginUpdate()
     ' On ajoute 50 éléments.
     Dim x As Integer
     For x = 1 To 50
         listBox1.Items.Add("Item " & x.ToString())
     Next x
     '
listBox1.EndUpdate()
```

# IX - Programmation procédurale

## IX-A - La programmation procédurale



En programmation 'procédurale' (pas en programmation objet) :

chaque problème complexe est décomposé en 'Fonctions' (les Sub et Fonctions) plus simples ;

ces fonctions sont stockées dans des modules standards (ou dans les modules de formulaire).

Dans une application en programmation 'procédurale' il y a habituellement :

- des modules de formulaires ;

- des modules standards contenant des Sub et Function.

Chaque fonction peut être appelée d'une autre fonction.

Exemple

Créons une Function nommée CalculCarré.

```
Public Function CalculCarré ( c As Single) As Single

    Return c*c

End Function
```

Cette fonction est Public (elle peut être appelée de n'importe où dans le programme).

Elle accepte un paramètre qui doit être un Single.

Comme c'est une fonction, elle retourne une valeur qui est aussi un Single.

Comment l'appeler ?

```
Dim carré As Single

carré= CalculCarré (12)
```

Une Sub par contre ne retourne pas de valeur.

```
Public Sub Affiche Carré ( c As Single)

…

End Sub
```

Comment l'appeler ?

```
AfficheCarré (12) ou Call AfficheCarré (12)
```

L'autre manière de programmer en VisualBasic est la programmation 'Objet'.

## IX-A-1 - Comment créer un module standard, une Sub ?

Faire Menu Projet>Ajouter un module. Donner un nom au module. C'est Module1.vb par défaut.

```
Module Module1        'Nom du Module

….

End Module
```

On remarque que le module est bien enregistré dans un fichier .vb.

Un module standard ne contient que du code.

Comment ajouter une Sub dans un module Standard ?

Taper Sub Calcul puis valider, cela donne :

```
Sub Calcul()

End Sub
```

## IX-A-2 - Exemple d'utilisation de procédures et de modules

Exemple simple de programmation procédurale.

L'utilisateur saisit un nombre puis il clique sur un bouton ; cela affiche le carré de ce nombre.

Il faut créer l'interface utilisateur : créer une fenêtre (Form1), y mettre un bouton (nommé Button1), une zone de texte (Text1) permettant de saisir un nombre, un label (label1) permettant l'affichage du résultat.

Créer un module standard (Module1) pour y mettre les procédures communes.

On observera uniquement l'agencement des procédures et non leur contenu. Pour un programme d'une telle complexité, la structure aurait pu être plus simple, mais l'intérêt de ce qui suit est didactique.

On décompose le programme en tâches plus simples : en particulier une procédure sert au calcul, une sert à l'affichage.

**La procédure CalculCarré** calcule le carré.

**La procédure AfficheCarre** affiche le résultat dans le label.

La procédure **Button1_Click** (qui est déclenchée par le Clic de l'utilisateur) :

lit le chiffre tapé par l'utilisateur dans la zone texte ;

appelle la procédure CalculCarré pour calculer le carré ;

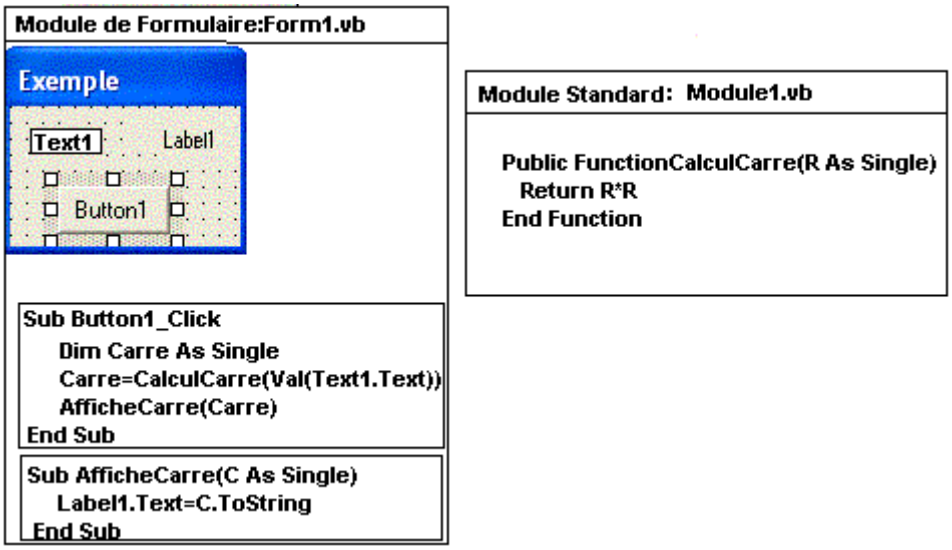
appelle la procédure AfficheCarré pour afficher le résultat.

Où sont placées les procédures ?

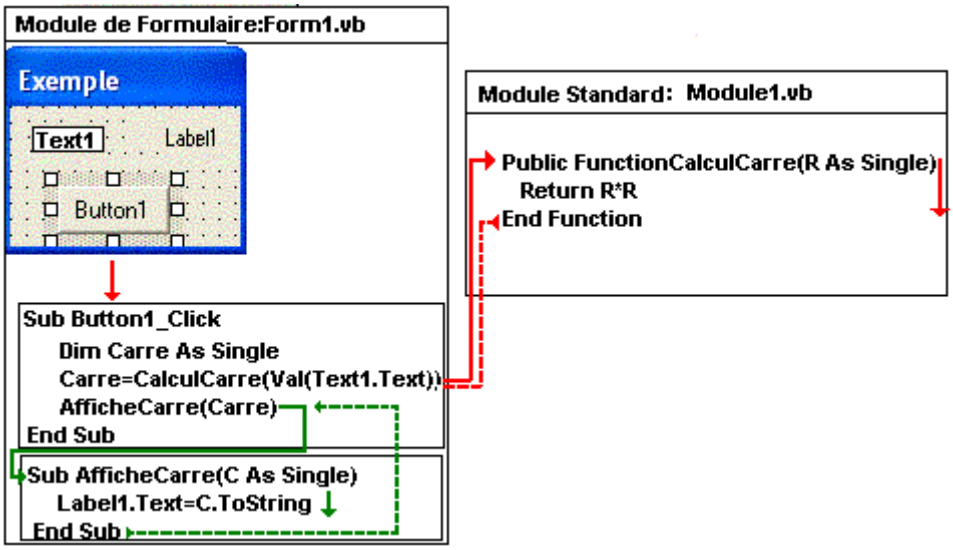
La procédure Button1_Click est automatiquement dans le module du formulaire, Form1 (elle est liée au contrôle Bouton1) elle est créée automatiquement quand on crée le bouton.

La procédure AfficheCarré est créée dans le module du formulaire (Form1), car elle agit sur le contrôle Label1 de ce formulaire.

La procédure CalculCarré est créée dans le module Standard (Module1), car elle doit être appelable de n'importe où ; elle est d'ailleurs 'Public' pour cette raison. Elle n'agit sur aucune fenêtre, aucun contrôle, elle est 'd'intérêt général', c'est pour cela qu'on la met dans un module standard.




Voyons le cheminement du programme :




Quand l'utilisateur clique sur le bouton la Sub Button1_Click démarre.

Elle appelle CalculCarre avec comme paramètre le nombre qui a été tapé dans le textbox (nommé Text1).

Val(Text1.Text) permet de transformer la String Text1.Text en numérique.

CalculCarre calcule le carré et renvoie la valeur de ce carré.

La Sub Button1_Click appelle ensuite AfficheCarre (en envoyant le paramètre Carré) qui affiche le résultat.

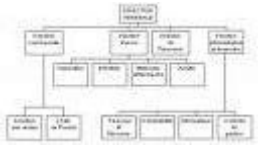
On remarque : on appelle la Function CalculCarre par :

```
Carre= CalculCarre(Valeur)
```

On envoie un paramètre Single, la fonction retourne dans la variable Carre, la valeur du carré.

Par contre la Sub AfficheCarre reçoit un paramètre, et ne retourne rien puisque c'est une Sub !!

## IX-B - Exemple : Calcul de l'IMC



Ce chapitre permet de 'réviser' pas mal de notions.

### IX-B-1 - Qu'est-ce que l'IMC ?

L'index de masse corporelle est très utilisé par les médecins. Il est calculé à partir du poids et de la taille :

**IMC=Poids/(Taille*Taille)** (avec Poids en kg, Taille en mètres)

Cela permet de savoir si le sujet est :

maigre (IMC inférieur à 18.5) ;

normal (IMC idéale=22) ;

en surpoids (IMC supérieur à 25) ;

obèse (IMC>30).

**On peut calculer le poids idéal** par exemple PI= 22* T*T

Nous allons détailler ce petit programme.

### IX-B-2 - Quel est le cahier des charges du programme ?

**L'utilisateur doit pouvoir** saisir un poids, une taille, cliquer sur un bouton 'Calculer'

Les routines doivent :

vérifier que l'utilisateur ne fait pas n'importe quoi ;

calculer et afficher les résultats : l'IMC, mais aussi, en fonction de la taille, le poids idéal, les poids limites de maigreur, surpoids, obésité.

## IX-B-3 - Création de l'interface

Il faut 2 zones de saisie pour saisir le poids et la taille :

on crée 2 'TextBox' que l'on nomme :

TexBoxPoids ;

TextBoxTaille.

On laisse la propriété Multiline à False pour n'avoir qu'une ligne de saisie.

Pour afficher les résultats, on crée 5 'label' les uns sous les autres. (Pour aller plus vite et que les labels soient de la même taille, on en crée un, puis par un copier et des coller, on crée les autres).

```
labelImc    'pour afficher l'Imc

labelPi     'pour afficher le poids idéal

labelM      'pour afficher le poids limite de la maigreur.

labelS      'pour afficher le poids limite du surpoids

labelO      'pour afficher le poids limite de l'obésité.
```

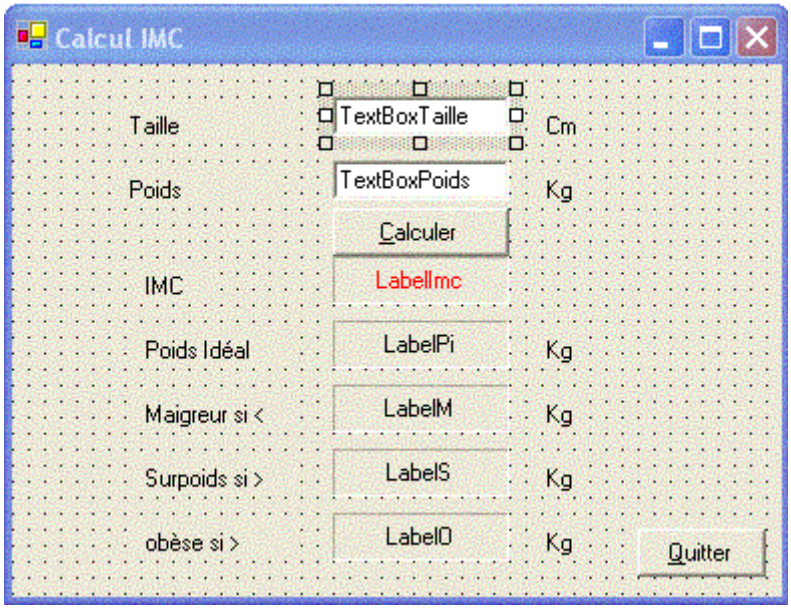
Ensuite on ajoute des labels devant et derrière chaque TextBox pour indiquer devant, ce qu'ils contiennent et derrière, l'unité.

On ajoute 2 boutons :

ButtonCalcul ayant pour propriété Text= "&Calculer" ;

ButtonQuitter ayant pour propriété Text= "&Quitter".

Cela donne :




Pour faire beau :

la propriété Text de la fenêtre contient "Calcul IMC", pour afficher cela dans la barre de titre ;

la propriété ForeColor de labelImc est en rouge ;

la propriété BorderStyle des labels a la valeur 'Fixed3d' ce qui rend les bords visibles.

Ajout du Code

La procédure événement **Form1_Load** qui se déclenche lorsque la fenêtre se charge initialise les zones d'affichage en les vidant :

```
Private Sub Form1_Load(…)

TextBoxTaille.Text = ""

TextBoxPoids.Text = ""

LabelImc.Text = ""

LabelPi.Text = ""

LabelM.Text = ""

LabelS.Text = ""

LabelO.Text = ""

End Sub
```

La procédure **ButtonCalcul_Click** qui se déclenche lorsque l'utilisateur clique sur le bouton 'Calculer' contient le code principal.

Voici la totalité du code, on le détaillera dessous.

```
Private Sub ButtonCalcul_Click(…)


Dim sPoids As Single  'Variable Single contenant le poids

Dim sTaille As Single 'Variable Single contenant la taille


'******contrôle de validité des entrées***********

'Les valeurs saisies sont-elles numériques ?

If Not (IsNumeric(TextBoxTaille.Text)) Then

    MsgBox("Entrez une valeur numérique pour la taille")

    Exit Sub

End If

If Not (IsNumeric(TextBoxPoids.Text)) Then

    MsgBox("Entrez une valeur numérique pour le poids")

    Exit Sub

End If


'Convertir les textes saisis en single
```

```
' et les mettre dans les variables

sTaille = CType(TextBoxTaille.Text, Single) / 100

sPoids = CType(TextBoxPoids.Text, Single)


'Les valeurs saisies sont-elles cohérentes?

If sTaille < 0.50 Or sTaille > 2.50 Then

    MsgBox("Entrez une taille valide")

    Exit Sub

End If

    If sPoids < 20 Or sPoids > 200 Then

    MsgBox("Entrez un poids valide")

Exit Sub

End If


'Effectuer les calculs et afficher les résultats.

LabelImc.Text = (Math.Round(sPoids / (sTaille * sTaille), 2)).ToString

LabelPi.Text = (Math.Round(22 * (sTaille * sTaille), 2)).ToString

LabelM.Text = (Math.Round(18.5 * (sTaille * sTaille), 2)).ToString

LabelS.Text = (Math.Round(25 * (sTaille * sTaille), 2)).ToString

LabelO.Text = (Math.Round(30 * (sTaille * sTaille), 2)).ToString

End Sub
```

Détaillons.

### Quelles sont les différentes étapes ?

- On déclare les variables.
- On vérifie que ce qui a été tapé est numérique.
- On convertit le texte qui est dans la TextBox en Single.
- On teste si les valeurs de poids et taille sont cohérentes.
- On fait le calcul et on affiche.

### Déclaration de variables :

```
Dim sPoids As Single  'Variable Single contenant le poids

Dim sTaille As Single 'Variable Single contenant la taille
```

Ce sont des variables 'privées' propres à la procédure (utilisation de Dim ou Private).

### Contrôle de validité

L'utilisateur est censé taper un poids et une taille puis cliquer sur le bouton 'Calculer'. Mais il ne faut absolument pas lui faire confiance : il a peut-être oublié de taper le poids ou à donner une taille=0 (l'ordinateur n'aime pas diviser par 0 !!), il a peut-être fait une faute de frappe et tapé du texte !!

Donc il faut tester ce qui a été tapé, s'il y a erreur, on prévient l'utilisateur avec une 'MessageBox' puis on sort de la routine par "Exit Sub" sans effectuer de calcul.

Ici par exemple, on teste si le texte saisi dans la zone taille n'est pas numérique :

```
If Not (IsNumeric(TextBoxTaille.Text)) Then

    MsgBox("Entrez une valeur numérique pour la taille")

    Exit Sub

End If
```

Amélioration : On aurait pu automatiquement effacer la valeur erronée et placer le curseur dans la zone à ressaisir :

```
If Not (IsNumeric(TextBoxTaille.Text)) Then

    MsgBox("Entrez une valeur numérique pour la taille")

    TextBoxTaille.Text=""

    TextBoxTaille.Select()

    Exit Sub

End If
```

### Conversion

Si le texte est bien 'Numéric', on fait la conversion en réel simple précision (Single) :

```
sTaille = CType(TextBoxTaille.Text, Single) / 100
```

On utilise CType pour convertir une String en Single.

On divise taille par 100, car l'utilisateur a saisi la taille en centimètres et les formules nécessitent une taille en mètre.

### Problème du séparateur décimal dans les saisies

Pourquoi saisir la taille en cm ? C'est pour éviter d'avoir à gérer le problème du séparateur décimal.

Si la taille était saisie en mètre, l'utilisateur aurait-il tapé "1.75" ou "1,75" ?

On rappelle que pour convertir un texte en Single VB accepte le point et pas la virgule.

Pour ma part, si j'avais demandé de saisir des mètres, voici ma solution : j'ajouterais en début de routine une instruction transformant les ',' en '.' :

```
TextBoxTaille.Text = Replace(TextBoxTaille.Text, ",", ".")
```

### Faire les calculs et afficher les résultats

Je fais le calcul :

```
sPoids / (sTaille * sTaille)
```

J'arrondis à 2 décimales après la virgule grâce à Math.Round( ,2) :

```
Math.Round(sPoids / (sTaille * sTaille), 2)
```

Je convertis en String :

```
(Math.Round(sPoids / (sTaille * sTaille), 2)).ToString
```

J'affiche dans le label 'labelImc' :

```
LabelImc.Text = (Math.Round(sPoids / (sTaille * sTaille), 2)).ToString
```

(J'aurais pu aussi ne pas arrondir le calcul, mais formater l'affichage pour que 2 décimales soient affichées).

La procédure ButtonQuitter_Click déclenchée quand l'utilisateur clique sur le bouton 'Quitter' ferme la seule fenêtre du projet (c'est Me, celle où on se trouve), ce qui arrête le programme.

```
Private Sub ButtonQuitter_Click()

    Me.Close()

End Sub
```

## IX-B-4 - Structuration

Ici on a fait simple : une procédure événement calcule et affiche les résultats.

On pourrait, dans un but didactique 'structurer' le programme.

On pourrait découper le programme en procédures.
Une procédure (une fonction) faisant le calcul.
Une procédure (une fonction) affichant les résultats.

**Si plusieurs procédures utilisent les mêmes variables, il y a dans ce cas 2 possibilités :**
**mettre les variables en 'Public' dans un module Standard ;**
**utiliser des variables privées et les passer en paramètres.**

**Première solution : Variables 'Public'.**
Créer dans un module standard des variables 'Public' pour stocker les variables Poids et Taille, résultats (Public sIMC A Single par exemple), créer dans ce même module standard une procédure Public nommée 'Calculer' qui fait les calculs et met les résultats dans les variables 'Public' ; enfin dans le module de formulaire créer une procédure 'AfficheResultat' affichant les résultats.

**Module standard :**

```
'Déclaration de variables Public

Public sPoids As Single

Public sTaille As Single

Public sIMC A Single

…

'Procedure Public de calcul

Public Sub Calculer

    sIMC=Math.Round(sPoids / (sTaille * sTaille), 2)
```

```
    ….
End Sub
```

**Module de formulaire Form1 :**

```
'Procédure événement qui appelle les diverses routines
Private Sub ButtonCalculer_Click
    ….
    sTaille = CType(TextBoxTaille.Text, Single) / 100
    Calculer()    'Appelle la routine de calcul
    AfficheResultat() 'Appelle la routine d'affichage
End Sub

'routine d'affichage toujours dans le formulaire
Private Sub AfficheResultat()

    LabelImc.Text = sIMC.ToString
    ….
End Sub
```

On voit bien que la routine de Calcul est générale et donc mise dans un module standard et d'accès 'Public', alors que la routine d'affichage affichant sur Form1 est privée et dans le module du formulaire.

**Seconde solution : Variables 'Privées' et passage de paramètres.**

On peut ne pas créer de variables 'public', mais créer des fonctions (CalculIMC par exemple) à qui on passe en paramètre le poids et la taille et qui retourne le résultat du calcul. Une procédure AfficheResultatIMC récupère en paramètre la valeur de l'IMC à afficher.

**Module standard :**

```
'Pas de déclaration de variables Public
…
'Function Public de calcul: reçoit en paramètre le poids et la taille
'retourne l'Imc
Public Function CalculerIMC (T as Single, P As Single) As Single
    Return Math.Round(P / (T*T), 2)
End Sub
```

**Module de formulaire Form1 :**

```
'Procédure événement qui appelle les diverses routines
Private Sub ButtonCalculer_Click
```

```
    ….
    sTaille = CType(TextBoxTaille.Text, Single) / 100

    'Appelle de la routine calcul avec l'envoi de paramètres sPoids et sTaille
    'Au retour on a la valeur de L'imc que l'on envoie à la routine d'affichage.
    AfficheResultatIMC(CalculerIMC(sTaille, sPoids)) 'Appelle la routine d'affichage
End Sub

'routine d'affichage
Private Sub AfficheResultatIMC(I As Single)
    LabelImc.Text = i.ToString
End Sub
```

Remarque

La ligne AfficheResultatIMC(CalculerIMC(sTaille, sPoids))

est équivalente à :

```
Dim s As single
s=(CalculerIMC(sTaille, sPoids)
AfficheResultatIMC(s))
```

mais on se passe d'une variable temporaire.

**Conclusion**

Faut-il travailler avec des variables Public ou passer des paramètres ?

Réponses

Les savants disent qu'il faut éviter les variables Public. Toutes les routines ayant accès à ces variables, il est toujours possible qu'une routine modifie une valeur sans qu'on le sache !!

*Utilisez donc des variables le plus privées possible.*

*(On y reviendra.)*

## IX-C - Exemple : Conversion francs/euros

Comment créer un programme de conversion francs=>euros et euros=> francs ?

Voici l'interface utilisateur :


Il y a une zone de saisie Euros, une zone Francs, si je tape dans la zone Euros '2', il s'affiche '13.12' dans la zone Francs ; cela fonctionne aussi dans le sens francs=>euros. On n'utilise pas de bouton pour déclencher le calcul ; le seul fait d'écrire dans un textBox déclenche le calcul et l'affichage des résultats.

Conseils

Un formulaire affichera les zones de saisie, un module standard contiendra les procédures de conversion.

On crée un formulaire Form1 contenant :

2 TextBox BoiteF et BoiteE, leurs propriétés Text="" ;

2 labels dont la propriété Text sera ="Euros" et "Francs", on les positionnera comme ci-dessus.

Un module Module1 contiendra 2 routines ConversionFE, ConversionEF.

Dans le formulaire, je dimensionne un **flag** (ou drapeau) : flagAffiche, il sera donc visible dans la totalité du formulaire. Je l'initialise à True.

```
Public Class Form1

Inherits System.Windows.Forms.Form

Dim flagAffiche As Boolean = True
```

Comme la conversion doit se déclencher automatiquement lorsque le texte de BoiteF ou BoiteE change, j'utilise les événements 'TextChanged' de ces TextBox.

Pour la conversion Euros=>Francs, dans la procédure TextChanged de BoiteE, je récupère le texte tapé (BoiteE.Text), j'appelle la fonction ConversionEF en lui envoyant comme paramètre ce texte. La fonction me retourne un double que je transforme en string et que j'affiche dans l'autre TextBox(BoiteF).

```
Private Sub BoiteE_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
_Handles BoiteE.TextChanged

If flagAffiche = True Then

flagAffiche = False

BoiteF.Text = (ConversionEF(BoiteE.Text)).ToString

flagAffiche = True

End If

End Sub
```

Idem pour l'autre TextBox :

```
Private Sub BoiteF_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs)
_Handles BoiteF.TextChanged
```

```
If flagAffiche = True Then

flagAffiche = False

BoiteE.Text = (ConversionFE(BoiteF.Text)).ToString

flagAffiche = True

End If

End Sub

End Class
```

À quoi sert le flag : flagAffiche ?

À éviter une boucle sans fin : sans flag, BoiteF_TextChanged modifie BoiteE_Text qui déclenche BoiteE_TextChanged qui modifie BoiteF_Text qui déclenche BoiteF_TextChanged…

Avec le flag, quand je vais modifier la propriété Text d'une TextBox, je mets le flag à False, cela indique à l'autre événement TextChanged de ne pas lui aussi convertir et afficher.

Enfin il faut écrire les procédures qui font la conversion : ConversionEF et ConversionFE dans un module standard. Ces procédures 'Function' appellent elles-mêmes une autre fonction qui arrondit les résultats à 2 décimales.

Pour transformer des euros en francs, je les multiplie par 6.55957 puis j'arrondis.

On remarque que ces procédures reçoivent une string en paramètre et retourne un double.

```
Module Module1

Public Function ConversionEF(ByVal e As String) As Double

Dim somme As Double

Dim resultat As Double

somme = Val(e)

resultat = Arrondir(somme * 6.55957)

Return resultat

End Function


Public Function ConversionFE(ByVal e As String) As Double

Dim somme As Double

Dim resultat As Double

somme = Val(e)

resultat = Arrondir(somme / 6.55957)

Return resultat

End Function
```

Enfin la Function Arrondir arrondit à 2 décimales : pour cela on multiplie par 100, on arrondit à l'entier avec Round puis on divise par 100.

```
Public Function Arrondir(ByVal Valeur As Double) As Double

'arrondi a 2 chiffres après la virgule

Return (Math.Round(Valeur * 100)) / 100

End Function

End Module
```

À noter que l'on aurait pu utiliser une surcharge de Round qui arrondit directement à 2 décimales :

```
Return (Math.Round(Valeur, 2))
```

**Exercice**

Quel code mettre dans la procédure Button_Click d'un bouton nommé 'Remise à zéro' qui met les 2 zones de saisie à zéro ?

(Penser au flag.)

**Amélioration**

Si l'utilisateur tape une virgule il y a problème, car la fonction Val utilisée pour convertir le nombre saisi en numérique reconnaît uniquement le point, il faut donc transformer les virgules en points avec

```
e = Replace(e, ",", ".")
```

On peut tester si l'utilisateur a bien tapé un nombre, avec la fonction IsNumeric.

## IX-D - Exemple : Calcul d'un prêt (les fonctions financières de VB)

Comment créer un programme qui calcule les mensualités d'un prêt ?

Dans l'espace Microsoft.VisualBasic, il existe des fonctions financières. (VB 2003 et VB 2005)

**Pmt** calcule les mensualités d'un prêt.

Remboursement mensuel= Pmt( Rate, NPer, PV, FV, Due)

| | |
|---|---|
| Rate | Obligatoire. Donnée de type Double indiquant le taux d'intérêt par période. Si taux d'intérêt annuel de 10 pour cent et si vous effectuez des remboursements mensuels, le taux par échéance est de 0,1/12, soit 0,0083. |
| NPer | Obligatoire. Donnée de type Double indiquant le nombre total d'échéances. Par exemple, si vous effectuez des remboursements mensuels dans le cadre |

| | |
|---|---|
| | d'un emprunt de quatre ans,il y a 4 * 12 (soit 48) échéances. |
| PV | Obligatoire. Double indiquant la valeur actuelle. Par exemple, lorsque vous empruntez de l'argent pour acheter une voiture, le montant du prêt correspond à la valeur actuelle (pour un emprunt il est négatif). |
| FV | Facultatif. Double indiquant la valeur future ou le solde en liquide souhaité au terme du dernier remboursement. Par exemple, la valeur future d'un emprunt est de 0 F, car il s'agit de sa valeur après le dernier remboursement. Par contre, si vous souhaitez économiser 70 000 F sur 15 ans, ce montant constitue la valeur future. Si cet argument est omis, 0 est utilisée par défaut. |
| Due | Facultatif. Objet de type Microsoft.VisualBasic.DueDate indiquant la date d'échéance des paiements. Cet argument doit être DueDate.EndOfPeriod si les paiements sont dus à terme échu ou DueDate.BegOfPeriod si les paiements sont dus à terme à échoir (remboursement en début de mois). Si cet argument est omis, DueDate.EndOfPeriod est utilisé par défaut. |

Noter que si Rate est par mois NPer doit être en mois ; si Rate est en années, NPer doit être en années.

```
Sub CalculPret()
Dim PVal, Taux, FVal, Mensualite, NPerVal As Double
Dim PayType As DueDate

Dim Response As MsgBoxResult
Dim Fmt As String

Fmt = "###,###,##0.00" ' format d'affichage.
FVal = 0  '0 pour un prêt.


PVal = CDbl(InputBox("Combien voulez-vous emprunter?"))
Taux = CDbl(InputBox("Quel est le taux d'intérêt annuel?"))
If Taux > 1 Then Taux = Taux / 100 ' Si l'utilisateur à tapé 4 transformer en 0.04.
NPerVal =12* CDbl(InputBox("Durée du prêt (en années)?"))
Response = MsgBox("Echéance en fin de mois?", MsgBoxStyle.YesNo)
If Response = MsgBoxResult.No Then
  PayType = DueDate.BegOfPeriod
Else
  PayType = DueDate.EndOfPeriod
End If
Mensualite = Pmt(Taux / 12, NPerVal, -PVal, FVal, PayType)
MsgBox("Vos mensualités seront de " & Format(Mensualite, Fmt) & " par mois")
End Sub



IPmt calcul les intérêts pour une période.
```

```
Calculons le total des intérêts:

Dim IntPmt, Total, P As Double

For P = 1 To TotPmts ' Total all interest.
IntPmt = IPmt(APR / 12, P, NPerVal, -PVal, Fval, PayType)
Total = Total + IntPmt
Next Period
```

Autres mots-clés :

| | |
|---|---|
| Calculer l'amortissement. | DDB, SLN, SYD |
| Calculer la valeur future. | FV |
| Calculer le taux d'intérêt. | Rate |
| Calculer le taux de rendement interne. | IRR, MIRR |
| Calculer le nombre de périodes. | NPer |
| Calculer les paiements. | IPmt, Pmt, PPmt |
| Calculer la valeur actuelle. | NPV, PV |

Par exemple :
Rate permet de calculer le taux d'un prêt en connaissant la somme prêtée, le nombre de mois et la mensualité.

## IX-E - Ordre des instructions dans un module : résumé

Contenu des modules.

Dans quel ordre écrire les instructions, options, énumérations, Class, Sub dans un module ?

Le code Visual Basic est stocké dans des modules (modules de formulaire, modules standards, modules de classe…), chaque module est dans un fichier ayant l'extension '.vb". Les projets sont composés de plusieurs fichiers '.vb', lesquels sont compilés pour créer des applications.

**Respecter l'ordre suivant :**

1 Instructions **Option** toujours en premier. (Force des contraintes de déclaration de variables, de conversion de variables, de comparaison.) ;
2 Instructions **Imports** (charge des espaces de nom) ;
3 Les **énumérations**, les **structures** 'générales' ;
4 Instructions **Class, Module et Namespace**, le cas échéant ;
5 En haut de la classe du module les énumérations et structures 'locales' ;
6 Les **Subs** et **Functions**.

Exemple1: Un Formulaire.

```
Option Explicit On      'Toujours en premier.


Imports System.AppDomain

Imports Microsoft.VisualBasic.Conversion


Enum Fichier            'Ici une énumération utilisable dans la totalité du programme

    Doc
```

```
    Rtf
End Enum


Public Class Form1          'la classe, le moule du formulaire
Inherits System.Windows.Forms.Form
    Dim WithEvents m As PrintDocument1
#Region " Code généré par le Concepteur Windows Form"


Public Sructure MaStructure  'Structure utilisable dans la Classe uniquement.
    i As Integer
    J As Integer
End Structure
Public d As Integer


Private Sub Form1_Load(….) Handles Form.load        'Ici une Sub
    Dim A As integer
    ….
End Sub
End Class
```

On remarque de nouveau l'importance de l'endroit où les variables sont déclarées. Dans notre exemple A est accessible uniquement dans Form_Load, alors que d est public.

Exemple2: Un module standard Module2.

```
Imports System.Activator
Enum MyEnum  'Ici une énumération utilisable dans la totalité du programme
    Toto
    titi
End Enum
Structure MyStructure  'Ici une structure utilisable dans la totalité du programme
    Dim i As Integer
End Structure


Module Module2
Sub Main()
End Sub
End Module
```

On remarque donc que Option et Imports sont toujours avant Class et Module.

La position de Enum et Structure (avant ou après les mots Class et module) gère leur visibilité.

*Si vous entrez les instructions dans un ordre différent, vous risquez de créer des erreurs de compilation.*

# X - Faire un vrai programme Windows Forms : ce qu'il faut savoir

## X-A - Démarrer, arrêter un programme : Sub Main(), fenêtre Splash

Quand vous démarrez votre programme, quelle partie du code va être exécutée en premier ?

**En Vb 2003**

Vous pouvez le déterminer en cliquant sur le menu Projet puis Propriétés de NomduProjet, une fenêtre Page de propriétés du projet s'ouvre.

Sous la rubrique Objet du démarrage, il y a une zone de saisie avec liste déroulante permettant de choisir :

- le nom d'un formulaire du projet ;

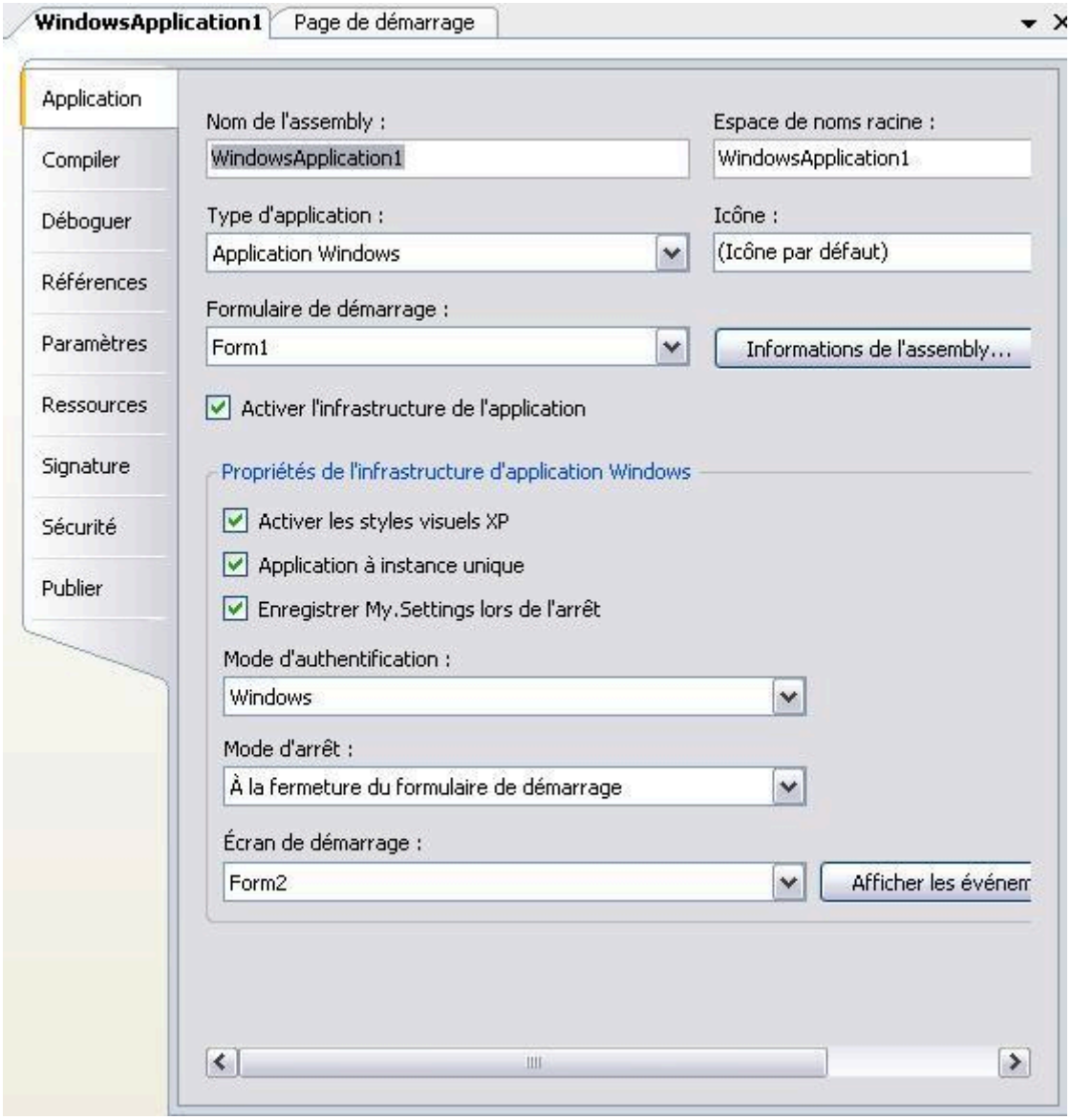
ou

- Sub Main().

**À partir de Vb 2005 (Framework 2) :**

ouvrir le 'Projet Designer', il est directement accessible dans l'explorateur de solution (double-cliquer sur 'My Projet') ou par le menu Projet-> Propriétés de …:

On définit : le Formulaire de démarrage (startUp Form).




Si 'Activer l'infrastructure de l'application' est coché, l'élément de démarrage ne peut être qu'un formulaire ; s'il est décoché, on peut lancer le programme par la Sub Main().

**Attention :** ne pas confondre Formulaire de démarrage en haut et Écran de démarrage (écran splash) en bas.

## X-A-1 - Démarrer par un formulaire

Si vous tapez le nom d'un formulaire du projet, c'est celui-ci qui démarre : cette fenêtre est chargée au lancement du programme et la procédure Form_Load de cette fenêtre est effectuée.

En théorie, si vous avez une application avec un formulaire, le fait de dessiner ce formulaire crée une Classe Form1 ; il faudrait donc théoriquement créer une instance de ce formulaire (par un Dim MyForm As New Form1) pour lancer l'application.

En pratique, dessinez un formulaire, lancez l'exécution, ça marche, car le runtime crée une instance du formulaire automatiquement à l'aide de sa méthode New et l'affiche (sans que l'on ait besoin de l'instancier soi-même).

## X-A-2 - Démarrer par Sub Main()

C'est cette procédure Sub Main qui s'exécute en premier lorsque le programme est lancé.

Elle peut servir à ouvrir le formulaire de démarrage.

**Exemple 1**

En mode design Form1 a été dessinée, c'est un modèle 'une Classe'.

Dans un module standard, dans une Sub Main(), on instancie initForm à partir de la Class Form1. Puis on affiche ce formulaire (cette fenêtre) avec .ShowDialog

```
Sub Main()

        Dim initForm As New Form1

        initForm.ShowDialog()

End Sub
```

**Exemple 2**

```
Sub Main()
' Démarre l' application et affiche une instance de Form1
Application.Run(New Form1())
End Sub
```

S'il y a plusieurs threads, Application.Run commence à exécuter une boucle de messages d'application standard sur le thread en cours et affiche le formulaire spécifié. Peut être utilisé aussi s'il y a un seul thread.



*Attention Sub Main() peut se trouver dans une Classe (y compris une classe de formulaire) ou dans un module.*

Si vous déclarez la procédure Main dans une classe, vous devez utiliser le mot-clé **Shared**.

```
Class Form1

Public Shared Sub Main()

    …

End Sub

…

End Classe
```

Dans un module, la procédure Main n'a pas besoin d'être partagée (Shared).

```
Module1

Sub Main()

….

End Sub

End Module
```

### Fonction Main()

On peut utiliser 'Function Main' (au lieu de 'Sub Main') qui retourne un Integer, que le système d'exploitation utilise comme code de sortie du programme. D'autres programmes peuvent tester ce code en examinant la valeur ERRORLEVEL Windows.

```
Function Main() As Integer
….
Return 0 ' Zéro signifie : tout est OK.
End Function
```

### Récupération de la ligne de commande

Main peut également avoir comme argument un tableau de String. Chaque élément du tableau contient un des arguments de ligne de commande utilisée pour appeler le programme. Vous pouvez réaliser diverses actions en fonction de leurs valeurs.

```
Function Main(ByVal CmdArgs() As String) As Integer
….

Return 0
End Function
```

On rappelle qu'en VB2005, si 'Activer l'infrastructure de l'application' est coché dans les propriétés du programme, le formulaire de démarrage ne peut être qu'un formulaire ; s'il est décoché, on peut lancer le programme par la Sub Main().

### Autre méthode de récupération de la ligne de commande en VB 2005

On trouve les arguments de la ligne de commande dans **My.Application.CommandLineArgs** (VB 2005)

Exemple

Cliquez sur un fichier de données, l'exécutable lié s'exécute et ouvre le fichier de données.

Exemple : quand on clique sur un fichier .bmp on lance automatiquement Paint qui charge l'image .bmp.

Il faut que l'extension du fichier soit liée avec le programme exécutable, si vous cliquez sur le fichier de données, cela lance l'exécutable.

Modifier l'extension liée Explorer->Outils-> Option des dossiers-> Type de fichiers.

Dans Form_Load mettre :

```
If My.Application.CommandLineArgs.ToString <> "" Then

Dim i

For i = 0 To My.Application.CommandLineArgs.Count - 1

If mid(My.Application.CommandLineArgs(i).ToString,1,2)  "-o" Then
' dans le cas ou la ligne de commande contient le nom du fichier à lancer et '-o'

FileName = Mid(My.Application.CommandLineArgs(i).ToString, 3)

OpenFile() ' charger les données

Exit For

End If
```

```
Next

End If
```

## X-A-3 - Fenêtre Splash

C'est une fenêtre qui s'ouvre au démarrage d'un programme, qui montre simplement une belle image, (pendant ce temps le programme peut éventuellement initialiser des données, ouvrir des fichiers…) ensuite la fenêtre 'Splash' disparait et la fenêtre principale apparait.


**En Vb 2003 (Framework 1) il faut tout écrire**

Dans la Sub Main il est possible de gérer une fenêtre Splash.

Exemple

Je dessine Form1 qui est la fenêtre Spash.

Dans Form2 qui est la fenêtre principale, j'ajoute :

Public Shared Sub Main()

```
Dim FrmSplash As New Form1     'instance la fenêtre Splash

Dim FrmPrincipal As New Form2 'instance la feuille principale

FrmSplash.ShowDialog()         'affiche la fenêtre Splash en Modale


FrmPrincipal.ShowDialog()      'a la fermeture de Splash,  affiche la fenêtre principale

End Sub
```

Dans Form1 (la fenêtre Splash)

```
Private Sub Form1_Activated

Me.Refresh() 'pour afficher totalement la fenêtre.

'ici ou on fait plein de choses, on ouvre des fichiers ou on perd du temps.

' s'il n'y a rien a faire on met un Timer pour que l'utilisateur admire la belle image.

Me.Close()

End Sub
```

On affiche FrmSplash un moment (Ho ! la belle image) puis on l'efface et on affiche la fenêtre principale. Word, Excel… font comme cela.

**Autre méthode :**

```
Public Sub main()

'création des formulaires frmmain and frmsplash avec le designer
Dim frmsplash As New frmsplash
Dim frmmain As New frmmain

'on affiche la Splash
frmsplash.Show()
Application.DoEvents()

'On attend (3000 milliseconds)
System.Threading.Thread.Sleep(3000)

'On efface la Splash
frmsplash.Close()

'On affiche le formulaire principal
Application.Run(frmmain)

End Sub
```

**En Vb 2005 (Framework 2) c'est très simple**

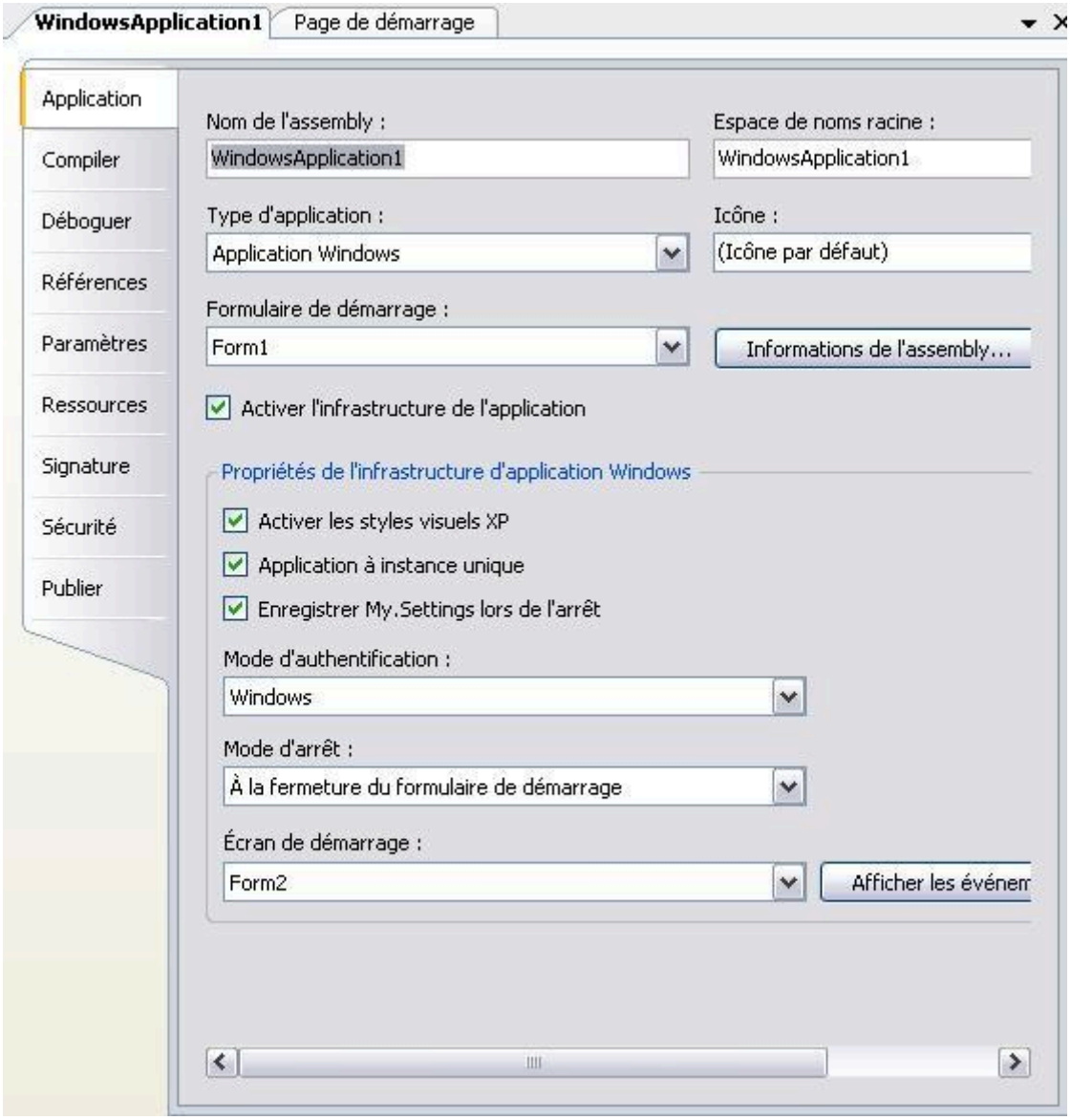
Ouvrir le 'Projet Designer', il est directement accessible dans l'explorateur de solution (My Projet)ou par le menu Projet-> Propriétés de…:

Il faut que 'Activer l'infrastructure de l'application' soit coché.

On définit :

le formulaire de démarrage (startUp Form) ;

l'écran de démarrage (Splash Screen), il suffit d'indiquer son nom (en mode Run, VB l'affiche et le fait disparaitre quand le formulaire de démarrage s'ouvre).




On peut aussi ajouter un écran splash tout fait :

Menu Projet, Ajouter un formulaire Windows, double-cliquer sur 'formulaire de démarrage'.

On obtient :


Le nom de l'application, la version, le nom de la société sont automatiquement mis à jour en utilisant les 'Informations de l'assembly' accessible par un bouton situé dans le projet designer, en face du nom du formulaire de démarrage.

L'inconvénient de cet écran Splash automatique est qu'il s'affiche et s'efface très rapidement, avant de charger le formulaire de démarrage !! Pour le voir une seconde, j'ai ajouté à la fin de la procédure Form_Load de cet écran :

```
Me.Show()

Application.DoEvents()

System.Threading.Thread.Sleep(1000)
```

Autre solution : utiliser **My.Application.MinimumSplashScreenDisplayTime**, qui détermine le temps d'affichage en ms. J'ai eu du mal à trouver où mettre l'instruction (dans le formulaire Splash ou Application_StartUp cela ne fonctionne pas !!
Il faut mettre la ligne dans Application New (propriété du projet, onglet application, bouton 'Afficher les événements de l'application", Liste déroulante à gauche 'MyApplication', liste déroulante à droite 'New'); rajouter la dernière ligne du code ci-dessous.

```
Partial Friend Class MyApplication

        <Global.System.Diagnostics.DebuggerStepThroughAttribute()>  _
        Public Sub New()

  MyBase.New(Global.Microsoft.VisualBasic.ApplicationServices.AuthenticationMode.Windows)
            Me.IsSingleInstance = false
            Me.EnableVisualStyles = true
            Me.SaveMySettingsOnExit = true

  Me.ShutDownStyle = Global.Microsoft.VisualBasic.ApplicationServices.ShutdownMode.AfterMainFormCloses
        End Sub

        <Global.System.Diagnostics.DebuggerStepThroughAttribute()>  _
        Protected Overrides Sub OnCreateMainForm()
            Me.MainForm = Global.WindowsApplication1.Form1
        End Sub

        <Global.System.Diagnostics.DebuggerStepThroughAttribute()>  _
        Protected Overrides Sub OnCreateSplashScreen()
            Me.SplashScreen = Global.WindowsApplication1.SplashScreen1
            My.Application.MinimumSplashScreenDisplayTime = 2000 '<= À rajouter
        End Sub
    End Class
```

## X-A-4 - Comment arrêter le programme ?

```
Me.Close()     'Ferme la fenêtre en cours
```

Noter bien Me désigne le formulaire, la fenêtre en cours.

```
Application.Exit()     'Ferme l'application
```

Vide la 'pompe à messages', ferme les formulaires. Si des fichiers sont encore ouverts, cela les ferme. (Il vaut mieux les fermer avant, intentionnellement.)

On peut aussi utiliser l'instruction **End**.

## X-A-5 - Fin de programme : Attention !

Outre l'usage de **Application.Exit()**, on peut terminer une application en fermant les formulaires, mais :

**dans Visual Basic 6.0**, une application ne se terminait que lorsque tous les objets créés étaient détruits ;

**dans Visual Basic .NET 2003**, l'application se termine lorsque l'objet de démarrage est détruit. Si le formulaire que vous fermez est le formulaire de démarrage de votre application, votre application se termine. Si la procédure Sub_Main est définie comme objet de démarrage l'application se termine dès que le code de Sub_Main a fini de s'exécuter.

Depuis VB 2005 vous avez le choix entre les 2 solutions : terminer l'application quand le formulaire de démarrage est fermé ou quand tous les formulaires sont fermés (dans l'application Designer voir 'Mode d'arrêt').

## X-B - Ouvrir plusieurs formulaires

Comment à partir d'un formulaire 'Form1' ouvrir un second formulaire à partir de la Classe 'Form2' ?

Voici le plan du chapitre :

- En VB2003

- En VB 2005

- Formulaire modal et non modale.

- Comment se nomment les formulaires?

- Autres

- Un formulaire est un objet

-E xemple

- DialogResult

- Bouton par défaut

## X-B-1 - Créer un formulaire en VB 2003

**A- Il faut d'abord créer la Classe Form2**

Ajoutez un formulaire (Menu Projet, Ajouter un formulaire au projet), nommez-le 'Form2'.

On se rend compte que quand on ajoute un formulaire (Form2 par exemple), on crée une nouvelle 'classe'.

'Class Form2' qui hérite de System.Windows.Forms.Form , elle hérite donc de toutes les propriétés et méthodes de la Classe Form qui est la classe 'formulaire'.

```
Public Class Form2

Inherits System.Windows.Forms.Form

End Class
```

Elle contient du code généré automatiquement par le concepteur Windows Forms et les procédures liées aux événements.

Dessinez dans Form2 les contrôles nécessaires.

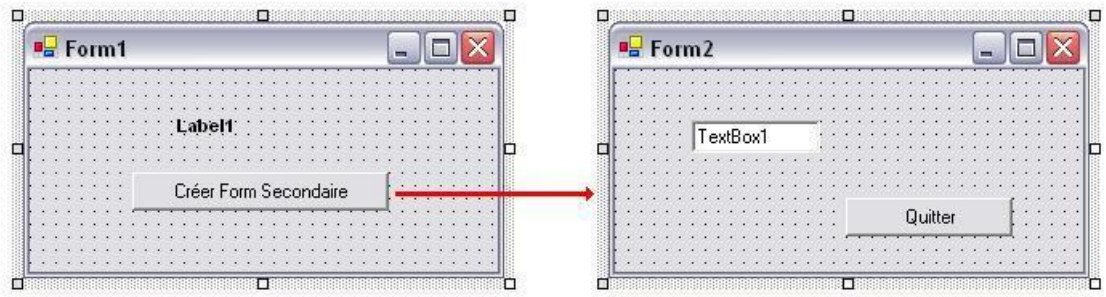
**B- Il faut créer ensuite le nouvel Objet formulaire, une instance de Form2.**

**Pour créer un nouveau formulaire dans le programme, il faut :**

- instancier un formulaire à partir du moule, de la Classe Form2 avec le mot New ;
- ouvrir ce formulaire, le faire apparaitre (avec ShowDialog, c'est un formulaire modal) :

```
Dim formSecondaire As New Form2()

formSecondaire.ShowDialog()
```

En résumé : on a Form1, on dessine Form2 :




Pour que le bouton nommé "Créer Form secondaire" ouvre le second formulaire, il faut y mettre le code :

```
Private ButtonCreerFormSecondaire_Click()

  Dim formSecondaire As New Form2()

  formSecondaire.ShowDialog()

End Sub
```

*En conclusion*

*Le fait d'ajouter un formulaire à un projet crée une Class, (un 'type' de formulaire, un moule) ce qui permet ensuite d'instancier (de créer) un objet formulaire.*

VB 2003 est tolérant pour le premier formulaire : si on dessine un formulaire et ses contrôles et qu'on lance le programme, il accepte de fonctionner bien qu'on n'ait pas instancié le formulaire. Par contre, si on crée une seconde classe formulaire, il faut créer une instance de ce formulaire.

```
Dim formSecondaire As New Form2()
formSecondaire.ShowDialog()
```

## X-B-2 - Créer un formulaire en VB 2005

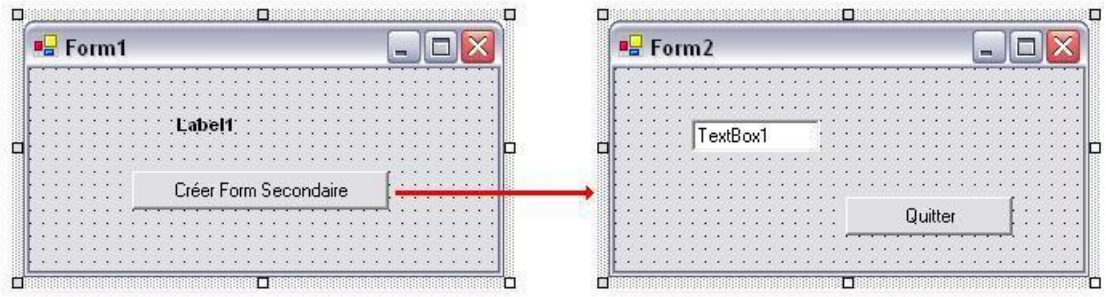
Pas besoin d'instancier systématiquement un formulaire.

On peut utiliser la Class Form2 sans instancier, en utilisant directement le nom de la Classe.

On dessine Form2 (la classe) puis on peut écrire directement :

```
Private ButtonCreerFormSecondaire_Click()
```

```
    Form2.ShowDialog()
End sub
```




On peut même utiliser les propriétés directement :

```
Form2.ForeColor = System.Drawing.Color.Coral
 Form2.BackColor = System.Drawing.Color.Cyan
```

En fait, comme il n'y a pas d'instance de Form2, VB en crée une.

On peut aussi faire comme en VB 2003 en instancier le formulaire, mais c'est plus complexe.

## X-B-3 - Formulaire modal ou non modal

**Un formulaire modal** est un formulaire qui, une fois ouvert, prend la main, interdit l'usage des autres fenêtres. Pour poursuivre, on ne peut que sortir de ce formulaire.

Exemple typique : une MessageBox est un formulaire modal, les fenêtres d'avertissement dans Windows sont aussi modales.

Pour ouvrir un formulaire modal, il faut utiliser la méthode .ShowDialog :

```
Dim f As New Form2

f.ShowDialog()
```

ou en VB 2005 :

```
form2.ShowDialog()
```

Noter, et c'est très important, que **le code qui suit .showDialog est exécuté après la fermeture de la fenêtre modale**.

**Pour avoir un formulaire non modal faire :**

```
Dim f As New Form2

f.Show()
```

ou en VB 2005 :

```
form2.Show()
```

Dans ce cas le formulaire f s'ouvre, le code qui suit .Show est exécuté immédiatement, et il est possible de passer dans une autre fenêtre de l'application sans fermer f.

**Instance multiple :** si un bouton1 contient le code :

```
Private Button1_Click

Dim f As New Form2

f.Show()

End Sub
```

À chaque fois que l'on clique sur le bouton cela ouvre un formulaire : on peut en ouvrir plusieurs. On se retrouve avec X instances de Form2 !!

**Pour éviter cela :**

- utiliser ShowDialog ;
- Mettre Dim f As New Form2 dans la partie déclaration, ainsi il n'y aura qu'une instance de Form2. Le second clic déclenche une erreur.

```
Class Form1

Dim f As New Form2


Private Button1_Click

    f.Show()

End Sub

End Class
```

## X-B-4 - Dénomination des formulaires après leur création

En VB 2003 et 2005 (avec instanciation d'un formulaire).

Une procédure qui est dans Form1 crée un formulaire par :

```
Class Form1
Private Buttonformsecondaire_Click ()

    Dim formSecondaire As New Form2

End Sub
End Class
```

- Dans le formulaire formSecondaire créé :
  utiliser **Me** pour désigner le formulaire où on se trouve. (Form2 ou formSecondaire ne sont pas acceptés).
  Exemple
  Me.Text= "Second formulaire" modifie le texte de la barre supérieure du formulaire
  Le formulaire formSecondaire pourra être fermé par Me.close() dans le code du bouton Quitter par exemple.
- Hors du formulaire formSecondaire, dans la procédure où a été instancié le formulaire :
  utiliser formSecondaire pour désigner le formulaire.
  Exemple
  Si la fenêtre appelante veut récupérer des informations dans le formulaire formSecondaire (un texte dans txtMessage par exemple), il faudra écrire :
  Text=formSecondaire.txtMessage.Text
- Par contre, hors de la procédure qui a créé le formulaire, formSecondaire n'est pas accessible, car on a créé le formulaire dans une procédure : cette instance du formulaire n'est visible que dans cette procédure. Pour

rendre un formulaire accessible partout on peut écrire Public formSecondaire As New Form2 dans la zone générale avant les procédures.

```
Class Form1
Public formSecondaire As New Form2
Private Buttonformsecondaire_Click ()



End Sub
End Class
```

**Exemple**

```
Class Form1


Sub MaRoutine()

    Dim formSecondaire As New Form2

    Text=formSecondaire.TextBox.Text

End Sub


Sub AutreRoutine()

…

End Sub


End Class
```

Dans la procédure MaRoutine() le formulaire formSecondaire est visible et formSecondaire.TextBox est utilisable, pas dans la procédure AutreRoutine().

En résumé : attention donc, si vous instanciez un formulaire dans une procédure, elle sera visible et accessible uniquement dans cette procédure.

Cela parait évident, car un formulaire est un objet comme un autre et sa visibilité obéit aux règles habituelles (j'ai mis malgré tout un certain temps à le comprendre !!).

*Un formulaire est un objet et sa visibilité obéit aux règles habituelles : il peut être instancié dans une procédure, un module, précédé de 'Public' ,'Private'… ce qui permet de gérer son accessibilité.*

**En VB 2005, sans instanciation des formulaires**

Par contre en VB 2005, si vous dessinez Form2 et que vous tapez :

```
Private ButtonCreerFormSecondaire_Click()
    Form2.Show()
End sub
```

Vous pouvez utiliser dans Form1 les propriétés et contrôles de Form2 directement :

```
Form2.ForeColor = System.Drawing.Color.Coral
```

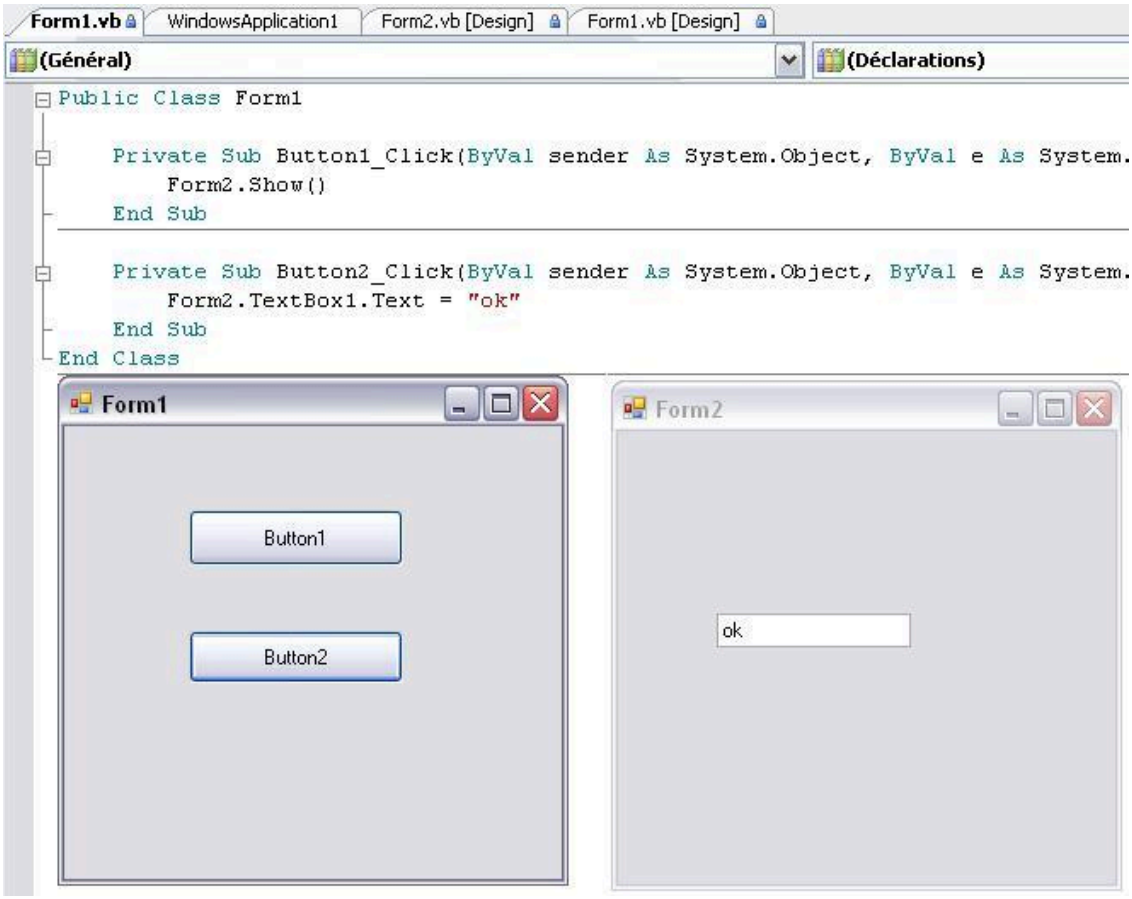
La Classe Form2 étant 'public', on a toujours accès au formulaire et aux contrôles.

(Par contre, on n'a pas accès aux procédures événements qui sont 'Private'.)

**Un exemple**

Dans Form1 Button1 affiche le formulaire Form2 (directement sans instanciation).

Dans la procédure Button2_Click de Form1 on a accès au TextBox qui est dans Form2 :




## X-B-5 - Autres remarques sur les formulaires

### X-B-5-a - Un formulaire est un objet : on peut ajouter des méthodes et des membres à un formulaire

On a vu que, en fait, il y a création d'une Classe quand on dessine un formulaire, et bien comme dans un module de Classe (on verra cela plus loin), on peut ajouter des propriétés et des méthodes.

Pour ajouter une méthode à un formulaire, il faut créer une Sub Public dans le corps de la fenêtre :

```
Class Form1

Public Sub Imprime()

    Code d'impression
```

```
End Sub

End Class
```

Si une instance de la fenêtre se nomme F, F.Imprime() exécute la méthode Imprime (donc la sub Imprime()).

De même, pour définir un membre d'un formulaire, il faut ajouter une variable 'public'.

```
Public Utilisateur As String
```

Permet d'utiliser en dehors du formulaire F.Utilisateur.

## X-B-5-b - Exemple plus complet : Afficher un formulaire

Comment savoir si un formulaire existe, s'il n'existe pas, le créer, s'il existe le rendre visible et lui donner la main :

```
        If f Is Nothing Then    'Si f=rien

            f = New Form2

            f.ShowDialog()

        Else

            If f.Visible = False Then

                f.Visible = True

            End If

            f.Activate()

        End If
```

Autre solution plus complète gérant aussi la taille du formulaire :

Si le formulaire existe et n'a pas été 'disposed' (détruit), le mettre à la taille normale et en avant.

```
        If Not IsNothing(F) Then
            'Si on n'en a pas déjà disposé
            If Not F.IsDisposed Then
                F.WindowState = FormWindowState.Normal  ' Optional
                F.BringToFront()  '  Optional
            Else
                F = New Form3
                F.Show()
            End If
        Else
            F = New Form3
            F.Show()
        End If
```

(Merci Michel de Montréal.)

## X-B-5-c - Récupération d'informations par DialogResult

On ouvre un formulaire modal, comment, après sa fermeture, récupérer des informations sur ce qui s'est passé dans ce formulaire modal ?

Par exemple, l'utilisateur a-t-il cliqué sur le bouton OK ou le bouton Cancel pour fermer le formulaire modal ?

Pour cela on va utiliser une propriété **DialogResult** des boutons, y mettre une valeur correspondant au bouton, quand l'utilisateur clique sur un bouton, la valeur de la propriété DialogResult du bouton est assignée à la propriété DialogResult du formulaire, on récupère cette valeur à la fermeture du formulaire modal.

Dans le formulaire modal Form2 on met :

```
ButtonOk.DialogResult= DialogResult.ok



 ButtonCancel.DialogResult= DialogResult.Cancel
```

Dans le formulaire qui appelle :

```
Form2.ShowDialog()

If form2.DialogResult= DialogResult.ok then

    'l'utilisateur a cliqué sur le bouton OK

End if
```

**Remarque**

1 On utilise comme valeur de DialogResult les constantes de l'énumération DialogResult : DialogResult.ok .Cancel .No .Yes .Retry .None .Abort .Ignore.
2 Si l'utilisateur clique sur la fermeture du formulaire modal (bouton avec X) cela retourne DialogResult.cancel.
3 On peut aussi utiliser la syntaxe : If form2.ShowDialog(Me) = System.Windows.Forms.DialogResult.OK Then qui permet en une seule ligne d'ouvrir form2 et de tester si l'utilisateur a cliqué sur le bouton OK de form2.
4 La fermeture du formulaire modal par le bouton de fermeture ou l'appel de la méthode Close ne détruit pas toujours le formulaire modal, il faut dans ce cas utiliser la méthode Dispose pour le détruire.

## X-B-5-d - Bouton par défaut

Parfois dans un formulaire, l'utilisateur doit pouvoir, valider (taper sur la touche 'Entrée') pour accepter et quitter rapidement le formulaire (c'est l'équivalent du bouton 'OK') ou taper 'Echap' pour sortir du formulaire sans accepter (c'est l'équivalent du bouton 'Cancel').

Il suffit pour cela de donner aux propriétés **AcceptButton** et **CancelButton** du formulaire, le nom des boutons OK et cancel qui sont sur la feuille.

```
form1.AcceptButton = buttonOk
form1.CancelButton = buttonCancel
```

Si l'utilisateur tape la touche 'Echap' la procédure buttonCancel_Click est exécutée.

## X-C - Faire communiquer les formulaires

*Rappel:Formulaire=fenêtre*

**Comment faire communiquer 2 formulaires ?+++**

A Comment à partir d'un formulaire consulter un objet d'un autre formulaire ?
B Comment à partir du second formulaire connaitre le formulaire propriétaire ?
C Les formulaires ouverts en VB 2005.

Cette question est fréquemment posée et crée beaucoup de problèmes !!

## X-C-1 - Comment, à partir du premier formulaire, consulter un objet du second formulaire ?

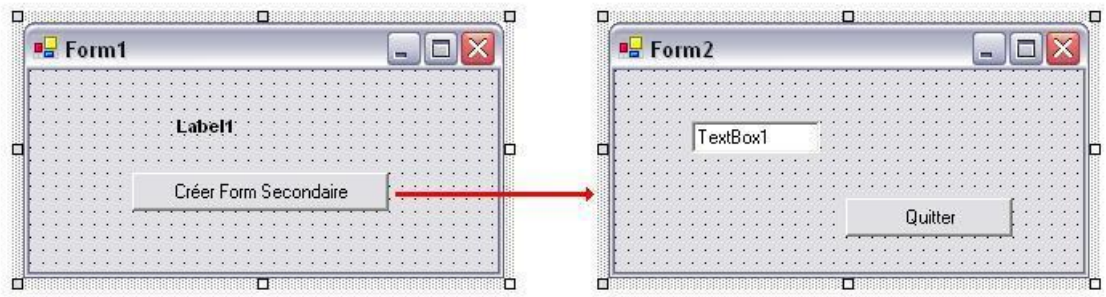
Un premier formulaire en ouvre un second, dans le second saisir un texte, puis l'afficher dans le premier.

## X-C-1-a - En VB 2003 2005 2008 si on instancie le formulaire

Reprenons toujours le même exemple : le premier formulaire (Class Form1) crée une instance de Form2 (l'utilisateur du programme clique sur ButtonCreerformsecondaire de Form1) cela crée formSecondaire. formSecondaire contient un textbox nommé TextBox2. L'utilisateur saisit un texte dans le textbox2 et quitte formsecondaire. Comment Form1 peut-il récupérer TextBox2.text et l'afficher dans un label1.

La Class Form1 contient un Button1 "Créer Form2" et contient un Label1.

La Class Form2 aura une instance : formsecondaire, elle contient TextBox2 et un Button2 "Quitter".




### X-C-1-a-i - Première solution

Pour qu'un formulaire puisse utiliser les objets d'un autre formulaire, il faut que le second formulaire soit visible.

Créer un formulaire formSecondaire en utilisant la Classe Form2.

```
Class Form1

Sub buttonCreerformsecondaire_Click

  Dim formSecondaire As New form2 ()     'On crée formSecondaire

  formSecondaire.ShowDialog()            'On ouvre formSecondaire

  label1.Text=formSecondaire.TextBox2.Text 'On récupère le texte de TextBox1

End Sub


End Class
```

formSecondaire n'est visible QUE dans button1_Click+++

Les contrôles de Form2 sont Public ce qui permet d'y avoir accès.

On peut se poser la question de savoir si après ShowDialog le formulaire modal formSecondaire existe encore.

La ruse c'est de mettre dans le code du bouton Quitter de Form2 Me.Hide() pour rendre la fenêtre Form2 invisible, mais accessible (et pas Me.Close() qui détruirait la fenêtre, le contrôle txtMessage et son contenu) :

```
Dim formSecondaireAs New Form2()

formSecondaire.ShowDialog()

label1.Text=formSecondaire.TextBox2.Text

formSecondaire.Close()
```

Une fois que le texte a été récupéré, on fait disparaitre le formulaire formSecondaire.

En réalité, curieusement, il semble que les propriétés de formSecondaire soient accessibles même après un Close !! Cela vient du fait que, bien que le formulaire soit fermé, il n'est pas encore détruit.

Si vous voulez créer un formulaire Form2 qui soit visible dans la totalité d'un formulaire Form1, il faut l'instancier dans la partie déclaration du formulaire Form1 :

```
Class Form1

    Public formSecondaire As New Form2()


Sub buttoncreerformsecondaire_Click

  formSecondaire.ShowDialog()           'On ouvre formSecondaire

  label1.Text=formSecondaire.TextBox2.Text 'On récupère le texte de TextBox1

End Sub


End Class
```

On peut ainsi l'ouvrir par formSecondaire.ShowDialog() dans une procédure et lire une zone texte dans une autre procédure.

## X-C-1-a-ii - Seconde solution

Si vous voulez créer un formulaire qui soit visible dans la totalité du programme et dont les contrôles ou propriétés soient accessibles par l'ensemble du programme, il faut l'instancier dans un module standard (les puristes ne vont pas aimer !!) :

```
Module MonModule

    Public formSecondaire As New Form2().

End Module



Class Form3

Sub Buttoncreerformsecondaire_Click
```

```
    formSecondaire.ShowDialog()
    'ou MonModule.formSecondaire.ShowDialog()
End Sub



Sub Button2_Click

    label1.Text= formSecondaire.TextBox2.Text

End Sub



End Class
```

On peut avoir accès au TextBox2 n'importe où !!

C'est un objet 'Public' et on n'aime pas bien !!!

## X-C-1-a-iii - Troisième solution

On peut créer dans le second formulaire un objet Public Shared :

```
Class Form2

Public Shared MonTextBox As TextBox

Private Sub Button2_Click 'Bouton quitter

MonTextBox = TextBox1    'On affecte à l'objet MonTexBox le TextBox1

Me.Close()

End Sub

End Class
```

Dans Form1

```
Class Form1

Sub buttoncreerformsecondaire_Click

Dim formSecondaire As New form2 ()    'On crée formSecondaire

formsecondaire.ShowDialog()

Label1.Text = formsecondaire.MonTextBox.Text()

End Sub

End Class
```

Noter que contrairement aux exemples donnés par certains sites, il faut bien écrire : formsecondaire.MonTextBox.Text() et pas Form2.MonTextBox.Text() du moins en VS 2003.

Cette troisième solution a le même principe que la première, en plus compliqué !!

On peut simplement retenir que si formsecondaire est visible, seuls ses membres "public" sont visibles bien entendu : par exemple ses propriétés, ses contrôles, les procédures "public", PAS les procédures événementielles qui sont privées.

Dans le même ordre d'idée, on peut créer une Property Public :

```
Class Form2

Public ReadOnly Property LeText() As String
    Get
        Return TextBox2.Text
    End Get
End Property
End Class

Class Form1

Sub button1_Click

Dim formSecondaire As New form2 ()     'On crée formSecondaire

formsecondaire.ShowDialog()

Label1.Text = formsecondaire.LeText    'On utilise la property

End Sub

End Class
```

Même conclusion, mais il faut toujours utiliser formsecondaire qui doit être visible, c'est ça l'important !!

## X-C-1-a-iv - Quatrième solution

Créer une variable ou une Classe 'Public' (dite 'Globale') et y faire transiter les données :

```
Module MonModule

    Public BAL As String      'Variable Public dite 'Boite aux lettres'

End Module

Class Form2

Private Sub Button2_Click 'Bouton quitter

BAL = TextBox2.Text     'On met TextBox1.Text dans BAL

Me.Close()

End Sub

End Class


Class Form1

Sub Button1_Click

    formSecondaire.ShowDialog()

    label1.Text= BAL     'On récupère ce qui est dans BAL

End Sub


End Class
```

Cela a tous les inconvénients : c'est une variable globale, source d'erreur, n'importe quel formulaire peut écrire dans BAL…
C'est très mal de faire cela.
Mais c'est simple et cela marche bien.

## X-C-1-b - En VB 2005, sans instanciation de formulaire

Par contre en VB 2005, si vous dessinez Form2 et que vous tapez :

```
Private ButtonCreerFormSecondaire_Click()
    Form2.Show()
End sub
```

Vous pouvez utiliser dans Form1 les propriétés et contrôles de Form2 directement :

```
Label1.Text = Form2.TextBox2.Text
```

La Classe Form2 étant "public" , on a toujours accès au formulaire et aux contrôles.

(Par contre, on n'a pas accès aux procédures événements qui sont 'Private'; on peut d'ailleurs les mettre 'Public' pour y avoir accès).

## X-C-2 - Comment, à partir du formulaire 'secondaire', connaitre le formulaire 'propriétaire' ?

Exemple : Comment savoir quel formulaire a ouvert le formulaire en cours ?

ShowDialog possède un argument facultatif, owner, qu'on peut utiliser afin de spécifier une relation 'propriétaire'-'formulaire en cours'. Par exemple, lorsque le code de votre formulaire principal ouvre un formulaire, vous pouvez passer Me comme propriétaire de la boite de dialogue, afin de désigner votre formulaire principal comme propriétaire, comme le montre le code de l'exemple suivant :

Dans Form1

```
Dim formSecondaire As New Form2

f.ShowDialog(Me)
```

Dans Form2
On peut récupérer le nom du 'propriétaire', qui a ouvert la fenêtre.
Il est dans Owner,et on peut par exemple afficher son nom.

Par exemple :

```
    Label1.text=Me.Owner.ToString
```

Cela affiche : NomApplication.NomFormulaire,Texte de la barre de titre.

Owner a toutes les propriétés (Name, Location, Controls…) d'un formulaire, car il hérite de Form, mais on ne peut pas consulter les contrôles de Owner directement. Il faut d'abord caster owner qui est une Form en Form1, ensuite on peut avoir accès aux contrôles de Form1.

```
Dim f As Form1 = CType(Me.Owner, Form1)

Label1.Text() = f.Button1.Text
```

**Comment obtenir le nom du formulaire propriétaire ? Autre méthode.**

Une autre méthode consiste à surcharger le constructeur de la Form2 afin qu'il accepte un paramètre qui indique le propriétaire (nom de l'instance de Form1).

Dans Form1 : Lors de l'instanciation de la form2 il faut écrire :

```
Dim FormSecondaire As New Form2(Me)
FormSecondaire .ShowDialog(Me) 'affichage modal de la form2
```

Dans Form2 :

```
Private FormProp As Form1
    Public Sub New(ByVal NomForm As Form1)
        MyBase.New()
        FormProp = NomForm

        InitializeComponent()

    End Sub
```

On crée donc dans Form2 une variable FormProp qui indique la form propriétaire.

Pour appeler une méthode de form1 à partir de FormSecondaire (Form2) :

```
FormProp.MaRoutine()
```

L'inconvénient de toutes ces méthodes est qu'il faut connaitre la classe du formulaire propriétaire (Form1 ici).

## X-C-3 - Les formulaires ouverts à partir de VB 2005

- **My.Application.OpenForms** contient les formulaires ouverts.

Afficher le texte contenu dans la barre de titre du formulaire nommé 'Form3'.

```
MyTextBox.Text= My.Application.OpenForms("Form3")
```

Afficher le texte contenu dans la barre de titre du premier formulaire ouvert.

```
MyTextBox.Text= My.Application.OpenForms(0)
```

Exemple: rajouter le texte 'ouvert' à la barre de tâches des formulaires ouverts :

```
For Each F As System.Windows.Forms.Form In My.Application.OpenForms
F.Text += "[ouvert]"
Next
```

- **My.Forms** contient tous les formulaires.

Afficher le texte contenu dans la barre de titre du formulaire Form1.

```
MyTextBox.Text= My.Forms.Form1.Text
```

Remarquons qu'il est interdit d'utiliser My.Forms.Form1 si on est dans Form1. (il faut utiliser Me)

-Différence avec My.Application.OpenForms(0) :

```
Dim f As New Form1

f.Text = "hello"

f.Show()

TextBox1.Text = My.Forms.Form1.Text
```

Affiche 'Form1' qui est le texte de la barre par défaut en design (celui de la Classe Form1).

```
TextBox2.Text = My.Application.OpenForms("Form1").Text

'Affiche 'hello' qui est le texte de l'instance f
```

## X-C-4 - Utilisation de DialogResult

Si un formulaire Form1 ouvre un formulaire modal nommé Dialog1, il y a un moyen élégant pour le formulaire Form1 de récupérer une information du formulaire Dialog1 quand ce dernier est fermé (il est modal).

Dans le formulaire modal Dialog1, si l'utilisateur clique sur le bouton OK, on met :

```
Sub ButtonOk_Click
 ButtonOk.DialogResult= DialogResult.ok
End Sub
```

Dans Form1 on a :

```
'On ouvre le formulaire Dialog1
Dialog1.ShowDialog()

If Dialog1.DialogResult= DialogResult.Ok then

    'l'utilisateur a cliqué sur le bouton OK

End if
```

Les seules valeurs possibles pour DialogResult sont OK, Cancel, Yes, No, None, Abort, Ignore, Retry. On ne peut malheureusement pas utiliser une autre valeur

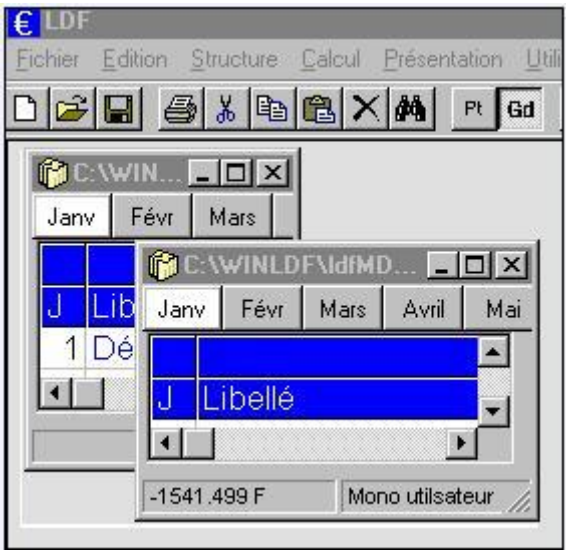
## X-D - Créer une fenêtre 'multi documents'

Comment créer un programme **MDI** (Multi Document Interface) en VB 2003 puis en VB 2005 ?

## X-D-1 - Comprendre les programmes MDI

L'exemple de Word : la fenêtre principale (fenêtre MDI) contient les menus en haut, on peut ouvrir plusieurs documents dans des fenêtres filles.

Ci-dessous l'exemple de LDF (Programme de comptabilité écrit par l'auteur) :




*On a une fenêtre MDI (conteneur) contenant 2 fenêtres filles affichant chacune une année de comptabilité.*

Dans VB.NET, un **MDIForm** (fenêtre principale MDI) est une fenêtre quelconque, dont la propriété :

**IsMDIContainer = True**.

Dans la fenêtre fille, **la propriété MDIParent indique le conteneur** (C'est-à-dire le nom de la fenêtre MDI).

Les applications MDI peuvent avoir plusieurs conteneurs MDI.

Une fenêtre principale MDI peut contenir plusieurs fenêtres filles de même type ou de type différent.

## X-D-2 - En VB 2003

### X-D-2-a - Création de la fenêtre conteneur parent

Exemple d'un programme MDI.

On va créer une Form1 qui est le conteneur.

Une Form2 qui est la fenêtre fille.

Dans Form1 le menu principal contient la ligne '&Nouvelle' qui crée une nouvelle instance de la fenêtre fille.

Créer la fenêtre Form1

Dans la fenêtre Propriétés, **affectez la valeur True à la propriété IsMDIContainer**. Ce faisant, vous désignez la fenêtre comme le conteneur MDI des fenêtres enfants.

Remarque: Affecter la valeur Maximized à la propriété WindowState, car il est plus facile de manipuler des fenêtres MDI enfants lorsque le formulaire parent est grand. Sachez par ailleurs que le formulaire MDI parent prend la couleur système (définie dans le Panneau de configuration Windows).

Ajouter les menus du conteneur

À partir de la boite à outils, faire glisser un contrôle MainMenu sur le formulaire. Créer un élément de menu de niveau supérieur en définissant la propriété Text avec la valeur &File et des éléments de sous-menu appelés &Nouvelle et &Close. Créer également un élément de menu de niveau supérieur appelé &Fenêtre.

Dans la liste déroulante située en haut de la fenêtre Propriétés, sélectionnez l'élément de menu correspondant à l'élément &Fenêtre et affectez la valeur true à la propriété MdiList. Vous activez ainsi le menu Fenêtre qui permet de tenir à jour une liste des fenêtres MDI enfants ouvertes et indique à l'utilisateur par une coche la fenêtre enfant active.

Il est conseillé de créer un module standard qui contient une procédure Main qui affiche la fenêtre principale :

```
Module StandartGénéral

Public FrmMDI as Form1

Sub Main()

    FrmMDI.ShowDialog()

End sub

End Module
```

Noter bien que FrmMDI est donc la fenêtre conteneur et est Public donc accessible à tous.

## X-D-2-b - Création des fenêtres filles

Pour créer une fenêtre fille, il suffit de **donner à la propriété MDIParent d'une fenêtre le nom de la fenêtre conteneur**.

Dessiner dans Form2 les objets nécessaires dans la fenêtre fille.

Comment créer une instance de la fenêtre fille à chaque fois que l'utilisateur clique sur le menu '&Nouvelle' ?

En premier lieu, déclarez dans le haut du formulaire Form1 (accessible dans tout le formulaire) une variable nommée MDIFilleActive de type 'Form2' qui contient la fenêtre fille active.

```
Dim MDIFilleActive As Form2
```

La routine correspondant au MenuItem &Nouvelle (dans la fenêtre MDI) doit créer une instance de la fenêtre fille :

```
Protected Sub MDIChildNouvelle_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
_Handles MenuItem2.Click

   MDIFilleActive = New Form2()

   'Indique à la fenêtre fille son 'parent'.

   MDIFilleActive.MdiParent = Me

   'Affiche la fenêtre fille

   MDIFilleActive.Show()

End Sub
```

Ainsi l'utilisateur peut ouvrir plusieurs fenêtres filles.

## X-D-2-c - Comment connaitre la fenêtre fille active ?

Quand on en a ouvert plusieurs ?

La fenêtre fille active est dans Me.ActiveMdiChild du conteneur

Comment voir s'il existe une fenêtre active ?

```
If Not Me.ActiveMdiChild Is Nothing then     'elle existe
```

En mettant dans la variable MDIFilleActive la fenêtre active, on est sûr de l'avoir toujours à disposition : pour cela dans la procédure Form1_MdiActivate de la fenêtre MDI (qui se produit à chaque fois que l'on rentre dans la fenêtre fille avec la souris, le menu…) je récupère **Me.ActiveMdiChild** qui retourne la fenêtre fille active.

Dans Form1 :

```
Private Sub Form1_MdiChildActivate…

    MDIFilleActive=Me.ActiveMdiChild

End Sub
```



*Il faut comprendre que peut importe le nom de la fenêtre fille active, on sait simplement que la fenêtre fille active est dans MIDFilleActive, variable que l'on utilise pour travailler sur cette fenêtre fille.*

## X-D-2-d - Comment avoir accès aux objets de la fenêtre fille à partir du conteneur ?

De la fenêtre conteneur j'ai accès aux objets de la fenêtre fille par l'intermédiaire de la variable MDIFilleActive précédemment mise à jour; par exemple le texte d'un label :

```
MDIFilleActive.label1.text
```

Comment avoir accès à des éléments de cette fenêtre fille, une sub Affichetotaux par exemple à partir de ActiveMdiChild :

si je tape Me.ActiveMdiChild.AfficheTotaux cela plante !! (car ActiveMdiChild est une instance de la classe Form).

Il faut écrire : CType(ActiveMdiChild, Form2).Affichetotaux() (car il faut convertir ActiveMdiChild en classe Form2)

## X-D-2-e - Comment parcourir toutes les fenêtres filles ?

La collection MdiChildren contient toutes les fenêtres filles, on peut les parcourir :

```
Dim ff As Form2

For Each ff In Me.MdiChildren

….

Next
```

Cela est valable s'il n'y a qu'un type de formulaire permettant de créer des formulaires enfants (Form2 par exemple).

Mais si on a 2 formulaires Form2 et Form3 cela se complique.

```
dim i_form as Form ' on utilise une variable Form :formulaire

dim i_form3 as form3
dim i_form2 as form2

For Each i_form  In Me.mdichildren
    if typeof  i_form is form2 then
        i_form2 = ctype(i_form,form2)
        msgbox i_form2.property_du_form2
    end if

if typeof  i_form is form3 then
        i_form1 = ctype(i_form,form3)
        msgbox i_form3.property_du_form3
    end if
next
```

Merci Gaël.

## X-D-2-f - Comment fermer toutes les fenêtres enfants ?

```
Dim form As Form

For Each form In Me.MdiChildren

form.Close()

Next
```

## X-D-2-g - Comment avoir accès aux objets du conteneur à partir de la fenêtre fille ?

En utilisant **Me.MdiParent** qui contient le nom du conteneur.

Dans la fenêtre fille le code Me.MdiParent.text ="Document 1" affichera 'Document 1' dans la barre de titre du conteneur.

## X-D-2-h - Comment une routine du module conteneur appelle une routine dans la fenêtre fille active ?

Si une routine public de la fenêtre fille se nomme Affiche, on peut l'appeler par :

```
MDIFilleActive.Affiche()
```

Il n'est pas possible d'appeler les événements liés aux objets de la fenêtre fille, par contre la routine Affiche() dans notre exemple peut le faire.

## X-D-2-i - Agencement des fenêtres filles

La propriété **LayoutMdi** de la fenêtre conteneur modifie l'agencement des fenêtres filles.

0 - MdiLayout.Cascade
1 - MdiLayout.TileHorizontal
2 - MdiLayout.TileVertical
3 - MdiLayout.ArrangeIcons

Exemple

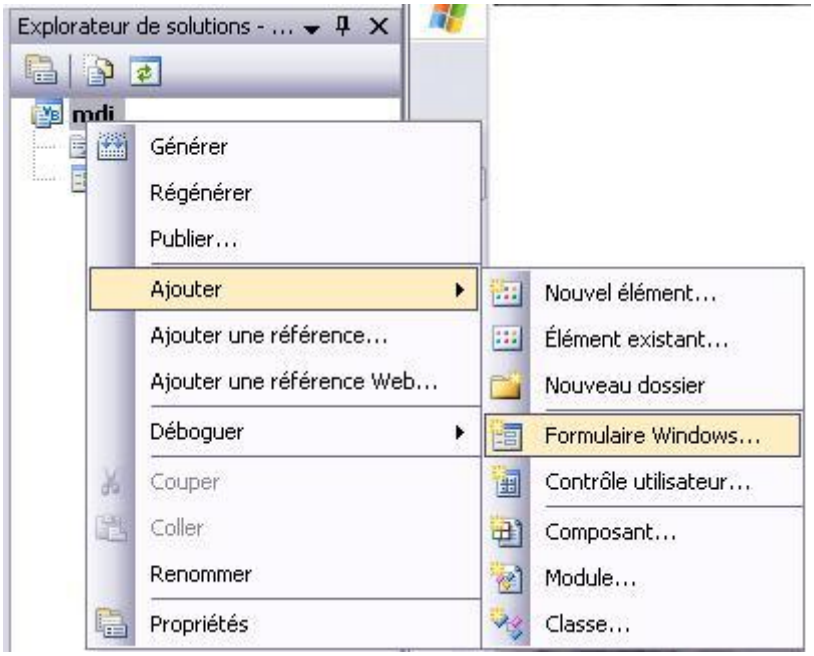
Le menu Item Cascade met les fenêtres filles en cascade.

```
Protected Sub CascadeWindows_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)

   Me.LayoutMdi(System.Windows.Forms.MdiLayout.Cascade)

End Sub
```

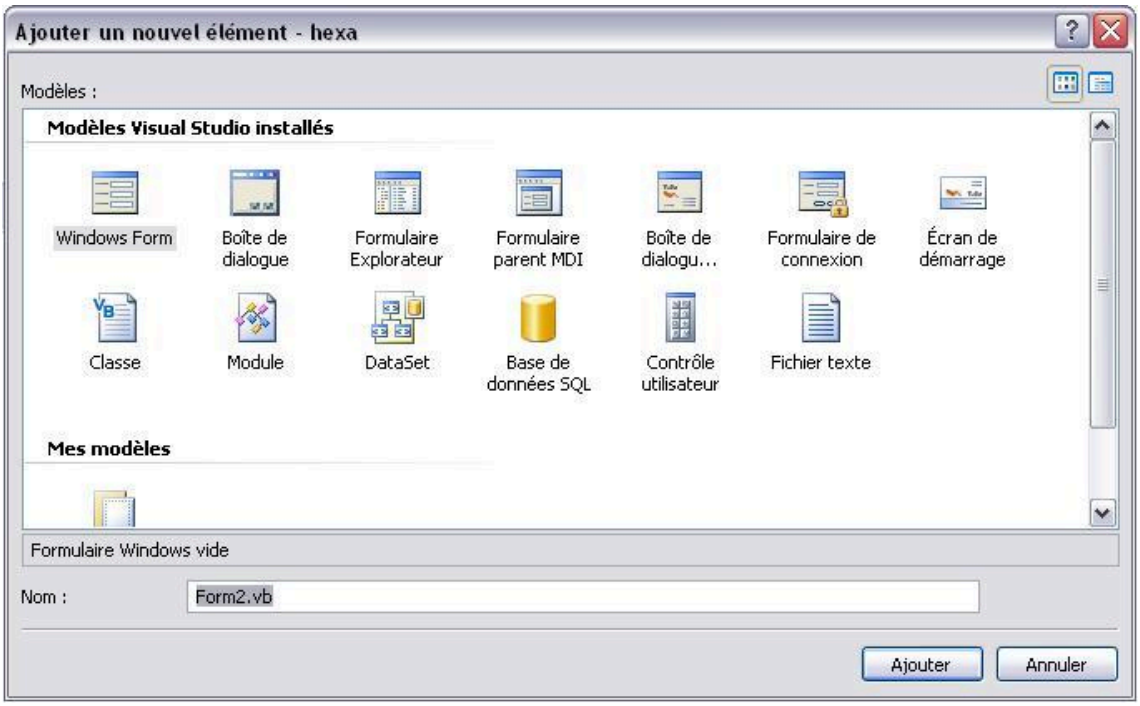
## X-D-3 - En VB 2005 2008

90 % du travail est fait automatiquement : c'est merveilleux !!

Dans l'explorateur de solution : clic droit sur le nom du programme ('mdi' ici) :




Dans le menu, passer par 'Ajouter' puis 'Formulaire Windows' :




Cliquer sur 'MDI Parent Form' ('Formulaire Parent MDI').

On obtient un formulaire MDI parent avec les menus, la barre d'icône, d'état, les menus déroutants avec image.

(Le logiciel a rajouté les 4 outils en bas, nécessaire pour réaliser l'interface.)



Le code qui génère les formulaires enfants est automatiquement créé :

```
Public Class MDIParent1

Private Sub ShowNewForm(ByVal sender As Object, ByVal e As EventArgs) Handles
 NewToolStripMenuItem.Clic,
    _NewToolStripButton.Clic, NewWindowToolStripMenuItem.Clic

' Créer une nouvelle instance du formulaire enfant.
```

```
Dim ChildForm As New System.Windows.Forms.Form

' Make it a child of this MDI form before showing it.

ChildForm.MdiParent = Me

m_ChildFormNumber += 1

ChildForm.Text = "Window " & m_ChildFormNumber

ChildForm.Show()

End Sub

Private Sub ExitToolsStripMenuItem_Click(ByVal sender As Object, ByVal e As EventArgs)
_Handles ExitToolStripMenuItem.Clic

'Quitter l'application

Global.System.Windows.Forms.Application.Exit()

End Sub

Private Sub CascadeToolStripMenuItem_Click(ByVal sender As Object, ByVal e As EventArgs)
_Handles CascadeToolStripMenuItem.Clic

'Positionnement des formulaires enfant

Me.LayoutMdi(MdiLayout.Cascade)

End Sub

Private Sub TileVerticleToolStripMenuItem_Click(ByVal sender As Object, ByVal e As EventArgs)
_Handles TileVerticalToolStripMenuItem.Clic

Me.LayoutMdi(MdiLayout.TileVertical)

End Sub

Private Sub TileHorizontalToolStripMenuItem_Click(ByVal sender As Object, ByVal e As EventArgs)
_Handles TileHorizontalToolStripMenuItem.Clic

Me.LayoutMdi(MdiLayout.TileHorizontal)

End Sub

Private Sub ArrangeIconsToolStripMenuItem_Click(ByVal sender As Object, ByVal e As EventArgs)
_Handles ArrangeIconsToolStripMenuItem.Clic

Me.LayoutMdi(MdiLayout.ArrangeIcons)

End Sub

Private Sub CloseAllToolStripMenuItem_Click(ByVal sender As Object, ByVal e As EventArgs)
_Handles CloseAllToolStripMenuItem.Clic

' Ferme tous les formulaires enfants

For Each ChildForm As Form In Me.MdiChildren

ChildForm.Close()

Next

End Sub

Private m_ChildFormNumber As Integer = 0

End Class
```

# X-E - Modifier le curseur, gérer la souris

Comment modifier l'apparence du curseur ?

Un curseur est une petite image dont l'emplacement à l'écran est contrôlé par la souris, un stylet ou un trackball. Quand l'utilisateur déplace la souris, le système d'exploitation déplace le curseur.

Différentes formes de curseur sont utilisées pour informer l'utilisateur de l'action que va avoir la souris.

## X-E-1 - Apparence du curseur

Pour modifier l'aspect du curseur, il faut modifier l'objet Cursor.Current ; l'énumération Cursors contient les différents curseurs disponibles :

```
System.Windows.Forms.Cursor.Current = System.Windows.Forms.Cursors.WaitCursor
```

ou plus simplement pour afficher le sablier :

```
Cursor.Current = Cursors.WaitCursor
```

Pour revenir au curseur normal :

```
Cursor.Current = Cursors.Default
```

Comme d'habitude il suffit de taper **Cursors.** pour voir la liste des curseurs.

Voici les principaux curseurs :


Le curseur peut disparaitre et être de nouveau affiché par **Hide** et **Show**.

Les fichiers curseur sont des .Cur ou des .Ico ; pour 'charger' le curseur personnalisé :

```
Button1.Cursor = New Cursor("MyCurseur.cur")
```

## X-E-2 - Curseur sur un contrôle

Un contrôle dans une fenêtre possède une propriété Cursor; en mode design, si je donne une valeur autre que celle par défaut, CursorWait par exemple, cela modifie le curseur quand la souris passe au-dessus de l'objet (cela met un sablier dans notre exemple).

```
Button1.Cursor = Cursors.WaitCursor
```

## X-E-3 - La souris

Pour tester si une souris est branchée : (ce qui est habituel).

```
SystemInformation.MousePresent  ' retourne True
```

**MouseButtons** retourne le nombre de boutons, **MouseWheelPresent** indique s'il y a une molette.

Chaque contrôle ou formulaire possède des procédures événements déclenchées par la souris :

**MouseEnter :** s'exécute lors de l'arrivée de la souris sur un contrôle ;

**MouseLeave :** s'exécute lors de la sortie de la souris d'un contrôle.

Plus important :

**MouseUp** se produit lorsque l'utilisateur lâche sur le bouton de la souris ;

**MouseDown** se produit lorsque l'utilisateur appuie sur le bouton de la souris.

Le déplacement de la souris exécute :

**MouseMove** se produit lorsque l'utilisateur déplace le pointeur de la souris

Un paramètre e de type **MouseEventsArgs** est alors disponible pour les 3 événements.

Exemple

La procédure événement se produit quand l'utilisateur appuie sur le bouton de la souris.

```
Private Sub Form1_MouseDown(ByVal sender As System.Object, ByVal e As
 System.Windows.Forms.MouseEventArgs) _
Handles MyBase.MouseDown

…

End Sub
```

Le paramètre e de type MouseEventsArgs permet de connaitre la position de la souris et l'état des boutons :

e.X et e.Y contiennent les coordonnées de la souris dans le contrôle en cours.



*Attention ce sont bien les coordonnées du contrôle alors que dans les opérations de drag and drop ce sont par contre les coordonnées 'écran' qui sont utilisées, il faudra les transformer en coordonnées 'control' avec PointToClient : voir le chapitre sur les coordonnées.*

e.Button() retourne le bouton qui a été enfoncé.

```
If e.Button()=MouseButtons.Left Then…
```

e.Click() retourne le nombre de clics.

e.Delta() retourne le nombre de crans de la molette souris.

Si la souris n'est pas sur le contrôle et que l'on veut quand même récupérer les événements de la souris sur ce contrôle, il faut mettre la propriété Capture de ce contrôle à True.

Bien sûr les événements les plus utilisés sont Click et DoubleClick.
Certains contrôles n'ont pas tous les événements : les boutons n'ont pas de DoubleClick.

## X-E-4 - Exemples

**Exemple 1**

Dans un programme de dessin, quand l'utilisateur enfonce le bouton gauche de la souris, on peut récupérer les coordonnées du point où se trouve la souris.

```
Private Sub PictureBox1_MouseDown(ByVal sender As Object, ByVal e As
 System.Windows.Forms.MouseEventArgs) Handles PictureBox1.MouseDown

Dim NewPoint As New Point(e.X, e.Y)

End Sub
```

Pour voir un exemple complet de dessin, regarder Programme simple de Dessin.

**Exemple 2 :** afficher dans une 'bulle' le numéro d'index de l'élément survolé dans une listbox.

Dans MouseMove les coordonnées de la souris permettent de déterminer l'index survolé grâce à IndexFromPoint. L'index est affiché dans un ToolTip.

```
Private Sub ListBox1_MouseMove(ByVal sender As Object, ByVal e As
 System.Windows.Forms.MouseEventArgs)
_Handles ListBox1.MouseMove

ToolTip1.SetToolTip(ListBox1, "Index:"& CType(ListBox1.IndexFromPoint(e.X, e.Y), String))

End Sub
```

la propriété IndexFromPoint n'existe pas pour les combos.

Voir aussi la page sur le Drag and Drop.

## X-F - Lancer une autre application, afficher une page Web

Comment lancer une autre application, un autre programme ?

## X-F-1 - L'ancienne méthode VisualBasic toujours valable : Shell

Shell lance un programme exécutable.

```
Id=Shell (NomdeProgramme)     'lance l'application NomdeProgramme
```

on peut utiliser aussi :

```
Id=Shell( NomdeProgramme, TypedeFenetre, Wait, TimeOut)
```

TypedeFenêtre utilise l'énumération AppWinStyle pour définir le type de fenêtre de l'application lancée: AppWinStyle.MaximizedFocus ouvre par exemple l'application en plein écran.

Si vous souhaitez attendre la fin du programme avant de continuer, vous devez définir Wait à True.

TimeOut est le nombre de millisecondes à attendre pour la fin du programme si Wait est True.

Exemple :

```
ID = Shell("""C:\Program Files\MonFichier.exe"" -a -q", , True, 100000)
```

Dans une chaine une paire de guillemets doubles adjacents ("") est interprétée comme un caractère de guillemet double dans la chaine. Ainsi, l'exemple précédent présente la chaine suivante à la fonction Shell :
"C:\Program Files\MonFichier.exe" -a -q

La fonction AppActivate rend active l'application ou la fenêtre définie par son nom ou son Id (Numéro indentificateur).

```
Dim ID As Integer
```

On peut utiliser :

```
AppActivate("Untitled - Notepad")
```

ou

```
ID = Shell(NOTEPAD.EXE", AppWinStyle.MinimizedNoFocus)
AppActivate(ID)
```

## X-F-2 - On peut utiliser la Classe 'Process' du Framework

La Classe Process fournit l'accès à des processus locaux ainsi que distants, et vous permet de démarrer et d'arrêter des processus système locaux.

Classe de nom à importer : Imports System.Diagnostics

- On peut instancier un Process

```
Dim monProcess As New Process()
```

Ensuite il faut fournir à la classe fille StartInfo les informations nécessaires au démarrage.

```
monProcess.StartInfo.FileName = "MyFile.doc"
monProcess.StartInfo.Verb = "Print"
monProcess.StartInfo.CreateNoWindow = True
```

Enfin on lance le process/

```
monProcess.Start()
```

Noter la puissance de cette classe: on donne le nom du document et VB lance l'exécutable correspondant, charge le document, effectue certaines actions.

Dans l'exemple du dessus on ouvre Word on y charge MyFile , on l'imprime, cela sans ouvrir de fenêtre.

- On peut utiliser la classe Process en statique (sans instanciation)

```
Process.Start("IExplore.exe")

Process.Start(MonPathFavori)
```

ou en une ligne :

```
Process.Start("IExplore.exe", "http//:developpez.com")
```

En local on peut afficher un fichier html ou asp

```
Process.Start("IExplore.exe", "C:\monPath\Fichier.htm")
Process.Start("IExplore.exe", "C:\monPath\Fichier.asp")
```

- On peut enfin utiliser un objet StartInfo

```
Dim startInfo As New ProcessStartInfo("IExplore.exe")
startInfo.WindowStyle = ProcessWindowStyle.Minimized

Process.Start(startInfo)

startInfo.Arguments = "http//:developpez.com"

Process.Start(startInfo)
```

Des propriétés du processus en cours permettent de connaitre l'Id du processus (Id) les threads, les modules, les Dll ,la mémoire, de connaitre le texte de la barre de titre (MainWindowsTitle)…

On peut fermer le processus par Close ou CloseMainWindows.

On peut instancier un 'Process' sur une application déjà en cours d'exécution avec GetProcessByName et GetProcessById :

```
Dim P As Process() = Process.GetProcessesByName("notepad")


 ' On peut récupérer le processus courant.
Dim ProcessusCourant As Process = Process.GetCurrentProcess()


' Récupérer toutes les instances de  Notepad qui tournent en local.
Dim localByName As Process() = Process.GetProcessesByName("notepad")


' Récupérer tous les processus en cours d'exécution grâce à GetProcesses:
Dim localAll As Process() = Process.GetProcesses()
```

Processus sur ordinateur distant.

Vous pouvez afficher des données statistiques et des informations sur les processus en cours d'exécution sur des ordinateurs distants, mais vous ne pouvez pas appeler Kill, Start, CloseMainWindows sur ceux-ci.

## X-G - Dessiner

Avec GDI+ utilisé par VB.NET, on utilise des **Graphics**.

Un **Graphics** est une simple référence à

- une Image ( un BitMap ou un MetaFile) ;

- la surface de dessin d'un contrôle.

Le Graphics n'est pas le dessin lui-même, il "pointe" simplement sur le dessin.

**Il faut utiliser le Graphics pour dessiner** (des points, lignes, rectangles, ellipses, couleur…) ou y écrire.

-Rectangle permet de définir une zone.

-Pen correspond à un Stylet.

-Font correspond à une police de caractères.

-Brush est une brosse.

### X-G-1 - Sur quoi dessiner ?

Sur un objet **Graphics**. Il faut donc définir une référence, un objet **Graphics**. On peut y associer des objets Image (des BitMap ou des MetaFile).

Pour obtenir un objet Graphics, il y a plusieurs façons :

- **Soit on instance un objet Graphics :**

```
Dim g as Graphics     'Graphics contient Nothing, je ne peux rien en faire.
```

Il faut donc y associer un objet **Image** (un BitMap ou un MetaFile) pour pouvoir travailler dessus.

Pour obtenir un BitMap par exemple, on peut :

soit créer un objet BitMap vide :

```
Dim newBitmap As Bitmap = New Bitmap(600, 400)

Dim g as Graphics = Graphics.FromImage(newBitmap)
```

Paramètres= taille du BitMap mais il y a plein de surcharges ;

soit créer un BitMap à partir d'un fichier sur disque.

C'est pratique si on veut modifier une image qui est dans un fichier : on la lit dans un BitMap puis on passe la référence dans l'objet Graphics.

```
Dim myBitmap as New Bitmap("maPhoto.bmp")     'Charge maPhoto dans le BitMap
Dim g as Graphics = Graphics.FromImage(myBitmap)     'Crée un Graphics et y associe le BitMap
```

g est un Graphics "pointant" sur l'image lue dans 'maPhoto.bmp' et je peux la modifier dans g.



*Attention : le Graphics n'est pas 'visible', pour le voir il faut le mettre dans un composant (un PictureBox par exemple) qui lui sera visible. On verra cela plus bas.*

- **Soit on appelle la méthode CreateGraphics d'un contrôle ou d'un formulaire.**

On appelle la méthode **CreateGraphics** d'un contrôle ou d'un formulaire afin d'obtenir une référence à un objet Graphics représentant la surface de dessin de ce contrôle ou de ce formulaire.

Cette méthode est utilisée si vous voulez dessiner sur un formulaire ou un contrôle existant ;

```
Dim g as Graphics
g = Me.CreateGraphics    'Pour un formulaire

Dim g as Graphics
g = Panel1.CreateGraphics    'Pour un contrôle Panel
```

On peut ensuite dessiner sur g, cela sera immédiatement visible sur le contrôle ou le formulaire (car g n'est pas un nouvel objet, il contient seulement la référence, le handles, de l'image du contrôle; quand on dessine sur g, en fait, on dessine sur l'image du contrôle).

Il faut quand on n'utilise plus l'objet graphics, utiliser la méthode Dispose pour le libérer.

- **Soit on récupère la référence de l'objet Graphics argument de l'événement Paint d'un contrôle.**

L'événement Paint des contrôles se déclenche lorsque le contrôle est redessiné, un objet Graphics est fourni comme une valeur de PaintEventArgs.

**Pour obtenir une référence à un objet Graphics à partir des PaintEventArgs de l'événement Paint et dessiner :**

- déclarez l'objet Graphics ;
- assignez la variable pour qu'elle référence l'objet Graphics passé dans les PaintEventArgs ;
- dessinez dans l'objet Graphics.

```
Private Sub Form1_Paint(sender As Object, e As PaintEventArgs) Handles _
   MyBase.Paint    'appelle la méthode Paint 'normale'

   Dim g As Graphics = e.Graphics
   ' Dessiner avec g , cela affichera dans la Form1
End Sub
```

Là aussi c'est la référence à l'objet qui a déclenché Paint (Form1 ici), si je dessine sur e, en fait je dessine sur Form1.

Noter bien que e est visible uniquement dans Form1_Paint

Pour forcer le déclenchement de l'événement Paint et dessiner, on utilise la méthode **OnPaint**.

# X-G-2 - Comment dessiner ?

**La classe Graphics fournit des méthodes permettant de dessiner :**

```
DrawImage     'Ajoute une image (BitMap ou MetaFile) à un graphics déjà créé.

DrawLine      'Trace une ligne

DrawString    'Écrit un texte

DrawPolygon   'Dessine un polygone

….
```

En GDI+ on envoie des paramètres à la méthode pour dessiner.

Exemple :

```
MonGraphique.DrawEllipse( New Pen(Couleur),r)     'cela dessine une ellipse
```

Les 2 paramètres sont : la couleur et le rectangle dans lequel on dessine l'ellipse.

**Pour travailler, on utilise les objets.**

| | |
|---|---|
| **Brush** (Brosse) | Utilisé pour remplir des surfaces fermées avec des motifs, des couleurs ou des bitmaps.<br>Il y a plusieurs types de brush:<br>La brosse peut être pleine et ne contenir qu'une couleur: Dim SB= New SolidBrush(Color.Red)<br>TextureBrush utilise une image pour remplir : Dim SB= New TextureBrush(MonImage)<br>LinéarGradientBrush permet des dégradés (passage progressive d'une couleur à une autre).<br>Dim SB= New LinearGradientBrush(PointDébut, PointFin,Color1, Color2)<br>Les HatchBrush sont des brosses hachurées<br>Dim HatchBrush hb = new HatchBrush(HatchStyle.ForwardDiagonal, Color.Green,Color.FromArgb(100, Color.Yellow))<br>Les PathGradient sont des brosses plus complexes. |
| **Pen** (Stylet) | Utilisé pour dessiner des lignes et des polygones,<br>tels que des rectangles, des arcs et des secteurs.<br>Comment créer un Stylet ?<br>Dim blackPen As New Pen(Color.Black) on donne la couleur<br>Dim blackPen As New Pen(Color.Black, 3)on donne couleur et l'épaisseur |

| | |
|---|---|
| | Dim blackPen As New Pen(MyBrush) on peut même créer un stylet avec une brosse<br>Propriétés de ce Stylet :<br>DashStyle permet de faire des pointillés.<br>StartCap et EndCap définissent la forme du début et de la fin du dessin (rond, carré, flèche…) |
| **Font** (Police) | Utilisé pour décrire la police utilisée pour afficher le texte.<br>Dim Ft= New Font("Lucida sans unicode",60) 'paramètres=nom de la font et taille<br>Il y a de nombreuses surcharges.<br>Dim Ft= New Font("Lucida sans unicode",60, FontStyle.Bold)'pour écrire en gras |
| Color (Couleur) | Utilisé pour décrire la couleur utilisée pour afficher un objet particulier.<br>Dans GDI+, la couleur peut être à contrôle alpha.<br>System.Drawing.Color.Red pour le rouge |
| **Point** | Ils ont des cordonnées x, y ou des PointF 'avec des Singles<br>`Dim point1 As New Point(120, 120) ' Point avec des Integer` |
| **Rectangle** | Permet de définir un rectangle.<br>Dim r As New RectangleF(0, 0, 100, 100)<br>On remarque que le F après Point ou Rectangle veut dire 'Float', et nécessite l'usage de Single. |

**Comment faire ?**

Dessiner une ligne sur le graphique

Pour dessiner une ligne, on utilise **DrawLine**.

```
Dim blackPen As New Pen(Color.Black, 3)     'créer un stylet noir d'épaisseur 3
' Créer des points
Dim point1 As New Point(120, 120)     'créer des points
Dim point2 As New Point(600, 100)
' Dessine la ligne
e.Graphics.DrawLine(blackPen, point1, point2)
```

On aurait pu utiliser une surcharge de Drawline en spécifiant directement les coordonnées des points.

```
Dim x1 As Integer = 120
Dim y1 As Integer = 120
Dim x2 As Integer = 600
Dim y2 As Integer = 100
e.Graphics.DrawLine(blackPen, x1, y1, x2, y2)
```

Dessiner une ellipse

Définir un rectangle dans lequel sera dessinée l'ellipse.

```
Dim r As New RectangleF(0, 0, 100, 100)

g.DrawEllipse(New Pen(Color.Red), r)' Dessinons l'ellipse
```

Dessiner un rectangle

```
myGraphics.DrawRectangle(myPen, 100, 50, 80, 40)
```

Comme d'habitude on peut fournir après le stylet des coordonnées(4), des points (2) ou un rectangle.

Dessiner un polygone :

```
Dim MyPen As New Pen(Color.Black, 3)
' Créons les points qui définissent le polygone
Dim point1 As New Point(150, 150)
Dim point2 As New Point(100, 25)
Dim point3 As New Point(200, 5)
Dim point4 As New Point(250, 50)
Dim point5 As New Point(300, 100)
Dim point6 As New Point(350, 200)
Dim point7 As New Point(250, 250)
Dim curvePoints As Point() = {point1, point2, point3, point4, _
point5, point6, point7}
' Dessinons le Polygone.
e.Graphics.DrawPolygon(MyPen, curvePoints
```

Dessiner un rectangle plein :

```
e.FillRectangle(new SolidBrush(Color.red), 300,15,50,50)
```

Il existe aussi DrawArc, DrawCurve, DrawBezier DrawPie…

Écrire du texte sur le graphique

Pour cela on utilise la méthode DrawString de l'objet graphique :

```
g.DrawString ("Salut", Me.Font, New SolidBrush (ColorBlack), 10, 10)
```

Paramètres :

- texte à afficher ;

- police de caractères ;

- brosse, cela permet d'écrire avec des textures ;

- coordonnées.

Si on spécifie un rectangle à la place des 2 derniers paramètres,
le texte sera affiché dans le rectangle avec passage à la ligne si nécessaire :

```
Dim rectangle As New RectangleF (100, 100, 150, 150 )

Dim T as String= "Chaine de caractères très longue"

g.DrawString (T, Me.Font, New SolidBrush (ColorBlack), Rectangle)
```

On peut même imposer un format au texte.

Exemple : centrer le texte.

```
Dim Format As New StringFormat()

Format.Aligment=StringAlignment.Center

g.DrawString (T, Me.Font, New SolidBrush (ColorBlack), Rectangle, Format)
```

On peut mesurer la longueur (ou le nombre de lignes) d'une chaine.

Avec MeasureString

Exemple 1: centrer le texte : pour cela, calculer la longueur de la chaine, puis calculer le milieu de l'écran moins la 1/2 longueur de la chaine :

```
Dim W As Double= Me.DisplayRectangle.Width/2

Dim L As SizeF= e.Graphics.MeasurString (Texte, TextFont)

Dim StartPos As Double = W - (L.Width/2)

g.Graphics.MeasureString (T, Me.Font, New SolidBrush (ColorBlack), Rectangle, StartPos, 10)
```

Exemple 2 : calculer le nombre de lignes et le nombre de caractères d'une chaine :

```
g.Graphics.MeasureString (T, Me.Font, New SolidBrush (ColorBlack), Rectangle,New StringFormat()
_NombredeCaractères, NombredeLignes)
```

Ajouter une image sur le graphique

Pour cela on utilise la méthode DrawImage de l'objet graphique :

```
g.Graphics.DrawImage(New Bitmap("sample.jpg"), 29, 20, 283, 212)
```

On peut travailler avec des images .jpeg .png .bmp .Gif .icon .tiff .exif.

## X-G-3 - Travailler sur un Objet Image

### Charger une image

Si on veut utiliser une image bitmap ou vectoriel, il faut la mettre dans un Graphics. C'est la méthode DrawImage qui reçoit l'objet Metafile ou Bitmap comme argument. L'objet BitMap, si on le désire peut charger un fichier qui sera affiché.

```
Dim myBMP As New BitMap ("MonImage.bmp")

myGraphics.DrawImage(myBMP, 10, 10)
```

Le point de destination du coin supérieur gauche de l'image, (10, 10), est spécifié par les deuxième et troisième paramètres.

**myGraphics.FromImage(myBMP)** est aussi possible.

On peut utiliser plusieurs formats de fichier graphique : BMP, GIF, JPEG, EXIF, PNG, TIFF et ICON.

### Cloner une image

La classe Bitmap fournit une méthode **Clone** qui permet de créer une copie d'un objet existant. La méthode Clone admet comme paramètre un rectangle source qui vous permet de spécifier la portion de la Bitmap d'origine à copier. L'exemple suivant crée un objet Bitmap en clonant la moitié supérieure d'un objet Bitmap existant. Il dessine ensuite les deux images.

```
Dim originalBitmap As New Bitmap("Spiral.png")'on charge un fichier png dans un BitMap
Dim sourceRectangle As New Rectangle(0, 0, originalBitmap.Width, _
originalBitmap.Height / 2) 'on définit un rectangle

Dim secondBitmap As Bitmap = originalBitmap.Clone(sourceRectangle, _
PixelFormat.DontCare)
'on définit un second BitMap Clonant une partie du 1ere BitMap avec le rectangle

'On met les 2 BitMap dans un  Graphics

myGraphics.DrawImage(originalBitmap, 10, 10)
myGraphics.DrawImage(secondBitmap, 150, 10)
```

### Enregistrer une image sur disque

On utilise pour cela la méthode **Save**.

Exemple : enregistrer le BitMap newBitMap dans 'Image1.jpg' :

```
newBitmap.Save("Image1.jpg", ImageFormat.Jpeg)
```

## X-G-4 - Comment voir un Graphics ?

Si on a instancié un objet Graphics, on ne le voit pas (sauf s'il est lié à un contrôle). Pour le voir, il faut le mettre dans un PictureBox par exemple.

Exemple :

```
Dim newBitmap As Bitmap = New Bitmap(200, 200) 'créons un BitMap

Dim g As Graphics = Graphics.FromImage(newBitmap)'créons un Graphics et y mettre le BitMap

Dim r As New RectangleF(0, 0, 100, 100)' Dessinons une ellipse

g.DrawEllipse(New Pen(Color.Red), r)
```

Comment voir l'ellipse ?

Ajoutons un PictureBox au projet, et donnons à la propriété Image de ce PictureBox le nom du BitMap du Graphics :

```
PictureBox1.Image = newBitmap
```

L'ellipse rouge apparait !! Si ,Si !!

### Si on utilise un objet Graphics représentant la surface de dessin d'un contrôle.

Le fait de dessiner sur le graphics affiche automatiquement le dessin sur le contrôle.

Pour que le dessin soit 'persistant' voir l'exemple du logiciel de dessin ci-dessous.

### Paint déclenché par Resize

On peut récupérer la référence de l'objet Graphics argument de l'événement Paint d'un contrôle lorsque le contrôle est redessiné, un objet Graphics est fourni comme une valeur de PaintEventArgs.

Mais par défaut Paint n'est pas déclenché quand un contrôle ou formulaire est redimensionné.
Pour forcer à redessiner en cas de redimensionnement, il faut mettre le style Style.Resizedraw du formulaire ou du contrôle à true.

```
SetStyle (Style.Resizedraw, true)
```

Cette syntaxe marche, la suivante aussi (pour le formulaire)

```
Me.SetStyle (Style.Resizedraw, true) 'pour tous les objets du formulaire?
```

Mais PictureBox1.SetStyle (Style.Resizedraw, true) n'est pas accepté !!

## X-G-5 - Un exemple : Afficher un texte en 3D

Afficher un texte en 3d.

```
PrivateSub TextEn3D(ByVal g As Graphics, ByVal position As PointF, ByVal text AsString, ByVal
 ft As Font,
_ByVal c1 As Color, ByVal c2 As Color)

    Dim rect AsNew RectangleF(position, g.MeasureString(text, ft))

    Dim bOmbre AsNew LinearGradientBrush(rect, Color.Black, Color.Gray, 90.0F)


    g.DrawString(text, ft, bOmbre, position)


    position.X -= 2.0F

    position.Y -= 6.0F


    rect = New RectangleF(position, g.MeasureString(text, ft))

    Dim bDegrade AsNew LinearGradientBrush(rect, c1, c2, 90.0F)


    g.DrawString(text, ft, bDegrade, position)

EndSub
```

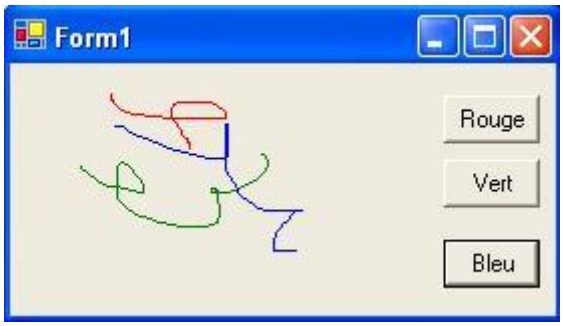
## X-G-6 - Espace de noms

Pour utiliser les graphiques, il faut que System.Drawing soit importé (ce qui est fait par défaut).

(System.Drawing.DLL comme références de l'assembly).

## X-G-7 - Programme simple de dessin

Un source très simple très didactique, pour dessiner avec la souris :




On appuie sur le bouton de la souris, on déplace la souris : cela dessine un trait.

Si on clique sur les boutons Rouge, Vert, Bleu cela change la couleur du trait.

Comment faire ?

Dans Form1 on crée : une PictureBox définissant la zone de dessin: PictureBox1.

Les boutons ButtonRouge, ButtonVert, ButtonBleu ayant dans leur propriété text respectivement 'Rouge', 'Vert', 'Bleu'.

Au niveau de la Classe on crée les variables ThePen de type Pen contenant la couleur en cours et PreviousPoint de type Point qui désigne le dernier point ou était la souris.

Les événements Click des boutons ButtonRouge, ButtonVert, ButtonBleu modifie la couleur en cours :

```
ThePen.Color = Color.Blue
```

Le dessin est effectué par les procédures événement de la souris.

MouseDown enregistre le précédent Point (les coordonnées de la souris sont fournies par le paramètre e, ce sont e.X et e.Y)

MouseMove : on teste si la souris est 'capturée' sur le PictureBox (c'est-à-dire si on est en glisser : déplacement de la souris bouton appuyé, cela donne à PictureBox1.Capture la valeur True).

Si la souris est capturée :

```
If PictureBox1.Capture Then
```

On récupère le Graphics de PictureBox1 :

```
Dim NewGraphic As Graphics = PictureBox1.CreateGraphics()
```

On trace sur le graphics une ligne du PreviousPoint au Point actuel avec ThePen :

```
NewGraphic.DrawLine(ThePen, PreviousPoint, NewPoint)
```

On détruit le graphics, puis on met dans PreviousPoint le point actuel.

Cela donne :

```
Imports System.Drawing
```

```
Public Class Form1

Inherits System.Windows.Forms.Form



Private PreviousPoint As Point ' Dernier point de la souris

Private ThePen As New System.Drawing.Pen(Color.Red) 'Un Pen pour dessiner.



Private Sub PictureBox1_MouseDown(ByVal sender As Object, ByVal e As
 System.Windows.Forms.MouseEventArgs)
_Handles PictureBox1.MouseDown

Dim NewPoint As New Point(e.X, e.Y)

PreviousPoint = NewPoint

End Sub




Private Sub PictureBox1_MouseMove(ByVal sender As Object, ByVal e As
 System.Windows.Forms.MouseEventArgs)
_Handles PictureBox1.MouseMove

If PictureBox1.Capture Then

Dim NewPoint As New Point(e.X, e.Y)

Dim NewGraphic As Graphics = PictureBox1.CreateGraphics()

NewGraphic.DrawLine(ThePen, PreviousPoint, NewPoint)

NewGraphic.Dispose()

PreviousPoint = NewPoint

End If

End Sub




Private Sub ButtonRouge_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
_Handles ButtonRouge.Clic

ThePen.Color = Color.Red

End Sub



Private Sub ButtonVert_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles ButtonVert.Clic

ThePen.Color = Color.Green

End Sub



Private Sub ButtonBleu_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
_Handles ButtonBleu.Clic

ThePen.Color = Color.Blue
```

```
End Sub

End Class
```

Dans une version VB6 de ce programme, je n'utilisais pas Capture, aussi j'utilisais une variable Globale 'Tracer' : MouseDown lui donnait la valeur True, MouseUp la valeur False et dans MouseMove on traçait si 'Tracer' était à True.

Mais c'est plus facile avec Capture.

Problème

Si une fenêtre est positionnée sur notre fenêtre de dessin puis enlevée, le dessin disparait !!

En VB6, la propriété AutoRedraw de l'image permettait de rappeler automatiquement le BitMap du dessin et de faire réapparaitre l'image, cela n'existe plus en VB.Net !!

A- On peut écrire une nouvelle méthode OnPaint de l'image (on la substitue grâce à Overrides), elle est déclenchée à chaque fois que le contrôle est redessiné ; on y met du code qui redessine (forme géométrique…).

```
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)

Dim myPen As New System.Drawing.Pen(System.Drawing.Color.Red)

Dim formGraphics As System.Drawing.Graphics

formGraphics = Me.CreateGraphics()

formGraphics.DrawLine(myPen, 0, 0, 200, 200)

myPen.Dispose()

formGraphics.Dispose()

End Sub
```

mais dans notre programme à moins d'enregistrer dans un tableau les coordonnées et couleur de chaque point puis de les redessiner un à un, ce n'est pas possible.

B- Comment faire l'équivalent de l'autoRedraw ?

```
Class Form1

Private PreviousPoint As Point ' Dernier point de souris

Private ThePen As New System.Drawing.Pen(Color.Red)

Private MyBitMap As Bitmap = New Bitmap(200, 200) 'Je crée un BitMap

Private NewPoint As New Point 'point actuel



Private Sub PictureBox1_MouseDown(ByVal sender As Object,
    _ByVal e As System.Windows.Forms.MouseEventArgs) Handles PictureBox1.MouseDown

NewPoint = New Point(e.X, e.Y)

PreviousPoint = NewPoint

End Sub
```

```
Private Sub PictureBox1_MouseMove(ByVal sender As Object, ByVal e As
 System.Windows.Forms.MouseEventArgs)
_Handles PictureBox1.MouseMove

If PictureBox1.Capture Then

Dim NewPoint As New Point(e.X, e.Y)

Dim newgraphic As Graphics = Graphics.FromImage(MyBitMap) 'je crée un Graphics à partir du BitMap


newgraphic.DrawLine(ThePen, PreviousPoint, NewPoint)      'je dessine sur le Graphics

PictureBox1.Image = MyBitMap                               'je passe le BitMap dans le PictureBox1



PreviousPoint = NewPoint

End If

End Sub


End Class
```

Quand on crée un Graphics depuis un contrôle comme dans la première version, le graphics ne pointe pas sur la propriété image, mais sur son affichage à l'instant T. Ce n'est pas mémorisé.

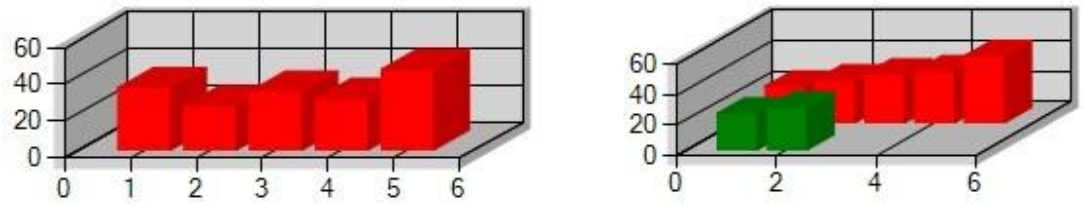
Beaucoup de contrôles .net ont une propriété image. À chaque rafraîchissement du contrôle si un bitmap est "attaché" à la propriété, le fond du contrôle est affiché automatiquement. Le picturebox est le plus simple de ces contrôles … il ne fait que ça.

Donc en fait pour dessiner sur un contrôle (quel qu'il soit) tu n'as que 2 possibilités :

- redessiner tout à chaque fois dans l'événement Onpaint ;
- lui affecter une image de fond et travailler sur celle-ci.

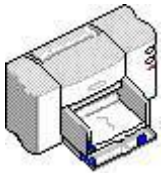
## X-G-8 - Faire un graphique

Comme cela:




Voir le cours sur Chart de Microsoft par moi-même ici : **https://plasserre.developpez.com/cours/chart/**

# X-H - Imprimer



Comment Imprimer ?

Prévoir une longue soirée, au calme, un bon siège, 1 g de paracétamol et un gros thermos de café !!

On verra que l'on peut utiliser pour imprimer :

A- la méthode .Net

soit avec un composant 'PrintDocument',

soit avec une instance de 'la Class PrintDocument' ;

B- on peut, en ajoutant une référence, imprimer comme en VB6 et c'est plus simple !!

## X-H-1 - Avec PrintDocument

### X-H-1-a - Imprimer 'Hello' avec le composant 'PrintDocument'

L'utilisateur clique sur un bouton, cela imprime 'Hello'.

Cet exemple utilise un 'composant PrintDocument'.

Comment faire en théorie ?

C'est le composant PrintDocument qui imprime.

En prendre un dans la boite à outils, le mettre dans un formulaire. Il apparait sous le formulaire et se nomme PrintDocument1.

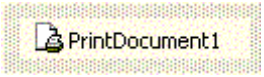
Pour imprimer, il faut utiliser la méthode Print de ce composant PrintDocument. Il faut donc écrire l'instruction suivante :

```
PrintDocument1.Print
```

Cette instruction appelle la procédure événement PrintDocument1_PrintPage du composant PrintDocument et qui contient la logique d'impression. Un paramètre de cet événement PrintPage est l'objet graphique envoyé à l'imprimante (nommé e). C'est à vous de dessiner dans l'objet graphique (e) ce que vous voulez imprimer. En fin de routine, l'objet graphique sera imprimé (automatiquement).

**En pratique**

- Je prends un PrintDocument dans la boite à outils, je le mets dans un formulaire. Il apparait sous le formulaire et se nomme PrintDocument1.




- Si je double-clique sur PrintDocument1 je vois apparaitre la procédure PrintDocument1_PrintPage (qui a été générée automatiquement) :

```
Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object,
_ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage

End Sub
```

C'est cette procédure qui est fondamentale et qui contient les routines d'impression écrites par le programmeur. Les routines d'impression agissent sur l'objet graphique qui sera utilisé pour imprimer, cet objet graphique est fourni dans les paramètres de la procédure (ici c'est e qui est de type PrintPageEventArgs) e.graphics est donc l'objet graphique sur lequel il faut écrire ou dessiner ce qui doit être imprimé.

- Dans cette routine PrintPage,on ajoute donc le code dessinant un texte (DrawString) sur l'objet graphique 'e' :

```
e.Graphics.DrawString("Hello", New Font("Arial", 80, FontStyle.Bold),
 Brushes.Black, 150, 125)
```

- Enfin je dessine un bouton nommé 'ButtonPrint' avec une propriété Text contenant "Imprimer Hello" et dans la procédure ButtonPrint_Click j'appelle la méthode Print

```
PrintDocument1.Print()
```

Voici le code complet :

```
Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object,
    _ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage

e.Graphics.DrawString("Hello", New Font("Arial", 80, FontStyle.Bold), Brushes.Black, 150, 125)

End Sub

Private Sub ButtonPrint_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
 ButtonPrint.Clic

PrintDocument1.Print()

End Sub
```

Si je clique sur le bouton 'ImprimerHello' cela imprime un gros 'Hello'.

*La méthode Print d'un PrintDocument déclenche l'événement PrintPage de ce PrintDocument qui contient le code dessinant sur le graphique de la page à imprimer. En fin de routine PrintPage le graphique est imprimé sur la feuille de l'imprimante.*

Toutes les méthodes graphiques écrivant, dessinant, traçant des lignes… sur un graphique permettent donc d'imprimer.

## X-H-1-a-i - Imprimer un dessin

Créons une ellipse bleue à l'intérieur d'un rectangle avec la position et les dimensions suivantes : début à 100, 150 avec une largeur de 250 et une hauteur de 250.

```
Private Sub PrintDocument1_PrintPage(ByVal sender As Object,
    _ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage
```

```
    e.Graphics.FillEllipse(Brushes.Blue, New Rectangle(100, 150, 250, 250))
End Sub
```

## X-H-1-a-ii - Afficher un Message Box indiquant 'Fin d'impression'

On a étudié l'événement PrintPage, mais il existe aussi les événements :

BeginPrint et EndPrint respectivement déclenchés en début et fin d'impression.

Il suffit d'utiliser l'événement EndPrint pour prévenir que l'impression est terminée :

```
Private Sub PrintDocument1_EndPrint(ByVal sender As Object, ByVal e As
 System.Drawing.Printing.PrintEventArgs)
_Handles PrintDocument1.EndPrint
    MessageBox.Show("Fin d'impression")
End Sub
```

On peut même fignoler et afficher "Fin d'impression de Nom du document".

Il faut avoir renseigné le DocumentName :

PrintDocument1.DocumentName = "MyTextFile"

Puis écrire :

```
Private Sub PrintDocument1_EndPrint(ByVal sender As Object, ByVal e As
 System.Drawing.Printing.PrintEventArgs)
_Handles PrintDocument1.EndPrint
    MessageBox.Show( "Fin d'impression de "+PrintDocument1.DocumentName)
End Sub
```

## X-H-1-b - Même programme : Imprimer 'Hello', mais avec la Classe PrintDocument

L'utilisateur clique sur un bouton, cela imprime 'Hello'.

Cet exemple utilise 'une instance de la Classe PrintDocument'. On ne met pas de composant 'PrintDocument' dans le formulaire.

Comment faire en théorie ?

Il faut importer l'espace de noms 'Printing' par :

```
Imports System.Drawing.Printing
```

Il faut créer une instance de la Classe PrintDocument dans le module.

```
Dim pd As new PrintDocument()
```

Il faut créer soi-même, une routine pd_PrintPage :

```
Private Sub pd_PrintPage(sender As object, ev As System.Drawing.Printing.PrintPageEventArgs)
End sub
```

Il faut indiquer le "lien" entre l'objet pd et la routine événement PrintPage

```
AddHandler pd.PrintPage, AddressOf Me.pd_PrintPage
```

Dans la procédure Button_Click d'un bouton "Imprimer" il faut appeler la méthode Print du PrintDocument pour effectuer l'impression du document :

pd.Print

Cela déclenche la procédure Private Sub pd_PrintPage précédemment écrite, dans laquelle on a ajouté :

```
ev.Graphics.DrawString ("Hello", printFont, Brushes.Black, leftMargin, yPos, new StringFormat()).
```

Cela donne le code complet :

```
Imports System.Drawing.Printing


Public Class Form1

Inherits System.Windows.Forms.Form


Dim pd As  New PrintDocument 'Assumes the default printer


Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles MyBase.Load

    AddHandler pd.PrintPage, AddressOf Me.Pd_PrintPage

End Sub


Private Sub Pd_PrintPage(ByVal sender As System.Object, ByVal e As
 System.Drawing.Printing.PrintPageEventArgs)

e.Graphics.DrawString("Hello", New Font("Arial", 80, FontStyle.Bold),   Brushes.Black, 150, 125)

End Sub


Private Sub ButtonPrint_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
_Handles ButtonPrint.Clic

    pd.Print()

End Sub



End Class
```

## X-H-1-b-i - Comment choisir l'imprimante ?

Le composant **PrintDialog** permet le choix de l'imprimante, de la zone à imprimer (tout, la sélection…) et donne accès aux caractéristiques de l'imprimante.

Comment l'utiliser ?

Il faut créer une instance de PrintDialog :

```
    Dim dlg As New PrintDialog
```

Il faut indiquer au PrintDialog sur quel PrintDocument travailler :

```
    dlg.Document = pd
```

Puis ouvrir la fenêtre PrintDialog avec la méthode ShowDialog.

L'utilisateur choisit son imprimante puis clique sur 'OK'.

**Exemple**

Quand l'utilisateur clique sur le bouton ButtonPrint ('Imprimer') la fenêtre PrintDialog s'ouvre.

Voici le code complet :

```
Private Sub ButtonPrint_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles ButtonPrint.Clic

Dim dlg As New PrintDialog

dlg.Document = pd

Dim result As DialogResult = dlg.ShowDialog()

If (result = System.Windows.Forms.DialogResult.OK) Then

    pd.Print()

End If


End Sub
```

## X-H-1-b-ii - Comment modifier la page à imprimer ?

Comment choisir d'imprimer en portrait ou paysage ? Modifier les marges…

Il faut utiliser un composant **PageSetUpDialog**.

Pour stocker les informations sur la page (marges…) il faut un PageSetting.

Je lie le PageSetting au PageSetUpDialog en donnant à la propriété PageSettings du PageSetUpDialog le nom du PageSetting.

Puis j'ouvre le PageSetUpDialog.

Au retour le PageSetting contient les modifications, je les 'passe' au PrintDocument avant d'imprimer.

Cela donne :

```
Dim psDlg As New PageSetupDialog

Dim LePageSettings As New PageSettings

psDlg. PageSettings = LePageSettings

psDlg.ShowDialog()

pd.DefaultPageSettings = LePageSettings
```

## X-H-1-c - Prévisualisation de la page à imprimer ?

On utilise pour cela un **PrintPreviewDialog**, on lui indique quel PrintDocument pré visualiser en l'assignant à sa méthode document puis on l'affiche par ShowDialog().

```
Dim dllg As New PrintPreviewDialog

dllg.Document = pd

dllg.ShowDialog()
```

## X-H-1-d - Construction d'une application d'impression complexe

Comment imprimer le contenu d'un fichier texte ?

Tous les didacticiels (Microsoft compris) donnent cet exemple.

La première chose que vous devez faire est d'écrire votre logique d'impression. Pour cela, quand la méthode PrintDocument.Print() est appelée, les événements suivants sont déclenchés :

- BeginPrint ;
- PagePrint (un ou plusieurs s'il y a plusieurs pages à imprimer) ;
- EndPrint.

L'argument d'événement de PagePrint (PagePrintEventArgs) comprend une propriété HasMorePages. Si celle-ci a la valeur True lors du retour de votre gestionnaire d'événements, PrintDocument définit une nouvelle page et déclenche de nouveau l'événement PagePrint.

Voyons la logique dans votre gestionnaire d'événements PagePrint.

- Imprimez le contenu de la page en utilisant les informations des arguments d'événement. Les arguments d'événement contiennent l'objet Graphics pour l'imprimante, le PageSettings pour cette page, les limites de la page, et la taille des marges.
  Il faut dans la procédure PagePrint imprimer ligne par ligne en se déplaçant à chaque fois vers le bas d'une hauteur de ligne.
  Pour 'simplifier', on considère que chaque ligne ne déborde pas à droite !! :
- déterminer s'il reste des pages à imprimer ;
- si c'est le cas, HasMorePages doit être égal à True ;
- s'il n'y a pas d'autres pages, HasMorePages doit être égal à False.

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

    …

    private printFont As Font
    private streamToPrint As StreamReader

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

    'Événement survenant lorsque l'utilisateur clique sur le bouton 'Imprimer'
    Private Sub printButton_Click(sender As object, e As System.EventArgs)

        Try
            streamToPrint = new StreamReader ("PrintMe.Txt")
            Try
```

```
                printFont = new Font("Arial", 10)
                Dim pd as PrintDocument = new PrintDocument() 'déclaration du PrintDocument
                AddHandler pd.PrintPage, AddressOf Me.pd_PrintPage
                pd.Print()
            Finally
                streamToPrint.Close()
            End Try

        Catch ex As Exception
            MessageBox.Show("Une erreur est survenue: - " + ex.Message)
        End Try

    End Sub

    'Événement survenant pour chaque page imprimer
    Private Sub pd_PrintPage(sender As object, ev As System.Drawing.Printing.PrintPageEventArgs)

        Dim lpp As Single = 0     'nombre de ligne par page
        Dim yPos As Single =  0   'ordonnée
        Dim count As Integer = 0 'numéro de ligne
        Dim leftMargin As Single = ev.MarginBounds.Left
        Dim topMargin As Single = ev.MarginBounds.Top
        Dim line as String

        'calcule le nombre de lignes par page
        ' hauteur de la page/hauteur de la police de caractère
        lpp = ev.MarginBounds.Height  / printFont.GetHeight(ev.Graphics)


        'lit une ligne dans le fichier
        line=streamToPrint.ReadLine()


        'Boucle affichant chaque ligne

        while (count < lpp AND line <> Nothing)

            yPos = topMargin + (count * printFont.GetHeight(ev.Graphics))

            'Ecrit le texte dans l'objet graphique
            ev.Graphics.DrawString (line, printFont, Brushes.Black, leftMargin, _
                                    yPos, new StringFormat())

            count = count + 1

            if (count < lpp) then
                line=streamToPrint.ReadLine()
            end if

        End While

        'S'il y a encore des lignes, on réimprime une page
        If (line <> Nothing) Then
            ev.HasMorePages = True
        Else
            ev.HasMorePages = False
        End If

    End Sub

    …

End Class
```

On a vu que pour 'simplifier', on considère que chaque ligne ne déborde pas à droite.

Dans la pratique, pour gérer les retours à la ligne automatiques on peut dessiner dans un rectangle en utilisant une surcharge de DrawString.

```
Dim rectangle As New RectangleF (100, 100, 150, 150 )

Dim T as String= "Chaine de caractères très longue"

g.DrawString (T, Me.Font, New SolidBrush (ColorBlack), Rectangle)
```

On peut mesurer la longueur (ou le nombre de lignes) d'une chaine : avec MeasureString.

(Voir la page sur les graphiques.)

## X-H-1-e - Propriétés du 'PrintDocument'

On peut sans passer par une 'boite de dialog' gérer directement l'imprimante, les marges, le nombre de copies…

Si pd est le PrintDocument :

pd.PrinterSetting désigne l'imprimante en cours ;

pd.PrinterSetting.PrinterName retourne ou définit le nom de cette imprimante ;

pd.PrinterSetting.Printerresolution donne la résolution de cette imprimante ;

pd.PrinterSetting.installedPrinted donne toutes les imprimantes installées.

La propriété DefaultPageSetting est en rapport avec les caractéristiques de la page :

pd.PrinterSetting.DefaultPageSetting.Margins donne les marges ;

pd.PrinterSetting.PrinttoFile permettrait d'imprimer dans un fichier (non testé).

## X-H-1-f - Imprime le formulaire en cours

Exemple fourni par Microsoft.

La routine CaptureScreen capture l'image du formulaire en cours et la met dans memoryImage. puis memoryImage est passé dans l'objet graphique e qui est imprimé.

```
Private Declare Function BitBlt Lib "gdi32.dll" Alias "BitBlt" (ByVal _
   hdcDest As IntPtr, ByVal nXDest As Integer, ByVal nYDest As _
   Integer, ByVal nWidth As Integer, ByVal nHeight As Integer, ByVal _
   hdcSrc As IntPtr, ByVal nXSrc As Integer, ByVal nYSrc As Integer, _
   ByVal dwRop As System.Int32) As Long
Dim memoryImage As Bitmap
Private Sub CaptureScreen()
   Dim mygraphics As Graphics = Me.CreateGraphics()
   Dim s As Size = Me.Size
   memoryImage = New Bitmap(s.Width, s.Height, mygraphics)
   Dim memoryGraphics As Graphics = Graphics.FromImage(memoryImage)
   Dim dc1 As IntPtr = mygraphics.GetHdc
   Dim dc2 As IntPtr = memoryGraphics.GetHdc
   BitBlt(dc2, 0, 0, Me.ClientRectangle.Width, _
      Me.ClientRectangle.Height, dc1, 0, 0, 13369376)
   mygraphics.ReleaseHdc(dc1)
   memoryGraphics.ReleaseHdc(dc2)
End Sub
Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object, _
   ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles _
   PrintDocument1.PrintPage
   e.Graphics.DrawImage(memoryImage, 0, 0)
```

```
End Sub
Private Sub PrintButton_Click(ByVal sender As System.Object, ByVal e As _
   System.EventArgs) Handles PrintButton.Clic
   CaptureScreen()
   PrintDocument1.Print()
End Sub
```

## X-H-1-g - Imprime un contrôle DataGrid

Exemple fourni par Microsoft.

**Cet exemple nécessite :**

- un contrôle Button, nommé ImprimerGrid, dans le formulaire ;
- un contrôle DataGrid nommé DataGrid1 ;
- un composant PrintDocument nommé PrintDocument1.

Comme d'habitude PrintPage imprime e.Graphics.

D'après ce que j'ai compris, l'événement Paint redessine un contrôle,

mais on peut choisir le contrôle et l'endroit ou le redessiner.

Je redessine donc grâce à Paint, le DataGrid dans e.graphics.

PaintEventArgs fournit les données pour l'événement Paint :

PaintEventArgs spécifie l'objet graphics à utiliser pour peindre le contrôle, ainsi que le ClipRectangle dans lequel le peindre.

InvokePaint déclenche l'événement Paint :

```
Private Sub ImprimerGrid_Click(ByVal sender As System.Object, ByVal e As _
   System.EventArgs) Handles PrintGrid.Clic
   PrintDocument1.Print()
End Sub

Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object, _
   ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles _
   PrintDocument1.PrintPage
   Dim myPaintArgs As New PaintEventArgs(e.Graphics, New Rectangle(New _
      Point(0, 0), Me.Size))
   Me.InvokePaint(DataGrid1, myPaintArgs)
End Sub
```

Imprimer le texte d'un RichTextBox

Voir Imprimer le contenu d'une RichtextBox.

Simple, non !! Je plaisante !!

## X-H-2 - Imprimer comme en VB6 avec un objet 'Printer'

Microsoft propose un PowerPack de compatibilité vb6 le 'Microsoft.VisualBasic.PowerPacks.Printing.Printer' qui ajoute à VB 2005 une Classe permettant d'imprimer 'comme en VB6'; cela permet d'utiliser votre code de routine d'impression VB6 ou de créer des routines d'impression beaucoup plus simple !!

Télécharger le PowerPack Printercompatibility 1 ici : http://msdn2.microsoft.com/en-us/vbasic/bb219077.aspx

Après avoir installé le Pack, allez dans le menu Project cliquez sur 'Ajouter une référence'.

Sur l'onglet NET, cliquez sur Microsoft.VisualBasic.PowerPacks.Printing.Printer, puis sur OK.

Dans le Code, ajoutez en haut du module :

**Imports Microsoft.VisualBasic.PowerPacks.Printing.Compatibility.VB6**

Maintenant on peut utiliser du code VB6 pour imprimer :

```
Dim pr As New Printer        'Instanciation de l'imprimante par défaut

Dim pFont As New Font("Arial", 14)    'Une nouvelle font

pr.Font = pFont

pr.Print("This text will print in 14 point ")  'Texte à imprimer

pr.EndDoc()    'fin, on lance l'impression
```

Simple, non !! Je ne plaisante pas !!

DrawMode, DriverName, hDC, Port, TrackDefault, et Zoom n'existent plus.

Par contre il y a en plus: PrintAction property qui permet un print preview ou permet d'imprimer dans un fichier.

Il y a aussi la PrinterCollection class qui contient les imprimantes. La global Printers collection permet de sélectionner une imprimante.

## X-I - Faire une aide pour l'utilisateur

Quand un utilisateur utilise votre logiciel, il se pose parfois des questions, comment l'aider ?

Avec des aides que le programmeur doit créer et ajouter au programme.

### X-I-1 - Généralisées sur les 4 sortes d'aide

A La Class Help permet d'ouvrir un fichier d'aide.
B Le composant HelpProvider offre 2 types d'aide :

  A le premier consiste à ouvrir un fichier d'aide grâce à F1 ;
  B quant au second, il peut afficher une aide brève pour chacun des contrôles en utilisant le bouton d'aide (?). Il s'avère particulièrement utile dans les boites de dialogue modales.

C Le composant ToolTip offre lui :

  A une aide propre à chaque contrôle des Windows Forms.

D Le composant ErrorProvider est un moyen de signaler une erreur dans un contrôle.

## X-I-2 - Les fichiers d'aide

On peut utiliser les formats :

**HTML** Fichier .htm ;

**HTMLHelp** 1.x ou version ultérieure Fichier .chm ;

**HLP Fichier** .hlp les plus anciens.

**Comment créer ces fichiers**

Pour les fichiers HTM :

Utiliser Word, ou FrontPage, ou Netscape Composeur…

Pour les fichiers HLP :

Utiliser Microsoft HelpWorkshop livré avec VB6 (les fichiers .HLP fonctionnent nativement sous Windows 98 et XP, mais ne fonctionne plus sous Windows Vista ; si on veut quand même les utiliser, il faut télécharger puis installer WinHelp32.exe pour Vista.

Pour les fichiers CHM :

Thierry AIM fournit sur le site developpez.com un excellent cours :

http://thierryaim.developpez.com/tutoriel/chm/

**On conseille d'utiliser les fichiers chm.**

## X-I-3 - Utilisation des fichiers d'aide

### X-I-3-a - Appel direct

**La classe Help** permet d'ouvrir directement par code un fichier d'aide.

C'est ce qu'on utilise dans le menu '?' d'un programme (sous-menu 'Aide') ; dans la procédure correspondante (Sub Aide_Click) on écrit :

```
Help.ShowHelp (Me, "MonAide.html")
```

MonAide.html doit être dans le répertoire de l'application (répertoire Bin).

Cela peut être une URL, l'adresse d'une page sur Internet !!

Il peut y avoir un 3e paramètre : on verra cela plus bas (c'est le même paramètre que la propriété HelpNagigator de HelpProvider).

```
Help.ShowHelp(Me, "file://C:\Windows\Help\calc.chm", HelpNavigator.TableOfContents)
```

## X-I-3-b - Appel par F1

Vous pouvez utiliser le composant **HelpProvider** pour attacher des rubriques d'aide figurant dans un fichier d'aide (au format HTML, HTMLHelp 1.x ou ultérieur) à des contrôles de l'interface utilisateur.

Quand on met un composant HelpProvider dans un formulaire (avec dans la propriété **HelpNamespace**, le nom de fichier d'aide), cela ajoute aux contrôles de ce formulaire les propriétés :

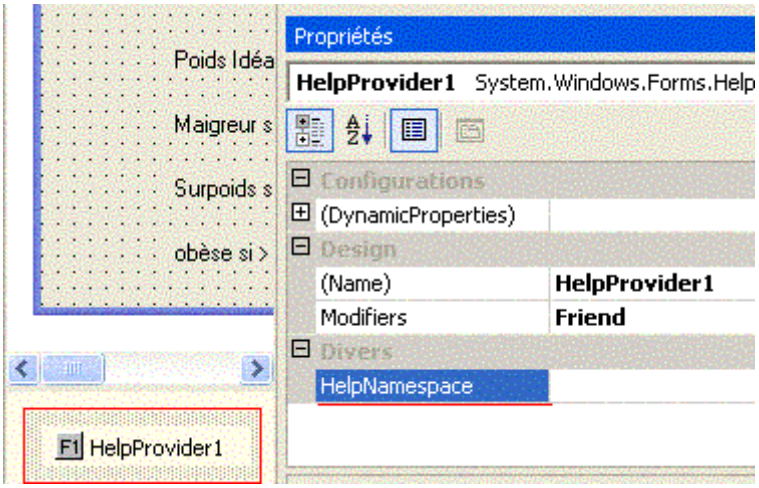
- **HelpNavigator** qui détermine le type d'appel (par numéro de rubrique, mot-clé…) ;
- **HelpKeyword** qui contient le paramètre de recherche (le numéro de rubrique, le mot-clé…).

Quand l'utilisateur est sur le contrôle et qu'il clique sur F1 la rubrique d'aide s'ouvre.

Pour créer cette aide :

faites glisser un composant HelpProvider de la boite à outils vers votre formulaire ;

le composant se place dans la barre d'état située au bas de la fenêtre.




Dans la fenêtre Propriétés du HelpProvider, donner à la propriété HelpNamespace, un nom de fichier d'aide .chm, ou .htm.

Dans la fenêtre propriétés du contrôle qui doit déclencher l'aide, donner à la propriété HelpNavigator une valeur de l'énumération HelpNavigator.

Cette valeur détermine la façon dont la propriété HelpKeyword est passée au système d'aide. HelpNagigator peut prendre la valeur :

| | |
|---|---|
| AssociateIndex | Indique que l'index d'une rubrique spécifiée est exécuté dans l'URL spécifiée. |
| Find | Indique que la page de recherche d'une URL spécifiée est affichée. |
| Index | Indique que l'index d'une URL spécifiée est affiché. |
| KeywordIndex | Spécifie un mot-clé à rechercher et l'action à effectuer dans l'URL spécifiée. |
| TableOfContents | Indique que le sommaire du fichier d'aide HTML 1.0 est affiché. |
| Topic | Indique que la rubrique à laquelle l'URL spécifiée fait référence est affichée. |

Définissez la propriété HelpKeyword dans la fenêtre Propriétés. (La valeur de cette propriété sera passée au fichier d'aide afin de déterminer la rubrique d'aide à afficher.)

Au moment de l'exécution, le fait d'appuyer sur F1 lorsque le contrôle (dont vous avez défini les propriétés HelpKeyword et HelpNavigator) a le focus ouvre le fichier d'aide associé à ce composant HelpProvider.

Remarque

Vous pouvez définir, pour la propriété HelpNamespace, une adresse http:// (telle qu'une page Web). Cela permet d'ouvrir le navigateur par défaut sur la page Web avec la chaine indiquée dans la propriété HelpKeyword utilisée comme ancre (pour accéder à une section spécifique d'une page HTML).

Dans le code il faut utiliser la syntaxe HelpProvider.SetHelpKeyword="…"

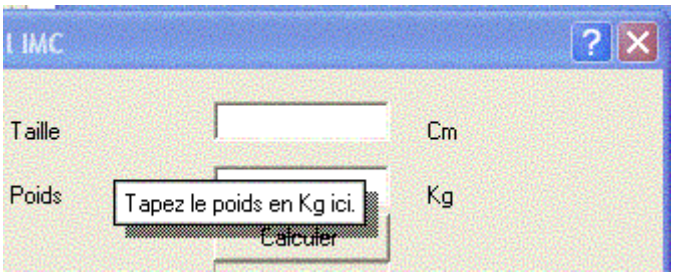
Exemple

Pour afficher la page d'aide sur les formes ovales, sélectionnez la valeur HelpNavigator.KeyWordIndex dans la liste déroulante Help Navigator ; dans la zone de texte HelpKeyword, 'tapez "ovales" (sans guillemets).

## X-I-3-c - Utilisation du bouton d'aide

Vous pouvez afficher l'aide pour un contrôle via le bouton Aide (?) situé dans la partie droite de la barre de titre.

Il faut que l'utilisateur clique sur le bouton d'aide (?) puis sur le contrôle qui nécessite une aide, ce qui entraine l'ouverture d'un carré blanc contenant un message d'aide.




L'affichage de l'aide de cette façon convient particulièrement aux boites de dialogue. En effet, avec un affichage modal des boites de dialogue, il n'est pas facile d'ouvrir des systèmes d'aide externe, dans la mesure où les boites de dialogue modales doivent être fermées avant que le focus puisse passer à une autre fenêtre. Le bouton Réduire ou Agrandir ne doit pas être affiché dans la barre de titre. Il s'agit d'une convention pour les boites de dialogue alors que les formulaires disposent généralement de boutons Réduire et Agrandir.

Pour afficher l'aide contextuelle

Faites glisser un composant HelpProvider de la boite à outils vers votre formulaire.

Le contrôle est placé dans la barre d'état des composants située au bas de la fenêtre.

Attribuer aux propriétés Minimize et Maximize de la fenêtre la valeur false.

Puis :

dans la fenêtre Propriétés de la fenêtre, donner à la propriété HelpButton la valeur true. Cette configuration permet d'afficher dans la partie droite de la barre de titre du formulaire un bouton contenant un point d'interrogation ;

sélectionnez le contrôle pour lequel vous souhaitez afficher l'aide dans votre formulaire et mettre dans la propriété HelpString la chaine de texte qui sera affichée dans une fenêtre de type ToolTip.

Test

Appuyez sur F5.

Appuyez sur le bouton Aide (?) de la barre de titre et cliquez sur le contrôle dont vous avez défini la propriété HelpString. Le toolTip apparait.

## X-I-3-d - Utilisation des info bulles : ToolTip

Le composant **ToolTip** peut servir à afficher des messages d'aide courts et spécialisés relatifs à des contrôles individuels.

Cela ouvre une petite fenêtre indépendante rectangulaire dans laquelle s'affiche une brève description de la raison d'être d'un contrôle lorsque le curseur de la souris pointe sur celui-ci.

Il fournit une propriété qui précise le texte affiché pour chaque contrôle du formulaire.

En outre, il est possible de configurer, pour le composant ToolTip, le délai qui doit s'écouler avant qu'il ne s'affiche.

Comment faire

1- Ajoutez le contrôle ToolTip au formulaire à partir de la ToolBox. Il se met en bas.

Chaque contrôle à maintenant, dans ses propriétés, une propriété **ToolTip** ou on peut mettre le texte a afficher dans l'info bulle.

2- Utilisez la méthode SetToolTip du composant ToolTip.

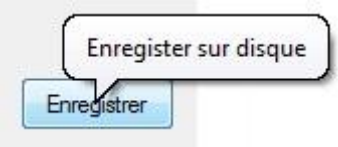
On peut aussi le faire par code :

```
ToolTip1.SetToolTip(Button1, "Save changes")
```

3- Par code créons de toute pièce un ToolTip.

```
Dim toolTip1 As New ToolTip()

' modifions les délais du ToolTip.
toolTip1.AutoPopDelay = 6000  'nombre de millisecondes d'affichage
```

```
toolTip1.InitialDelay = 2000 'nombre de millisecondes avant l'affichage
toolTip1.ReshowDelay = 500
' Force le ToolTip a être visible que la fenêtre soit active ou non .
toolTip1.ShowAlways = True

' donne le texte de l'info bulle à 2 contrôles.
toolTip1.SetToolTip(Me.button1, "My button1")
toolTip1.SetToolTip(Me.checkBox1, "My checkBox1")
toolTip1.IsBalloon= True 'Donne une forme de ballon comme dans les BD
```




On peut aussi modifier les couleurs ajouter un titre, une icône au ToolTip.
(ToolTipIcon, ToolTipTitle, BackColor, ForeColor).

## X-I-3-e - Utilisation d'ErrorProvider

Le composant ErrorProvider peut servir à afficher un panneau d'erreur et un message d'aide court.

Exemple : une zone TextBox doit permettre de saisir une valeur numérique. Si cela n'est pas le cas et qu'on tente de sortir du textbox ou de fermer la fenêtre, le panneau (!) s'affiche , et on ne peut pas sortir le focus du textbox.

Si on survole le panneau(!) cela affiche le message.


Pour cela on utilise l'événement Validating du textBox qui est déclenché quand on tente de sortir, si le texte n'est par numérique, on donne le message au ErrorProvider et on annule la sortie (e.Cancel=True) :

```
Private Sub MyBox_Validating(ByVal sender As Object, ByVal e As
 System.ComponentModel.CancelEventArgs)
_Handles MyBox.Validating

If Not IsNumeric(Me.MyBox.Text) Then

    Me.ErrorProvider1.SetError(Me.MyBox, "L'entrée doit être numérique")

    e.Cancel = True

Else

    Me.ErrorProvider1.SetError(Me.MyBox, "")

End If

End Sub
```

# X-J - Appeler une API

Les **API** (Application Programming Interface) permettent d'utiliser des bibliothèques de liaisons dynamiques (DLL, Dynamic-Link Libraries), qui contiennent des fonctions (généralement écrites en C) et qui sont compilées dans une DLL.

Les Dll contiennent donc des procédures qu'on peut appeler par les API.

Elles font :

**- soit partie intégrante du système d'exploitation Windows.**(API Windows).

Ce sont ces API (Kernel32.Dll=cœur du système, User32.Dll= fonctionnement des applications, gdi32…dll=interface graphique) que Windows utilise pour fonctionner.

Les fonctions sont donc écrites pour Windows ; parfois on n'a pas d'équivalent VB, aussi, plutôt que de les réécrire quand on en a besoin, on appelle celles de Windows.

Elles permettent d'effectuer des tâches lorsqu'il s'avère difficile d'écrire des procédures équivalentes. Par exemple, Windows propose une fonction nommée FlashWindowEx qui vous permet de varier l'aspect de la barre de titre d'une application entre des tons clairs et foncés.

Il faut avouer que, le Framework fournissant des milliers de classes permettant de faire pratiquement tout ce que font les API Windows, on a rarement besoin d'utiliser les Api Windows. Chaque fois que cela est possible, vous devez utiliser du code managé à partir du .NET Framework plutôt que les appels API Windows pour effectuer des tâches ;

**- soit partie de dll spécifiques fournies par des tiers** pour permettre d'appeler des fonctions n'existant pas dans VB ni Windows.

Par exemple, il existe des Api MySql qui donnent accès aux diverses fonctions permettant d'utiliser une base de données MySql. (Ces API contiennent 'le moteur' de la base de données.)

Les Api Windows sont en code non managé. De plus elles n'utilisent souvent pas les mêmes types de données que VB . L'appel des API se faisant avec des passages de paramètres, il y a des précautions à prendre!! Sinon cela plante!!!cela plante vraiment.

Il y a des API .Net en code managé.

## X-J-1 - Appel d'une fonction dans une API

Il faut déclarer la fonction (contenue dans la Dll) en haut du module :

```
Declare  Function  MyFonction Lib "MyLibary.dll" () as Integer
```

On indique ici qu'on veut utiliser la fonction MyFonction située dans la dll MyLibrary.dll; cette fonction retourne un Integer.

Ensuite on peut utiliser MyFonction dans le code :

```
Dim a = MyFonction()
```

Exemple

On va récupérer le nom de l'utilisateur en appelant la fonction **GetUserNameA** de la dll "advapi32.dll"

```
Declare Function getUserName Lib "advapi32.dll" Alias "GetUserNameA"(ByVal
 lpBuffer As String, ByRef nSize As Integer)
_As Integer

Sub Test

Dim buffer As String = New String(Char(" "), 25)

Dim retVal As Integer = getUserName(buffer, 25)

Dim userName As String = Strings.Left(buffer, InStr(buffer, Chr(0)) - 1)

End Sub
```

Le terme **Alias** permet de donner un nom a la fonction (getusername) dans le module alors que le vrai nom dans la dll est différent (GetUserNameA).

## X-J-2 - Les API Windows

L'avantage de l'utilisation d'API Windows dans votre code réside dans le gain de temps de développement, car elles contiennent des centaines de fonctions utiles déjà écrites et prêtes à être utilisées. L'inconvénient des API Windows est qu'elles peuvent être complexes à utiliser et implacables lorsqu'une opération se déroule mal.

Pour plus d'informations sur les API Windows, consultez la documentation du kit de développement Win32 SDK dans les API Windows du kit de développement Platform SDK. Pour plus d'informations sur les constantes utilisées par les API Windows, examinez les fichiers d'entête, tels que Windows.h, fournis avec le kit de développement Platform SDK. MSDN donne aussi une description des API

**Appels API avec Declare**

La façon la plus courante d'appeler les API Windows consiste à utiliser l'instruction Declare.

Exemple (repris de chez Microsoft) : appel de la fonction Windows 'MessageBox' qui est dans user32.dll et qui affiche une MessageBox.

- **Rechercher de la documentation de la fonction :** MSDN et les API donnent la définition de la fonction MesssageBox :

```
int MessageBox(
    HWND hWnd,
    LPCTSTR lpText,
    LPCTSTR lpCaption,
    UINT uType
);
```

Parameters

hWnd
[in] Handle to the owner window of the message box to be created. If this parameter is NULL, the message box has no owner window.
lpText
[in] Pointer to a null-terminated string that contains the message to be displayed.
lpCaption
[in] Pointer to a null-terminated string that contains the dialog box title. If this parameter is NULL, the default title Error is used.
uType

[in] Specifies the contents and behavior of the dialog box. This parameter can be a combination of flags from the following groups of flags.

Constantes API Windows : vous pouvez déterminer la valeur numérique des constantes utilisées dans les API par l'examen des instructions #define dans le fichier WinUser.h. Les valeurs numériques sont généralement affichées au format hexadécimal. Par conséquent, vous pouvez les convertir au format décimal. Par exemple, si vous voulez combiner les constantes pour le style exclamation MB_ICONEXCLAMATION 0x00000030 et le style Oui/Non MB_YESNO 0x00000004, vous pouvez ajouter les nombres et obtenir un résultat de 0x00000034, ou 52 décimales.

Return Value

| | |
|---|---|
| IDABORT | Abort button was selected |
| IDCANCEL | Cancel button was selected. |
| IDCONTINUE | Continue button was selected. |
| IDIGNORE | Ignore button was selected. |
| IDNO | No button was selected |
| IDOK | OK button was selected. |
| IDRETRY | Retry button was selected. |
| IDTRYAGAIN | Try Again button was selected |
| IDYES | Yes button was selected |

- **Il faut déclarer la procédure DLL**
  Ajoutez la fonction Declare suivante à la section de déclaration du formulaire de départ de votre projet ou à celle de la classe ou du module où vous voulez utiliser la DLL :

```
Declare Auto Function MBox Lib "user32.dll" _
Alias "MessageBox" (ByVal hWnd As Integer, _
   ByVal txt As String, ByVal caption As String, _
   ByVal Typ As Integer) As Integer
```

Declare comprend les éléments suivants :

le modificateur Auto indique de suivre les règles du Common Language Runtime ;

le nom qui suit Function est celui que votre programme utilise pour accéder à la fonction importée ;

le mot-clé Alias indique le nom réel de cette fonction ;

Lib suivi du nom et de l'emplacement de la DLL qui contient la fonction que vous appelez. Vous n'avez pas besoin d'indiquer le chemin d'accès des fichiers situés dans les répertoires système Windows.

Utilisez le mot-clé Alias si le nom de la fonction que vous appelez n'est pas un nom de procédure Visual Basic valide ou est en conflit avec le nom d'autres éléments de votre application. Alias indique le nom réel de la fonction appelée.

Les types de données que Windows utilise ne correspondent pas à ceux de Visual Studio. Visual Basic effectue la plupart des tâches à votre place en convertissant les arguments en types de données compatibles, processus appelé marshaling. Vous pouvez contrôler de manière explicite la façon dont les arguments sont marshalés en utilisant l'attribut MarshalAs défini dans l'espace de noms System.Runtime.InteropServices.

*Remarque Les versions antérieures de Visual Basic vous autorisaient à déclarer des paramètres As Any (tout type). Visual Basic .NET ne le permet pas.*

**Ajoutez des instructions Const à la section des déclarations de votre classe ou module pour rendre ces constantes disponibles pour l'application. Par exemple :**

```
Const MB_ICONQUESTION = &H20L
Const MB_YESNO = &H4
Const IDYES = 6
Const IDNO = 7
```

**Pour appeler la procédure DLL :**

```
    Dim RetVal As Integer ' Valeur de retour.

    RetVal = MBox(0, "Test DLL", "Windows API MessageBox", _
                  MB_ICONQUESTION Or MB_YESNO)
    If RetVal = IDYES Then
       MsgBox("Vous avez cliqué sur OUI")
    Else
       MsgBox("Vous avez cliqué sur NON")
    End If
```

Visual Basic .NET convertit automatiquement les types de données des paramètres et valeurs de retour pour les appels API Windows, mais vous pouvez utiliser l'attribut MarshalAs pour indiquer de façon explicite les types de données non managés attendus par une API.

On peut aussi appeler une API Windows à l'aide de l'attribut **DllImport,** mais c'est compliqué.

## X-J-3 - Autre exemple classique

Utilisation de la routine **BitBlt** qui déplace des octets.

**La documentation donne les renseignements suivants :**

```
      Declare Function BitBlt Lib "gdi32" ( _
         ByVal hDestDC As Long, _
         ByVal x As Long, _
         ByVal y As Long, _
         ByVal nWidth As Long, _
         ByVal nHeight As Long, _
         ByVal hSrcDC As Long, _
         ByVal xSrc As Long, _
         ByVal ySrc As Long, _
         ByVal dwRop As RasterOps _
         ) As Long
```

Parameter Information

· hdcDest
Identifies the destination device context.

· nXDest
Specifies the logical x-coordinate of the upper-left corner of the destination rectangle.

· nYDest
Specifies the logical y-coordinate of the upper-left corner of the destination rectangle.

· nWidth
Specifies the logical width of the source and destination rectangles.

· nHeight

Specifies the logical height of the source and the destination rectangles.

· hdcSrc
Identifies the source device context.

· nXSrc
Specifies the logical x-coordinate of the upper-left corner of the source rectangle.

· nYSrc
Specifies the logical y-coordinate of the upper-left corner of the source rectangle.

· dwRop
Specifies a raster-operation code.

**Les Constantes dwRop :**

```
        ' Copies the source bitmap to destination bitmap
         SRCCOPY = &HCC0020
        '
        ' Combines pixels of the destination with source bitmap using the Boolean AND operator.
         SRCAND = &H8800C6
        '
        ' Combines pixels of the destination with source bitmap using the Boolean XOR operator.
         SRCINVERT = &H660046
        '
        ' Combines pixels of the destination with source bitmap using the Boolean OR operator.
         SRCPAINT = &HEE0086
        '
        ' Inverts the destination bitmap and then combines the results with the source bitmap
        ' using the Boolean AND operator.
         SRCERASE = &H4400328
        '
        ' Turns all output white.
         WHITENESS = &HFF0062
        '
        ' Turn output black.
         BLACKNESS = &H42
```

**Return Values :**

If the function succeeds, the return value is nonzero.

**Ici on va utiliser cette routine pour copier l'image de l'écran dans un graphics.**

```
Private Declare Function BitBlt Lib "gdi32.dll" Alias "BitBlt" (ByVal _
   hdcDest As IntPtr, ByVal nXDest As Integer, ByVal nYDest As _
   Integer, ByVal nWidth As Integer, ByVal nHeight As Integer, ByVal _
   hdcSrc As IntPtr, ByVal nXSrc As Integer, ByVal nYSrc As Integer, _
   ByVal dwRop As System.Int32) As Long
Dim memoryImage As Bitmap

Private Sub CaptureScreen()
   Dim mygraphics As Graphics = Me.CreateGraphics()
   Dim s As Size = Me.Size
   memoryImage = New Bitmap(s.Width, s.Height, mygraphics)
   Dim memoryGraphics As Graphics = Graphics.FromImage(memoryImage)
   Dim dc1 As IntPtr = mygraphics.GetHdc
   Dim dc2 As IntPtr = memoryGraphics.GetHdc

   BitBlt(dc2, 0, 0, Me.ClientRectangle.Width, Me.ClientRectangle.Height, dc1, 0, 0, 13369376)

   mygraphics.ReleaseHdc(dc1)
   memoryGraphics.ReleaseHdc(dc2)
End Sub
```

Le dernier paramètre a pour valeur= 13369376= SRCCOPY = &HCC0020 et correspond à 'Copies the source bitmap to destination bitmap'.

Annexe : Conversions de type

Au cours du regroupement, l'une des principales étapes consiste à convertir les types non gérés en types gérés, et inversement. Le service de regroupement du CLR sait comment effectuer nombre de ces conversions, mais vous devez quand même connaitre les correspondances entre les différents types lors de la conversion de la signature non gérée vers la fonction gérée. Vous pouvez utiliser le tableau de conversion suivant pour mettre en correspondance les différents types.

| Type de données Windows | Type de données .NET |
|---|---|
| BOOL, BOOLEAN | Boolean ou Int32 |
| BSTR | String |
| BYTE | Byte |
| CHAR | Char |
| DOUBLE | Double |
| DWORD/LPDWORD | Int32 or UInt32 |
| FLOAT | Single |
| HANDLE (et tous les autres types de pointeurs, tels que HFONT et HMENU) | IntPtr, UintPtr ou HandleRef |
| HRESULT | Int32 or UInt32 |
| INT | Int32 |
| LANGID | Int16 ou UInt16 |
| LCID | Int32 or UInt32 |
| LONG | Int32 |
| LPARAM | IntPtr, UintPtr ou Object |
| LPCSTR | String |
| LPCTSTR | String |
| LPCWSTR | String |
| LPSTR | String ou StringBuilder* |
| LPTSTR | String ou StringBuilder |
| LPWSTR | String ou StringBuilder |
| LPVOID | IntPtr, UintPtr ou Object |
| LRESULT | IntPtr |
| SAFEARRAY | type de tableau .NET |
| SHORT | Int16 |
| TCHAR | Char |
| UCHAR | SByte |
| UINT | Int32 or UInt32 |
| ULONG | Int32 or UInt32 |
| VARIANT | Object |
| VARIANT_BOOL | Boolean |
| WCHAR | Char |
| WORD | Int16 ou UInt16 |
| WPARAM | IntPtr, UintPtr ou Object |

# X-K - Faire du glisser-déplacer (Drag & Drop)

L'exécution d'opérations **glisser-déplacer** (Drag and Drop) peut être ajoutée dans un programme.

La méthode **DoDragDrop** du contrôle de départ autorise la collecte des données au début de l'opération.

Les événements **DragEnter, DragLeave et DragDrop** permettent de 'poser' les données dans le contrôle d'arrivée.

## X-K-1 - Exemple N° 1 (simple)

Exemple : Le contrôle de départ est un contrôle Button, les données à faire glisser sont la chaine représentant la propriété Text du contrôle Button, et les effets autorisés sont la copie ou le déplacement. Le texte sera déposé dans un textBox :

Le contrôle de départ.

La fonctionnalité qui autorise la collecte des données au début de l'opération glisser dans la méthode **DoDragDrop**.

L'événement **MouseDown** du contrôle de départ est généralement utilisé pour démarrer l'opération glisser parce qu'il est le plus intuitif (la plupart des glisser-déplacer commencent par un appui sur le bouton de la souris). Mais, souvenez-vous que n'importe quel événement peut servir à initialiser une procédure glisser-déplacer.

Remarque Les contrôles ListView et TreeView, ont un événement ItemDrag qui est spécifique.

```
Private Sub Button1_MouseDown(ByVal sender As Object, ByVal e As
 System.Windows.Forms.MouseEventArgs)
_Handles Button1.MouseDown
   Button1.DoDragDrop(Button1.Text, DragDropEffects.Copy Or DragDropEffects.Move)
End Sub
```

Le premier argument indique les données à déplacer.

Le second les effets permis = copier ou déplacer.

Le contrôle d'arrivée.

Toute zone d'un Windows Form ou d'un contrôle peut être configurée pour accepter les données déplacées en définissant la propriété **AllowDrop** et en gérant les événements **DragEnter** et **DragDrop**.

Dans notre exemple, c'est un contrôle TextBox1 qui est le contrôle d'arrivée.

```
TextBox1.AllowDrop  =True.      'autorise le contrôle TextBox à recevoir
```

Dans l'événement DragEnter du contrôle qui doit recevoir les données déplacées.

Vérifier que le type des données est compatible avec le contrôle d'arrivée (ici, vérifier que c'est bien du texte).

Définir ensuite l'effet produit lorsque le déplacement a lieu en lui attribuant une valeur de l'énumération **DragDropEffects**. (ici il faut copier)

```
Private Sub TextBox1_DragEnter(ByVal sender As Object, ByVal e As
 System.Windows.Forms.DragEventArgs)
_Handles TextBox1.DragEnter
   If (e.Data.GetDataPresent(DataFormats.Text)) Then
     e.Effect = DragDropEffects.Copy
   Else
     e.Effect = DragDropEffects.None
   End If
End Sub
```

Dans l'événement DragDrop du contrôle d'arrivée, utilisez la méthode **GetData** pour extraire les données que vous faites glisser.

```
Private Sub TextBox1_DragDrop(ByVal sender As Object, ByVal e As
 System.Windows.Forms.DragEventArgs)
_Handles TextBox1.DragDrop
   TextBox1.Text = e.Data.GetData(DataFormats.Text).ToString
End Sub
```

## X-K-2 - Exemple N° 2 (plus complexe)

Glisser déplacer une ligne d'une listBox 'ListBox1' vers une listBox 'ListBox2'.

Créer une ListBox1.

Créer une listBox2 avec sa propriété AllowDrop=True 'listBox2 accepte le 'lâcher'.

Dans l'entête du module, ajouter :

```
Public IndexdInsertion As Integer    ' Variable contenant l'index où sera insérée la ligne

'Éventuellement, pour l'exemple charger les 2 ListBox avec des chiffres pour pouvoir tester.

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase

Dim i As Integer

For i = 1 To 100

ListBox1.Items.Add(i.ToString)

Next

For i = 1 To 100

ListBox2.Items.Add(i.ToString)

Next

End Sub

'Dans le listBox de départ, l'événement MouseDown  déclenche le glisser-déplacer par DoDragDrop.

Private Sub ListBox1_MouseDown(ByVal sender As Object, ByVal e As
 System.Windows.Forms.MouseEventArgs)
_Handles ListBox1.MouseDown

ListBox1.DoDragDrop(ListBox1.Items(ListBox1.IndexFromPoint(e.X, e.Y)), _
 DragDropEffects.Copy Or DragDropEffects.Move)

End Sub
```

**ListBox1.IndexFromPoint(e.X, e.Y)** retourne l'Index de l'item ou se trouve la souris

à partir des coordonnées e.x et e.y du pointeur) .

DoDragDrop a 2 arguments : l'élément à draguer et le mode.

DragOver qui survient quand la souris se balade sur le contrôle d'arrivée, vérifie si le Drop reçoit bien du texte et met dans IndexdInsertion le listItem qui est sous la souris.

Noter que e.x et e.y sont les coordonnées écran, il faut les transformer en coordonnées client (du contrôle) par PointToClient afin d'obtenir l'index de l'item ou se trouve la souris (en utilisant IndexFromPoint.

```
Private Sub ListBox2_DragOver(ByVal sender As Object, ByVal e As
 System.Windows.Forms.DragEventArgs)
_Handles ListBox2.DragOver

If Not (e.Data.GetDataPresent(GetType(System.String))) Then

e.Effect = DragDropEffects.None

Else

IndexdInsertion = ListBox2.IndexFromPoint(ListBox2.PointToClient(New Point(e.X, e.Y)))

e.Effect = DragDropEffects.Copy

End If

End Sub



'Enfin dans DragDrop, on récupère le texte dans Item et on ajoute un item après l'item pointé.

Private Sub ListBox2_DragDrop(ByVal sender As Object, ByVal e As
 System.Windows.Forms.DragEventArgs)
_Handles ListBox2.DragDrop

Dim item As Object = CType(e.Data.GetData(GetType(System.String)), System.Object)

ListBox2.Items.Insert(IndexdInsertion + 1, item)

End Sub
```

## X-L - Utiliser le 'Registre'

Le Registre est un fichier de stockage des informations relatives aux applications, aux utilisateurs et aux paramètres système. Les applications peuvent utiliser le Registre pour stocker les informations à conserver après leur fermeture et accéder à ces mêmes informations après leur rechargement. On peut stocker les paramètres de taille, couleur, les positions d'affichage ou la taille de la fenêtre. Vous pouvez contrôler ces paramètres pour chaque utilisateur en stockant les informations qui lui correspondent à un emplacement différent du Registre.

Le dictionnaire de l'informatique Microsoft (Microsoft Computer Dictionary, cinquième édition) définit le Registre :

Base de données hiérarchique centrale, utilisée dans Microsoft Windows 9x, Windows CE, Windows NT et Windows 2000, permettant de stocker les informations nécessaires pour configurer le système pour un ou plusieurs utilisateurs, programmes et périphériques matériels, les profils des utilisateurs, les applications installées sur l'ordinateur et les types de documents qu'elles peuvent créer, les paramètres de la feuille de propriétés pour les dossiers et les icônes des applications, le matériel du système et les ports utilisés.

Le Registre remplace la plupart des fichiers texte .ini utilisés dans les fichiers de configuration Windows 3.x et MS-DOS, tels que Autoexec.bat et Config.sys. Bien que le Registre soit commun à Windows 9x Xp Vista, il y a certaines différences.

voir Informations Microsoft sur le registre.

On peut modifier directement le registre 'à la main' en lançant le logiciel Regedit, mais un programme VB peut aller écrire ou lire dans le registre.

Le registre est composé de Clé, sous-clé et valeur. (une valeur a 'un nom de valeur' et une valeur) ; cela forme une arborescence.

Une clé donnée peut avoir des sous-clés. Chaque clé peut également avoir plusieurs valeurs qui servent à stocker les informations relatives à l'application qui vous intéresse. Chaque valeur contient des informations spécifiques qui peuvent être extraites ou mises à jour (noms de valeur et valeurs).

Vous remarquerez que les informations stockées dans le Registre sont accessibles aux autres applications et utilisateurs, aussi il est fortement déconseillé d'y mettre des codes d'accès.

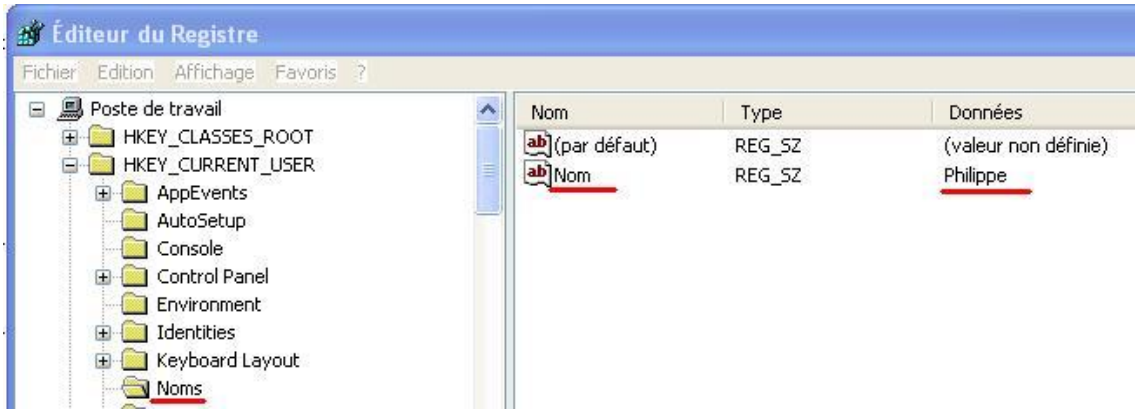
Microsoft nous donne les Clés de base, (elles sont en lecture seule). Les clés de base sont le premier niveau de l'arborescence des clés :

CurrentUser
Stocke les informations relatives aux préférences de l'utilisateur.
LocalMachine
Stocke les informations de configuration pour l'ordinateur local.
ClassesRoot
Stocke les informations relatives aux types (et classes) et à leurs propriétés.
Users
Stocke les informations relatives à la configuration utilisateur par défaut. PerformanceData
Stocke les informations relatives aux performances pour les composants logiciels.
CurrentConfig
Stocke les informations concernant le matériel qui ne sont pas spécifiques à l'utilisateur.
DynData

Stocke les données dynamiques

Pour voir les clés de base et l'arborescence des clés exécutons 'Regedit' :

Menu 'Démarrer'-> menu 'Exécuter' taper 'Regedit'



On voit ici les 2 premières clés de base HKEY_CLASSES_ROOT et HKEY_CURRENT_USER, on a développé les sous-clés de HKEY_CURRENT_USER qui sont 'AppEvents', 'AutoSetup'…, "Noms". La sous-clé 'Noms' à un nom de valeur nommé 'Nom' qui contient la valeur 'Philippe'.

Pour des raisons de sécurité, il est préférable d'écrire des données dans le dossier utilisateur (Microsoft.Win32.Registry.CurrentUser) plutôt que sur l'ordinateur local (Microsoft.Win32.Registry.LocalMachine).



*Attention, c'est très dangereux de modifier des clés de registre : une erreur sur une clé importante et une application ou Windows plante !!*

L'auteur de ce cours décline toute responsabilité…. …. …. ….bla bla bla

Comment ça marche ?

L'objet RegistryKey représente un nœud de niveau clé dans le Registre

On instancie un nœud :

```
Dim key As Microsoft.Win32.RegistryKey
```

La classe Registry fournit les Clés de base (CurrentUser, LocalMachine, ClassesRoot, Users, CurrentConfig…).

On met par exemple dans key le nœud CurrentUser :

```
key = Microsoft.Win32.Registry.CurrentUser
```

**Ensuite on peut :**

- créer une sous-clé avec .CreateSubKey ;
- ouvrir une sous-clé avec .OpenSubKey ;
- écrire ou lire une valeur avec .GetValue ou .SetValue ;
- effacer une valeur avec .DeleteValue.

Création d'une Key, écriture de valeur.

Cet exemple ajoute le nom de valeur "Nom" et la valeur "Philippe" au Registre de l'utilisateur en cours.

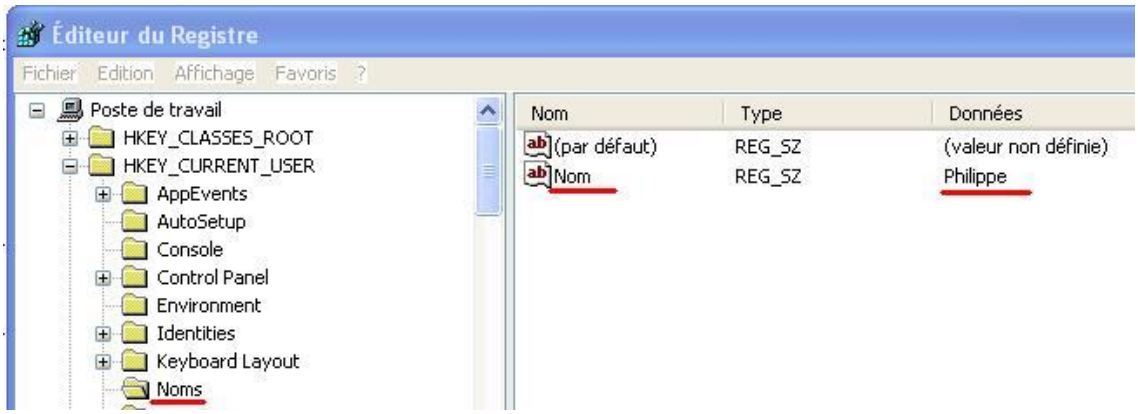
Cela sous la clé "Noms" dans HKEY_CURRENT_USER du Registre.

Exemple

```
Dim key As Microsoft.Win32.RegistryKey
key = Microsoft.Win32.Registry.CurrentUser.CreateSubKey("Noms")
key.SetValue("Nom", "Philippe")
```

Pour voir dans le registre le résultat :

Menu 'Démarrer'-> menu 'Exécuter' taper 'Regedit'




on obtient dans le registre.

Lecture de valeur.

```
Dim key As Microsoft.Win32.RegistryKey
key = Microsoft.Win32.Registry.CurrentUser.OpenSubKey("Noms")
Dim name As String = CType(key.GetValue("Nom"), String)
```

La méthode GetValue retourne un Object. Si votre code comporte Option Strict On, vous devez effectuer un cast de la valeur de retour vers le type de données que vous avez placé précédemment dans la valeur.

GetValue retourne Nothing si la valeur n'existe pas.

Supprimer la Key

```
Dim key As Microsoft.Win32.RegistryKey
key = Microsoft.Win32.Registry.CurrentUser.OpenSubKey("Noms", True)
key.DeleteValue("Nom")
```

Lorsque vous changez le paramètre True par False, la clé est accessible en lecture seule et vous ne pouvez pas la supprimer à l'aide de la variable key.

Bien sûr on peut tout faire: ajouter, supprimer une Clé, une valeur, voir la liste des sous-clés, la liste des valeurs…

En VB 2005 la Classe My.Computer.Registry permet un accès à la base de registre :

```
Dim key As RegistryKey = My.Computer.Registry.ClasseRoot pointe sur la classe de base ClassRoot.
```

Pour un nœud on a les propriétés : Name, SubKeyCount ; si ce dernier est supérieur à 0 GetSubKeyNames retourne un tableau de chaines contenant le nom des sous-clés. On peut ensuite ouvrir le sous-nœud grâce à OpenSubKey.

Si ValueCount est supérieur à 0, le nœud contient des valeurs GetValueNames retourne un tableau de chaine contenant le nom des valeurs. On peut ensuite lire la valeur avec GetValue.

N'oubliez pas de fermer tous les objets RegistryKey (ouverts ou crées) par Close.

Exemples pratiques

Exemple 1

Word est-il installé ? il faut voir si le registre contient HKEY_CLASSES_ROOT\Word.Application

```
Dim key As RegistryKey = My.Computer.Registry.ClasseRoot.OpenSubKey ("Word.Application")
```

```
If key Is Nothing: Word n'est pas installé.
```

Exemple 2

Lire rapidement une valeur? (sans instanciation)

```
Dim valeur As String = My.Computer.Registry.GetValue ("HKEY_CLASSES_ROOT\Noms","nom","")
```

Trois paramètres : nom de la clé, nom de la valeur (mettre NOTHING pour la valeur par défaut), valeur à retourner si pas de valeur.

Exemple 3:

Modifier rapidement une valeur?

Il existe SetValue.

Avant de l'utiliser, il faut ouvrir un registryKey en écriture :

```
Dim rc As RegistryKey = My.Computer.ClasseRoot.OpenSubKey ("Noms",True) 'True permet l'écriture

rc.SetValue("Nom valeur", valeur)
```

Il existe enfin DeleteValue, DeleteSubKey et DeleteSubTreeKey…

Exemple 4

Ajouter une ligne dans le menu contextuel de Windows: clique droit sur un raccourci ou sur un fichier dans l'explorer. Cela ouvre un menu contextuel avec une ligne 'MonProgram'. Si on clique dessus, on lance l'exécutable 'MonProg.exe' et on charge le fichier pointé.

```
Dim regKey As RegistryKey

regKey = My.Computer.Registry.ClassesRoot.CreateSubKey("*\Shell\MonProgram\command")

My.Computer.Registry.SetValue("HKEY_CURRENT_USER\*\Shell\MonProgram\command", "",
    _My.Application.Info.DirectoryPath & "\" & "MonProg.exe -o" & Chr(34) & "%L" & Chr(34))
```

Si MonProg.exe est en VB, il faut récupérer au démarrage le nom du fichier (voir 4-1).

## X-M - Utiliser le 'Presse-papier'

On a tous coupé un bout de texte pour le coller à un autre endroit.

Pour cela on a utilisé le presse-papier ou '**Clipboard**' de Windows.

Comment faire cela par code ?

Voyons cela en VB 2003 et VB 2005.

# X-M-1 - En VB 2003 (Framework 1)

## X-M-1-a - Mettre dans le presse-papier

```
Clipboard.SetDataObject(Texte)
```

il y a une surcharge :

```
Clipboard.SetDataObject(Texte, True)
```

Le second paramètre spécifie si les données doivent rester dans le presse-papier lorsque l'utilisateur quitte l'application.

## X-M-1-b - Récupérer le texte du presse-papier

```
Dim iData As IDataObject

iData = Clipboard.GetDataObject()
```

Pour la récupération, il faut s'occuper du format des données: si on récupère un BipMap et qu'on le colle dans un textbox, il y a problème!! On récupère donc un objet qui contient les données, mais aussi des indications sur le format des données ;on peut tester ce format avant de 'coller'.

On récupère un objet de type **IDataObject** qui contient.

La méthode **GetFormats** permettant de connaitre tous les formats contenus dans IDataObject

```
        Dim myFormatsArray As String() = iData.GetFormats(False)
```

La méthode **GetDataPresents** permettant de savoir si un format est présent :

```
       If iData.GetDataPresent(DataFormats.Text) Then
```

La méthode **GetData** permettant de récupérer les données (paramètre : le format).

```
        textBox2.Text = CType(iData.GetData(DataFormats.Text), String)
```

Les différents formats :

```
DataFormats.Text

DataFormats.Rtf

DataFormats.Html

DataFormats.CommaSeparatedValue

DataFormats.Dif
```

```
DataFormats.BitMap

DataFormats.Dib

DataFormats.WaveAudio

….
```

## X-M-1-c - Exemple

Mettre le texte sélectionné de TextBox1 dans le presse-papier.

Un Button1 portant le texte "Copier" contient le code :

```
If textBox1.SelectedText <> "" Then
    Clipboard.SetDataObject(textBox1.SelectedText)
End if
```

Remarque : si je voulais un button1 "Couper" il aurait fallu ajouter textBox1.SelectedText=""

Récupérer le texte du presse-papier et le mettre dans TextBox2.

Un Button2 nommé "Coller" contient le code :

```
Dim iData As IDataObject = Clipboard.GetDataObject()

' Détermine si c'est du texte.
If iData.GetDataPresent(DataFormats.Text) Then
' récupère le texte par GetData puis le cast en String
textBox2.Text = CType(iData.GetData(DataFormats.Text), String)
Else
' C'est pas du texte.
textBox2.Text = "Pas possible de récupérer."
End If
```

## X-M-1-d - Alternative

Au lieu d'utiliser l'objet Clipboard, on peut utiliser une méthode plus simple.

Le formulaire étant actif et la textbox ayant le Focus envoyez CTRL C avec SendKeys.Send pour couper, CTRL V pour coller : c'est Windows qui s'occupe de tout !!

## X-M-2 - My.Computer.Clipboard à partir de VB 2005

My.Computer.Clipboard : permet de récupérer des informations sur le contenu du presse-papier, de récupérer et de définir son contenu avec une certaine simplification.

On peut mettre du texte dans le presse-papier :

```
My.Computer.Clipboard.SetText(TextBox3.Text)
```

On peut même indiquer le format du texte (rtf, html, text, unicode…) :

```
My.Computer.Clipboard.SetText(TextBox3.Text, TextDataFormat.Rtf)
```

Il existe aussi SetImage, SetAudio…

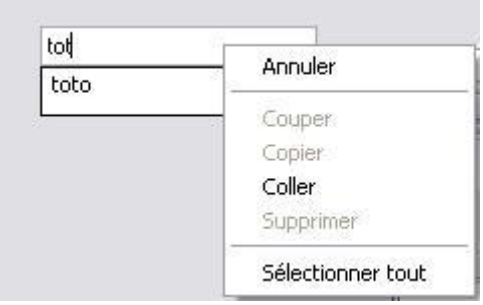
On peut vérifier si le Presse-papier contient une image (ContainsImage), du texte( ContainsText), du son (ContainsAudio)… et récupérer cette image (GetImage ) ou ce texte (GetText).

```
If My.Computer.Clipboard.ContainsImage Then
    PictureBox1.Image = My.Computer.Clipboard.GetImage
ElseIf My.Computer.Clipboard.ContainsText Then
    TextBox1.Text = My.Computer.Clipboard.GetText
End If
                    <paragraph>

                    </paragraph>
```

## X-M-3 - TextBox et couper-coller

À partir de VB 2005 (le Framework 2) pour copier coller dans un **TextBox**, c'est encore plus facile.

La propriété **ShortCutsEnabled = True** permet à l'utilisateur d'ouvrir un menu contextuel avec le clic droit; ce menu permet les annuler, couper, copier, coller, supprimer, Sélectionner tout; ce qui peut aussi être fait avec les raccourcis clavier Shift/Inser Ctrl/Inser…




## X-N - Paramètres de configuration (App.ini, registre, App.config)

En plus des données, une application a besoin d'utiliser des paramètres.

Ceux de l'application :

- nom des bases de données à ouvrir ;

- chemin des bases de données ;

- chaines de connexion ;

- ceux de chaque utilisateur ;

- état, position et dimension des formulaires ;

- couleurs ;

- paramètres d'imprimante: dimension du papier, police de caractères ;

-préférences de l'utilisateur.

…

Toutes ces informations doivent bien être stockées quelque part !!!

On peut les stocker dans :

- de simples fichiers ;

- des fichiers .INI ;

- le registre ;

- des fichiers de configuration .Config (VB 2003) ;

- des fichiers de configuration .Config (VB 2005).

En VB2005 on peut même lier une propriété à un paramètre de configuration.

## X-N-1 - Les Fichiers

Il y a longtemps on utilisait un **fichier** séquentiel.

On ouvrait le fichier avec FileOpen(), on écrivait les paramètres avec Writeline.

Lors d'une utilisation ultérieure du logiciel on lisait les paramètres avec LineInput.

On utilisait aussi parfois **une table dans une base de données** pour stocker les paramètres.

Avec Vb.Net certains stockent les paramètres dans un **fichier XML** (directement ou à l'aide de la serialization).

Mais VB offre d'autres méthodes dédiées à cet usage.

## X-N-2 - Fichiers .INI

Il y a quelque temps (jusqu'à Windows 95 et même après), on utilisait **les fichiers .INI.**

Chaque application avait son fichier INI, Windows avait aussi ses propres fichiers INI: System.Ini Boot.Ini…

Exemple d'un fichier MonApp.INI , il est situé dans le répertoire de l'application ou dans le répertoire Windows.

Si on le regarde avec NotePad :

```
[Configuration matérielle]
RépertoireDonnées=C:\M\Donnees
RépertoireSauvegardes=C:\M\Sauve
RépertoireLocal=C:\M\Local
[Configuration Logiciel]
SoftAgenda=Calendar.Exe
TypeAgenda=0
PathAgenda=
```

Dans notre exemple, le fichier donne les répertoires de données et le nom de l'agenda à utiliser.

Il comporte **de grandes rubriques**, [Configuration Matérielle] entourées de [].

Dans chaque rubrique, il y a des **clés**, RépertoireDonnées par exemple, suivi d'un = puis de la valeur correspondant à la clé C:\M\Donnees ici.

Comment lire ou écrire dans un fichier Ini 'Privé' (propre à l'application).

Il faut pour cela utiliser des fonctions fournies par les API Windows (elles se trouvent dans Kernel32.dll).

**GetPrivateProfileString** permet de lire une string correspondant à une clé.

**GetPrivateProfileInt** permet de lire un entier correspondant à une clé.

**PutPrivateProfileString** permet d'écrire une string correspondant à une clé.

**PutPrivateProfileInt** permet d'écrire un entier correspondant à une clé.

### A- Avant, il faut les déclarer.

```
'Fonction lisant un Integer

Declare Function GetPrivateProfileInt Lib "Kernel32" Alias "GetPrivateProfileIntA" _
(ByVal lpApplicationName As String, ByVal lpKeyName As String, _
ByVal nDefault As Long, ByVal lpFileName As String) As Long



'Fonction lisant une string

Declare Function GetPrivateProfileString Lib "Kernel32" Alias "GetPrivateProfileStringA" _
(ByVal lpApplicationName As String,  _
ByVal lpKeyName As String, ByVal lpDefault As String, ByVal lpReturnedString As String, ByVal
 nSize As Short, _
ByVal lpFileName As String) As Integer



'Fonction écrivant une string

Declare Function WritePrivateProfileString Lib "Kernel32" Alias "WritePrivateProfileStringA" _
(ByVal lpApplicationName As String, _
ByVal lpKeyName As String, ByVal lpString As String, ByVal lplFileName As String) As Long



'Il existe aussi une fonction écrivant un Entier
```

### B- Pour utiliser plus simplement ces appels à la dll, écrivons des fonctions qui gèrent les appels.

```
Function Get_Private_Profile_Int(ByVal cAppName As String, ByVal cKeyName As String, _
ByVal nKeyDefault As Integer, ByVal cProfName As String) As Long

'

' LIRE UN ENTIER

' Parametres:

' cAppName Correspond à [Rubrique]

' cKeyName Nom de l'entrée, de la clé

' nKeyDefault Valeur par défaut de la chaine cherchée

' cProfName Nom du Fichier "INI" Privé

' Sortie:

' La fonction retourne une valeur numérique entière

 Get_Private_Profile_Int = GetPrivateProfileInt(cAppName, cKeyName, nKeyDefault, cProfName)
```

```
End Function



Function Get_Private_Profile_String(ByVal cAppName As String, _
ByVal cKeyName As String, ByVal cKeyDefault As String, ByRef cKeyValue As String,  _
ByVal cProfName As String) As Integer

'

' LIRE UNE STRING

' Parametres:

' cAppName Correspond à [Rubrique]

' cKeyName Nom de l'entrée, de la clé

' cKeyDefault Valeur par défaut de la chaine cherchée

' cKeyValue Valeur lue en face de l'Entrée ou cKeyDefault si l'Entrée est vide

' cProfName Nom du Fichier "INI" Privé

'

' Sortie:

' Valeur lue dans cKeyValue

' La fonction retourne le nombre de caractères dans cKeyValue

Dim iReaded As Integer

Const sLongueur As Short = 255

If cKeyName = "" Then

cKeyValue = Space$(1025)

iReaded = GetPrivateProfileString(cAppName, "", "", cKeyValue, 1024, cProfName)

Else

cKeyValue = Space$(255)

iReaded = GetPrivateProfileString(cAppName, cKeyName, cKeyDefault, cKeyValue, sLongueur,
 cProfName)

End If

cKeyValue = Trim$(cKeyValue)

'Enlever le dernier caractère?

'If Len(cKeyValue) <> 0 Then

' cKeyValue = Mid$(cKeyValue, 1, Len(cKeyValue) - 1)

'End If

Get_Private_Profile_String = iReaded

End Function



Function Put_Private_Profile_String(ByVal cAppName As String, ByVal cKeyName As String,  _
ByVal cKeyValue As String, ByVal cProfName As String) As Boolean
```

```
' ÉCRIRE UNE STRING
' Parametres:
' cAppName Correspond à [Rubrique]
' cKeyName Nom de l'entrée de la clé
' cKeyValue Valeur lue en face de l'Entrée ou cKeyDefault si l'Entrée est vide
' cProfName Nom du Fichier "INI" Privé
' Sortie:
' La fonction retourne True si cela a marché
Dim Status As Long
Status = WritePrivateProfileString(cAppName, cKeyName, cKeyValue, cProfName)
If (Status <> 0) Then
Put_Private_Profile_String = True
Else
Put_Private_Profile_String = False
End If
End Function
End Class
```

## C- Exemple de lecture et d'écriture :

```
- Lecture du répertoire de données dans MonApp.Ini:


Dim cRubrique As String = "Configuration matérielle"    'Nom de la rubrique
Dim cKey As String = "RépertoireDonnées"                'Nom de la clé
Dim cRepertoire As String = Space(255)                  'Variable récupérant la string
Dim cIniFile As String = "c:\MonApp\MonApp.ini"         'Nom du fichier Ini
Dim istat As Integer


'Appel de la fonction
istat = Get_Private_Profile_String(cRubrique, cKey, "", cRepertoire, cIniFile)
'Affichage du répertoire de données dans une textbox par exemple
TextBox1.Text = Trim(cRepertoire)


' Dans notre exemple, cela affiche C:\M\Donnees
```

- Écriture : dans MonApp.Ini, dans la rubrique "Configuration matérielle", derrière la clé "cRepOld", écrire "No" :

```
Dim cRubrique As String = "Configuration matérielle"    'Nom de la rubrique
```

```
Dim cIniFile As String = "c:\MonApp\MonApp.ini"          'Nom du fichier Ini

Dim bOk As Boolean


bOk = Put_Private_Profile_String(cRubrique, "cRepOld", "No", cIniFile)


'on note que la clé "cRepOld" qui n'existait pas a été créée.
```

Il a bien été ajouté dans le fichier .Ini : cRepOld=No.

## X-N-3 - Registre

À partir de Windows95 et WIndows NT.

**Attention :** l'application doit posséder des droits d'accès au registre suffisants.

Écriture dans le registre

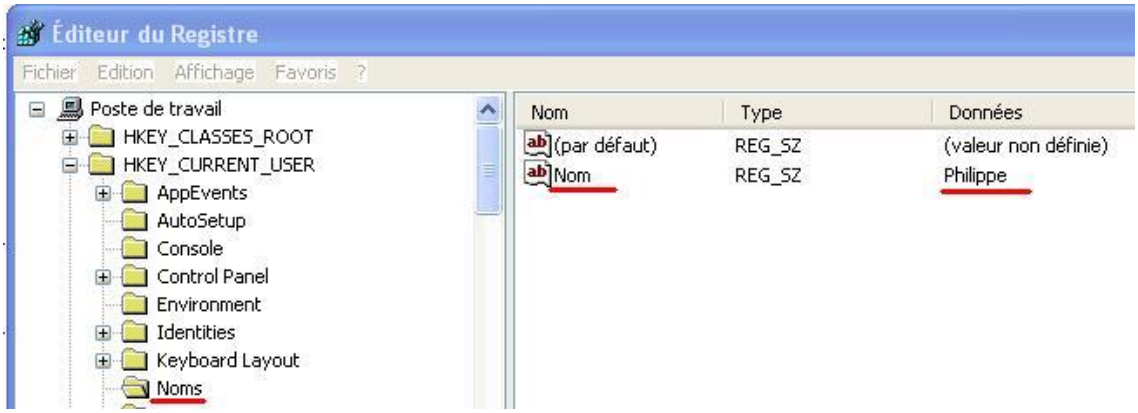
Exemple : ajouter la clé "Nom" et la valeur "Philippe" au Registre de l'utilisateur en cours.

Cela sous la clé "Noms" dans HKEY_CURRENT_USER du Registre.

```
Dim key As Microsoft.Win32.RegistryKey
key = Microsoft.Win32.Registry.CurrentUser.CreateSubKey("Noms")
key.SetValue("Nom", "Philippe")
```

Pour voir dans le registre le résultat :

Menu 'Démarrer'-> menu 'Exécuter' taper 'Regedit'




**Lecture de valeur**

```
Dim key As Microsoft.Win32.RegistryKey
key = Microsoft.Win32.Registry.CurrentUser.OpenSubKey("Noms")
Dim name As String = CType(key.GetValue("Nom"), String)
```

La méthode **GetValue** retourne un Object. Si votre code comporte Option Strict On, vous devez effectuer un cast de la valeur de retour vers le type de données que vous avez placé précédemment dans la valeur.

GetValue retourne Nothing si la valeur n'existe pas.

Voir le chapitre sur le registre pour les détails.

## X-N-4 - Fichier de configuration App.Config File de VB2003 (Framework 1)

VB enregistre les paramètres dans un fichier au format XML.

**Créer le fichier de configuration**

Projet->Ajouter un nouvel élément-> cliquer sur 'Fichier de configuration' : 'App.config' le fichier de configuration en XML est ajouté au projet;(cela ne parait pas possible en VB 2005 Express).

On le voit dans une fenêtre :

```
<?xml version="1.0" encoding="utf-8" ?>

<configuration>

</configuration>
```

Il faut ajouter la partie 'AppSettings' qui contient la configuration, cela donne :

```
<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<appSettings>

<add key="TaillePage" value="5" />

</appSettings>

</configuration>
```

On remarque que l'on a ajouté les balises <appSettings> et </appSettings>.

Entre ces balises, on peut ajouter des lignes contenant une clé et sa valeur :

```
<add key="TaillePage" value="5" />
```

On peut modifier les couples 'clé-valeur' dans l'ide ou bien en ouvrant le fichier App.config avec NotePad.

**Pour lire un élément de la configuration :**

```
Dim TaillePage As String = Configuration.ConfigurationSettings.AppSettings("TaillePage")
```

(on récupère une string :"5" dans l'exemple).

Pour lire tous les éléments :

```
Dim mAppSet As Specialized.NameValueCollection

mAppSet = Configuration.ConfigurationSettings.AppSettings



'Affiche dans MaList les clés:

If Not mAppSet Is Nothing Then

Me.maliste.Items.Clear()
```

```
Dim keys() As String

keys = mAppSet.AllKeys

Dim key As String

For Each key In keys

Me.maliste.Items.Add(key & ": " & mAppSet.Item(key))

Next

End If
```

On utilise une collection mAppSet de type 'NameValueCollection', on y met AppSettings.

On peut voir le nom des clés grâce à mAppSet.GetKey et les valeurs grâce à mAppSet.Item().

Le problème est que pour modifier une valeur par code, il faut charger le fichier XML, modifier puis enregistrer le fichier XML. C'est complexe!! Il existe des classes qui font cela automatiquement.

## X-N-5 - Configuration par paramètres Settings de VB2005 (Framework 2)

Dans le Framework 2 Configuration.ConfigurationSettings n'existe plus !!

il faut utiliser **Configuration.ConfigurationManager,** mais il y a beaucoup plus simple avec la classe My.

Les valeurs des paramètres sont enregistrées automatiquement **si dans les propriétés du projet, dans l'infrastructure de l'application, "Enregistrer My.setting lors de l'arrêt" est coché**.

Les paramètres de configuration concernent l'application ou l'utilisateur.

Les paramètres d'application sont en lecture seule. Dans la mesure où ces paramètres sont des informations du programme, vous n'avez en principe pas besoin de les modifier. Ce sont par exemple le chemin d'une base de données ou une URL.

En revanche, les paramètres d'utilisateur peuvent être lus et modifiés. Ce sont les positions de formulaires, couleur…

On peut créer les paramètres dans le 'Projet designer'.

En vb 2005, pas besoin d'ajouter le fichier comme en vb 2003, le fichier de configuration est crée et ajouté au projet dès que l'on crée un paramètre.

Les paramètres de configuration sont donc directement accessibles dans les propriétés du projet (dans l'explorateur de solution double-cliquer sur My Projet ou passer par le menu Projet-> Propriétés de…

Exemple

Onglet 'Paramètres' , créons un paramètre nommé 'Para1' et contenant '1' (c'est une string)


Un paramètre à un nom, un type, une portée (Application ou Utilisateur), une valeur.

Pour créer, se mettre dans la colonne 'Nom' dans la ligne '*' et taper le nom du paramètre, choisir le type en déroulant la liste avec le bouton de la colonne type, choisir la portée puis éventuellement une valeur.

Donc pas besoin de passer par NotePad et de manipuler du XML.

Les fichiers de configuration sont automatiquement créés lors du démarrage de l'application (avec les valeurs par défaut indiquées dans l'onglet 'paramètres').

**Pour utiliser un paramètre dans le programme,** on fera :

```
MonParamètre= My.Parametre.Para1
```

Les paramètres sont en Read-Only si la portée est 'Application', et en Read-Write (donc modifiable par code) si la portée est 'Utilisateur'.

Ces 'variables paramètres utilisateur' ont des valeurs qui seront conservées et enregistrées automatiquement dans l' 'environnement' d'un utilisateur (en WindowsForms). Si on en modifie la valeur, on retrouve la valeur modifiée lors d'une utilisation ultérieure ce qui permet de conserver les habitudes des utilisateurs.

Modifions un paramètre utilisateur :

```
My.Parametre.Para1= "666"
```

La nouvelle valeur sera automatiquement enregistrée dans le fichier de config.

Bien sur le paramètre 'NomConnexuinS' qui est un paramètre application n'est pas modifiable.

En VB2005 vous pouvez créer des paramètres de type Color, Font, Point, Size :

```
My.Settings.MyFont= New Font (Me.Font, FontStyle.Italics)
```

**Profil utilisateur**

On peut créer des 'Profils' avec plusieurs noms de profil qui auront chacun des paramètres ayant leur propre valeur pour chaque utilisateur Windows.

On peut par exemple créer un profil "Philippe" et, dans ce profil, donner à Para1 la valeur '333'. Créer un second profil "Odile" et, dans ce profil, donner à Para1 la valeur '222'. Dans ce cas si l'utilisateur Philippe ouvre une session

Windows sous le nom l'utilisateur 'Philippe' et qu'il lance l'application, c'est la valeur '333' qui se trouvera dans My.Parametre.Para1

**Événement survenant lors du changement de valeur des paramètres.**

En haut à droite, il y a un bouton ou un menu déroulant qui donne accès à 'Afficher le code'.

Cela donne accès à une Classe partielle Setting qui donne accès aux routines PropertySettings, ChangingSettings, SavingSettings; ainsi quand un paramètre change par exemple, on peut mettre ici le code nécessaire à la mise à jour.

**Dans quels fichiers et ou est enregistré le 'setting' ?**

Dans des fichiers XML.

En mode design.

Les paramètres d'application et utilisateurs (ceux de départ qui ont été écrits à la main dans les propriétés) sont stockés dans la section MonApplication.My.Settings de :

**app.Config** qui est dans le source, dans "c:\Documents and setting\NomUtilisateur\Mes Document\Visual Studio 2005\MonApplication\MonApplication" ;

**MonApplication.exe**.config dans les répertoires bin et release quand on génère l'application. (c'est ce fichier qui sera déployé).

En cours d'exécution, les paramètres d'application sont dans :

**MonApplication.exe.config** dans le répertoire de l'exécutable, (c'est ce fichier qui sera déployé), on rappelle que ces paramètres ne sont pas modifiables.

Les paramètres utilisateurs sont dans

**user.config** dans le répertoire C:\Documents and Settings\NomUtilisateur\Local Settings\Application Data \NomSociete\MonApplication.exe_Url_43f52d0fihtu0kzyyxxngiyacs5ljtnb\1.0.0.0 . On remarque que VB crée un répertoire dans C:\Documents and Settings\NomUtilisateur\Local Settings\Application Data . Vb utilise ensuite le nom de la société qui distribue le logiciel (le nom qui est dans l'assembly) puis le nom de l'application et enfin la version du programme. Comme le framework s'occupe de tout, a priori, on n'a pas à s'occuper des chemins.

**Contenu du fichier WindowsApplication1.exe.config :**

Situé dans "c:\Documents and setting\NomUtilisateur\Mes Document\Visual Studio 2005\WindowsApplication1\WindowsApplication1" (et dans bin\release après génération)

L'application se nomme "WindowsApplication1" , on regarde où est enregistré 'Para1' :

```
<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<configSections>

<userSettings>

<WindowsApplication1.My.MySettings>

<setting name="parmetre1" serializeAs="String">
```

```
<value>12</value>

</setting>

</Bonjour.My.MySettings>

</userSettings>

</configuration>
```

On peut le voir plus rapidement en cliquant dans l'explorateur de solution sur App.Config.

## X-N-6 - Liaison propriétés-Settings de VB2005 (PropertyBinding)

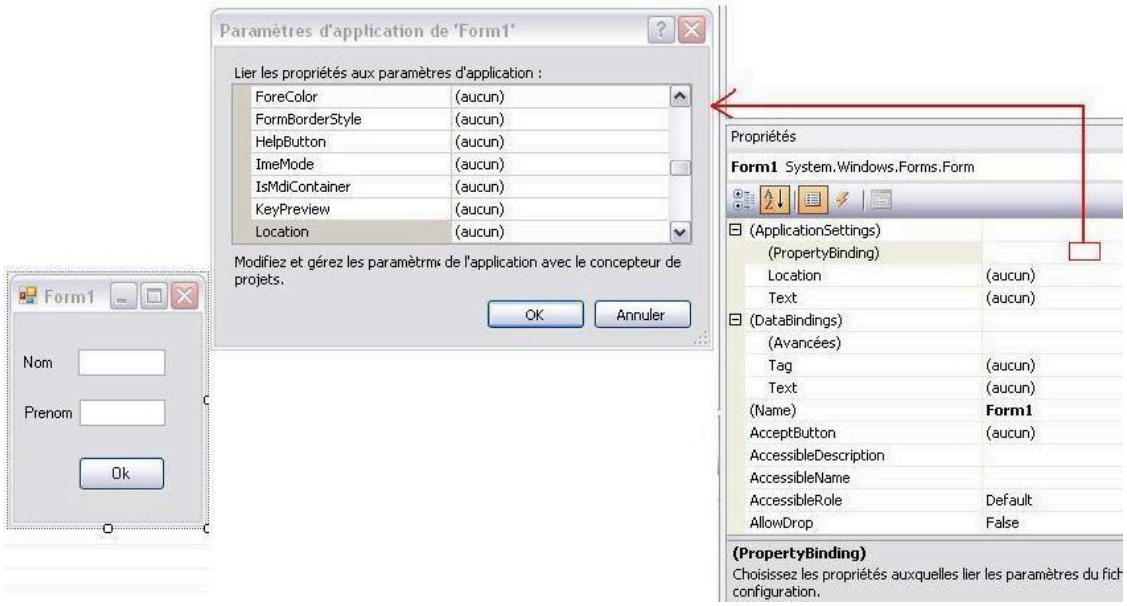
Depuis VB2005, **on peut 'lier' la propriété d'un contrôle avec un paramètre**. C'est-à-dire enregistrer automatiquement les valeurs d'une propriété dans les paramètres de configuration.

Prenons un exemple.

J'ai créé une Form1, je veux que l'utilisateur, lorsqu'il lance l'application, retrouve Form1 à la position où il l'avait laissée.

Il faut donc enregistrer la valeur de la propriété Location de Form1 dans le Setting, la configuration, pour cela on va faire du PropertyBinding : on va lier cette propriété à un paramètre du setting.

Dans la fenêtre de propriété de Form1, aller en haut de la liste, cliquer sur le '+'de (ApplicationSettings), puis au niveau de la ligne (PropertyBinding), cliquer sur le bouton à droite. Cela ouvre la fenêtre 'Paramètres d'application de Form1'




Cliquer sur la ligne "location", puis sur le bouton à droite.

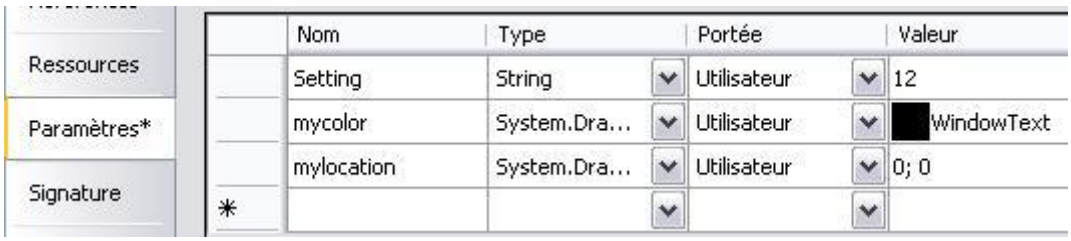

Dans la liste qui s'ouvre, cliquer sur 'Nouveau": La fenêtre 'Nouveau paramètre d'application' s'ouvre :


Taper un nom pour le nouveau paramètre (mylocation par exemple), une valeur par défaut si nécessaire.

Puis OK, OK.

C'est fait : le paramètre 'mylocation' est lié à la propriété Location de Form1.

Regardons dans les paramètres de l'application (menu Projet=>Propriétés de…)




On voit bien le nouveau paramètre mylocation de type System.Drawing.Point. Il est lié à Form1.Location (dans la fenêtre de propriété de form1, sur la ligne location, il y a une toute petite icône qui le signale).

La valeur de Form1.Location sera enregistrée automatiquement et restituée lorsque l'on lancera ultérieurement le programme.

En fait VB génère du code prenant en charge ce BindingProperty: dans l'explorateur de solution cliquer sur MyProjet->Setting.Setting->Setting.Designer.vb on voit le code généré.

## X-O - Utiliser les 'Ressources'

**Ressources :** document informatique de toute nature (texte, image, son, programme).

Une ressource est une donnée non exécutable qui est déployée logiquement avec une application.

Les ressources sont un ensemble d'éléments : images, icônes, textes (chaines), sons, fichiers ou autres qui sont utilisés par le programme. Elles servent habituellement à enrichir l'interface utilisateur. Ce ne sont pas des données. Elles sont contenues dans un fichier .resx et dans un répertoire de ressource.

### X-O-1 - Intérêt des ressources ?

Si j'ai une image à utiliser dans un contrôle, pourquoi ne pas la mettre directement dans la propriété de mon contrôle ?

Vous créez cette image, si une semaine plus tard vous trouvez une image hyper meilleure, avec les ressources, vous n'avez plus qu'à la placer dans le répertoire Ressources à la place de votre précédente image. Vous recompilez et c'est fait.

Si vous avez une même image qui est utilisée à plusieurs endroits de votre application, avec les ressources, vous n'aurez qu'une seule ressource (donc diminution en taille de l'exe) et une facilité de maintenance: plus besoin d'effectuer la modification partout où l'image est utilisée.

Enfin si vous voulez écrire une version de votre exécutable en français et en anglais, avec les ressources, il n'est pas nécessaire d' écrire 2 programmes, il suffit de faire 2 fichiers de ressources.

### X-O-2 - Les types de ressources ?

Les types de ressources sont Chaines, Images, Icônes, Audio, Fichiers et Autre. chaines est l'affichage par défaut.

**chaines**
Affiche des chaines dans une grille de paramètres avec les colonnes Nom, Valeur et Commentaire . Vous pouvez accéder aux paramètres au moment de l'exécution via My.Resources en tant que String. Pour une description des colonnes dans cette grille, consultez la rubrique "Grille des paramètres" ci-après.

**Images**
Affiche tous les fichiers image, y compris les formats .bmp, .jpg .png et .gif. Ces fichiers sont exposés au moment de l'exécution en tant que Bitmap.

**Icônes**
Affiche les fichiers icône (* .ico) qui sont exposés en tant que Icon.

**Audio**
Affiche les fichiers audio, y compris les fichiers .wav, .wma et .mp3. Ces fichiers sont exposés en tant que tableaux d'octets. Le double-clic sur un élément audio permet de l'ouvrir et de le jouer dans Lecteur Windows Media.

**Fichiers**
Affiche tous les fichiers qui n'entrent pas dans les catégories précitées. Les éléments dans cet affichage peuvent être des fichiers texte exposés en tant que String ou des fichiers binaires exposés en tant que tableaux d'octets.

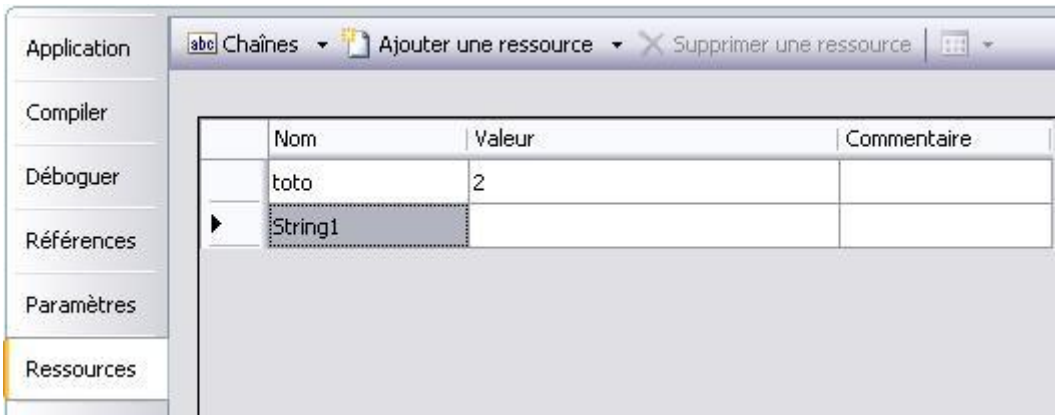
**Autres**

Affiche une grille de paramètres pour ajouter d'autres types qui prennent en charge la sérialisation de chaines (par exemple, Font, Enum, Color et Point). La grille contient les colonnes suivantes : Nom, Type, Valeur et Commentaire.

## X-O-3 - Voir les ressources

Pour voir les ressources, il faut aller dans les propriétés du projet : double-cliquez sur MyProjet dans l'explorateur de solution ou menu 'Projet'=>'Propriétés de…', Onglet 'Ressources').

On voit immédiatement les chaines ; il est possible de voir les autres types de ressources en déroulant la liste à gauche.




Ici on voit une ressource 'chaine' qui se nomme 'toto' (c'est nul!!) et qui contient "2".

## X-O-4 - Ajouter des ressources

**1-Mettre du texte dans une ressource**

Si nécessaire dérouler la liste à gauche et cliquez sur 'chaines'.

Deux manières d'ajouter une chaine :

cliquez sur le bouton 'Ajouter une ressource' puis sur 'Ajouter une nouvelle chaine'. le curseur se retrouve dans la zone de saisie du nouveau nom: tapez le nom puis la valeur ;

ou cliquez directement dans la zone du nom sur la ligne '*', tapez le nom puis la valeur.


L'enregistrement est automatique.

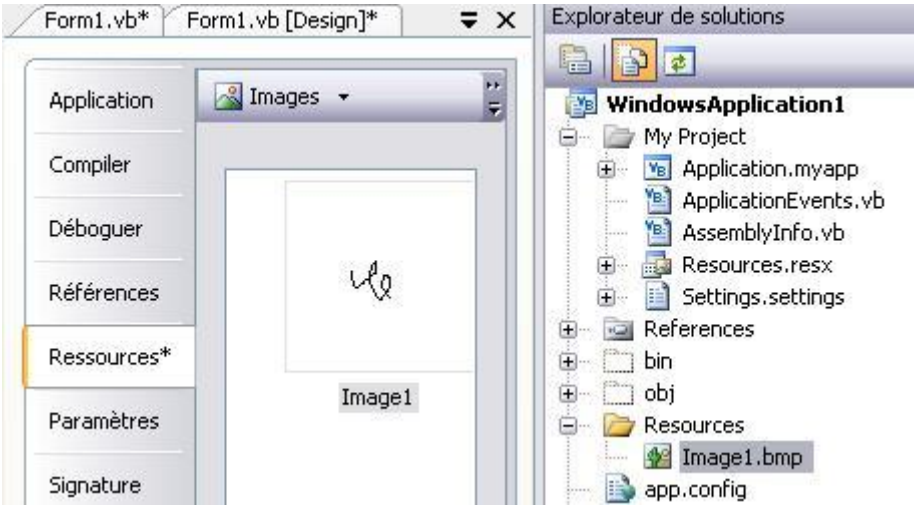
**2-Mettre le dessin dans les ressources**

Aller dans les ressources.

Dérouler la liste à gauche pour y mettre 'Images' puis cliquer sur 'ajouter une ressource' ; on vous demande le nom de la ressource (tapez par exemple 'button_blue'), vous vous trouvez dans Paint, dessinez (ou collez) l'image de votre bouton. Puis menu 'Fichier'=>'Enregistrer' : le dessin apparait dans les ressources. Fermez Paint.

## X-O-5 - Où se trouvent les ressources

Dans l'explorateur de solutions, on voit bien le répertoire de ressources et la ressource qui vient d'être créée (le fichier Image1.bmp par exemple)




On peut d'ailleurs aller dans ce répertoire et modifier la ressource, remplacer le fichier par un autre, la modification est immédiatement mise à jour dans VB.

Vous créez une image, si une semaine plus tard vous trouvez une image hyper plus belle, vous n'avez plus qu'à la placer dans le répertoire Ressources à la place de votre précédente image.

## X-O-6 - Modifier une ressource

Une image par exemple.

Cliquez sur la ressource, puis cliquez avec le bouton droit ; dans le menu.

Ouvrir : ouvre l'éditeur par défaut (Paint pour une image).

Ouvrir avec : permet de modifier la ressource avec un autre programme.

## X-O-7 - Utiliser une ressource dans le programme

Les ressources sont accessibles par leur nom dans la Classe **My.Ressources**.

Mettre le texte d'une ressource dans la barre de titre d'un formulaire :

```
Me.Text = My.Resources.Form1Title
```

Mettre une ressource image dans le plan BackGround d'un formulaire :

```
Me.BackgroundImage = My.Resources.Form1Background
```

Mettre une ressource image dans le plan BackGround d'un bouton :

```
MyButton.BackGroundImage= MonProgramme.My.Ressources.Ressources.button_Blue
```

Mettre une ressource icône comme icône d'un formulaire :

```
Me.Icon = My.Resources.MyIcon
```

Jouer un son qui est dans les ressources :

```
My.Computer.Audio.Play(My.Resources.Form1Greeting, AudioPlayMode.Background)
```

## X-O-8 - Ressources localisées

On a parfois besoin d'avoir des ressources pour différentes langues. (Texte des boutons en français ou en anglais par exemple).

Pour chaque culture faire un fichier **.resx**.

En fait, copier le fichier de ressources et attribuer au fichier de ressources le nouveau nom Resources.CultureSignature.resx.

Il semble que les fichiers de ressources de la langue par défaut soit dans le répertoire ressources avec l'extension .resx et que pour chaque autre culture, un sous-répertoire soit nécessaire (sous-répertoire nommé 'fr', 'en-us'…) Enfin dans ce sous-répertoire, le fichier se nomme monprogramme.Fr-fr.resx À vérifier !!

Quand la culture de l'ordinateur change, le fichier de ressources correspondant est utilisé.

Exemple : ici on modifie la culture puis on utilise la ressource correspondante.

```
    Dim Currentculture As String = My.Application.UICulture.Name
    My.Application.ChangeUICulture("fr-FR")
    MsgBox(My.Resources.MyMessage)
    My.Application.ChangeUICulture(Currentculture)
```

## X-O-9 - Ressources liées ou incorporées

Les projets Visual Studio fournissent deux options différentes pour gérer les ressources : celles-ci peuvent être liées (par défaut) ou incorporées. S'il est possible d'avoir à la fois des ressources liées et incorporées dans un même projet, il est plus pratique dans la plupart des cas de choisir une option pour toutes les ressources du projet.

**Les ressources liées** sont stockées comme des fichiers dans le projet. Pendant la compilation, les données de ressources sont extraites des fichiers et placées dans le manifeste de l'application. Le fichier de ressources (.resx) de l'application stocke uniquement un chemin d'accès relatif ou un lien au fichier sur le disque.

**Avec les ressources incorporées**, les données de ressources sont stockées directement dans le fichier .resx dans une représentation textuelle des données binaires.

Dans l'un et l'autre cas, les données de ressources sont compilées dans le fichier exécutable.

Les ressources peuvent passer de la valeur "liées" à "incorporées" en modifiant la propriété Persistence du fichier de ressources.

Dans la fenêtre Propriétés, sélectionnez la propriété **Persistance** et affectez-lui la valeur incorporée dans .resx.

Dans la fenêtre Propriétés, sélectionnez la propriété Persistance et affectez-lui la valeur Lié au moment de la compilation.

Les ressources incorporées sont le meilleur choix si vous devez partager des fichiers de ressources d'application (.resx) entre plusieurs projets. Par exemple, si vous disposez d'un fichier de ressources communes contenant des informations telles que les logos et les informations relatives aux marques de votre société, l'utilisation de ressources incorporées signifie que vous pouvez vous contenter de copier le fichier .resx et non les fichiers de données de ressources associés.

Vous ne pouvez pas modifier directement les ressources incorporées. Si vous tentez de modifier une ressource incorporée, un message vous invitant à convertir l'élément en ressource liée afin de la modifier s'affichera ; la conversion est recommandée, mais facultative. Vous devez les exporter, effectuer vos modifications dans un programme externe, puis les réimporter dans votre projet.

Les ressources de type chaine sont toujours incorporées et ne peuvent pas être modifiées. Les ressources de fichier sont toujours liées et ne peuvent pas être modifiées.

### X-O-10 - Comment cela marche ?

Quand on ajoute par exemple une image aux ressources, VB crée (si ce n'est déjà fait), un répertoire Ressources dans votre projet. Puis il copie votre fichier image dans ce répertoire. Ensuite, il crée (ou modifie) un fichier .resX avec une référence vers le fichier image correspondant et une classe interne Ressources. Cette dernière possède des propriétés en lecture seule statiques qui retourne les ressources de votre fichier ressource.

À la compilation, ces images seront incluses dans votre exécutable.

Vous créez cette image, si une semaine plus tard vous trouvez une image hyper meilleure, vous n'avez plus qu'à la placer dans le répertoire Ressources à la place de votre précédente image. Vous recompilez et c'est fait.

## X-P - Où mettre les programmes et les données

### X-P-1 - Il faut séparer les données des programmes !!!

Les programmes, les données n'ont pas la même vocation.

**Les fichiers programmes** (les fichiers .exe) sont copiés sur votre disque au moment de l'installation du logiciel ou au moment de sa mise à jour. En dehors de ces périodes, ces fichiers ne changent pas.

Il n'est donc pas utile de sauvegarder fréquemment, car en plus, vous possédez ceux-ci par ailleurs sur un support d'installation (CD-ROM, fichier téléchargé).

**Par contre les fichiers de données** sont régulièrement créés, enregistrés ou modifiés : mails, documents Word ou Excel, photos, fichiers musicaux, bases de données. Pour ne pas perdre toutes ces informations, il faut les sauvegarder : les enregistrer régulièrement sur un autre support (CD-ROM, disque dur externe, clé USB, autre ordinateur d'un réseau…).

Pour favoriser vos sauvegardes, il est donc logique de séparer vos données de vos programmes, et de les mettre dans des répertoires différents.

Habituellement on a un répertoire de programmes et un répertoire de données, il est possible d'avoir un répertoire de données pour chaque utilisateur si les données sont distinctes et parfois un répertoire commun à tous les utilisateurs (AllUsers).

Ainsi on peut simplement sauvegarder régulièrement la totalité du répertoire de données et uniquement ce répertoire.

De même pour éviter les soucis lors de la réinstallation de Windows, il est plus facile de protéger les données si celles-ci se trouvent sur un disque ou une partition séparée.

## X-P-2 - Sécurité

Un virus, un ver ou un spyware est un programme qui va essayer d'utiliser votre accès à Windows (vos droits d'utilisateur) pour s' "autoinstaller" et prendre la main de votre système.

Or par défaut sous Windows XP vous avez tous les droits (car vous avez un compte administrateur) : mettre à jour le système, installer des programmes, ce qui n'est pas vraiment indispensable pour lire vos mails, faire du traitement de texte ou surfer sur Internet.

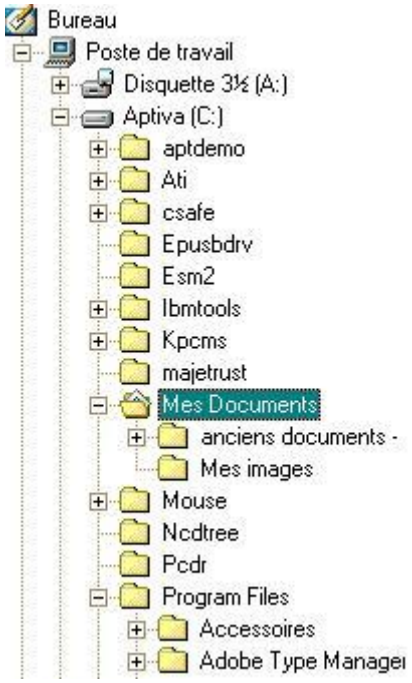
**Vous pouvez donc utiliser votre Windows de la manière suivante :**

- créer des comptes utilisateurs limités pour l'usage quotidien ;
- utiliser un compte administrateur pour toutes les installations et mises à jour.

C'est une contrainte, mais cela garantit le maximum de sécurité.

## X-P-3 - Quels répertoires utiliser ?

- **Windows 98 :**

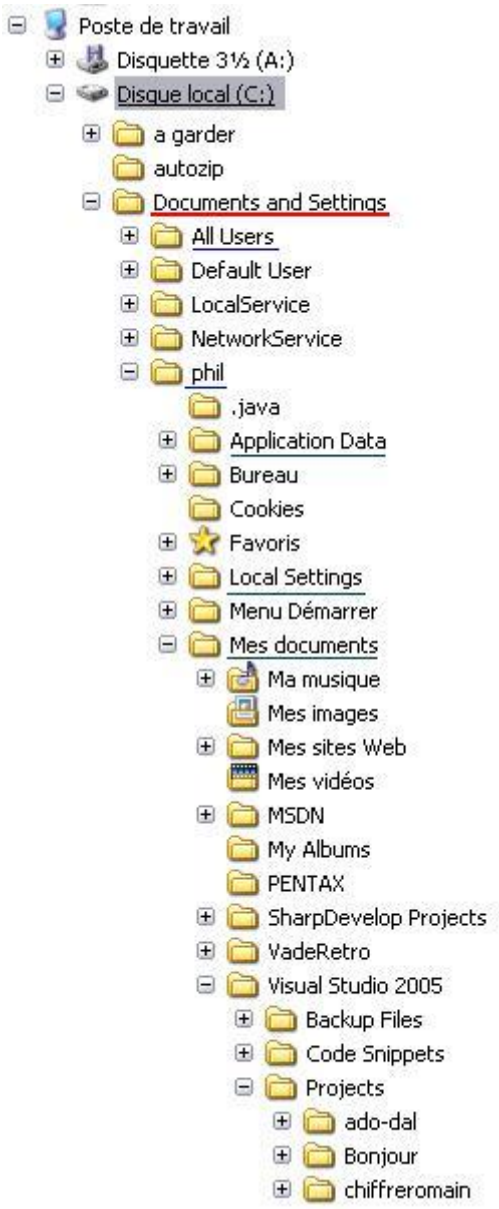



Un répertoire 'Programs Files' contient des sous-répertoires pour chaque programme contenant les exécutables.

Un répertoire 'Mes documents' contenant des sous-répertoires pour chaque programme contenant les fichiers de données.

Les paramètres de l'application sont dans un fichier .ini qui se trouve dans le répertoire Windows ou le répertoire de l'application ou la base de registre.

- **Windows XP :**




Un répertoire 'Programs Files' contient des sous-répertoires pour chaque programme contenant les exécutables.

Un répertoire 'Documents ans Settings' contient des sous-répertoires pour chaque utilisateur (ici le mien se nomme 'Phil') et un répertoire commun à tous les utilisateurs: AllUsers.

Dans chaque répertoire utilisateur, il y a :

- un répertoire 'Application Data' pour chaque programme contenant les fichiers de données.

C:\Documents and Settings\phil\Application Data\OpenOffice.org2\user\database ;

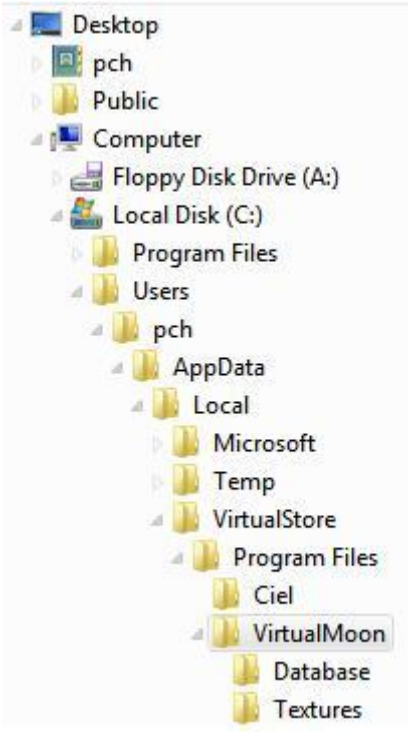
- un répertoire 'Mes documents'

C:\Documents and Settings\phil\Mes documents\Visual Studio 2005\Projects\chiffreromain ;

- mais aussi un répertoire 'Local Settings' pour les fichiers de configuration et les exécutables.

C:\Documents and Settings\phil\Local Settings\Application Data\Pentax\Digital Camera Utility.

- **Windows Vista :**




Version anglaise

Dans le Desktop,(information trouvée sur le Net)

Il y a :

un répertoire 'Programs Files' ;

un répertoire 'Users' qui contient des répertoires au nom de chaque utilisateur ('pch' ici par exemple.)

Dans chaque répertoire utilisateur, il y a un répertoire AppData pour les données (qui remplace Application Data). Dessous , il y a un répertoire Local et encore dessous un répertoire au nom de la société qui a fait le logiciel (ici Microsoft) puis un répertoire pour le programme et pour la version.

Ainsi les données du programme LDF version 1.1.0 crées par PolySoft seront dans le répertoire :

C:\Users\MyName\AppData\Local\PolySoft\LDF\1.1.0\

Version française il y a :

un répertoire 'Ordinateur'(au lieu de computer) ;

un sous-répertoire 'Disque local(C:) ;

un sous-répertoire 'Programmes'(au lieu de'Program Files') ;

un sous-répertoire MonProgramme ou se trouvent le programme exe.

Ainsi l'exécutable du programme LDF est dans le répertoire :

C:\Programmes\LDF

Il 'faudrait' utiliser ces répertoires pour les exécutables et les données !!!

Quand, par exemple, on installe un logiciel avec Click Once de VB2005 sur un ordinateur **Windows XP**, l'exécutable s'installe dans C:\Documents and Settings\Nom Utilisateur\Local Settings\Apps :

C:\Documents and Settings\phil\Local Settings\Apps\2.0\WD2P2BPB.7A5\OVEEZQLA.BC1\chif…
tion_6625898959f0f00b_0001.0000_56fb0b87e9540260

Le fichier de configuration user.config et les données dans C:\Documents and Settings\Nom Utilisateur\Application Data

C:\Documents and Settings\phil\Application Data\Polytel
\WindowsApplication1.vshos_Url_dhzke3q02e3bxuhunbsnp0jhjwtnzb1i\1.0.0.0

(On remarque que après '…Application Data' le chemin comporte \CompanyName\ProductName\ProductVersion)

Il ne 'faudrait' pas utiliser le répertoire 'Program Files\Mon Application' pour les données !!

En fait, beaucoup de programmes écrivent des données directement dans le répertoire courant de l'application (dans C;\Program Files\MonApplication) ; comment cela se passe ?

- Aucun problème avec Windows 98.
- Avec Windows XP puisque les comptes limités (comptes utilisateurs standards) n'ont pas accès en écriture au dossier %Program Files% et n'ont pas accès à HKEY-Local_Machine du registre; le problème a été résolu simplement : De manière générale les utilisateurs ont un compte administrateur, donc, ils ont tous les droits et peuvent écrire leurs données dans Program Files et modifier le registre !!
- Vista "contourne ce problème" : si on installe un programme qui veut utiliser le répertoire 'Program Files', Vista utilise un répertoire VirtualStore dans lequel il place un faux répertoire 'Program Files', en effet tous les programmes qui tentent d'écrire vers %Program Files% ou %windir% seront automatiquement redirigés vers 'C:\Users\Nom Utilisateur\AppData\Local\VirtualStore\Program Files'. Si le programme tente ensuite de lire le fichier précédemment écrit, le programme lira automatiquement le bon fichier. Tout ceci est invisible pour le programme, c'est Vista qui gère tout( le répertoire AppData\Local\VirtualStore serait caché). On appelle cela la 'virtualisation'. Attention donc, si le programme veut écrire dans C:\Program Files\MonProg\utilisateur , en fait, il écrira dans C:\Users\nom d'utilisateur\AppData\Local\VirtualStore\Program Files\MonProg\utilisateur.

## X-P-4 - Obtenir le répertoire de l'exécutable et des données

Rien de plus simple (dans un programme VB 2005 sous Windows XP), pour un utilisateur :

**Application.StartupPath** donne le répertoire de l'exécutable.

Exemple : dans l'environnement de développement VB 2005, pour le programme WindowsApplication2 :

C:\Documents and Settings\phil\Mes documents\Visual Studio 2005\Projects\WindowsApplication2\WindowsApplication2\bin\Debug

**Application.UserAppDataPath** donne le répertoire de données.

C:\Documents and Settings\phil\Application Data\WindowsApplication1\WindowsApplication1\1.0.0.0

De même on peut obtenir la clé de Registre des données d'application.

**Application.UserAppDatRegistry**

Pour les données et le registre commun à tous les utilisateurs :

```
Application.CommonAppDataPath

Application.CommonAppDatRegistry
```

La classe **Environment** donne des informations concernant l'environnement et la plateforme en cours ainsi que des moyens pour les manipuler. Par exemple: les arguments de la ligne de commande, le code de sortie, les paramètres des variables d'environnement, le contenu de la pile des appels, le temps écoulé depuis le dernier démarrage du système ou le numéro de version du Common Language Runtime, mais aussi certains répertoires.

```
Environment.CurrentDirectory  'donne le répertoire courant : ou le processus en cours démarre.

Environment.MachineName       'Obtient le nom NetBIOS de l'ordinateur local.

Environment.OsVersion 'Obtient un Identificateur et le numéro de version de la plateforme en
 cours.

Environment.SystemDirectory   'Obtient le chemin qualifié complet du répertoire du système

Environment.UserName          'Obtient le nom d'utilisateur de la personne qui a lancé le thread
 en cours.
```

La fonction GetFolderPath avec un argument faisant partie de l'énumération SpecialFolder retourne le répertoire d'un tas de choses.

**Exemple : Quel est le répertoire Système ?**

```
Environment.GetFolderPath(Environment.SpecialFolder.System)
```

En vb 2010 on trouve les répertoires :
Cookies ;
CDBurning ;
Desktop ;
Favories ;
History ;
Programs ;
MyMusic ;
MyPicture ;
Recent ;
SendTo ;
System ;
Templates ;
Personal (Mydocuments) ;
ProgramFiles ;
UserProfile ;
CommonDocuments, CommonMusic, CommonPictures, CommonVideos ;

MyVideos ;
Ressources ;
Windows.

## X-P-5 - Droits d'accès utilisateur dans Vista et Windows 7 : l'UAC

Pour réduire les effets des logiciels malveillants, Windows Vista inclut un nouveau modèle de sécurité dénommé User Account Control (UAC) qui marque un tournant comparé au modèle traditionnel des privilèges accordés aux utilisateurs de Windows, et affecte presque tous les utilisateurs Windows. L'UAC vise à améliorer l'expérience des utilisateurs standards de Windows, tout en réduisant le risque constitué par les logiciels malveillants.

Dans l'UAC, tous les utilisateurs Windows Vista, dont ceux qui possèdent des droits d'administrateur, interagissent avec leurs PC en tant qu'utilisateurs standards la plupart du temps. Le compte utilisateur standard Windows continue de ne pas avoir de privilèges d'administrateur, ce qui évite à un logiciel malveillant téléchargé malencontreusement par un tel compte de s'installer sur l'ordinateur. Ainsi, le logiciel malveillant qui s'infiltre d'une façon ou d'une autre dans un PC ne peut accéder ni à la base des registres ni aux répertoires protégés.

Si vous essayez d'effectuer une tâche qui nécessite des privilèges d'administrateur, comme l'installation d'un logiciel ou le changement d'état du pare-feu Windows, Windows Vista vous invite explicitement à donner votre autorisation et à vous identifier avant de vous faire passer temporairement au rang d'administrateur pour achever la tâche en cours. Pour un utilisateur standard, cela signifie concrètement la saisie d'un nom d'utilisateur et d'un mot de passe qui appartiennent à un membre du groupe Administrateurs

Dans Windows Vista avec le modèle UAC, les comptes utilisateurs standards ont été modifiés pour offrir des privilèges supplémentaires :

- visualiser les horloges systèmes et les calendriers ;
- modifier les fuseaux horaires ;
- changer les paramètres de gestion de la puissance ;
- ajouter des imprimantes si les bons drivers sont installés sur l'ordinateur ;
- créer et configurer des connexions VPN ;
- installer les mises à jour Windows essentielles.

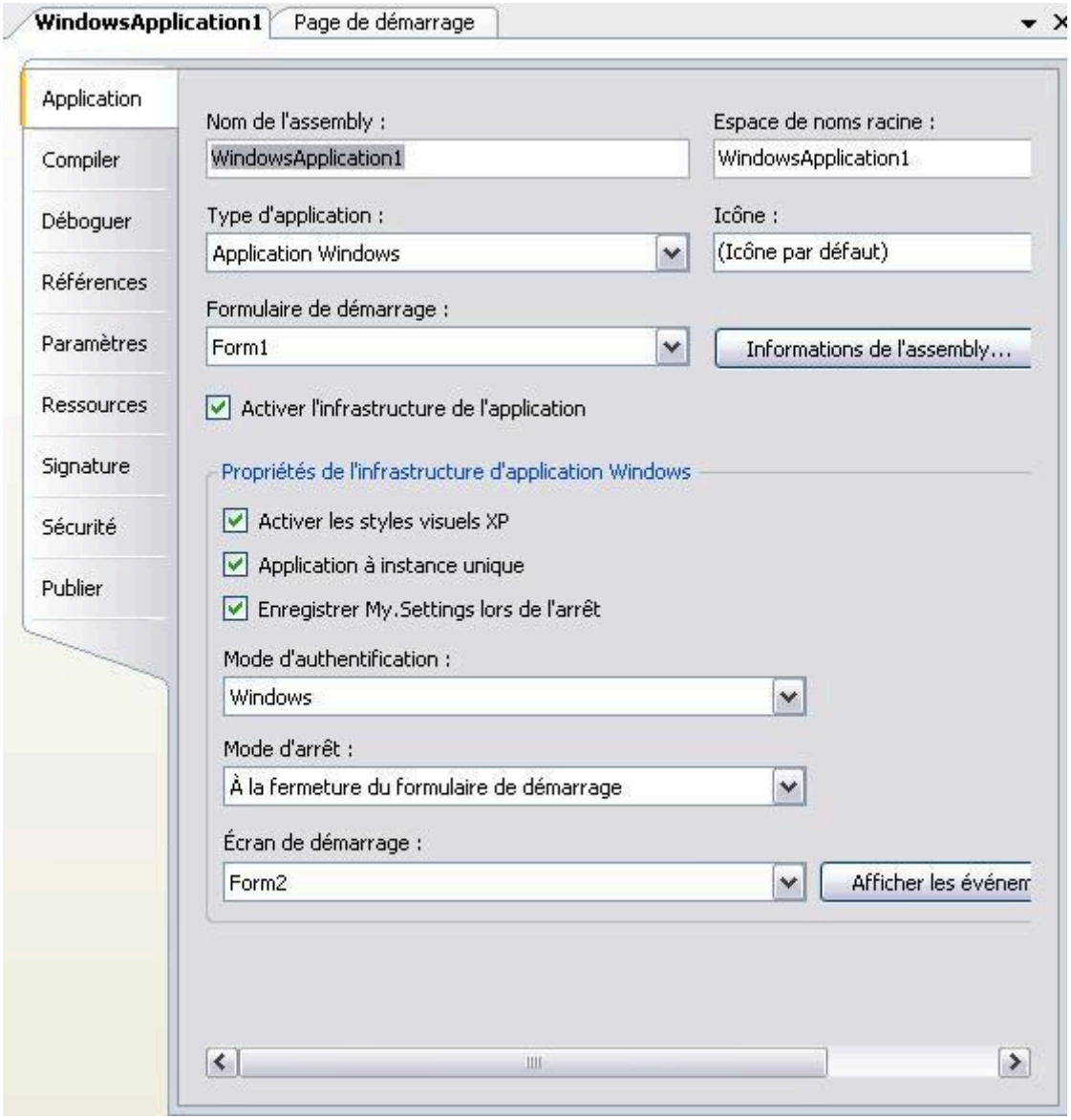
Dans les versions précédentes de Windows, un non-administrateur ne pouvait pas facilement comprendre quelles actions lui étaient autorisées. Dorénavant, Windows Vista utilise une icône de protection pour vous aider à comprendre quelles tâches seuls les administrateurs ont le droit d'effectuer

## X-Q - Choisir une icône, utiliser la barre de tâches - Créer un raccourci, lancer au démarrage

En VB 2005.

### X-Q-1 - Icône de l'application

Pour avoir une icône liée à notre application, nous pouvons le faire au travers de l'onglet "Application" (dans les propriétés du projet) au niveau "Icône". Remarquez que nous retrouvons les icônes importées dans les ressources projets




## X-Q-2 - Bouton dans la barre des tâches

Comment mettre un bouton correspondant à la form dans la barre de tâches ?


Mettre la propriété **ShowInTaskBar** de la form à True.

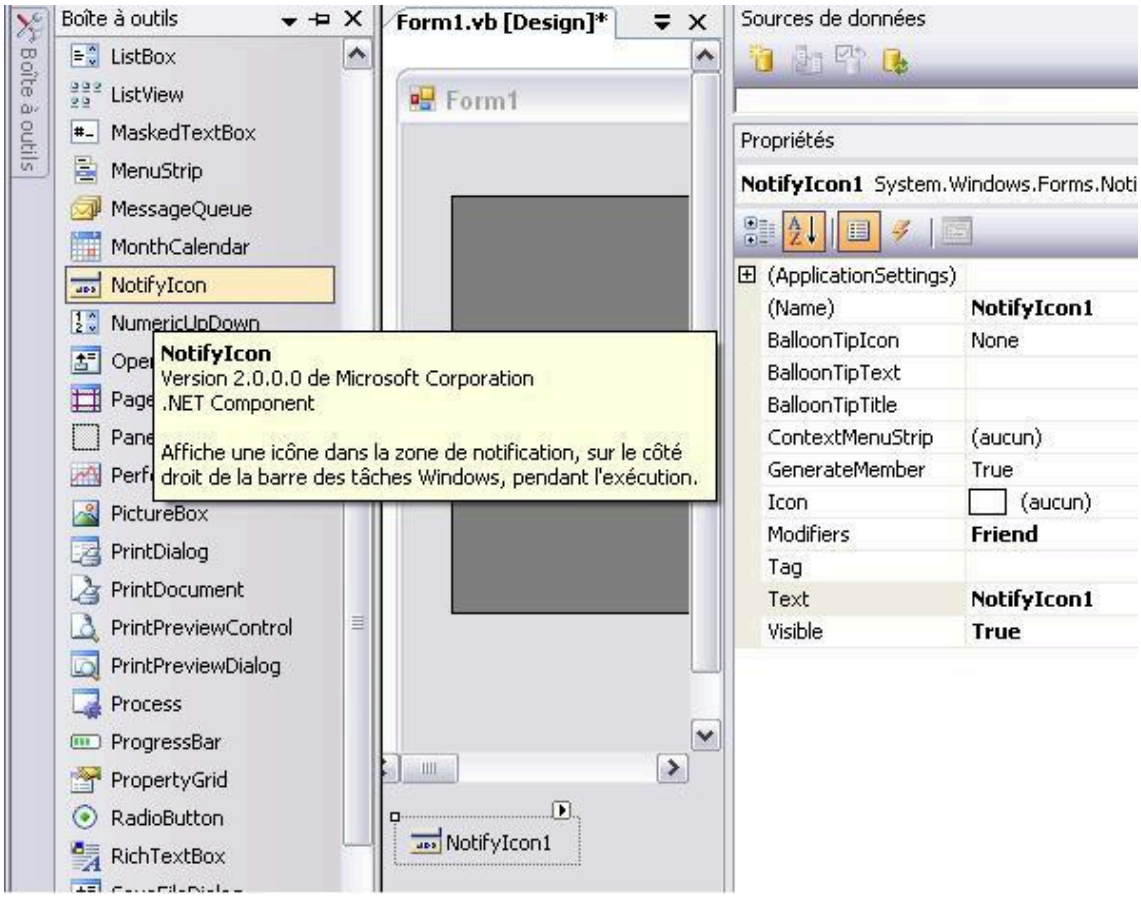
Il est préférable de le faire pour une seule form de l'application.

## X-Q-3 - Icône dans la barre de processus : NotifyIcon

Comment mettre une petite icône dans la barre de processus ? Exemple : mettre une icône bleue dans la barre de tâches avec une info bulle indiquant le nom du programme (ici : LDF).


Le Framework .Net 2.0 dispose de composants vous permettant d'ajouter très rapidement cette fonctionnalité. Ce sont le **NotifyIcon** et le **ContextMenuStrip**.

Dans la fenêtre "ToolBox", glissez un composant "NotifyIcon" sur notre formulaire. Il apparait un composant NotifyIcon1 sous le formulaire :




Dans la fenêtre de propriétés de NotifyIcon1, donner à sa propriété "Text" la valeur «LDF» ; ceci correspond à l'info bulle qui s'affiche lorsque le pointeur de la souris vient se positionner au-dessus de l'icône, dans la barre de processus.

Pour indiquer l'icône à utiliser dans la barre de tâches :

clic droit sur le composant NotifyIcon1 en bas du formulaire, puis choisir une icône. (Fichier avec extension .ico).

Par code dans FormLoad par exemple : **NotifyIcon1.Icon = MyIcone.ico**

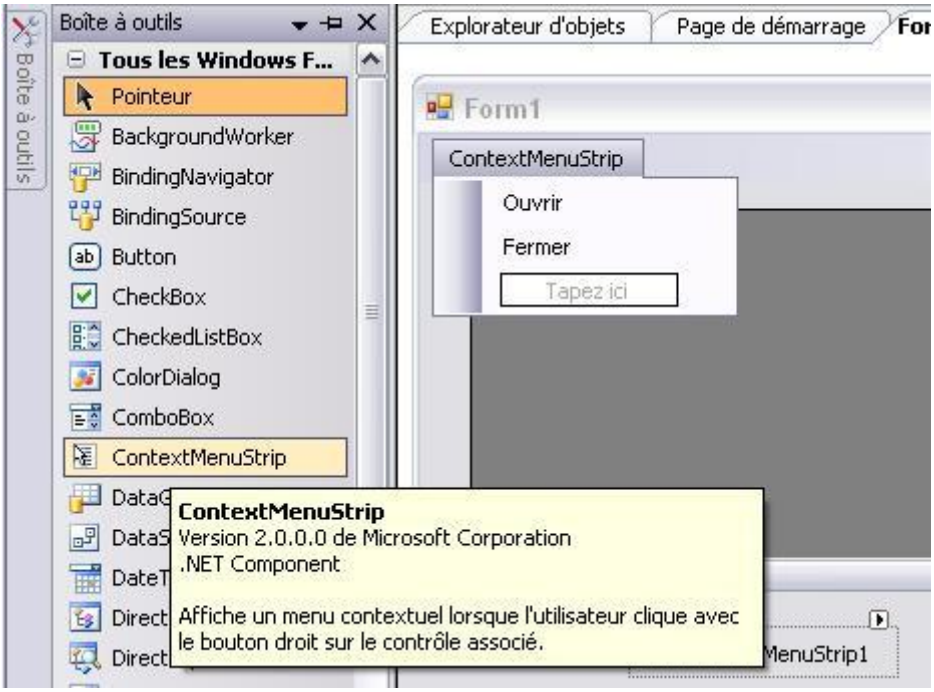
Noter que dans la fenêtre de propriétés on ne peut pas changer l'icône.

**Création du menu associé à l'icône de la barre de tâches.**

L'idée à présent est de créer un menu contextuel sur l'action "clic droit" de notre icône de notification.

Déposez un composant "ContextMenuStrip" sur notre formulaire dans la partie Design. Pour associer les 2 composants, dans les propriétés de "NotifyIcon1", il suffit d' affecter à la propriété "ContextMenuStrip" le composant "ContextMenuStrip1".

Pour définir les menus du "ContextMenuStrip", il suffit de cliquer sur le composant "ContextMenuStrip" pour faire apparaitre un menu (en mode design uniquement). Celui qui sera visible dans la barre de tâches en cours d'exécution.




Il suffit de remplir le menu comme d'habitude. On ajoute par exemple 'Ouvrir' et 'Fermer'.

En double-cliquant sur le menu, vous générez automatiquement la procédure événement dans la partie Code.

Exemple de code pour 'Ouvrir' et 'Fermer'.

```
Private Sub OuvrirToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
 System.EventArgs)
_Handles OuvrirToolStripMenuItem.Clic

    If Me.WindowState = FormWindowState.Minimized Then

        Me.WindowState = FormWindowState.Normal

    End If

End Sub



Private Sub FermerToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
 System.EventArgs)
_Handles FermerToolStripMenuItem.Clic

    Application.Exit()

End Sub
```

Au niveau du menu "Ouvrir", nous testons si l'application est réduite dans la barre des tâches. Si c'est le cas, nous l'affichons normalement. Sur l'événement "Fermer", nous allons tout simplement quitter l' application.

## X-Q-4 - Créer un raccourci sur le bureau

```
'Il faut d'abord ajouter la référence wshom.ocx qui est dans C:\Windows\System32
```

(menu Projet=>Propriétés de… Références , bouton Ajouter, Onglet Parcourir, aller dans C:\Windows\System32, cliquer sur wshom.ocx puis OK)

Exemple: créer un raccourci sur le bureau avec l'icône "euro.ico', comme texte LDF et qui lance c:\Program Files \LDF\ldf.exe

```
Dim Bureau As IWshRuntimeLibrary.WshShell

Dim Raccourci As IWshRuntimeLibrary.WshShortcut

Dim Nom As String

Bureau = New IWshRuntimeLibrary.WshShell

'   Chemin et nom du raccourci

Nom = My.Computer.FileSystem.SpecialDirectories.Desktop & "\LDF.lnk" 'pour 'LDF'

Raccourci = CType(Bureau.CreateShortcut(Nom), IWshRuntimeLibrary.WshShortcut)

'   Cible à exécuter

Raccourci.TargetPath = "c:\Program Files\ldf\ldf.exe"

'   Icône à utiliser

Raccourci.IconLocation = "C:\WINLDF\Icone\euro.ico"

'   Enregistrement du raccourci

Raccourci.Save()
```

## X-Q-5 - Lancer le programme au démarrage de Windows

Pour cela, il faut dans le registre (Software\Microsoft\Windows\CurrentVersion\Run) ajouter le nom du programme.

```
Imports Microsoft.Win32 'en haut du module.

Dim obj_RegistryKey As RegistryKey

obj_RegistryKey = Registry.CurrentUser.OpenSubKey("Software\Microsoft\Windows\CurrentVersion
\Run", True)

obj_RegistryKey.SetValue("LDF", "C:\Program Files\LDF.exe")
```

Voir aussi le chapitre sur le registre.

## X-Q-6 - Interdire de démarrer le programme dans une plage horaire.

C'est pas gentil !! d'empêcher le programme de démarrer entre 0H et 8H.

il faut mettre le code dans Application_StartUp (propriété du projet, onglet application, bouton 'Afficher les événements de l'application", Liste déroulante à gauche 'MyApplication', liste déroulante à droite 'StartUp').

```
Private Sub MyApplication_StartUp(…)
 If Now.TimeOfDay> New TimeSpan(0,0,0) OrElse Now.TimeOfDay< New TimeSpan(8,0,0)Then
   e.Cancel = True
```

```
  End If
End Sub
```

Dans la Sub MyApplication_ShutDown, on peut par exemple enregistrer les données avant que l'application se ferme.

# X-R - Multithreads

## X-R-1 - Un Thread, c'est quoi ?

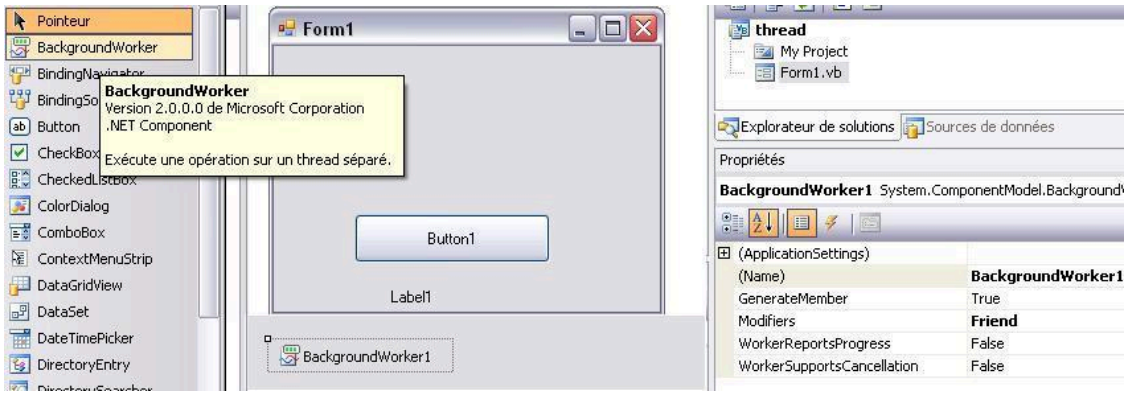
Le **thread** représente l'exécution d'un processus en mémoire. Un système multithread tel que Windows offre la capacité d'exécuter en parallèle plusieurs threads et donc plusieurs traitements en simultané.

On peut utiliser **la Classe Thread**, créer autant de thread que l'on veut, mais il faut gérer un tas de choses et c'est l'horreur.

On peut aussi (Framework 2) utiliser un **Thread d'arrière-plan** (et un seul) qui est très simple d'utilisation. Son intérêt est que lorsqu'on a une tâche très longue (très long calcul par exemple), il est possible d'effectuer le calcul long en arrière-plan, pendant ce temps, on peut continuer à travailler dans le formulaire (thread principal) ; quand le thread d'arrière-plan est terminé, on affiche les résultats.

## X-R-2 - Comment ajouter un Thread d'arrière-plan ?

Il faut aller chercher un composant BackgroundWorker dans la boite à outils et le déposer sur le formulaire, il apparait en dessous et se nomme par défaut BackgroundWorker1.




La propriété WorkerReportsProgress donne à notre BackgroundWorker la possibilité de nous informer ou non de son état d'avancement.

La propriété WorkerSupportsCancellation nous permet d'autoriser l'annulation de la tâche en cours du BackgroundWorker.

Dans le code :

```
BackGroundWorker1.RunWorkerAsync(Objet) permet de déclencher le thread d'arrière-plan.

BackGroundWorker1.DoWork : est l' événement qui se déclenche lorsque nous faisons appel au
 BackgroundWorker.
C'est cette routine qui tourne en arrière-plan.

ProgressChanged : Cet événement, si la propriété WorkerReportsProgress est activée,
se déclenche lorsque nous voulons indiquer que l'état d'avancement du BackgroundWorker change.

RunWorkerCompleted : Une fois le traitement du BackgroundWorker terminé cet événement est
 déclenché.
```

Exemple

Si on clique sur un bouton cela crée un thread d'arrière-plan qui effectue un calcul long.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles Button1.Click

 'La méthode RunWorkerAsync() du BackgroundWorker déclenche le thread d'arrière-plan.

  BackgroundWorker1.RunWorkerAsync()

End Sub



'La procédure DoWork contient le code effectué en arrière-plan.

Private Sub BackgroundWorker1_DoWork(ByVal sender As System.Object, _
 ByVal e As System.ComponentModel.DoWorkEventArgs) Handles BackgroundWorker1.DoWork

   'mes calculs très longs

End Sub

'Quand le code d'arrière-plan est terminé, la procédure RunWorkerCompleted est exécutée.

Private Sub BackgroundWorker1_RunWorkerCompleted(ByVal sender As Object, _
 ByVal e As System.ComponentModel.RunWorkerCompletedEventArgs) _
 Handles BackgroundWorker1.RunWorkerCompleted

' ici, elle affiche un message indiquant que le thread d'arrière-plan est terminé.

Label1.Text = "terminé"

End Sub
```

**La méthode RunWorkerAsync peut avoir un paramètre qui sera transmis au thread d'arrière-plan.**

Mais un seul. Ce paramètre étant de type objet, vous pouvez passer un tableau d'objets (string, int, etc.) ou même une structure.

Ici dans l'exemple, on a un paramètre numérique, utilisé dans le thread d'arrière-plan pour faire un calcul.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
 Handles Button1.Click

  BackgroundWorker1.RunWorkerAsync(180)

End Sub
```

Le paramètre , dans DoWork, se retrouve dans e.Argument , comme c'est un Objet, il faut le convertir en Integer pour l'utiliser :

```
Private Sub BackgroundWorker1_DoWork(ByVal sender As System.Object, _
 ByVal e As System.ComponentModel.DoWorkEventArgs) Handles BackgroundWorker1.DoWork

   a=a + CType (e.Argument, Integer)

End Sub
```

**Le thread d'arrière-plan peut appeler une Sub.**

```
Private Sub BackgroundWorker1_DoWork(ByVal sender As System.Object, _
 ByVal e As System.ComponentModel.DoWorkEventArgs) Handles BackgroundWorker1.DoWork
```

```
    Calcul()

End Sub

Sub Calcul ()

'Mes calculs

End Sub
```

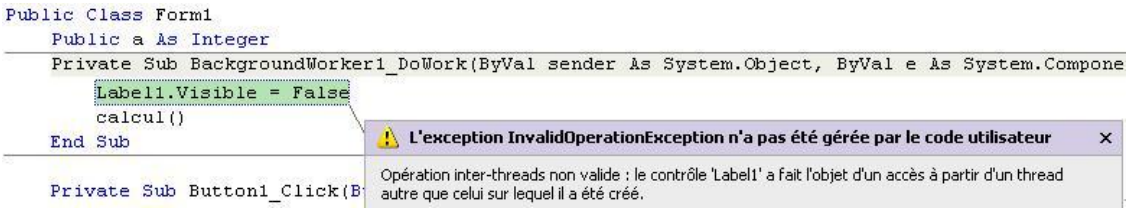
(Le thread principal peut lui aussi appeler la routine Calcul.)

**Les variables sont accessibles dans le thread d'arrière-plan :**

```
'MyVar par exemple qui est Public et déclarée en tête de module.

Public  MyVar As Integer = 1



Private Sub BackgroundWorker1_DoWork(ByVal sender As System.Object, _
 ByVal e As System.ComponentModel.DoWorkEventArgs) Handles BackgroundWorker1.DoWork

      MyVar=Myvar +1

End Sub
```

**Par contre les objets de l'interface (du thread principal) ne sont pas accessibles dans le thread d'arrière-plan.**

Cela déclenche une exception si on tente d'y accéder.




## X-R-3 - État d'avancement

Si la tâche d'arrière-plan est très longue, il peut être intéressant de montrer dans l'interface utilisateur, l'état d'avancement de cette tâche.

Mais on rappelle que la tâche de fond ne peut pas intervenir sur l'interface.

Il faut donc : mettre la propriété WorkerReportsProgress de notre BackgroundWorker à True.

Dans le thread d'arrière-plan, il faut, à chaque fois que l'on veut indiquer la progression, appeler la méthode ReportProgress en indiquant l'état d'avancement avec un paramètre.

```
Private Sub BackgroundWorker1_DoWork()

 Dim MyThread As BackgroundWorker = CType(sender, BackgroundWorker)'récupération du thread
 d'arrière plan

 MyThread.ReportProgress(pourcent)'pourcent est un Integer indiquant l'état d'avancement.

End Sub
```

Noter que c'est au programmeur de créer la logique calculant d'état d'avancement (et donc la valeur de la variable pourcent).

Enfin dans le thread principal, la Sub BackgroundWorker1_ProgressChanged() s'exécute à chaque fois que le thread d'arrière-plan le demande et met à jour un index visuel sur l'interface.

```
Private Sub BackgroundWorker1_ProgressChanged( _
ByVal sender As Object, _
ByVal e As ProgressChangedEventArgs) _
Handles BackgroundWorker1.ProgressChanged

MyProgressBarr.Value = e.ProgressPercentage
End Sub
```

## X-R-4 - Arrêter le thread en cours

Il suffit de faire dans le thread principal :

```
BackgroundWorker1.CancelAsync()
```

Dans le thread d'arrière-plan, il faut vérifier si l'arrêt à été demandé.

Dans DoWork on récupère le thread d'arrière-plan qui est le sender, on regarde si sa propriété CancellationPending est à True, si oui on met e.cancel à True ce qui arrête le thread d'arrière-plan.

```
Dim MyThread As BackgroundWorker = CType(sender, BackgroundWorker)

If MyThread.CancellationPending Then e.Cancel = True  'ce qui arrête le thread d'arrière-plan.
```

Si on veut tester la demande d'arrêt dans une Sub, il faut envoyer en paramètre à cette sub MyThread et e.

## X-R-5 - Résultat retourné par le thread d'arrière-plan

Il peut y avoir plusieurs types de résultats à la fin, on peut le voir dans l'argument e de type RunWorkerCompletedEventArgs retourné par la procédure BackgroundWorker1.RunWorkerCompleted.

* Il y a eu une erreur pendant le traitement. Dans ce cas la propriété e.Error est différente de null.

* Le traitement a été annulé. Dans ce cas la propriété e.Canceled est à true.

* Le traitement s'est déroulé normalement. Le résultat se trouve dans la propriété e.Result .(bien sûr ,dans DoWork il faut avoir mis le résultat des calculs dans e.Result)

Exemple de traitement :

```
Private Sub BackgroundWorker1_RunWorkerCompleted( _
    ByVal sender As Object, _
    ByVal e As RunWorkerCompletedEventArgs) _
    Handles BackgroundWorker1.RunWorkerCompleted

    If Not (e.Error Is Nothing) Then
        lblResult.Text = "Il y a eu une erreur : " + e.Error.Message
    ElseIf e.Cancelled Then
        lblResult.Text = "Opération annulée "
    Else
        lblResult.Text = "Opération OK  Résultat : " + e.Result.ToString
    End If

End Sub
```

# XI - Interface utilisateur en WPF

## XI-A - Définition : WPF, XAML, SilverLight

Les WPF sont un nouveau système d'affichage.

À partir du Framework 3.0 et dans le Framework 3.5 (VB 2008), dans Visual Basic, on peut utiliser deux manières de dessiner l'interface utilisateur :

- avec les **Windows Forms** qui utilisent **GDI+ et travaillent sur des Bitmaps** ;

- avec **WPF** qui utilise un moteur de rendu vectoriel et des accélérations matérielles (Direct X) pour afficher. Cela permet d'afficher de la 2D, de la 3D, des animations, des documents, des dessins vectoriels ou BitMap, de l'audio, de la vidéo; de plus l'affichage étant vectoriel, il n'y a pas de dépendance avec les dimensions de l'écran.

Que signifie WPF ?

**W**indows **P**resentation **F**oundation (ex. : 'avalon').

Les WPF fonctionnent à partir de l'édition VB Express 2008, il y a même un designer WPF orienté développeur permettant de créer une interface WPF 'simple'. On peut charger VB Express 2008 et le Framework 3.5 ou plutôt VB Express 2010 et le Framework 4 en français (usage gratuit) pour utiliser les WPF.

Il existe aussi un logiciel autonome nommé 'Expression Blend' qui permet de dessiner l'interface en WPF, ce logiciel payant est orienté graphiste et permet de créer des interfaces complexes qui pourront ensuite être utilisées en VB.net , en C#…

Que signifie XAML ?

XAML= "eXtensible Application Markup Language", en français: "langage à balises d'applications extensibles" est un langage utilisant la syntaxe XML et permettant de définir des interfaces d'applications. En d'autres termes, quand vous dessinez une interface utilisateur en WPF, elle est enregistrée en XAML.

Rapport avec SILVERLIGHT ?

Silverlight est en fait un sous-ensemble de WPF allégé pour être portable et utilisé par les navigateurs Web. C'est une alternative à Flash, un plugin léger, qui s'installe sur le poste utilisateur et complète le navigateur (IE, Firefox) avec une interface graphique ayant une partie des capacités graphiques de WPF permettant des applications Internet riches, réalisant des interfaces intégrant des animations, des vidéos. - Se programme en JavaScript. Utilise XAML pour décrire l'interface graphique, - Graphisme 2D vectoriel avec changement de taille des objets. - Fonctionne avec Ajax, donc JavaScript, DOMet XMLHttpRequest. Silverlight fonctionne aussi en mode local sous environnement .NET. Sur le Web, les composantes sont accessibles par le biais de Active X sous Internet Explorer tandis que Firefox et les autres navigateurs utilisent le système de plugin de Mozilla.
Silverlight utilise aussi du XAML et la même syntaxe que WPF.

Une autre fonction de WPF consiste en **la séparation de la présentation et de la logique**. Cela signifie que les concepteurs peuvent travailler sur l'apparence d'une application et le comportement de l'apparence en utilisant uniquement XAML pendant que les développeurs travaillent sur la programmation logique à l'aide de Visual Basic.

**Ou trouvez de la documentation ?**

### Msdn WPF: la bible

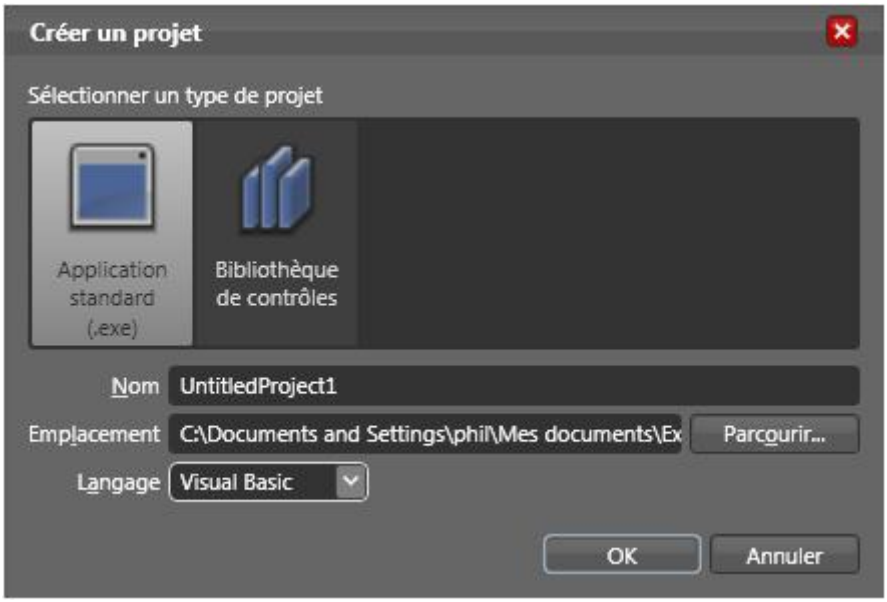
(http://msdn.microsoft.com/fr-fr/library/ms754130.aspx)

# XI-B - Créer une interface WPF avec Expression Blend

Expression Blend est un logiciel autonome permettant de créer des interfaces WPF, il est très puissant et orienté graphiste. Les interfaces pourront ensuite être utilisées dans Visual Studio. On peut aussi créer une interface Wpf dans VB 2008, mais Vb est plutôt orienté développeur. Écrit par Microsoft, Expression Blend, coûte quelques centaines d'euros.

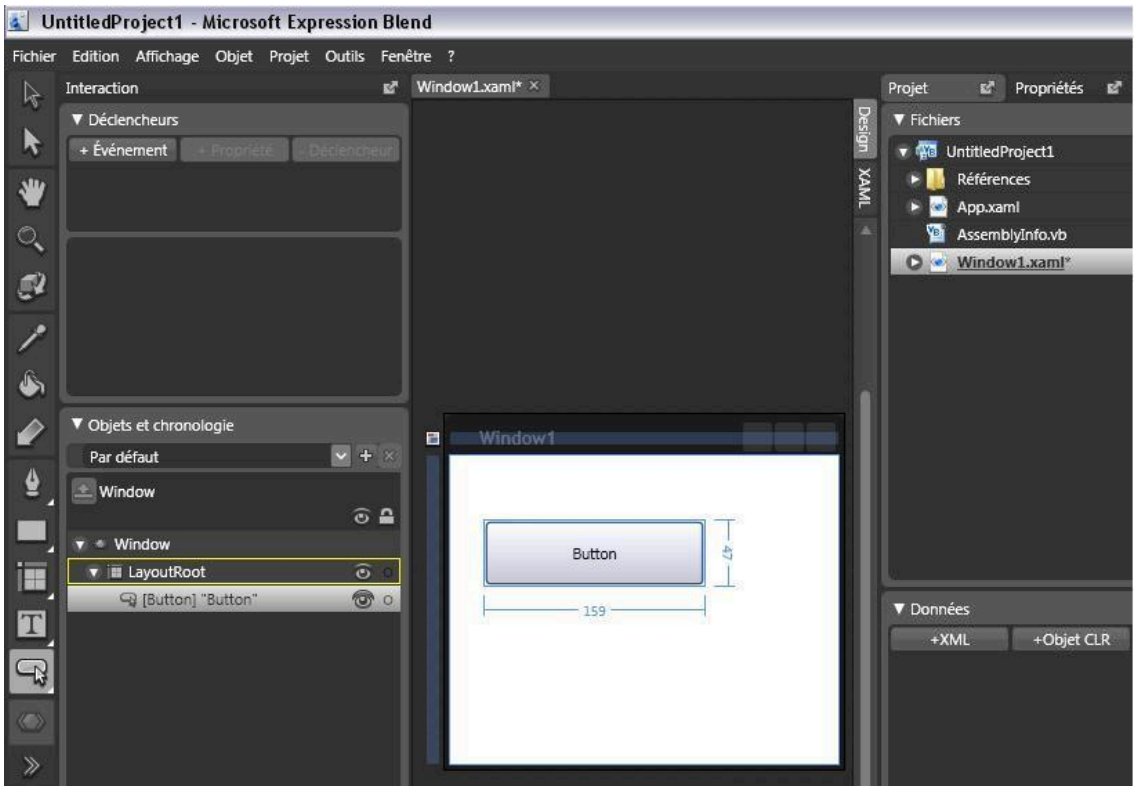
## XI-B-1 - Ouvrir Expression Blend

Expression Blend est téléchargeable gratuitement sur le Web en version démo 30j. Lancer Expression Blend: Faire menu 'Fichier', 'Nouveau Projet'.



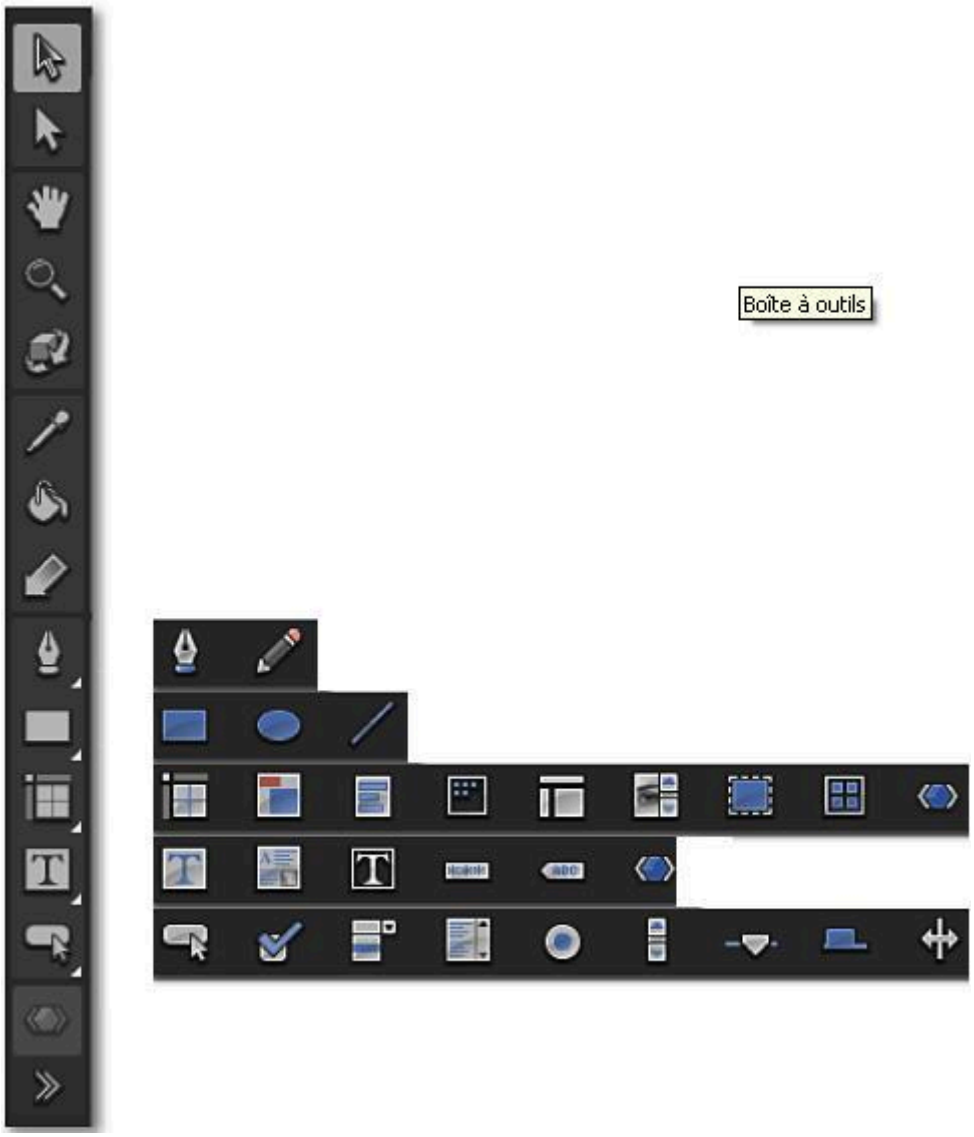

Choisir Application standard, dérouler la liste langage pour choisir 'Visual Basic' puis 'OK'.

On se retrouve dans un nouvel environnement, avec un formulaire nommé 'Window1'

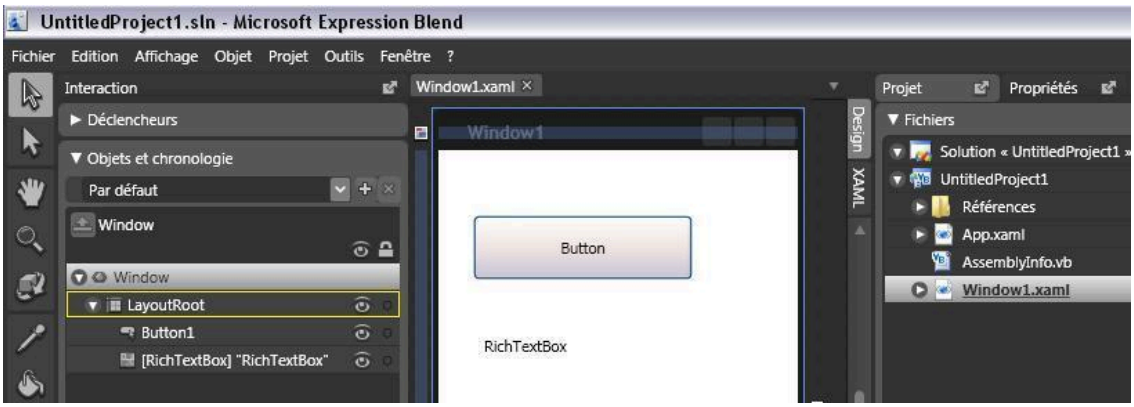



On est en mode 'design', mais si on veut, on peut voir le code XAML correspondant à l'interface en cliquant sur l'onglet vertical à droite.

On peut mettre dans le formulaire Window1 des objets visuels. La barre d'outils à gauche permet de choisir un objet et de le mettre dans le formulaire: cliquer sur l'objet puis sur le formulaire appuyer sur le bouton gauche de la souris, déplacer, lâcher. Si on clique droit sur un des objets dans la barre, on a accès à plusieurs objets différents.




Exemple:Ci dessous, on a mis un bouton et un RichTextBox.

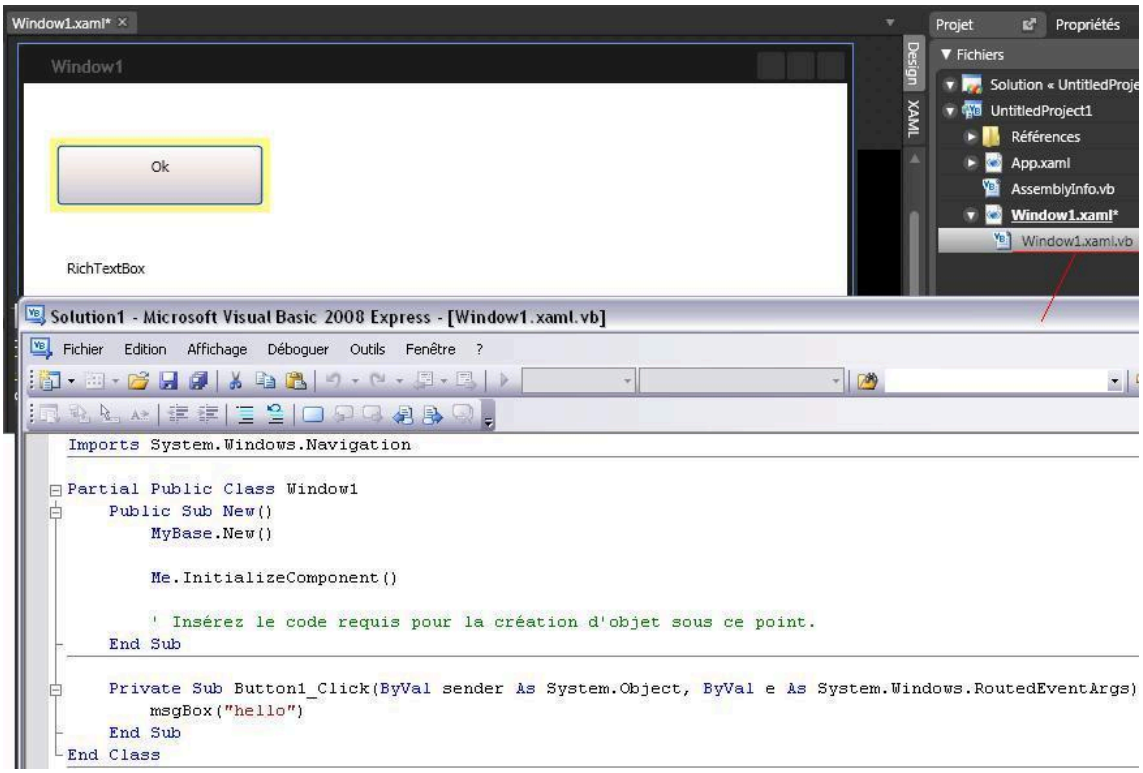



Quand le focus est sur un objet, on a accès aux propriétés de cet objet en cliquant sur l'onglet 'Propriétés' à droite :


On a une quantité phénoménale de propriétés. Pour modifier le texte afficher sur le bouton, on peut faire menu "Objet" puis "Modifier le texte" ou sur le bouton faire clic droit puis "Modifier le texte". Le texte sur le bouton passe en mode édition.

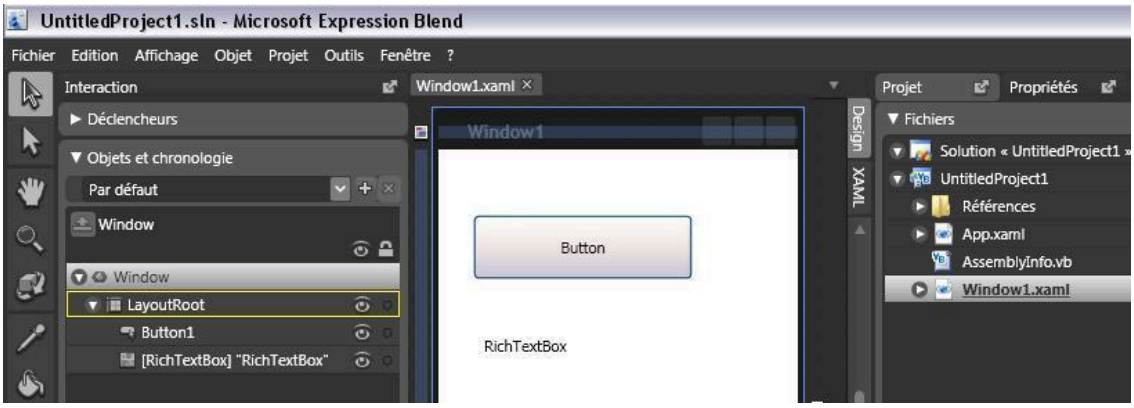
## XI-B-2 - Écrire du code VB

On peut écrire du code VB à partir de Blend. Dans la fenêtre de projet à droite, dérouler Window1.xaml qui correspond au formulaire, puis double-cliquer sur Window1.xaml.vb, VB 2008 s'ouvre et vous avez accès aux procédures événement, vous pouvez écrire du code VB.

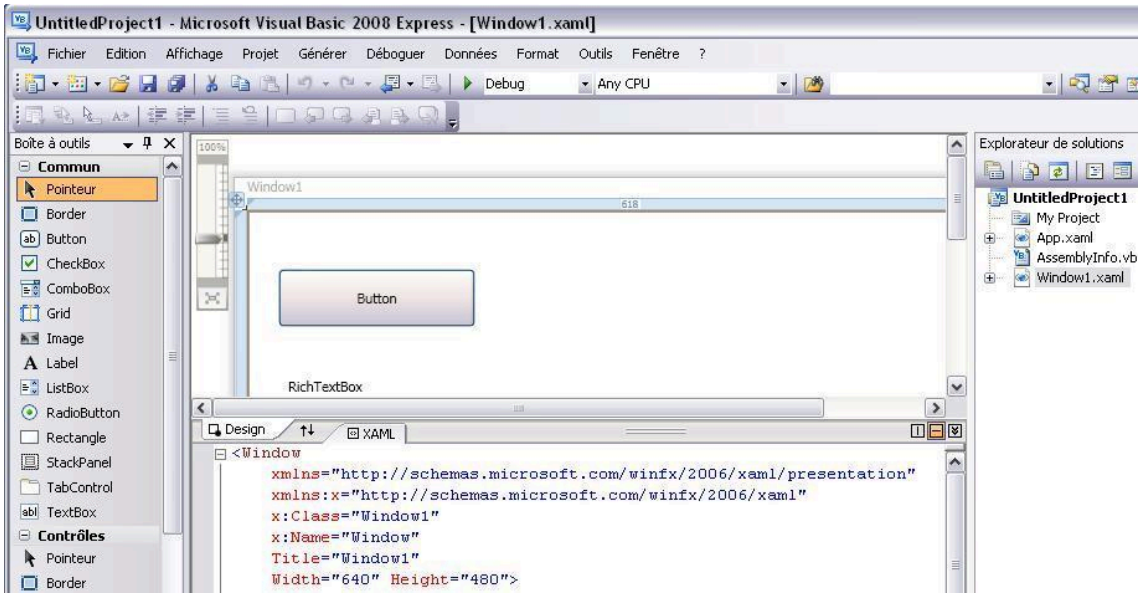



## XI-B-3 - Passer l'interface dans VB

À partir d'Expression Blend il faut enregistrer le projet : menu 'Fichier' puis 'Enregistrer'




Puis on peut 'récupérer' l'interface WPF dans VB 2008 : ouvrir VB 2008, charger le Projet WPF (on passe automatiquement par le mode d'importation)



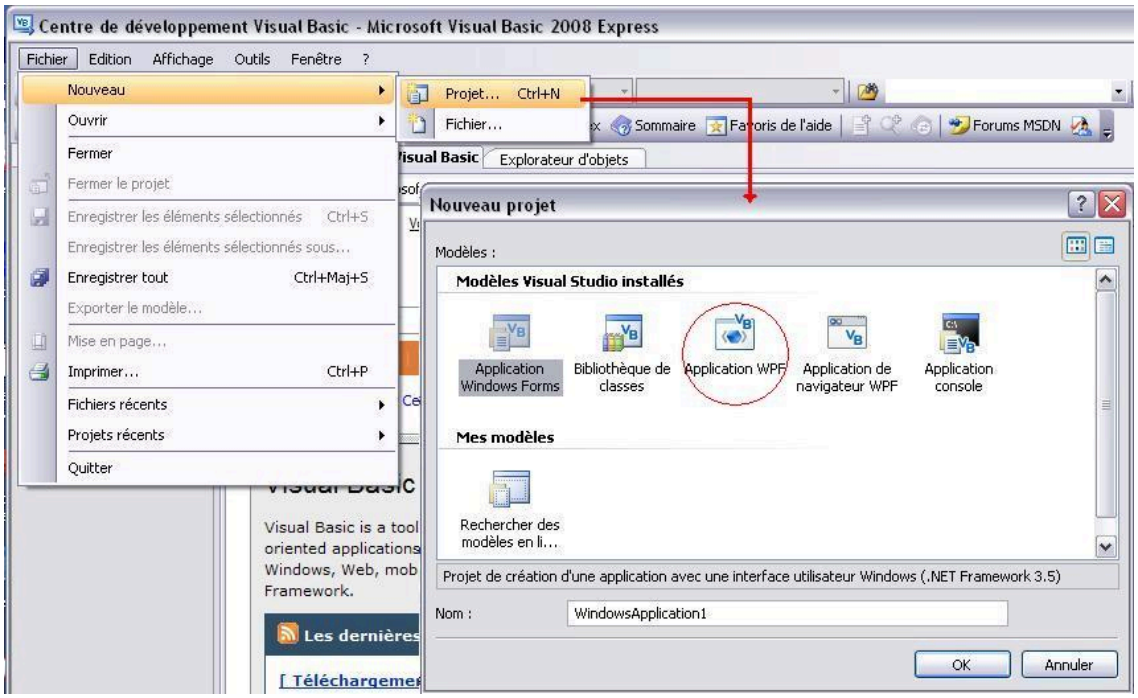

La belle interface WPF se charge dans VB 2008. Il suffit de double-cliquer sur le bouton et d'entrer le code dans les procédures événements.

## XI-C - Créer une interface WPF avec Visual Studio

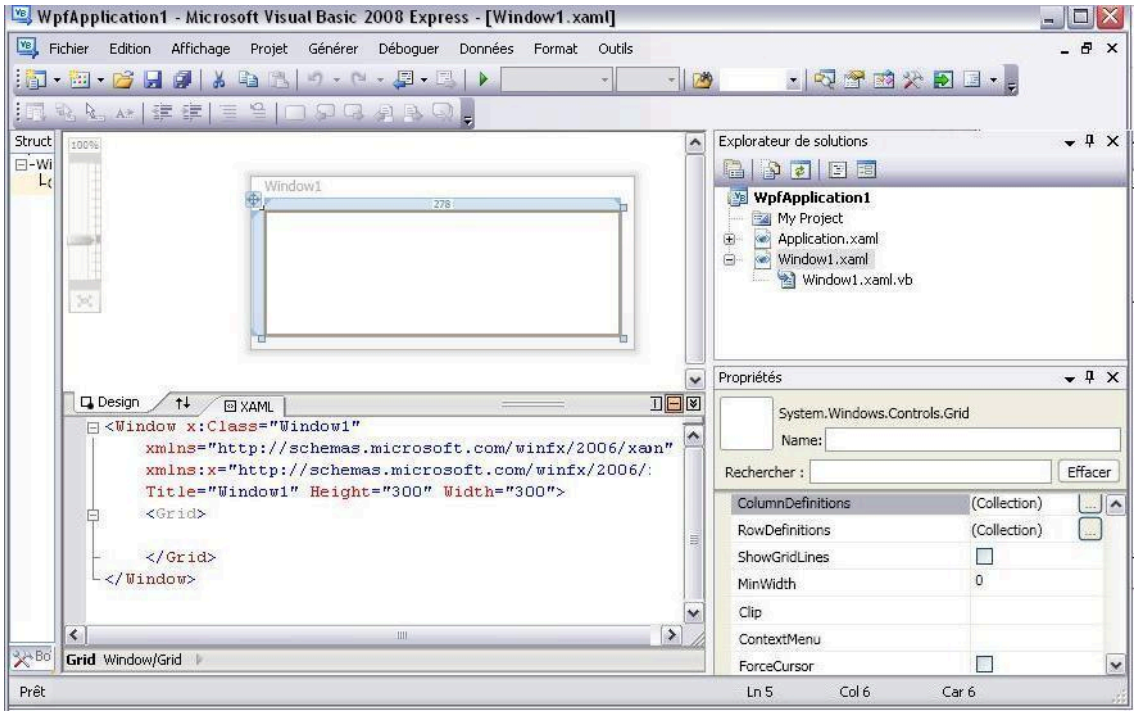
À partir de VB 2008, on peut utiliser les WPF.

Le concepteur WPF de Visual Basic est optimisé pour le développement de logiciel (Expression Blend programme extérieur, est plutôt orienté conception graphique). Voyons comment faire une interface utilisateur avec les WPF dans Visual Basic Express 2008.

Ouvrir VB 2008. Faire menu 'Fichier', 'Nouveau', 'Projet'.

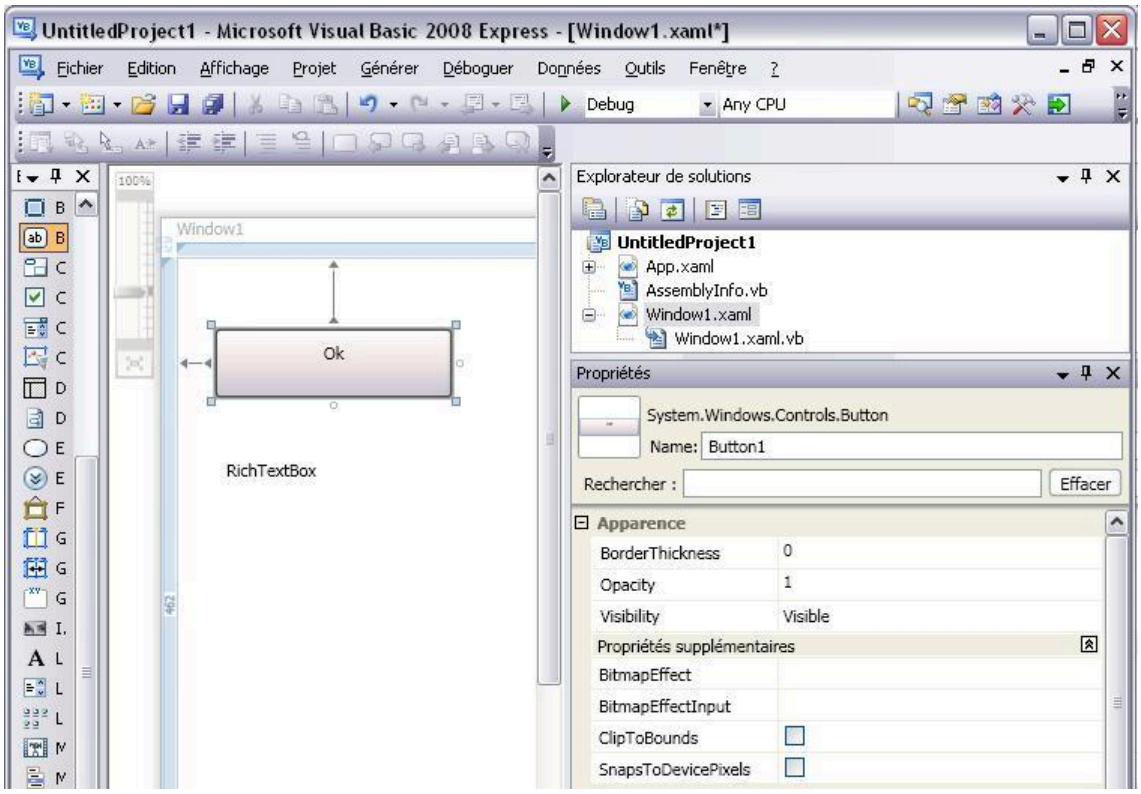



On choisit 'Application WPF',puis on clique sur 'OK' ; on se retrouve dans un nouvel environnement :




Il y a le 'designer' en haut qui permet de dessiner l'interface que verra l'utilisateur. Le designer génère le code XAML, on le voit en bas, il décrit l'interface en XAML. On peut aussi écrire directement du XAML en bas, ce qui modifie l'interface dans le designer. Les formulaires et contrôles sont différents de ceux des Windows Forms, ainsi que les propriétés !!

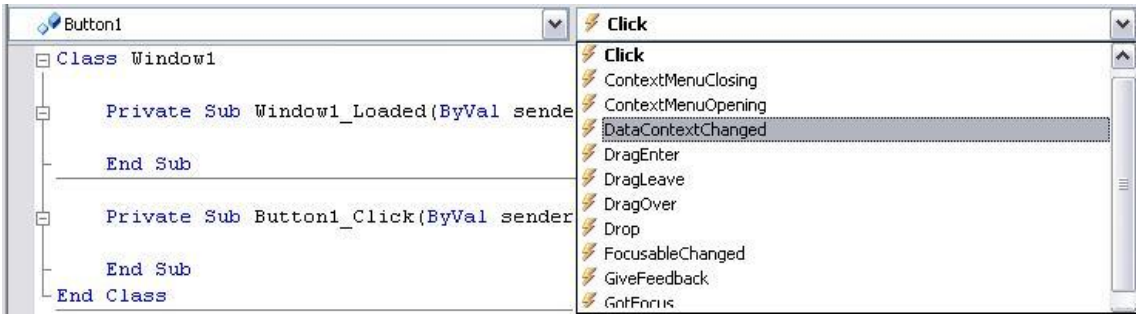
Le logiciel a créé un formulaire vide nommé 'Window1'. Nous pouvons ajouter un bouton par exemple, pour cela on clique sur le bouton dans les outils à gauche puis sur le formulaire, on appuie (bouton gauche de la souris), on déplace puis on lâche. Le bouton apparait.




En bas à droite, on a la fenêtre de propriétés du bouton. Les contrôles font partie de l'espace de noms System.Windows.Controls.

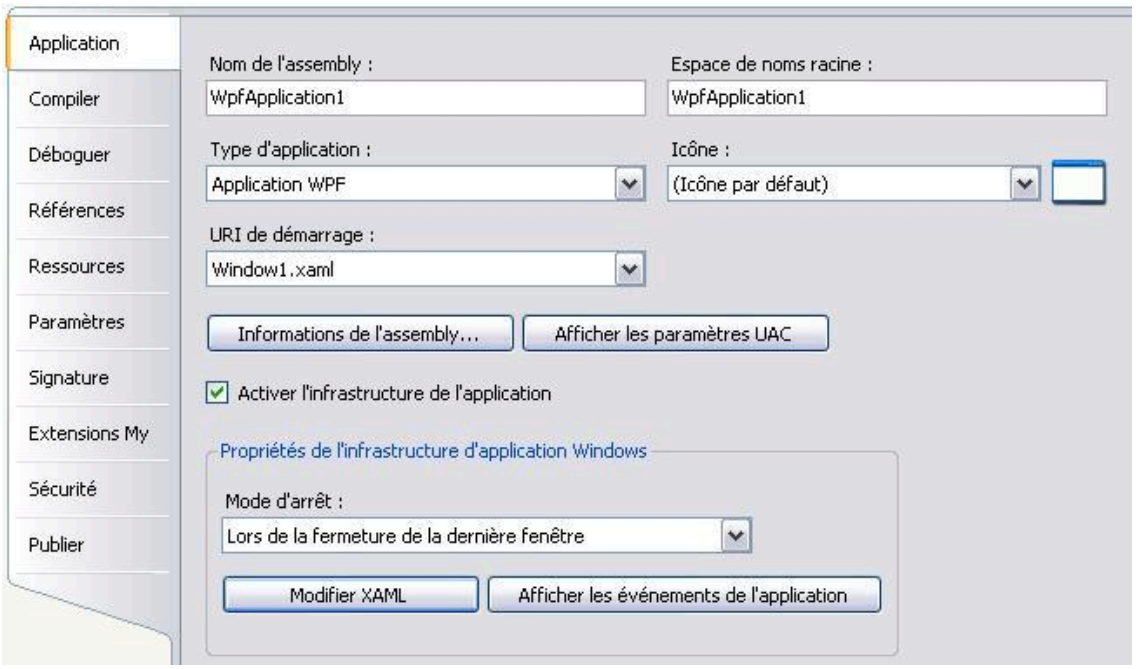
Dans la version Express on peut créer une interface simple, mais pour une interface extrêmement élaborée (dégradé de couleur, animation…) il faut mieux utiliser Expression Blend. On peut aussi écrire du code XAML pour faire du graphisme élaboré.

Si on double-clique sur un bouton, par exemple, on se retrouve dans la procédure événement correspondante :




On se rend compte que les événements là aussi ne sont pas les mêmes que pour les Windows Forms.

Les propriétés du projet (menu 'Projet', 'Propriétés de…') sont là aussi un peu différentes.




Conclusion : le XAML permet de décrire l'apparence de l'application, tandis que les événements sont gérés par un langage .NET classique comme le Visual Basic.

# XI-D - Le XAML

## XI-D-1 - Définition du XAML

XAML= "eXtensible Application Markup Language", en français : "langage à balises d'applications extensibles". Il s'agit d'un langage XML permettant de définir des interfaces d'applications des WPF (dans Silverlight, VB 2008). En d'autres termes, quand vous dessinez une interface utilisateur en WPF, elle est enregistrée en XAML.

Le code XAML respecte la syntaxe XML (voir le chapitre sur le format XML dans les annexes) ; il est enregistré dans un fichier ayant l'extension .xaml.

## XI-D-2 - Balisage

XAML est un langage 'balisé'.

Qu'est-ce qu'une balise ? 'Élément Sémantique de base' des langages de balisage. En XML une balise est un 'mot-clé', un élément, compris entre crochets qui possède un nom et parfois des attributs. En XAML l'élément est le nom d'un objet visuel (Buttom, TextBlock…)d'une propriété de cet objet visuel (Background,Content…), ou de tout élément intervenant dans lUI (une Brush, un ListItem…) .

Toutes les balises doivent être ouvertes puis refermées . On retrouvera donc toujours une balise de début et une balise de fin. La balise de fin porte le même nom que la balise de début à l'exception du nom de la balise qui est précédé du signe /.

Exemple :

```
<Button>
</Button>
```

On peut combiner balise de début et de fin :

```
<Button />
```

Cela suffit à afficher un bouton.

Les balises peuvent être intriquées :

```
<Grid>
 <Button>
 </Button>
</Grid>
```

La dernière balise ouverte doit être la première fermée.

## XI-D-3 - Case, espace, tabulation, commentaire

XAML est sensible à la Case: majuscules et minuscules ne sont pas équivalentes. Les espaces et tabulations sont ignorés.

Attention donc quand on tape du code XAML : 'grid' ou 'click' ne sont pas acceptés, il faut taper 'Grid' et 'Click'.

Voici un commentaire en XAML :

```
<!--Commentaire
    ici -->
```

## XI-D-4 - Attribut, Propriété

Un objet visuel a des propriétés: la couleur de fond, les dimensions d'un bouton, le texte dans un TextBlock sont les propriétés d'un bouton par exemple. Il y a différentes manières d'indiquer les valeurs d'une propriété d'un objet en XAML.

1- Les propriétés des objets visuels peuvent s'écrire sous forme d' **attributs**. Un attribut est le nom d'une propriété de la balise souvent associé à une valeur, il est mis dans la balise d'ouverture après le nom de la balise :

```
<Button Background="Blue" Foreground="Red" Content="C'est un bouton"/>
```

Ici dans un bouton, on a utilisé les attributs Background, Foreground Content, pour indiquer la couleur du fond, du texte et le texte à afficher dans le bouton. À un attribut est attribuée une valeur, cette valeur qui est entre guillemets est une simple String. On remarque que la syntaxe est simple, mais on ne peut même que des valeurs simples.

2- Les propriétés des objets visuels peuvent aussi s'écrire sous forme d'objet balisé. Dans un format alternative appelée "Property element syntax" (**syntaxe d'élément propriété**), qui étend la syntaxe de XML. Une propriété a aussi une balise de début et de fin, la syntaxe est celle du XML c'est-à-dire de la forme :

```
<TypeName.Property>
```

Voyons par exemple la propriété Background d'un bouton :

```
<Button.Background>
</Button.Background>
```

L'intérêt de cette syntaxe est qu'entre les 2 balises, on peut mettre d'autres balises ou mettre un élément complexe (comme une Brush par exemple).

Voici l'exemple de notre bouton :

```
<Button>
<Button.Background>
<SolidColorBrush Color="Blue"/>
</Button.Background>
<Button.Foreground>
<SolidColorBrush Color="Red"/>
</Button.Foreground>
<Button.Content>
C'est un bouton
</Button.Content>
</Button>
```

3- Propriété par défaut

Pour le texte affiché dans un bouton, on a vu qu'on pouvait l'écrire avec l'attribut content.

```
<Button Content="Cliquez ici !"/>
```

et aussi sous forme d'objet.

```
<Button>
<Button.Content>
Cliquez ici
</Button.Content>
</Button>
```

Mais comme Content est la propriété par défaut des boutons, on peut l'écrire aussi comme cela :

```
<Button>Cliquez ici !</Button>
```

De même la propriété 'Text' d'un TextBlock peut s'écrire ainsi :

```
<TextBlock>
Hello!
</TextBlock>
```

On parle ici d'élément de contenu.

Remarque : on a accès dans un objet aux propriétés qui sont héritées de la classe de base : par exemple dans un bouton, on a accès à la propriété BackGround qui est une propriété héritée de 'control'.

**Propriété attachée**

On peut utiliser dans un objet une propriété 'attachée' appartenant au parent :

```
<DockPanel>
    <Button DockPanel.Dock="Left">Annuler</Button>
</DockPanel>
```

Le bouton est collé sur le bord gauche du DockPanel grâce à la propriété 'attachée' Dock qui prend la valeur 'Left' de l'énumération Dock.

## XI-D-5 - Élément racine

Il doit y avoir un élément racine (root) et un seul dans un fichier xaml, il contient tous les autres. En VB c'est une fenêtre (Window).

```
<Window x:Class="Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
</Window>
```

La balise Root a des attributs spécifiques :

**x:Class="Window1"**

Indique la Classe partielle de la fenêtre. Quand le code est compilé c'est la class qui fait le lien avec le code VB.

**xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"**

**xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"**

Indique l'espace de noms par défaut. Puis l'espace de noms XAML ('xmlns:' permet en effet d'inclure un espace de nom).

Une fenêtre Window ne pouvant contenir qu'un objet, VB y met une grid dedans ; la grid est un conteneur qui lui peut contenir plusieurs objets.

```
<Window x:Class="Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Window1" >
<Grid >
</Grid>
</Window>
```

Pour Expression Blend dans une 'Windows' il y a un 'FlowPanel'.

Pour Silverlight l'élément root (correspondant à une page) est UserControl (on m'a dit !).

```
<UserControl x:Class="MySilverlight.Page"
xmlns="http://schemas.microsoft.com/client/2007"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
>
<Grid Background="OldLace">
<!--more content here-->
</Grid>
</UserControl>
```

Pour un objet comme une Window, il ne peut y avoir qu'un objet enfant.

Pour un Panel (Conteneur permettant de positionner les contrôles comme StackPanel, DockPanel…), il peut y avoir plusieurs enfants (donc plusieurs objets contenus dans le Panel). Ils sont dans la collection Children du Panel, dont la balise est :

```
<StackPanel.Children>
```

Exemple d'un StackPanel contenant 2 boutons.

```
<StackPanel>
<StackPanel.Children>
<Button>
<Button.Content>
Click Me
</Button.Content>
</Button>
<Button>
<Button.Content>
Re Click Me
```

```
</Button.Content>
</Button>
</StackPanel.Children>
</StackPanel>
```

Cette balise Children peut être omise.

```
<StackPanel>
<Button>
<Button.Content>
Click Me
</Button.Content>
</Button>

<Button>
<Button.Content>
Re Click Me
</Button.Content>
</Button>
</StackPanel>
```

## XI-D-6 - Code 'Behind', événements

L'interface est en XAML et le code (dit code behind) lié aux événements est en VB. Lorsqu'un fichier XAML est compilé, l'emplacement du fichier code-behind pour chaque page XAML est identifié en spécifiant un espace de noms et une classe en tant qu'attributs x:Class de l'élément racine de la page XAML.

Soit la fenêtre Window2, elle correspond à une classe partielle par x:Class="Window2" (ici on a donné seulement un nom de Class, on aurait pu ajouter un espace de nom, un namespace, de la forme MonEspaceDeNom.Window2).

```
<Window x:Class="Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
<Button Name="Button2">
</Button>
</Grid>
</Window>
```

On y ajoute un Bouton.

Si on double-clic sur ce bouton, on se retrouve dans le code behind en VB qui a été créé automatiquement dans le fichier window2.xaml.vb :

```
Partial Public Class Window2

Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
 System.Windows.RoutedEventArgs)
_Handles Button2.Click

End Sub

End Class
```

On note qu'il y a création d'une seconde classe partielle nommée Window2 qui contient le code lié à l'événement Click.

On aurait pu créer soi-même la gestion des événements en nommant la procédure liée au 'Click' du bouton dans le code XAML :

```
<Button Name="Button1" Click="MyButtonClick">
```

Il faudra ensuite créer une Sub MyButtonClick en VB :

```
Partial Public Class Window2

Private Sub MyButtonClik()

End Sub

End Class
```

ou en ajoutant les paramètres :

```
Partial Public Class Window2

Private Sub MyButtonClik(ByVal sender As System.Object, ByVal e As
 System.Windows.RoutedEventArgs)

End Sub

End Class
```

**sender** est l'objet qui a déclenché l'événement.

**e** est de type RoutedEventArgs et contient les arguments de l'événement.

On remarque que pour pallier la disparition des 'groupes de contrôles' de VB6, et pour que le clic sur plusieurs boutons exécute la même Sub, on peut indiquer Click="MyButtonClick" pour plusieurs boutons.

```
<Button Name="Button1" Click="MyButtonClick">
<Button Name="Button2" Click="MyButtonClick">
```

## XI-D-7 - Extension de balisage

On a vu qu'un attribut avait une valeur sous forme de String, mais il peut aussi avoir comme valeur une extension.

Les principales extensions sont les Binding et les ressources.

Les caractères **{** et **}** indiquent une extension de balisage (markup extension).

Exemple d'une extension, ici une ressource (staticResource).

```
<Border Style="{StaticResource PageBackground}">
```

Exemple d'une extension avec ici une liaison de données (Binding).

```
<TextBlock Text="{Binding Title}" />
```

**Préfixe x:**

**x:Key**

Définit une clé unique pour chaque ressource dans un ResourceDictionnary, autrement dit, donne un 'nom' unique à la ressource; pour utiliser cette ressource, on l'appellera par son nom : créons une ressource pour une grid, c'est un LinearGradientBrush que l'on 'nomme' "FunkyBrush" (c'est une clé unique).

```
<Grid.Resources>
    <LinearGradientBrush x:Key="FunkyBrush">
      <GradientStop Color="Yellow" Offset="0" />
      <GradientStop Color="Green" Offset="1" />
    </LinearGradientBrush>
```

```
</Grid.Resources>
```

Ensuite on peut l'utiliser dans la grid :

```
<Button Background="{StaticResource FunkyBrush}">Click Me</Button>
```

Code complet :

```
<Grid>
  <Grid.Resources>
    <LinearGradientBrush x:Key="FunkyBrush">
      <GradientStop Color="Yellow" Offset="0" />
      <GradientStop Color="Green" Offset="1" />
    </LinearGradientBrush>
  </Grid.Resources>
  <Button Background="{StaticResource FunkyBrush}">Click Me</Button>
</Grid>
```

**x:Class**

Définit l'espace de noms et le nom d'une classe, permet de faire le lien avec le code exécutable VB (code-behind).

Pour une fenêtre Windows, correspond au nom de la fenêtre.

```
<Window x:Class="Window1">
```

**x:code**

Permet de mettre dans le code XAML du code VB qui agit sur l'UI.

```
<Button Click="Clicked"  Name="Button1">
<x:Code>
```

```
Sub Clicked(sender As Object,e As RoutedEventArgs )
  button1.Content = "Hello World"
End Sub
```

```
</x:Code>
</Button>
```

## XI-D-8 - Espace de noms

On a vu que dans l'élément racine, on pouvait inclure un espace de noms grâce à 'xmlns:' et ainsi donner accès à des objets extérieurs .

**xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"**

Indique l'espace de noms XAML.

On va maintenant voir comment inclure d'autres espaces de noms dans le code xaml avec quelques exemples.

On peut vouloir inclure un autre espace de noms (WindowsForms par exemple), pour pouvoir utiliser le DataGrid des WindowsForms. C'est un bel exemple de l'intégration d'un contrôle non WPF dans l'interface WPF.

Dans le code Xaml de la Window, il faut importer les espaces de noms correspondant au WindowsForms :

```
xmlns:wfint="clr-namespace:System.Windows.Forms.Integration;assembly=WindowsFormsIntegration"
```

```
xmlns:wf="clr-namespace:System.Windows.Forms;assembly=System.Windows.Forms"
```

Remarquons qu'on a donné un nom, un alias à l'espace de noms 'wf' dans la seconde ligne grâce à 'xmlns:wf'.

Ensuite, toujours en Xaml on utilise une DataGrid :

```
<wfint:WindowsFormsHost Height="100" Width="200">
   <wf:DataGrid x:Name="myDataGridWF"><wf:DataGrid>
<wfint:WindowsFormsHost>
```

Remarquons aussi qu'on utilise la syntaxe 'Espacedenom.Objet' avec 'wf:DataGrid'.

On peut vouloir avoir accès **aux objets qui sont dans l'application**, mais pas dans l'espace xaml.
On a par exemple un module de class écrit en Vb avec une classe nommée 'Names', si on veut l'utiliser dans le code xaml, il va donc falloir inclure l'espace de noms "clr-namespace:NomDuProgramme" en lui donnant un nom d'alias ('local' ici) ; ensuite on pourra utiliser 'local:Names' pour avoir accès à cette classe dans le code xaml (dans l'exemple on l'utilise dans une ressource; pour créer une ressource, on verra cela plus tard) :

```
    <Window x:Class="Window1"
    ……
    xmlns:Local="clr-namespace:NomDuProgramme"
    >


    <Window.Resources>
        <local:Names  x:Key="ListData"/>
    </Window.Resources>
</Window>
```

## XI-D-9 - Remarque importante

Il est important de comprendre que tout le code écrit en XAML peut aussi être écrit en VB.

```
<Button Name="MonBouton">OK</Button>
```

dans la fenêtre xaml crée un bouton ; on peut aussi créer ce bouton dans le code VB :

```
Dim MonBouton As New Button
MonBouton.Content = "OK"
```

## XI-D-10 - Compilaton

Quand on compile une application WPF dans Visual Studio, les fichiers XAML sont compilés en une représentation compressée nommée Binary Application Markup Language (BAML). Ce BAML est ensuite sauvegardé comme une ressource dans l'assembly. Quand cet assembly est chargé et que la ressource est demandée, le BAML est rapidement transformé pour produire les objets décrits par le XAML d'origine.

## XI-E - Grands Principes

**Ou trouvez de la documentation ?**

### Msdn WPF: la bible

(http://msdn.microsoft.com/fr-fr/library/ms754130.aspx)

# XI-E-1 - La Classe 'Controls'

Les contrôles WPF permettent de créer une interface utilisateur WPF.

Les contrôles WPF font partie de l'espace de noms **System.Windows.Controls** (ne pas confondre avec System.Windows.Forms.Control des contrôles Windows Forms habituels).

## XI-E-1-a - Créer un contrôle

Pour mettre un contrôle dans un formulaire (Window), un bouton par exemple, il y a 3 manières.

- En mode 'Design', aller le chercher dans la boite à outils et le déposer sur le formulaire.

- Le créer à l'aide de code Xaml : il faut taper dans la fenêtre XAML :

```
<Button>OK</Button>
```

Cela crée un bouton sur lequel est affiché 'OK'. Il apparait dans le formulaire.

- Le créer dans le code VB :

```
Dim MyButton As New Button
Grid.Children.Add(myButton)
```

On note qu'après avoir déclaré le bouton avec Dim, on l'a ajouté aux enfants de la Grid (dans un formulaire, il y a par défaut une Grid).

Il est possible ensuite de modifier ses propriétés.

Modifions la couleur de l'arrière-plan du bouton par exemple.

- En mode 'Design', dans la fenêtre de propriété en bas à droite.

- À l'aide de code Xaml (ajout d'attribut 'simple') :

```
<Button Background="Blue" >OK</Button>
```

ou de manière plus complexe ajout d'une brush sur le fond :

```
<Button FontSize="14" FontWeight="Bold">
   <Button.Background>
    <LinearGradientBrush StartPoint="0,0.5"
                         EndPoint="1,0.5">
      <GradientStop Color="Green" Offset="0.0" />
      <GradientStop Color="White" Offset="0.9" />
    </LinearGradientBrush>
  </Button.Background>
  OK
</Button>
```

- À l'aide de code VB :

```
MyButton.Background= New SolidColorBrush (Colors.Blue)
```

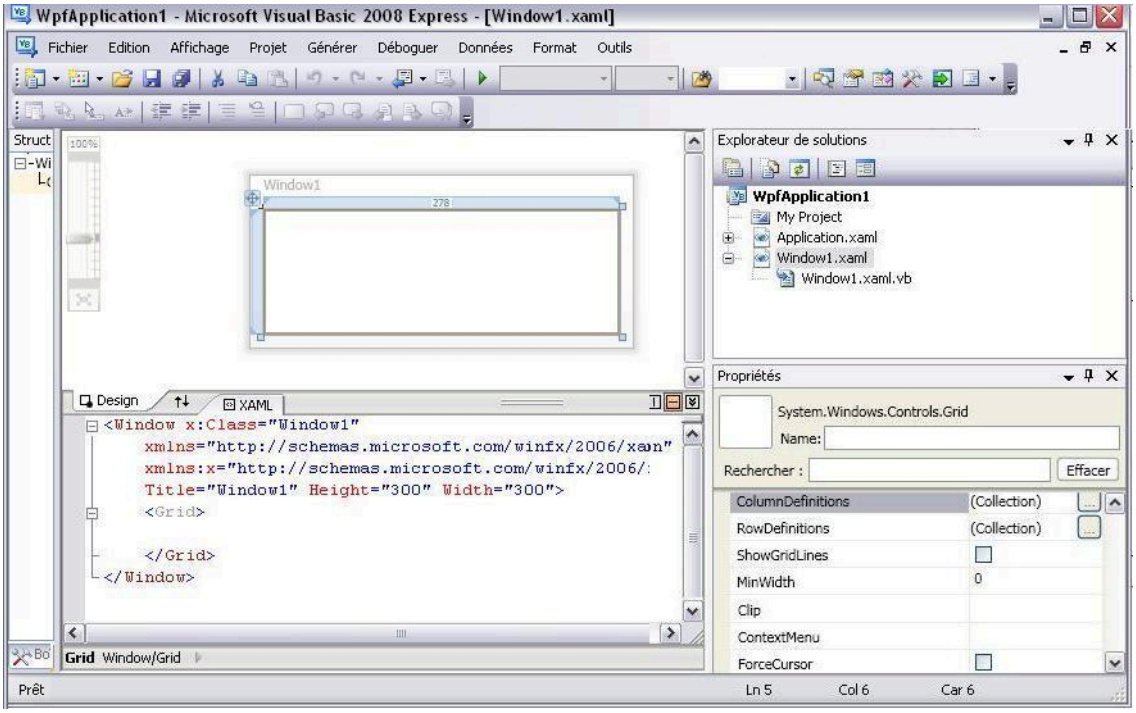
(On crée une nouvelle instance de SolidColorBrush de couleur bleue.)

En résumé

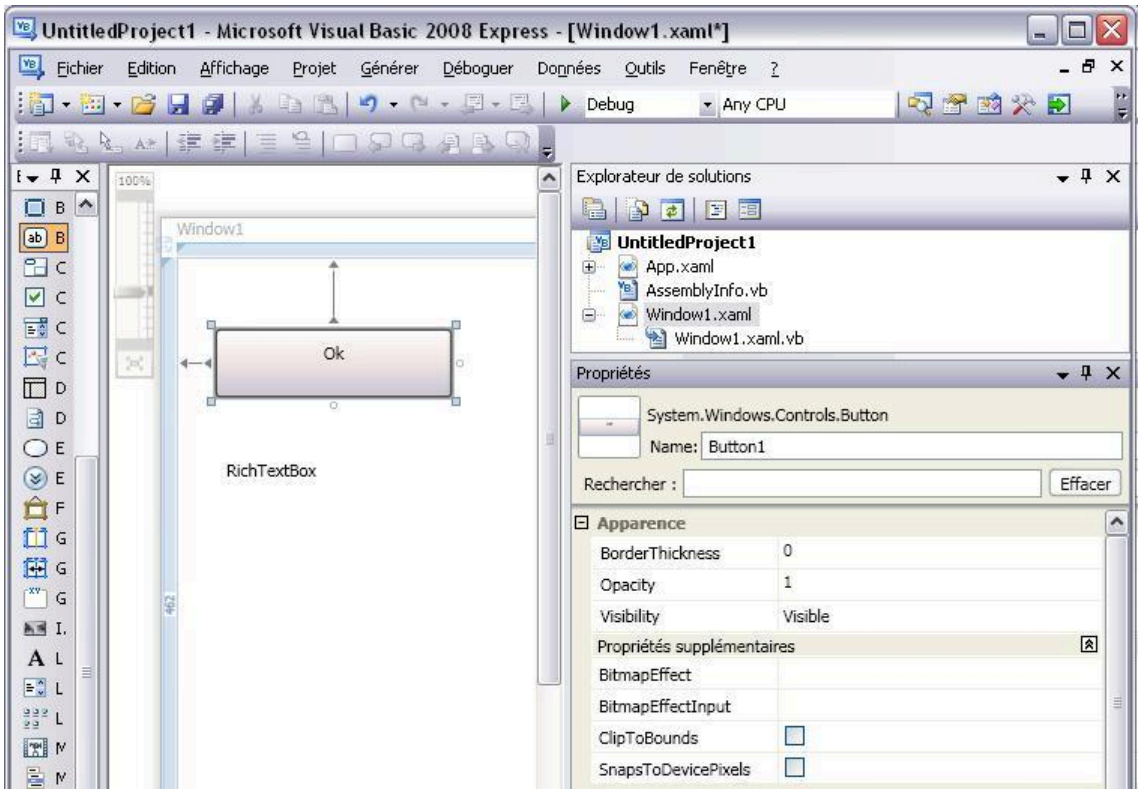
On voit dessous :

- la partie 'Design' où on peut 'dessiner' l'interface en haut ;
- la fenêtre de code XAML en bas ;
- la liste des propriétés en bas à droite.

En double-cliquant sur window1.xaml.vb dans l'explorateur de solution en haut à droite on voit apparaitre le code VB.




Ajoutons notre bouton.




## XI-E-1-b - Particularités des contrôles WPF

### Propriété de dépendances

Les propriétés des contrôles et leur valeur ne sont pas 'figées' (comme dans les Windows Forms où elles ont une valeur donnée), mais elles peuvent 'dépendre' d'autres choses : les propriétés de dépendance permettent de calculer la valeur d'une propriété en fonction de la valeur d'autres entrées. Ces autres entrées peuvent être des propriétés système (des thèmes et des préférences utilisateur), des valeurs déterminées au dernier moment (liaison de données animations), des ressources et des styles ou des valeurs issues de relations parent-enfant d'autres éléments.

### Propriété attachée

Les propriétés attachées permettent à des éléments enfants de spécifier des valeurs pour une propriété définie dans un élément parent. Cela permet de faire en sorte que les éléments enfants informent l'élément parent sur la manière dont ils doivent être présentés dans interface.

La propriété DockPanel.Dock en est un exemple: Dans un Panel, il y a plusieurs enfants, plusieurs boutons par exemple. La propriété DockPanel.Dock de chaque bouton (enfant) permet au Panel (le parent) de positionner les boutons dans le Panel.

### Particularités des contrôles WPF

Un contrôle WPF 'complexe' est composé de contrôles plus simples: Alors que dans les WindowsForms un ListBox est un ListBox, en WPF un ListBox est en fait Un StackPanel (pouvant contenir des TextBlocks, si on affiche du texte, mais aussi des CheckBox ou des Images…) plus un ScrollView. Cela explique que si on applique un style aux TextBlocks en général, ce style sera appliqué à chaque ligne du ListBox.

Enfin si un contrôle WindowsForms avait une propriété Text, le contrôle WPF équivalent a lui une propriété 'Content' qui peut contenir un Objet (du texte oui, mais aussi une Grid, une Image, un StackPanel…), c'est extrêmement puissant.

## XI-E-1-c - événements

Créons un bouton avec le code XAML suivant :

```
<Button>OK</Button>
```

Si je double-clique sur ce bouton dans le Designer, je me retrouve dans le code behind et VB a créé automatiquement la Sub événement suivante :

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
 System.Windows.RoutedEventArgs) _
Handles Button1.Click

End Sub
```

Quand l'utilisateur du logiciel cliquera sur le bouton, c'est cette Sub qui sera exécutée.

Mais si j'ajoute l'attribut **Click="OnClick5"**, cela donne le code XAML suivant :

```
<Button Click="OnClick5">OK</Button>
```

Dans ce cas si je double-clique sur le bouton, la routine suivante est créée :

```
Sub OnClick5(ByVal sender As Object, ByVal e As RoutedEventArgs)

End Sub
```

**sender** est l'objet qui a déclenché l'événement.

**e** contient les arguments de l'événement, remarquons que e est de type RoutedEventArgs (et pas EventArgs comme dans les Windows Forms).

### Événement routé

Quand il survient un événement sur un contrôle, un clic sur un bouton par exemple, cet événement pourra être traité au niveau du bouton, mais aussi par exemple dans un parent comme le Panel qui contient le bouton.

Dans les routines événement, l'argument e est de type **RoutedEventArgs** et non EventArg.

## XI-E-1-d - Principaux contrôles

### Contrôles

- Button : Bouton.
- CheckBox : Case à cocher.
- ComboBox, contenant des ComboItem.
- Image.
- ListBox contenant des ListItem.
- Menu contenant des MenuItem.
- RadioButton : bouton radio.
- ScrollBar : barre de défilement.
- Table contenant : TableCell, TableRow, TableColumn.
- TextBox : champ de texte à taper.
- RichTextBox : champ de texte enrichi.
- Text ou TextBlock : texte.

- VerticalSlider : ascenseur.
- Window : fenêtre.

**Positionnement**

- Canvas : positionnement absolu.
- DockPanel : conteneur qui positionne selon les points cardinaux (ancrage).
- FlowPanel :conteneur pour agencement d'autres éléments.
- Grid : conteneur qui se subdivise en lignes et colonnes.
- Équivalents aux éléments HTML.

**Équivalent Html**

- Block:
- Bold: gras
- Heading: H1, H2, etc.
- HyperLink:
- Image:
- Italic:
- LineBreak:
- List:
- Paragraph:

**Graphisme**

- Canvas: une zone de dessin.
- Ellipse:
- Line:
- Path: , série de lignes connectées.
- Polygon:
- Polyline:, série de lignes droites connectées.
- Rectangle:
- TransformDecorator: transform, rotation, etc.

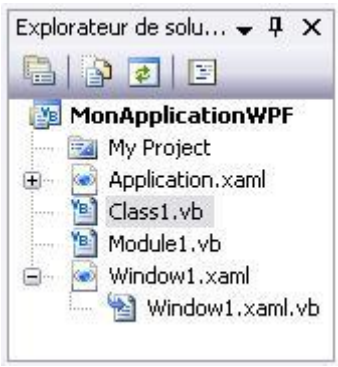
**Nouveaux contrôles VB 2010**

- DataGrid.
- Calendar:
- DataPicker:

## XI-E-2 - Les applications WPF

Créons une application WPF nommée MonApplicationWPF dans VB. Un formulaire Window1 est créé (on y met une grid et un bouton). On ajoute au projet un module et une Classe.

L'explorateur de solution en haut à droite permet de voir de quoi se compose le projet.




**MyProjet** donne accès aux propriétés du projet.

**Application.xaml** est le fichier d'application :

```
<Application x:Class="Application"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
StartupUri="Window1.xaml">
<Application.Resources>

</Application.Resources>
</Application>
```

On peut mettre dans ce fichier les ressources de l'application qui pourront être utilisées dans toute l'application.

Si l'application a été créée dans Blend, on peut voir dans vb que cela diffère un peu :

```
<Application
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
x:Class="App"
StartupUri="Window1.xaml">
<Application.Resources>

<!-- Les ressources réparties au niveau Application sont définies ici. -->

</Application.Resources>
</Application>
```

Il y a aussi : **AssemblyInfo.vb.**

**Class1.vb** et **Module1.vb** qui correspondent à la Classe et au module.

**Window1.xaml** contient le code xaml décrivant le formulaire Window1 avec ici, dans le formulaire, une grid contenant un bouton.

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
x:Class="Window1"
x:Name="Window"
Title="Window1"
Width="640" Height="480">
<Grid>
<Button Name="Button2">Button</Button>
</Grid>
</Window>
```

Dans un formulaire, il y a toujours UN conteneur parent, ici c'est Window.

x:Class="Window1" indique qu'il s'agit d'une classe partielle "Window1"

**Window1.xaml.vb** contient le code vb et les procédures en VB :

```
Partial Class Window1

Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
 System.Windows.RoutedEventArgs)
_Handles Button2.Click

End Sub

End Class
```

Si l'application a été créée dans Blend, on peut voir dans vb que cela diffère un peu :

```
Imports System
Imports System.IO
Imports System.Net
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Data
Imports System.Windows.Media
Imports System.Windows.Media.Animation
Imports System.Windows.Navigation
Imports System.Xml
Public Class Window1
Public Sub New()
MyBase.New()
Me.InitializeComponent()

' Insérez le code requis pour la création d'objet sous ce point.

End Sub

End Class
```

## XI-E-3 - Les formulaires WPF

Dans le designer VB, on a un projet avec une fenêtre 'Window1', on va ajouter une fenêtre nommée 'Window2' au projet en mode designer.

Menu 'Projet', puis 'Ajouter une fenêtre'.


Cliquer sur Fenêtre WPF puis sur le bouton 'Ajouter'.

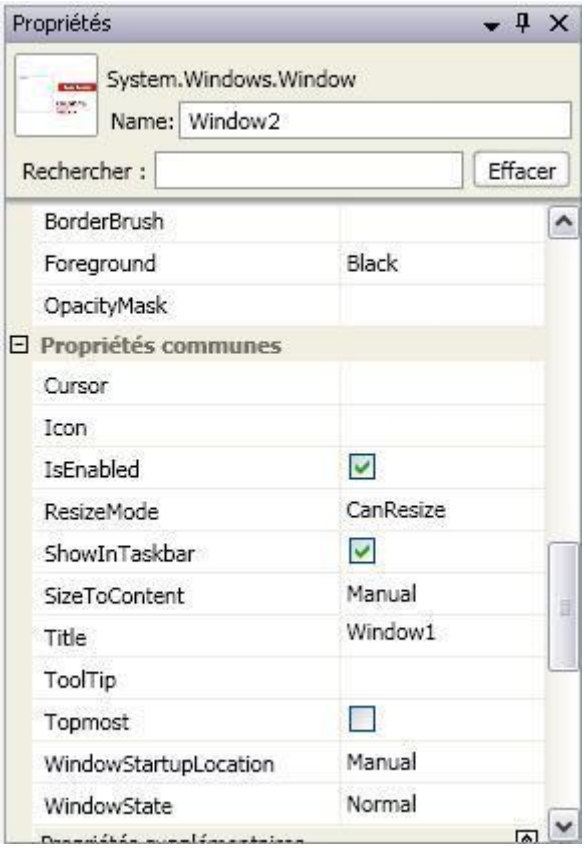
On a créé une seconde fenêtre.


On peut modifier ses dimensions en appuyant le bouton gauche de la souris sur le coin inférieur droit de la fenêtre puis en déplaçant la souris.

On peut ajouter à ce formulaire des contrôles que l'on va chercher dans la boite à outils à gauche.

Remarque : on peut agrandir ou réduire le dessin de la fenêtre avec un curseur de zoom qui est à gauche ; cela prouve bien que le dessin est vectoriel.

On peut modifier ses propriétés dans la fenêtre de propriétés en bas à droite.




**Dans le code vb de la première fenêtre, comment ouvrir la seconde fenêtre Window2 ?**

Window2 qui vient d'être dessiné est en fait la 'Classe Window2'.

Il faut donc instancier un objet formulaire (ou fenêtre) que nous nommerons 'w' à partir de la classe Window2.

```
Dim w As New Window2
```

Ensuite il faut l'afficher :

```
w.Show()
```

ou

```
w.ShowDialog()
```

pour ouvrir la fenêtre Window2 en mode modale.

**Code XAML de ce formulaire, propriétés.**

Pour créer un formulaire, on utilise la Classe System.Windows.Window.

Code XAML correspondant à une fenêtre vide :

```
<Window x:Class="WpfApplication3.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
```

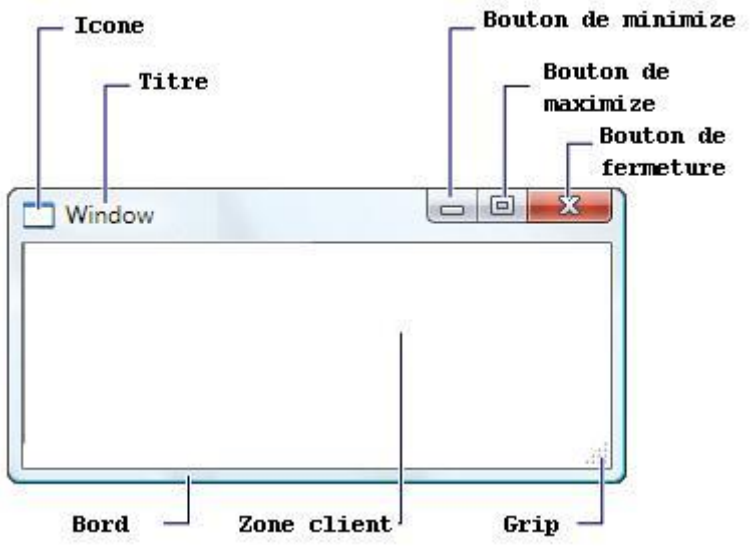
La balise 'Window' indique que c'est un formulaire.

x:Class="WpfApplication3.Window1" indique une classe partielle nommée Window1 dans l'espace de noms WpfApplication3 (qui est le nom de l'application). Cela permet au formulaire d'être 'relié' au code VB behind qui est dans une classe partielle de même nom.

Puis on a les 2 namespaces principaux (le premier pour les composants de base et le second pour les propriétés et extensions).

On peut ajouter des propriétés directement dans le code XAML :

```
<Window x:Class="WpfApplication3.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Height="355" Width="300"
    Title="Window"
    ResizeMode="NoResize"
    SizeToContent="Height"
    WindowStartupLocation="CenterScreen"
    WindowStyle="None"
    ShowInTaskbar="False"
    Foreground="Navy"
    AllowsTransparency="True"
    Opacity="0.75">
```




Détaillons.

**Height="355" Width="300"**

Indiquent les dimensions de la fenêtre.

**Title="Window"**

Indique le titre dans la barre de titre.

**Foreground="Navy"**

Indique la couleur du fond.

**ResizeMode= "NoResize"**

"NoResize" indique: pas de bouton minimize ni maximize ni bord modifiable. "CanResize" c'est l'inverse :bouton minimize et maximize bord modifiable. "CanMinimize" n'affiche que le bouton minimize. "CanResizeWithGrip" ajoute un petit indicateur de redimensionnement (un grip) qui apparait en bas à droite de la fenêtre.

**SizeToContent="Height"**

Si "Manual", WPF utilise les propriétés "Height" et "Width" pour fixer la taille de la fenêtre. Si "Height" WPF calculera la hauteur en fonction du contenu. Si "Width" WPF calculera la largeur en fonction du contenu.

**WindowStartupLocation="CenterScreen"**

Position de départ de la fenêtre."CenterScreen" positionne au centre de l'écran.

Si "Manual" c'est les propriétés Left et Top qui sont utilisées pour positionner la fenêtre. Si "CenterOwner" c'est centré sur la fenêtre appelante.

**WindowStyle="None"**

"SingleBorderWindow" correspond à une bordure simple. "ThreeDBorderWindow" bordure est légèrement plus accentuée. "ToolWindow" bordure utilisée pour les boites à outils (petite bordure avec une petite barre de titre). "None" pas de bordure et pas de barre de titre. Si vous avez autorisé le redimensionnement (par "ResizeMode") il y aura quand même une petite bordure pour permettre le redimensionnement qui reste possible.

**Topmost="None"**

Si "True" la fenêtre sera toujours au-dessus de toutes les autres (jamais masquée).

**ShowInTaskbar="False"**

Si "False" nous désactivons la fenêtre dans la barre des tâches. Celle-ci ne sera donc pas visible à côté du menu démarrer.

**AllowsTransparency="True"**

**Opacity="0.75"**

Pour la transparence des fenêtres. Pour rendre totalement transparentes vos fenêtres vous devez activer la propriété "AllowsTransparency" à True et "WindowStyle" à None. Une fois la transparence activée, vous verrez que la propriété "Opacity" modifie la transparence : un nombre entre 0 (transparent) et 1 (totalement opaque).

**Code VB, propriétés, événements**

Comment ouvrir par code VB une seconde fenêtre qui a été créée en mode design (class Window2) ?

```
Dim w As New Window2
w.Show()
```

ou

```
w.ShowDialog()
```

pour ouvrir une fenêtre modale.

La fenêtre qui s'ouvre est activée en même temps.

Si on ajoute

```
w.ShowActivated = False
```

avant d'utiliser Show, la fenêtre qui s'ouvre n'est pas activée.

Si on voulait créer une fenêtre de toute pièce avec du code VB (sans l'avoir dessinée en mode design avant) :

```
Dim w As New Window
w.Show()
```

On rappelle que les propriétés de la fenêtre seront accessibles là où la fenêtre est visible en fonction de sa portée.

Si la fenêtre est en arrière-plan, pour l'activer et la mettre au premier plan :

```
w.Activate
```

Une fenêtre activée reçoit les événements de la souris et du clavier.

Pour la fermer :

```
w.Close
```

**On peut modifier toutes les propriétés de la fenêtre par code VB :**

```
Me.Background = New SolidColorBrush(Colors.Red)
```

couleur du fond en rouge

```
Me.Title = "Ma fenetre"
```

texte de la barre de titre

```
Me.Height = "200"
```

hauteur de la fenêtre.

On remarque que dans la fenêtre on peut utiliser **Me** pour désigner celle-ci.

On peut utiliser **BorderThickness** pour l'épaisseur des bords ; **MaxHeight, MinHeight, MaxWidth, MinWidth** pour limiter la réduction et l'agrandissement de la fenêtre par l'utilisateur.

**SizeToContent** permet d'ajuster la fenêtre à la taille de son contenu, il peut prendre la valeur :

- Manual (pas d'effet) ;
- Width ajustement de la largeur au contenu ;
- Height ajustement de la hauteur au contenu ;
- WidthAndHeight ajustement de la largeur et de la hauteur au contenu.

**Quand la fenêtre s'ouvre surviennent les événements suivants :**

**Initialized** Survient avant le chargement. À utiliser si on n'a pas besoin d'avoir accès aux éléments visuels qui ne sont par encore en place.

```
Private Sub Window1_Initialized(ByVal sender As Object, ByVal e As
 System.EventArgs) Handles Me.Initialized

End Sub
```

**Loaded** Fenêtre chargée, mais non visible, permet de modifier l'IU, les boutons…

```
Private Sub Window1_Loaded(ByVal sender As System.Object, ByVal e As
 System.Windows.RoutedEventArgs)
_Handles MyBase.Loaded

End Sub
```

**ContentRendered**

**SourceInitialised**

*= À CE MOMENT, LE FORMULAIRE EST AFFICHÉ=*

**Activated** se produit lorsque la fenêtre est activée (l'utilisateur a cliqué dessus ou on a utilisé la méthode Window1.Activate).

**Desactivated** lorsque la fenêtre passe en arrière-plan.

**Quand une fenêtre est fermée, les événements suivants se déclenchent :**

**Closing** avant la fermeture, on peut annuler la fermeture de la fenêtre en donnant à l'argument e de type CancelEventArgs la valeur True.

```
Private Sub Window1_Closing(ByVal sender As Object, ByVal e As
 System.ComponentModel.CancelEventArgs)
_Handles Window1.Closing
    e.Cancel = True
End Sub
```

**Closed** quand la fenêtre se ferme ;

**Unloaded :** déchargement de la fenêtre.

(Microsoft indique que Activated survient avant loaded et ContentRendered dans une ouverture initiale de fenêtre, mais je n'ai pas vu cela !!!).

**GotFocus** et **LostFocus** surviennent lors de la prise ou de la perte du focus, il faut éviter de les utiliser, car en pratique le fonctionnement est curieux, Activated et Deactivated semblent plus judicieux ;

**SizeChanged** lorsque l'utilisateur modifie la taille de la fenêtre.

Remarque

VB met automatiquement dans les formulaires une 'Grid' qui servira de conteneur pour tous les autres contrôles.

**Formulaire MDI**

Les formulaires MDI sont absents en WPF. Il ne faut plus les utiliser !! À la place, on utilise les onglets (TabControl).

# XI-E-4 - Positionnement, dimensions des contrôles

Le positionnement des contrôles est fondamental en WPF, il faut bien le comprendre.

**Comparaisons WindowsForms-WPF**

En Windows Forms on pouvait mettre autant de contrôles que l'on voulait dans une form (fenêtre) et on utilisait simplement les coordonnées du coin supérieur gauche d'un contrôle pour définir sa position. En Windows Forms, en cas de changement d'écran ou de modification des dimensions d'une fenêtre, si on voulait un reagencement, il fallait tout gérer soi-même.

En WPF il faut utiliser la notion de conteneur et de position dans ces conteneurs. Le positionnement est relatif. Il y a capacité d'adaptation aux modifications d'affichage et de disposition des fenêtres. WPF gère la négociation entre les contrôles pour déterminer la disposition. Un contrôle informe son parent de l'emplacement et de la taille dont il a besoin ; deuxièmement, le parent informe le contrôle de l'espace dont il peut disposer. Cette disposition dynamique fait qu'un contrôle est positionné sur un formulaire en fonction de son contenu, de son conteneur de disposition parent et de la dimension d'écran disponible. Cela facilite la localisation en ajustant automatiquement la taille et la position des éléments lorsque la longueur des chaines qu'ils contiennent est modifiée.

La disposition WPF est basée sur les grilles (Grid), l'empilement (Stack) et l'ancrage (Dock).

## XI-E-4-a - Hiérarchie des contrôles

**Contrôles conteneurs et non-conteneurs**

Certains contrôles (dits 'non-conteneur') ne peuvent contenir qu'un contrôle. Essayer de mettre 2 boutons dans une fenêtre vide avec le designer ou une image et un texte dans un bouton : impossible !!

Certains contrôles (dit 'conteneur') peuvent contenir plusieurs contrôles. Une Grid, par exemple qui peut contenir un élément dans chacune de ses cellules ; un StackPanel peut contenir en empilement de contrôles.

Une fenêtre (Window) ou un bouton (dit 'non-conteneur'), ont une propriété nommée 'Content' qui ne peut donc contenir d'un seul élément.

Pour un bouton par exemple, en VB :

```
    MyButton.Content = "OK"
```

affiche le texte 'OK' dans le bouton.

En XAML :

```
<Button Content="OK"/>
```

**Content**

Dans un contrôle non-conteneur, la propriété 'Content' ne peut contenir d'un objet et un seul.
Pour un bouton par exemple, on a 2 manières d'écrire le texte du bouton :

```
<Button Content="OK"/>
```

ou

```
<Button> OK </Button>
```

Car Content est la propriété par défaut.

Content peut contenir un texte (comme en Windows Forms), mais pas seulement ; en fait il peut contenir un Objet (un seul) : du texte comme dans l'exemple précédent, mais aussi un objet conteneur comme une Grid, un Canvas, un StackPanel, un DockPanel… qui lui peut contenir plusieurs contrôles. Cela revient à mettre autant de contrôles que l'on veut dans un contrôle.

C'est pour cela que quand on crée une fenêtre en VB, il est automatiquement ajouté une grid.

Les conteneurs comme une Grid par exemple, ont une propriété nommée Children qui permet d'ajouter plusieurs contrôles.

En VB :

```
MyGrid.Children.Add (Button1)
```

À titre d'exemple, on peut mettre une Grid dans un bouton et mettre dans les cellules de cette Grid 2 textes, une image, une vidéo…

Voyons maintenant un exemple en xaml d'un conteneur, un StackPanel contenant un TextBlock et 3 boutons :

```
<StackPanel >
<TextBlock>Faire un choix:</TextBlock>
<Button >Option 1</Button>
<Button >Option 2</Button>
<Button >Option 3</Button>
</StackPanel>
```

**Modèle de contenu**

Pour être complet, pour tous les objets visuels WPF, il y a 4 modèles de contenu :

*-ContentControl*

Sa propriété Content contient un objet. Exemple Window, Button… ;

*-HeaderContentControl*

Sa propriété Content contient un objet. Sa propriété Header fournit un titre au contrôle. Exemple GroupBox, TabItem ;

*-ItemsControl*

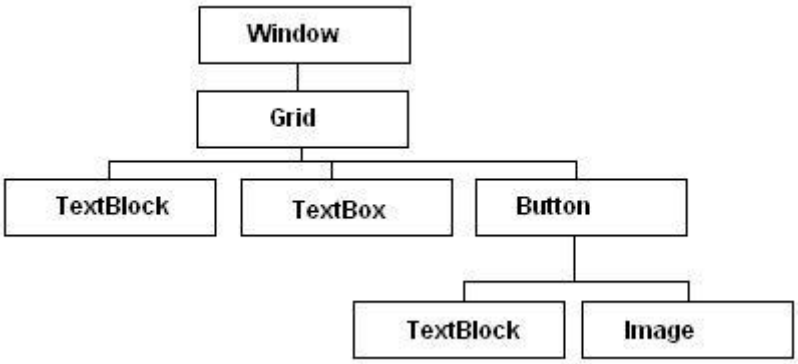
Sa propriété Items contient une collection d'objets de même type. Exemple ListBox contient des ListBoxItem, il y a les objets conteneurs 'Panel' (comme les Grid, StackPanel…) qui contiennent une collection d'objets (Children).

*-HeaderItemsControl*

Sa propriété Header fournit un titre au contrôle. Sa propriété Items contient une collection d'objets. Exemple MenuItem, ToolBar, TreeViewItem.

**Graphes d'objets**

Ainsi il y a des objets dans d'autres et donc une hiérarchie des contrôles qui peut être représentée par un graphe.




Cela a son importance, car on verra qu'un événement (un clic par exemple) qui survient sur un contrôle (l'image en bas à droite) peut être routé, remonter et être exploité plus haut (au niveau de la grille par exemple).

Remarquons aussi qu'un contrôle WPF 'complexe' est composé de contrôles plus simples : alors que dans les WindowsForms un ListBox est un ListBox, en WPF un ListBox est en fait Un StackPanel (pouvant contenir des TextBlocks, si on affiche du texte, mais aussi des CheckBox ou des Images…) plus un ScrollView. Cela explique que si on applique un style aux TextBlocks en général, ce style sera appliqué à chaque ligne du ListBox. Là aussi, un contrôle complexe, on peut être vu comme un graphe hiérarchisé de contrôles simples.

## XI-E-4-b - Position et dimensions d'un contrôle

En WPF, les dimensions sont en "device independent pixels".

Un pixel indépendant du périphérique ou **DIP** (Device Independent Pixel) mesure 1/96 d'un pouce et est indépendant de la résolution du périphérique d'affichage ou d'impression.

Noter que les valeurs des dimensions et coordonnées, en WPF, sont des 'Double' (nombre réel en virgule flottante double précision).

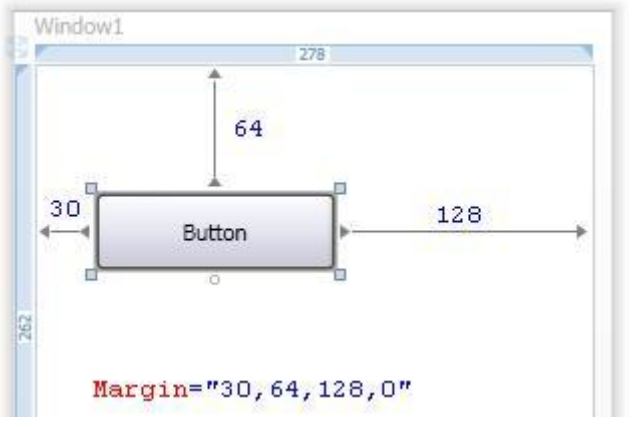
Dans ce chapitre, on voit le positionnement d'un contrôle dans un conteneur, pour être complet, il faut aussi voir le chapitre sur les conteneurs. Dans une grille par exemple, dans chaque cellule on pourra positionner un contrôle comme indiqué ci-dessous.

**Width="50" Height="30"** indiquent les dimensions d'un objet (largeur, hauteur).

On peut aussi leur donner la valeur "Auto" afin que le contrôle occupe la totalité de la largeur ou de la hauteur restante.

Exemple 1

Mettre un bouton dans un conteneur (la cellule d'une grille par exemple), on le positionne dans le Designer avec la souris.




Cela donne par exemple dans le code XAML :

```
<Button Height="39" Margin="30,64,128,0" Name="Button1" VerticalAlignment="Top"
 Click="OnClick5">OK</Button>
```

Voyons le détail de cette ligne :

```
<Button>OK</Button>
```

crée un bouton.

**Name="Button1"** indique le nom du contrôle dans le designer.

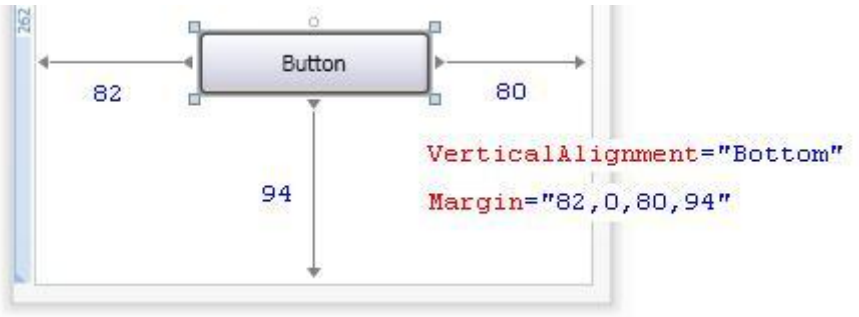
On peut indiquer sur quoi le contrôle est aligné :

**VerticalAlignment="Top"** bouton aligné sur le bord supérieur. Si la fenêtre est redimensionnée, le bouton sera toujours aligné et à égale distance du bord supérieur ;

**Margin="30,64,128,0"** indique les marges gauche, supérieure, droite, inférieure.

Exemple 2

Si **VerticalAlignment="Bottom"** voici un exemple de margin.




**Les grands principes**

**VerticalAlignement** et **HorizontalAlignment** définissent sur quoi est aligné le contrôle. Le contrôle y sera 'ancré', si on modifie les dimensions de la fenêtre, le contrôle suivra.

(Cela remplace la propriété 'Anchor' des Windows Forms.)

Rappelons que si on modifie la taille de la fenêtre, le contrôle reste à égale distance du bord sur lequel il est aligné (ancré).

Valeurs possibles pour VerticalAlignement : Top, Bottom, Center, Stretch (étire le contrôle et remplit verticalement).

Valeurs possibles pour HorizontalAlignment : Left, Right, Center, Stretch(étire le contrôle et remplit horizontalement)…

Attention, si on renseigne Width et Height Stretch ne fonctionne pas !!

**Margin** définit les distances au conteneur.

**Margin="30,64,128,0"** indique les marges gauche, supérieure, droite, inférieure.

**Margin="30"** indique que toutes les marges sont à 30.

**Paddding** définit l'épaisseur d'une marge dans le contrôle (valable dans certains contrôles seulement).

**Largeur de bouton et texte**

Si on donne une valeur à Width=50, la largeur est imposée.

Si on spécifie les Margin=20,20,20,20 et qu'on laisse Width= Auto, les marges imposent les dimensions du bouton.

Si on spécifie les Margin=20,20,0,20 (marge de droite à 0) et qu'on a Width= Auto, le bouton se dimensionne pour afficher le texte qu'il contient.

**Priorités des propriétés**

Dans les dimensions horizontales par exemple

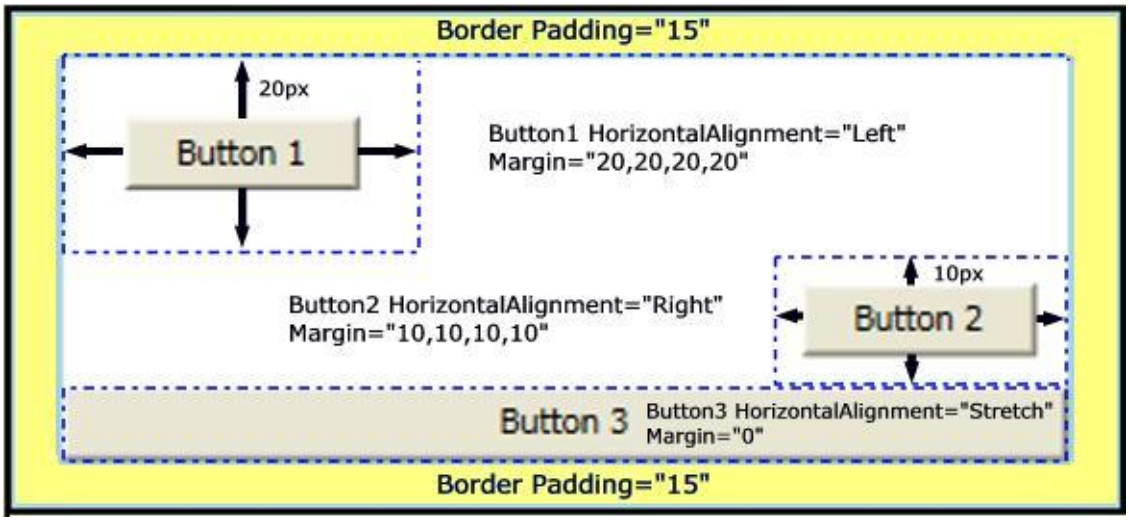
La propriété Width, si celle-ci à une valeur, est utilisée pour la largeur du composant, même si HorizontalAlignment="Strech".

Si Width = "Auto" ou rien, WPF regarde la valeur de HorizontalAlignment.

Si HorizontalAlignment= "Strech", la largeur sera tout l'espace disponible.

Sinon, la propriété Width étant sur "Auto", la largeur s'adaptera au contenu.

Pour résumer, voici un conteneur avec trois boutons :




**Comment bien positionner des boutons ?**

On met 3 boutons dans un StackPanel (conteneur qui met les boutons les uns au-dessus des autres).

On sélectionne les 3 boutons (Ctrl+Clic sur les 3 boutons), dans la fenêtre de propriété, on met :

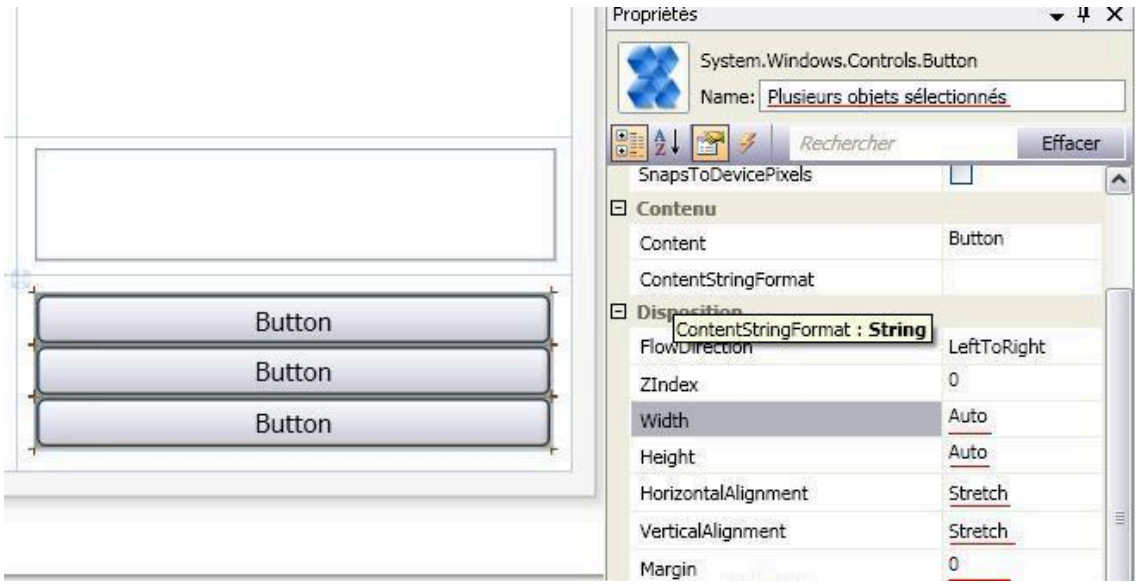
Width=Auto ;

Height=Auto ;

VerticalAlignment=Strech ;

HorizontalAlignment=Strech.

Cela met les 3 boutons à la même dimension et ils remplissent la totalité du StackPanel :




Et en VB cette fois :

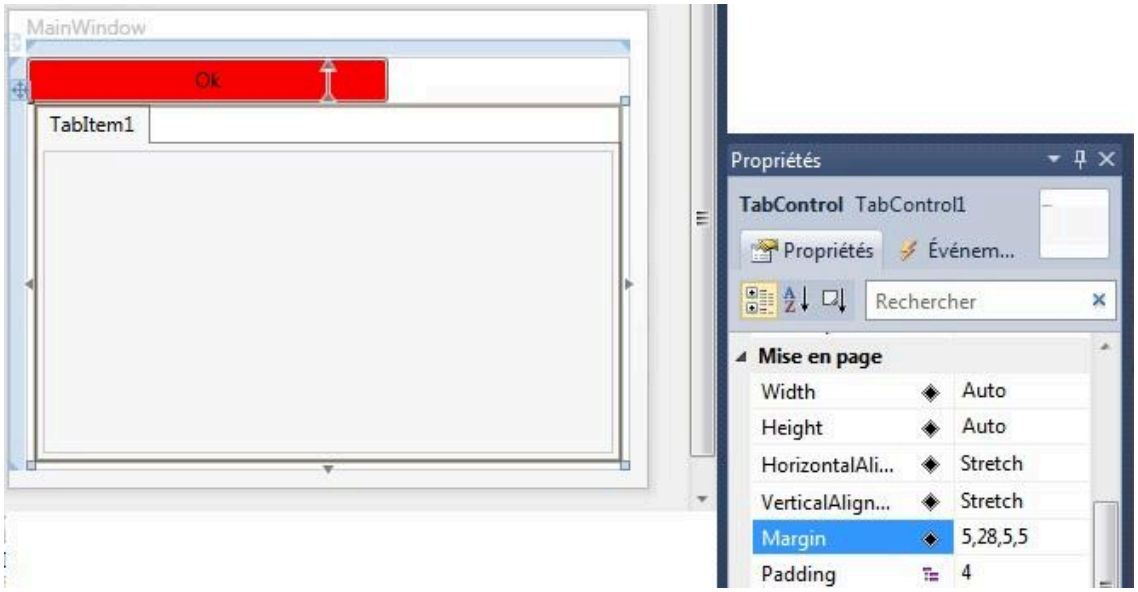
```
Dim myButton As New Button() 'création d'un bouton
myButton.Height = "33"       'modification des dimensions du bouton
```

```
myButton.Width = "55"
myButton.HorizontalAlignment = Windows.HorizontalAlignment.Stretch
myButton.Content = "Button  Stretch"
Grid1.Children.Add(myButton) 'met le bouton dans le conteneur grid
```

Autre exemple :

```
Dim myButton As New Button()
myButton.Margin = New Thickness(10, 20,0, 20)
myButton.Padding= New Thickness(10)
myButton.HorizontalAlignment = Windows.HorizontalAlignment.Center
GrosBouton.Content = myButton 'on met myButtom dans un autre gros bouton, cela ne sert à rien,
'c'est juste pour illustrer la propriété Content.
```

Dernier exemple
Positionnons un TabControl dans une fenêtre ; il doit remplir la fenêtre, mais laisser voir les boutons en haut. HorizontalAlignement et VerticalAlignement auront la valeur Stretch pour remplir, Height et Width = Auto pour ne pas forcer une dimension. Pour laisser voir les boutons en haut, on joue sur la Margin supérieure.




On vient de voir comment sont positionnés les contrôles dans un conteneur.

On verra dans le chapitre sur les contrôles conteneurs **les différents conteneurs**.

- les Grid ;
- les Panels ;
- les DockPanel ;
- les StackPanel ;
- …

## XI-E-5 - Aspect des contrôles

Quand on dépose un bouton dans un formulaire, il a un aspect 'standard', on a besoin souvent de modifier sa couleur ou sa forme, son aspect visuel ; c'est ce que nous allons voir. Dans ce chapitre on travaille sur UN contrôle. Dans le chapitre sur les ressources, on verra comment créer des modèles de contrôles (avec les Styles et les Templates) pour les appliquer à tous les contrôles.

## XI-E-5-a - Propriétés des contrôles

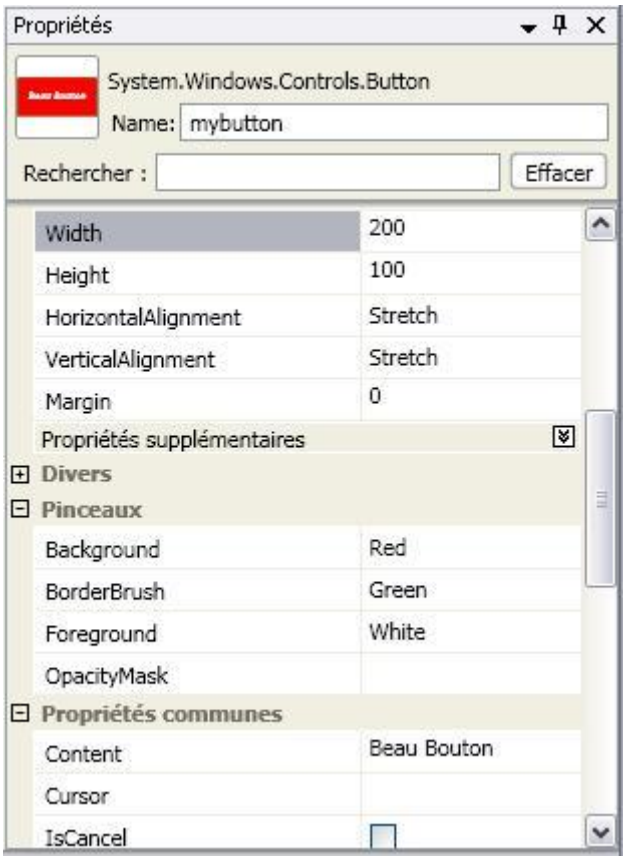
Ici on garde l'aspect général du contrôle, on modifie simplement ses propriétés (Couleur, texte, font…).

On prendra l'exemple d'un bouton:

On veut avoir ce beau bouton (bof !).


Avec le designer, on prend un bouton dans la boite à outils, on le pose dans un conteneur (une grid par exemple).

Pour définir les dimensions, la couleur, le texte dans le bouton, on va modifier les propriétés dans la fenêtre de propriétés en bas à droite :




- **Height** et **Width** permettent de modifier les dimensions.
- **Foreground** et **Background** donnent la couleur du texte et du fond.
- **BorderBrush** et **BorderThickness** indiquent la couleur et l'épaisseur des bords
- **FontFamily** indique le nom de la font.
- **FontSize** indique la hauteur de la font.
- **FontStyle** permet de choisir entre Normal, Italic, Oblique.
- **FontWeight** permet de choisir entre Normal, Bold, Heavy…
- **Name** donne le nom du bouton en haut de la fenêtre.

- **Content** indique le texte à afficher.

En bas de l'écran dans la fenêtre XAML, apparait le code xaml correspondant au bouton.

```
<Button
Height=" 100" Width="200"
Foreground="White" Background="Red"
BorderBrush="Green" BorderThickness="100"
FontFamily="Algerian" FontSize="24" FontStyle="Italic" FontWeight="Bold" Name="mybutton">Beau
 Bouton</Button>
```

Dans une fenêtre vide, on aurait pu coller le code xaml entre les balises grid et voir apparaitre le bouton.

On aurait pu créer le bouton avec du code Visual Basic :

```
Dim myButton As New Button() 'on crée le bouton nommé 'MyButton'
myButton.Height = "100"      'modification des dimensions du bouton
myButton.Width = "200"
myButton.Background = New SolidColorBrush(Colors.Red)
myButton.Foreground = New SolidColorBrush(Colors.White)
myButton.BorderBrush = New SolidColorBrush(Colors.Green)
myButton.BorderThickness = New Thickness(100)
myButton.FontFamily = New FontFamily("Algerian")
myButton.FontSize = "24"
myButton.FontStyle = FontStyles.Italic
myButton.FontWeight = FontWeights.Bold
myButton.Content = "Beau Bouton"
Grid.Children.Add(myButton) 'on met le bouton dans le conteneur grid
```

On a à notre disposition une énumération Colors qui énumère les couleurs.

Mais en vb, on ne peut pas affecter directement la couleur : myButton.Background = Colors.Red ne fonctionne pas.

Il faut utiliser une brush :

```
myButton.Background = New SolidColorBrush(Colors.Red)
```

On a aussi une énumération de SolidColorBrush que l'on peut utiliser comme cela :

```
myButton.Background = Brushes.Red
```

On peut 'enrichir' une partie du texte.

On utilise dans ce cas des balises comme 'bold' ou 'italic' par exemple dans un TextBlock :

```
<TextBlock Width="150" Height="50"
TextAlignment="Center" > <Bold>Un</Bold> <Italic>bouton</Italic> </TextBlock>
```

Au lieu de mettre une couleur unie dans le fond d'un contrôle, on peut y mettre une 'Brush' ce qui produit un bel aspect :


En XAML :

```
<Button  Content="OK">
<Button.Background>
<LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
```

```
<GradientStop Color="White" Offset="0" />
<GradientStop Color="LightGray" Offset="1" />
</LinearGradientBrush>
</Button.Background>
</Button>
```

On peut aussi modifier les propriétés d'un contrôle, ou d'un type de contrôle (tous les boutons de l'application) à l'aide des **Ressources** des **Styles**; voir le chapitre 'Ressources' Enfin on peut utiliser les **templates** et **contrôles templates** (voir plus bas).

## XI-E-5-b - Contrôle contenant des contrôles

On a vu que la propriété Content d'un bouton pouvait contenir un objet, mais un seul. On peut y mettre du texte, mais comment mettre un texte et une image dans un bouton ?

Il faut mettre un StackPanel (ou une Grid) dans le bouton (puisque celui-ci ne peut contenir qu'un seul objet), dans ce StackPanel (qui lui est un conteneur qui peut contenir x contrôles empilés) mettre un TextBlock et une Image. Le faire en tapant du code XAML (dans le designer VB c'est difficile de mettre un StackPanel dans un Button, il se met dessus et pas dedans, donc copier-coller le code XAML). De plus l'image doit être dans les ressources : voir ce chapitre.

```
<Button  Name="Button1">
            <StackPanel   Name="StackPanel">
                <TextBlock TextAlignment="Center">
                 OK
                </TextBlock>
                <Image Source="oktransp.GIF" Height="25.704" Width="127.092">
                </Image>
            </StackPanel>
            </Button>
```

Cela donne:


On se rend compte de la puissance des WPF : on peut mettre autant de contrôles que l'on veut dans un contrôle en utilisant des conteneurs. Dans notre bouton on aurait pu mettre 2 images, 2 textes…

## XI-E-5-c - Aspect visuel des contrôles : Template visuel, Style

Ici on modifie l'aspect profond des contrôles. Débutant s'abstenir.

Chaque contrôle possède un **template visuel** ou **Control Template** qui permet de définir l'aspect visuel du contrôle,le template indique comment il sera dessiné (forme, coins…).

Ce template est entre les balises 'Button.Template' pour un bouton. On peut bien sûr modifier ce Template ce qui modifie l'aspect du bouton : on peut ainsi obtenir des boutons ronds, elliptiques…

Dans le template du bouton, ici, on va définir la forme qui est rectangle, mais aussi la forme des coins (radius).

On va donc faire un bouton rouge avec des coins ronds (grâce à Rectangle RadiusX="9" RadiusY="9").

```
<Button Name="Button2" >
                <Button.Template>
                    <ControlTemplate>
```

```
                        <Grid>
                            <Rectangle RadiusX="9" RadiusY="9" Fill= "Red">
                            </Rectangle>
                            <ContentPresenter HorizontalAlignment="Center"
                             VerticalAlignment="Center" Content="BoutonTemplate"/>
                        </Grid>
                    </ControlTemplate>
                </Button.Template>
            </Button>
```

Cela donne:


On voit qu'à partir du moment ou on utilise le ControlTemplate, il faut tout refaire, la forme du contrôle (Rectangle), mais aussi le contenu grâce à ContentPresenteur.

Si au lieu de mettre 'Rectangle Radius…' on met 'Ellipse Fill="Green"', le bouton est vert en forme d'ellipse.

Cet exemple fonctionne, mais est incomplet, car comme dans le Template, on n'a pas défini l'aspect du bouton en cas de clic ou quand il a le focus, le bouton ne change donc jamais d'aspect même quand on clique dessus !!

On pourrait créer un **Style** pour un bouton :

```
<Button  Height="51"  Margin="74,35,0,0" Name="Button1"  Width="213" >
           OK
            <Button.Style>
                <Style>
                    <Setter Property="Button.Background" Value="Red"/>
                </Style>
            </Button.Style>
        </Button>
```

En fait c'est beaucoup plus simple d'utiliser une propriété (les Styles sont plutôt utilisés dans les ressources) :

```
<Button
Height=" 100" Width="200"
Background="Red"
Name="mybutton">Beau Bouton</Button>
```

Ici on a modifié UN contrôle, il est possible de modifier TOUS les contrôles de même type. Voir le chapitre sur les ressources qui parle des modèles de contrôles et donne un exemple complet. Enfin pour être complet il est possible de modifier l'aspect des données dans un contrôle grâce aux Data Template (modèle de données).

## XI-E-5-d - Modification du Bitmap d'un contrôle

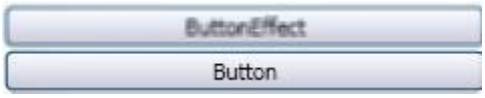
Ici on modifie le BitMap (les pixels) du dessin du contrôle en appliquant des effets spéciaux. Débutant s'abstenir.

BlurBitmapEffect permet d'appliquer un flou comme à travers un objectif non mis au point.

Appliquons cet effet à un bouton.

```
<Button  Name="Button1" Width="Auto">ButtonEffect
<Button.BitmapEffect>
        <BlurBitmapEffect Radius="1" KernelType="Box" />
</Button.BitmapEffect>
</Button>
```

Voici ce que cela donne sur le bouton supérieur (notez que l'aspect flou n'apparait pas dans le designer, mais uniquement lors de l'exécution).




Il existe d'autres effets.

**OuterGlowBitmapEffect** crée un halo de couleur autour d'un objet.

**DropShadowBitmapEffect** crée une ombre derrière un objet.

**BevelBitmapEffect** crée un biseau qui déclenche la surface d'une image d'après une courbe spécifiée.

**EmbossBitmapEffect** crée un placage de relief pour un Visual.

## XI-E-6 - Remplissage de surface

On peut 'remplir' le fond d'un bouton, un rectangle, le même du texte…

Pour cela, on utilise des **Brush**.

### XI-E-6-a - SolidColorBrush

C'est simple, c'est une couleur unie.

Exemple 1. On veut choisir la couleur du Background d'un bouton.

On peut le définir en mode design dans la fenêtre de propriétés.

En XAML :

```
<Button Background="Red"/>
```

En VB :

```
myButton.Background = New SolidColorBrush(Colors.Red)
```

Comme en winforms, on ne peut pas affecter directement la couleur : (myButton.Background = Colors.Red ne fonctionne pas) on est obligé d'instancier une nouvelle SolodColorBrush et de lui donner la valeur Red qui appartient à la collection Colors.

Exemple 2 : créer un rectangle rouge :

```
Dim Rect As new Rectangle()
Rect.Width = 75
Rect.Height = 75
'Creation d'une SolidColorBrush
Dim  myBrush As  new SolidColorBrush(Colors.Red)
Rect.Fill = myBrush
Grid.Content =Rect
```

Remarquer qu'on a utilisé la méthode **Fill** pour remplir le rectangle avec la Brush.

En XAML :

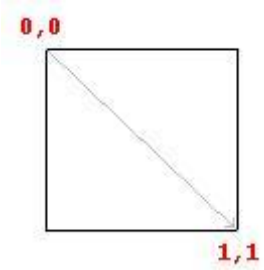
```
<Rectangle Width="75" Height="75">
<Rectangle.Fill>
<SolidColorBrush Color="Red" />
</Rectangle.Fill>
</Rectangle>
```

## XI-E-6-b - LinearGradientBrush

C'est une couleur qui se transforme progressivement en une autre puis éventuellement en une autre encore.

Il y a un system de coordonnées sur la surface à remplir : (0,0) est au coin supérieur gauche, (1,1) au coin inférieur droit.

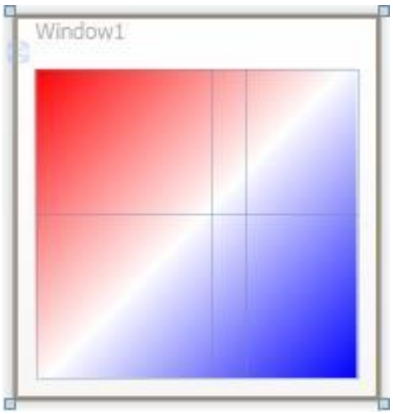
'0.5, 0.5 ' correspond au centre.




StartPoint indique les coordonnées du début du gradient, EndPoint les coordonnées de la fin du gradient, GradientStop indique la position relative de la couleur sur la ligne qui rejoint le point de début au point de la fin.

Exemple 1 :sur une grid, on positionne 3 couleurs dans la diagonale.

En VB, on ne peut pas le faire dans la fenêtre designer, le plus simple est de coller le code XAML dans la fenêtre XAML.




Pour suivre la diagonale StartPoint="0,0" EndPoint="1,1".

Le rouge sera à 0 % de la diagonale, le blanc à 50 % le bleu à 100 % de la diagonale.

En XAML :

```
<Grid >
<Grid.Background>
```

```
<LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
<GradientStop Color="Red" Offset="0" />
<GradientStop Color="White" Offset="0.5" />
<GradientStop Color="Blue" Offset="1" />
</LinearGradientBrush>
</Grid.Background>
</Grid>
```

En VB :

```
Dim  myBrush As new LinearGradientBrush
myBrush.GradientStops.Add(new GradientStop(Colors.Red, 0.0)))
myBrush.GradientStops.Add(new GradientStop(Colors.White, 0.5))
myBrush.GradientStops.Add(new GradientStop(Colors.Blue, 1.0))
Grid.background= myBrush
```

Exemple 2 sur un bouton, on positionne 2 couleurs (blanc et gris) de haut en bas.


Pour que le gradient s'applique de haut en bas StartPoint="0,0" EndPoint="0,1".

le blanc sera à 0 % de la verticale, le gris à 100 % de la verticale.

En XAML :

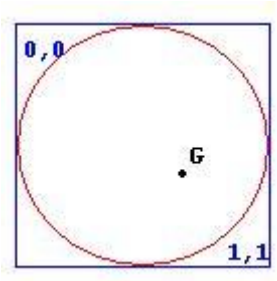
```
<Button  Content="OK">
<Button.Background>
<LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
<GradientStop Color="White" Offset="0" />
<GradientStop Color="LightGray" Offset="1" />
</LinearGradientBrush>
</Button.Background>
</Button>
```

Remarque : on aurait pu mettre un GradientStop en dehors de la zone (à 2 par exemple pour le LightGray) ce qui permet s'estomper la couleur grise et de faire un bouton plus clair.

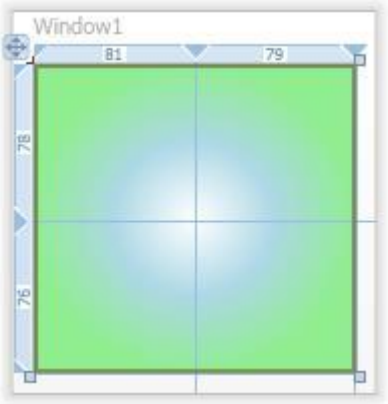
## XI-E-6-c - RadialGradientBrush

Ici on applique 2 couleurs ou plus dans un cercle qui occupe la totalité du conteneur.

Visualisons ce cercle et le système de coordonnées :




Le GradientOrigin donne le centre du gradient. Ci-dessous le centre du gradient est à 0.50 0.50 c'est-à-dire au centre du cercle. Les couleurs seront donc concentriques.



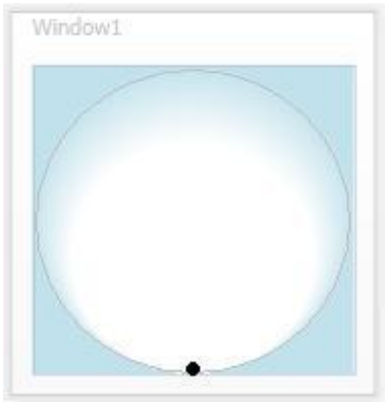

Les GradientStop indiquent la position des couleurs par rapport au centre: le blanc est à l'offset 0 : cercle de blanc au centre ; le bleu est à l'offset 0.5, cela donne un cercle de bleu autour du blanc ; le vert est à l'offset 1 (100 %) : le cercle vert est autour du cercle bleu.

En XAML

```
<RadialGradientBrush GradientOrigin="0.50,0.50">
<GradientStop Color="white" Offset="0" />
<GradientStop Color="LightBlue" Offset="0.5" />
<GradientStop Color="LightGreen" Offset="1" />
</RadialGradientBrush>
```

Pour l'exemple suivant le centre du gradient est à 0.50 1 excentré en bas du cercle.

On a ajouté artificiellement, pour mieux comprendre, le cercle gris qui occupe la totalité du conteneur et le centre du gradient symbolisé par un point noir.




L'offset du blanc est 0.75 : le blanc monte haut.

En XAML :

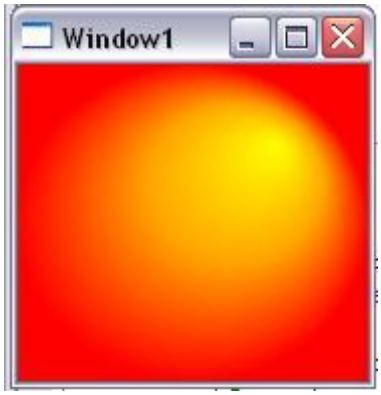
```
<RadialGradientBrush GradientOrigin="0.50,1">
<GradientStop Color="white" Offset="0.75" />
<GradientStop Color="LightBlue" Offset="1.0" />
</RadialGradientBrush>
```

Ici l'offset du blanc est 0.50 : le blanc monte moins haut.


En XAML :

```
<RadialGradientBrush GradientOrigin="0.50,1">
<GradientStop Color="white" Offset="0.50" />
<GradientStop Color="LightBlue" Offset="1.0" />
</RadialGradientBrush>
```

Exemple sur une grid. Centre du gradient excentré en haut à droite (0.75 0.25) et il y a 3 couleurs.




En XAML :

```
<Grid >
<Grid.Background >
<RadialGradientBrush GradientOrigin="0.75,0.25">
<GradientStop Color="Yellow" Offset="0.0" />
<GradientStop Color="Orange" Offset="0.5" />
<GradientStop Color="Red" Offset="1.0" />
</RadialGradientBrush>
</Grid.Background>
</Grid >
```

En VB :

```
Dim myBrush As New RadialGradientBrush
myBrush.GradientOrigin = New Point(0.75, 0.25)
myBrush.GradientStops.Add(New GradientStop(Colors.Yellow, 0.0))
myBrush.GradientStops.Add(New GradientStop(Colors.Orange, 0.5))
myBrush.GradientStops.Add(New GradientStop(Colors.Red, 1.0))
grid.Background = myBrush
```

## XI-E-6-d - ImageBrush

On peut mettre une image dans un rectangle par exemple.

En XAML :

```
<Rectangle Width="75" Height="75">
<Rectangle.Fill>
<ImageBrush ImageSource="c:\graphe.bmp" />
</Rectangle.Fill>
</Rectangle>
```

En VB :

```
Dim Rect As New Rectangle()
Rect.Width = 75
Rect.Height = 75
Dim myBrush = New ImageBrush
myBrush.ImageSource = New BitmapImage(New Uri("c:\graphe.bmp", UriKind.Relative))
Rect.Fill = myBrush
grid.Children.Add(Rect)
```

## XI-E-6-e - Autre

Il existe aussi les DrawingBrush qui permettent de dessiner des motifs géométriques que l'on peut même répéter.

Les VisualBrush, elles, permettent certains effets comme l'effet 'miroir'.

## XI-E-7 - Ressources

Les ressources sont un ensemble d'éléments :
- images, icônes, textes, sons,(ressources contenues dans des fichiers externes) ;
- couleurs, brush, style, ControlTemplate(modification de l'aspect du contrôle), DataTemplate (modification de l'affichage des données) ressources dites 'internes'.

Nous allons donc voir ;

1-Les ressources internes ;

2-Les fichiers de ressources.

## XI-E-7-a - Ressources 'internes'

Ici on va voir comment créer des styles, des modèles de contrôles, des modèles de données pour les appliquer à tous les contrôles ou données.

### XI-E-7-a-i - Ressources 'simples'

Plutôt que d'indiquer X fois quelle couleur ou Brush utiliser dans une fenêtre, il est plus simple de définir une ressource contenant la couleur ou la Brush puis X fois, indiquer quelle ressource utiliser.

Les ressources sont dans un **dictionnaire de ressources**. Dans les ressources d'une fenêtre par exemple ou les ressources d'un objet entre les balises de début et de fin de ressources.

```
<Window.Resources>

</Window.Resources>
```

Remarquez : Resources avec un 's' et non Ressources.

Ici dans ses ressources simples, la ressource est une Brush, une couleur…

Chaque ressource doit avoir une clé unique. Il faut affecter la clé unique via l'**x:Key**, attribut.

En principe, la clé est une chaine.

```
<SolidColorBrush x:Key="MyBrush" Color="Gold"/>
```

Ici la clé unique de la SolidColorBrush est "MyBrush", c'est son 'nom', son nom d'index.

Ensuite dans le code xaml de l'UI, on utilise cette ressource grâce à {StaticResource MyBrush}.
Seul le contrôle utilisant la ressource la prendra en compte !!

```
<Button Background="{StaticResource MyBrush}"/>
<Ellipse Fill="{StaticResource MyBrush}"/>
```

En VB :

```
Me.Background = Window1.FindResource("MyBrush")
```

Exemple complet

Dans une grid, on crée une ressource de type LinearGradientBrush nommé BrushBizarre, ensuite on applique cette Brush au fond d'un bouton.

```
<Grid>
  <Grid.Resources>
    <LinearGradientBrush x:Key="BrushBizarre">
      <GradientStop Color="Yellow" Offset="0" />
      <GradientStop Color="Green" Offset="1" />
    </LinearGradientBrush>
  </Grid.Resources>


  <Button Background="{StaticResource BrushBizarre}">Click Me</Button>
</Grid>
```

**Où mettre les ressources ?**

- Dans un objet. Dans une Grid comme dans l'exemple précédent : Syntaxe : Grid.Resources ; dans ce cas la Grid, mais aussi tous les objets contenus dans la Grid peuvent utiliser la ressource.

- Dans une fenêtre, Entre les balises :

```
  <Window.Resources>

  </Window.Resources>
```

- Dans le fichier Application avec la balise :

```
  <Application.Resources>

  </Application.Resources>
```

- Dans un dictionnaire de ressources :

Menu 'Projet', puis 'Ajouter un dictionnaire de ressources. Nommer le Dictionnary1 puis OK.

Y mettre une solidColorBrush.

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
<SolidColorBrush x:Key="MyBrush" Color="Gold"/>
</ResourceDictionary>
```

Pour que cela fonctionne, il faut ajouter dans le fichier Application la référence au dictionnaire.

```
<Application x:Class="Application"

……

<Application.Resources>

<ResourceDictionary Source="Dictionary1.xaml"/>

</Application.Resources>

</Application>
```

L'emplacement où est déclarée la ressource affecte l'emplacement où la ressource peut s'appliquer.

Si vous déclarez la ressource dans l'élément racine de votre fichier de définition d'application XAML avec la balise 'Application.Resources', elle pourra être utilisée n'importe où dans votre application.

Si vous déclarez la ressource dans une grid grâce à 'Grid.Resources', elle pourra être utilisée uniquement dans la grid.

**Ressources 'Statique' et 'Dynamique'**

Les ressources utilisées avec le mot-clé StaticResource sont récupérées au chargement de l'application alors que pour celles utilisées avec le mot-clé DynamicResource le chargement est différé au moment de l'exécution et la valeur est réévaluée à chaque accès à cette ressource. Dans ce mode WPF crée une expression qui est évaluée à l'utilisation et permet notamment de gérer le cas ou la valeur dépend d'information connue seulement au runtime.

La syntaxe est la même sauf que lors de l'utilisation de la ressource, on utilise DynamicResource.

```
<Window x:Class="Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Window1" Height="300" Width="300">
<Window.Resources>
<SolidColorBrush x:Key="BackgroundBrush" Color="Green" />
</Window.Resources>
<Button Margin="10" Padding="10" Background="{DynamicResource
BackgroundBrush}">
Mon Button
</Button>
</Window>
```

En VB :

```
<Window.Resources>
    <SolidColorBrush x:Key="Mybrush" Color="Red" />
</Window.Resources>


<Button x:Name="btn"Content="Find Position" Click="Button_Click" />
```

```
btn.SetResourceReference(BackgroundProperty, "Mybrush")
```

## XI-E-7-a-ii - Les Styles

Le style est une ressource qui est appliquée à un 'Type' d'objet. On peut créer un style pour tous les Button, toutes les List…

Ici on va créer un style pour tous les TextBlock de la fenêtre.

**TargetType="TextBlock"** indique quel type cela concerne.

**Setter** va servir à définir une **Property** à laquelle on va donner une **Value**. Dans ce style , grâce à 'Setter Property=' on affecte des valeurs aux propriétés des TextBlock.

```
<Window x:Class="Window1"

…….

<Window.Resources>
<Style TargetType="TextBlock">
<Setter Property="Background" Value="Blue"/>
<Setter Property="DockPanel.Dock" Value="Top"/>
<Setter Property="FontSize" Value="18"/>
<Setter Property="Foreground" Value="#4E87D4"/>
<Setter Property="FontFamily" Value="Trebuchet MS"/>
<Setter Property="Margin" Value="0,40,10,10"/>
</Style>
</Window.Resources>
```

Ensuite, si on utilise un textBlock la ressource est appliquée automatiquement :

```
<TextBlock>Titre</TextBlock>
```

Attention,la première ligne est 'Style TargetType="TextBlock"', dans ce cas , le style est appliqué à tous les TextBlock.

Si on utilise x:Key=, la ressource n'est pas appliquée automatiquement à tous les TextBlock !!

Il faut dans ce cas utiliser l'attribut Style= dans le TextBlock sur lequel on veut appliquer le style.

Donc si on écrit :

```
<Style TargetType="TextBlock" x:Key="TitleText">
```

Pour appliquer le style à un TextBlock particulier, il faut écrire ensuite :

```
<TextBlock Style="{StaticResource TitleText}">Titre</TextBlock>
```

On vient d'utiliser un **Setter** pour modifier une propriété, mais il est possible dans un Style de mettre un **DataTemplate**, un **ControlTemplate**, un **EventTrigger** ou un **Trigger**.
Voyons un exemple avec un Trigger.

On peut modifier le Style quand un événement se déclenche : si le 'Target' est le survol d'un MenuItem dans un menu, mettre le texte en rouge :

```
<Window.Resources>
        <Style  TargetType="{x:Type MenuItem}">
            <Style.Triggers>
                <Trigger Property="MenuItem.IsMouseOver" Value="true">
```

```
                    <Setter Property = "Foreground" Value="Red"/>
                    <Setter Property = "FontSize" Value="12"/>
                    <Setter Property = "FontStyle" Value="Italic"/>
                </Trigger>
            </Style.Triggers>
        </Style>

    </Window.Resources>
```

En VB pour appliquer le style à un contrôle :

```
Button1.Style = Resources("TitleText")
ou
textblock1.Style = CType(Me.Resources("TitleText"), Style)
'Option Explicit= On  obligeant un Casr de Objet à Style)
```

On peut étendre un style existant :

```
<Style BasedOn="{StaticResource {x:Type TextBlock}}"
 TargetType="TextBlock" x:Key="TitleText">
 <Setter Property="FontSize" Value="26"/>
</Style>
```

**ATTENTION :** si vous créez un style pour le type TextBlock, le style est appliqué à tous les TextBlock, y compris si le TextBlock fait partie d'un autre contrôle, tel qu'un ListBox.

## XI-E-7-a-iii - Les modèles de Contrôle: Control Template

On a vu plus haut qu'on pouvait mettre un Template dans un contrôle :

```
<Button>
  <Button.Template>
  <ControlTemplate>

  <ControlTemplate>
  </Button.Template>
</Button>
```

Mais on peut aussi créer des **Modèles de contrôles** dans les ressources permettant de modifier tous les contrôles d'un même type.

On spécifie la structure visuelle et les aspects comportementaux d'un type de Contrôle. Là, on ne modifie pas une simple propriété du contrôle, mais son aspect profond.

Comment avoir des boutons en forme d'ellipse ?

Il faut créer un style, redéfinir le style par défaut et créer un nouveau modèle avec :

```
<Setter Property="OverridesDefaultStyle" Value="True"/>
<Setter Property="Template"/>
```

puis définir une ellipse.

Cela donne :

```
<Window.Resources>
<Style TargetType="Button">
<Setter Property="OverridesDefaultStyle" Value="True"/>
<Setter Property="Template">
<Setter.Value>
```

```
<ControlTemplate TargetType="Button">
<Grid>
<Ellipse Fill="Green" />
<ContentPresenter HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</Window.Resources>
```

Maintenant tous les boutons sont comme cela :


Cet exemple fonctionne, mais est incomplet, car comme dans le Template, on n'a pas défini l'aspect du bouton en cas de clic ou quand il a le focus, le bouton ne change jamais d'aspect même quand on clique dessus !!

**On peut faire plus fort et écrire le Template pour gérer les événements.**

En haut de la Window, dans les ressources je mets un 'Template de bouton' :

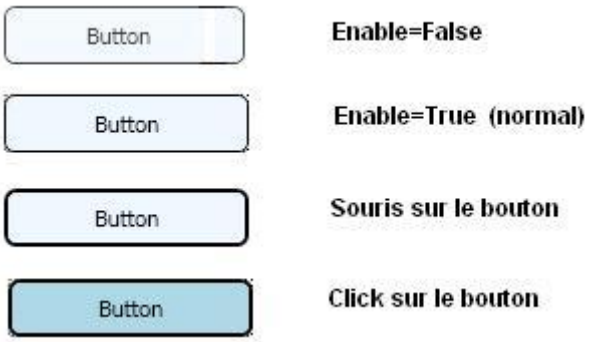
```
<Window.Resources >
        <Style TargetType="Button">
        <Setter Property="SnapsToDevicePixels" Value="true"/>
        <Setter Property="OverridesDefaultStyle" Value="true"/>
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="Button">
                    <Border
          x:Name="Border"
          CornerRadius="5"
          BorderThickness="1"
          BorderBrush="Black"
          Background="AliceBlue">
          <ContentPresenter
            Margin="2"
            HorizontalAlignment="Center"
            VerticalAlignment="Center"
            RecognizesAccessKey="True"/>
            </Border>
                    <ControlTemplate.Triggers>
                        <Trigger Property="IsMouseOver" Value="true">
                            <Setter TargetName="Border" Property="BorderThickness" Value="2" />
                        </Trigger>
                        <Trigger Property="IsPressed" Value="true">
                            <Setter TargetName="Border" Property="Background" Value="LightBlue" /
>
                            <Setter TargetName="Border" Property="BorderBrush" Value="Black" />
                        </Trigger>
                        <Trigger Property="IsEnabled" Value="false">
                            <Setter Property="Foreground" Value="White"/>
                        </Trigger>
                    </ControlTemplate.Triggers>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
</Window.Resources>
```

Le Trigger Property=""IsEnabled"" à False met du blanc comme couleur de fond si le contrôle n'est pas 'enabled'.

Le Trigger Property="IsMouseOver" permet d'épaissir les bords quand la souris est sur le contrôle.

Le Trigger Property="IsPressed" permet d'assombrir le fond quand on clique sur le bouton.

Tous les boutons du formulaire auront cet aspect et ce comportement.

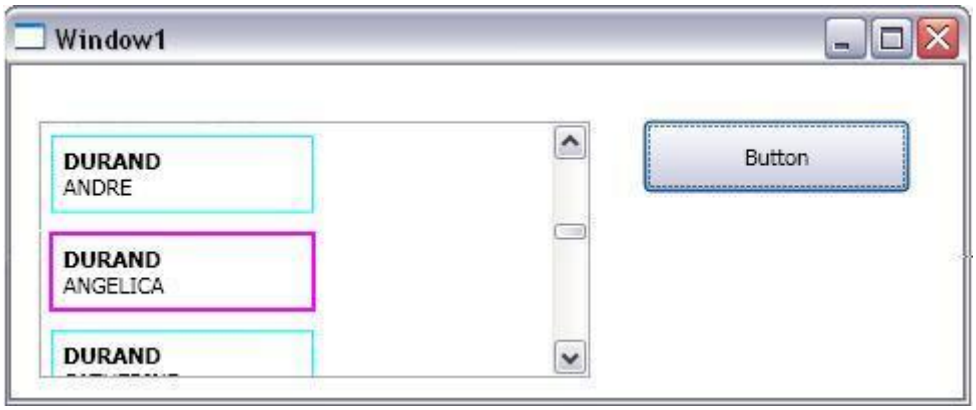



## XI-E-7-a-iv - Les modèles de Données : Data Template

Enfin pour être complet il est possible de modifier l'aspect des données affichées dans un contrôle grâce aux Data Template (modèle de données) ; voir le chapitre sur les liaisons de données pour le détail.

Ici par exemple plutôt que d'afficher bêtement une liste de nom, je vais les afficher dans un cadre coloré en fonction du sexe (pour les données, on utilise un Binding, voir plus loin).

```
<Window.Resources>
<DataTemplate x:Key="MyDataTemplate">
            <Border Name="border" BorderBrush="Aqua" BorderThickness="1"
          Padding="5" Margin="5">

                <Grid>
                    <Grid.RowDefinitions>
                        <RowDefinition/>
                        <RowDefinition/>
                        <RowDefinition/>
                    </Grid.RowDefinitions>

                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="250" />
                        <ColumnDefinition Width="100" />
                        <ColumnDefinition Width="*"/>
                    </Grid.ColumnDefinitions>
                    <TextBlock Text="{Binding Path=NOM}" Grid.Row="0"
        FontWeight="Bold" />
                    <TextBlock Text="{Binding Path=PRENOM}" Grid.Row="1" />

                </Grid>
            </Border>
              <DataTemplate.Triggers>
                <DataTrigger Binding="{Binding Path=SEXE}">
                    <DataTrigger.Value>F</DataTrigger.Value>
                    <Setter TargetName="border" Property="BorderBrush" Value="Pink"/>
                </DataTrigger>
            </DataTemplate.Triggers>

        </DataTemplate>
    </Window.Resources>
```




Comment afficher une liste d'images ?
(Mesphotos étant une liste de noms d'image, on montre ici le principe simplement).

```
<ListBox ItemsSource="{Binding Source={StaticResource MesPhotos}}"
         Background="Silver" Width="600" Margin="10" SelectedIndex="0"/>
```

Ici on affiche la liste des noms d'image.
Comment afficher les images elles-mêmes ? en créant un DataTemplate :

```
<Window.Resources>

…

<!--DataTemplate to display Photos as images
    instead of text strings of Paths-->
<DataTemplate DataType="{x:Type local:Photo}">
  <Border Margin="3">
    <Image Source="{Binding Source}"/>
  </Border>
</DataTemplate>

…

</Window.Resources>
```

## XI-E-7-b - Les fichiers de ressources externes

Exemple de fichier Image.

J'ai une image nommée 'noel.bmp', je veux la mettre dans les ressources pour la charger dans un contrôle image.

i- 'Charger' le fichier image dans les ressources.

Créons un répertoire nommé image sous le répertoire du source : (C:\Documents and Settings\phil\Mes documents\Visual Studio 2008\Projects\MyApplication\MyApplication\image)

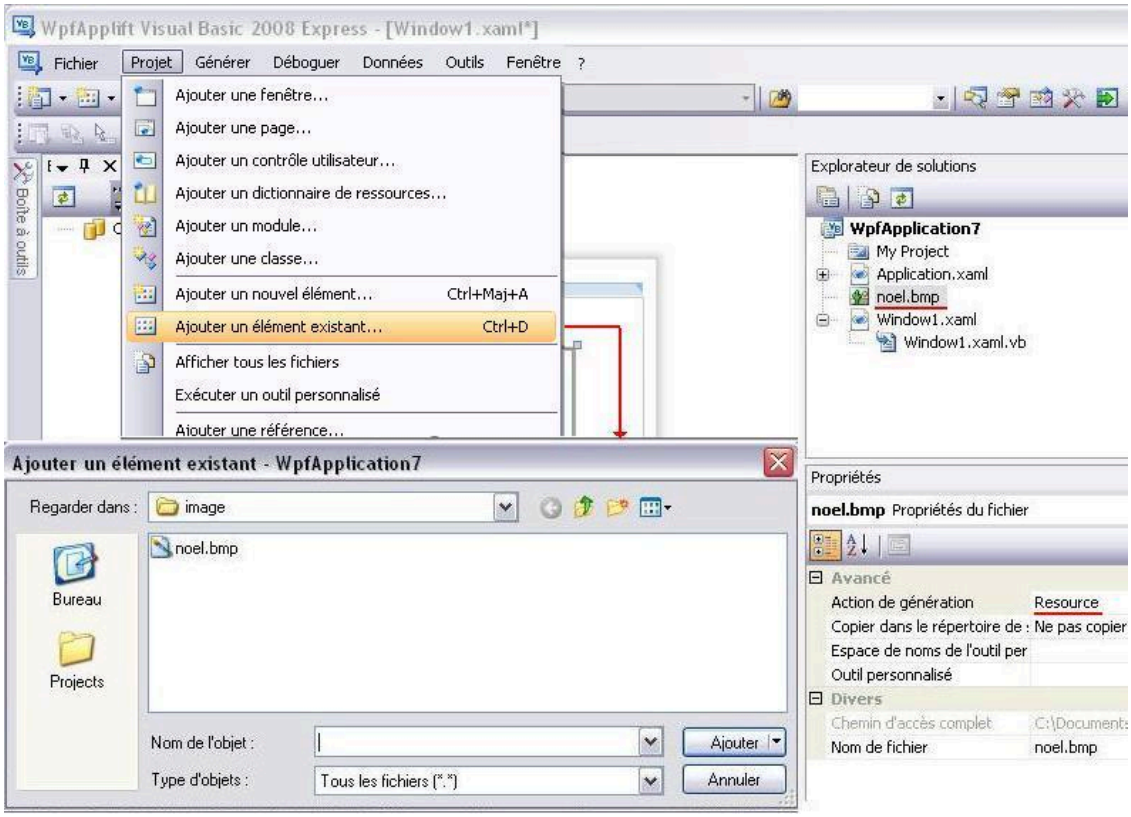
Y mettre le fichier noel.bmp.

Puis incorporons-le dans le projet :

Menu Projet=>Ajouter un élément existant.

Pointer C:\Documents and Settings\phil\Mes documents\Visual Studio 2008\Projects\MyApplication\MyApplication\image\noel.bmp

Puis cliquer sur 'Ajouter'




Noel.bmp apparait dans l'explorateur de solution (en haut à droite).

Cliquer sur noel.bmp puis dans la fenêtre propriété indiquer 'Resource' comme Action de génération (Build Action) ; c'est souvent déjà le cas, car c'est 'Resources' par défaut.

ii- Mettre la ressource dans l'image.

```
<Image Margin="34,36,48,46" Name="Image1" Stretch="Fill" Source = "noel.bmp" />
```

Noter bien : pas de chemin.

L'image noel.bmp apparait dans le contrôle image.

En VB :

```
Image1.Source = New BitmapImage(New Uri("Noel.bmp", UriKind.Relative))
```

Lors de la génération du projet, l'image sera dans l'exécutable.

On peut passer par les ressources de l'application :

o-Charger Noel.bmp dans le projet comme ci-dessus.

oo-Créer une ressource nommée "someImage" dans Application.xaml

```
<Application.Resources>
<ImageSource x:Key= "someImage">noel.bmp</ImageSource>
</Application.Resources>
```

ooo-Mettre la ressource dans une image

```
<Image Margin="34,36,3,8" Name="Image1" Stretch="Fill" Source="{StaticResource someImage}"/>
```

## XI-E-8 - Les liaisons de données ou Binding

"Les liaisons de données sont des processus qui établissent une connexion entre l'interface utilisateur de l'application et la logique métier." En d'autres termes, elles permettent d'établir une connexion entre un contrôle et une source de données. Cela permet d'afficher automatiquement le contenu d'une base de données, d'une collection… dans une DataGrid, une ListBox…

Il faut donc un objet visuel, **la cible**, (ListBox, TextBox…) ayant une propriété de dépendante et faire une liaison avec **la source** de liaison qui est la propriété d'un objet (collection, tableau, base de données…) La liaison peut être uni (OnWay= en lecture seule de la source) ou bidirectionnelle (TwoWay), ce qui permet dans ce dernier cas de mettre à jour la source quand on fait une modif dans l'UI.

## XI-E-8-a - Principes du Binding

### Binding entre objets

Pour qu'une propriété d'un objet (dit 'cible') soit liée à une source, il faut lui affecter un objet **Binding** ( Text="{Binding…) puis indiquer l'objet source avec **ElementName** et la propriété source avec **Path :**

```
<TextBox Text="{Binding Path=Text, ElementName=txtsource}"  />
```

Ici la propriété Text du TextBox est liée à la propriété Text d'un autre contrôle. Si on tape un texte dans le contrôle txtsource, il apparaitra dans la propriété Text de notre TextBox.

### DataContext

La propriété DataSource des contrôles WindowsForms n'existe plus ; les contrôles WPF ont un DataContext. Il peut être renseigné enXaml ou dans le code VB. Le **DataContext** indique donc la source, mais ce qui est fondamental c'est que les contrôles enfants vont hériter du DataContext : si un DockPanel a un DataContext, les Buttons qui sont dedans hériteront de ce DataContext, il suffira pour chaque contrôle enfant d'indiquer une partie de la source (une propriété, un champ…). Autrement dit, si vous avez une source possédant plusieurs propriétés, la source est spécifiée en utilisant **DataContext** du contrôle (pratique pour un groupe d'élément donc). Chaque propriété à afficher sera indiquée par **Path**.

```
<DockPanel Name="DockPanel1">
<DockPanel.DataContext>
    <Binding Source="{StaticResource myDataSource}"/>
</DockPanel.DataContext>
  <Button Background="{Binding Path=ColorName}">OK</Button>
  <Button Background="{Binding Path=ColorName}">OK</Button>
</DockPanel>
```

Ici on crée une liaison du DockPanel avec une collection nommée myDataSource faisant partie des ressources Les boutons héritent du DataContext du DockPanel. Cela permet ensuite de lier la propriété BackGround de chaque bouton à différentes propriétés de myDataSource.

Vous pouvez indiquer le DataContext dans le code VB :

```
Public MyCollection As ObservableCollection (Of String)
DockPanel1.DataContext=  MyCollection
```

Dans ce cas en Xaml, il faut simplement indiquer qu'il y a un Binding avec l'expression "{Binding}" :

```
<ListBox Margin="14,16,8,2" Name="ListBox1"  ItemsSource="{Binding }" />
```

On insiste sur le fait que si vous souhaitez créer une liaison avec un objet source (comme une collection par exemple) qui a déjà été instancié dans le code VB, vous devez définir la propriété DataContext dans le code VB (et pas en XAML).

Si on n'utilise pas le DataContext, on peut aussi utiliser la propriété **Source** du Binding :

```
<Button  Background="{Binding Source={StaticResource myDataSource}/>
```

Si on utilise un contrôle de type ItemsControl tel qu'un ListBox, ListView ou un TreeView pour afficher une collection de données, il faut utiliser **ItemsSource** pour indiquer la source :

```
<ListBox Name="ListBox1"  ItemsSource="{Binding Source={StaticResource ListData}}" />
```

**Mode** indique le mode de mise a jour (OneWay, TwoWay, OneTime, OneWayToSource).

```
<TextBox Name="txtcible" Margin="21,0,25,21"
             Text="{Binding Path=Text, ElementName=txtsource,  Mode=TwoWay}"  />
```

**UpdateTrigger** détermine le moment des mises à jour de la source de liaison.

## XI-E-8-b - Liaison entre contrôles

A - Créons **deux TextBox ;** quand on tape un texte dans la première, il apparait dans la seconde :

```
<TextBox Name="txtsource"  Text="" />
<TextBox Name="txtcible" Margin="21,0,25,21"
             Text="{Binding Path=Text, ElementName=txtsource}"  />
```

Dans la cible, la propriété Text= '**Binding**' entre accolades, ce qui permet d'indiquer qu'il y a une liaison, '**ElementName**' indique la source de la liaison. **Path** la propriété de la source. Il y a une ',' entre chaque élément à partir du second.

La syntaxe suivante est aussi acceptée.

```
<TextBox Name="txtcible"
             Text="{Binding Text, ElementName=txtsource}"  />
```

Par défaut le **Mode** est égal à **OneWay** (la liaison se fait uniquement dans le sens source=>cible).

On peut donner la valeur **TwoWay** au mode, ce qui permet de faire en plus la liaison cible=>source : quand on tape dans la cible, la source est modifiée (cela est visible dans notre exemple ci-dessous quand on quitte la cible) :

```
<TextBox Name="txtsource"  Text="" />
<TextBox Name="txtcible" Margin="21,0,25,21"
             Text="{Binding Path=Text, ElementName=txtsource,  Mode=TwoWay}"  />
```

Pour être complet, il existe aussi les modes OneTime (mise à jour une seule fois au départ), et OneWayToSource (liaison cible=>source uniquement).

On a vu qu'en mode TwoWay, la mise à jour de la source, en cas de modification de la cible était effectuée quand on quittait la cible. C'est parce que la valeur par défaut de la propriété **UpdateSourceTrigger** a la valeur LostFocus (pour un TextBox). Si on veut que la modification se fasse quand on tape sur une touche, il faut écrire :

```
                <TextBox Name="txtcible"
```

```
    Text="{Binding Text,ElementName=txtName, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged }"
  />
```

Ainsi le Trigger qui déclenche la mise à jour dans la liaison est le changement de la propriété Text.

À noter que pour une ListBox **PropertyChanged** est la valeur par défaut de UpdateSourceTrigger. Il y a une autre valeur qui est 'Explicit' ; la mise à jour se fait uniquement quand on fait UpdateSource.

B - Créons **deux ListBox ;** quand on ajoute des éléments dans la première, ils apparaissent dans la seconde :

```
<Grid>
<ListBox  Name="listBox1" SelectionMode="Multiple" />
<ListBox  Name="listBox2" ItemsSource="{Binding ElementName=listBox1, Path=Items, Mode=OneWay}"
</Grid>
```

C - Créons maintenant **un bouton et un Slider ;** quand on bouge le curseur du Slider cela modifie la hauteur du bouton : il faut donc comme source la propriété Value du slider et comme cible la propriété Height du bouton :

```
<Grid>
<Button  Name="Button1"  Width="97" Height="{Binding ElementName=Slider1, Path=Value}">Button</
Button>
<Slider Height="20"  Name="Slider1" Value="5" Maximum="200" Minimum ="0" />
</Grid>
```

**En résumé :** dans le Binding, **ElementName** indique le contrôle source. Si la source est un objet, il faut utiliser la propriété **Path** pour spécifier la propriété de l'objet.

## XI-E-8-c - Liaison Collection-ListBox et Tableau-ListBox

On crée une collection List(Of), on la remplit avec des nombres, comment faire une liaison List() ListBox et afficher automatiquement les nombres dans la listbox ?

Pour une ListBox, c'est **ItemsSource** qui indique la source. Voici le code XAML de la ListBox : ItemsSource ="{Binding}" indique que la source est une liaison, mais n'indique pas la source.

```
<ListBox Name="listbox1" ItemsSource ="{Binding}">
</ListBox>
```

Voici le code VB : ListBox1.DataContext = coll indique quelle est la source de la liaison :

```
Class Window1
    Private Sub Window1_Loaded(ByVal sender As System.Object, ByVal e As
 System.Windows.RoutedEventArgs) _
    Handles Window1.Loaded
        Dim coll As New List(Of String)
        For i As Integer = 0 To 100
            coll.Add(i.ToString)
        Next
        listbox1.DataContext = coll
    End Sub
End Class
```

La liaison est ici par défaut : OneWay.

Cela marche aussi avec un tableau.

```
Class Window1
```

```
    Private Sub Window1_Loaded(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs) _
    Handles Window1.Loaded
        Dim coll(100) As String
        For i As Integer = 0 To 100
            coll(i) = i.ToString
        Next
        listbox1.DataContext = coll
    End Sub
    End Class
```

Si vous souhaitez créer une liaison avec un objet source (comme une collection par exemple) qui a déjà été instancié dans le code VB, vous devez définir la propriété DataContext dans le code VB.

## XI-E-8-d - Liaison avec une collection d'objets

On veut une collection contenant des personnes ayant un nom et un prénom et les afficher par liaison dans une liste.

On va créer une classe 'NomPerson' qui a 2 propriétés : 'Nom' et 'Prenom'. On va aussi créer une classe nommée 'Names' qui hérite de la Classe ObservableCollection(Of). Noter qu'au lieu d'utiliser une collection habituelle(List(Of)), on utilise une ObservableCollection, on verra plus tard pourquoi.

Remarquons que pour utiliser une ObservableCollection il faut ajouter 'Imports System.Collections.ObjectModel'

```
Imports System.Collections.ObjectModel


Public Class Names
    Inherits ObservableCollection(Of NomPerson)

    ' Methods
    Public Sub New()
        MyBase.Add(New NomPerson("Paul", "Durand"))
        MyBase.Add(New NomPerson("Emilie", "Dupont"))
        MyBase.Add(New NomPerson("Philippe", "Lasserre"))
        MyBase.Add(New NomPerson("Jules", "Dubout"))
    End Sub

End Class

Public Class NomPerson
    ' Methods
    Public Sub New(ByVal first As String, ByVal last As String)
        Me._Prenom = first
        Me._Nom = last
    End Sub


    ' Properties
    Public Property Prenom() As String
        Get
            Return Me._Prenom
        End Get
        Set(ByVal value As String)
            Me._Prenom = value
        End Set
    End Property

    Public Property Nom() As String
        Get
            Return Me._Nom
        End Get
        Set(ByVal value As String)
            Me._Nom = value
        End Set
    End Property
```

```
    ' Fields
    Private _Prenom As String
    Private _Nom As String
End Class
```

Dans le code Xaml, il faut avoir accès au namespace du code vb, (on lui donne le préfixe, on le nomme : 'm') on ajoute donc :

```
xmlns:m="clr-namespace:Wpfbindingcoll"
```

Il faut aussi mettre la collection dans les ressources :

```
<Window.Resources>
        <m:Names  x:Key="ListData"/>
```

m:Names indique que dans l'espace de noms 'm', on utilise la classe 'Names' comme ressource ; (la clé se nomme 'ListData'). Pas besoin d'instancier explicitement la classe Names.

Ensuite on fait la liaison ListBox1-ListData, pour cela on utilise **ItemsSource** qui permet de lier un contrôle de type ItemsControl (ListBox, ListWiev, TreeWiev) à une collection. On indique dans ItemsSource qu'il s'agit d'un Binding dont la source est une ressource nommée ListData.

```
<ListBox Margin="14,16,8,2" Name="ListBox1"  ItemsSource="{Binding Source={StaticResource
 ListData}}" />
```

Enfin, il faut afficher correctement les données dans la listbox :

- soit on se contente d'afficher uniquement le nom: on indique dans le code XAML de la ListBox, grâce à **DisplayMemberPath**, quelle propriété afficher :

```
<ListBox Margin="14,16,8,2" Name="ListBox1"  ItemsSource="{Binding}" DisplayMemberPath="Nom"   />
```

- soit on crée un affichage plus élaboré : on met dans les ressources un DataTemplate :

```
<DataTemplate x:Key="MyDataTemplate">
                <Grid>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="100" />
                        <ColumnDefinition Width="100" />
                        <ColumnDefinition Width="*"/>
                    </Grid.ColumnDefinitions>
                    <TextBlock Text="{Binding Path=Nom}" Grid.Column="0"
        FontWeight="Bold" />
                    <TextBlock Text="{Binding Path=Prenom}" Grid.Column="1"  />
                </Grid>
            </DataTemplate>
```

Le DataTemplate (modèle de données) indique pour chaque élément d'afficher dans une grid 2 TextBox liés à la propriété nom et prenom.

Il faut, bien sûr, indiquer à la listbox d'utiliser le datatemplate avec.

```
ItemTemplate="{StaticResource MyDataTemplate}"
```

On voit bien que comme il y a héritage du DataContext, il suffit de mettre le Path dans le Binding Des TextBlock.

Cela donne, au total (version avec DataTemplate) le code XAML suivant :

```
                    <Window x:Class="Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:m="clr-namespace:Wpfbindingcoll"
        Title="Window1" Height="300" Width="300">



    <Window.Resources>
        <m:Names  x:Key="ListData"/>

            <DataTemplate x:Key="MyDataTemplate">
                <Grid>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="100" />
                        <ColumnDefinition Width="100" />
                        <ColumnDefinition Width="*"/>
                    </Grid.ColumnDefinitions>
                    <TextBlock Text="{Binding Path=Nom}" Grid.Column="0"
        FontWeight="Bold" />
                    <TextBlock Text="{Binding Path=Prenom}" Grid.Column="1"  />
                </Grid>
            </DataTemplate>
        </Window.Resources>




        <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="214*" />
            <RowDefinition Height="48*" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="198*" />
            <ColumnDefinition Width="80*" />
        </Grid.ColumnDefinitions>
        <ListBox Margin="14,16,8,2" Name="ListBox1"  ItemsSource="{Binding Source={StaticResource
 ListData}}"
         ItemTemplate="{StaticResource MyDataTemplate}"   />
    </Grid>
</Window>
```

Et voilà, la liaison avec la collection fonctionne et cela donne :


## Création de la collection en VB

Une autre manière de faire est (au lieu de créer la collection 'ListData' dans le code XAML) d'instancier un objet 'MyNames' avec la classe 'Names' dans le code VB et ensuite de le lier au contrôle ListBox1 grâce à la propriété 'DataContext' du ListBox en VB :

```
Class Window1
    Public MyNames As New Names

    Private Sub Window1_Loaded(ByVal sender As System.Object, ByVal e As
 System.Windows.RoutedEventArgs) _
    Handles MyBase.Loaded
        ListBox1.DataContext = MyNames
    End Sub
End Class
```

Dans ce cas en XAML le code de la ListBox est :

```
<ListBox Margin="14,16,8,2" Name="ListBox1"  ItemsSource="{Binding }"
ItemTemplate="{StaticResource MyDataTemplate}"   />
```

Il faut bien sûr enlever dans les ressources la ligne : 'm:Names x:Key="ListData"'

**Ajout d'élément à la collection en VB**

Dans le code vb, si j'ajoute une nouvelle personne à la collection, elle apparait automatiquement dans la liste.

```
MyNames.Add(New NomPerson("toto", "Zorro"))
```

On note bien qu'il faut modifier la collection, ce qui entraine une mise à jour de la ListBox. On est en mode OneWay. Ajouter un élément à la ListBox génère une erreur.

**Nécessité d'utiliser ObservableCollection(Of)**

Si on avait utilisé dans la classe 'Names' une collection comme List(Of), la mise à jour n'aurait pas eu lieu dans la ListBox. Il faut donc bien utiliser ObservableCollection(Of) qui possède l'interface 'INotifyCollectionChanged' qui entraine, en cas de modification de la collection, une mise à jour de la ListBox.

Pour que toute modification de la source soit **immédiatement** répercutée dans l'affichage dans l'affichage, on peut ajouter : UpdateSourceTrigger=PropertyChanged.
Exemple dans un TextBox :

```
<TextBox Grid.Column="1" Grid.Row="0" Margin="3" Text="{Binding Nom,
 UpdateSourceTrigger=PropertyChanged}" />
```

## XI-E-8-e - Liaison avec une base de données

Exemple

Dans une base de données Accès nommé 'Nom.Mdb', j'ai une table 'NOMPATIENT' avec plusieurs colonnes (NOM, PRENOM…SEXE), je veux afficher la colonne des noms et prénoms dans une listBox.

Dans un formulaire, je vais mettre une ListBox1 et un Button1.

Ensuite, dans le code VB je vais créer un DataSet et le 'remplir' avec la BD.

Il faut importer les espaces de noms OLEDB. Créer le DataSet, le remplir. Indiquer la liaison ListBox1-DataSet grâce à 'ListBox1.DataContext'.

```
Imports System
Imports System.Data
Imports System.Data.OleDb

    Class Window1
    'Déclarer la connexion
    Private ObjetConnection As OleDbConnection
    ' Déclaration l'Objet Commande
    Private ObjetCommand As OleDbCommand
    ' Déclaration Objet DataAdapter
    Private ObjetDataAdapter As OleDbDataAdapter
    ' Déclaration Objet DataSet
    Private ObjetDataSet As New DataSet
    'String contenant la 'Requête SQL'
    Private strSql As String
    ' Déclaration Objet DataTable
    Private ObjetDataTable As DataTable
    'Paramètres de connexion à la DB
    Private strConn As String


    Private Sub Button1_Click_1(ByVal sender As System.Object, ByVal e As
 System.Windows.RoutedEventArgs) _
    Handles Button1.Click

    'Initialisation de la chaine de paramètres pour la connexion
    strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" & "Data Source= c:\nom.mdb;"
    'Initialisation de la chaine contenant l'instruction SQL
    strSql = "SELECT FICHEPATIENT.* FROM FICHEPATIENT"
    'Instanciation d'un Objet Connexion
     ObjetConnection = New OleDbConnection
    'Donner à la propriété ConnectionString les paramètres de connexion
    ObjetConnection.ConnectionString = strConn
    'Ouvrir la connexion
     ObjetConnection.Open()
    'Instancier un objet Commande
    ObjetCommand = New OleDbCommand(strSql)
    'Instancier un objet Adapter
    ObjetDataAdapter = New OleDbDataAdapter(ObjetCommand)
    'initialiser l'objet Command
    ObjetCommand.Connection() = ObjetConnection
    'Avec l'aide de la propriété Fill du DataAdapter charger le DataSet
    ObjetDataAdapter.Fill(ObjetDataSet, "FICHEPATIENT")

    'Indiquer au ListBox d'afficher le DataSet

        ListBox1.DataContext = ObjetDataSet

     End Sub
End Class
```

Ensuite dans le code XAML il faut indiquer dans 'ItemsSource' quelle table afficher grâce à **Path :**

```
<ListBox  Name="ListBox1" ItemsSource="{Binding Path=FICHEPATIENT}"  />
```

Maintenant cela devrait marcher, car la liaison est correcte : on clique sur le bouton et on voit :


Cela n'est pas le résultat espéré !! Cela n'affiche qu'une représentation sous forme de chaine du type de l'objet auquel il est lié.

Comment faire ?

On pourrait dans la classe NomPerson Overrider la métisse ToString afin qu'elle retourne une String comportant le nom et le prénom, mais c'est rigide. Il faut plutôt utiliser un MODÈLE DE DONNÉES (DataTemplate) pour indiquer comment afficher les données. On peut mettre ce DataTemplate dans les ressources du formulaire par exemple en le nommant 'MyDataTemplate'.

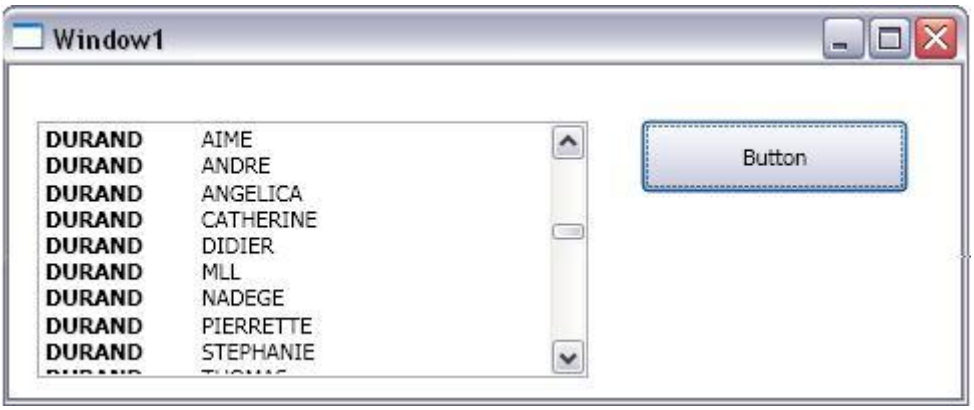
Ici on va indiquer comme modèle, au sein d'une Grid, de mettre 2 TextBlocks liés au Champ NOM et PRENOM. Chaque ListItem de la ListBox sera composé de cette Grid Contenant les 2 TextBlock (le premier sera en gras).

```
        <Window.Resources>
        <DataTemplate x:Key="MyDataTemplate">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="250" />
                    <ColumnDefinition Width="100" />
                    <ColumnDefinition Width="*"/>
                </Grid.ColumnDefinitions>
                <TextBlock Text="{Binding Path=NOM}" Grid.Column="0"
        FontWeight="Bold" />
                <TextBlock Text="{Binding Path=PRENOM}" Grid.Column="1" />
            </Grid>
        </DataTemplate>
    </Window.Resources>
```

Enfin, il faut indiquer à la listBox d'utiliser le DataTemplate grâce à l'attribut ItemTemplate :

```
    <ListBox  Name="ListBox1" ItemsSource="{Binding Path=FICHEPATIENT}"
    ItemTemplate  ="{StaticResource MyDataTemplate}"  />
```

Et là enfin, cela marche et donne :




Pour résumer, voilà le code XAML complet :

```
<Window x:Class="Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="280" Width="655">
    <Window.Resources>


        <DataTemplate x:Key="MyDataTemplate">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="250" />
                    <ColumnDefinition Width="100" />
                    <ColumnDefinition Width="*"/>
                </Grid.ColumnDefinitions>
                <TextBlock Text="{Binding Path=NOM}" Grid.Column="0"
        FontWeight="Bold" />
                <TextBlock Text="{Binding Path=PRENOM}" Grid.Column="1" />

            </Grid>
        </DataTemplate>
    </Window.Resources>

    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="465*" />
            <ColumnDefinition Width="168*" />
        </Grid.ColumnDefinitions>
        <ListBox Margin="14,28,14,10" Name="ListBox1" ItemsSource="{Binding Path=FICHEPATIENT}"
        ItemTemplate  ="{StaticResource MyDataTemplate}"  />
        <Button Height="37" Margin="0,37,42,0" Name="Button1" VerticalAlignment="Top"
        Grid.ColumnSpan="2" HorizontalAlignment="Right" Width="134">Button</Button>
    </Grid>
</Window>
```

Notez qu'on aurait pu mettre plus simplement le DataTemplate directement dans le ListBox : ici le DataTemplate comporte un StackPanel avec 3 TextBocks.

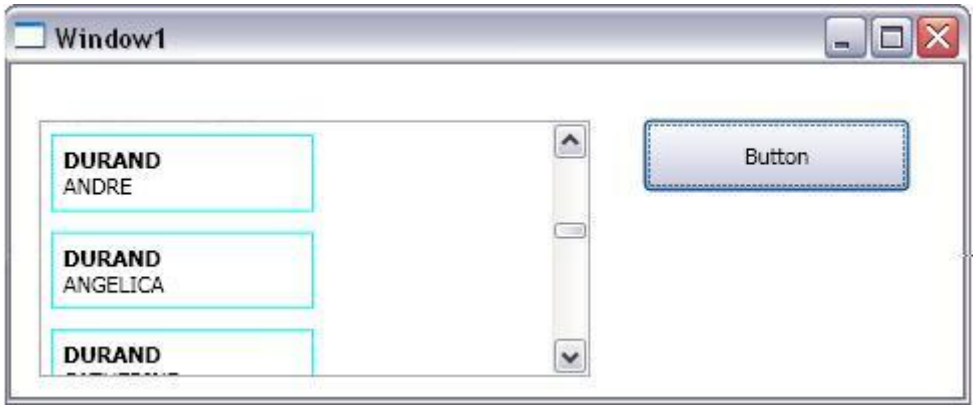
```
<ListBox Width="400" Margin="10"
         ItemsSource="{Binding Source=FICHEPATIENT}">
   <ListBox.ItemTemplate>
     <DataTemplate>
       <StackPanel>
         <TextBlock Text="{Binding Path=NOM}" />
         <TextBlock Text="{Binding Path=PRENOM}"/>
         <TextBlock Text="{Binding Path=RUE}"/>
       </StackPanel>
     </DataTemplate>
   </ListBox.ItemTemplate>
 </ListBox>
```

Les Data Template ont une puissance extraordinaire : je vais modifier mon modèle, ajouter un bord bleu à la grid et mettre le NOM sur la ligne 1, et le prenom sur la ligne 2 (on remarque qu'on a ajouté au préalable des RowsDefinitions, voir le chapitre sur les grid).

Voici le Data Template :

```
<Window.Resources>
<DataTemplate x:Key="MyDataTemplate">
            <Border Name="border" BorderBrush="Aqua" BorderThickness="1"
          Padding="5" Margin="5">

                <Grid>
                    <Grid.RowDefinitions>
                        <RowDefinition/>
                        <RowDefinition/>
                        <RowDefinition/>
                    </Grid.RowDefinitions>

                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="250" />
                        <ColumnDefinition Width="100" />
                        <ColumnDefinition Width="*"/>
                    </Grid.ColumnDefinitions>
                    <TextBlock Text="{Binding Path=NOM}" Grid.Row="0"
        FontWeight="Bold" />
                    <TextBlock Text="{Binding Path=PRENOM}" Grid.Row="1" />

                </Grid>
            </Border>
        </DataTemplate>
    </Window.Resources>
```

Cela donne :



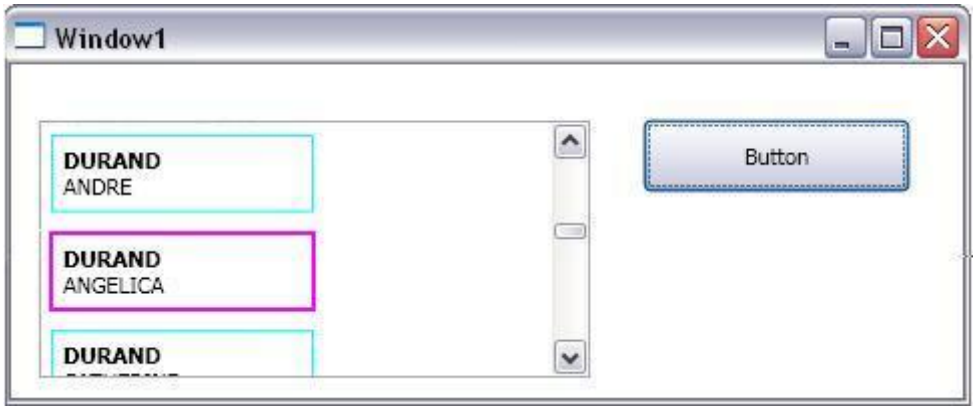

De plus en plus fort, on peut modifier l'aspect des données suivant certaines conditions. Si dans la ListBox des noms il s'agit d'une femme, le cadre doit être rose.

On utilise le **DataTemplate.Trigger**, on le lie au champ SEXE ; si sa valeur est égale à 'F', la couleur du bord devient égale à Pink.

```
<Window.Resources>
            <DataTemplate x:Key="MyDataTemplate">
            <Border Name="border" BorderBrush="Aqua" BorderThickness="1"
          Padding="5" Margin="5">

        ……
            </Border>
            <DataTemplate.Triggers>
                <DataTrigger Binding="{Binding Path=SEXE}">
                    <DataTrigger.Value>F</DataTrigger.Value>
```

```
                    <Setter TargetName="border" Property="BorderBrush" Value="Pink"/>
                </DataTrigger>
            </DataTemplate.Triggers>

        </DataTemplate>
    </Window.Resources>
```

Cela donne :




## XI-E-9 - Les Triggers, les StoryBoard

On peut déclencher une action quand un événement se produit.
Pour cela on va utiliser un **Trigger** (gâchette, détente) dans un Style.
Exemple
On veut modifier le Style des boutons (ici la couleur de l'arrière-plan, la taille et le style des caractères) quand un événement se déclenche (ici le survol d'un bouton) :

```
<Window x:Class="MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="525">

    <Window.Resources>
        <Style  TargetType="Button">
            <Style.Triggers>
                <Trigger Property="Button.IsMouseOver" Value="true">
                    <Setter Property = "Foreground" Value="Red"/>
                    <Setter Property = "FontSize" Value="20"/>
                    <Setter Property = "FontStyle" Value="Italic"/>
                </Trigger>
            </Style.Triggers>

        </Style>

    </Window.Resources>

    <Button  >
       OK
    </Button>
</Window>
```

Ainsi quand la souris passe sur le bouton, les caractères deviennent rouges en italique et augmentent de taille.

**Attention : Quand la condition n'est plus remplie, les Setter n'agissent plus et les caractères redeviennent comme avant.**

S'il y a plusieurs conditions, il faut utiliser les multiTriggers :

```
<MultiTrigger>
<MultiTrigger.Conditions>
  <Condition Property="IsMouseOver" Value="True" />
  <Condition Property="IsFocused" Value="True" />
</MultiTrigger.Conditions>
<Setter Property="Button.Foreground" Value="Red" />
</MultiTrigger>
```

On peut aussi mettre un Trigger sur la valeur d'une donnée, on parle de DataTrigger :

```
<DataTrigger Binding="{Binding Path=Sexe}" Value="F">
<Setter Property="Button.Foreground" Value="Pink" />
</DataTrigger>
```

On met le DataTrigger sur le champ 'Sexe' ; si celui-ci est égal à 'F', on affiche en Rose !!

On peut mettre les triggers dans un Style, mais aussi dans un Template ; on ne peut pas le mettre directement dans un contrôle (du moins je n'y suis pas arrivé).

On peut aussi mettre un Trigger sur un **événement** (**eventTrigger**), mais comme action, on utilise un StoryBoard (scénarimage).
(Quand l'événement se produit, il n'y a pas de retour à l'état antérieur comme avec le Trigger simple ; le StoryBoard permet lui un changement avec un temps donné).

Ici on va créer dans les ressources de la fenêtre un style pour les boutons; il contient un EventTrigger déclenché par le clic et qui exécute un StoryBoard effectuant une animation qui augmente la largeur du bouton puis une animation qui la réduit :

```
<Window x:Class="MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="525">

    <Window.Resources>
        <Style  TargetType="Button">
            <Style.Triggers>
                        <EventTrigger RoutedEvent="Click">
                            <BeginStoryboard>
                                <Storyboard>
                                    <DoubleAnimation To="150" Duration="0:0:1.5"
                                    AccelerationRatio="0.10" DecelerationRatio="0.25"
                                    Storyboard.TargetProperty="(Button.Width)" />
                                    <DoubleAnimation To="100" Duration="0:0:1.5"
                                    AccelerationRatio="0.10" DecelerationRatio="0.25"
                                    Storyboard.TargetProperty="(Button.Width)" />
                                </Storyboard>
                            </BeginStoryboard>
                        </EventTrigger>
            </Style.Triggers>
        </Style>
    </Window.Resources>

<Grid>
 <Button
  HorizontalAlignment="Left" Margin="4,0,0,0" Name="Button1" VerticalAlignment="Top" Width="100"
  Height="100">
 </Button>
</Grid>
</Window>
```

### Détaillons la notion de StoryBoard

Dans le StoryBoard suivant **DoubleAnimation** permet d'animer la valeur d'une propriété de type 'Double' sur un Duration spécifié. Il existe aussi **ColorAnimation**, **PointAnimation**.

On peut utiliser dans le storyBoard **From** (Valeur de départ), **To** (valeur d'arrivée), **Duration**="H:M:S" (durée) des **AcceleratorRatio** ou **DeceleratorRatio**.
Pour spécifier une valeur finale par rapport à la valeur de départ, définissez la propriété **By**(au lieu de la propriété To).
On peut ajouter AutoReverse= True pour un retour à l'état initial et RepeatBehavior= "Forever" pour que l'animation se répète sans fin.

```
<Storyboard>
  <DoubleAnimation From="1.0" To="0.0" Duration="0:0:1"
    AutoReverse="True" RepeatBehavior="Forever" />
</Storyboard>
```

On peut ajouter un BeginTime pour démarrer le StoryBoard au bout d'un certain temps.

```
<Storyboard BeginTime="0:0:2" x:Name="myStoryboard">
</StoryBoard>
```

**TargetName** indique l'objet à animer.
**TargetProperty** indique sur quelle propriété agir.
Ici on va agir sur un rectangle nommé 'MonRectangle' et sur sa propriété 'Opacity'.

```
<Storyboard>
  <DoubleAnimation
    Storyboard.TargetName="MonRectangle"
    Storyboard.TargetProperty="Opacity"
    From="1.0" To="0.0" Duration="0:0:1"
    AutoReverse="True" RepeatBehavior="Forever" />
</Storyboard>
```

Pour que le StoryBoard démarre il faut utiliser **Begin**.
- Soit dans le code XAML :

```
<BeginStoryboard>
   <Storyboard>
   </Storyboard>
</BeginStoryboard>
```

- Soit avec du code vb.

```
myStoryboard.Begin()
```

Dans ce cas le StoryBoard doit avoir un x:Name :

```
<Storyboard x:Name="myStoryboard">
</StoryBoard>
```

En plus de Begin, il existe Stop, Resume, Pause.

On peut aussi effectuer des 'Transformation' : un exemple de Microsoft sur la rotation d'un rectangle :

```
<StackPanel Margin="15">
  <StackPanel.Resources>
    <Storyboard x:Name="myStoryboard">
      <DoubleAnimation
       Storyboard.TargetName="myTransform"
       Storyboard.TargetProperty="Angle"
       From="0" To="360" Duration="0:0:5"
       RepeatBehavior="Forever" />
    </Storyboard>
  </StackPanel.Resources>
  <Rectangle Width="50" Height="50" Fill="RoyalBlue"
   MouseLeftButtonDown="StartAnimation">
```

```
    <Rectangle.RenderTransform>
      <RotateTransform x:Name="myTransform" Angle="45" CenterX="25" CenterY="25" />
    </Rectangle.RenderTransform>
  </Rectangle>
</StackPanel>
```

On peut ajouter des effets d'accélération : ici un effet rebondissant avec BouceEase.

```
<Storyboard x:Name="myStoryboard">
            <DoubleAnimation From="30" To="200" Duration="00:00:3"
                Storyboard.TargetName="myRectangle"
                Storyboard.TargetProperty="Height">
                <DoubleAnimation.EasingFunction>
                    <BounceEase Bounces="2" EasingMode="EaseOut"
                                Bounciness="2" />
                </DoubleAnimation.EasingFunction>
            </DoubleAnimation>
        </Storyboard>
```

# XI-F - Les différents contrôles

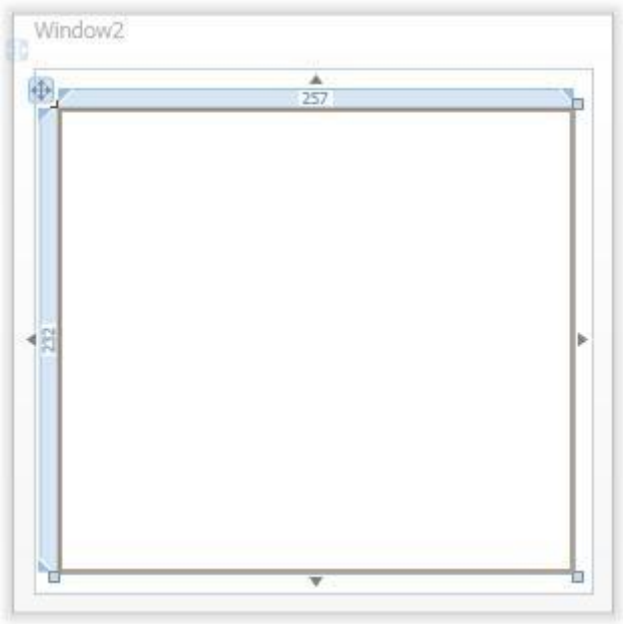
## XI-F-1 - Les Conteneurs

Les conteneurs peuvent contenir plusieurs contrôles et permettent de les positionner.

### XI-F-1-a - Les Grid

En WPF une Grid est un conteneur, contenant des cellules servant à aligner les contrôles (et pas une grille de données comme dans les Windows forms).

Aller chercher un 'Grid' dans la boite à outils à gauche et la mettre dans une fenêtre vide.

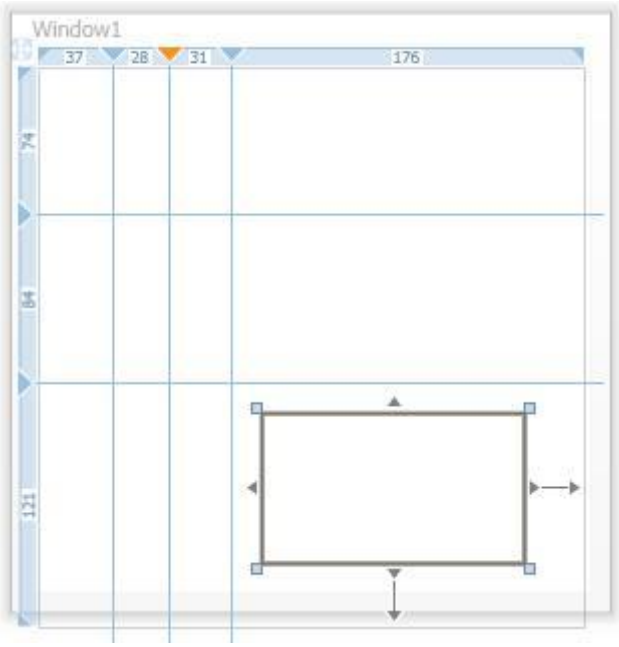



Cela donne dans le code XAML :

```
<Grid Margin="12,20,9,10" Name="Grid1" />
```

En fait dans VB quand on ajoute une fenêtre vide à un projet, la fenêtre contient déjà une grid.

Pour ajouter des lignes et des colonnes dans la grille, il faut cliquer dans les barres bleutées horizontales ou verticales et positionner les traits.




On voit dans le code XAML les largeurs des colonnes et les hauteurs des lignes.

```
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="74*" />
<RowDefinition Height="84*" />
<RowDefinition Height="121*" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="37*" />
<ColumnDefinition Width="28*" />
<ColumnDefinition Width="31*" />
<ColumnDefinition Width="176*" />
</Grid.ColumnDefinitions>
</Grid>
```

On peut aussi vouloir définir le nombre de lignes et de colonnes sans définir leur largeur : 3 lignes, 2 colonnes.

```
<Grid>
      <Grid.RowDefinitions>
       <RowDefinition/>
       <RowDefinition/>
       <RowDefinition/>
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
      </Grid.ColumnDefinitions>
    </Grid>
```

On peut ensuite ajouter des contrôles dans les cellules de la grille pour les positionner.

Pour chaque contrôle ajouté dans une cellule de la grid, on définira ses marges et son ancrage.

Ici on va ajouter une ListBox en bas à droite de la grille (en allant la chercher dans la boite à outils et en la déposant avec la souris), voyons ce que cela donne dans le code xaml.

```
<Grid Height="279" Name="Grid1" Width="272">
<Grid.RowDefinitions>
<RowDefinition Height="74*" />
<RowDefinition Height="84*" />
<RowDefinition Height="121*" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="37*" />
<ColumnDefinition Width="28*" />
<ColumnDefinition Width="31*" />
<ColumnDefinition Width="176*" />
</Grid.ColumnDefinitions>
<ListBox Grid.Column="3" Grid.Row="2" Margin="22,26,21,19" Name="ListBox1">
<ListBoxItem>toto</ListBoxItem>
<ListBoxItem>lululu</ListBoxItem>
</ListBox>
</Grid>
```

**ListBox Grid.Column="3" Grid.Row="2"** indiquent dans quelle cellule est la ListBox.

Attention la première colonne est Column=0 et la première ligne est Row=0.

On aurait pu ajouter **Grid.ColumnSpan=2** pour indiquer que la ListBox occupe 2 colonnes (la 3 et la 4 dans ce cas).

Il existe aussi Grid.RowSpan pour étendre le contrôle sur plusieurs lignes de la Grid.

La aussi Margin="22,26,21,19" indique les distances au conteneur autour de la ListBox.

Dans la définition des dimensions, '*' signifie 'autant que possible, 'Auto' force à prendre la taille des éléments qu'elle contient.

```
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
```

Comment faire la même chose en VB ? la Grid1 étant présente, ajoutons une ListBox comme ci-dessus :

```
Dim lb As New ListBox
lb.Margin = New Thickness(22, 26, 11, 19)
Grid.SetColumn(lb, 3)
Grid.SetRow(lb, 2)
Grid1.Children.Add(lb)
```

Il existe aussi une **UniformGrid** qui reste similaire à la Grid à la seule différence que dans un UniformGrid les colonnes et les lignes ont forcément la même taille.

Enfin il y a à notre disposition un **GridSplitter**, qui ressemble au SplitContainer des WindowsForms, il permet de redimensionner les lignes et colonnes pendant de l'exécution de l'application.

## XI-F-1-b - Les StackPanel

Arrange les contrôles sur une même ligne qui peut être horizontale ou verticale.

(On parle d'empilement.)

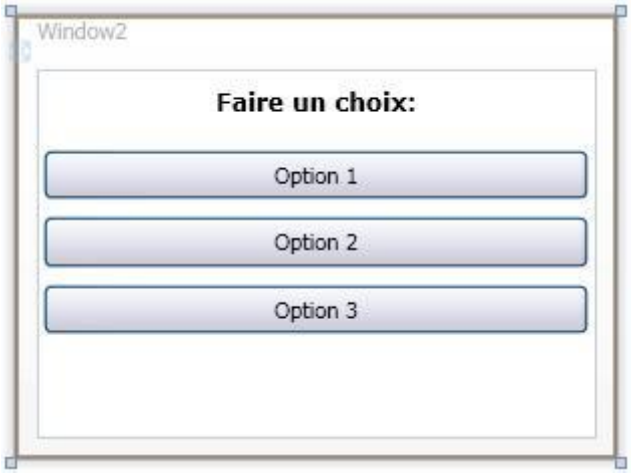
En VB :

```
Dim instance As StackPanel
```

En XAML :

```
<StackPanel> Children </StackPanel>
```

La valeur par défaut de Orientation du StackPanel est Vertical, aussi les contrôles seront positionnés de haut en bas. L'attribut **Orientation="Horizontal"** permet de mettre les contrôles enfant de gauche à droite.




On a ajouté dans le StackPanel un TextBlock et 3 boutons.

En XAML :

```
<StackPanel Orientation="Horizontal">
<TextBlock Margin="6" Padding="3" HorizontalAlignment="Center"
FontFamily="Verdana" FontSize="12" FontWeight="Bold">
Faire un choix:
</TextBlock>
<Button Margin="3,8,3,4" Padding="2">Option 1</Button>
<Button Margin="3,4,3,4" Padding="2">Option 2</Button>
<Button Margin="3,4,3,4" Padding="2">Option 3</Button>
</StackPanel>
```

On voit qu'il suffit de mettre les objets dans le StackPanel.

Ils seront empilés.

En VB, voici un autre exemple, qui peut être collé dans la Sub New de Window1 :

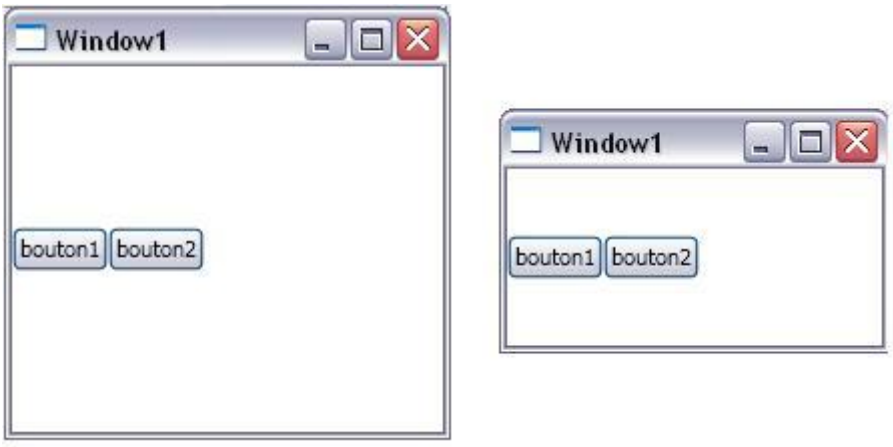
```
Dim StackPanel As New StackPanel

StackPanel.Orientation = System.Windows.Controls.Orientation.Horizontal

StackPanel.HorizontalAlignment = System.Windows.HorizontalAlignment.Right

StackPanel.VerticalAlignment = System.Windows.VerticalAlignment.Center

Dim MyButton1 As New Button

Dim MyButton2 As New Button

MyButton1.Content = "bouton1"

MyButton2.Content = "bouton2"
```

```
StackPanel.Children.Add(MyButton1)

StackPanel.Children.Add(MyButton2)

Me.content= StackPanel
```

On a instancié des boutons que l'on ajoute à la collection Children du StackPanel.

Enfin, on est probablement dans une Window complètement vide, on met donc le StackPanel dans la propriété Content de Me (c'est-à-dire de la Window ici).

Cela donne



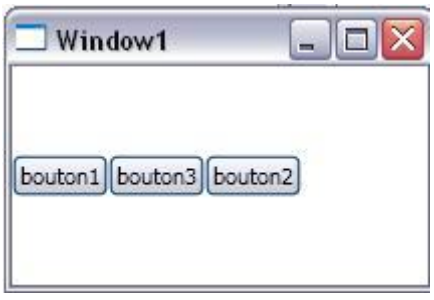

La dimension du bouton est définie par la dimension de son texte.

Si on modifie la hauteur de la fenêtre, on remarque que les boutons sont repositionnés automatiquement toujours au centre de la hauteur grâce au VerticalAlignment. C'est l'avantage du Layout.

On peut insérer un bouton3 supplémentaire à une position définie :

```
Dim MyButton3 As New Button
MyButton3.Content = "bouton3"
StackPanel.Children.Insert(1, MyButton3)  '1 indique la position d'insertion.
```

Le bouton2 est automatiquement repositionné.

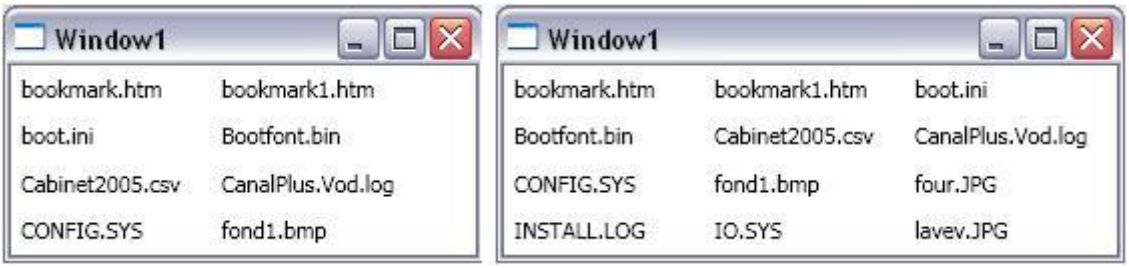



## XI-F-1-c - Les WrapPanel

Le WrapPanel est une variante où les éléments sont mis à la suite l'un de l'autre, mais où il y a "retour à la ligne" quand on atteint la limite du conteneur.

Exemple : affichons les fichiers du répertoire c : dans un WrapPanel existant :

```
Dim WrapPanel As New WrapPanel
Dim dirs() As String = Directory.GetFiles("C:\")
For Each f As String In dirs
Dim file As New FileInfo(f)
Dim lbl As New Label
lbl.Width = 100
lbl.Content = file.Name.ToString
WrapPanel.Children.Add(lbl)
Next
Me.Content = WrapPanel
```




L'intérêt de ce contrôle est que si on élargit la fenêtre, l'affichage est automatiquement reformaté (il passe de 2 à 3 colonnes ici).

## XI-F-1-d - Les DockPanel

Arrange les contrôles soit horizontalement soit verticalement, mais en plus les contrôles sont dockés (attachés) à un des bords.

Cela remplace la propriété 'Dock' des windowsForms.

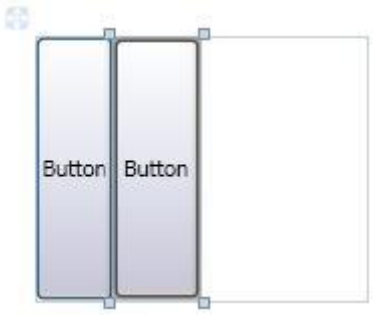
Dans un DockPanel, on met 2 boutons.

On peut utiliser dans la fenêtre de propriété des boutons la propriété DockPanel.Dock :

pour le premier bouton DockPanel.Dock="Left" met le premier bouton contre le bord gauche ;

pour le second bouton DockPanel.Dock="Left" met le second bouton à gauche contre le premier.

Bel exemple d'une propriété des boutons attachée au DockPanel.




En XAML cela donne :

```
<DockPanel>
```

```
    <Button DockPanel.Dock=&#8220;Left&#8220;>Button</Button>
    <Button DockPanel.Dock=&#8220;Left&#8220;>Button</Button>
</DockPanel>
```

Ne pas oublier qu' on peut donner à Width et à Height la valeur "Auto" afin que le contrôle occupe la totalité de la largeur ou de la hauteur restante.

## XI-F-1-e - Les Canvas

Il existe un contrôle Canvas qui permet de positionner des contrôles en indiquant leurs coordonnées (comme dans les Windows Forms), mais il n'y a pas de repositionnement automatique quand on modifie les dimensions de la fenêtre, ce qui limite son intérêt.

On positionne les contrôles avec Canvas.Top et Canvas.Left qui se trouve dans le code de chaque contrôle enfant.

En XAML :

```
<Canvas>
<Image Source="MyImage.gif" Canvas.Left="100" Canvas.Top="50"/>
<CheckBox Canvas.Top="80" Canvas.Left="0">Check Me</CheckBox>
<RadioButton Canvas.Top="0" Canvas.Left="80">Yes</RadioButton>
</Canvas>
```



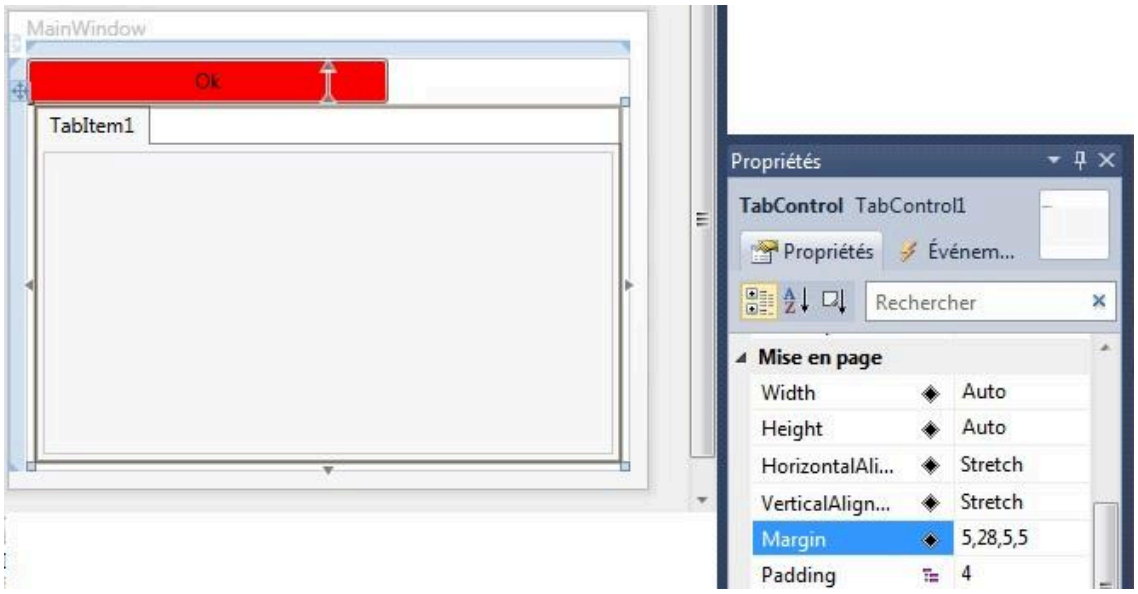
*Les Grid, DockPanel StackPanel, WrapPanel gèrent le positionnement automatique des enfants ; pas Canvas. Cela veut dire que si la taille de la fenêtre change, les panneaux vont automatiquement modifier la position des éléments enfants.*

*Certains panneaux, comme DockPanel ou Grid, peuvent aussi affecter la taille de leurs enfants.*

## XI-F-1-f - Les Onglets

Plutôt que d'utiliser des fenêtres MDI on utilise des onglets (TabControl).

```
<TabControl Height="Auto" HorizontalAlignment="Stretch" Name="TabControl1" VerticalAlignment="Stretch" Width="Aut
            <TabItem Header="TabItem1" Name="TabItem1">
                <Grid />
            </TabItem>
        </TabControl>
```




Dans le TabControl chaque onglet correspond à un **TabItem**, la propriété Header permettant d'afficher le titre de l'onglet, Le TabItem a aussi une propriété Content dans laquelle on peut mettre une Grid ou un StackPanel.

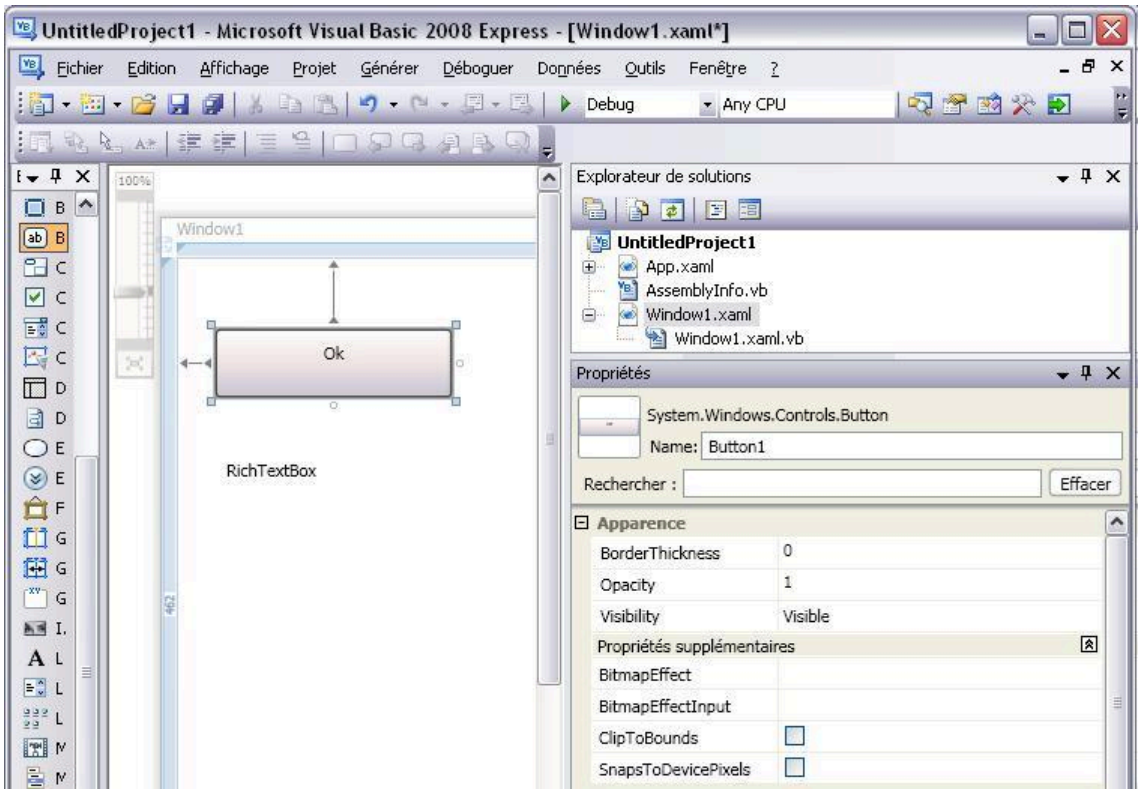
Les onglets peuvent être à droite, à gauche, en haut ou en bas ; ici à gauche :

```
<TabControl TabStripPlacement="Left" Margin="0, 0, 0, 10">

</TabControl>
```

## XI-F-2 - Les Boutons et RepeatButton

### XI-F-2-a - Les 'Button'

Pour ajouter un bouton, on clique sur le bouton dans les outils à gauche puis sur le formulaire, on appuie (bouton gauche de la souris), on déplace puis on lâche. Le bouton apparait.

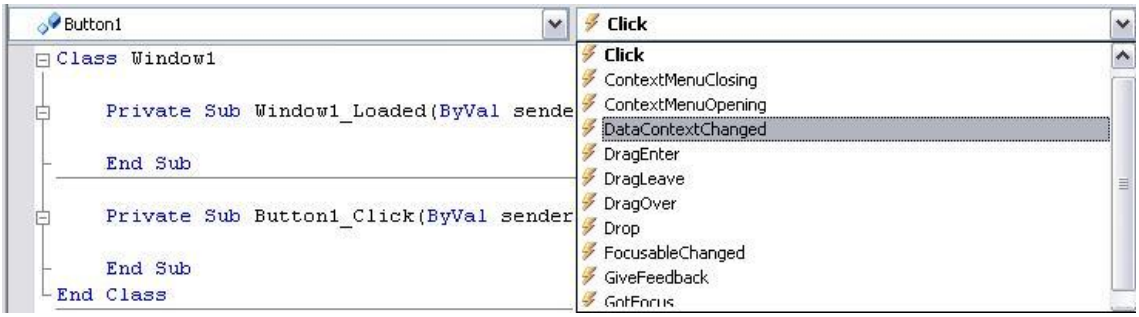



En bas à droite, on a la fenêtre de propriétés du bouton.

Le nom du contrôle est en haut, après 'Name'.

Dans le designer, la propriété 'Content' contient le texte à afficher sur le bouton.

Si on double-clique sur un bouton, par exemple, on se retrouve dans la procédure événement correspondante qui est Button1_Click :




**Créons un bouton avec du code XAML**

Il faut taper dans la fenêtre XAML :

```
<Button>OK</Button>
```

Cela crée un bouton sur lequel est affiché 'OK'. Il apparait en haut.

Faisons plus complet :

```
<Button Height="39" Margin="40,51,118,0" Name="Button1" VerticalAlignment="Top" Click="OnClick5">OK</
Button>
```

Voyons le détail de cette ligne.

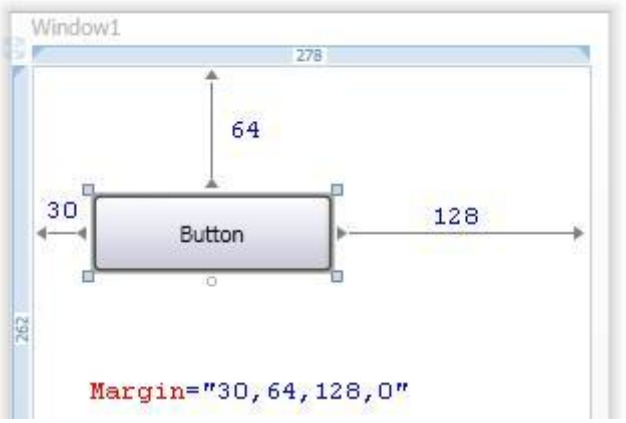
La balise '**Button**' crée un bouton.

**Name="Button1"** indique le nom du contrôle dans le designer.

**VerticalAlignment="Top"** indique que le contrôle est ancré en haut (il reste toujours à la même distance du bord sur lequel il est ancré. Valeurs possibles :T op, Bottom, Center.

**HorizontalAlignment="Left"** même chose pour l'alignement horizontal.

**Margin** définit la distance par rapport aux bords.




**Width="50" Height="30"** indiquent les dimensions du bouton (largeur, hauteur).

On peut aussi ajouter MinWidth MinHeight MaxWidth et MaxHeight qui indiquent les tailles minimales et maximales.

On aurait pu utiliser l'attribut **Content="OK"** pour mettre un texte dans le bouton.

**Click="OnClick5"** indique que l'action de cliquer sur le bouton déclenche la procédure OnClick5.

Cela crée automatiquement, dans le code VB, la routine suivante :

```
Sub OnClick5(ByVal sender As Object, ByVal e As RoutedEventArgs)
End Sub
```

**sender** est l'objet qui a déclenché l'événement.

**e** contient les arguments de l'événement, remarquons que e est de type RoutedEventArgs (et pas EventArgs comme dans les Windows Forms).

Dans VB si on n'ajoute pas 'Click=…' dans le code XAML la routine Button1_Click est automatiquement créée.

On peut ajouter du code dans la routine, pour modifier les propriétés du bouton par exemple :

```
Sub OnClick5(ByVal sender As Object, ByVal e As RoutedEventArgs)
 Button1.FontSize = 16
 Button1.Content = "Ok Ok…"
 Button1.Background = Brushes.Red
```

```
End Sub
```

Voici le bouton de départ et le bouton quand on a cliqué dessus.


Pour mettre une image dans un bouton, pas de propriété 'Image' !! il faut avoir le fichier image et ajouter une balise Image :

```
<Button Height="38" Margin="94,22,14,0" Name="Button2" VerticalAlignment="Top" >
<Image Source="oktransp.GIF"></Image>
</Button>
```

Dans ce cas, pas de texte sur le bouton en même temps qu'une image : content="OK" est refusé, car content ne peut contenir plus d'un objet et il y a déjà une image. Pour mettre un texte et une image dans le bouton, il faut mettre un conteneur comme une grid dans le bouton puis mettre dans les cellules de la grid le texte et l'image.

On a choisi un fichier GIF avec un fond 'transparent' ce qui permet de ne pas voir le fond de l'image.

**Comment mettre un texte et une Image dans un bouton ?**

Il faut mettre un StackPanel dans le bouton (puisque celui-ci ne peut contenir qu'un seul objet), dans ce StackPanel mettre un TextBlock et une Image. Le faire en tapant du code XAML (dans le designer VB c'est difficile de mettre un StackPanel dans un Button, il se met dessus et pas dedans, donc copier-coller le code XAML). De plus l'image doit être dans les ressources : voir ce chapitre.

```
<Button  Name="Button1">
            <StackPanel   Name="StackPanel">
                <TextBlock TextAlignment="Center">
                 OK
                </TextBlock>
                <Image Source="oktransp.GIF" Height="25.704" Width="127.092">
                </Image>
            </StackPanel>
            </Button>
```

Cela donne :


Voir aussi le chapitre sur les ressources.

**Bouton par défaut et Cancel.**

Le bouton par défaut est activé quand l'utilisateur tape sur 'Enter'. Pour indiquer que le bouton est 'par défaut' il faut ajouter l'attribut IsDefault.

```
<Button Name="okButton" Click="okButton_Click" IsDefault="True">OK</Button>
```

Le bouton cancel est activé quand l'utilisateur tape sur 'Echap'. Pour indiquer que le bouton est 'Cancel' il faut ajouter l'attribut IsCancel.

```
<Button Name="cancelButton" IsCancel="True">Cancel</Button>
```

Bouton avec touche d'accès rapide (raccourci).

Il faut autoriser dans le ContentPresenter la reconnaissance des AccesKey puis dans le texte du bouton mettre un '_' avant la lettre désirée :

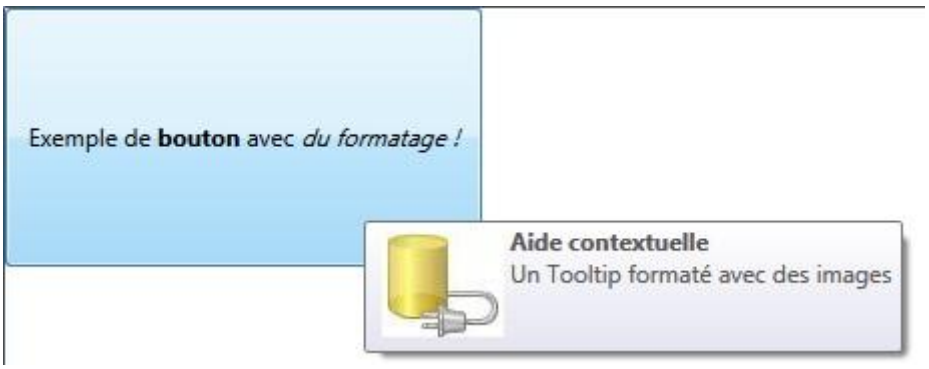
```
<Button Margin="0,0,30,15" Name="Button1" HorizontalAlignment="Right" Width="62"
Height="51" VerticalAlignment="Bottom">
<ContentPresenter RecognizesAccessKey="True" Content=" new _button"/>
</Button>
```

Ici le raccourci 'ALT B' déclenchera un clic sur le bouton.

Autre exemple d'un bouton contenant du texte avec du gras de l'italique et ayant un ToolTip contenant une image et du texte :

```
<Button Margin="0,0,271,183">
            <Button.ToolTip>
                <StackPanel Orientation="Horizontal">
                    <Image Source="c:/test.jpg" Margin="3"/>
                    <TextBlock> <Run FontWeight="Bold">Aide contextuelle</Run> <LineBreak/> Un
 Tooltip formaté avec des images</TextBlock>
                </StackPanel>
            </Button.ToolTip>
            <TextBlock>Exemple de <Run FontWeight="Bold">bouton</Run>
 avec <Run FontStyle="Oblique">du formatage !</Run></TextBlock>
        </Button>
```

Cela donne :




Avec Expression Blend on peut créer des boutons d'aspect plus complexe avec par exemple un gradient de couleur, le code Xaml est beaucoup plus complexe. Exemple correspondant au bouton situé en haut de la page :

```
<Button HorizontalAlignment="Left" Margin="25,47,0,0" VerticalAlignment="Top" Width="159" Height="47"
Name="Button1" RenderTransformOrigin="0.5,0.5">
<Button.Background>
<LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
<GradientStop Color="#FFFFFFFF" Offset="0"/>
<GradientStop Color="#FFF0E5E3" Offset="0.505"/>
<GradientStop Color="#FFC6C5D7" Offset="1"/>
</LinearGradientBrush>
</Button.Background>
<Button.RenderTransform>
<TransformGroup>
<ScaleTransform ScaleX="1" ScaleY="1"/>
```

```
<SkewTransform AngleX="0" AngleY="0"/>
<RotateTransform Angle="0"/>
<TranslateTransform X="0" Y="0"/>
</TransformGroup>
</Button.RenderTransform>
</Button>
```

**Créons un bouton avec du code VB :**

```
Dim myButton As New Button                      'Création du bouton
myButton.Content = "OK"                         'Afficher 'OK' dans le bouton
myButton.Background = Brushes.AliceBlue         'Fond en bleu
grid.Children.Add(myButton)                     'Ajouter le bouton à la grid
AddHandler myButton.Click, AddressOf click      'Indiquer que le clic sur le bouton
                                                'doit exécuter la Sub nommée 'Click'
```

On remarque que comme on crée par code, il faut soi-même écrire la gestion des événements.

Quand on crée le bouton en mode designer, et que l'on double-clique sur le bouton, la routine myButton_Click est automatiquement affichée.

**Utiliser les événements**

L'événement principalement utilisé est le Click() : quand l'utilisateur clique sur le bouton la procédure.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
 System.Windows.RoutedEventArgs)
_Handles Button2.Click
End Sub
```

est exécutée.

Cette procédure contient le code qui doit être exécuté lorsque l'utilisateur clique sur le bouton.

Le bouton peut être sélectionné grâce à un clic de souris, à la touche ENTRÉE ou à la BARRE d'espacement si le bouton a le focus.

## XI-F-2-b - RepeatButton

C'est comme un bouton, mais quand on clique dessus et qu'on tient le bouton gauche appuyé, il déclenche l'événement Click plusieurs fois.

```
<RepeatButton Width="100" DockPanel.Dock="Top" Delay="500" Interval="100" Click="Increase">
</RepeatButton>
```

**Delay** indique à quel moment démarrer l'événement Click.

**Interval** indique l'intervalle entre chaque déclenchement.

**Click=** indique la sub événement déclenchée par le clic.

Exemple de la routine événement qui affiche le nombre d'événements dans un champ text.

```
Sub Increase(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Num = CInt(valueText.Text)
    valueText.Text = ((Num + 1).ToString())
End Sub
```

## XI-F-3 - Les contrôles contenant du texte

Les contrôles permettant de voir ou de modifier du texte sont :

- les Labels ;
- les TextBlock ;
- les TextBox ;
- les RichTextBox ;
- les PasswordBox.

## XI-F-3-a - Les Labels

Permettent d'**afficher du texte**, non modifiable, et qui ne peut pas prendre le focus mais accepte les raccourcis clavier.

Création d'un label en XAML :

```
<Label Name="MonLabel" FontSize="20">
        Ceci est un cours de programmation
</label>
```

Cela affiche le texte "Ceci est un cours de programmation".

Le nom du label "MonLabel" n'est pas affiché, il sert à accéder à l'élément à partir d'un programme.

Dans le code VB la propriété 'Content' permet de modifier le texte affiché.

```
Montexte.Content= "nouveau texte"
```

Les labels permettent de créer des '**raccourcis clavier**'.

Il y a une propriété nommée **Taget** qui définit l'élément qui reçoit le focus lorsque l'utilisateur appuie sur la touche d'accès rapide de l'étiquette :

Target="{Binding ElementName=listbox1}" indique que c'est listbox1 qui recevra le focus.

Pour indiquer la touche d'accès rapide (de raccourci) mettre un '_' avant la lettre désirée dans le texte du label.

Exemple : _Liste des noms

Cela donne :

```
<Label Target="{Binding ElementName=listbox1}" Name="MonLabel" >_Liste des noms</Label>
```

Quand on appuie sur Alt, le L de 'Liste des noms' se souligne indiquant que ALT L est un raccourci clavier.

Si l'utilisateur appuie toujours sur ALT et en même temps sur L, le focus passe sur la ListBox1.

Le label est souvent utilisé à côté d'un autre contrôle de saisie pour lequel il indique la fonction.

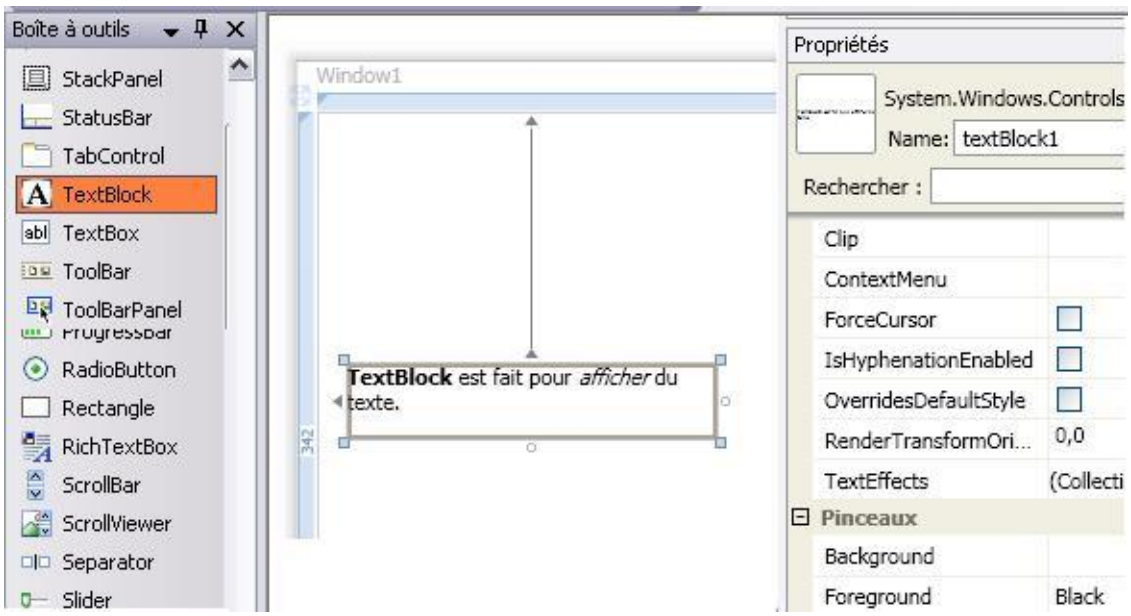
Les dimensions du contrôle ne s'adaptent pas au texte.

## XI-F-3-b - Les TextBlock

Permettent d'**afficher du texte** sur un formulaire sans possibilité de le modifier. Il est bien adapté pour afficher une ou au maximum quelques lignes.

Pas de raccourci clavier ici, mais on peut utiliser le gras, l'italique, les liens hypertextes.

Prendre un TextBlock dans les outils et le mettre dans un formulaire :




Pour mettre rapidement un petit texte dedans, utiliser la propriété Text.

En VB :

```
TextBlock1.Text= "Mon texte"
```

Ici pas de texte enrichi.

Le cadre gris ne sera pas affiché.

Cela donne en Xaml :

```
<TextBlock>
Mon texte
</TextBlock>
```

On peut utiliser les balises Bold (gras), Italic (Italique) pour 'enrichir' le texte.

```
<TextBlock Name="textBlock1" TextWrapping="Wrap">
<Bold>TextBlock</Bold> est fait pour <Italic>afficher</Italic> du texte.
</TextBlock>
```

On peut rajouter plein d'attributs : Background="AntiqueWhite" TextAlignment="Center"

En VB pour mettre du texte enrichi, on utilise la collection InLine :

```
Dim textBlock1 = new TextBlock()
textBlock1.TextWrapping = TextWrapping.Wrap
textBlock1.Background = Brushes.AntiqueWhite
textBlock1.TextAlignment = TextAlignment.Center
```

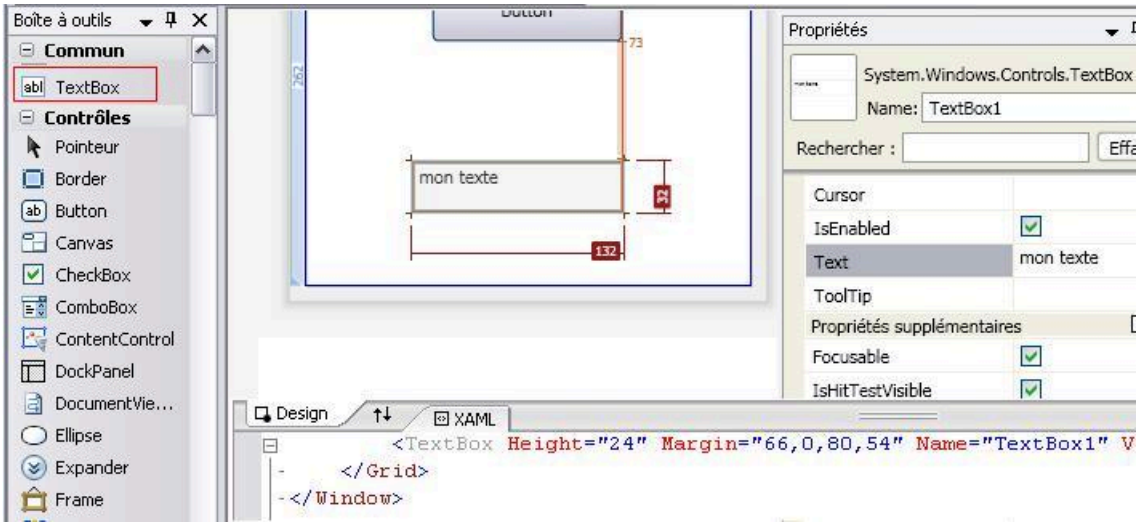
```
textBlock1.Inlines.Add(new Bold(new Run("TextBlock")))
textBlock1.Inlines.Add(new Run(" est fait pour "))
textBlock1.Inlines.Add(new Italic(new Run("afficher")))
textBlock1.Inlines.Add(new Run(", du texte "))
```

Les dimensions du contrôle ne s'adaptent pas au texte.

## XI-F-3-c - Les TextBox

Permettent d'afficher du texte, il est modifiable par l'utilisateur.

La police de caractères, sa taille, la couleur, l'enrichissement des caractères affectent la totalité du texte. Il n'est pas possible d'enrichir (gras, italique…) une partie du texte seulement.




Créons un TextBox en XAML :

```
<TextBox></TextBox>
```

Donnons-lui un nom :

```
<TextBox Name="TextBox1"></TextBox>
```

Mettons un texte dedans :

```
<TextBox Name="TextBox1">Ceci est le texte</TextBox>
```

En VB :

```
Dim TextBox1 As New TextBox
```

**TextBox1.Text** contient le texte affiché dans la textBox. **AppendText** permet d'ajouter du texte.

**TextBox1.IsReadOnly=True** interdit les modifications du texte.

**TextBox1.MaxLines** et **TextBox1.MinLines** permettent de définir si le TextBox est multiligne. (S'ils sont égaux à 1, on a une seule ligne).

**TextBox1.AcceptEnter=True** et **TextBox1.AcceptTab=True** autorisent respectivement le passage à la ligne quand on tape 'Enter' et l'insertion de tabulation. Si AcceptEnter=False, on ne peut saisir qu'une seule ligne.

**VerticalScrollBarVisibility=True** affiche une scroll bar verticale.

**TextBox1.MaxLength** permettent de définir le nombre de caractères. 0 pour saisie illimitée.

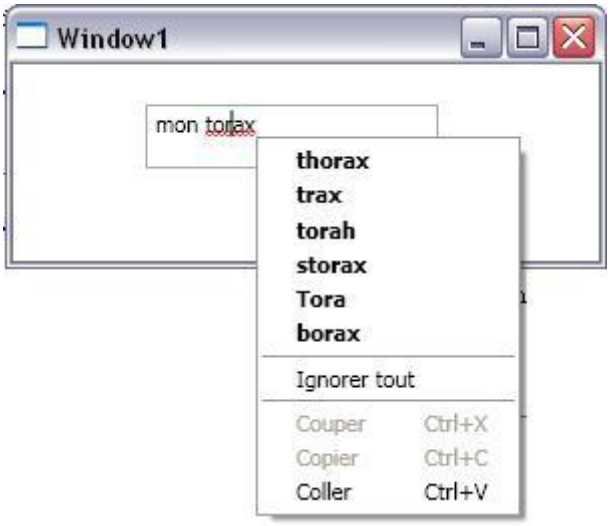
**TextBox1.Clear** efface le texte.

**TextBox1.LineCount** donne le nombre de lignes **TextBox1.GetLineText(2)** indique le contenu de la ligne numéro 2 (sans oublier que la première ligne est la ligne 0).

Si l'utilisateur a sélectionné du texte, il est dans **TextBox1.SelectedText**.

**TextBox1.SelectionStart**, **TextBox1.SelectionLength** indique la position du premier caractère sélectionné (le premier caractère du texte étant le caractère 0) et le nombre de caractères sélectionnés. On peut utiliser ces propriétés pour sélectionner du texte avec du code ou utiliser TextBox1.Select(3, 2), il existe enfin SelectAll.

Si **TextBox1.SpellCheck.IsEnabled = True** le correcteur d'orthographe est opérationnel : il souligne les fautes et un clic droit permet de voir une liste des corrections possibles :




**TexBox.LineDown**, **LineUp** permettent de faire défiler le texte en bas ou en haut. Il existe aussi **LineLeft** et **LineRight** PageDown, PagUp, PageRight, PageLeft.

**TexBox1.ScrollToHome ScrollToEnd**, **ScrollToLine** permettent de déplacer le texte visible.

On peut aussi utiliser **TexBox1.Undo**, **Redo**, **Cut**, **Copy**, **Paste**.

Si le texte est modifié, cela déclenche :

```
Private Sub TextBox1_TextChanged(ByVal sender As System.Object, ByVal e As _
System.Windows.Controls.TextChangedEventArgs) _
Handles TextBox1.TextChanged
End Sub
```

Quand l'utilisateur frappe une touche, cela déclenche les événements **KeyDown** puis **KeyUp** (pas de keyPress !!).

```
Private Sub TextBox1_KeyUp(ByVal sender As Object, ByVal e As System.Windows.Input.KeyEventArgs)
 _
```

```
Handles TextBox1.KeyUp
    If e.Key = Key.OemComma Then…
End Sub
```

On note que la **Sub KeyUp** a un paramètre **e** de type System.Windows.Input.KeyEventArgs qui a la propriété **Key** qui contient la touche tapée. Malheureusement e.Key est en lecture seule!! on ne peut donc pas modifier le caractère tapé !!

## XI-F-3-d - Les RichTextBox

Rich Text veut dire 'Texte enrichi'.

Le contrôle RichTextBox permet d'afficher, d'entrer et de manipuler du texte mis en forme. Il effectue les mêmes tâches que le contrôle TextBox, mais il peut également afficher des polices, des couleurs pour une partie du texte et des liens, charger du texte et des images incorporées à partir d'un fichier, ainsi que rechercher des caractères spécifiques.

Toutes les propriétés de correction d'orthographe des TextBox s'appliquent aux RichTextBox.

Les RichTextBox en WPF n'affichent pas du RTF comme dans les WindowsForms, mais des 'FlowDocument' qui sont en XML.

En Xaml c'est simple, un FlowDocument est de la forme :

```
<FlowDocument>
</FlowDocument>
```

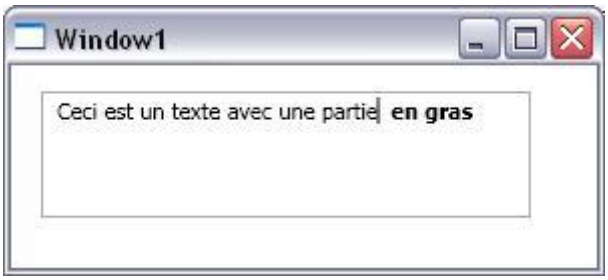
Dedans il y a des 'Paragraph', dans les paragraphes il y a le texte à afficher et l'enrichissement à l'aide de balise (Bold par exemple).

```
<FlowDocument>
<Paragraph>
Ceci est un texte avec une partie
<Bold>en gras</Bold>
</Paragraph>
</FlowDocument>
```

On le met dans une RichTextBox :

```
<RichTextBox>
<FlowDocument>
<Paragraph>
Ceci est un texte avec une partie
<Bold>en gras</Bold>
</Paragraph>
</FlowDocument>
</RichTextBox>
```

Cela donne :




Le texte est éditable, modifiable.

Dans le code, en VB, cela se complique.

Il faut instancier un FlowDocument, un Paragraph, ajouter une ligne au paragraphe puis ajouter le paragraphe au Blocks du FlowDocument. Enfin il faut affecter à la propriété Document du RichTextBox le FlowDocument.

```
Dim doc As New FlowDocument
Dim para As New Paragraph
para.Inlines.Add("Ceci est un texte")
doc.Blocks.Add(para)
RichTextBox1.Document = doc
```

Si on veut ajouter l'enrichissement, il faut instancier un Bold, y ajouter le texte, ajouter au paragraphe puis au Block !!

```
Dim doc As New FlowDocument
Dim para As New Paragraph
para.Inlines.Add("Ceci est un texte avec une partie")
Dim b As New Bold
b.Inlines.Add("en gras")
para.Inlines.Add(b)
doc.Blocks.Add(para)
RichTextBox1.Document = doc
```

Sélectionner la totalité du texte :

```
Dim tr As New TextRange(RichTextBox1.Document.ContentStart, RichTextBox1.Document.ContentEnd)
MessageBox.Show(tr.Text)
```

C'est du texte non enrich i!!

**load save print**

Créons un RichTextBox et 3 boutons pour enregistrer lire dans un fichier le texte ou pour l'imprimer.

```
<RichTextBox Name="richTB">
<FlowDocument>
  <Paragraph> <Run>Paragraph 1</Run> </Paragraph>
</FlowDocument>
</RichTextBox>
  <Button Click="SaveRTBContent">Enregistre texte </Button>
  <Button Click="LoadRTBContent">Charge texte</Button>
  <Button Click="PrintRTBContent">Imprime texte</Button>
```

Voici le code VB (donné par Microsoft) contenant les 3 routines permettant les 3 actions.

```
Imports System
Imports System.IO
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Documents
```

```
Imports System.Windows.Media
Namespace SDKSample

    Public Partial Class SaveLoadPrintRTB
        Inherits Page
       ' Handle "Save RichTextBox Content" button click.

        Private Sub SaveRTBContent(ByVal sender As Object, ByVal args As RoutedEventArgs)
             ' Send an arbitrary URL and file name string specifying
            ' the location to save the XAML in.

            SaveXamlPackage("C:\test.xaml")
        End Sub



       ' Handle "Load RichTextBox Content" button click.

        Private Sub LoadRTBContent(ByVal sender As Object, ByVal args As RoutedEventArgs)
            ' Send URL string specifying what file to retrieve XAML
            ' from to load into the RichTextBox.

            LoadXamlPackage("C:\test.xaml")

        End Sub



        ' Handle "Print RichTextBox Content" button click.

        Private Sub PrintRTBContent(ByVal sender As Object, ByVal args As RoutedEventArgs)

            PrintCommand()

        End Sub



       ' Save XAML in RichTextBox to a file specified by _fileName

        Private Sub SaveXamlPackage(ByVal _fileName As String)
            Dim range As TextRange
            Dim fStream As FileStream
            range = New TextRange(richTB.Document.ContentStart, richTB.Document.ContentEnd)
            fStream = New FileStream(_fileName, FileMode.Create)
            range.Save(fStream, DataFormats.XamlPackage)
            fStream.Close()
        End Sub



       ' Load XAML into RichTextBox from a file specified by _fileName

        Private Sub LoadXamlPackage(ByVal _fileName As String)
            Dim range As TextRange
            Dim fStream As FileStream
            If File.Exists(_fileName) Then
            range = New TextRange(richTB.Document.ContentStart, richTB.Document.ContentEnd)
            fStream = New FileStream(_fileName, FileMode.OpenOrCreate)
            range.Load(fStream, DataFormats.XamlPackage)
            fStream.Close()
        End If

        End Sub



       ' Print RichTextBox

        Private Sub PrintCommand()
        Dim pd As New PrintDialog()
        If (pd.ShowDialog() = True) Then
```

```
        'use either one of the below
        pd.PrintVisual(TryCast(richTB, Visual), "printing as visual")
        pd.PrintDocument((DirectCast(richTB.Document,
 IDocumentPaginatorSource).DocumentPaginator), _
        "printing as paginator")
        End If
        End Sub

    End Class

End Namespace
```

## XI-F-3-e - Les PasswordBox

Permet de saisir un mot de passe.

La propriété **PasswordChar** détermine le caractère affiché à la place des caractères tapés.

En XAML :

```
<PasswordBox Name="pwdBox" MaxLength="64" PasswordChar="#" PasswordChanged="PasswordChanged" />
```

Ici le fait de taper un mot de passe déclenche la **Sub PasswordChanged**.

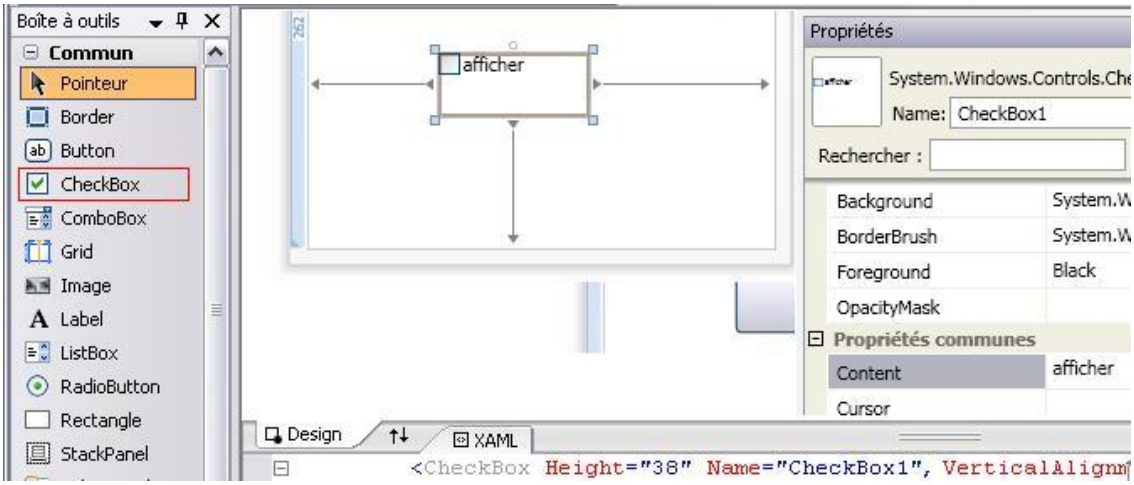
En VB on récupère le mot de passe dans :

```
pwdBox.Password
```

## XI-F-4 - Les cases à cocher et RadioButton

### XI-F-4-a - Case à cocher

Créons une case à cocher :




C'est une CheckBox. L'utilisateur peut la cocher ou non en cliquant dessus.

Dans le designer, la propriété **Content** contient le texte à afficher à côté de la case.

Les événements les plus utiles sont :**Checked** et **Unchecked**.

```
Private Sub CheckBox1_Checked(ByVal sender As System.Object, ByVal e As
 System.Windows.RoutedEventArgs)
_Handles CheckBox1.Checked

End Sub
```

```
Private Sub CheckBox1_Unchecked(ByVal sender As System.Object, ByVal e As
 System.Windows.RoutedEventArgs)
_Handles CheckBox1.Checked

End Sub
```

Il y a aussi l'événement **Click** qui est exécuté lorsqu'on clique sur la case.

```
Private Sub CheckBox1_Click(ByVal sender As Object, ByVal e As System.Windows.RoutedEventArgs)
_Handles CheckBox1.Click

    If CheckBox1.IsChecked = True Then MsgBox("c'est coché&#8218; maintenant")

End Sub
```

On note que la propriété **IsChecked** permet de voir si la case est cochée ou non.

## XI-F-4-b - RadioButton

Les RadioButton peuvent être cochés ou non.


Ils sont généralement regroupés pour offrir aux utilisateurs un choix unique parmi plusieurs options, un seul bouton à la fois peut être coché. Si on clique sur un RadioButton dans un groupe, on le sélectionne, cela désélectionne les autres.

Vous pouvez regrouper des contrôles RadioButton en les plaçant dans un parent commun ou en leur attribuant un nom de groupe.

Quand un RadioButton est sélectionné, l'événement Checked est déclenché. Comme le montre l'exemple de code suivant, si votre application doit prendre une action quand la sélection de RadioButton change, vous pouvez gérer l'événement Checked.

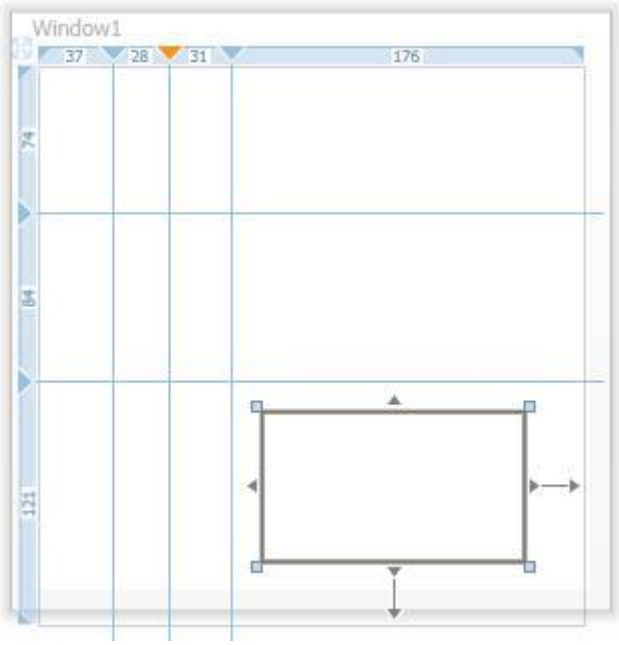
```
<RadioButton Name="MyRadioBUtton"
             IsChecked="True"
             Checked="Myroutine"
             GroupName="MyGroupe">
 Texte
</RadioButton>
```

## XI-F-5 - Les Listes

Une **ListBox** est un contrôle WPF qui contient une collection de **ListBoxItem**. Chaque ListBoxItem a une propriété '**Content**'.

Créons une listBox.

On va la chercher dans la boite à outils et on la dépose ici sur une grid.




Dans la fenêtre XAML, cela donne :

```
<ListBox  Name="ListBox1" />
```

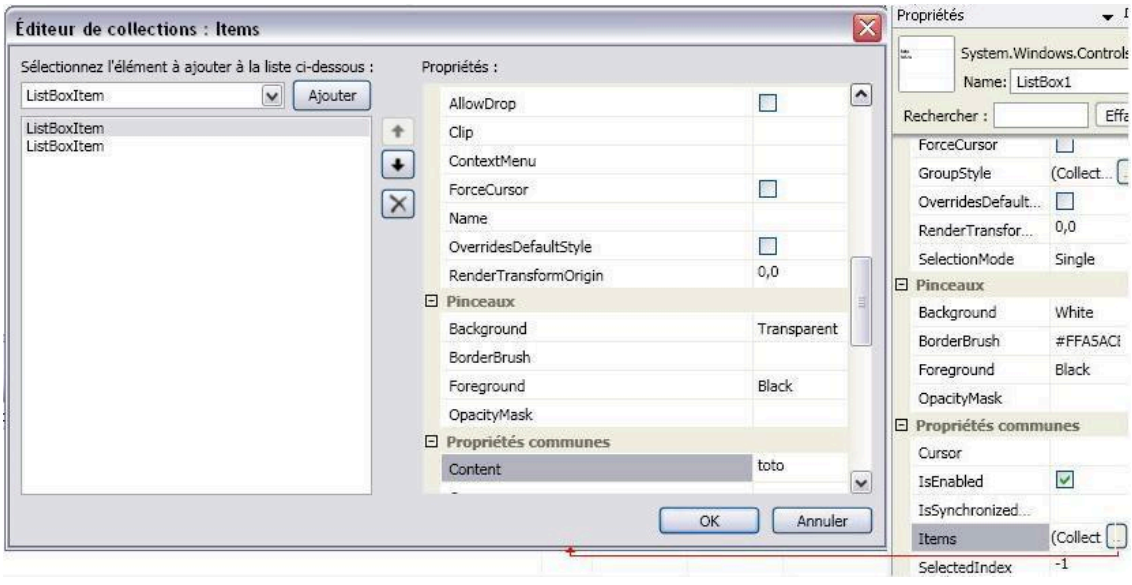
Pour être plus complet, il y a aussi les attributs donnant la position de la ListBox dans la Grid et les marges autour :

```
<ListBox Grid.Column="3" Grid.Row="2" Margin="22,26,21,19" Name="ListBox1" />
```

Grid.Column="3" Grid.Row="2" indiquent les coordonnées de la cellule de la grid.

On peut ajouter des éléments dans la ListBox en mode conception dans le Designer.

Dans la fenêtre "propriétés" de la ListBox cliquer sur le bouton en face de Items. Cela ouvre une fenêtre permettant d'ajouter des éléments à la ListBox



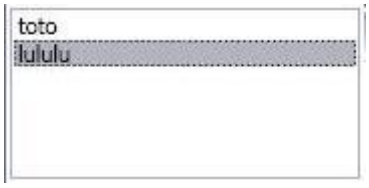

Le bouton "Ajouter" permet d'ajouter des ListBoxItem, chacun ayant des propriétés. La propriété "Content" indique ce que contient chaque ListBoxItem, ici le texte qui sera affiché.

Cela modifie le code XAML en ajoutant les ListBoxItem.

```
<ListBox Grid.Column="3" Grid.Row="2" Margin="22,26,21,19" Name="ListBox1">
<ListBoxItem>toto</ListBoxItem>
<ListBoxItem>lululu</ListBoxItem>
</ListBox>
```

Pour créer une ListBox puis ajouter des éléments à la ListBox dans le code VB :

```
Private Sub Window1_Loaded
Dim ListBox1 As ListBox
ListBox1.Items.Add("toto")
ListBox1.Items.Add("lulu")
Grid.Children.Add(ListBox1)
End Sub
```




La ListBox à des procédures événements dans le code VB. 'MouseDoubleClick' par exemple :

```
Private Sub ListBox1_MouseDoubleClick(ByVal sender As Object, ByVal e As _
System.Windows.Input.MouseButtonEventArgs) _
Handles ListBox1.MouseDoubleClick

'ici on affiche dans un messagebox l'item sur lequel l'utilisateur a cliqué.

MsgBox(ListBox1.SelectedItem.ToString)

End Sub
```

On remarque que l'élément sélectionné est SelectedItem.

Il y a aussi la procédure :

```
Private Sub ListBox1_SelectionChanged
```

Comment modifier la couleur de fond de l'élément sélectionné ?

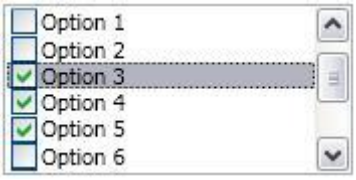
```
<SolidColorBrush x:Key="{x:Static SystemColors.HighlightBrushKey}" Color="Green" />
```

**Mettre des boutons, des cases à cocher dans une listBox**

Les ListBoxItem d'une ListBox ont une propriété nommée 'Content' qui peut contenir un objet : du texte, mais aussi une CheckBox, c'est ça la puissance des WPF.

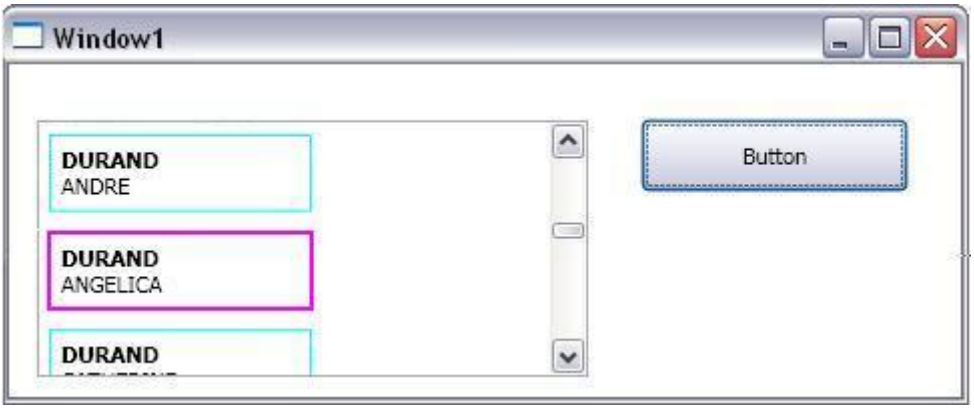
Les CheckedListBox n'existent pas en WPF. On va les créer dans un ListBox1 :

```
Private Sub Window1_Loaded
For i As Integer = 0 To 10
Dim chk As New CheckBox
chk.content = "Option " + i.ToString
Me.ListBox1.Items.Add(chk)
Next
End Sub
```




De la même manière ; on peut créer une liste de boutons…

On peut aussi modifier fortement l'affichage des éléments dans un ListBox. Ici par exemple plutôt que d'afficher bêtement une liste de nom, je vais les afficher dans un cadre coloré en fonction du sexe grâce à un modèle de données (Data Template). Voir le chapitre sur les ressources




**ListBox Horizontale**

Un ListBox a une propriété ItemsPanel qui permet de définir un ItemsPanelTemplate qui contrôle la disposition des éléments du ListBox. Une méthode consiste à créer un style ListBox et à définir la propriété ItemsPanel.

À mettre dans les ressources de la fenêtre.

```
<Style TargetType="ListBox">
  <Setter Property="ItemsPanel">
    <Setter.Value>
      <ItemsPanelTemplate>
        <StackPanel Orientation="Horizontal"
                    VerticalAlignment="Center"
                    HorizontalAlignment="Center"/>
      </ItemsPanelTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

## XI-F-6 - Les boites de dialogue

Les projets WPF permettent d'utiliser des MessageBox et InputBox et des PrintDialog, pour le reste il faut ruser, car nativement il n'y a rien d'autre !! (erreur de jeunesse de WPF ?)

## XI-F-6-a - MessageBox

Fenêtre de dialogue permettant de fournir un message, un titre, des boutons, une image (icône) et éventuellement de retourner une information.

Pas de problème dans un projet WPF, MessageBox appartient à System.Windows.Controls.

Ressemble au MessageBox des 'Windows Forms' sauf qu'on parle d'image et non d'icône, qu'il y a un argument de moins (celui de l'aide) et que les paramètres n'ont pas le même nom parfois !!

Dans le code VB, on utilise la méthode Show pour afficher la boite.

```
MessageBox.Show("mon text", "titre", MessageBoxButton.YesNo, MessageBoxImage.Exclamation, _
MessageBoxResult.OK)
```

On doit fournir le texte à afficher, on peut aussi fournir le titre dans la barre, le type de bouton, le type d'image et le bouton par défaut, une option d'affichage.

Exemple

Afficher simplement un texte d'information :

```
MessageBox.Show("Error")
```


Affiche une box avec le message et un bouton 'Error', pas de valeur de retour.

Afficher une question et voir sur quel bouton l'utilisateur a cliqué :

```
Dim rep As String = MessageBox.Show("Annuler", "Attention", MessageBoxButton.OKCancel,
_MessageBoxImage.Exclamation, MessageBoxResult.OK, MessageBoxOptions.RightAlign)
```


Le nom du bouton cliqué par l'utilisateur (de type MessageBoxResult) est retourné dans Rep qui est une String.

On peut ensuite tester sa valeur :

```
If rep = MessageBoxResult.OK Then

…

End If
```

**Paramètres**

*TexteAAfficher*

Obligatoire. Expression String affichée comme message de la boite de dialogue (longueur maximale 1 024 caractères comme dans les WIndows Forms). N'oubliez pas d'insérer un retour chariot si le texte est long, cela crée 2 lignes.

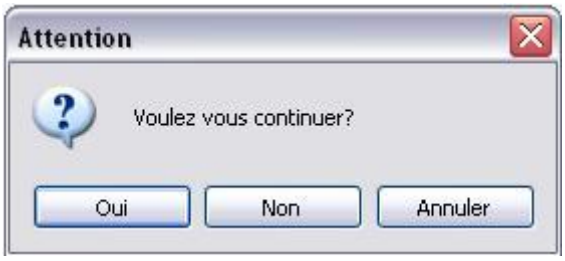
*Titre*

Expression String affichée dans la barre de titre de la boite de dialogue. Si l'argument Titre est omis, il n'est rien affiché (voir premier exemple).

*TypeBouton*

- le nombre et le type de boutons à afficher (MessageBoxButton sans 's' maintenant):

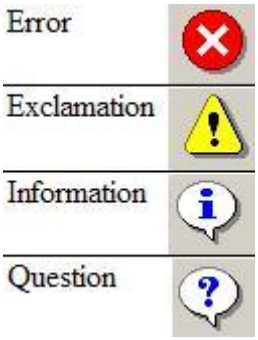
- MessageBoxButton.OK = Un seul bouton 'OK' ;
- MessageBoxButton.YesNo = Deux boutons 'Oui' 'Non' ;
- MessageBoxButton.OkCancel = 'OK' et 'Annuler' ;
- MessageBoxButton.YesNoCancel :




*Image*

- l'icône à utiliser

- MessageBoxImage.Error ;
- MessageBoxImage.Exclamation ;
- MessageBoxImage.Information ;
- MessageBoxImage.Question :




et aussi

- MessageBoxImage.Asterisk ;
- MessageBoxImage.Hand ;
- MessageBoxImage.Warning ;
- MessageBoxImage.Stop ;
- MessageBoxImage.None.

*L'identité du bouton par défaut.*

- MessageBoxResult.Ok ;
- MessageBoxResult.Cancel.

*Les options*

MessageBoxOptions.RightAlign.

Comme d'habitude, il suffit de taper MessageBox.Show (pour que VB proposer les paramètres).

**Retour de la fonction**

Retourne une constante de type MessageBoxResult (et non plus DialogResult comme dans les Windows Forms) qui indique quel bouton a été pressé.

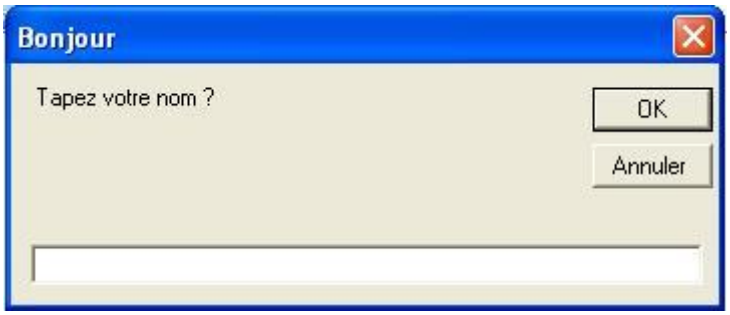
- MessageBoxResult.Yes ;
- MessageBoxResult.No ;
- MessageBoxResult.Cancel ;
- MessageBoxResult.Retry ;
- MessageBoxResult.Ok.

## XI-F-6-b - InputBox

Pose une question, retourne une String contenant la réponse tapée par l'utilisateur.

Pas de problème dans un projet WPF, InputBox appartient à System.Windows.Controls.

```
Dim Nom As String

Nom = InputBox("Bonjour", "Tapez votre nom ?")
```




## XI-F-6-c - PrintDialog

Pas de problème dans un projet WPF, PrintDialog appartient à System.Windows.Controls. (Ne pas confondre avec PrintDialog de System.Windows.Forms). Cette Classe va plus loin que l'ancienne, car elle permet donc d'afficher la boite de dialogue, mais aussi d'imprimer. On peut faire l'un ou l'autre ou les 2. Il faut instancier une PrintDialog qui permet d'afficher la boite de dialogue 'Imprimer' avec ShowDialog et/ou d'imprimer avec la méthode PrintDocument.

Exemple

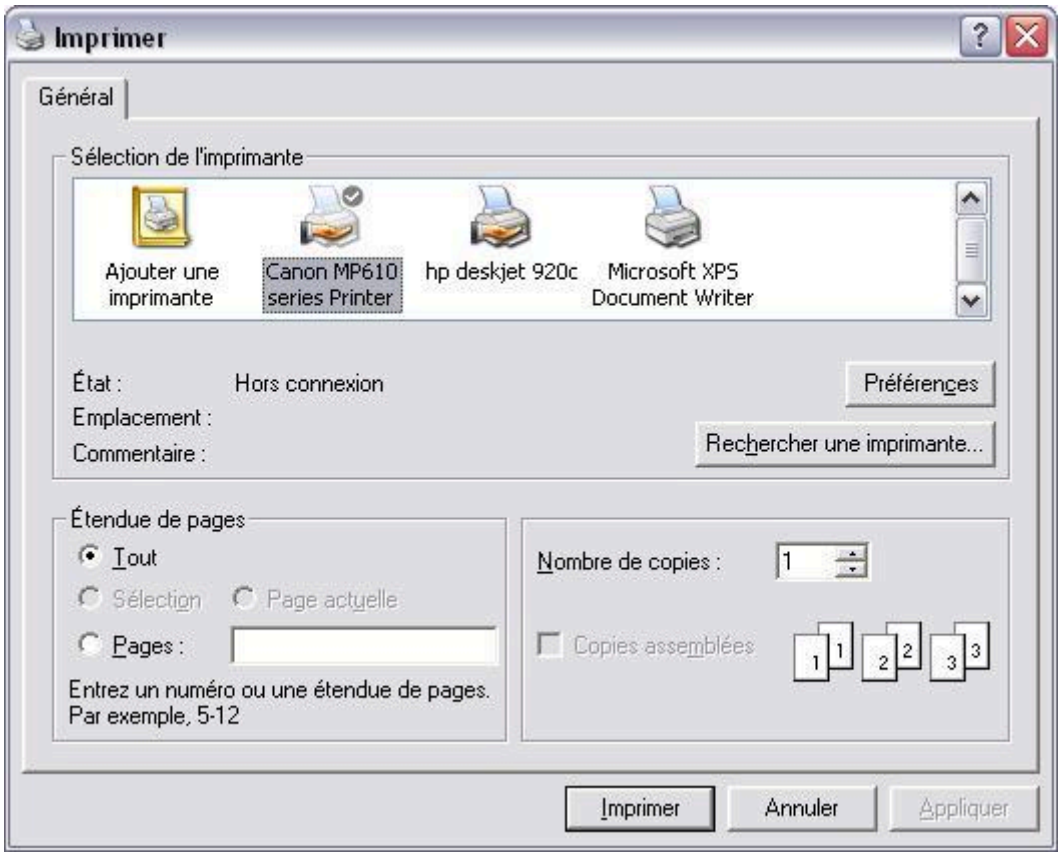
On crée un bouton ayant comme texte 'Imprimer' en XAML.

```
<Button Height="33" Margin="41,0,68,12" Name="Button1" >Imprimer</Button>
```

Dans le code, on affiche la boite de dialogue d'impression.

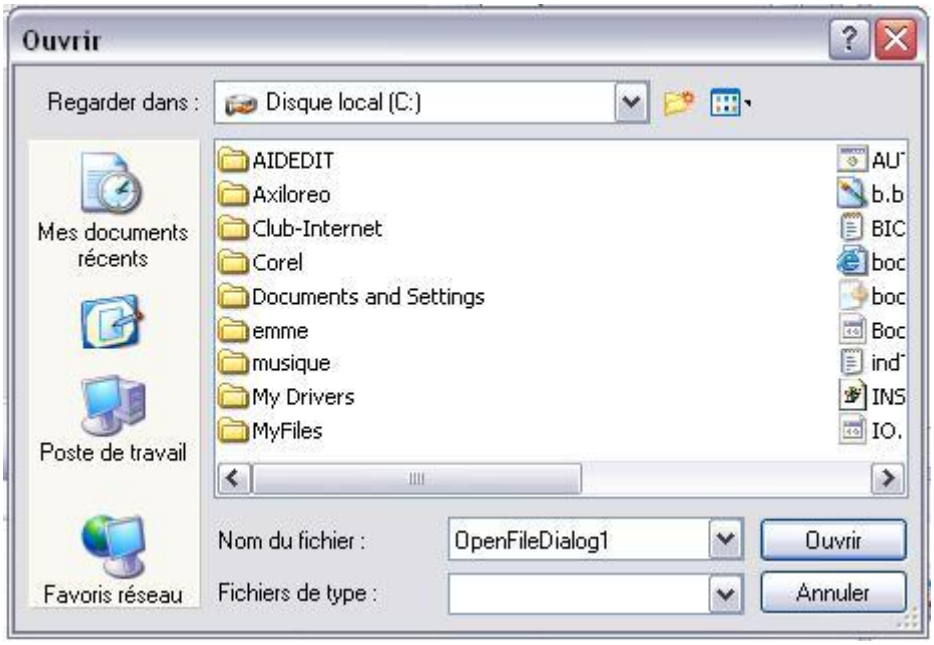
```
Private Sub Button1_Click
```

```
Dim pDialog As New PrintDialog()

pDialog.PageRangeSelection = PageRangeSelection.AllPages

pDialog.UserPageRangeEnabled = True

'Affiche la PrintDialog, retourne True si l'utilisateur a cliqué sur 'Imprimer'…

Dim print As Boolean = pDialog.ShowDialog()

If print = True Then

'Ici routine d'impression avec pDialog.PrintDocument(,)

End If

End Sub
```




## XI-F-6-d - OpenFileDialog

Fenêtre de dialogue permettant de choisir un fichier à ouvrir :




Elle n'existe pas en Wpf on va donc utiliser la boite de dialogue standard de Windows.

Il faut donc importer l'espace Microsoft.Win32 en haut du module pour pouvoir l'utiliser !!

```
Imports Microsoft.Win32
```

Puis, par exemple quand on clique sur le bouton Buttom1, cela ouvre la boite de dialogue permettant de choisir un fichier.

```
Private Sub Button1_Click

Dim dlg As New OpenFileDialog

dlg.ShowDialog()

Dim fichier As String = dlg.FileName

End Sub
```

Le nom du fichier à ouvrir est dans la propriété FileName.

On peut ajouter un filtre et un chemin initial :

```
Dim dlg As New OpenFileDialog()

' Filter by Word Documents
dlg.Filter = "Word Documents|*.doc"
dlg.ItinialDirectory = "c:\"
dlg.ShowDialog()
```

## XI-F-6-e - SaveFileDialog

Fenêtre de dialogue permettant de choisir un nom de fichier d'enregistrement. Elle n'existe pas en Wpf. Il faut importer l'espace Microsoft.Win32 en haut du module pour pouvoir l'utiliser !!

```
Imports Microsoft.Win32
```

Puis, par exemple quand on clique sur le bouton Buttom1, ouvrir la boite de dialogue permettant de choisir un fichier.

```
Private Sub Button1_Click

Dim dlg As New SaveFileDialog

dlg.ShowDialog()

Dim fichier As String = dlg.FileName

'Ici routine d'enregistrement à écrire

End Sub
```

## XI-F-6-f - FolderBrowserDialog

Fenêtre de dialogue permettant de choisir un répertoire :


Elle n'existe pas en Wpf et n'est pas dans Win32. Pour cela on va faire de l'interopérabilité entre WPF et Windows Forms.

Il faut ajouter dans les références (menu 'Projet', 'Propriétés de…'; onglet 'Références' bouton 'Ajouter') System.Windows.Forms.

Il faut importer l'espace System.Windows.Forms en haut du module pour pouvoir l'utiliser !!

On aurait pu écrire : Imports System.Windows.Forms, mais comme certains éléments sont dans plusieurs espaces de noms et entrainent des ambiguïtés, on va importer avec un alias (swf qui sera l'alias de System.Windows.Forms) :

```
Imports swf = System.Windows.Forms
```

Puis, par exemple quand on clique sur le bouton Buttom1, on ouvre la boite de dialogue permettant de choisir un répertoire.

```
Private Sub Button1_Click

Dim dlg As New swf.FolderBrowserDialog

dlg.ShowDialog()
```

```
Dim fichier As String = dlg.SelectedPath

MessageBox.Show(fichier)

End Sub
```

De la même manière, on peut ouvrir

FontDialog ;

PrintPreviewDialog ;

ColorDialog.

Ce n'est pas du WPF !!

## XI-F-7 - Les Menus et ToolBar

La balise *Menu* permet de créer un menu, les *MenuItem* permettent d'ajouter les items des menus et sous-menus.
Le Header permet de donner le libellé du menu.

```
<Grid>

        <Menu>
            <MenuItem Header="Calcul">
                <MenuItem Header="Calcul somme"/>
                <Separator/>
                <MenuItem Header="Calcul différence"/>
            </MenuItem>
        </Menu>



      …
    </Grid>
```

Ici on a un menu 'Calcul' et 2 sous-menus 'Calcul somme' et 'Calcul différence'.
Le fait d'ajouter un MenuItem dans un MenuItem crée un sous-menu.
On remarque que ,bien sur ,le menu est dans le conteneur principal, ici une grid.

La balise 'Separator' permet d'ajouter une ligne de séparation entre les sous-menus.

On peut ajouter une touche de raccourci :

```
<MenuItem Header="_Find" InputGestureText="Ctrl+F"/>
```

Il faut gérer l'événement 'Click' du menu pour que quand l'utilisateur du logiciel clique sur un item cela exécute une routine :

```
<Menu>

            <MenuItem Header="Calcul">
                <MenuItem Header="Calcul total"
                Click="Total_Click"/>
            </MenuItem>

</Menu>
```

Quand on clique sur le sous-menu 'Calcul total' la Sub Total_Click s'exécute (elle est dans le code VB).

```
Private Sub Total_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
        MsgBox("hello")
 End Sub
```

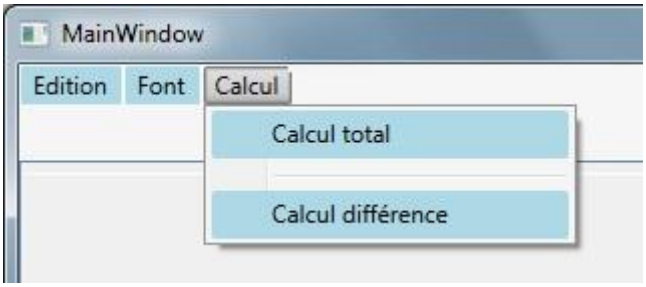
On peut ajouter facilement des sous-menus couper, copier, coller (grâce à 'Command') et des sous-menus coché/décoché (grâce à 'IsCheckable="True"') :

```
<Menu>
        <MenuItem Header="_Edition">
            <MenuItem Command="ApplicationCommands.Copy"/>
            <MenuItem Command="ApplicationCommands.Cut"/>
            <MenuItem Command="ApplicationCommands.Paste"/>
        </MenuItem>
        <MenuItem Header="_Font">
            <MenuItem Header="_Gras" IsCheckable="True"
              Checked="Bold_Checked"
              Unchecked="Bold_Unchecked"/>
            <Separator/>
            <MenuItem Header="_Italic" IsCheckable="True"
              Checked="Italic_Checked"
              Unchecked="Italic_Unchecked"/>
            </MenuItem>
            <MenuItem Header="Calcul">
                <MenuItem Header="Calcul total"
              Clic="Total_Click"/>
                <Separator/>
                <MenuItem Header="Calcul différence" IsCheckable="True"
              Checked="Italic_Checked"
              Unchecked="Italic_Unchecked"/>
            </MenuItem>

        </Menu>
```

Il faut des Sub nommées 'Bold_Checked', 'Bold_Unchecked', 'Italic_Checked'.

On peut ajouter un style au menu pour en modifier l'aspect : ici grâce à un Style, on va modifier la couleur de fond de tous les MenuItem :

```
<Window x:Class="MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="525">
<Window.Resources>
        <Style  TargetType="{x:Type MenuItem}">
            <Setter Property = "Background" Value= "LightBlue"/>
        </Style>
</Window.Resources>
```



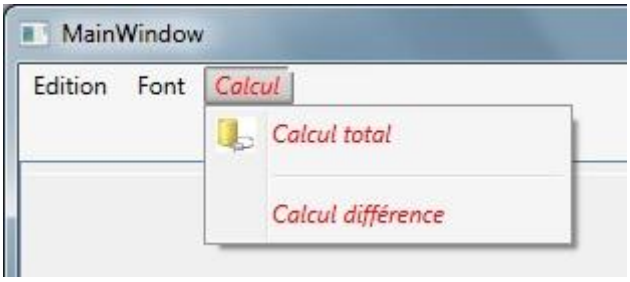

On peut modifier le Style quand un événement se déclenche : si le 'Target' est le survol du MenuItem, mettre le texte en rouge :

```
<Window.Resources>
        <Style  TargetType="{x:Type MenuItem}">
            <Style.Triggers>
                <Trigger Property="MenuItem.IsMouseOver" Value="true">
```

```
                    <Setter Property = "Foreground" Value="Red"/>
                    <Setter Property = "FontSize" Value="12"/>
                    <Setter Property = "FontStyle" Value="Italic"/>
                </Trigger>
            </Style.Triggers>
        </Style>

    </Window.Resources>
```

Comment ajouter une image à un MenuItem ?

```
<MenuItem Header="Calcul total"
              Click="Total_Click">
        <MenuItem.Icon>

    <Image Source="c:/test.jpg" Width="20" Height="20" ToolTip="Calcul"
 ToolTipService.HasDropShadow="True"/>

        </MenuItem.Icon>
</MenuItem>
```



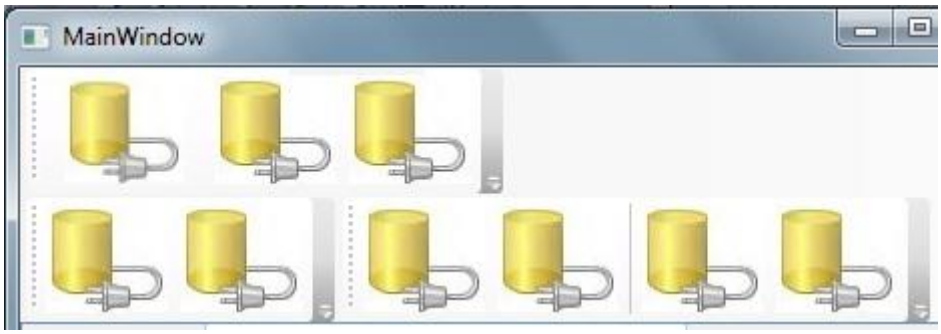

En plus on a mis un ToolTip sur l'image !

Pour les ToolBar on utilise la balise ToolBarTray qui permet de positionner les ToolBar. La position de chaque ToolBar est dépendante de la Band (de haut en bas) et de la BandIndex qui permet de définir des groupes dans la band. Dans chaque ToolBar on met des boutons.

```
        <Grid>
         <ToolBarTray >
            <ToolBar Band="1" BandIndex="1">
                <Button Click="Total_Click" ToolTip="Open">
                    <Image Source="C:/test.JPG" Height="52" Width="78" />
                </Button>
                <Button>
                    <Image Source="C:/test1.JPG" />
                </Button>
                <Button>
                    <Image Source="c:/test2.jpg" />
                </Button>
            </ToolBar>
            <ToolBar Band="2" BandIndex="1">
                <Button>
                    <Image Source="c:/test3.jpg" />
                </Button>
                <Button>
                    <Image Source="c:/test4.jpg" />
                </Button>
            </ToolBar>
            <ToolBar Band="2" BandIndex="2">
                <Button>
                    <Image Source="c:\test.jpg" />
                </Button>
                <Button>
                    <Image Source="c:/test.jpg" />
                </Button>
                <Separator/>
```

```
                <Button>
                    <Image Source="c:/test.jpg" />
                </Button>
                <Button>
                    <Image Source="c:/test.jpg" />
                </Button>
            </ToolBar>
        </ToolBarTray>
    </Grid>
```

Ici on a 2 bands, 2 groupes de boutons dans la seconde band, on peut ajouter des séparators.




Dans le premier bouton (uniquement pour ne pas alourdir l'exemple) on gère l'événement Click et on ajoute un ToolTip.

## XI-F-8 - Les DataGrid

Les datagrid sont des grilles du style tableur, où on peut afficher des données.

## XI-F-8-a - Le DataGrid des WindowsForms

On peut utiliser le DataGrid des WindowsForms. C'est un bel exemple de l'intégration d'un contrôle non WPF dans l'interface WPF.

Dans le code Xaml de la Window, il faut importer les espaces de noms :

```
xmlns:wfint="clr-namespace:System.Windows.Forms.Integration;assembly=WindowsFormsIntegration"
xmlns:wf="clr-namespace:System.Windows.Forms;assembly=System.Windows.Forms"
```

Ensuite, toujours en Xaml, on met la DataGrid :

```
<wfint:WindowsFormsHost Height="100" Width="200">
   <wf:DataGrid x:Name="myDataGridWF"><wf:DataGrid>
<wfint:WindowsFormsHost>
```

Enfin, on peut l'utiliser en VB :

```
DataTable dt = new DataTable()
dt.Columns.Add("Colonne 1", typeof(string))
dt.Columns.Add("Colonne 2", typeof(string))
dt.Rows.Add("Ligne1")
myDataGridWF.DataSource = dt
```

## XI-F-8-b - Le DataGrid WPF

Dans les WPF du Framework 3.5, pas de DataGrid WPF.

Microsoft en propose un gracieusement dans les **ToolKit**.

Il faut avoir installé la mise à jour SP1 du Framework 3.5, charger le ToolKit (à cette adresse : "http://www.codeplex.com/wpf/Release/ProjectReleases.aspx?ReleaseId=15598"), puis l'installer.

Dans le Framework 4, à partir de vb 2010 le DataGrid est déjà là. Il n'y a rien à installer.

Ensuite dans la boite à outils on a :


On peut maintenant déposer un DataGrid dans la Window. Cela ajoute le code Xaml suivant :

```
<my:DataGrid  Name="DataGrid1" xmlns:my="http://schemas.microsoft.com/wpf/2008/toolkit" />
```

On va maintenant, grâce à un binding, remplir la DataGrid avec une ObservableCollection de NomPerson.

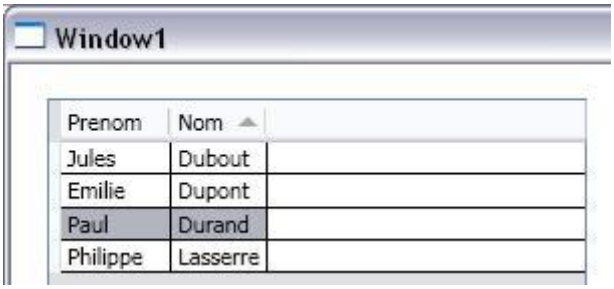
Dans un module de classe, on ajoute la ObservableCollection 'Names' (voir le code dans le chapitre sur le binding). Puis on ajoute le code VB qui instancie MyNames et qui relie DataGrid et MyNames :

```
Class Window1
    Public MyNames As New Names

Private Sub Window1_Loaded(ByVal sender As System.Object, ByVal e As
 System.Windows.RoutedEventArgs) _
Handles MyBase.Loaded
        DataGrid1.DataContext = MyNames
    End Sub
```

Enfin on ajoute dans le code xaml de la grid le ItemsSource :

```
<my:DataGrid  Name="DataGrid1" xmlns:my="http://schemas.microsoft.com/wpf/2008/toolkit"
ItemsSource="{ Binding }"/>
```

Et là, en exécution, la grille se remplit automatiquement :




Si on ajoute un élément à la collection avec le code VB suivant, la DataGrid est mise à jour (c'est l'avantage de la ObservableCollection).

```
MyNames.Add(New NomPerson("toto", "Zorro"))
```

## XI-F-9 - Image, Video, Son

Comment mettre une image dans un contrôle ?
Avec la balise 'Image' :

```
<Button
  HorizontalAlignment="Left" Margin="4,0,0,0" Name="Button1" VerticalAlignment="Top" Width="150"
  Height="100">
    <Image Source="{Binding Source=test.JPG}" />
</Button>
```

Le fichier test.jpg doit être dans le répertoire 'Document/Visual Studio 2010/Projet/MonProje/MonProjet' avec les .vb

On peut aussi utiliser les ressources pour mettre l'image (voir chapitre Ressources).

## XI-F-10 - Formes

On peut ajouter **un bord** autour d'un conteneur (ou d'un contrôle) :

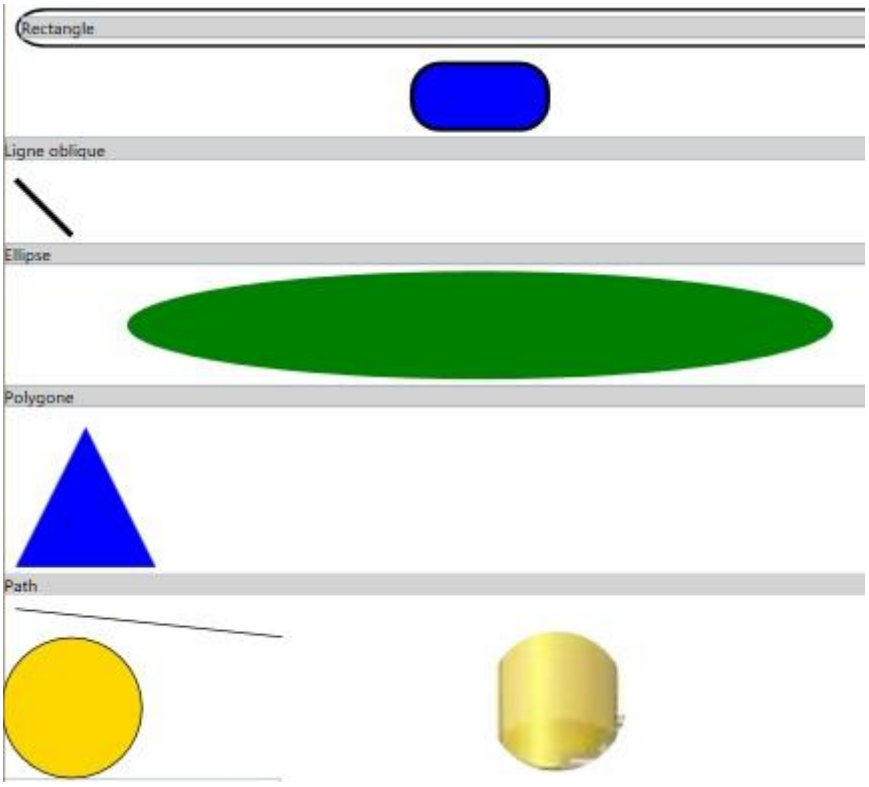
```
<Border BorderBrush="Black" BorderThickness="2"  CornerRadius="20">
              <TextBlock Background="LightGray" Margin="2,4">Rectangle</TextBlock>
</Border>
```

L'attribut CornerRadius permet d'arrondir les angles.

On peut afficher des **rectangles, lignes, ellipses, polygones, paths.**
L'exemple montre un rectangle avec un bord, des angles arrondis.
Une ligne oblique, une ellipse, un triangle, puis grâce à Path qui est très puissant, mais très complexe un cercle.
Enfin une image limitée par une ellipse.

```
  <StackPanel Margin="0,0,-37,-84">
            <Border BorderBrush="Black" BorderThickness="2" Margin="10" CornerRadius="20">
                <TextBlock Background="LightGray" Margin="2,4">Rectangle</TextBlock>
            </Border>

            <Rectangle
    Width="100"
    Height="50"
    Fill="Blue"
    Stroke="Black" StrokeThickness="3"
    RadiusX="20" RadiusY="20"
    Canvas.Left="10"
    Canvas.Top="100"/>


            <TextBlock Background="LightGray" Margin="2,4">Ligne oblique</TextBlock>
            <Line
    X1="10" Y1="10"
    X2="50" Y2="50"
    Stroke="Black"
    StrokeThickness="4" />

            <TextBlock Background="LightGray" Margin="2,4">Ellipse</TextBlock>
            <Ellipse Fill="Green" Height="77" Width="503" />

            <TextBlock Background="LightGray" Margin="2,4">Polygone</TextBlock>
            <Polygon Points="10,110 60,10 110,110"
    Fill="Blue" />
            <TextBlock Background="LightGray" Margin="2,4">Path</TextBlock>
            <Path Stroke="Black" StrokeThickness="1">
                <Path.Data>
                    <PathGeometry>
                        <PathFigure StartPoint="10,50">
```

```
                        <LineSegment Point="200,70" />
                    </PathFigure>
                </PathGeometry>

            </Path.Data>
        </Path>

        <Path Fill="Gold" Stroke="Black" StrokeThickness="1">
            <Path.Data>
                <EllipseGeometry Center="50,50" RadiusX="50" RadiusY="50" />
            </Path.Data>
        </Path>
        <Image
Source="c:\test.jpg"
Width="200" Height="150" HorizontalAlignment="Left">
            <Image.Clip>
                <EllipseGeometry
    RadiusX="50"
    RadiusY="50"
    Center="50,75"/>
            </Image.Clip>
        </Image>


    </StackPanel>
```




Voici un exemple en code vb :

```
Dim monEllipse As New Ellipse()
monEllipse.Stroke = Brushes.Black
monEllipse.Fill = Brushes.DarkBlue
monEllipse.HorizontalAlignment = HorizontalAlignment.Left
monEllipse.VerticalAlignment = VerticalAlignment.Center
monEllipse.Width = 70
myEllipse.Height = 95
maGrid.Children.Add(monEllipse)
```

# XII - Débogage

## XII-A - Débogage du code VB (rechercher les 'Bugs')

**Le débogage est la recherche des bugs, les erreurs de logique.** (Voir Traiter les erreurs.)


Voir la vidéo : **au format 'Flash'**> ou **au format 'Avi'** en Visual Basic 2005.

**Rappelons qu'il existe :**

- les erreurs de syntaxe ;
- les erreurs de logique ;
- les erreurs d'exécution.

**Les erreurs de syntaxe** sont détectées automatiquement par l'éditeur de Visual Studio ou lors de la génération du projet en mode Run. La compilation vérifie les types des données, les paramètres…

**Les erreurs d'exécution** surviennent lorsque l'exécution d'une instruction échoue (tentative d'ouverture d'un fichier qui n'existe pas par exemple). Cela provoque l'apparition d'un message et provoque l'arrêt brutal de l'application. Il faut donc prévoir une gestion des éventuelles erreurs d'exécution afin d'éviter l'arrêt de l'application. À l'aide de Try Catch, on pourra intercepter l'erreur et informer l'utilisateur, prévoir une correction.

**Les erreurs de logique** sont plus difficiles à détecter. Le code est syntaxiquement correct, mais il ne réalise pas les opérations prévues. C'est là qu'intervient le débogage afin de diagnostiquer l'origine de l'erreur.

**Pour déboguer, il faut lancer l'exécution du programme puis**

- suspendre l'exécution à certains endroits du code ;
- voir ce qui se passe puis faire avancer le programme pas à pas ;
- afficher des informations de débogage quand le programme tourne.

## XII-B - Suspendre l'exécution en vb 2008 ou vb 2010

En VB 2008, pour démarrer et arrêter l'exécution, on utilise les boutons suivants :

On lance le programme avec le premier bouton, on le suspend avec le second, on l'arrête définitivement avec le troisième…

**On peut suspendre (l'arrêter temporairement) le programme :**

- **avec le second bouton** ;
- **grâce à des points d'arrêt** (pour définir un point d'arrêt en mode de conception, cliquez en face d'une ligne dans la marge grise : la ligne est surlignée en marron. Quand le code est exécuté, il s'arrête sur cette ligne marron).
- en appuyant sur Ctrl-Alt-Pause

```
        For i= 1 To 6

        Tableau(i)=i*i
```

```
    Next i
```

**On peut suspendre (arrêter temporairement) le programme**

En plus si on clique sur le rond de gauche avec le bouton droit de la souris, on ouvre un menu permettant de modifier les propriétés de ce point d'arrêt (il y a la possibilité d'arrêter au premier ou au Xième passage sur le point d'arrêt, ou arrêter si une expression est à True ou à changer)

- **On peut suspendre (arrêter temporairement) le programme en incluant dans le code une instruction Stop**.



*Attention : si vous utilisez des instructions Stop dans votre programme, vous devez les supprimer avant de générer la version finale.*

Les transformer en commentaire :

```
' Stop
```

ou utiliser des instructions conditionnelles :

```
#If DEBUG Then
Stop
#End If
```

Dans ce cas, en mode Debug l'instruction Stop sera exécutée, pas en mode Release.

## XII-C - Débogage pas à pas en vb 2008 ou 2010

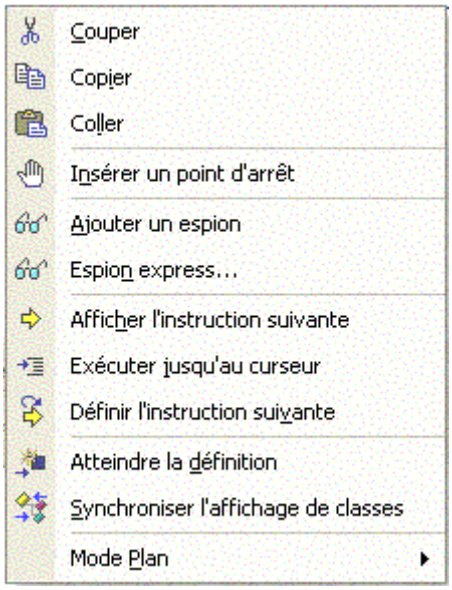
Quand le programme est suspendu, on peut observer les variables, déplacer le point d'exécution, on peut aussi faire marcher le programme pas à pas (instruction par instruction) et observer l'évolution de la valeur des variables, on peut enfin modifier la valeur d'une variable afin de tester le logiciel avec cette valeur.

**F11** permet l'exécution pas à pas, instruction par instruction (y compris des procédures appelées : s'il y a appel à une autre procédure, le pas à pas saute dans l'autre procédure).

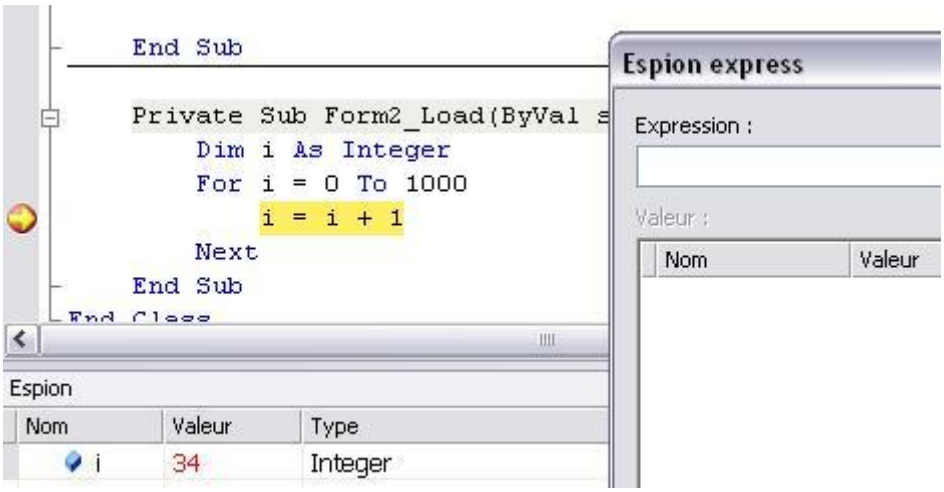
**F10** permet le pas à pas (sans détailler les procédures appelées : exécute la procédure appelée en une fois).

**Maj+F11** exécute jusqu'à la fin de la procédure en cours.

On peut afficher ou définir l'instruction suivante, exécuter jusqu'au curseur, insérer un point d'arrêt ou un espion en cliquant sur le **bouton droit de la souris** et en choisissant une ligne du menu (VB 2003).




Espion express (VB 2003 et 2005) permet de saisir une expression (variable, calcul de variables) et de voir ensuite dans une fenêtre 'espion' les modifications de cette expression au cours du déroulement du programme.




Exemple en VB 2005, c'est pareil en VB 2003 : suivre la valeur de la variable de boucle 'i'.

Sur la ligne, bouton droit, 'espion express…' dans la fenêtre qui s'ouvre à droite,

tapez 'i' puis 'Fermer', la fenêtre espion apparait en bas avec la valeur de i.

Tapez F10 , F10… la boucle tourne et i est incrémenté.

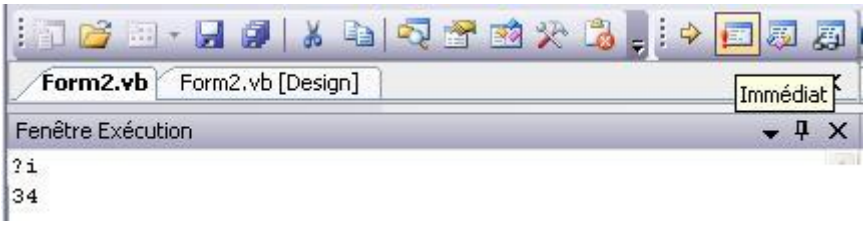
**Atteindre la définition** permet d'afficher la déclaration de la variable et ainsi de voir quelle est sa portée et si elle est initialisée. S'il s'agit du nom d'une procédure, on se retrouve dans la procédure (pour voir ce qu'elle contient).

**On peut grâce au menu 'Débogage' puis 'Fenêtre' ouvrir les fenêtres :**

**Automatique** uniquement en VB 2003, qui affiche les valeurs des variables de l'instruction en cours et des instructions voisines ;

**Immédiat** (VB 2003 2005) où il est possible de taper des instructions ou expressions pour les exécuter ou voir des valeurs.

Taper "?I" (c'est l'équivalent de "Print I" qui veut dire : afficher la valeur de la variable I) puis valider, cela affiche la valeur de la variable I.




Autre exemple, pour voir le contenu d'un tableau A(), tapez sur une seule ligne : "For i=0 to 10: ?A(i): Next i"

Enfin, il est possible de modifier la valeur d'une variable : Taper"I=10" puis valider, cela modifie la valeur de la variable.

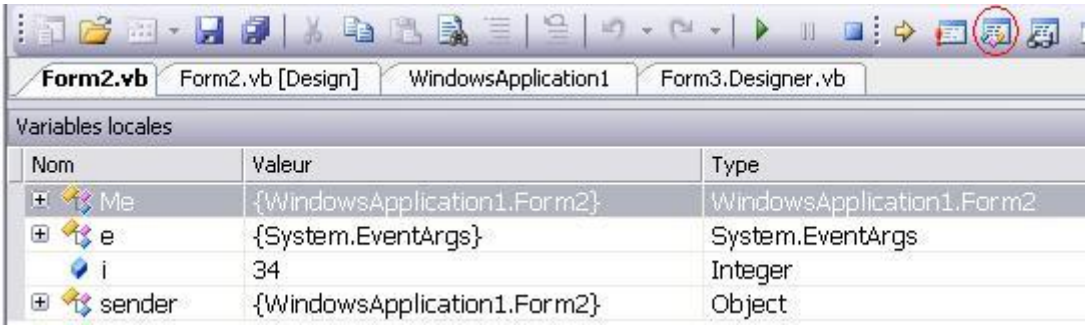
**Espions** permettant d'afficher le contenu de variables ou d'expressions.

**Espions Express** permet d'afficher la valeur de l'expression sélectionnée.

**Points d'arrêts** (VB 2003 et 2005) 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** (VB 2003) affiche les données du module en cours.

**Variables locales** (VB 2003 et 2005) affiche les variables locales.




**Modules** (VB 2003) affiche les dll ou .exe utilisés.

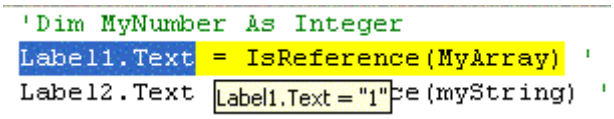
**Mémoire, Pile d'appels, Thread, Registres, Code Machine** permettent d'étudier le fonctionnement du programme à un niveau plus spécialisé et technique.

Toutes ces fenêtres ne sont pas disponibles dans les versions Express.

## XII-C-1 - Comment voir rapidement la valeur des propriétés ou de variables ?

Il est toujours possible de voir la valeur d'une propriété d'un objet en la sélectionnant avec la souris.

Exemple on sélectionne label1.Text et on voit apparaitre sa valeur.




Pour les variables, il suffit que le curseur soit sur une variable pour voir la valeur de cette variable.

On peut aussi copier une expression dans la fenêtre 'immédiat', mettre un ? avant et valider pour voir la valeur de l'expression.

Attention à l'affichage

Parfois en mode pas à pas on regarde le résultat d'une instruction dans la fenêtre du programme. Par exemple on modifie la propriété text d'un label et on regarde si le label a bien changé. Parfois la mise à jour n'est pas effectuée, car le programme met à jour certains contrôles seulement en fin de procédure. Pour pallier cela et afficher au fur et à mesure, même si la procédure n'est pas terminée, on utilise la méthode **Refresh** de l'objet qui 'met à jour'.

Exemple :

```
Label1.text="A"

Label1.Refresh
```

Cela ne semble pas toujours fonctionner. Avez-vous une explication ?

## XII-C-2 - Modification du code source

En version 2003, les modifications effectuées lors de la suspension de l'exécution ne sont pas prises en compte lors du redémarrage.

Depuis VB 2005 il y a maintenant le '**Edit and continue**' : en mode Debug, on peut modifier une ligne et poursuivre le programme qui tiendra compte de la modification (sauf pour les déclarations).

Depuis la version 2005, il est proposé des solutions pour corriger les erreurs de code :

une fenêtre vous indique les corrections à faire.

Si je veux afficher une valeur numérique (avec option Strict=On),il y a erreur, VB me propose la correction :


## XII-D - Débogage en vb 2010

Vb propose des solutions pour corriger les erreurs de code.

Si je fais une erreur, Vb la souligne en ondulé bleu, si je mets le curseur dessus il m'explique l'erreur :

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.
    Dim Appa As Integer
    Dim b As Integer = 2
    C = b + 1
```

'C' n'est pas déclaré. Il peut être inaccessible en raison de son niveau de protection.

S'il y a un soulignement rouge, mettre le curseur dessus affiche un bouton avec un point d'exclamation qui ouvre une fenêtre donnant la correction de l'erreur :

```
Dim b As Integer = 2
Button1.Text = b
```

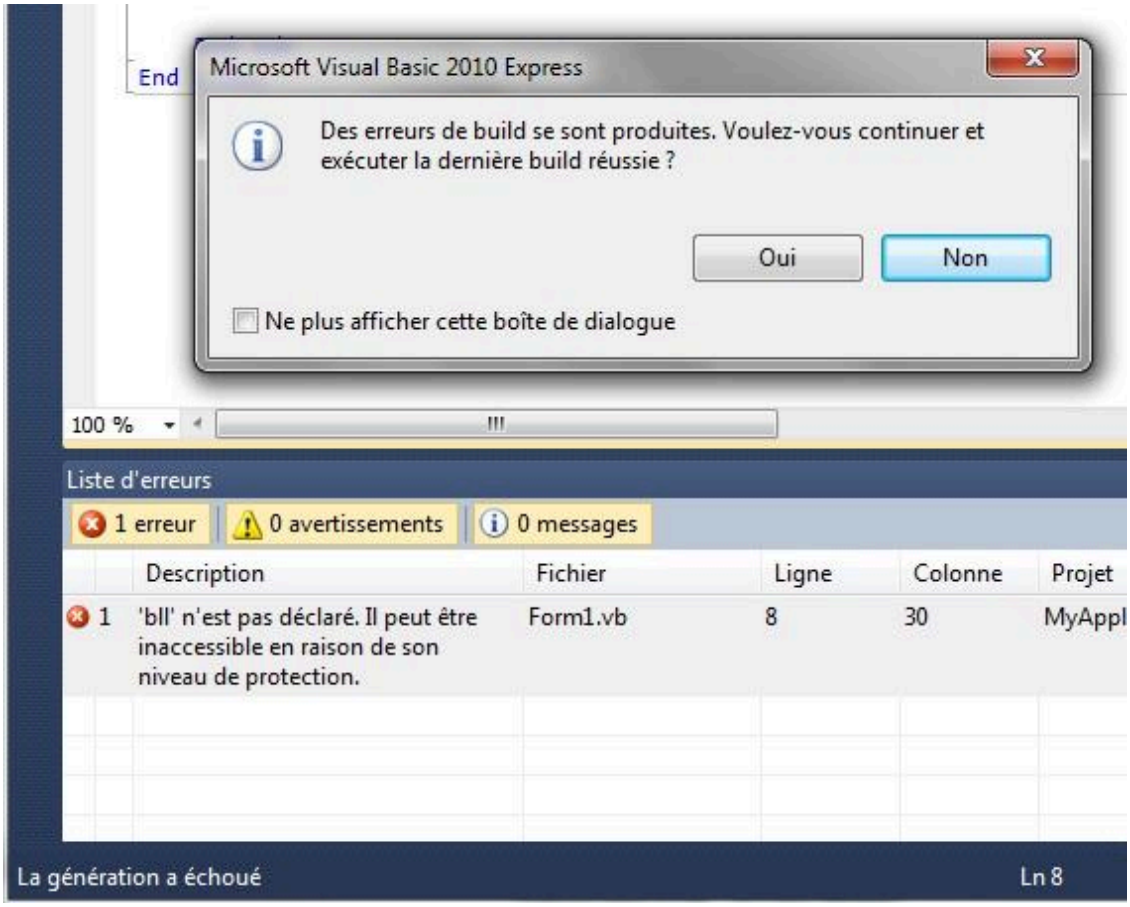
**Option Strict On interdit les conversions implicites de 'Integer' en 'String'.**

Remplacez 'b' par 'CStr(b)'.

```
Dim Appa As Integer
Dim b As Integer = 2
Button1.Text = CStr(b)
```

Développer tous les aperçus

Si je lance le programme en mode 'Run' et qu'il y a des erreurs, Vb me le signale et répertorie les erreurs dans la liste des tâches en bas.

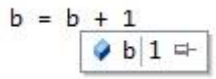



### Mode débogage (mode BREAK)

Une fois lancée l'exécution (**F5**),on peut arrêter temporairement l'exécution :
- en tapant **Ctrl +Pause** ;
- en cliquant sur  (visible lors de l'exécution) ;
- en utilisant un point d'arrêt (avant l'exécution ou en cours d'exécution, en cliquant dans la colonne grise avant la ligne de code, cela affiche un rond, un point d'arrêt) ;
- en ajoutant une instruction STOP dans le code.

On peut modifier le code puis relancer l'exécution avec F5.

On peut voir la valeur d'une propriété d'un objet , d'une variable en le pointant avec la souris :

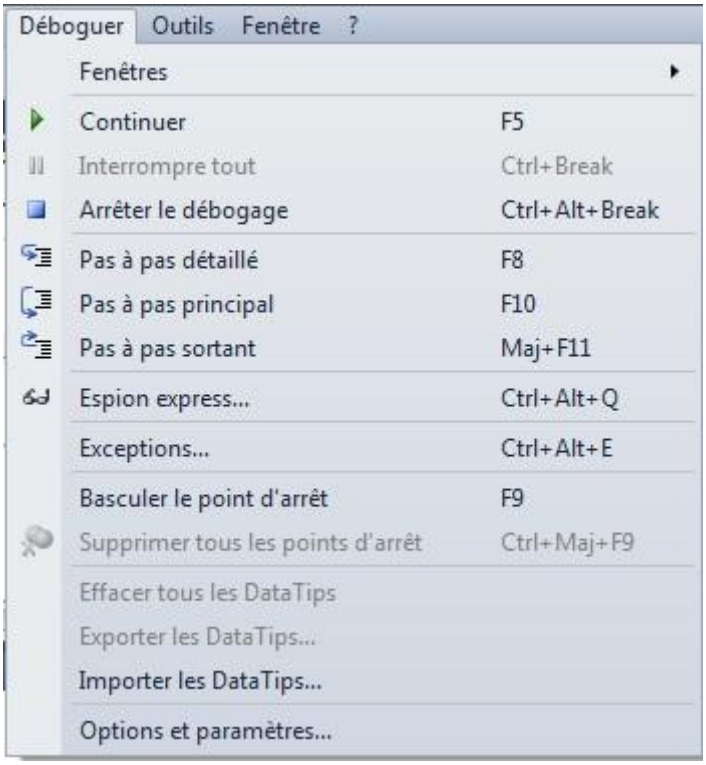



Il s'affiche un petit cadre donnant la valeur de la propriété d'un objet.
Si on clique sur la punaise, l'affichage de la variable et de sa valeur devient permanent et la valeur est mise à jour.

Quand on clique sur une variable, cette variable est surlignée dans l'ensemble du code :

```
Dim b As Integer
Button1.Text = CType(b, String)

b = b + 1
```

Il y a un menu déboguer :




**F8** permet l'exécution pas à pas (y compris des procédures appelées).

**MAJ F8** permet le pas à pas (sans détailler les procédures appelées).

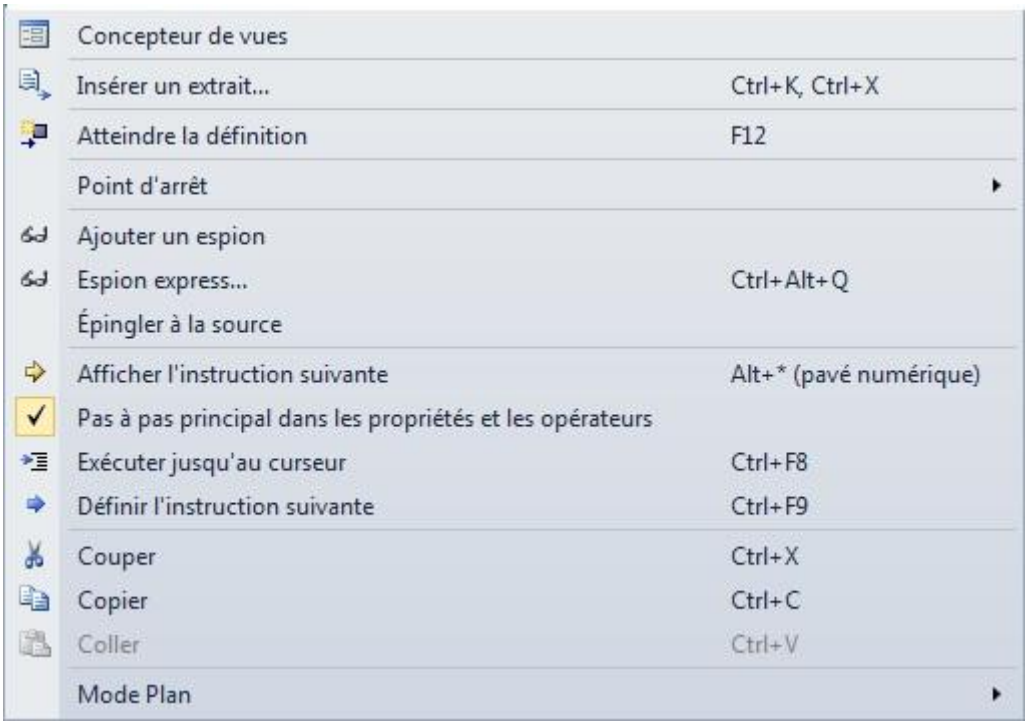
**CTRL+F10** pas à pas principal.

**MAJ+F11** pas à pas sortant.

**Espions** permettant d'afficher le contenu de variables ou d'expressions.

**Espions Express** permet d'afficher la valeur de l'expression sélectionnée.

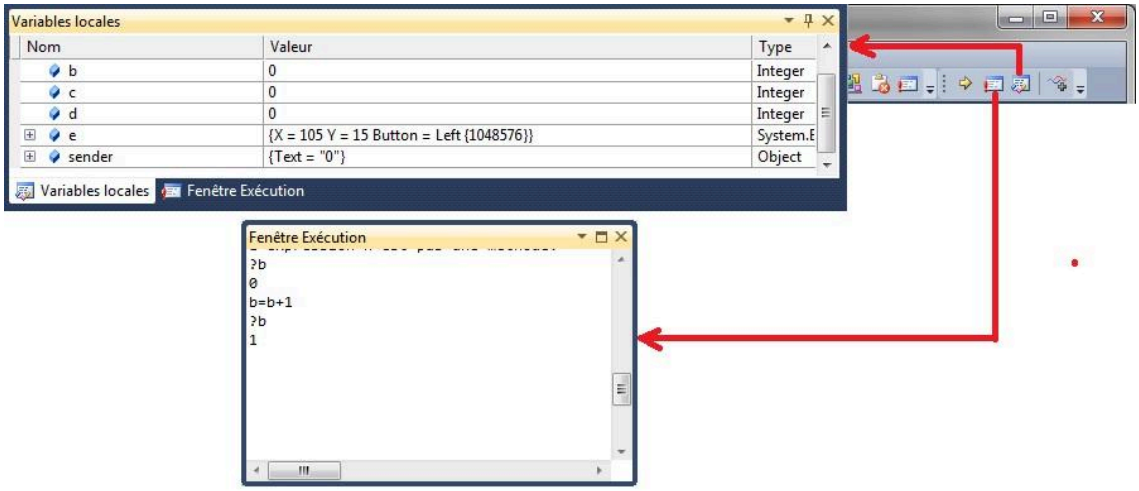
Si dans le code on clique droit s'ouvre un menu contextuel :




**CTRL+F8** exécute jusqu'au curseur.

**Alt+*** afficher l'instruction suivante.

Toutes ces actions peuvent être effectuées par les menus, les raccourcis.
On peut aussi ajouter des boutons de déboguer dans la barre d'outils (menu 'Affichage', 'Barre d'outils', 'déboguer').
Le bouton 'variables locales'(CTRL+ALT+V puis L) affiche toutes les variables locales et leur valeur :




Le bouton immédiat ouvre une fenêtre qui permet de taper du code qui sera immédiatement exécuté.
(Ici on affiche la valeur de b avec '?b', on modifie sa valeur avec 'b=b+1'.)

Vb choisit automatiquement la configuration Debug (compilée avec des informations de débogage symboliques et aucune optimisation) lorsque vous cliquez sur Démarrer dans le menu Déboguer et la configuration Release (ne contient pas d'informations de débogage relatives aux symboles et est entièrement optimisée) lorsque vous utilisez le menu Générer.

## XII-E - Sortie des informations de débogage

On veut parfois afficher automatiquement des résultats intermédiaires, un message destiné au programmeur dans telle circonstance…quand le programme tourne.

### XII-E-1 - Objet Console

On peut écrire **sur la fenêtre console**, quand on a parfois besoin d'afficher des informations, mais uniquement pour le programmeur :

```
Console.WriteLine( myKeys(i))
```

Mais dans un programme Windows, il n'y a pas de console !! La sortie est donc envoyée vers la fenêtre de sortie (voir Debug) (Menu Affichage>Autres fenêtres>Sortie pour voir la fenêtre).

Autre exemple :

```
Dim amis() As String = {"pierre", "jean", "jacques", "toto"}

For Each nom As String In amis

Console.Out.WriteLine(nom)

Next
```

### XII-E-2 - Objet Debug

L'espace de noms **Systems.Diagnostics** est nécessaire.

Pour déboguer du code, on a parfois besoin d'afficher des informations, mais uniquement pour le programmeur, en mode debug afin de suivre le cheminement du programme ou la valeur d'une variable ou si une condition se réalise; pour cela on utilise une fenêtre nommée 'Sortie'(Output). (Menu Affichage>Autres fenêtres>Sortie.)

Pour écrire dans la fenêtre Output (sans arrêter le programme) :

- **du texte :**

```
Debug.Write(Message)
```

et pour ajouter un passage à la ligne :

```
Debug.WriteLine(Message)
```

- **le contenu d'une variable :**

```
Debug.Write(Variable)
```

- **les propriétés d'un objet :**

```
Debug.Write(Objet)
```

Exemple :

```
Debug.Write("ça marche")    'Affiche 'ça marche'

Dim A as Integer=2

Debug.Write(A)    'Affiche 2

Debug.Write(A+2)  'Affiche 4
```

On voit que s'il y a une expression, elle est évaluée.

On peut aussi afficher un message si une condition est remplie en utilisant WriteLineIf ou WriteIf :

```
Debug.WriteLineIf(i = 2, "i=2")
```

Affiche 'i=2' si i=2.

Cela vous permet, sans arrêter le programme (comme le fait Assert), d'être informé quand une condition est vérifiée.

**Debug.Assert** par contre affiche une fenêtre Windows et stoppe le programme si une assertion (une condition) passe à False.

```
Debub.Assert(Assertion)

Debub.Assert(Assertion, Message1)

Debub.Assert(Assertion, Message1, Message2)
```

L'exemple suivant vérifie si le paramètre 'type' est valide. Si le type passé est une référence null (Nothing dans Visual Basic), Assert ouvre une boite de dialogue nommé 'Echec Assertion' avec 3 boutons 'Abandonner, Recommencer' 'Ignorer'… La liste des appels est affichée dans la fenêtre (procédure en cours en tête de liste, module et numéro de ligne en première ligne)

```
Public Shared Sub UneMethode (type As Type, Typedeux As Type)
```

```
Debug.Assert( Not (type Is Nothing), "Le paramètre Type est=Nothing ",
    _"Je ne peux pas utiliser un Nothing")

…
End Sub UneMethode


Debug.Fail
```

Fait pareil, mais sans condition.

## XII-E-3 - Trace

**Trace** possède les mêmes fonctions que Debug (Write, WriteIf, Assert, Fail…), mais la différence est que Trace permet d'afficher à l'utilisateur final par défaut.

## XII-E-4 - Mode 'Trace', 'Release', 'Debug'

En VB 2003, en haut de la fenêtre de l'IDE il y a une liste déroulante elle contient :

**Release** (à utiliser pour générer la version à distribuer) ;

**Debug** (à utiliser pour générer une version à tester).

En VB 2005, si vous choisissez les paramètres de développement Visual Basic, l'outil qui permet de choisir entre la configuration Debug et Release n'apparait pas dans la barre d'outils. Visual Studio choisit automatiquement la configuration Debug lorsque vous cliquez sur Démarrer dans le menu Débogueur et la configuration Release lorsque vous utilisez le menu Générer.

Trace est activé par défaut. Par conséquent, du code est généré pour toutes les méthodes Trace dans les versions release et debug. Ceci permet à un utilisateur final d'activer le traçage pour faciliter l'identification du problème sans que le programme ait à être recompilé.

Par opposition, Debug est désactivé par défaut dans les versions release, donc aucun code exécutable n'est généré pour les méthodes Debug.

**On peut utiliser une constante nommée DEBUG** qui aura la valeur True si on est en mode Debug.

Cela permet d'écrire :

```
#If Debug Then

    Stop

#End If
```

Ici Stop se produira uniquement si on est en mode Debug ; en mode Release, il n'y aura pas d'arrêt.

## XII-F - Comprendre les 'Messages d'erreur'

Quand il y a une erreur de syntaxe, VB souligne l'erreur. Mettre le curseur sur l'erreur, un message d'erreur apparait :


En VB 2005 un panneau d'exclamation permet d'ouvrir une fenêtre proposant le moyen de corriger l'erreur :


Ici on met dans la propriété text d'un label un Integer, alors qu'il faut mettre une String (Option Strict est probablement égal à On) ; Vb montre la correction : CStr(i) convertit i en String.

Certains messages d'erreur, pour les débutants, sont parfois totalement incompréhensibles !!

Voyons quelques messages très fréquents.

## XII-F-1 - Instance d'objet

**"La référence d'objet n'est pas définie à une instance d'un objet"**

La plupart du temps cela veut dire qu'on utilise un membre d'un Objet alors qu'on n'a pas instancié cet objet.

Il y a bien une Classe, mais pas d'objet instancié à partir de cette Classe, on veut utiliser un membre de la Classe alors qu'on ne peut utiliser un membre que sur une instance.

Exemple :

```
Dim bt As  Button

bt.BringToFront()
```

Il n'existe pas réellement d'objet Button : la référence d'objet (bt) n'est donc pas une instance.

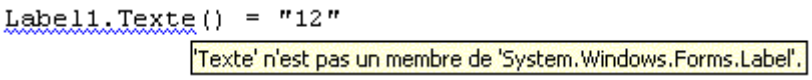
Il aurait fallu écrire :

```
Dim bt As New Button

bt.BringToFront()
```

## XII-F-2 - Membre absent

**"Texte n'est pas un membre de System.Windows.Forms.TextBox"**

Parfois on fait une bête faute de frappe :




Il faut taper : text !!

D'autres fois, on se trompe sur la classe et instance.

```
TextBox1.IsNullOrEmpty  donne le message:
```

**"IsNullOrEmpty n'est pas un membre de System.Windows.Forms.TextBox"**

On se trompe : on pense que IsNullOrEmpty est un membre de l'instance TextBox1, en fait c'est un membre de la classe String, il faut écrire :

```
If String.IsNullOrEmpty(TextBox1.Text)
```

## XII-F-3 - Type Attendu

**"Type attendu"**

Exemple :

```
Private MyVariable As New Obladioblada
```

Après **As New** VB attend un type (Integer, Short, String) ou un nom de classe (Form1…). Obladioblada n'est pas une classe (un type !!)

Exemple2:

```
' Déclaration Objet DataSet

Private ObjetDataSet As New dataset
```

dataset est souligné comme une erreur "type attendu" !! Après New Vb attend un type d'objet, une classe, dataset n'est donc pas considéré comme une classe, car il n'est pas reconnu : la Classe correspondante n'a pas été importée.

En fait dataset ne peut être utilisé que si Imports System.Data a été ajouté avant, dans ce cas Vb sait que c'est un type.

## XII-F-4 - Identificateur attendu

```
Dim 2a As Integer
```

2 est surligné, si on met la souris sur le "2", le message **'Identificateur attendu'** apparait. Erreur : un nom de variable ne doit pas commencer par un chiffre, Vb considère donc que '2a' n'est pas un identificateur (un nom de variable' valide).

# XIII - Comprendre le fonctionnement de VB

## XIII-A - Comprendre le fonctionnement de VB.net

Comment fonctionne un programme VB.NET ?

### XIII-A-1 - Le Framework.NET le CLR

Jusqu'à VB6, un programme VB avait besoin d'un RunTime (fichier Dll : VBRUN300.DLL par exemple pour VB3).

En VB.net, pour qu'un programme fonctionne, il faut installer le Framework.NET.

C'est une plateforme informatique, une couche entre Windows et l'application VB.

**'Framework' veut dire : bibliothèque de classes.**

Le Framework.Net est donc la bibliothèque de classes .NET et contient ADO.NET, ASP.NET et Windows Forms.

**Cette infrastructure offre un vaste accès à :**

- l'ensemble du système d'exploitation ;
- une collection de Classes pour fournir des objets utilisables pour créer des programmes ;
- des routines d'exécution de programme.

Ceci de manière homogène et très fiable.


**L'exécutable en Visual Basic :**

- utilise les objets (WindowsForms, WPF, type de variable…) du Framework ;
- appelle les fonctions (exécution, affichage, gestion de la mémoire, lecture dans une base de données…) du Framework.

À la limite on peut considérer le Framework comme une machine virtuelle (comme celle de Java). Il suffirait de porter le FrameWork sous Linux pour que les programmes VB fonctionnent (il existe bien le projet '**Mono**' sous Linux, mais je ne sais pas s'il contient les Windows Forms).

On rappelle que ce qui fait tourner le Framework c'est le CLR (**Commnon Language RunTime**) ; de la version 2 à la version 3.5 du Framework, c'était toujours la version 2.0.50727 du CLR qui est utilisée.
Avec le Framework 4 c'est la version 4 !! du CLR qui est utilisée.

### XIII-A-2 - Inconvénients ?

La couche supplémentaire ralentit le programme ?

À peine dit Microsoft, cela serait insignifiant ! En fait cela ralentit beaucoup. Le Framework, quand il ouvre un formulaire appelle les routines de Windows, c'est bien une couche supplémentaire.

Et s'il y a une nouvelle version du Framework ?

Les versions successives devraient être compatible ascendante et descendante !!

En fait, on peut installer le Framework 1 le Framework 2 et le Framework 3. En VB 2008 et 2010 on peut choisir la version du Framework utilisée par un logiciel (en vb 2010 menu 'Projet', 'Propriété de…', onglet 'Compiler', en bas liste : 'Framework cible'); les frameworks ne sont donc pas compatibles.

On peut installer à côté la version 1, 2 ,3 ,3.5, 4; il faut en installer plusieurs si nécessaire (si on installe différents logiciels qui utilisent des Frameworks de version différente), car il n'y a pas de compatibilité ascendante.

## XIII-A-3 - Intérêts ?

On installe une seule fois le Framework.

Une fois le Framework installé, il suffit pour installer un nouveau programme de n'installer que l'exécutable.

On peut utiliser plusieurs langages.

Nous appelons les fonctions du FrameWork avec Visual Basic, mais on peut aussi le faire avec C# et 30 autres langages. La vitesse d'exécution sera la même, les types de variables, les objets seront les mêmes. Plus de problèmes de transfert de paramètres.

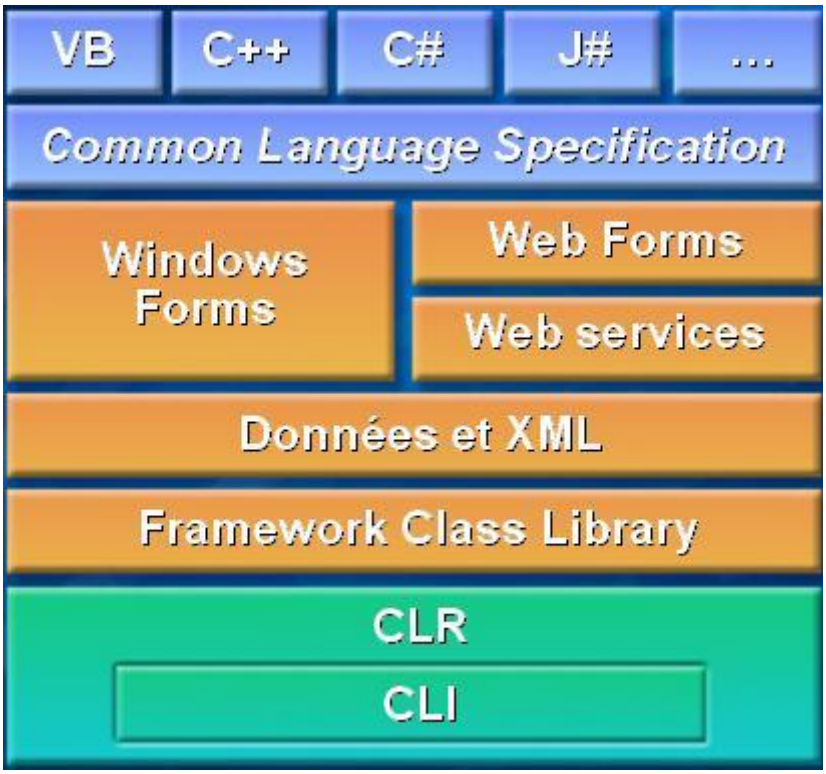
Il est même possible de faire cohabiter plusieurs langages dans une même application.

Le code est homogène.

Plus de bidouille, de ficelle de programmation pour contourner les lacunes du langage et l'accès limité au système d'exploitation: Les milliers de Classes du FrameWork donnent accès à une multitude de fonctions et aux services du système d'exploitation, cela nativement.

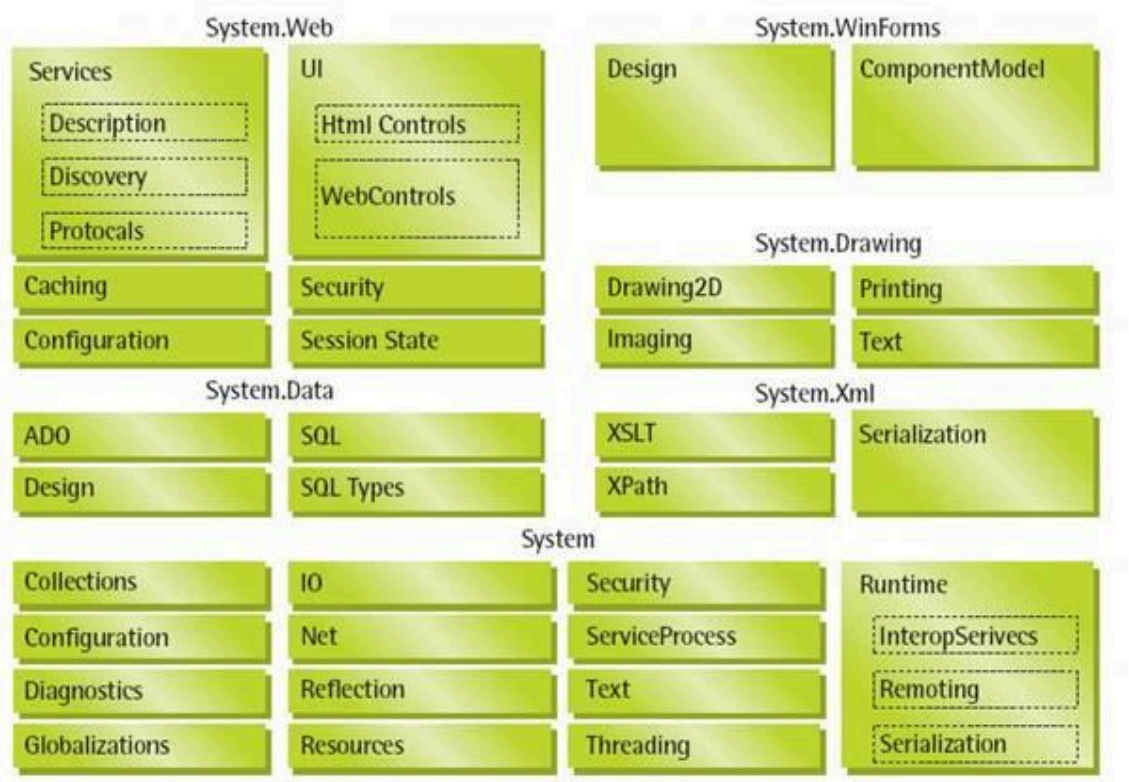
## XIII-A-4 - Revoyons en détail le contenu du Framework

Le **CLS** (Common Language Spécification) représente la partie visible , interface entre les langages VB, C++, C#, J# et le Framework.
Pour interagir entièrement avec des objets managés, quel que soit le langage dans lequel ils ont été implémentés, les objets managés ne doivent exposer aux appelants que les fonctionnalités qui sont communes à tous les langages avec lesquels ils doivent fonctionner.
La spécification CLS permet d'optimiser et d'assurer l'interopérabilité des langages. Si votre composant n'utilise que des fonctionnalités CLS dans des interfaces API qu'il expose à un autre code (y compris des classes dérivées), le composant est assuré d'être accessible à partir d'un langage de programmation prenant en charge la spécification CLS. Il est important de comprendre que le respect du CLS augmente la sécurité de votre programme.




**Il contient deux composants principaux :**

- le **Common Language Runtime** (CLR).
  Il permet :
  - exécution du code managé,
  - gestion de la mémoire (Garbage collector),
  - utilisation des objets du FrameWork.
  Le runtime peut être considéré comme un agent qui manage le code au moment de l'exécution, qui l'exécute, fournit des services essentiels comme la gestion de la mémoire, la gestion des threads, et l'accès distant ;
- **la bibliothèque de classes du .NET FrameWork** (FrameWork Class Library).
  C'est une collection complète orientée objet, que vous pouvez utiliser pour développer des applications allant des traditionnelles applications à ligne de commande ou à interface graphique utilisateur (GUI, Graphical User Interface) jusqu'à des applications qui exploitent les dernières innovations fournies par ASP.NET, comme les services Web XML et Web Forms.
  Par exemple, les classes Windows Forms sont un ensemble complet de types réutilisables qui simplifient grandement le développement Windows. Si vous écrivez une application Web Form ASP.NET, vous pouvez utiliser les classes Web Forms.
  Il existe un éventail de tâches courantes de programmation y compris des tâches comme la gestion de chaines, la collection de données, la connectivité de bases de données, et l'accès aux fichiers.
  Il y a 3300 Classes dans le Framework 1 !!




**Plus d'appel aux Api Windows**, on fait tout directement en utilisant les classes du Framework.

## XIII-A-5 - Framework 1, 2, 3, 3.5, 4

Un Framework est donc un ensemble de Classes.

| | 2002 | 2003 | 2005 | 2006 | 2008 | 2010 |
|---|---|---|---|---|---|---|
| Visual Studio | VS.NET 2002 | VS.NET 2003 | VS2005 | VS2005 + Extensions | VS2008 | VS 2010 |
| Langage | VB.NET v7.0 | VB.NET v7.1 | VB.NET v8.0 | VB.NET v8.0 | VB.NET v9.0 | VB.NET |
| Framework | .NET 1.0 | .NET 1.1 | .NET 2.0 | .NET 3.0 | .NET 3.5 | .NET 4.0 |
| CLR | CLR v1.0 | CLR v1.1 | CLR v2.0 | CLR v2.0 | CLR v2.0 | CLR v4.0 |

**Le framework 1.0 est utilisé par VB 2003 (année 2002).**

**Le framework 2.0 est utilisé par VB 2005 (année 2005).**

Il contient des classes supplémentaires.

**Le framework 3.0 peut être utilisé par VB 2005.**

Le framework 3.0 est composé du framework 2.0 auquel s'ajoutent WCF (Windows Communication Foundation), WF (Windows Workflow Foundation), WPF (Windows Presentation Foundation) et infocard pour l'authentification des utilisateurs.

**Windows Presentation Foundation** (WPF) ex 'avalon' utilise un moteur de rendu vectoriel et des accélérations matérielles pour afficher. Présent dans la version Express.

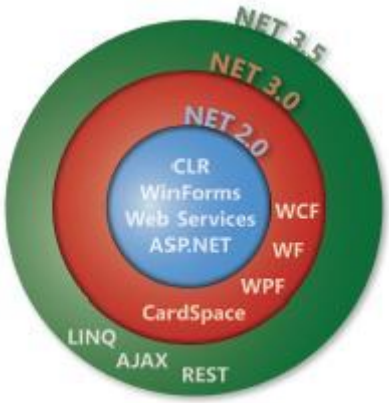
**Windows Communication Foundation** (WCF) ex 'indigo' permet de développer des applications distribuées interopérables, fiables et sécurisées. WCF simplifie le développement d'applications connectées. Non présent dans la version Express.

**Windows Workflow Foundation** (WF) est constitué d'un modèle de programmation, d'un moteur d'exécution et d'outils, pour développer et intégrer des workflows dans les applications .NET. (Un workflow est une succession d'actions ou d'étapes qui s'exécutent dans un ordre prédéfini.) Non présent dans la version Express.

**Windows CardSpace** (WCS), ex Infocard, est une nouvelle technologie qui permet aux utilisateurs de prouver leur identité. Non présent dans la version Express.

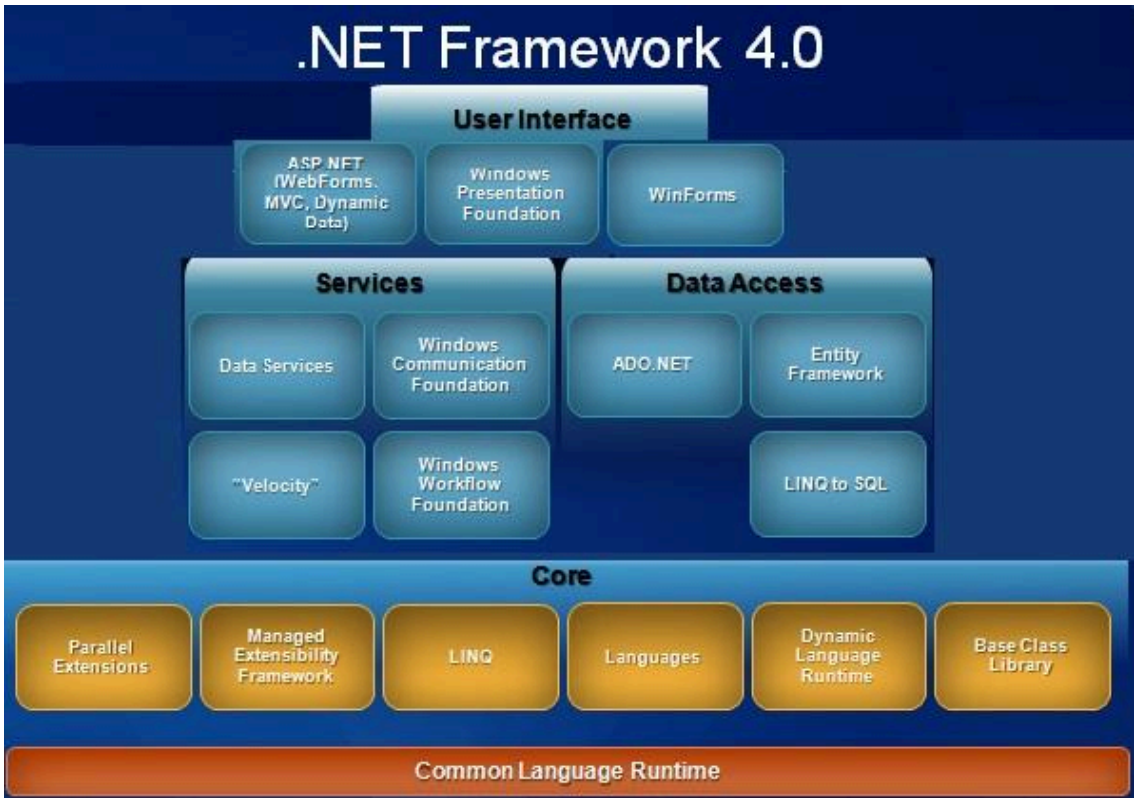
Le **Framework 3.5** est utilisé par VB2008.

C'est le Framework 3 auquel s'ajoute AJAX (utilisable dans les WebForms), LINQ et REST (c'est quoi ?).



Versions cumulées du .NET Framework

**Le Framework 4 utilisé par VB2010.**




**Voir la documentation Msdn du Framework 4**

Sous Windows 98, XP, il faut installer le framework (avant d'utiliser l'environnement VisualBasic ou un exécutable VB).

Sous Windows Vista le Framework 3 est installé nativement (et les versions précédentes?).
Sous Windows 7 les Framework 1, 1.1, 2, 3, 3.5 sont installés nativement.
Quand on installe vb 2010 cela installe le framework 4.

## XIII-A-6 - Code managé

Le code écrit pour le FrameWork est dit managé (ou géré): il bénéficie des avantages du Framework: gestion de la mémoire, optimisation de cette mémoire, règles communes, utilisation de plusieurs langages…

Les anciens composants COM sont utilisables, mais non managés.

L'interopérabilité entre les codes managés et non managés permet aux développeurs de continuer à utiliser des composants COM et des DLL nécessaires.

## XIII-A-7 - Garbage Collector

La destruction effective des objets et variables est effectuée par le 'garbage collector' (ou ramasse-miettes).

Avant .NET quand on détruisait une variable, en lui affectant Nothing, la destruction était immédiate (déterministe) ; si on oubliait (une application) de détruire une variable de multiple fois, il y avait problème !!

Maintenant, quand on détruit une variable, elle reste présente en mémoire ; c'est le garbage collector qui, quand il a le temps ou quand il n'y a plus de place, détruit la variable et fait le ménage. On ne sait pas quand il va le faire (non déterministe).

## XIII-A-8 - Compilation

Le code que vous écrivez en Visual Basic est le code source.

Lors de la génération (compilation) du projet, un code intermédiaire est produit (IL : **Intermédiaire Langage**), ce code est commun à tous les langages.

Lors de la première exécution d'exécutable (.exe), le code IL est compilé en binaire par le compilateur '**Just in Time**' puis exécuté. Les exécutions suivantes seront plus rapides.

En résumé

Code Source (VB, C#…) => Intermediate Language (IL). par le compilateur syntaxique.

IL => Binaire exécutable par le compilateur 'Just in Time'.

## XIII-A-9 - Comment voir le code source, le code IL, le code exécutable ?

Exemple effectué sur VB 2003, édition professionnelle. Avec les versions Express 2005, 2008, 2010, certaines fenêtres ne sont pas disponibles.

Prenons un exemple : il y a dans une form1 une procédure Button1_Click.

**a-Voici le code source :**

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
 Button1.Click

TextBox1.Text = "hello"

End Sub
```

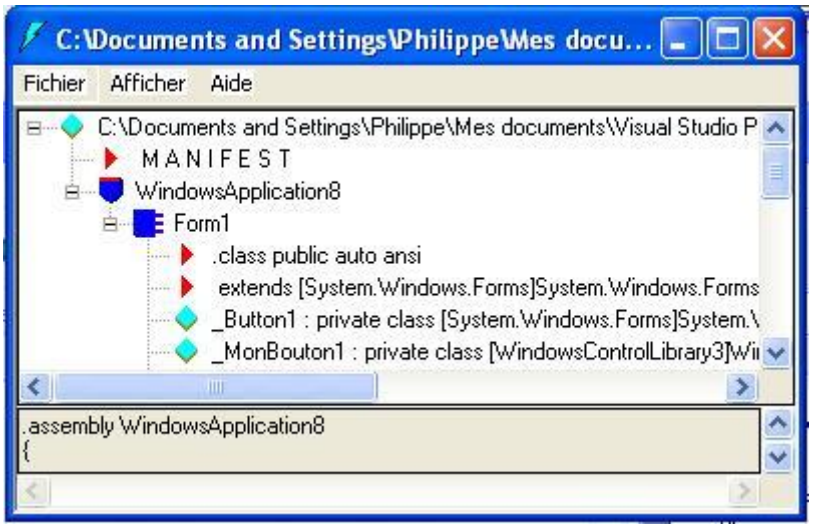
on génère ce code source cela donne un fichier .exe (dans le répertoite /bin)

**b-Voir le code Intermédiaire (IL)**

Avec Ildasm.exe (désassembleur code IL) qui est dans le SDK :

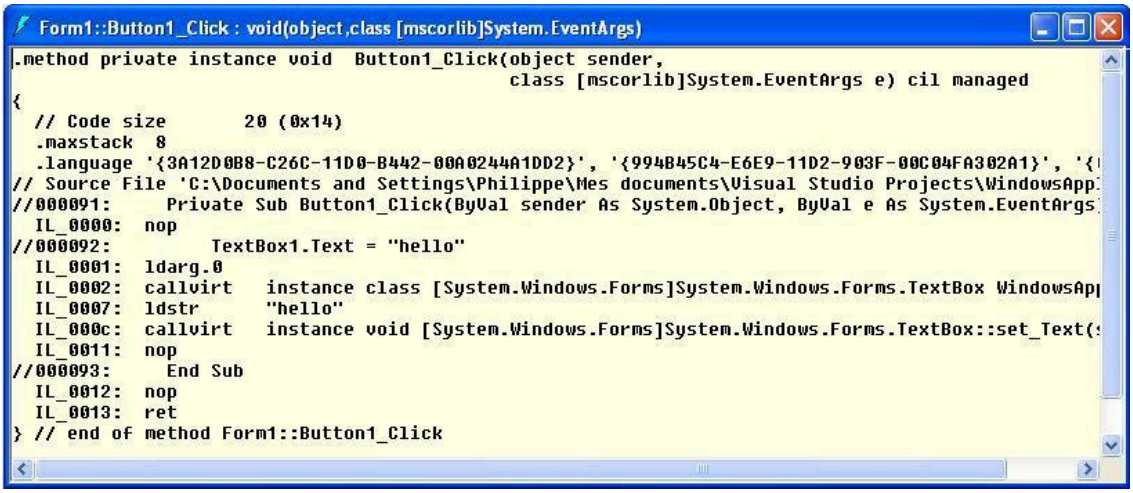
(C:\Program Files\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin)

On peut ouvrir un fichier exe (menu 'Fichier') :




Outre le MANIFEST qui contient les métadonnées de l'assembly, il y a le code IL de l'application.

Si on clique sur la ligne _Button1 , on voit le code IL correspondant à la routine: (un menu de ildasm.exe permet de voir le code source en même temps)




**c-Voir le code binaire (en assembleur)**

Dans l'IDE (pas dans les versions Express), si je stoppe le fonctionnement de la routine (par un point d'arrêt), le menu 'Fenêtre', 'Code machine' permet de voir l'exécutable, mais sous forme d'assembleur.

Le véritable code binaire est composé d'instructions ayant chacune un code.

Voici du code exécutable représenté en hexadécimal : 55 8b ec

Comme c'est totalement illisible, on affiche le code binaire sous une autre forme, en langage d'assemblage ou les instructions sont représentées par des mots mnémoniques : push (pour utiliser la pile), mov (pour déplacer un octet en mémoire…), le code est plus lisible.

Voici le code en assembleur de notre routine :

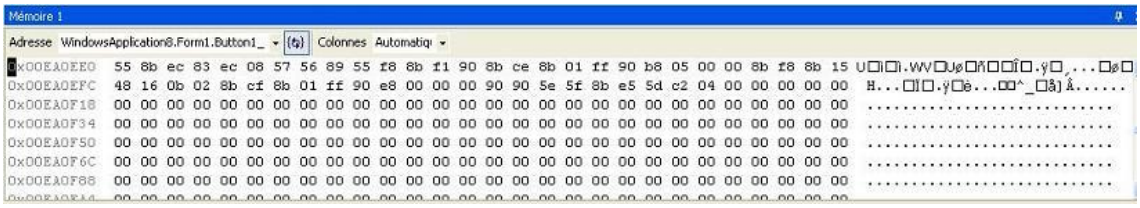
```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
 Button1.Click
```

```
00000000 push ebp
00000001 mov ebp,esp
00000003 sub esp,8
00000006 push edi
00000007 push esi
00000008 mov dword ptr [ebp-8],edx
0000000b mov esi,ecx
0000000d nop
TextBox1.Text = "hello"
0000000e mov ecx,esi
00000010 mov eax,dword ptr [ecx]
00000012 call dword ptr [eax+000005B8h]
00000018 mov edi,eax
0000001a mov edx,dword ptr ds:[020B1648h]
00000020 mov ecx,edi
00000022 mov eax,dword ptr [ecx]
00000024 call dword ptr [eax+000000E8h]
0000002a nop
End Sub
0000002b nop
0000002c pop esi
0000002d pop edi
0000002e mov esp,ebp
00000030 pop ebp
00000031 ret 4
```

On note que pour information et amélioration de la lisibilité, le code source est ajouté.

### d-Voir le véritable code binaire

Si je fais menu 'Déboguer', 'Fenêtre', 'Mémoire'(Version non Express), j'ai un dump de la mémoire en hexadécimal qui montre le véritable code binaire.



## XIII-A-10 - Installation du Framework

Les applications et contrôles écrits pour le .NET Framework imposent que celui-ci soit installé sur l'ordinateur où ils s'exécutent. Microsoft fournit un programme d'installation redistribuable, Dotnetfx.exe , qui contient les composants Common Language Runtime et .NET Framework nécessaire à l'exécution des applications .NET Framework.

**Ou le trouver**

La version Visual Basic Express 2008 (Framework compris) se trouve ici :

http://www.microsoft.com/express/download/

Choisir autre langue=Français.

(Bien choisir la version française, car on ne peut pas installer plusieurs versions de langues différentes).

L'utilisateur final de l'exécutable installera le FrameWork (une seule fois pour tous les programmes) et l'exécutable (programme simple).

**Ou est le Framework ?**

Dans :

c:\Windows\Microsoft.NET\Framework\v3.5\

On voit dans les sous-répertoires les DLL qui composent le Framework : System.dll, System.Drawing.dll…

On peut installer à côté la version 1, 2 ,3 ,3.5, 4 ; il faut en installer plusieurs si nécessaire (si on installe différents logiciels qui utilisent des Frameworks de version différente), car il n'y a pas de compatibilité ascendante.
On se souvient que sous Windows 7 les Frameworks de la version 1 à la version 3.5 sont installés.

## XIII-B - Comprendre le code crée par VB

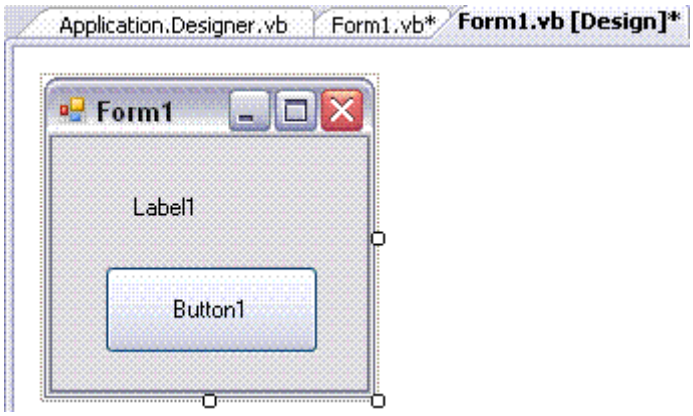
**Comprendre le code crée par VB quand on crée un formulaire ou un contrôle.**

### XIII-B-1 - Code généré automatiquement lors de la création d'un formulaire ou d'un contrôle

Une application 'Windows Forms' est principalement constituée de **formulaires** (ou fenêtre), de **contrôles** et de leurs **événements.**

Effectivement, pendant la création de l'interface utilisateur de votre application, vous créez généralement une fenêtre contenant des contrôles et des procédures événements.

Quand vous créez un nouveau projet 'Windows Forms' cela dessine un formulaire, une fenêtre vide et le code correspondant. Ajoutons-y un bouton cela donne l'interface utilisateur suivante :




Comme on l'a vu, VB crée le code correspondant et dans ce code une Classe correspondant à la fenêtre.

Décortiquons le code.

Vb crée une Class nommé Form1, elle est 'Public' (accessible partout)

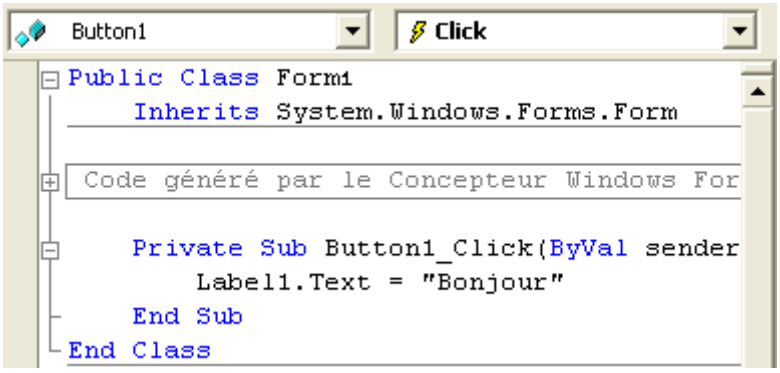
```
Public Class Form1


End Class
```

**En VB.Net 2003**

L'onglet Form1.vb contient la Class.

Cette Classe hérite des propriétés de la Classe Form (celle-ci est fournie par le Framework)

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




(On rappelle que la véritable fenêtre, l'objet sera instancié à partir de cette classe.)

Ensuite il y a une région (partie du code que l'on peut 'contracter' et ne pas voir ou 'dérouler'. Cette région contient : "Le Code généré (automatiquement) par le Concepteur Windows Form", **si on le déroule en cliquant sur le '+', on voit :**

**- le constructeur de la fenêtre :** la routine Sub New

MyBase fait référence à la classe de base,

MyBase.New appelle le constructeur de la classe de base (Form dans notre cas) ;

**- le destructeur de la fenêtre :** la routine Sub Dispose ;

- **le créateur des contrôles** de la fenêtre: la procédure Sub InitializeComponent

Elle est nécessaire pour créer les contrôles et définir les propriétés de ces contrôles:

Exemple : création d'un label Me.Label1= NewSystem.Windows.forms.Label

Modification d'une propriété:Me.Label.Text="Hello"

Elle définit aussi les propriétés du formulaire :

```
    Me.Name = "Form1"
```

Exemple d'un formulaire vide nommé Form1

```
Public Class Form1

Inherits System.Windows.Forms.Form

#Region " Code généré par le Concepteur Windows Form "

Public Sub New()

MyBase.New()

'Cet appel est requis par le Concepteur Windows Form.

InitializeComponent()

'Ajoutez une initialisation quelconque après l'appel InitializeComponent()

End Sub


'La méthode substituée Dispose du formulaire pour nettoyer la liste des composants.

Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)

If disposing Then

If Not (components Is Nothing) Then

components.Dispose()

End If

End If

MyBase.Dispose(disposing)

End Sub

'Requis par le Concepteur Windows Form

Private components As System.ComponentModel.IContainer

'REMARQUE : la procédure suivante est requise par le Concepteur Windows Form

'Elle peut être modifiée en utilisant le Concepteur Windows Form.

'Ne la modifiez pas en utilisant l'éditeur de code.

<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()

'
```

```
'Form1
'
Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
Me.ClientSize = New System.Drawing.Size(292, 266)
Me.Name = "Form1"
Me.Text = "Form1"
End Sub
#End Region

End Class
```

Si dans la fenêtre Design **on ajoute un bouton Button1** cela ajoute le code.

Cette ligne contenant WithEvents indique qu'il y a une gestion d'événement sur les boutons.

```
Friend WithEvents Button1 As System.Windows.Forms.Button
```

Puis dans Sub InitializeComponent()

Cette ligne crée le bouton :

```
Me.Button1 = New System.Windows.Forms.Button
```

Cette ligne le positionne :

```
Me.Button1.Location = New System.Drawing.Point(56, 144)
```

Cette ligne lui donne un nom :

```
Me.Button1.Name = "Button1"
```

Cette ligne détermine sa taille :

```
Me.Button1.Size = New System.Drawing.Size(104, 24)
```

Cette ligne indique ce qui est affiché sur le bouton :

```
Me.Button1.Text = "Button1"
```

Cela donne :

```
Private components As System.ComponentModel.IContainer
Friend WithEvents Button1 As System.Windows.Forms.Button
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
Me.Button1 = New System.Windows.Forms.Button
Me.SuspendLayout()
'
```

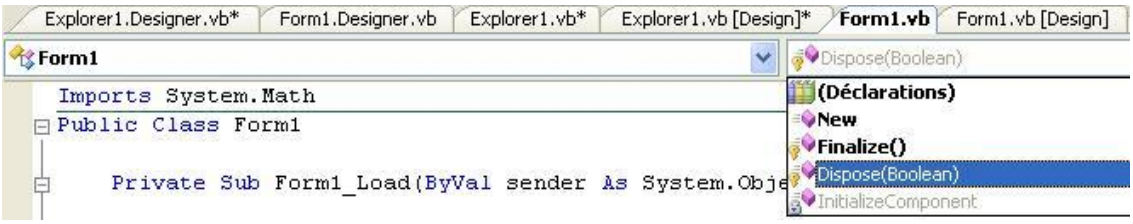
```
'Button1

'

Me.Button1.Location = New System.Drawing.Point(56, 144)

Me.Button1.Name = "Button1"

Me.Button1.Size = New System.Drawing.Size(104, 24)

Me.Button1.TabIndex = 0

Me.Button1.Text = "Button1"
```

### En VB.Net 2005

L'onglet Form1.vb contient :

```
Public Class Form1

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles MyBase.Load

End Sub

Public Sub New()

' This call is required by the Windows Form Designer.

InitializeComponent()    '<=======Appel d'une routine qui 'initialise les composants de la form

' Add any initialization after the InitializeComponent() call.

End Sub

Protected Overrides Sub Finalize()

MyBase.Finalize()

End Sub

Private Sub Label1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
 Label1.Click

End Sub
```

**Le Code généré par le 'Concepteur Windows Form' est par contre caché**, il n'apparait pas dans la Class Form1, il faut pour y avoir accès, passer par le menu de droite :




Si on clique sur InitializeComponent, l'onglet **Form1.Designer.vb** apparait.

On a ainsi accès à InitialiseComponent et à Dispose qui sont dans une **classe Partielle de Form1.**

(En VB 2005, une Classe peut être 'découpée' en Classes partielles.)

C'est ici qu'il est indiqué que la Class hérite de System.Windows.Forms.Form.

Exemple du contenu de Dispose et ItinialiseComponent :

```
Partial Public Class Form1

Inherits System.Windows.Forms.Form

'Form overrides dispose to clean up the component list.

<System.Diagnostics.DebuggerNonUserCode()> _

Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)

If disposing AndAlso components IsNot Nothing Then

components.Dispose()

End If

MyBase.Dispose(disposing)

End Sub

'Required by the Windows Form Designer

Private components As System.ComponentModel.IContainer

'NOTE: The following procedure is required by the Windows Form Designer

'It can be modified using the Windows Form Designer.

'Do not modify it using the code editor.

<System.Diagnostics.DebuggerStepThrough()> _

Private Sub InitializeComponent()

Me.Button1 = New System.Windows.Forms.Button

Me.Label1 = New System.Windows.Forms.Label

Me.SuspendLayout()

'

'Button1

'

Me.Button1.FlatStyle = System.Windows.Forms.FlatStyle.System

Me.Button1.Location = New System.Drawing.Point(47, 38)

Me.Button1.Name = "Button1"

Me.Button1.Size = New System.Drawing.Size(177, 42)

Me.Button1.TabIndex = 0

Me.Button1.Text = "Button1"

….
```

### En vb 2010

Il y a dans Form1.vb (Form1(événement)) la classe partielle Form1 et la Sub New (le constructeur de la classe Form1)qui appelle InitializeComponent().

```
Public Class Form1
```

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles MyBase.Load

    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
 Button1.Click

    End Sub

    Public Sub New()

        ' Cet appel est requis par le concepteur.
        InitializeComponent()

        ' Ajoutez une initialisation quelconque après l'appel InitializeComponent().

    End Sub
End Class
```

En ouvrant Form1 sous l'onglet, si on déroule la liste à droite on découvre les routines New, Finalise, Dispose,InitializeComponent() :

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Partial Class Form1
    Inherits System.Windows.Forms.Form

    'Form remplace la méthode Dispose pour nettoyer la liste des composants.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        Try
            If disposing AndAlso components IsNot Nothing Then
                components.Dispose()
            End If
        Finally
            MyBase.Dispose(disposing)
        End Try
    End Sub

    'Requise par le Concepteur Windows Form
    Private components As System.ComponentModel.IContainer

    'REMARQUE : la procédure suivante est requise par le Concepteur Windows Form
    'Elle peut être modifiée à l'aide du Concepteur Windows Form.
    'Ne la modifiez pas à l'aide de l'éditeur de code.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Me.Button1 = New System.Windows.Forms.Button()
        Me.Label5 = New System.Windows.Forms.Label()
        Me.SuspendLayout()
        '
        'Button1
        '
        Me.Button1.Location = New System.Drawing.Point(16, 46)
        Me.Button1.Name = "Button1"
        Me.Button1.Size = New System.Drawing.Size(261, 35)
        Me.Button1.TabIndex = 0
        Me.Button1.Text = "Button1"
        Me.Button1.UseVisualStyleBackColor = True
        '
        'Label5
        '
        Me.Label5.AutoSize = True
        Me.Label5.Location = New System.Drawing.Point(69, 130)
        Me.Label5.Name = "Label5"
        Me.Label5.Size = New System.Drawing.Size(39, 13)
        Me.Label5.TabIndex = 1
        Me.Label5.Text = "Label1"
        '
        'Form1
```

```
        '
        Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
        Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
        Me.ClientSize = New System.Drawing.Size(284, 262)
        Me.Controls.Add(Me.Label5)
        Me.Controls.Add(Me.Button1)
        Me.Name = "Form1"
        Me.Text = "Form1"
        Me.ResumeLayout(False)
        Me.PerformLayout()

    End Sub
    Friend WithEvents Button1 As System.Windows.Forms.Button
    Friend WithEvents Label5 As System.Windows.Forms.Label

End Class
```

'Partial Class Form1' indique qu'on est dans la classe partielle Form1.
'Inherits System.Windows.Forms.Form' indique que Form1 hérite de la Classe Form du Framework.
La Sub 'Dispose' servira à libérer les composants quand on fermera la form. C'est le destructeur.
La Sub 'InitializeComponent()' permet de construire la form (C'est le constructeur); pour chaque composant il y a du code indiquant le nom du composant et ses propriétés.

```
'Création du bouton
        Me.Button1 = New System.Windows.Forms.Button()

'Propriété du bouton
        Me.Button1.Location = New System.Drawing.Point(16, 46)
        Me.Button1.Name = "Button1"
        Me.Button1.Size = New System.Drawing.Size(261, 35)
        Me.Button1.TabIndex = 0
        Me.Button1.Text = "Button1"
        Me.Button1.UseVisualStyleBackColor = True

 'Permet au bouton de recevoir des événements
        Friend WithEvents Button1 As System.Windows.Forms.Button
```

Ce code est donc généré automatiquement. Ne pas y toucher !!

Pour VB.Net 2003 et 2005

**Les procédures événements** correspondant au formulaire par exemple sont automatiquement créées :

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles MyBase.Load

End Sub
```

**Attention :** Form1_Load est un nom de Sub donné par VB, mais cela pourrait être n'importe quoi, ce qui indique l'événement déclencheur est ce qui est après Handles ( MyBase.Load ici).

Les 2 paramètres sont :

sender l'objet à la source de l'événement (l'objet qui a déclenché l'événement: ici la form) ;

e un objet EventArgs qui détaille l'événement qui s'est produit et fournissant des informations sur cet événement.

On constate qu'il y a une liaison entre la fenêtre Design et le code généré ; on pourrait modifier dans le code l'interface utilisateur. **C'est déconseillé d'aller trafiquer dans cette zone de "Code généré par le Concepteur Windows Form"** , il faut plutôt faire des modifications dans la partie design et dans la fenêtre de propriété.

# XIII-B-2 - Substitution de procédures événement

Il est possible de substituer une méthode (utiliser sa propre méthode à la place de la méthode normale qui existe normalement dans un contrôle).

**Exemple 1 créer un contrôle simple affichant toujours 'Bonjour'**

Il faut créer une classe héritant des 'Control', détourner son événement OnPaint qui dessine le contrôle (avec Overides). Dans la nouvelle procédure On Paint simplement afficher 'Bonjour'

```
Public Class ControleAffichantBonjour

Inherits Control

Overrides Protected Sub OnPaint ( e As PaintEventArgs )

    e.Graphics.DrawString ("Bonjour", Font, new SolidBrush(ForeColor)

End Sub

End Class
```

Cet exemple ne sert strictement à rien !! Pour une fois !!

Il est aussi possible de détourner des événements.

Dans le chapitre **'Impression'** il y a un bel exemple de création de "lien" entre un objet printdocument et la routine événement PrintPage (imprimer hello avec un printdocument)

**Exemple 2 créer un bouton personnalisé**

1-Il faut créer une classe qui hérite de Button :

```
Public Class MonBouton

Inherits System.Windows.Forms.Button

End Class
```

Le 'Design' devient :


2-Il faut modifier l'aspect graphique du bouton.

Pour cela si vous voulez modifier l'apparence du contrôle, il faut remplacer la méthode OnPaint de Button par la vôtre( celle-ci dessine le contrôle). Au sein de cette méthode, vous devez appeler la méthode OnPaint de la base (de la classe mère), puis ajouter vos propres fonctions de dessin.

Il faut donc ajouter dans la classe, la procédure :

```
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)

MyBase.OnPaint(e) 'Appel à la méthode de la classe de base, ce qui dessine le bouton

Dim myPen As New Pen(Color.Purple, 3)
```

```
e.Graphics.DrawRectangle(myPen, 3, 3, Me.Width - 6, Me.Height - 6) 'Ajoute un cadre sur le dessin
 du bouton

End Sub
```

On rappelle que l'argument e est le graphique du bouton.

**Dans le chapitre suivant, on va utiliser ces connaissances pour, dans le code, créer soi-même des contrôles et leurs événements.**

## XIII-C - Les délégués, les événements

**Super complexe ? Non !! mais vous pouvez sauter ce chapitre.**

### XIII-C-1 - Définition

Un délégué est une référence (un **type référence**) qui fait référence à une méthode, qui pointe sur une méthode. L'équivalent le plus proche d'un délégué dans d'autres langages est le pointeur de fonction.

On peut créer directement un délégué avec le mot Delegate et AddressOf.

Dans la gestion des événements des contrôles, on crée aussi des délégués avec Handles et AddHandler.

### XIII-C-2 - Création d'un délégué avec 'Delegate'

Ici on ne parle pas d'événement.

On déclare un délégué (un pointeur) :

```
Delegate Sub D()
```

on l'instance : on le fait pointer sur une fonction F().

```
Dim M As New D(AddressOf F)
```

Quand on utilise l'instance du délégué, cela exécute la fonction.

M() exécute la fonction F()

AddressOf permet de préciser l'adresse de la procédure F(), le pointeur vers la procédure F().

Exemple hyper simple :

```
Delegate Sub SimpleDelegate() 'On crée un délégué sans paramètres

Module Test

   Sub MaSub() 'On crée une Sub
      System.Console.WriteLine("Test")
   End Sub

   Sub Main()
      Dim Mondelegue As New SimpleDelegate(AddressOf MaSub) 'le délégué pointe sur la Sub
      Mondelegue()     'On utilise le délégué
   End Sub
End Module
```

Il n'est pas d'un grand intérêt d'instancier un délégué pour une méthode et d'appeler ensuite immédiatement la méthode via le délégué, puisqu'il serait plus simple d'appeler directement la méthode, mais c'est un exemple.

Exemple avec une fonction et des paramètres

Bien sûr le délégué peut pointer vers une Sub ou une fonction avec des paramètres.

Dans une Classe MaClasse : on déclare un delegate en indiquant paramètres envoyés et de retour :

```
Delegate Function maMethodDelegate(myInt As Integer) As [String]
```

On déclare une fonction :

```
Public Shared Function maMethode(myInt As Integer) As [String]

    Return myInt.ToString
End Function
```

Ici l'exemple est bête : on donne un Integer, cela retourne une String !!

On crée le délégué :

```
Dim myD As New maMethodDelegate(AddressOf maClasse.maMethode)
```

myD est maintenant un pointeur sur maClasse.maMethode.

Je peux utiliser myD(2) qui retournera une String "2".

Intérêts d'un délégué par rapport à une fonction ?

Un délégué est un type référence qui fait référence à une méthode **Shared** d'un type ou à une méthode d'instance d'un objet.

**Un délégué peut référencer à la fois des méthodes de classe (static) et des méthodes d'instance.** Lorsque le délégué référence une méthode d'instance, il stocke non seulement une référence au point d'entrée de la méthode, mais également une référence à l'instance de classe pour laquelle la méthode est appelée. Contrairement aux pointeurs fonction, les délégués sont orientés objet, de type sécurisé et fiables.

Voici un exemple de Microsoft utilisant un délégué avec les 2 types de méthodes :

```
Imports System

Public Class SamplesDelegate

' Declares un delegate à partir d'une méthode qui accepte un paramètre integer et retourne une
 String.
Delegate Function myMethodDelegate(myInt As Integer) As [String]

' Definir les méthodes.
Public Class mySampleClass

' Definir une méthode d'instance.
Public Function myStringMethod(myInt As Integer) As [String]
If myInt > 0 Then
Return "positive"
End If
If myInt < 0 Then
Return "negative"
End If
Return "zero"
```

```
End Function 'myStringMethod

' Definir une méthode de classe.
Public Shared Function mySignMethod(myInt As Integer) As [String]
If myInt > 0 Then
Return "+"
End If
If myInt < 0 Then
Return "-"
End If
Return ""
End Function 'mySignMethod
End Class 'mySampleClass

'Utilisation du délégué

Public Shared Sub Main()

' Instanciation de délégué pour chaque méthode.
Dim mySC As New mySampleClass()
Dim myD1 As New myMethodDelegate(AddressOf mySC.myStringMethod)
Dim myD2 As New myMethodDelegate(AddressOf mySampleClass.mySignMethod)

'Utilisation des délégués.
Console.WriteLine("{0} is {1}; use the sign ""{2}"".", 5, myD1(5), myD2(5))
Console.WriteLine("{0} is {1}; use the sign ""{2}"".", - 3, myD1(- 3), myD2(- 3))
Console.WriteLine("{0} is {1}; use the sign ""{2}"".", 0, myD1(0), myD2(0))

End Sub 'Main

End Class 'SamplesDelegate


'Le code produit les sorties suivantes:
'
'5 is positive; use the sign "+".
'-3 is negative; use the sign "-".
'0 is zero; use the sign "".
```

Les membres d'un délégué sont les membres hérités de la classe **System.Delegate**. Un délégué contient également un ensemble de constructeurs et de méthodes définis par le système.

**L'utilité des délégués réside dans leur anonymat**. L'exemple suivant illustre une méthode MultiCall qui appelle de façon répétitive une instance SimpleDelegate :

```
Sub MultiCall(d As SimpleDelegate, count As Integer)
      Dim i As Integer
      For i = 0 To count - 1
         d()
      Next i
   End Sub
```

Pour la méthode MultiCall, l'identité de la méthode cible de SimpleDelegate n'a pas d'importance, pas plus que l'accessibilité qu'a cette méthode, ni le fait qu'il s'agisse d'une méthode **Shared** ou non partagée.

**La seule chose qui importe est que la signature de la méthode soit compatible** avec SimpleDelegate.

## XIII-C-3 - Délégué et appel asynchrone

Quand une procédure A appelle une autre procédure B, cela se passe de manière **synchrone**: pendant l'exécution de la procédure B, la procédure A est en attente, elle se poursuit après le retour de la procédure B.

Si vous appelez la procédure B à partir d'un délégué avec BeginInvoke , le fonctionnement sera **asynchrone**, c'est-à-dire que les 2 procédures se dérouleront en parallèle.

Si la méthode **BeginInvoke** est appelée, le Common Language Runtime mettra la demande en file d'attente et le retour à l'appelant sera immédiat. La méthode cible sera appelée sur un autre thread. Le thread d'origine, qui a soumis la demande, est libre de poursuivre en parallèle son exécution. La méthode **BeginInvoke** est donc utilisée pour établir l'appel asynchrone. Elle possède les mêmes paramètres que la méthode à exécuter de façon asynchrone, plus deux paramètres. Le premier paramètre supplémentaire sert quand l'appel asynchrone se termine (voir plus bas) ; le second paramètre supplémentaire sert à fournir un objet quelconque.

```
Delegate Function myDelegate(myInt As Integer) As [String]

Dim ta As New myMethodDelegate(AddressOf MyMethode)

ta.BeginInvoque(2 ,Nothing, Nothing)  'On met les 2 paramètres supplémentaires à Nothing pour le
 moment.
```

Ici on a exécuté dans un autre thread, en parallèle, MyMethode.

On reprend l'exemple bête, d'une fonction qui transforme un integer en String.

```
Public Shared Function myMethode(myInt As Integer) As [String]

    Return myInt.ToString
End Function
```

**Le problème est de savoir quand MyMethode se termine et éventuellement récupérer les résultats de MyMethode.**

Là intervient **le premier paramètre supplémentaire :** il est chargé d'indiquer l'adresse de la procédure à exécuter lorsque l'opération asynchrone se termine.

```
ta.BeginInvoque(2 ,AdresssOf AfficheResultat, Nothing)
```

Ici quand ta se termine cela exécute la procédure AfficheResultat.

Il faut donc créer une procédure AfficheResultat avec comme paramètre une variable de type IAsyncResult qui permet de récupérer le résultat.

```
Sub AfficheResultat (By Val ia As IAsyncResult)

    Dim Resultat As String

    Resultat= ta.EndInvoke(ia)

End Sub
```

La méthode **EndInvoke** est utilisée pour obtenir la valeur de retour de la fonction et les paramètres de sortie.

**On peut aussi 'surveiller' si ta est terminé et récupérer le résultat.**

Il faut déclarer une variable de type IasyncResult :

```
Dim ia As IAsyncResult
```

Quand on instance ta avec BeginInvoke cela retourne un IAsyncResult si on écrit :

```
ia= ta.BeginInvoque(2 ,Nothing, Nothing)
```

Il suffit de tester ia avec ia.IsCompleted pour voir si ta est terminé; à titre d'exemple, créons un bouton testant si la procédure asynchrone est terminée.

```
Private Sub Button1_Click(…)

    Dim Resultat As String

    If ia.IsCompleted Then

        Resultat=ta.EndInvoke(ia)

    Else

        MsgBox(" Traitement en cours…")

    End If

End Sub
```

Il faut cliquer sur le bouton à intervalle régulier pour voir si ta est terminé. L'exemple est débile, car tout cela à de l'intérêt que si la procédure asynchrone est très longue.

## XIII-C-4 - Délégué et événement

Les *délégués sont utilisés* pour lier les événements aux méthodes servant à les gérer. Lorsque l'événement se produit, le délégué appelle la méthode liée.

Délégué crée automatiquement par Visual Basic .NET

Dans le cas d'événements associés à des formulaires ou contrôles, Visual Basic .NET crée automatiquement un gestionnaire d'événements et l'associe à un événement.

Effectivement, en mode design, lorsque vous double-cliquez sur un bouton de commande dans un formulaire, Visual Basic .NET crée automatiquement un gestionnaire d'événements vide et une variable WithEvents pour le bouton de commande, comme dans le code suivant.

Dans la Region "Code généré par le Concepteur Windows Form" il y a :

```
Friend WithEvents Button1 As System.Windows.Forms.Button


Protected Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click


End Sub
```

WithEvents indique que l'objet Button1 a des événements.

Le terme Handles provoque l'association d'un événement (Button1.Click situé après Handles) à un gestionnaire d'événements (la Sub Button1_Click ; la Sub pourrait d'ailleurs se nommer différemment, cela n'a pas d'importance).

L'association d'événements aux gestionnaires d'événements se fait au moment de la compilation et ne peut pas être modifiée.

Lorsque vous cliquez sur un Button1 cela déclenche bien la Sub Button1_Click.

Délégué et événements créés par vous

On reprend les mêmes concepts que dans le chapitre sur la création de contrôles par code.

Lorsque vous créez par code de toutes pièces des contrôles, vous pouvez faire de même avec Handles.

Déclaration **dans la partie déclaration du module**(en haut) :

```
Private WithEvents Button1 As  New Button

Me.Controls.Add(Button1)

Sub OnClique ( sender As Object, EvArg As EventArgs) Handles Button1.Click

End Sub
```

La aussi, l'association d'événements aux gestionnaires d'événement se fait **au moment de la compilation** et ne peut pas être modifiée. On a bien créé un délégué.

Vous pouvez aussi utiliser la méthode AddHandler :

```
    Dim TB As New System.Windows.Forms.TextBox

     Me.Controls.Add(TB)

     AddHandler TB.Keyup, AddressOf TextboxKeyup.


Sub TextboxKeyup.(ByVal sender As Object, ByVal e As KeyEventArgs)

….

End Sub
```

AddHandler permet donc d'associer à l'événement TB.Keyup la Sub TextboxKeyup. On a bien créé un délégué.

le mot-clé **addhandler** permet d'associer un événement à une procédure **au moment de l'exécution et peut être annulé par RemoveHandler.**

# XIV - Diffuser le programme

## XIV-A - Assembly

Avant VB.Net, on enregistrait les références des programmes, des dll… dans le registre, l'installateur enregistrait le nom et l'emplacement du fichier .exe ou de la Dll ; c'était une belle pagaille :

quand on installait 2 dll différentes de même nom ;

quand il y avait plusieurs versions d'une même dll ;

quand on déplaçait un programme !!

quand on mettait à jour une dll qui ne respectait pas la compatibilité ascendante.

Maintenant cela ne se fait plus ; en VB.Net on utilise les **Assembly**.

### XIV-A-1 - Assembly : définition, composition

Un Assembly est une unité de déploiement indivisible.

Il se caractérise par son identité (propriétés de l'assembly) :

- un nom ;

- une version ;

- un identificateur de culture ;

- une clé publique.

Il contient :

la liste de l'ensemble des fichiers (exe, dll, données, images, ressources) ;

les métadonnées (informations descriptives des Types et Classes publiques) ;

l'énumération des autres Assembly dont l'application dépend et leurs dépendances ;

l'ensemble des autorisations requises pour que l'assembly fonctionne correctement.

Ces informations sont utilisées au moment de l'exécution pour résoudre les références, appliquer la stratégie de liaison des versions et valider l'intégrité des assemblys chargés.

Toutes ces informations sont stockées dans le "manifeste" de l'Assembly.

En conclusion

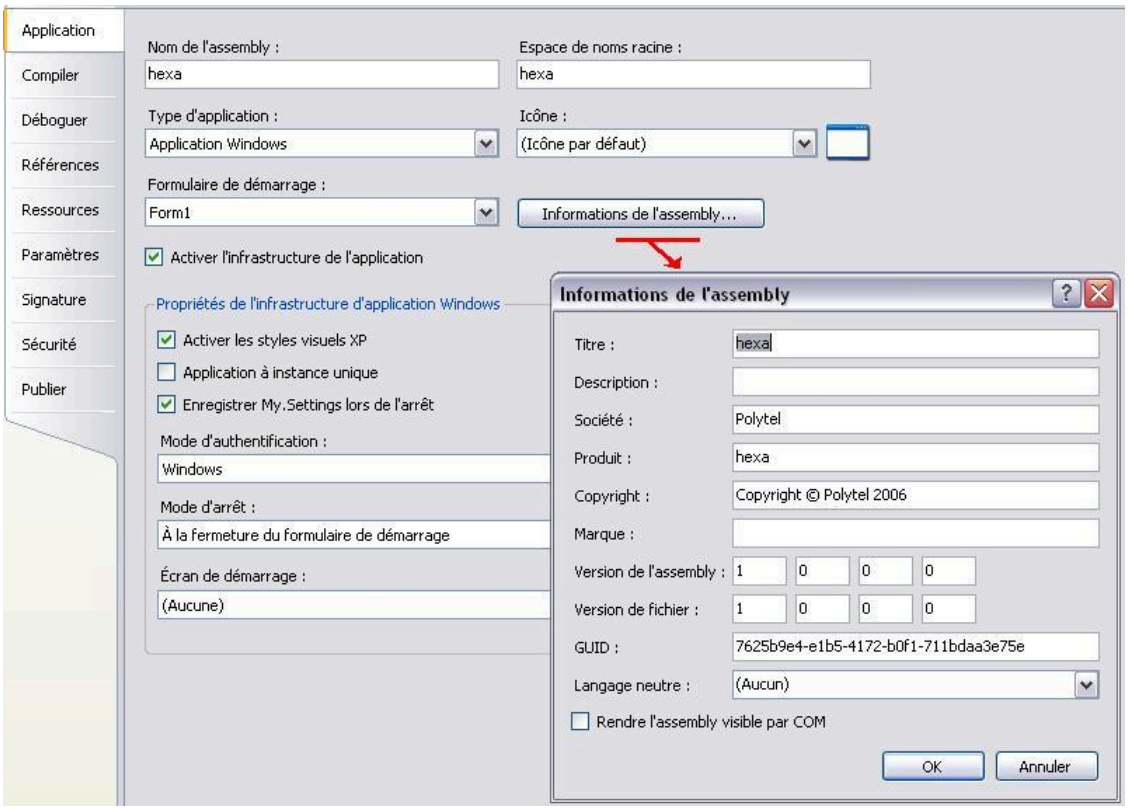
Pour les installations de programme, mises à jour, utilisation de composants propres au programme ou partagés avec d'autres programmes ; pour gérer les versions, éviter les problèmes de conflit de composants, VB.Net utilise donc les assembly (ou assemblage).

# XIV-A-2 - Les propriétés de l'assembly

Voir les propriétés de l'Assembly

Pour cela, ouvrir les propriétés du projet (cliquer sur MyProjet dans l'explorateur de solution ou passer par le menu Projet->Propriétés de…)

Dans l'onglet Application, cliquer sur le bouton 'Informations de l'assembly' :




On a accès au titre, à la description, à la société, au produit, au copyright, à la marque, à la version de l'assembly, à la version du fichier, au GUID, à la langue.

On peut aussi le voir en XML : dans l'explorateur de solution, double-cliquer sur Assemblyinfo.vb, la fenêtre principale s'ouvre permettant d'avoir accès à certaines données :

```
Imports System

Imports System.Reflection

Imports System.Runtime.InteropServices

' Les informations générales relatives à un assembly dépendent de

' l'ensemble d'attributs suivant. Changez les valeurs de ces attributs pour modifier les
 informations

' associées à un assembly.

' Vérifiez les valeurs des attributs de l'assembly

<Assembly: AssemblyTitle("Bonjour")>

<Assembly: AssemblyDescription("")>
```

```
<Assembly: AssemblyCompany("Polytel")>

<Assembly: AssemblyProduct("Bonjour")>

<Assembly: AssemblyCopyright("Copyright &#184; Polytel 2006")>

<Assembly: AssemblyTrademark("")>

<Assembly: ComVisible(False)>

'Le GUID suivant est pour l'ID de la typelib si ce projet est exposé à COM

<Assembly: Guid("9a8cb33c-3392-44a0-a86d-c7164dfa91c1")>

' Les informations de version pour un assembly se composent des quatre valeurs suivantes :

'

' Version principale

' Version secondaire

' Numéro de build

' Révision

'

' Vous pouvez spécifier toutes les valeurs ou indiquer les numéros de build et de révision par
 défaut

' en utilisant '*', comme indiqué ci-dessous :

' <Assembly: AssemblyVersion("1.0.*")>

<Assembly: AssemblyVersion("1.0.0.0")>

<Assembly: AssemblyFileVersion("1.0.0.0")>
```

## XIV-A-3 - Le manifeste

Toutes les informations de l'assembly sont stockées dans le "manifeste".

Le manifeste qui est un fichier en XML se trouve dans :

myapplication\myapplication\bin\debug\myapplication.publish\myapplication_1_0_0_0\myapplivation.exe.manifest

myapplication\myapplication\publish\myapplication_1_0_0_0\myapplivation.exe.manifest

Pour info, voici un exemple de contenu :

```
<?xml version="1.0" encoding="utf-8"?>

<asmv1:assembly xsi:schemaLocation="urn:schemas-microsoft-com:asm.v1 assembly.adaptive.xsd"
 manifestVersion="1.0"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" xmlns="urn:schemas-microsoft-com:asm.v2"
xmlns:asmv1="urn:schemas-microsoft-com:asm.v1" xmlns:asmv2="urn:schemas-microsoft-com:asm.v2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<asmv1:assemblyIdentity name="myapplication.exe" version="1.0.0.0"
 publicKeyToken="612c3b94c96b9edf"
language="neutral" processorArchitecture="msil" type="win32" />

<application />

<entryPoint>
```

```
<assemblyIdentity name="myapplication" version="1.0.0.0" language="neutral"
 processorArchitecture="msil" />

<commandLine file="myapplication.exe" parameters="" />

</entryPoint>

<trustInfo>

<security>

<applicationRequestMinimum>

<PermissionSet Unrestricted="true" ID="Custom" SameSite="site" />

<defaultAssemblyRequest permissionSetReference="Custom" />

</applicationRequestMinimum>

</security>

</trustInfo>

<dependency>

<dependentOS>

<osVersionInfo>

<os majorVersion="4" minorVersion="10" buildNumber="0" servicePackMajor="0" />

</osVersionInfo>

</dependentOS>

</dependency>

<dependency>

<dependentAssembly dependencyType="preRequisite" allowDelayedBinding="true">

<assemblyIdentity name="Microsoft.Windows.CommonLanguageRuntime" version="2.0.50727.0" />

</dependentAssembly>

</dependency>

<dependency>

<dependentAssembly dependencyType="install" allowDelayedBinding="true"
 codebase="myapplication.exe" size="28672">

<assemblyIdentity name="myapplication" version="1.0.0.0" language="neutral"
 processorArchitecture="msil" />

<hash>

<dsig:Transforms>

<dsig:Transform Algorithm="urn:schemas-microsoft-com:HashTransforms.Identity" />

</dsig:Transforms>

<dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />

<dsig:DigestValue>IK0J8Ge5ABv5RfyMrgdRoMoy/Gc=</dsig:DigestValue>

</hash>

</dependentAssembly>

</dependency>
```

```
<publisherIdentity name="CN=CABINET\Philippe"
 issuerKeyHash="6d35a155f760c5d6ce1866b24dc5b27e833af918" />
<Signature Id="StrongNameSignature" xmlns="http://www.w3.org/2000/09/
xmldsig#"><SignedInfo><CanonicalizationMethod

.....

.....
```

Vous n'avez pas à l'ouvrir et à le modifier.

Signature d'un Assembly : Article par webman sur developpez.com :

http://webman.developpez.com/articles/dotnet/assemblysigning/

## XIV-B - Distribuer l'application

Comment distribuer une application VB.NET avec les outils Microsoft ?

Il faut la "déployer".

- Introduction
- Avant de publier
- Installation simple
- Exemple Windows Installer en VB 2003
- Exemple ClickOnce en VB 2005
- Autres programmes d'installation

### XIV-B-1 - Introduction

Microsoft propose 2 modes d'installation des logiciels.

- Le déploiement avec un programme d'installation traditionnel à l'aide de la technologie **Windows Installer**.
  Avec le déploiement Windows Installer, vous empaquetez l'application dans un fichier setup.exe et distribuez ce fichier aux utilisateurs ; ceux-ci exécutent le fichier Setup.exe pour installer l'application.
  Les fichiers du programme d'installation peuvent être distribués sur des disquettes ou des CD-ROM, ou peuvent être placés sur un lecteur réseau pour une installation sur un réseau.
  Pour déployer une application, vous créez d'abord un projet d'installation et définissez les propriétés du projet.
  Ce mode de déploiement est disponible en VB 2003 (c'est le seul d'ailleurs en VB 2003) en VB 2005 et en VB 2008 (**sauf pour la version Express**).
  Voir ci-dessous un exemple en VB 2003.
- La publication d'une application à l'aide de la technologie **ClickOnce**
  Avec le déploiement ClickOnce, c'est très simple vous publiez l'application à un emplacement centralisé et l'utilisateur l'installe ou l'exécute à partir de cet emplacement.
  Les applications déployées avec ClickOnce se mettent à jour automatiquement et représentent le meilleur choix pour les applications exigeant des modifications fréquentes.
  Vous utilisez l'Assistant Publication pour empaqueter votre application et la publier sur un site Web ou un partage de fichiers réseau ; l'utilisateur installe et lance directement l'application à partir de cet emplacement en une seule étape.
  Ce mode de déploiement est disponible en VB 2005 et VB 2008 (c'est le seul dans la version Express).
  Ce type d'installation convient bien pour créer des installations à partir d'Internet.
  Voir ci-dessous un exemple en VB 2005.

## XIV-B-2 - Avant de 'Publier'

ATTENTION

Avant de publier votre programme, assurez-vous que vous l'avez testé et qu'il s'exécute sans erreur. Créer un fichier d'aide.

On peut choisir le mode **Release** ou le mode **Debug**.

En VB 2003, en haut de la fenêtre de l'IDE il y a une liste déroulante elle contient :

**Release** (à utiliser pour générer la version à distribuer) ;

**Debug** (à utiliser pour générer une version à tester).

En VB 2005 et VB 2010, si vous choisissez les paramètres de développement Visual Basic, l'outil qui permet de choisir entre la configuration Debug et Release n'apparait pas dans la barre d'outils. Visual Studio choisit automatiquement la configuration Debug lorsque vous cliquez sur Démarrer dans le menu Débogueur et la configuration Release lorsque vous utilisez le menu Générer.

On peut aussi utiliser une constante nommé DEBUG qui aura la valeur True si on est en mode Debug.

Cela permet d'écrire :

```
#If Debug Then

    Stop

#End If
```

Ici Stop se produira uniquement si on est en mode Debug; en mode Release, il n'y aura pas d'arrêt.

Puis, il faut **générer** en utilisant le Menu Générer-> Générer la Solution.

Le programme exécutable ainsi crée se trouve dans le répertoire \bin.

## XIV-B-3 - Comment installer simplement un programme chez l'utilisateur ?

S'il s'agit d'un programme exe simple isolé, sans dll ou avec des dll locales non partagées par d'autres programmes : on peut l'installer 'à la main'.

- Il faut installer le Framework.NET sur l'ordinateur de destination.
- Copier le répertoire \bin contenant l'exécutable (et éventuellement les fichiers de données et les dll) dans un répertoire destination (avec XCopy ou avec l'exploreur). On peut aussi le mettre sur un CD puis à partir du CD copier dans un répertoire sur un autre ordinateur.
- Pour utiliser le programme, l'utilisateur lance l'exécutable.

On peut créer un raccourci permettant de lancer le programme : dans l'explorateur, cliquer sur le fichier.exe, puis clic droit, cela ouvre un menu, cliquer sur 'Créer un raccourci'. Ensuite ce raccourci peut être déplacé sur le bureau par drag and drop.

Cette méthode ne prend pas en compte les composants, dll partagées avec d'autres applications. **Il faut plutôt créer un programme d'installation** qui est nécessaire dans les autres cas.

# XIV-B-4 - Créer un programme d'installation classique en VB 2003 (de type Windows Installer)

Le déploiement avec un programme d'installation traditionnel peut être effectué à l'aide de la technologie Windows Installer. Avec le déploiement Windows Installer, vous empaquetez l'application dans un fichier setup.exe et distribuez ce fichier aux utilisateurs ; ceux-ci exécutent le fichier Setup.exe pour installer l'application.

Pour cela, vous devez **ajouter un projet d'installation à votre solution** et définir les propriétés du projet de déploiement afin de créer un fichier d'installation distribué aux utilisateurs. Les fichiers du programme d'installation peuvent être distribués sur des supports traditionnels, comme les disquettes ou les CD-ROM, ou peuvent être placés sur un lecteur réseau pour une installation sur un réseau.

Pour un déploiement via des supports traditionnels, vous copiez les fichiers à partir de l'emplacement de génération vers une disquette ou autre support.

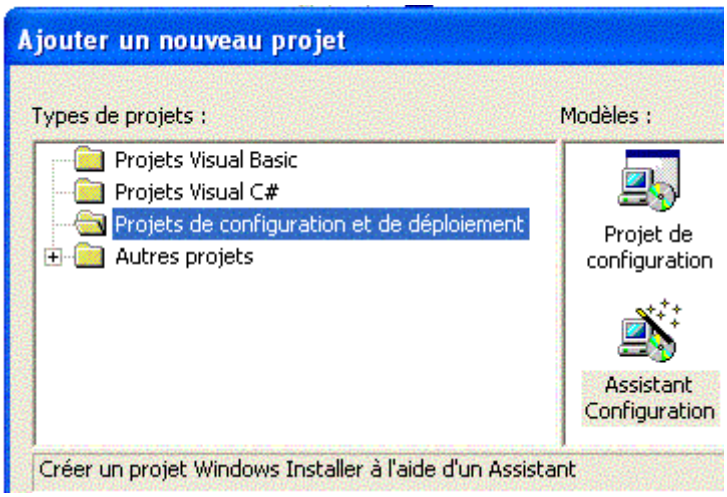
L'utilisateur lance Setup.exe qui est sur un CD d'installation et ce programme installe le logiciel.

Voyons cela dans VB 2003.

Pour cela, il faut créer un projet de configuration et déploiement, en modifier certaines propriétés puis le générer.

Menu Fichiers->Ajouter un projet->Nouveau Projet-> Cliquez dans la liste sur 'Projet de configuration et de déploiement.' Puis sur l'icône 'Assistant de configuration'.

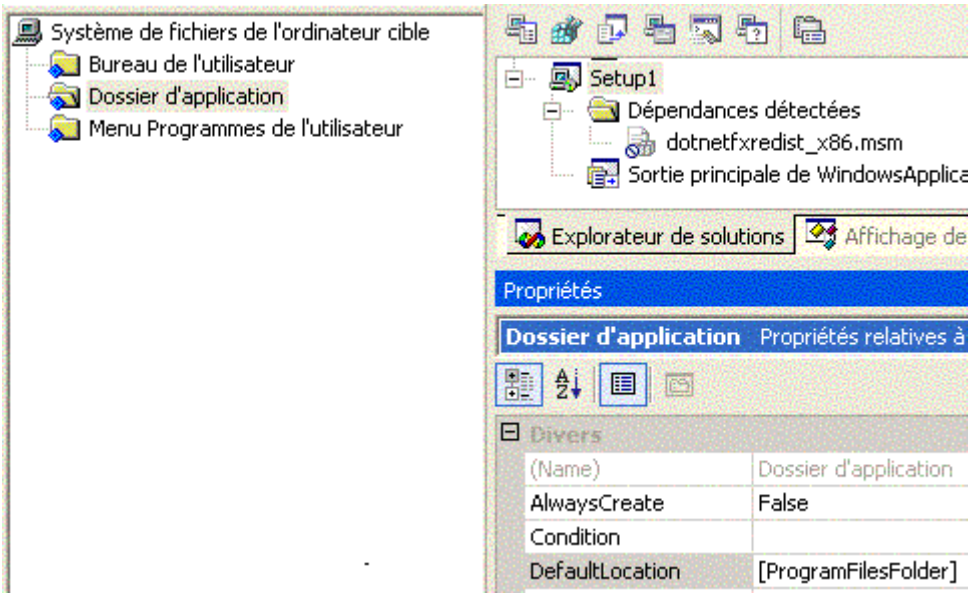
Il faut vérifier en bas de la fenêtre 'Ajouter un nouveau projet' le chemin.




Suivez les divers écrans en vous rappelant que vous utilisez une application Windows en sortie principale, n'oubliez pas de rajouter si nécessaire certains fichiers (les fichiers de données nécessaires).

Après le bouton 'Terminez', il est ajouté dans la fenêtre de l'explorateur de solution une ligne nommée par défaut 'Setup1' correspondant au projet de l'installateur. Il est créé un onglet 'Système de fichiers' dans la fenêtre principale.

Vous venez de créer votre projet de configuration et déploiement, vous pouvez maintenant le modifier.

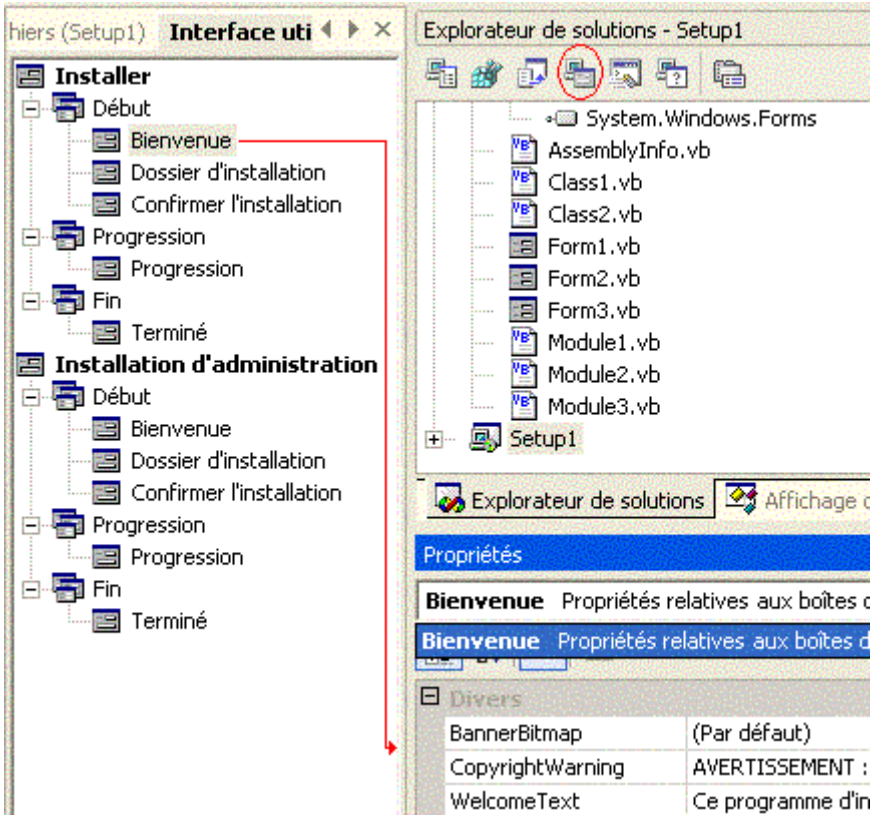



Le fait de cliquer sur le 'dossier d'application' dans l'explorateur de solution affiche dans la fenêtre de propriétés, les propriétés de l'installation.

La propriété DefaultLocation donne par exemple l'emplacement, le répertoire d'installation. Il y a bien d'autres propriétés permettant de personnaliser votre installateur (Auteur, nom de l'entreprise, Version…).

Enfin quand on clique sur Setup1 dans l'explorateur de solutions, il apparait des boutons donnant accès à des éditeurs de registre, de l'interface de l'installateur, de condition de lancement…

Si on clique sur le 3e bouton on ouvre l'éditeur d'interface qui donne accès au déroulement de l'installateur. En cliquant sur la première fenêtre ('Bienvenue') on a accès aux propriétés de cette fenêtre : texte, image…



Pour créer effectivement l'installateur, il faudra enregistrer puis utiliser le Menu Générer-> Générer Setup1.

Un répertoire nommé dans notre exemple 'SeptUp1' est créé, il contient :

Setup.exe ;

Setup1.msi ;

Setup.ini.

il suffit de mettre ces fichiers sur un CD et de le donner à l'utilisateur final qui installera votre logiciel en lançant Setup.exe.

Le logiciel d'installation vérifie si le FrameWork est bien installé.

## XIV-B-5 - Créer un programme d'installation 'ClickOnce' en VB 2005

Avec le déploiement ClickOnce, vous publiez l'application à un emplacement centralisé et l'utilisateur l'installe ou l'exécute à partir de cet emplacement. ClickOnce se base sur le protocole HTTP pour effectuer les installations ou les mises à jour.

L'emplacement centralisé peut-être une page WEB.

Les applications déployées avec ClickOnce se mettent à jour automatiquement et représentent le meilleur choix pour les applications exigeant des modifications fréquentes.

Avec ClickOnce, vous utilisez l'Assistant Publication pour empaqueter votre application et la publier sur un site Web ou un partage de fichiers réseau ; l'utilisateur installe et lance directement l'application à partir de cet emplacement en une seule étape.

**Exemple d'installation à partir d'un CD-ROM sans mise à jour**

- Lancer l'assistant de publication.

Une fois que vous êtes prêt à le publier code vérifié, génération effectuée), vous pouvez lancer l'Assistant Publication en sélectionnant la commande Publier dans le menu Générer.

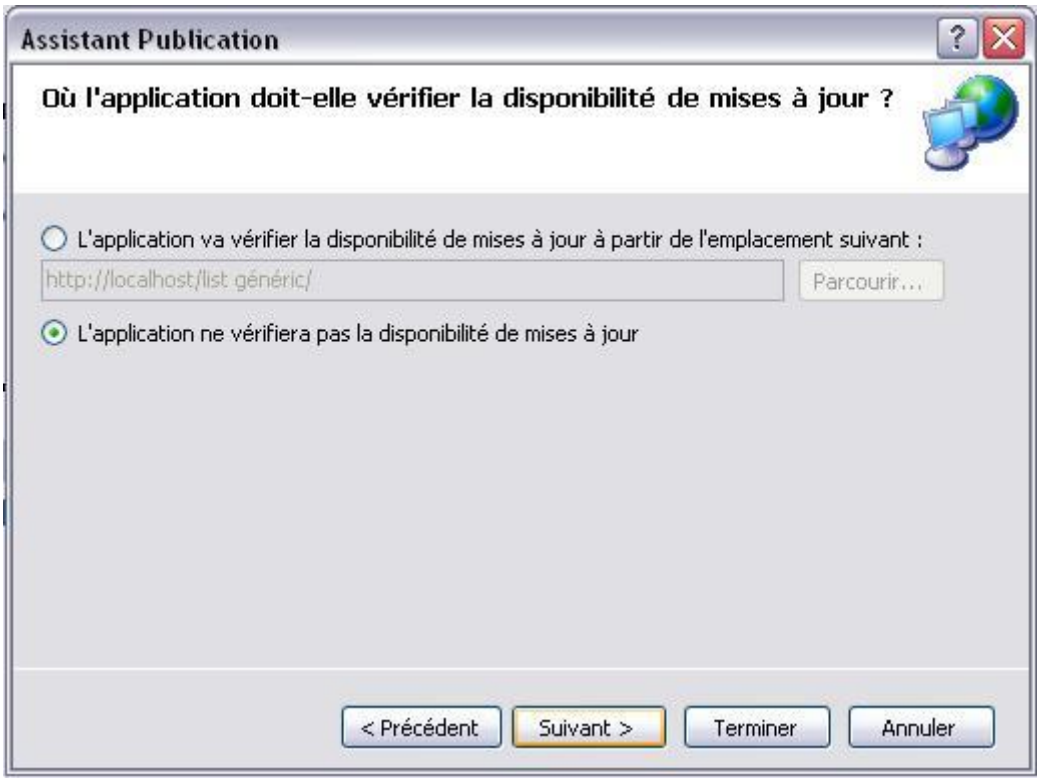
L'Assistant Publication comprend trois étapes.

- La première étape consiste à sélectionner l'emplacement où vous souhaitez placer le programme d'installation et tous les fichiers associés. Si vous publiez votre programme sur un CD-ROM, sélectionnez un dossier sur votre disque local. (Vous graverez ensuite ce dossier sur le CD-ROM.)


(publish\ crée un répertoire sous les sources (au même niveau que le répertoire bin.)

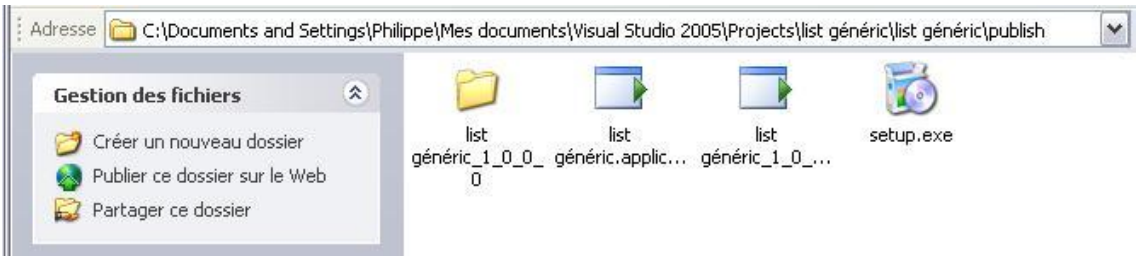
- La deuxième étape consiste à spécifier la manière dont les utilisateurs installeront votre programme ; dans le cas présent, à partir d'un CD-ROM.


- L'étape finale implique le fait de spécifier si votre programme vérifie ou non automatiquement à chaque démarrage la présence d'une version plus récente.




- Puis cliquez sur le bouton 'Terminer'.

Quand tout est terminé, cela ouvre une fenêtre montrant le contenu du répertoire \publish :




Il y a

Setup.exe ;

listgénéric.application ;

listgénéric_1_0_0_0.application ;

et un répertoire 'listgénéric_1_0_0_0' ;

listgénéric.exe.deploy ;

listgénéric.exe.manifest.

**Exemple d'installation à partir d'un Site WEB avec mise à jour automatique**

L''Assistant Publication' s'exécute.

Dans la page Où souhaitez-vous publier l'application ? entrez l'URL du site Web où vous souhaitez publier votre programme. Par exemple, http://www.mysite.com/myprogram.

Attention

Pour publier votre programme sur un serveur Web, ce dernier doit exécuter IIS (Internet Information Services), les extensions FrontPage doivent être installées, et vous devez disposer de privilèges d'administration dans IIS.

Dans la page suivante de l'Assistant : sur la page L'application sera-t-elle disponible hors connexion ? sélectionnez Oui, cette application est disponible en ligne ou hors connexion, la valeur par défaut.

L'application peut être accessible que de façon online (retéléchargée à chaque fois) ou de façon offline, téléchargée, installée et accessible via le menu Démarrer.

Cliquez sur Terminer pour publier le programme.

Le programme est publié sur le site Web spécifié, et une page HTML est créée.

Sur un autre ordinateur, ouvrez Internet Explorer, naviguez jusqu'à l'URL saisie auparavant, puis cliquez sur le lien Installer pour installer le programme.

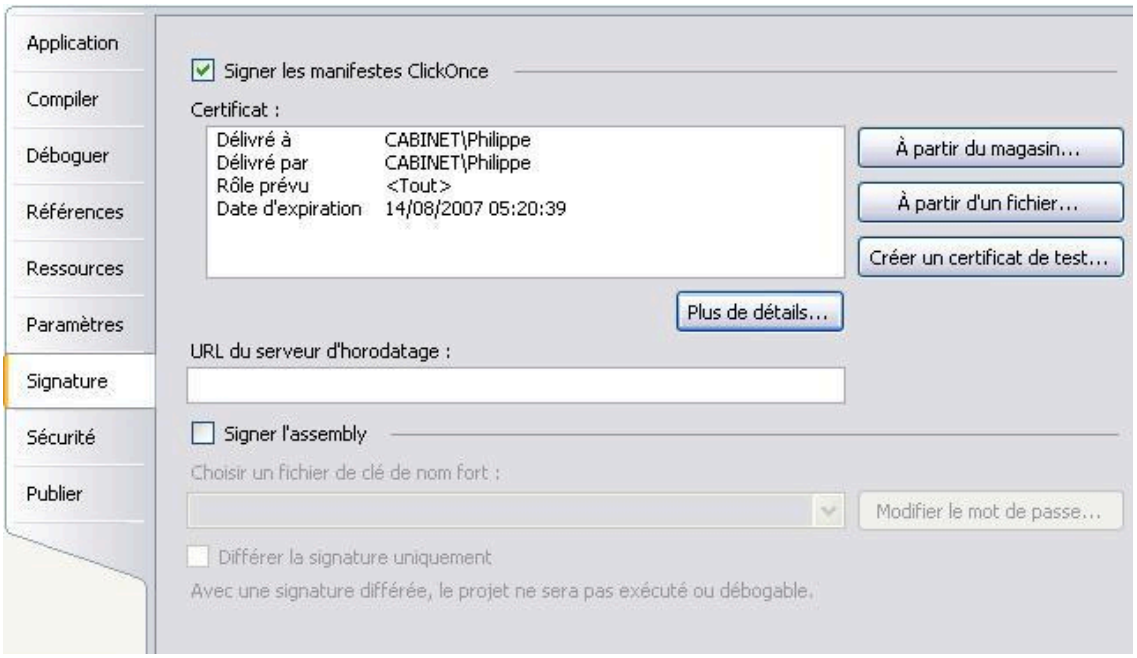
**Configuration avancée du projet de déploiement**

Nous allons modifier plein de choses avant de déployer.

Veuillez ouvrir le panneau des propriétés de votre projet (menu Projet > Propriétés de nomdeprogramme ou double-cliquer sur MyProjet dans la fenêtre d'explorateur de solution).

Signer le projet

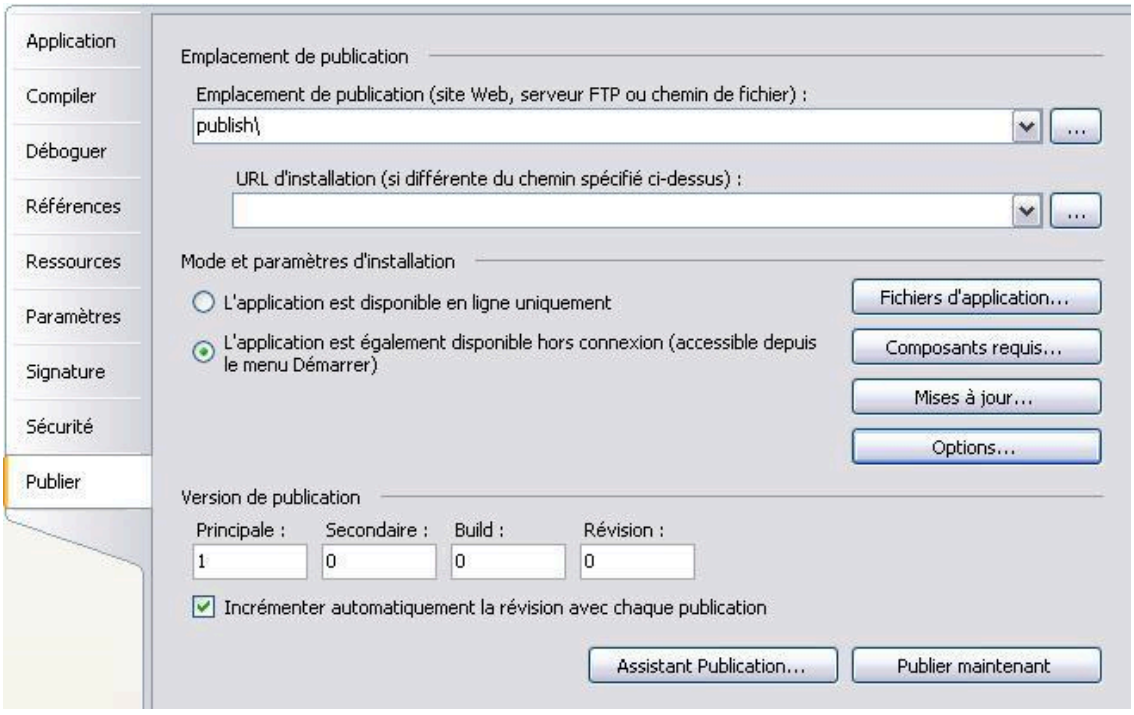
Cliquez sur l'onglet signature puis sur la case 'Signer les manifestes CliclOnce'.




Activer la Sécurité

Onglet sécurité, activer les paramètres de sécurité.


Onglet publier :

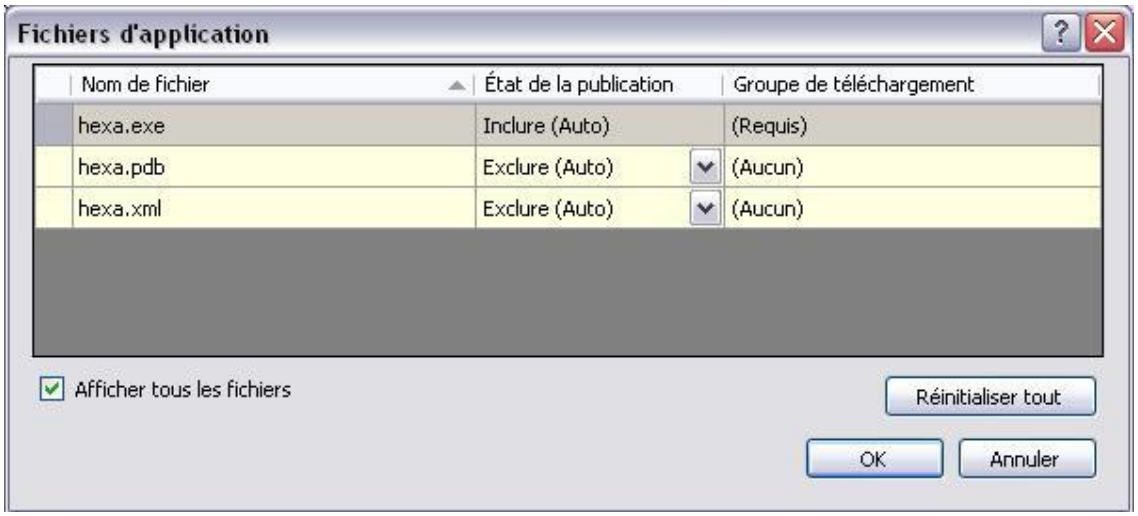



Pour indiquer l'emplacement de la publication, si l'application est disponible en ligne ou hors connexion (installée).

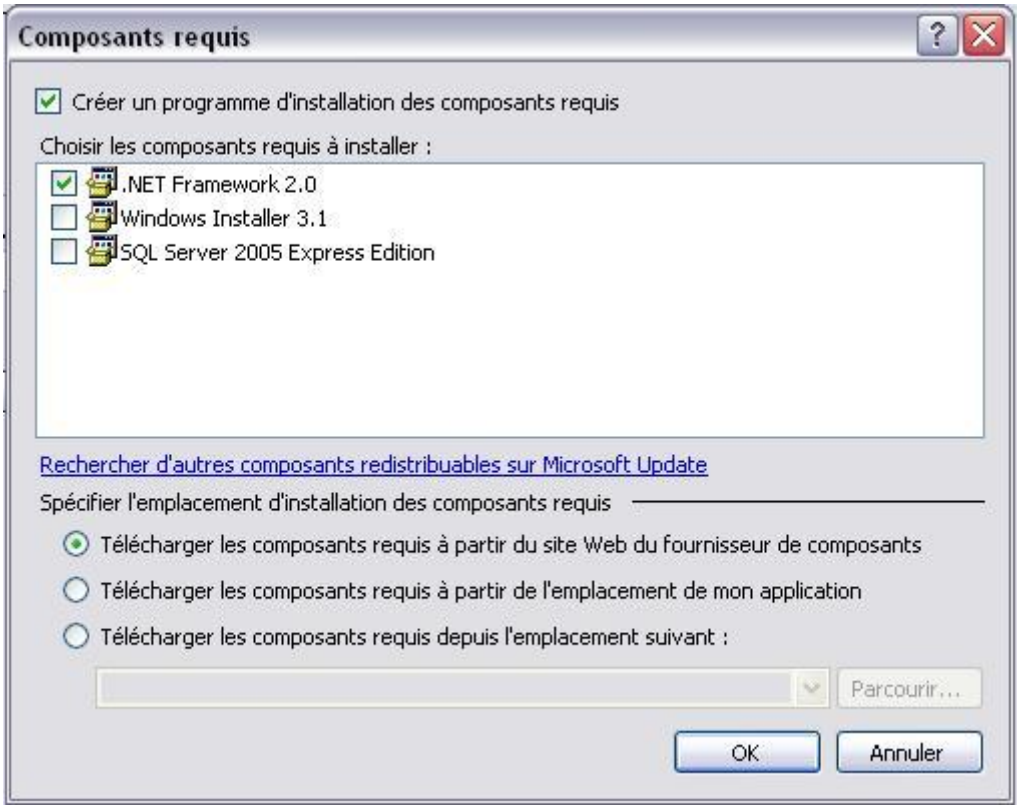
Si on coche la case incrémenter… cela incrémente automatiquement les versions.

4 boutons donnent accès :

au fichier d'application :




aux composants requis :

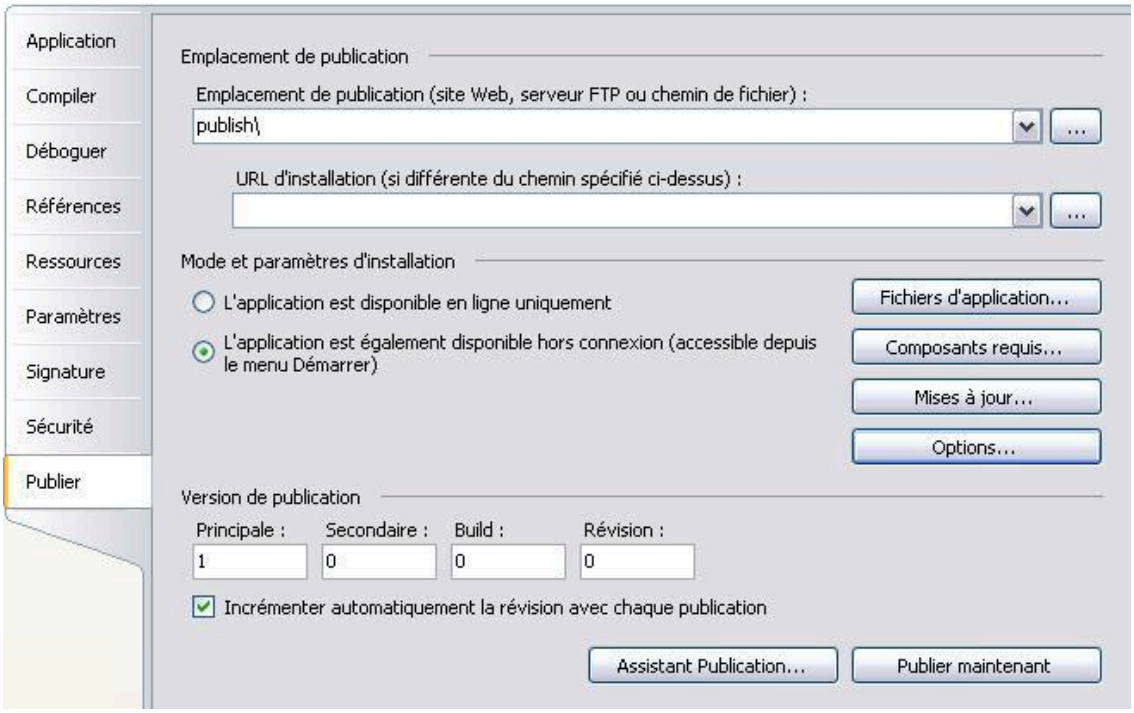



aux 'mises à jour' :


aux options de publication :


On peut ensuite utiliser les boutons qui sont en bas.

Bouton 'Assistant de publication' et 'Publier maintenant'.



### Ajouter enlever des fichiers

- Comment ajouter les fichiers publiés via ClickOnce ?

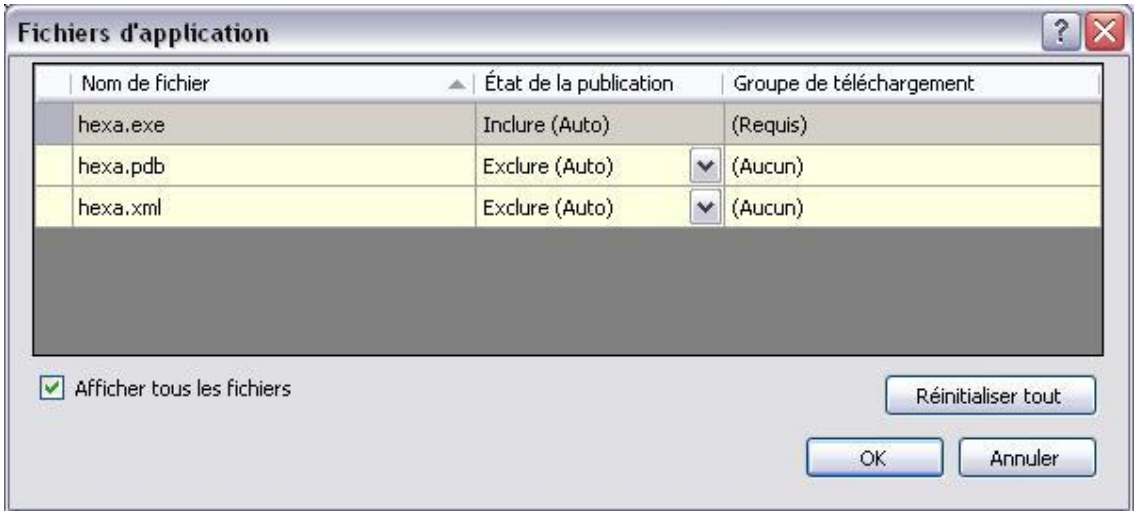
Tous les fichiers du projet qui ne contiennent pas de code sont déployés avec l'application.

Il suffit donc d'inclure dans le projet des fichiers de données.

### Comment inclure des fichiers ?

Il suffit de glisser les fichiers que nous voulons ajoutons dans le dossier "Bin" dans l'explorateur de solution; ensuite les fichiers de type image ou autre seront installés sur le poste du client et pour nous développeurs, il suffira d'indiquer leurs chemins de cette manière : "BinomDuFichier.extension".

Les fichiers d'une application ClickOnce sont ensuite gérés dans la boite de dialogue Fichiers d'application, accessible à partir de la page Publier du Concepteur de projets. Dans la fenêtre de l'explorateur de solutions à droite, double-cliquer sur 'MyProjet' puis sur l'onglet Publier enfin sur le bouton 'Fichiers d'application'.




### Exclure un fichier

Dans la boite de dialogue Fichiers d'application, sélectionnez le fichier que vous souhaitez exclure.

Dans le champ État de la publication, sélectionnez Exclure dans la liste déroulante.

### Fichier de données

Dans le champ État de la publication, sélectionnez Fichier de données dans la liste déroulante.

### Composant requis

Dans la boite de dialogue Fichiers d'application, sélectionnez l'assembly d'application (fichier .dll) que vous souhaitez marquer comme composant requis. Notez que votre application doit posséder une référence à l'assembly d'application pour figurer dans la liste.

Dans le champ État de la publication, sélectionnez Composant requis dans la liste déroulante.

### Comment cela se passe chez celui qui installe ?

Par exemple il faut installer le programme chiffreromain, il a un CD avec les fichiers :

Setup.exe ;

chiffreromain.application ;

chiffreromain_1_0_0_0.application ;

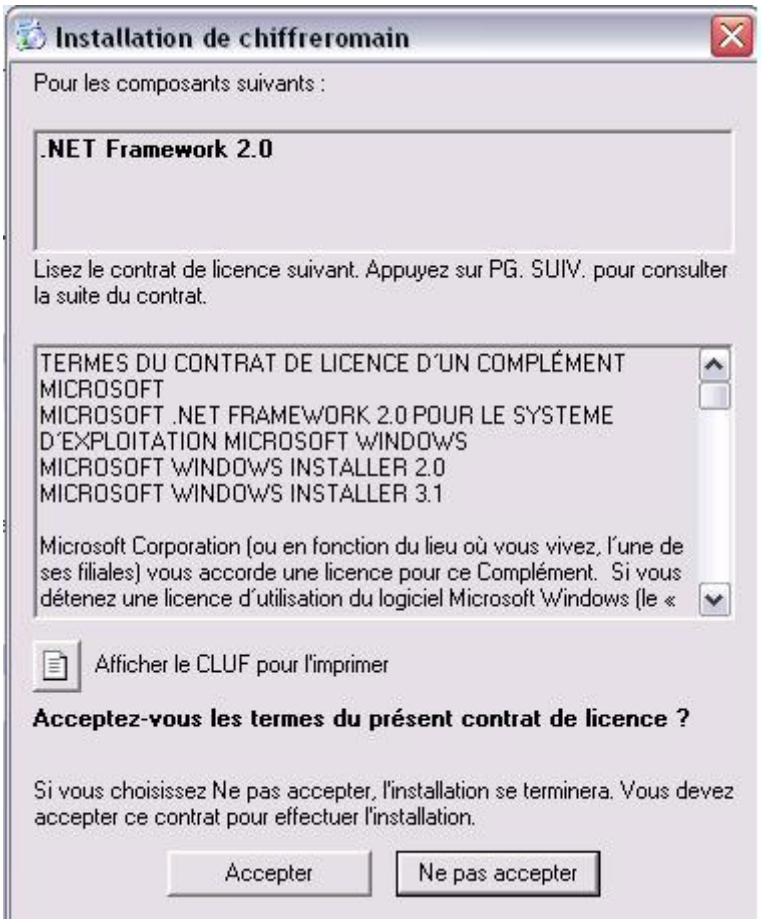
et un répertoire 'chiffreromain_1_0_0_0' ;

chiffreromain.exe.deploy ;

chiffreromain…exe.manifest.

Pour installer, on double-clique sur Setup.exe.

Si le Framework n'est pas installé et s'il y a une connexion Internet, la fenêtre suivante s'ouvre et permet de télécharger et d'installer le Framework.




Puis une fenêtre permet d'installer le programme.


Pour lancer le programme, pas de problème, l'installateur ajoute le nom du programme dans le menu 'Démarrer' et aussi dans le menu programme, menu 'nom de l'éditeur' puis chiffreromain.

Pour désinstaller, pas de problème, dans menu 'Démarrer', "Paramètres", "Ajouter et supprimer programmes", le programme est répertorié et un bouton permet de le désinstaller.

**Mais, où s'installe le programme ?**

C:\Documents and Settings\NomUtilisateur\LocalSettings\Apps\NomSociété\JHBVHR0G.E57\ZBOQP5EG.EYY\chif…tion_6625898959f0f00b_0001.0000_c9deafec99019f28

On remarque que l'exécutable n'est plus dans un répertoire de 'Programs Files', mais dans les documents , le Local Setting, sous le nom de la société (celui qui est dans l'assembly), sous le nom du programme, mais aussi sous le numéro de version…

## XIV-B-6 - Autres installateurs

Il existe des installateurs gratuits non Microsoft.

Exemple : **DreamShiel:https://dreamshield.developpez.com/**

DreamShield est un outil de publication puissant qui permet de déployer les applications utilisant le Microsoft .NET Framework 2.0+ sur les systèmes allant de Windows 2000 à Windows Seven.

## XIV-C - Exemples de petites applications par Microsoft

101 exemples de programme Vb 2003 : une mine :

```
http://www.microsoft.com/downloads/details.aspx?
familyid=87951cb9-5aeb-4f46-9bf0-2b3e3664be77&displaylang=en
```

101 exemples de programme Vb 2005 : une autre mine :

**http://msdn.microsoft.com/fr-fr/vbasic/ms789075.aspx**

101 exemples de programme Vb 2008 :

**http://code.msdn.microsoft.com/vbsamples**

# XV - Programmation Objet : création de classes et de composants

## XV-A - Programmation orientée objet, Propriétés des Classes (Rappel)

**VB.NET permet maintenant de faire de la POO (Programmation Orientée Objet) à part entière.**

**Il y a : les Classes du FrameWork.**

**On peut aussi CRÉER soi-même (dans des modules de Classe) de nouvelles Classes qui suivront elles aussi les règles de la POO. Ces classes serviront à instancier des objets.**

*Pour ce chapitre, nous sommes du côté de l'application utilisatrice des objets (et non dans les objets).*

*L'objet est une boite (jaune ici !!), je l'utilise, mais je ne sais pas ce qui se passe à l'intérieur.*

### XV-A-1 - Interface et Implémentation

Nous savons déjà : on utilise une Classe (le moule) pour instancier (créer) un objet.

**Une classe est une combinaison de code et de données.**

- Le code et la définition des données constituent **l'implémentation (c'est à l'intérieur de la boite)**.
- L'interface de l'objet est l'ensemble de ses membres visibles et utilisables (les membres sont les propriétés, les méthodes, les événements).

Exemple

**Prenons un objet d'une classe ListBox.**

- L'**interface** ListBox.Visible ListBox.AddItem…c'est je la vois , je peux l'utiliser.
- L'**implémentation**, je ne la vois pas, c'est le code qui gère la ListBox, la définition des éléments, c'est une 'boite noire', je ne sais pas ce qui s'y passe, je n'y est pas accès, et c'est tant mieux !!!

### XV-A-2 - Encapsulation

Le fait de ne pas voir l'implémentation (le code), c'est **l'encapsulation**.

Le code, les définitions de données sont privés à l'objet et non accessibles, ils sont enfermés, encapsulés dans une boite noire.

*L'encapsulation permet donc d'exposer aux applications clientes uniquement l'interface.*

Les applications clientes n'ont pas à se soucier du fonctionnement interne.

Cela a une conséquence, si je modifie le code, mais pas l'interface, l'application cliente n'a pas à être modifiée.

## XV-A-3 - Héritage

On a vu qu'un objet issu d'une Classe dérivée hérite des membres de la classe de base (la classe parent), cela crée une relation mère/fille (parent/enfant), la classe fille pouvant réutiliser les membres de la classe mère.

À noter qu'une classe ne peut hériter que d'une classe en VB.NET.

La Classe fille peut utiliser **les membres de la classe mère**, mais aussi **ajouter ses propres membres** ou **redéfinir certains membres de la classe mère**.

Exemple

On a vu que quand on dessine une Form1, cela crée une Classe 'Form1' qui hérite des Windows.Forms (Inherits System.Windows.Forms.Form).

Autre exemple : **ListBox** hérite de Control.

## XV-A-4 - Polymorphisme

Le nom de *polymorphisme* signifie "qui peut prendre plusieurs formes".

Tous les sites français donnent les mêmes définitions et entretiennent une certaine confusion. Ils indiquent 3 types de polymorphisme et la différence entre polymorphisme 'paramétrique' de surcharge et ad hoc n'est pas évidente. Je vais donc expliquer les choses à ma manière !!

Il y a 4 **sortes de polymorphisme.**

- **Le polymorphisme ad hoc.**
  Le polymorphisme ad hoc permet d'avoir des fonctions de même nom, avec des fonctionnalités similaires, dans des classes sans aucun rapport entre elles. Par exemple, la classe Integer, la classe Long et la classe Date peuvent avoir chacune une fonction **ToString**. Cela permettra de ne pas avoir à se soucier du type de l'objet que l'on a si on souhaite le transformer en String.
- **Le polymorphisme de *surcharge*** ou en anglais ***overloading***.
  Une méthode gère des paramètres de type et de nom différents
  Ce polymorphisme représente la possibilité de définir plusieurs méthodes de même nom, mais possédant des paramètres différents (en nombre et/ou en type).
  Pour ouvrir une fenêtre MessageBox, la méthode Show a 12 signatures, en voici 2.
  Ici on donne 4 paramètres.

```
Reponse= MessageBox.show(TexteAAfficher,Titre, TypeBouton etIcone, BoutonParDéfaut)
```

  Ici 1 seul paramètre.

```
Reponse= MessageBox.show(TexteAAfficher)
```

  On appelle signature chaque combinaison d'arguments d'une fonction (combinaison en nombre et en type). Une fonction a donc autant de signatures que de manière d'appeler cette fonction. C'est donc la signature d'une méthode qui détermine quel code sera appelé.
- **Le polymorphisme d'héritage** (redéfinition, spécialisation ou en anglais overriding)

Quand une classe hérite d'une autre classe, elle hérite des méthodes. On peut redéfinir substituer une méthode de la classe parent par une méthode de même nom dans la classe enfant.

- **Le polymorphisme générique** (en anglais template)
  C'est la forme la plus naturelle du polymorphisme : elle permet d'appeler la méthode d'un objet sans devoir connaitre son type. En VB 2005 cela correspond aux génerics.

## XV-A-5 - Constructeur, destructeur

Un **constructeur** est une fonction effectuée **lors de l'instanciation** d'un objet de la Classe, il sert généralement à 'initialiser' l'objet. Il est appelé quand on fait **New**.

Souvent il y a plusieurs signatures. Il y a habituellement un constructeur par défaut qui n'a pas de paramètres.

Exemple : pour créer un objet graphique Point, j'utilise un constructeur permettant de définir les coordonnées du point :

```
Dim P As New Point(45, 78)
```

**La destruction** d'un objet est effectuée lorsqu'on lui affecte la valeur Nothing ou lorsqu'on quitte la portée où il a été défini.

```
P= Nothing
```

## XV-A-6 - Accesseur, mutateur

Un accesseur (accessor en anglais) est un membre renvoyant la valeur d'une propriété d'un objet.

**MyObjet.GetName** est un accesseur, car elle renvoie la valeur de la propriété **Name**.

Un mutateur (mutator en anglais) ou encore modifieur (modifier en anglais) est un membre qui modifie la valeur d'une propriété d'un objet.

**MyObjet.SetName** est un mutateur, car elle modifie la valeur de la propriété Name.

## XV-A-7 - Déclaration, instanciation

On peut déclarer et instancier en même temps :

```
Dim P As New MaClasse
```

On peut séparer les 2 actions :

```
Dim P As Maclasse    'déclaration

P As New Maclasse    'instanciation
```

La déclaration et l'instanciation peuvent être effectuées dans les endroits différents :

```
Module Mon module

    Public P As MaClasse


Sub MaRoutine
```

```
    P As New MaClasse

End Sub

End Module
```

Ici P est déclaré comme Public, il est instancié dans une Sub.

**Initialisation simplifiée**

Soit une Classe Personne ayant les property Nom et Id

En VB 2005 on pouvait écrire :

```
Dim per2 As New Personne

With per2

  .Nom = "Philippe"

  .Id = 2

End With
```

Si le constructeur le permet (s'il accepte 2 arguments pour New) on peut aussi écrire :

```
Dim per1 As New Personne ("Philippe", 2)
```

En VB 2008 on peut écrire :

```
Dim per1 As New Personne With {.Nom = "Philippe", .Id = 2}
```

## XV-A-8 - Propriétés, Méthodes

Un Objet peut avoir une ou des propriétés :

```
Dim B As New Button

B.Name ="toto"
```

Un Objet peut avoir une ou des Méthodes :

```
Dim B As New Button
```

**B.Click** est une méthode.

## XV-A-9 - Les Classes : elles sont 'By Ref'

On rappelle que les classes sont des Objets 'By Ref' (Par référence).



*Il faut comprendre qu'une variable Objet contient la référence, le pointeur de l'objet, mais pas l'objet lui-même.*

Cela entraine…

## XV-A-9-a - Création de variables objet

Soit une classe Class1.

```
Dim I As Class1
```

On crée un pointeur vide, entraine : I contient Nothing : il ne pointe sur aucun objet.

```
    I = New Class1
```

Maintenant I contient la référence, le pointeur vers un objet de type Class1.

Le constructeur New a bien créé une instance de Class1.

Habituellement on utilise en une fois :

```
Dim I As New Class1
```

On peut voir si la variable contient Nothing : If IsNothing( I ) then… ou If I Is Nothing…

## XV-A-9-b - Affectation

Si on affecte **une variable par référence** à une autre, elle pointe toutes les 2 sur le même endroit mémoire : si j'en modifie une, cela modifie l'autre.

Créons une Classe contenant un entier :

```
Class Class1
   Public Value As Integer = 0
End Class

Dim C1 As New Class1()
Dim C2 As Class1 =C1    'on crée C2, on affecte C1 à C2
C2.Value = 123          'on modifie C2


=> C1.Value=123  C2.Value=123
```

Modifier C2 a modifié C1, car elles pointent sur le même endroit mémoire.

On le redit autrement : quand on crée C1 et C2, il n'y a pas 2 objets C1 et C2, mais 2 pointeurs vers le même objet.

Si on veut faire une copie 'indépendante', il faut utiliser **Clone** :

```
Class Class1
   Public Value As Integer = 0
End Class

Dim C1 As New Class1()
C1.Value= 555
Dim C2 As Class1 =C1.Clone    'on crée C2, on clone
C2.Value = 123          'on modifie C2

'C1.Value=555  C2.Value=123
```

## XV-A-9-c - Comparaison

Deux objets peuvent être comparés par "Is".

```
Dim O As Object

Dim Q As Object

If O Is Q then…
```

Equals peut être utilisé pour la comparaison :

```
Obj1.Equals(Obj2))  'Retourne True si Obj1 et Obj2 ont le même pointeur.
```

## XV-A-10 - Nommage

Pour les noms de Classe, utiliser le case Pascal : chaque mot qui compose le nom a sa première lettre en majuscule.

Exemple Class MaClasse

Idem pour les événements, espaces de noms, méthodes.

Exemple : System.Drawing, ValueChange…

Dans les objets, il ne faut pas inclure des noms de classe dans les noms de propriétés Patient.PatientNom est inutile, utiliser plutôt Patient.Nom.

# XV-B - Créer une Classe

## XV-B-1 - Création de Classe

On a vu qu'il existait des classes prédéfinies (celle du Framework par exemple), mais on peut soi-même CRÉER SES PROPRES CLASSES :

*Maintenant, on est DANS la boite.*

### XV-B-1-a - Revenons une nouvelle fois sur la notion de Classe et d'Objet

**On va créer une classe 'Médicament' (c'est l'objet de ce chapitre).**

Un 'Médicament' possède les variables :

```
Nom
Laboratoire
Nombre de médicaments.
…
```

Il faut donc pouvoir 'regrouper' ces variables pour un médicament précis. Pour regrouper ces variables, on utilise une structure particulière : la Classe.

Une Classe 'Medicament' aura :

```
Medicament.Nom
Medicament.Laboratoire
Medicament.NbdeMedicament
```

Avec cette Classe (la structure, le moule), on peut créer (instancier) un Objet MonMedicament.

```
Dim MonMedicament As New Medicament
```

Il comporte les propriétés (données) :

```
MonMedicament.Nom
MonMedicament.Laboratoire
MonMedicament.NbdeMedicament
```

Il comporte des méthodes (comportement) :

**MonMedicament.GetNom** est une méthode.

L'utilisateur de l'objet ne voit que le nom de la méthode (**l'interface**), il ne voit pas le code de la procédure (**l'implémentation**). Il y a bien **encapsulation.**

**Une fois cette classe créée, du côté utilisateur, comment l'utiliser ?**

Pour instancier un objet Medicament :

```
Dim MonMedicament As New Medicament()
```

Donner une valeur à une propriété :

```
MonMedicament.Nom= "Aspirine": MonMedicament.Laboratoire="RP"
```

Récupérer la valeur d'une propriété :

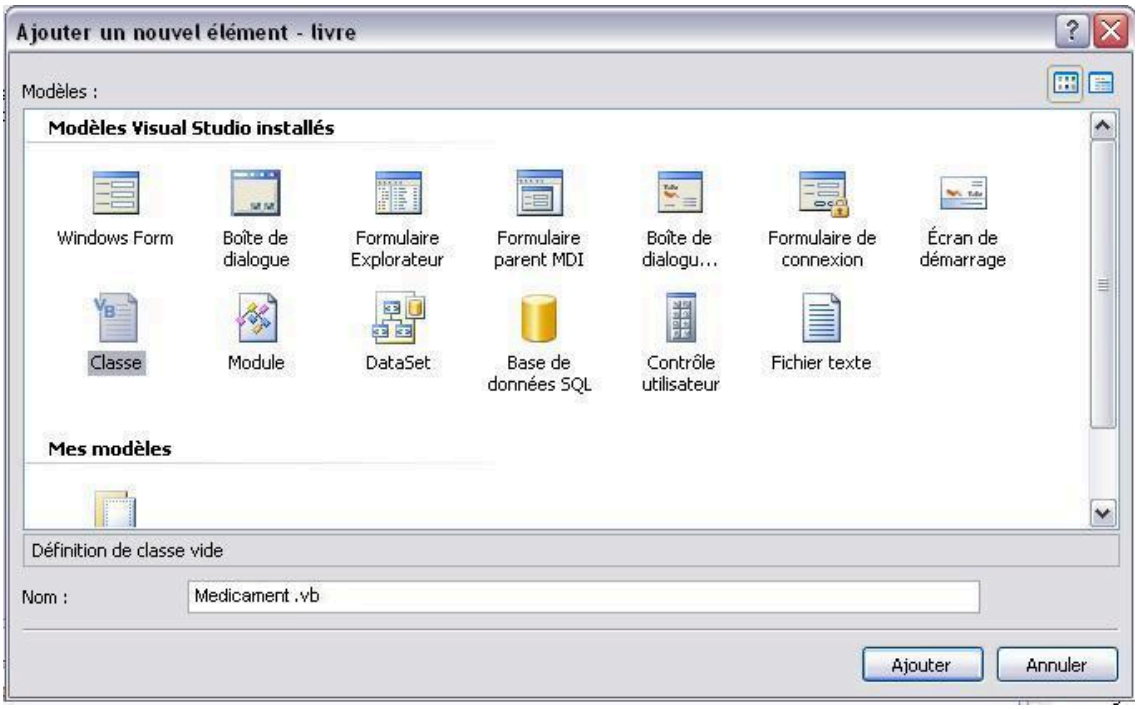
```
LeNom=MonMedicament.Nom  : Nb= MonMedicament.NombreMedicament
```



*Pour la suite de ce chapitre, nous sommes DANS la Classe de l'objet (et non dans l'application utilisatrice).*

## XV-B-1-b - Créer une Classe

Menu **Projet** puis **Ajouter une Classe.**




Entrer 'Medicament' dans la zone nom puis OK

Une nouvelle fenêtre de module est ajoutée à notre projet, contenant le code suivant :

```
Public Class Medicament
…
End Class
```

Le mot avant Class indique la portée de la classe :

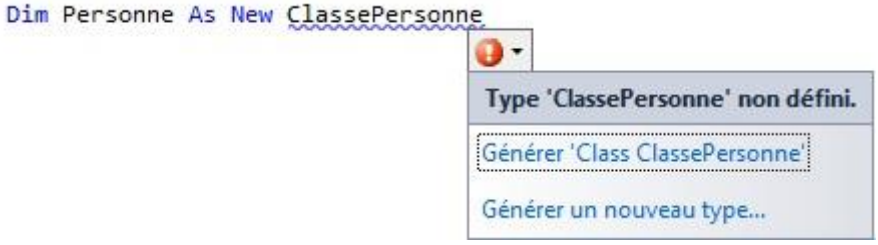
- **Public** (Classe instanciable dans et hors du projet, utilisable par un autre programme) ;
- **Private** (Classe instanciable que dans elle même !!) ;
- **Friend** (Classe instanciable dans le projet) ;
- **Protected** (Classe instanciable uniquement dans les classes dérivées).

On peut ajouter :

**MustInherit :** cela donne une classe non instanciable, on ne peut pas créer d'objet avec !! Alors à quoi cela sert !! À fournir une base pour des classes qui en hériteront. On appelle ces classes des **classes abstraites.**

**NotInheritable :** cette classe ne peut-être héritée.

En vb 2010, si dans le code, on utilise une classe qui n'existe pas, vb souligne le nom de la classe et vous propose (en cliquant sur le bouton dessous) de créer une classe vide :



# XV-B-1-c - Ajouter des variables dans une classe

On parle de variable ou 'Attribut' permettant d'associer **des données** à une Classe.

La variable peut être '**Privée**' et non visible à l'extérieur :

```
Private _mNom As String
```

Il est conseillé de mettre un '_' au début du nom d'une variable privée puis une majuscule au début de chaque mot sauf le premier.

Elle peut être '**Public**', visible à l'extérieur et donc non encapsulée.

```
Public Laboratoire As String
```

Il est conseillé de mettre une majuscule au début de chaque mot formant le nom de la variable public.

À l'extérieur, si on instance M comme un médicament (**Dim M As New Médicament**), **M.Laboratoire** est valide.

On peut définir un champ 'Public' en lecture seule qui a une valeur constante :

```
Public ReadOnly NombreMedicament=2000
```

Vous pouvez ajouter à votre variable : **Shared.** Cela signifie que la variable déclarée comme Shared est **partagée par toutes les instances de la classe : c'**est une variable de classe et non une variable d'instance.
Une variable Shared est même utilisable sur le nom de la classe, sans instancier.

Exemple :

```
Public Class Medicament
    Public Laboratoire As String    'variable d'instance
    Public Shared Societe as String 'variable de classe Shared
End Class

Dim M1 As New Medicament
Dim M2 As New Medicament

M1.Laboratoire= "MVV"
M2.Laboratoire= "VVT"
```

Chaque instance à sa propre variable Laboratoire (Non Shared).

Par contre :

```
M1.Societe="ooo"
```

entraine que **M2.societe** est aussi égal à "ooo".

La propriété **Societe** de toutes les instances de Medicament a la même valeur.

**Medicament.Societe** est aussi égal à "ooo" Ici on a directement utilisé le nom de la classe.

(Ne pas confondre avec une variable Static qui est une variable qui conserve sa valeur, mais qui a une valeur pour chaque instance).

Une variable, comme tous les membres d'une classe peut être **Private, Public,** mais aussi **Protected** (accessible par les seules méthodes de la classe ou des objets dérivés).



*Il faut en général déclarer les variables internes en 'Private': elles sont internes et non accessibles à l'extérieur de la Classe.*

## XV-B-1-d - Ajouter des propriétés grâce à 'Property'

Les propriétés permettent d'associer **des données** à une Classe.

On rappelle qu'une Property peut être utilisée comme cela :

```
Dim MonMedicament As New Medicament()
MonMedicament.Nom= "Amoxicilline" : Nom est une Property.
```

Vu de l'extérieur cela pourrait ressembler à une variable, en fait dans l'implémentation c'est complètement différent. Il y a du code permettant d'écrire ou de lire la property.

Pour créer une Property

Tapez 'Property Nom' puis validez, la définition de la propriété est générée en faisant apparaitre

- un bloc **Get** qui correspond à la lecture de la propriété par l'utilisateur ;
- un bloc **Set** qui correspond à l'écriture de la propriété par l'utilisateur, on récupère la valeur dans le paramètre value.

```
Property Nom()
Get
End Get
Set (By Val Value)
End Set
End Property
```

J'ajoute **Public** pour que cette propriété soit accessible hors de la classe, j'indique que c'est une **String.** Lors de la lecture de la propriété (par l'utilisateur de l'instance) **Get** retourne (grâce à Return) la valeur _mNom qui est une variable privée de la classe et qui sert à stocker la valeur de la propriété. Lors de l'écriture de la variable, Set récupère la valeur dans Value et la met dans _mNom :

```
Public Class Medicament

'Variable privée (Attribut) servant à stocker en interne le nom.
Private _mNom As String

'Propriété Nom
Public Property Nom() as String
Get
    Return _mNom
End Get
Set(By Val Value)
    _mNom=value
End Set
End Property

End Class
```

### Encapsulation

La variable '**-mNom' est encapsulée :** l'utilisateur qui utilise une instance de la classe, ne voit pas ce qui s'y passe (ici c'est très simple) quand il utilise le nom, **il ne voit pas l'implémentation** (l'implémentation c'est Get…Set…), **il ne voit que l'interface** c'est-à-dire .Nom ; Il n'a pas accès à _mNom.

Si l'utilisateur tape : **MonMedicament.Nom="Aspirine"** , c'est le code **Set…EndSet** qui est exécuté.

Si l'utilisateur tape : **Dim s As String= MonMedicament.Nom** , c'est le code **Get…EndGet** qui est exécuté.

Une propriété peut être en **lecture seule :**

```
Public ReadOnly Property InitialeNom() As String
Get
 Return Left(_mNom,1)
End Get
End Property
```

Mais aussi en **écriture seule** grâce à WriteOnly.

**Les propriétés comme les méthodes peuvent être Public, Private, Protected, Friend, ProtectedFrient.**

**À partir de VB 2005, on peut créer des Property avec une portée différente pour le Set et le Get :**

```
Public Class employee
    Private salaryValue As Double
    Protected Property salary() As Double
        Get
            Return salaryValue
        End Get
        Private Set(ByVal value As Double)
            salaryValue = value
        End Set
    End Property
End Class
```

Implémentation automatique. À partir de vb 2010 une propriété peut être écrite (implémentée) de manière simple et rapide : plus besoin de Get et Set :

```
'Code complet
Private _MaProperty As String = "Empty"
Property MaProperty As String
    Get
        Return _MaProperty
    End Get
    Set(ByVal value As String)
        _MaProperty = value
    End Set
End Property

'Peut être remplacé par:
Property MaProperty As String = "Empty"
```

La dernière ligne génère '_MaProperty' qui est un champ privé.

On peut même indiquer une valeur par défaut ou un type :

```
Property ID() As Integer = -2
Property ManList() As New List(Of Man)
```

Petit exemple : créons une classe 'Personne' avec 2 propriétés : 'Nom' et 'Prenom'.
On instancie un objet 'FirstPersonne' de type 'Personne' et on donne une valeur aux 2 propriétés de l'objet.

```
Public Class Personne
    Property Nom As String
    Property Prenom As String
End Class
…


Dim FirstPersonne As New Personne
FirstPersonne.Nom= "Las"
FirstPersonne.Prenom= "Philippe"

'En VB 2010 on peut aussi écrire:
Dim FirstPersonne As New Personne With {.Nom = "Las", .Prenom = "Philippe"}
```

## XV-B-1-e - Méthode

Une Classe peut contenir un **comportement** sous forme de procédures et fonctions contenant du code.

On peut ajouter une méthode à une Classe.

**Les méthodes d'une classe sont les procédures Sub ou Function** déclarées dans la classe.
Elles sont habituellement 'Public'.

Une méthode peut contenir du code effectuant un traitement :

```
Public Sub Dessine() As Double
' Code
End Sub
```

Une méthode peut aussi être utilisée pour lire ou écrire des variables privées, on parle dans ce cas d'**accesseurs :**

```
Public Class Medicament
'Variables ou Attributs
Private _nom As String

'Accesseurs
Public Sub GetNom() As String
    Return _nom
End Sub

'modificateurs
Public Sub SetNom (ByVal N As String)
    Me._nom= N
EndSub
End Class
```

SetNom est un mutateur.

## XV-B-1-f - Récapitulatif Property, méthodes

a Un objet peut être vu comme un **regroupement de données** (variables et propriétés).
La différence entre une variable et une Propriété (Property) est qu'alors que la variable est une simple variable, la Property est contrôlée par du code interne qui peut modifier le comportement de la lecture ou de l'écriture de la Property.

b Mais un objet va plus loin : il peut contenir un **comportement** sous forme de procédures et fonctions contenant du code. (Ce sont les **méthodes** de l'objet.). Elles représentent le comportement commun de tous les objets appartenant à la classe.

Tous les éléments de la Classe peuvent être **Public** (visibles par l'utilisateur de la classe) ou **Private** (non accessibles à l'utilisateur de la classe et utilisés en interne dans la classe).

*On voit aussi qu'il faut que les variables soient 'Private' et les méthodes et Property 'Public'.*

## XV-B-1-g - Constructeur

Un constructeur est une procédure exécutée lors de l'instanciation.

Dans le module de classe, elle est définie par :

```
Sub New()
End sub
```

On peut ajouter des paramètres qui serviront à instancier.

Par exemple pour instancier et donner le nom en même temps :

```
Sub New(ByVal LeNom As String)
    _mNom=LeNom
End sub
```

Cela permet **Dim M As New Medicament("Aspirine")**

On peut définir plusieurs constructeurs avec un nombre de paramètres différents (plusieurs signatures), dans ce cas il y a **surcharge,** le constructeur par défaut étant celui sans paramètre.

Une autre manière de faire est d'utiliser une méthode **Initialise.**

Créons une classe avec une méthode Initialise :

```
Classe Homme
Private _Nom As String
Private _Prenom As String
Public Sub initialise(ByVal N As String, ByVal P As String)
    Me._nom = N
    Me._prenom = P
End Sub
End Class
```

*_nom, _prenom* étant des données privées de la classe Homme, les instructions :

```
dim p as New Homme()
p._prenom="Philippe" est refusée
On écrira donc :
dim p as New Homme
p.initialise("Monnom", "Philippe")
```

## XV-B-1-h - Destructeur

**Un objet est détruit :**

- en lui affectant la valeur **Nothing** ;
- si on sort de sa portée.

Une procédure **Finalize** (appartenant à la Classe Objet) est automatiquement appelée quand l'objet est détruit.

On peut ajouter une procédure **Finalize** à notre classe, qui redéfinit la procédure Finalise de la Classe 'Objet'.

```
Protected Overrides Sub Finalize ()
End Sub
```

On peut y mettre le code libérant les ressources ou d'autres choses.

Noter que la procédure Finalize est ici la redéfinition (d'où 'Overrides') de la procédure Finalize (qui est Overridable) de la Classe Objet.

Attention la méthode Finalize est exécutée quand l'objet est réellement détruit (Objet=Nothing le rend inutilisable, mais il est toujours présent en mémoire). C'est parfois très tardivement que l'objet est détruit : quand il y a besoin de mémoire (c'est le Garbage Collector qui entre en action) ou à la sortie du programme.

**Pour forcer la destruction,** on peut utiliser l'interface **Idisposable.**

Il faut mettre dans l'entête de la classe :

```
Implements IDisposable
```

et mettre dans le code de la classe :

```
Public Sub Dispose() Implements System.IDisposable.Dispose
' Code libérant les ressources (Base de données, connexion, Objets systèmes…)….
End sub
```

C'est une méthode Public, on peut l'appeler de l'application cliente :

```
M.Dispose()
M=Nothing
```

Là aussi attention, **Dispose doit être appelé de manière explicite** (il faut le faire, ce n'est pas automatique), quand on fait M.dispose, la procédure Dispose et le code qu'il contient sont exécutés immédiatement, par contre c'est toujours le Garbage Collector qui décide quand Finalise sera exécuté.

Dans la pratique **quelle est d'utilité de gérer la destruction** autrement que par Objet=Nothing si le Garbage Collector nettoie la mémoire ? C'est une question.

Réponse donnée par Microsoft :

*Votre composant a besoin d'une méthode **Dispose** s'il alloue des objets système, des connexions à la base de données et d'autres ressources rares qui doivent être libérées dès qu'un utilisateur a fini de se servir d'un composant.*

Vous devez également implémenter une méthode **Dispose** si votre composant contient des références à d'autres objets possédant des méthodes **Dispose.**

## XV-B-1-i - Surcharge

On peut **surcharger un constructeur.**
Pour cela, il suffit de rajouter autant de procédures New que l'on veut avec **pour chacune un nombre de paramètres différents** (signatures différentes).

Exemple : o**n peut surcharger un constructeur :**

```
Class Figure
Sub New()
    Bla Bla
End Sub
```

```
Sub New( ByVal X As Integer, ByVal Y As Integer)
     Blabla
End Sub.
End Class
 On peut donc instancier la classe correspondante de 2 manières:
Dim A As New Figure    'Constructeur par défaut
ou
Dim A As New Figure(100,150)
```

On peut **surcharger une property.**
Pour cela il suffit de rajouter des procédures Property **ayant le même nom de méthode** avec pour chacune un nombre de paramètres différent (signatures différentes).

On peut ajouter **Overloads,** mais c'est facultatif.

Exemple surchargeons un membre :

```
Class Figure
Public Overloads Property Calcul()
    Bla Bla
End Sub

Public Overloads Property Calcul( ByVal X As Integer, ByVal Y As Integer)
     Blabla
End Sub.
End Class
```

## XV-B-1-j - Événement

On peut définir un événement pour la classe.

**Dans la classe :**

- il faut **déclarer l'événement,** avec le mot **Event ;**
- il faut utiliser l'instruction **RaiseEvent** pour le déclencher lorsqu'un état ou une condition le nécessite.

Exemple

je crée une classe nommée 'Class1' qui contient un membre 'Texte'( Property Texte), si 'Texte' change, alors on déclenche l'événement TextChange :

```
Public Class Class1
Private _mTexte As String

' Déclare un événement
Public Event TextChange(ByVal UserName As String)

Public Property Texte()
Get
    Return _mTexte
End Get
Set(ByVal Value)
If Value <> _mTexte Then
  RaiseEvent TextChange("hello")'<= déclenchement de l'événement par RaiseEvent
End If
_mTexte = Value
End Set
End Property

End Class
```

Si l'application cliente modifie la propriété .Texte d'un objet Class1 alors on compare l'ancien et le nouveau texte, s'il est différent on déclenche un événement TextChange.

**Dans l'application cliente**

- Il faut définir **dans la partie déclaration** l'objet M de classe Class1 en indiquant qu'il gère des événements :

```
    Private WithEvents  M As Class1
```

**Dans l'application cliente**

- Dans Form_Load par exemple il faut instancier l'objet :

```
    M= New Class1()
```

**Dans l'application cliente**

- Il faut écrire la procédure événement avec son code :

```
Private Sub M_TexteChange(ByVal v As String) Handles M.TextChange
    MsgBox("le texte a changé")
End Sub
```

**Dans l'application cliente**

Ici on demande simplement quand le texte change, d'ouvrir une MessageBox.

Lors de l'utilisation :

**M.Text="Nouveau text"** 'déclenche la Sub M.TextChange qui dans notre exemple simple ouvre une MessageBox indiquant que le texte à changer.

On remarque que la Classe définit l'événement, la procédure correspondant à l'événement est dans l'application cliente. (De la même manière que quand on clique sur un objet bouton cela déclenche la procédure Bouton-Click.)

## XV-B-1-k - Exception

Il est parfois utile de déclencher une exception : si l'utilisateur de la classe donne par exemple une valeur invalide à une property, une exception se déclenche indiquant que la donnée est invalide.

Dans l'exemple suivant si l'utilisateur affecte à la propriété Mavaleur une valeur négative, une exception est déclenchée **(Maclasse.Mavaleur=-2) :**

```
Public Property MaValeur() As Integer
    Get
    …
    End Get
    Set (ByVal Value As Integer)
        If Value>=0 Then
            _mValeur= Value
        Else
            Throw New Exception ("La valeur " & Value.ToString & " est invalide")
        End If
    End Set
End Property
```

## XV-B-1-l - Les Classes partielles

2005

Les classes 'Partielles' sont possibles en VB2005 (Framework 2).

Les types partiels permettent la définition d'une classe unique dans plusieurs fichiers source. La définition a une déclaration normale dans un fichier :

```
Public Class MaClasse
    'implémentation….
End Class
```

Dans d'autres fichiers source, utilisez le mot-clé **Partial** pour indiquer au compilateur que ce code est la suite de la définition de classe d'origine :

```
Partial Public Class MaClasse
    'implémentation….
End Class
```

## XV-B-1-m - Méthodes partielles

À partir de VB 2008. Elles sont présentes dans des Classes Partielles. Et sur des méthodes privées. Voici la syntaxe :

```
Partial Private Sub MyMethode()
End Sub
```

Exemple

Voici la Classe :

```
Partial Class Product
Private _Quantity As Integer
Property Quantity() As Integer
….
End Property

Partial Private Sub QuantityChanged()
End Sub

End Class
```

Ici dans la Classe initiale la méthode partielle QuantityChanged() sert à donner la signature.

L'autre Classe partielle, qui est dessous, ajoute des fonctionnalités à la méthode partielle :

```
Partial Class Product</b>

Private Sub QuantityChanged()

     MsgBox("Quantity was changed to " & Me.Quantity)

End Sub

End Class
```

# XV-B-2 - Classe suite et astuces


*Pour la suite de ce chapitre, nous sommes toujours DANS la Classe de l'objet (et non dans l'application utilisatrice).*

*DANS la boite.*

## XV-B-2-a - Me, My, MyClass, MyBase

Dans une classe, on peut utiliser Me, My, MyClass, MyBase pour appeler une méthode.

Résumons

**Me** c'est l'instance en cours.

**Me.Calcul** dans une classe, appelle la Sub Calcul qui est dans la Classe.

**MyClasse** c'est l'instance en cours. Mais à la différence de Me, c'est la méthode de la classe de base (Parente) qui est appelée si cette méthode a été substituée.

**MyBase** c'est la classe parente.

**MyBase** est couramment utilisé pour accéder aux membres de la classe de base qui sont substitués ou occultés dans une classe dérivée. **MyBase.New** permet d'appeler explicitement un constructeur de classe de base (parente) d'un constructeur de classe dérivé.

Voyons un exemple.

On a une classe de base, une classe dérivée et on voit, quand on utilise une méthode de la classe dérivée, ce que retourne Me, MyClass et MyBase :

```
Class baseClass
    'Classe de base (parente)
    Public Overridable Sub testMethod()
        MsgBox("Base class string (parente)")
    End Sub
    Public Sub useMe()
        ' Utilise la classe en cours même si
        ' la méthode a été overridé.
        Me.testMethod()
    End Sub
    Public Sub useMyClass()
        ' Utilise la classe en cours si elle a été overidée
```

```
        ' Utilise la classe parent si  pas d' override.
        MyClass.testMethod()
    End Sub

    Public Sub useMyBase()
        ' Utilise la méthode de la classe parente.
        MyBase.testMethod()
    End Sub
End Class

Class derivedClass : Inherits baseClass
    'Classe dérivée debaseClass
    Public Overrides Sub testMethod()
        MsgBox("Derived class string (enfant)")
    End Sub
End Class

Class testClasses
    'Classe pour tester
    Sub startHere()
        Dim testObj As derivedClass = New derivedClass()
        ' Usage de Me
        ' Ce qui suit affiche  "Derived class string (enfant)" toujours.
        testObj.useMe()

        ' Usage de MyClasse
        ' Ce qui suit affiche  "Base class string(parente)", car testMethod overidé.
        testObj.useMyClass()

        ' Usage de MyBase
        ' Ce qui suit affiche  "Base class string(parente)" toujours .
        testObj.useMyBase()
    End Sub
End Class
```

**My** c'est complètement différent, c'est un moyen rapide d'avoir accès à des classes de l'application, de l'ordinateur, des ressources…

Utiliser une icône nommée MyIcon qui est dans les ressources et en faire l'icône du formulaire en cours.

```
Me.Icon = My.Resources.MyIcon
```

Jouer un son qui est dans les ressources.

```
My.Computer.Audio.Play(My.Resources.Form1Greeting, AudioPlayMode.Background)
```

## XV-B-2-b - Propriété par défaut

**Default** sert à indiquer que la propriété est la propriété par défaut.

Créons une property par défaut :

```
Class MyClasse
Default Property MonItem As String
…
End Property
End Class
```

Maintenant, il n'est plus nécessaire d'utiliser le nom de la propriété quand on y accède.

```
Dim MyCollection As New MyClasse
```

On peut écrire :

```
MyCollection.MonItem (index)
```

Ou

```
MyCollection(index)    'MonItem est omis
```

Bien sûr, il ne peut y avoir qu'une seule property par défaut.

## XV-B-2-c - Méthode de Classe avec Shared

Vous pouvez ajouter à votre membre : **Shared.** Cela signifie que la variable, la propriété ou la méthode déclarée comme Shared est **partagée par toutes les instances de la classe : c'est une méthode de classe.**

Une méthode déclarée dans une classe **sans** modificateur **Shared** est appelée *méthode d'instance*. Une méthode d'instance opère sur une instance donnée.

Pour une variable non Shared, dans chaque instance la variable a sa valeur propre.

Une variable Shared est commune à toutes les instances de la Class et même à la Classe elle-même sans avoir besoin de l'instancier. C'est une variable de Classe.

L'exemple suivant illustre les règles d'accès aux membres d'instance et **Shared :**

```
Class Test
   Private _x As Integer
   Private Shared _y As Integer

   Sub F()
      _x = 1 ' OK, c'est équivalent à Me._x = 1.
      _y = 1 ' OK, c'est équivalent à Test._y = 1.
   End Sub

   Shared Sub G()
      _x = 1 ' Erreur, on ne peut accéder à Me._x.
      _y = 1 ' OK, c'est équivalant à Test._y = 1.
   End Sub
End Class

   Shared Sub Main()
      Dim t As New Test()
      t._x = 1 ' OK.
      t._y = 1 ' OK.
      Test._x = 1 ' Erreur, on accède à x seulement par une instance.
      Test._y = 1 ' OK  on peut accéder à y avec le nom de la classe, car y est shared.
   End Sub
```

La méthode F montre que, dans une fonction d'instance membre, un identificateur peut être utilisé pour accéder à des membres d'instance et à des membres **Shared.** La méthode G montre que, dans une fonction membre **Shared,** il est erroné d'accéder à une instance membre via un identificateur. En d'autres termes dans une méthode Shared on n'a accès qu'aux variables Shared. Quant à la méthode Main, elle montre que, dans une expression d'accès au membre, l'accès aux membres d'instance s'effectue par l'intermédiaire des instances, alors que l'accès aux membres **Shared** est possible via des types ou des instances.

## XV-B-2-d - Création d'un compteur d'instances

Je veux savoir combien il a été créé d'instances de 'Médicament' :

```
Class MyClasse
```

Créons une variable commune à toutes les instances :

```
Private Shared Nb as Integer=0
```

Le constructeur va l'incrémenter à chaque instanciation :

```
Sub New()
    Nb+=1
End sub
```

Il suffit de lire sa valeur pour savoir le nombre d'objets Medicament :

```
Public ReadOnly Property NbInstance()
    Get
        NbInstance=Nb
    End Get
End Property

End Class
```

## XV-B-2-e - Création d'un identifiant unique d'instance

Pour chaque instance, je veux avoir un identifiant unique, un **globally unique identifier** (GUID).

```
Private m_ID As System.Guid
Sub New()
    m_ID = System.Guid.NewGuid
End Sub
```

Exemple d'identifiant unique :

```
'0f8fad5b-d9cb-469f-a165-70867728950e
```

Chaque instance ayant un identifiant unique, je peux ainsi 'repérer' les instances (en créant une property ReadOnly afin de lire cet identifiant).

Il existe une très faible probabilité pour que le nouveau GUID soit égal à zéro ou égal à un autre GUID.

## XV-B-2-f - Singleton

Parfois, on a une classe avec laquelle on veut **créer une instance et une seule.**

```
Private Sub New()
```

**New** doit-être toujours private pour empêcher d'instancier avec **New.**

Un singleton est construit grâce à une méthode de classe nommée **'GetInstance'** (sans avoir à instancier).

```
Shared Function getInstance() As sing
```

Cette fonction qui s'appelle toujours getInstance va servir à instancier une fois la variable Instance.

```
Shared instance As sing
```

Cette variable est la base du Singleton. Elle s'appelle **Instance** (par convention) elle est du même type que la class et contient l'instance unique.

```
Public Class sing
'Déclaration de l'instance Singleton
Shared instance As sing

Private Sub New()'Pas oublier de mettre Private
    MyBase.New()
End Sub
Shared Function getInstance() As sing 'Voici la méthode de classe

If IsNothing(instance) Then 'Si le singleton n'est pas créé, alors faut le créer
    instance = New sing
End If
Return instance
End Function
….
…
End Class
```

Comment utiliser cette Classe ?

```
Dim t As sing =  sing.getInstance
```

**Remarque**

- Si on fait ensuite Dim t1 As sing = sing.getInstance c'est la même instance qui est retournée. On ne peut en créer qu'une…
- Si on écrit Dim t As New sing : cela plante.

On peut ajouter une protection contre les multithread trop rapides avec SyncLock GetType(sing)

```
    Shared Function getInstance() As sing

        If IsNothing(instance) Then
            SyncLock GetType(sing)
                 If IsNothing(instance) Then
                    instance = New sing
                end if
            End SyncLock
        End If
        Return instance
    End Function
```

## XV-B-2-g - Surcharge d'opérateur

À partir de VB 2005, on peut surcharger les opérateurs + - * / mais aussi _ ^ & Like And Or Xor Not < > = << >> CType IsTrue, IsFalse.

Cela signifie que pour cette classe l'opérateur aura le comportement que vous lui donnerez.

Exemple : surchargeons l'opérateur +

```
Public Classe height
    …
    Public Shared Operator +(ByVal h1 As height, ByVal h2 As height)As height
        Return New height(h1.feet + h2.feet, h1.inches + h2.inches)
    End Operator
End Structure
```

La routine doit être Shared, de plus si on surcharge certains opérateurs, il faut aussi surcharger leur inverse : si on surcharge '>' , il faut surcharger '<'.

### Surcharge de IsTrue, IsFalse CType

Si on teste un boolean, il a la valeur True ou False.

Si par contre je crée une classe nommée 'Personne', je peux définir comment une instance sera considérée comme égale à True. Il faut surcharger l'opérateur IsTrue en lui indiquant dans quelle condition l'instance sera considérée comme =True.

Exemple

J'ai une instance e de type Personne, si e.Present =True, dans ce cas je veux que e soit considéré comme True, il faut écrire dans la Classe 'personne' :

```
Public Shared Operator IsTrue(ByVal e As Personne) As Boolean

If e Is Nothing Then

    Return False

Else

    Return e.Present

End If

End Operator
```

Pour définir l'opérateur IsFalse, c'est simple : c'est Not e

```
Public Shared Operator IsFalse(ByVal e As Personne) As Boolean

    Return Not e

End Operator
```

Ensuite je pourrai utiliser des instructions de la forme :

```
If e then…
```

### Surcharge de CType

Je peux définir dans une classe comment CType va fonctionner.

Pour cela dans la classe Personne, je vais définir les 3 possibilités :

```
Public Shared Widening Operator CType(ByVal e As Personne) As String

Return e.Nom + " " + e.Prenom

End Operator


Public Shared Widening Operator CType(ByVal  e As Personne) As Date

If e Is Nothing Then

    Return Nothing

Else

    Return e.DateDeNaissance
```

```
End If

End Operator


Public Shared Widening Operator CType(ByVal  e As Personne) As Boolean

    If e Is Nothing Then Return False Else Return e.Present

End Operator
```

**Ainsi**

CType(UnePersonne,String) retourne Nom Prenon.

CType(UnePersonne,Date) retourne la date de naissance.

CType(UnePersonne,Boolean) retourne True ou False.

## XV-B-2-h - Surcharge de ToString

On va créer une classe 'temperature' et on va surcharger ToString pour créer un format propre à 'temperature'.

```
Public Class Temperature
    Private temp As Decimal

    Public Sub New(ByVal temperature As Decimal)
        Me.temp = temperature
    End Sub

    Public Overrides Function ToString() As String
        Return Me.temp.ToString("N3") + "°C"  'format nombre avec 3 chiffres après la virgule =
 '°C'
    End Function
End Class
```

Pour utiliser :

```
        Dim currentTemperature As New Temperature(23.6D)
        Console.WriteLine("The current temperature is {0}.", currentTemperature)
        'Affiche The current temperature is 23,600°C.
```

## XV-C - Créer un composant (Bibliothèque de Classe et de Contrôles)

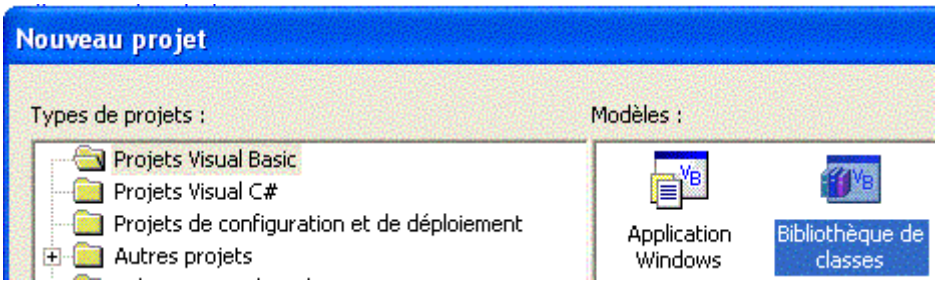

**On a vu qu'on pouvait CRÉER SES PROPRES CLASSES dans un projet, mais on peut aussi :**

- **créer une Classe autonome** qui sera utilisée par plusieurs autres programmes, c'est une **Bibliothèque de Classe ;**
- **créer un contrôle utilisateur** utilisé lui aussi par d'autres programmes.

Maintenant nous allons créer des classes ou contrôles, ils seront utilisés par une application cliente.

## XV-C-1 - Créer une Bibliothèque de classes

Pour créer une bibliothèque de Classe, il faut faire menu 'Fichier', 'Nouveau', 'Projet' :




Cliquer sur l'icône 'Bibliothèque de Classes'.

Le nom par défaut est **ClassLibrary1**, valider sur *OK*.

Dans la fenêtre principale, il y a :

```
Public Class Class1
End Class
```

On peut écrire le code, la description d'une classe avec ses propriétés, ses méthodes, ses constructeurs… (Voir page précédente.)

On peut **ajouter une autre Classe** (Menu Projet, ajouter une Classe), ou **importer une Classe** (Menu Projet, Ajouter un élément existant)

*Il n'y a **pas de procédure Sub Main.** (c'est évident, un composant n'est jamais autonome, c'est l'application cliente qui a cette procédure).*

*Une bibliothèque de classe ne possède pas les composants que possède une application Windows, **il n'y a pas d'interface utilisateur,** pas de MessageBox, pas de gestion du curseur, c'est l'application cliente qui s'occupe de gérer l'interface.*

## XV-C-1-a - Namespace

Permet de **créer un espace de noms** dans le composant :

```
NameSpace Outils
End
```

Il peut y avoir plusieurs niveaux :

```
NameSpace Outils
NameSpace Marteau
…
End
End
```

Équivalent à :

```
NameSpace Outils
```

```
Classe Marteau
…
End Class
End
```

Dans l'application il faudra après avoir référencé le composant (la Dll) importer l'espace de noms pour utiliser le composant.

```
Imports Outils.Marteau
```

## XV-C-1-b - Utilisation du composant

Il faut enfin enregistrer la bibliothèque, la compiler.

**Comment utiliser ce composant ?**

- Si la bibliothèque de Classe a été compilée, on obtient une DLL.
  Il faut **la référencer :** Ajouter la Dll au projet (Menu Projet, Ajouter une référence)
  Importer l'espace de noms par **Imports espace de noms** au début du module.
  On peut ensuite utiliser la Classe dans l'application cliente.
- On peut travailler en même temps sur l'application cliente et le projet de la bibliothèque de Classe en les chargeant tous les 2 dans **une solution.**

## XV-C-2 - Créer un 'contrôle utilisateur' à partir d'un contrôle existant en le modifiant

Permet de **créer un contrôle spécifique** qui enrichira la 'Boite à outils'.

(Pour les 'anciens' c'est comme un contrôle OCX, sauf que c'est un contrôle .NET et que c'est un fichier .dll.)

Exemple : créer un bouton avec un aspect spécifique à partir du bouton habituel (Ajout d'un cadre).

1 **Il faut créer une bibliothèque de contrôle.**
Pour créer une bibliothèque de Contrôle, il faut faire menu Fichier, Nouveau, Projet, Icône '**Bibliothèque de contrôle Windows' :** Nom du projet : WindowControle par exemple.
On obtient une fenêtre qui ressemble à un formulaire, mais sans bord,on peut y ajouter un contrôle (un bouton par exemple).
Si on regarde le code correspondant, Vb a créé une Classe UserControl1 qui hérite de la classe **Forms.UserControl :**

```
Public Class UserControl1
Inherits System.Windows.Forms.UserControl
End Class
```

Il suffit de substituer à **UserControl** le nom du contrôle que vous voulez utiliser comme base pour hériter de tous ses éléments. On remplace donc par :

```
Public Class MonBouton
Inherits System.Windows.Forms.Button
End Class
```

Le 'Design' devient :


2 **Il faut modifier l'aspect graphique du bouton.**

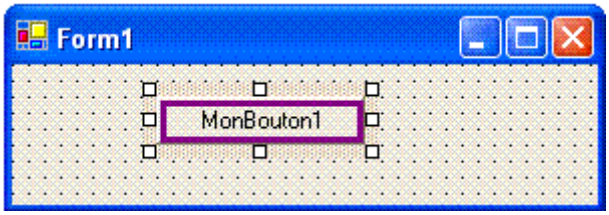
Pour cela si vous voulez modifier l'apparence du contrôle, il faut remplacer la méthode **OnPaint** de Button par la vôtre (celle-ci dessine le contrôle). Au sein de cette méthode, vous devez appeler la méthode **OnPaint** de la base (de la classe mère), puis ajouter vos propres fonctions de dessin.
Il faut donc ajouter la procédure :

```
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
MyBase.OnPaint(e) 'Appel à la méthode de la classe de base, ce qui dessine le bouton
Dim myPen As New Pen(Color.Purple, 3)
e.Graphics.DrawRectangle(myPen, 3, 3, Me.Width - 6, Me.Height - 6) 'Ajoute un cadre sur
 le dessin du bouton
End Sub
```

On rappelle que l'argument e est le graphique du bouton.

3 **Compilation**
Quand le composant est terminé, il faut créer la Dll.
Menu 'Générer' puis 'Générer WindowControle'
On obtient **WindowControle.Dll** dans le répertoire **/bin** sous les sources.

4 **Utilisation du composant dans une autre application VB**
Dans un autre projet VB, si je veux utiliser mon composant MonBouton, il faut l'**ajouter dans la boite à outils.**
Pour cela, cliquer avec le bouton droit de la souris dans la boite à outils. Un menu contextuel s'ouvre, cliquer sur *'Ajouter/Supprimer des éléments'* puis dans la fenêtre qui s'ouvre cliquer sur *'parcourir'…* cliquer sur **WindowControle.Dll**, le composant **MonBouton** apparait dans la Boite à outils.
Il suffit de le saisir et de le déplacer dans un des formulaires comme un bouton normal.




Toutes les procédures événements (Comme **MonBouton1_Click**) sont disponibles, elles ont été héritées de Button.

**Ce qu'il faut comprendre**

C'est que votre nouveau bouton hérite de **Control**, une classe qui possède un grand nombre d'événements et de méthodes qui n'apparaissent pas dans les listes déroulantes, car inutiles dans l'utilisation de l'objet, mais très utiles dans le comportement d'un objet. C'est le cas de **OnPaint OnKeyDown** et **OnMouseDown…** qui déclenchent les événements **Paint, KeyDown, MouseDown**. On redéfinit ces méthodes (avec Overrides), dans la méthode redéfinie on appelle quand même la méthode de la classe mère puis on ajoute les modifications de fonctionnalités.

Exemple

Différencier une zone droite et une zone gauche sur le bouton.

On utilise l'événement **OnMouseDown**, il a pour paramètre **e** qui contient les coordonnées de la souris (**e.x et e.y**)

```
Protected Overrides Sub OnMouseDown(ByVal e As MouseEventArgs)

MyBase.OnMouseDown(e) 'Appel à la méthode de la classe de base
If e.X < Me.Width / 2 Then
    MessageBox.Show("Click sur partie gauche")
Else
    MessageBox.Show("Click sur partie droite")
End if
End Sub
```

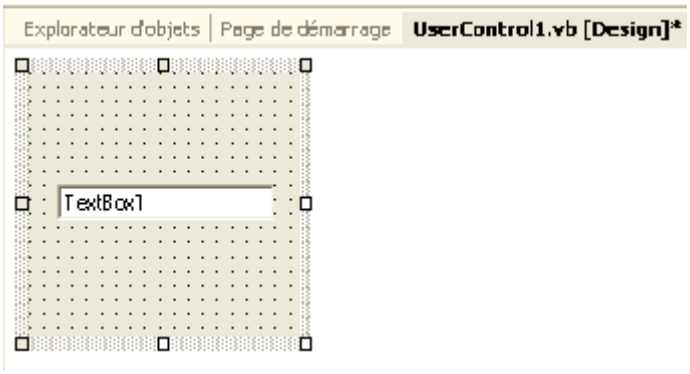
MessageBox peut être remplacé par un raisevent pour déclencher un événement.

*On vient de créer un composant héritant d'un contrôle, puis on en a modifié les fonctionnalités.*

## XV-C-3 - Créer un 'contrôle utilisateur' contenant un ou plusieurs contrôles pilotés

Pour créer une bibliothèque de Contrôle (un contrôle utilisateur), il faut faire menu Fichier, Nouveau, Projet, Icône **'Bibliothèque de contrôle Windows' :** Nom du projet : **WindowControle** par exemple.

On obtient une fenêtre qui ressemble à un formulaire, mais sans bord,on peut y ajouter un contrôle (un textBox par exemple comme ici) ou plusieurs contrôles.




*Ici la zone de fond grise est importante, le contrôle créé correspond à la totalité de la zone grise du formulaire sans bord, ce qui n'est pas pratique quand on utilise le contrôle (j'en ai dessiné un petit, et je ne voyais pas la textebox !!) Il est donc conseillé de réduire la surface.*

Si on regarde le code correspondant, Vb a créé une Classe **UserControl1** qui hérite de la classe **Forms.UserControl**

```
Public Class UserControl1
Inherits System.Windows.Forms.UserControl
End Class
```

**Dans ce Usercontrol** il y a les procédures **privées** des événements des composants,

```
Private Button1_Click
End Sub
```

Bien sûr, elles ne seront pas visibles ni accessibles par l'utilisateur du composant.

**L'interface du Usercontrol** (ce que verra l'utilisateur du composant) est créée de toute pièce comme dans un module de Class. Si on double-clique sur le fond, on voit quand même apparaitre :

```
Private Sub UserControl1_Load(…
End Sub
```

Mais il faut rajouter **des membres Public** qui seront accessibles à l'utilisateur du composant. On utilise pour cela les **'Property'**, **'Sub'** et **'variables' Public** pour créer une interface. Le code contenu dans ces procédures de l'interface va **'piloter'** le ou les contrôles (comme le **TextBox1**). Ce code modifie (dans notre exemple) le comportement du TextBox initial.

Ici je vais créer une propriété LeTexte qui est le texte qui sera affiché dans le **TextBox1**. Cette propriété **LeTexte** va 'piloter' **TextBox1.text**. Je modifie le comportement de **TextBox1.text** en empêchant d'afficher Toto (c'est idiot !!).

```
Public Property LeTexte() As String
    Get
        LeTexte=TextBox1.Text
    End Get
    Set (ByVal Value As String)
        If Value <> "toto" Then
            TextBox1.Text= Value
        End if
    End Set
End Property
```

Je génère la solution, ajouter WindowControle.Dll à la boite à outils, je mets le nouveau composant sur un formulaire, il se nommera UserControl1.

Pour utiliser la propriété précédemment écrite :

```
UserControl1.LeTexte="lulu"     'lulu apparait dans le textbox

UserControl1.LeTexte="toto"     'rien n'apparait dans le textbox: : j'ai bien modifié le
 comportement du textbox
'et c'est transparent pour l'utilisateur.
```

*On vient de créer un composant avec la Classe UserControl (au lieu de Forms dans un formulaire), on a écrit son interface.*

**Remarque :** si on veut avoir accès à une sub événement du contrôle qui est dans un composant, il faut que dans le composant la Sub événement soit Public.

**Autre exemple**

Voir cet autre **remarquable exemple** de création d'un composant (par CGi et neo51 sur developpez.com): **Composant horloge :** code source, explication claire : un régal.

## XV-D - Les interfaces


### XV-D-1 - Définition : Interface et implémentation

Ce que je vois de l'objet, c'est son **interface** (le **nom** des propriétés, le **nom** des méthodes…) exemple : le nom de la méthode **Clear** fait partie de l'interface d'une ListBox. Par contre le code qui effectue la méthode (celui qui efface physiquement toutes les lignes de la ListBox), ce code se nomme **implémentation**, lui n'est ni visible ni accessible (quand on est du côté du développeur qui utilise l'objet).

Un objet a donc une interface et une implémentation.

Quand maintenant on est du côté du créateur d'objet, dans un module de classe, si on a créé un objet et ses membres, sans le savoir, on crée en même temps l'interface et l'implémentation.

Mais il est possible de dissocier les 2 ?

**Quel intérêt d'utiliser les interfaces ?**

Elles permettent **d'organiser le code** et de **créer un 'contrat' sur les méthodes** qui seront présentes dans les classes qui l'utilisent.

Vous pouvez développer des implémentations avancées pour vos interfaces sans endommager le code existant, ce qui limite les problèmes de compatibilité. Vous pouvez également ajouter de nouvelles fonctionnalités à tout moment en développant des interfaces et implémentations supplémentaires.

Différences entre Classe et Interface

Tout comme les classes, les interfaces définissent un ensemble de propriétés, méthodes et événements. Cependant, contrairement aux classes, les interfaces n'assurent pas l'implémentation. Elles sont implémentées par les classes et définies en tant qu'entités distinctes des classes.

Les interfaces ne doivent pas être modifiées après publication. En effet, toute modification apportée à une interface publiée risque d'endommager le code existant. Il faut partir du principe qu'une interface est une sorte de contrat. (La partie qui publie une interface accepte de ne jamais modifier cette dernière et l'implémenteur accepte de l'implémenter exactement comme elle a été conçue.) Si on modifie l'interface, malgré tout, il faut modifier l'implémentation dans toutes les classes qui utilisent cette interface.

**Comme d'habitude, il y a :**

- **les interfaces présentes dans les classes du Framework (IList, ICollection…)** ;
- **les interfaces que vous créez de toutes pièces pour créer des objets.**

Visual Basic .NET vous permet de définir des interfaces à l'aide de l'instruction **Interface** et de les implémenter avec le mot-clé **Implements.**

## XV-D-2 - Les interfaces présentes dans les classes du Framework

Pour 'uniformiser' le comportement des objets, les interfaces sont largement utilisées dans VB.

Prenons l'exemple **des collections.**

Plutôt que de rendre communs à toutes les collections une méthode (Clear par exemple), VB donne la même interface à plusieurs types de Collections, ce qui uniformise la totalité des membres.

Les collections reposent sur l'interface **ICollection, IList ou IDictionary.** Les interfaces **IList et IDictionary** sont toutes les deux dérivées de l'interface **ICollection**.

*Le nom des interfaces commence toujours par 'I'.*

Dans les collections fondées sur l'interface **IList** ou directement sur l'interface **ICollection** (telles que Array, ArrayList, Queue ou Stack), chaque élément contient une valeur unique. Dans les collections reposant sur l'interface **IDictionary** (telles que Hashtable ou SortedList), chaque élément contient à la fois une clé et une valeur.

Détaillons l'interface Ilist.

L'interface Ilist permet de présenter une collection d'objets accessibles séparément par index.

Les méthodes de l'interface IList sont répertoriées ici.

**Méthodes publiques**

| | |
|---|---|
| Add | Ajoute un élément. |
| Clear | Supprime tous les éléments. |
| Contains | Détermine si la liste contient une valeur spécifique. |
| IndexOf | Détermine l'index d'un élément spécifique. |
| Insert | Insère un élément dans la liste à la position spécifiée. |
| Remove | Supprime la première occurrence d'un objet spécifique. |
| RemoveAt | Supprime l'élément correspondant à l'index spécifié. |

**Propriétés publiques**

| | |
|---|---|
| IsFixedSide | Obtient une valeur indiquant si IList est de taille fixe. |
| IsReadOnly | Obtient une valeur indiquant si IList est en lecture seule. |
| Item | Obtient ou définit l'élément correspondant à l'index spécifié. |

Les tableaux (Array) utilisent l'interface Ilist, mais aussi les collections (ArrayList), des contrôles utilisent aussi cette interface (les ListBox, ComboBox), mais aussi les DataView…

Les ListBox possèdent donc l'interface Ilist , on s'en doutait, car on utilisait les méthodes Clear, Insert, Item…

Il y a plein d'autres interfaces.

Autre exemple : **IEnumerable.**

La Classe **System.Array** (et d'autres) implémente l'interface **IEnumerable**, ce qui permet d'utiliser une boucle For Each pour parcourir tous les éléments de l'Array.

(Voir le chapitre sur les Patron pour plus de détail.)

## XV-D-3 - Les interfaces créées par le programmeur

De même que vous savez créer des classes, il est possible de créer de toutes pièces des interfaces.

**Pour créer une Interface**

- Dans un nouveau Module, définissez votre Interface en commençant par le mot-clé Interface et le nom de l'interface et se terminant par l'instruction End Interface. Par exemple, le code suivant définit une Interface appelée Cryptage :

```
Interface Cryptage
End Interface
```

- Ajoutez des instructions définissant les propriétés, méthodes et événements pris en charge par votre Interface. Par exemple, le code suivant définit deux méthodes, une propriété et un événement :

```
Interface Cryptage
   Function Encrypt(ByVal estring As String) As String
   Function Decrypt(ByVal dstring As String) As String
   Property CledeCodage() As Integer
   Event FinDecoding(ByVal RetVal As Integer)
End Interface
```

L'interface est créée.

**Pour implémenter une Interface**

- Si l'interface que vous implémentez ne fait pas partie de votre projet, ajoutez une référence à l'assembly qui contient l'interface.
- **Créez une nouvelle classe** qui implémente votre Interface et ajoutez le mot-clé Implements dans la ligne à la suite du nom de la classe. Par exemple, pour implémenter l'interface Cryptage, vous pouvez nommer la classe d'implémentation MonEncrypte, comme dans le code suivant :

```
Class MonEncrypte
Implements Cryptage
End Class
```

- Ajoutez des procédures **pour implémenter les propriétés, méthodes et événements** de la classe :

```
Class MonEncrypte
   Implements Cryptage
   Event FinDecoding(ByVal RetVal As Integer) Implements Cryptage.FinDecoding

   Function Encrypt(ByVal estring As String) As String Implements Cryptage.Encrypt
      ' Placer le code de cryptage ici.
   End Function
   Function Decrypt(ByVal dstring As String) As String Implements Cryptage.Decrypt
      ' Placer le code de décryptage ici.
   End Function

   Property CledeCodage() As Integer Implements Cryptage.CledeCodage
      Get
         'Placer ici le code qui retourne la valeur de la propriété.
      End Get
      Set
         'Placer ici le code qui donne une valeur à la propriété.
      End Set
   End Property
End Class
```

Noter que :

Pour chaque membre implémenté dans ce code, une instruction **Implements** indique le nom de l'interface et du membre implémenté.

**Tous** les membres de l'interface doivent être implémentés.

Enfin utiliser la classe MonEncrypte dans votre programme.

```
Dim C As New MonEncrypte()
C.CledeCodage=3
Dim ChaineEncryptée As String= C.Encrypt( "ChaineAEncrypter")
```

Ou

Il faut créer une instance de la classe qui implémente MonEncrypte, crée une variable du type de l'interface, qui associe un gestionnaire d'événements à l'événement déclenché par l'instance, qui définit une propriété et exécute une méthode via l'interface.

```
Dim C As New MonEncrypte() 'Classe
Dim I As Cryptage()'Variable d'interface (ne pas mettre de 'New'
I=C
I.CledeCodage=3
Dim ChaineEncryptée As String = I.Encrypt( "ChaineAEncrypter")
```

***Les 2 versions marchent.***

S'il y a un RaiseEvent dans une procédure qui déclenche un événement de la classe, il faut aussi ajouter une ligne AddHandler.

**Il peut y avoir héritage de plusieurs interfaces :**

```
Interface IComboBox
   Inherits ITextBox, IListBox
End Interface
Public Class EditBox
   Inherits Control
   Implements ITextBox
   Implements IListBox
   Public Sub Paint()Implements ITextBox.Paint
      …
   End Sub
   Public Sub Bind(b As string) Implements IListBox.Clear
   End Sub
End Class
```

## XV-E - L'héritage


### XV-E-1 - Définition de l'héritage

À partir d'une classe existante, la ***classe de base (ou classe mère), on peut créer*** une nouvelle classe, la ***classe dérivée (ou classe fille)*** qui hérite des propriétés de la classe de base. La classe fille peut être modifiée.

Exemple

Soit la Classe 'Animal', on peut créer une Classe 'Cheval' qui aura toutes les propriétés de 'Animal'.

La Classe 'Cheval' **est un** 'Animal'. Quand on peut dire 'est un', il s'agit bien d'héritage.

**Une classe peut hériter d'une autre classe**, il suffit d'utiliser : 'Inherits'.

**Inherits** permet de déclarer une nouvelle classe, la ***classe dérivée (ou classe fille),*** basée sur une classe existante, la ***classe de base (ou classe mère)***. Les classes dérivées héritent des propriétés, des méthodes, des événements, des champs et des constantes de la classe de base et peuvent les étendre.

Voici une classe de base :

```
Class Salarié1
Public Property SalaireAnnuel() As Integer
….
```

```
End Property

End Class
```

Créons une classe dérivée qui hérite de Salarié1 :

```
Public Class Salarié2
Inherits Salarié1

End Class
```

On peut ajouter :

**MustInherit :** cela donne une classe non instanciable, on ne peut pas créer d'objet avec !! Alors à quoi cela sert !! À fournir une base pour des classes qui en hériteront. On appelle ces classes des **classes abstraites ;**

**NotInheritable :** cette classe ne peut-être héritée.

## XV-E-2 - Membres de la classe dérivée

**La classe fille possède tous les membres de la classe mère.**

Cela si le membre est 'Protected' ou 'Public', pas s'il est Private.

Exemple :une variable 'Private' n'est pas visible dans la Classe fille.

Une variable 'Public' est visible dans la Classe fille, mais aussi par l'utilisateur de l'objet.

Une variable 'Protected' est visible dans la Classe fille, mais pas à l'extérieur.

Dans la classe Salarié2, on peut utiliser la méthode SalaireAnnuel.

Il est possible de **rajouter** des membres propres à la classe fille, mais aussi de **redéfinir, de surcharger ou de masquer des membres de la classe mère.**

## XV-E-2-a - Redéfinition de membres (Overrides)

Il est possible en plus de **redéfinir (de substituer, de remplacer) un des membres** de la classe mère dans la classe fille. (De créer une nouvelle définition du membre dans la classe fille et uniquement pour cette classe fille si besoin.) Pour que cela marche, il faut que le membre de la classe mère soit modifiable (**overridable**) et que le membre de même nom de la classe fille soit modifié (**Overrides**).

*Dans la Classe fille (classe dérivée) :*

**Overrides**
Indique que cette procédure Sub substitue une procédure de même nom dans une classe de base. Le nombre et les types de données des arguments doivent correspondre exactement à ceux de la procédure de la classe de base. Dans la Classe mère (classe de base) ;

**Overridable**
Indique que cette procédure peut être substituée par une procédure de même nom dans une classe dérivée. Overridable est le paramètre par défaut ;

**NotOverridable**

Indique que cette procédure ne peut pas être substituée dans une classe dérivée. NotOverridable est le paramètre par défaut d'une procédure qui ne se substitue pas à une procédure de classe de base ;

**MustOverride**
Indique que cette procédure Sub n'est pas implémentée dans cette classe et qu'elle doit l'être dans une classe dérivée pour que cette classe puisse être créée.

Exemple:

Créons une Classe Salarié1 avec une méthode 'Salaire annuel sur 13 mois'

```
Class Salarié1
Public Overridable ReadOnly Property SalaireAnnuel() As Integer
Get
    SalaireAnnuel = SalaireMensuel * 13
End Get
End Property
End Class
```

Créons maintenant une classe Salarié2 qui hérite de toutes les propriétés public et protected de la classe salarié1 donc la méthode SalaireAnnuel qui est sur 12 mois :

```
Public Class Salarié2
Inherits Salarié1
Public Overrides ReadOnly Property SalaireAnnuel() As Integer
Get
    SalaireAnnuel = SalaireMensuel * 12
End Get
End Property
End Class
```

Quand on instancie un objet avec la classe Salarié1, si on utilise la méthode SalaireAnnuel() il sera calculé sur 13 mois.

Quand on instancie un objet avec la classe Salarié2, si on utilise la méthode SalaireAnnuel() il sera calculé sur 12 mois.

Attention le membre substitué doit avoir la même signature (les mêmes paramètres).

## XV-E-2-b - Surcharge de membres (Overloads)

Cela crée plusieurs membres de même nom, mais avec des signatures différentes. Il peut y avoir une version dans la classe de base et une version surchargée de même nom, mais avec une signature différente dans la classe fille.

**Overloads**
Indique que ce membre surcharge un ou plusieurs membres définis avec le même nom dans une classe de base. La liste d'arguments de cette déclaration doit être différente de la liste d'arguments de chaque membre surchargé. Les listes doivent différer au niveau de leur nombre d'arguments, de leurs types de données ou des deux. Cela permet au compilateur de distinguer la version à utiliser.

Exemple :

```
Public Overloads ReadOnly Property SalaireAnnuel( Prime As Integer) As Integer
Get
    SalaireAnnuel = (SalaireMensuel * 12) + Prime
End Get
End Property
```

Vous ne pouvez pas spécifier Overloads et Shadows dans la même déclaration.

## XV-E-2-c - Cacher un membre de la classe de base (Shadows)

"Shadows"

Indique que ce membre cache un élément de programmation de même nom ou un ensemble d'éléments surchargés, dans une classe de base.
Vous pouvez occulter tout type d'élément déclaré par un autre type. Si vous masquez une procédure avec une autre procédure, les arguments et le type retourné n'ont pas besoin de correspondre à ceux de la procédure de la classe de base.
Un élément occulté est indisponible à partir de la classe dérivée qui l'occulte, à moins que l'élément d'occultation soit inaccessible, comme c'est le cas de Private.

## XV-E-2-d - Classe abstraite

Une classe abstraite est une classe avec laquelle on ne peut pas créer (instancier) directement d'objet. Elle sert uniquement à créer des classes dérivées.

**CollectionBase** est une **classe abstraite** (ne pouvant pas être utilisée telle quelle), on peut créer une classe qui en hérite.

**CollectionBase** contient déjà quelques fonctions propres aux collections (**Clear et Count**), les fonctions qui manquent, qui n'existent pas (**Add, Remove, Item**) vont être implémentées par vous et à votre manière.

Une propriété **Protected** appelée **List** est fournie par CollectionBase et utilisée pour le stockage et l'organisation interne. Quand on crée Add, Remove, Item, on utilise cette propriété List.

```
Public Class MaCollection
Inherits System.Collections.CollectionBase
Public Sub Add(ByVal a As Salarié)
' Appelle la méthode Add de l'objet List pour ajouter un salarié.
List.Add(a)
End Sub

End Class
```

## XV-E-3 - MyBase

Dans le membre de la classe fille, on peut avoir besoin d'appeler le membre de la classe mère, on le fait avec **MyBase :**

```
Public Overrides Property OnPaint()  'on redéfinit OnPaint
MyBase.OnPaint                'on appelle le OnPaint de la Classe mère
…                    'on ajoute de nouvelles choses
End Property
```

Se souvenir que Me est l'instance en cours, MyClass est aussi l'instance en cours si la méthode est overridée.

## XV-E-4 - Constructeur dans une classe fille

Les membres privés de la classe mère, comme on l'a dit, ne sont pas accessibles à partir de la classe fille.

Seuls les membres **'Public'** et **'Protected'** de la classe mère sont accessibles à partir de la classe fille, il faut donc utiliser ces membres dans la classe fille.

Exemple avec un constructeur :

```
Public Class Mere
'Attribut privé
Private _Nom As String

'Constructeur
Public Sub New( ByVal Nom As String)
    _Nom=Nom
End Sub
End Class


Public Class Fille
Inherits Mere
'Constructeur
Public New ( ByVal Nom As String)
    MyBase.New (Nom)
End Sub
End Class
```

On voit ici que dans la classe fille, on appelle le constructeur de la classe mère.

Car dans la classe fille _Nom de la classe mère n'est pas accessible.

Dans une clase fille, on passe donc les paramètres à la classe mère en utilisant les membres 'Public' ou 'Protected' de cette classe mère, on initialise en plus directement les attributs propres à la classe fille s'ils existent.

## XV-E-5 - Héritage successif : exemple

Une classe peut hériter d'une classe qui en hérite d'une autre.

Prenons l'exemple suivant :

C hérite de B qui hérite de A, les membres sont hérités s'ils sont Overridables.

```
Class A
Public Overridable Sub F() ' le membre F pourra être modifié dans une classe fille
Console.WriteLine("A.F")
End Sub

Public Overridable Sub G()'le membre G pourra être modifié dans une classe fille
Console.WriteLine("A.G")
End Sub
End Class

Class B
Inherits A    'Hérite de A
Public Overrides NotOverridable Sub F()    'On interdit la modification de F dans une Classe
 fille
Console.WriteLine("B.F")
End Sub
Public Overrides Sub G()
Console.WriteLine("B.G")
End Sub
End Class

Class C    'Hérite de B qui hérite de A
Inherits B
Public Overrides Sub G()
Console.WriteLine("C.G")
End Sub
End Class
```

*En VB.Net une Classe ne peut hériter que d'une seule Classe.*

## XV-E-6 - Création de classes à partir de classes du Framework

**Il est possible de créer une classe qui hérite d'une classe du Framework.**

Exemple d'une classe MyArray héritant de la Collection ArrayList, on peut dans la classe appeler des membres de la classe de base **(MyBase.Add(S))** ou modifier les membres de cette classe de base (ici on 'cache' les membres de la classe de base par **Shadows** et on crée ses propres membres.

```
Imports System.Collections
Public Class MyArray
     Inherits ArrayList 'MyArray héritant de la Collection ArrayList


     Public  Shadows Sub Add(ByVal S As Salarié)
          MyBase.Add(S)        'On appelle la méthode Add de la classe de base (classe mère)
     End Function

      Public Shadows ReadOnly Property Index(ByVal i As Integer) As Salarié
          Get
               Return CType (MyBase.Item(i), Salarié)
          End Get
     End Property


End Class
```

## XV-E-7 - Création de composants et héritage

On a vu que dans la création de composants, on peut utiliser un composant qui existe déjà :

```
Public Class MonBouton
Inherits System.Windows.Forms.Button
End Class
```

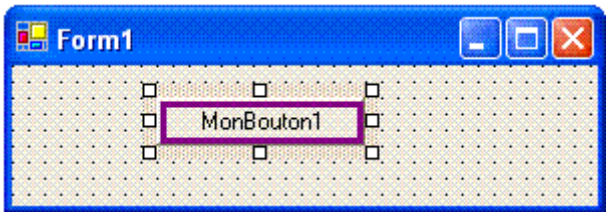
Ici on crée un composant MonBouton qui hérite de Button, ce composant Monbouton fonctionne exactement comme Button (le bouton habituel).

Pour modifier l'apparence du bouton, il faut remplacer (Overrides) la méthode OnPaint de Button par la vôtre (celle-ci dessine le contrôle). Au sein de cette méthode, vous devez appeler la méthode OnPaint de la base qui dessine le bouton habituel, puis ajouter vos propres fonctions de dessin.

Il faut donc ajouter dans la classe MonBouton la procédure :

```
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
MyBase.OnPaint(e) 'Appel à la méthode de la classe de base, ce qui dessine le bouton
Dim myPen As New Pen(Color.Purple, 3)
e.Graphics.DrawRectangle(myPen, 3, 3, Me.Width - 6, Me.Height - 6) 'Ajoute un cadre sur le dessin
 du bouton
End Sub
```

Si on compile cette Classe et qu'on ajoute le composant à un projet, on obtient le bouton suivant :




## XV-F - Les espaces de noms, portée des classes et membres (friend protected public private)


### XV-F-1 - Intérêts des espaces de noms (NameSpace)

On peut créer une Classe dans un espace de noms.

Le but de ces espaces de noms est d'éviter les conflits et ambiguïtés sur les objets.

**Exemple :** deux programmeurs Prog1 et Prog2 distribuent des classes qui sont empaquetées et distribuées respectivement dans les *dll, Prog1.dll et Prog2.dll.*

Les deux programmeurs ont défini une classe nommée *PrintText*. N'ayant aucune relation, ils l'ont appelée de la même manière !!

Si dans un programme incluant les 2 dll, vous utilisez la classe *PrintText*, VB ne saura pas s'il doit prendre la classe *PrintText* de *Prog1.dll* ou celle de *Prog2.dll.*

Si le premier programmeur crée ses classes dans un espace de noms appelé Prog1 et le second programmeur dans un espace de noms appelé prog2, les deux classes s'appelleront alors *Prog1.PrintText* et *Prog2.PrintText*, ce qui lève toute ambiguïté.

### XV-F-2 - Pour créer une classe dans un espace de noms

On crée une Classe, puis on ajoute le **NameSpace** sur la ligne dessus.

```
Namespace Prog1
Public Class PrintText
' définition de la classe
….
…
End Class
End Namespace
```

Ensuite dans le programme on peut utiliser

```
Prog1.PrintText
```

ou bien

```
Imports Prog1
```

puis **PrintText**

Un programme a son propre espace de noms (qui est le nom du programme) : si dans **MyProgramme**, il y a le NameSpace **MySpace** contenant la classe **MyClasse**, on peut utiliser

```
MyProgramme.MySpace.MyClasse
```

## XV-F-3 - Portée des Classes, procédures, membres

On savait que les procédures pouvaient être **publiques** ou **privées**.

En fait pour indiquer une portée, en particulier dans une classe, les membres peuvent être :

**Public**
Les procédures déclarées avec le mot-clé Public ont un accès public. Il n'existe aucune restriction quant à l'accessibilité des procédures publiques ;

**Protected**

**Dans un module de classe**
Les procédures déclarées avec le mot-clé Protected ont un accès protégé. Elles sont accessibles seulement à partir de leur propre classe ou d'une classe dérivée.

**Friend**
Les procédures déclarées avec le mot-clé Friend ont un accès ami. Elles sont accessibles à partir du programme contenant leur déclaration et à partir de n'importe quel autre endroit du même assembly.

**Protected Friend**
Les procédures déclarées avec les mots-clés Protected Friend ont l'union des accès ami et protégé. Elles peuvent être utilisées par du code dans le même assembly, de même que dans les classes dérivées. L'accès Protected Friend peut être spécifié uniquement pour les membres des classes.

**Private**
Les procédures déclarées avec le mot-clé Private ont un accès privé. Elles ne sont accessibles qu'à partir de leur contexte de déclaration, y compris à partir des membres de types imbriqués, tels que des procédures.

## XV-G - Composition et groupe d'objets : Tableau, collection d'objets, Classe contenant un groupe d'objets


**Est-il possible de mettre un objet dans un autre ?**

**On a souvent besoin d'utiliser un ensemble, un groupe d'objets. Comment faire ?**

## XV-G-1 - Un Objet dans un autre : Composition d'objets

On parle de contenant-contenu.

On peut créer une Classe qui contient des Objets, une classe qui se **compose** d'objets.

Exemple classique

Créons une Classe Point.

```
Class Point
Private _x As Integer
Private _y As Integer
Public Sub New(ByVal x As Integer, y As Integer)
          _x=x
          _y=y
End Sub
….
End Class
```

On a écrit uniquement le constructeur, il faudrait aussi écrire les property permettant de lire et écrire les coordonnées du point.

Maintenant on a besoin de créer une Classe rectangle qui est définie par 2 points (le coin supérieur gauche et le coin inférieur droit) :

```
Class Rectangle
Private _p1 As Point
Private _p2 As Point
Public Sub New(ByVal x1 As Integer, ByVal y1 As Integer,ByVal x2 As Integer, ByVal y2 As Integer)
          _p1= New Point (x1,y1)
          _p2= New Point (x2,y2)
End Sub
…
End Class
```

Et bien, on utilise la classe Point dans la Classe Rectangle. Le constructeur de la classe rectangle instancie 2 points, ce qui appelle le constructeur de la Classe Point.

## XV-G-2 - Groupe d'objets

Exemple : créons une Classe 'Salarié' avec une Property Nom et un constructeur New nécessitant un nom.

```
Public Class Salarié
Private _sNom As String
Public Property Nom() As String
          Get
               Nom = _sNom
          End Get
          Set
               _sNom = Value
          End Set
     End Property

Public Sub New(ByVal sNom As String)
          Nom = sNom
End Sub

End Class
```

Ensuite pour travailler sur **un ensemble de salariés** on peut :

- utiliser un tableau ou une collection de Salariés (pas bien !!) ;
- créer une Classe contenant des Salariés (good !).

## XV-G-2-a - Comment utiliser un tableau ou une collection d'objets 'Salarié'

**Voyons comment faire avec une approche non objet :**

- Tableau

```
Dim LesSalaries(100) As Salarié
```

Attention, cela crée un tableau de références d'objets, mais pas les objets (Salariées(1) contient Nothing).

Il faut donc créer les objets:

```
LesSalaries(0) = New Salarié("toto")
LesSalaries(1) = New Salarié("lulu")
LesSalaries(2) = New Salarié("tata")
…
```

On peut ensuite utiliser **LesSalaries(1).Nom** pour connaitre le nom du salarié d'index 1.

- Collection

```
Dim LesSalaries As New ArrayList  'On crée une collection d'objet ArrayList
Dim s As New Salarié("toto")      'On crée un Salarié
LesSalaries.Add (s)               'On l'ajoute dans l'Arraylist

LesSalariées.Count   retourne 1
```

Pour afficher dans une MsgBox le nom du Salarié d'index 1 :

```
MsgBox(CType(LesSalaries.Item(0), Salarié).Nom)
```

On remarque que **LesSalaries.Item(0)** est de type Objet (Les éléments d'un **ArrayList** sont des objets; **Salaries.Item(0).GetType.BaseType.ToString retourne 'Objet')** il faut donc caster en 'Salarié' à l'aide de CType:

**CType(LesSalaries.Item(0), Salarié)** ensuite , on peut utiliser **.Nom**

En Vb2005 (Framework 2): 


On peut utiliser des collections '**génériques**'

```
Dim LesSalaries As New System.Collections.Generic.List(Of Salarié)
```

On peut ainsi créer une collection fortement typée de '**Salarié**'.

**Mais utiliser un tableau ou une collection d'objets directement accessibles n'est pas une bonne méthode.**

**La bonne méthode est de créer une classe qui encapsule la collection (Une Classe qui contient la collection). Voyons donc la suite.**

## XV-G-2-b - Utiliser Une Classe contenant des Salariés

Voyons comment faire avec une approche objet.

**Il faut créer une classe 'LesSalariés' contenant :**

- un tableau ou une collection 'Private' complètement encapsulée, donc non accessible à l'extérieur ;
- les méthodes 'Public' permettant d'avoir accès aux données et de les modifier.

**Il y a 4 manières de faire.**

## XV-G-2-b-i - Créer une Classe contenant une ArrayList

Voyons un exemple utilisant une ArrayList (collection d'objets) :

L'arrayList est créée par le constructeur.

**Noter l'encapsulation.**

- une méthode Add permet d'ajouter un 'Salarié' aux 'Salariés' (vu du côté utilisateur).
- La méthode Remove permet d'enlever un Salarié.
- Dans la classe une méthode Add permet d'ajouter un 'Salarié' à l'ArrayList.
- La Property Item permet de retourner un Salarié d'index lIndex.
- La Property Count retourne le nombre de Salariés.

```
Imports System.Collections
Public Class LesSalariésClasse
     Private maCol As ArrayList

     Public Sub New()
          maCol = New ArrayList()     'cela crée une ArrayList
     End Sub

     Public Function GetEnumerator() As IEnumerator     'permet d'utiliser For Each
          GetEnumerator = maCol.GetEnumerator
     End Function

     Public Function Add(ByVal LeNom As String) As Salarié
          Dim UnSalarié As New Salarié(LeNom)
          maCol.Add(UnSalarié)
          Add = UnSalarié
     End Function

      Public ReadOnly Property Item(ByVal lIndex As Integer) As Salarié
          Get
               Item = (CType(maCol.Item(lIndex),Salarié)
          End Get
     End Property

      Public ReadOnly Property Count() As Integer
          Get
               Count = maCol.Count
          End Get
     End Property

      Public Sub Remove(ByVal Key As String)
          maCol.Remove(Key)
     End Sub
End Class
```

Pour utiliser la Classe :

```
Dim LesSalariés As New LesSalariésClasse() 'création
LesSalariés.Add("Philippe")     'Ajout d'un salarié
LesSalariés.Count retourne 1    'connaitre le nombre de salariés
Pour afficher le nom du premier salarié dans une MessageBox:
MsgBox(LesSalariés.Item(0).Nom)
```

Bien sûr on peut utiliser une boucle **For Each** pour avoir accès à l'ensemble des salariés.

## XV-G-2-b-ii - Créer une Classe héritant de la Classe ArrayList

On crée une classe héritant de ArrayList, ensuite on va créer une méthode Add pour ajouter un Salarié et une Property permettant de lire le nom du salarié d'index i ; comme il existe déjà les membres **Add** et **Index** avec la même signature dans la classe parente, il faut remplacer ces membres, on le fait grâce à '**Shadows**'.

Dans les nouveaux membres, on appelle les membres de la classe mère (grâce à **MyBase**)

```
Imports System.Collections
Public Class Salariés
     Inherits ArrayList


     Public  Shadows Sub Add(ByVal S As Salarié)
          MyBase.Add(S)
     End Function

      Public Shadows ReadOnly Property Index(ByVal i As Integer) As Salarié
          Get
               Return CType (MyBase.Item(i), Salarié)
          End Get
     End Property


End Clas
```

Là aussi, les éléments d'un ArrayList sont des objets, il faut donc caster en 'Salarié' à l'aide de CType l'Item de l'ArrayList.

Dans l'exemple, on a utilisé une ArrayList, il est possible d'utiliser tout autre type de collections.

## XV-G-2-b-iii - Créer une Classe héritant de la Classe CollectionBase

**CollectionBase** est une **classe abstraite** (ne pouvant pas être utilisée telle quelle), on peut créer une classe qui en hérite.

Cela tombe bien : **CollectionBase** contient déjà quelques fonctions propres aux collections (**Clear et Count**), les fonctions qui manquent, qui n'existent pas (**Add, Remove, Item**) vont être implémentées par vous et à votre manière.

Une propriété **Protected** appelée **List** est fournie par CollectionBase et utilisée pour le stockage et l'organisation interne. Quand on crée Add, Remove, Item, on utilise cette propriété List.

```
Public Class MaCollection
Inherits System.Collections.CollectionBase
Public Sub Add(ByVal a As Salarié)
' Appelle la méthode Add de l'objet List pour ajouter un salarié.
List.Add(a)
End Sub

End Class
```

Remarquons que, bien que l'objet **List** soit non typé, cette méthode interdit l'ajout de tous les objets n'appartenant pas au type Salarié. Alors que CollectionBase n'était pas typée, MaCollection est **fortement typée,** car n'acceptant que des 'Salarié'.

On peut aussi ajouter une méthode éliminant un objet de la collection :

```
Public Sub Remove(ByVal index as Integer)
' contrôle.
```

```
If index > Count - 1 Or index < 0 Then
    System.Windows.Forms.MessageBox.Show("Index non  valide!")
Else
' Appelle la méthode RemoveAt de l'objet List.
List.RemoveAt(index)
End If
End Sub
```

On peut enfin ajouter Item :

```
Public ReadOnly Property Item(ByVal index as Integer) As Salarié
Get
Return CType(List.Item(index), Salarié)
End Get
End Property
```

Avec le Framework 2, il est possible d'utiliser, au lieu de ListArray, une collection générique fortement typée :

```
System.Collections.Generic.List(Of Salarié)
```

## XV-G-2-b-iv - Créer une Classe contenant une Classe générique

C'est possible avec le Framework 2.

Au lieu de créer une ArrayList dans la Classe, on peut créer une collection générique (**System.Collections.Generic**) et lui imposer un type.

Ici on va créer une collection de salariés.

```
Imports System.Collections
Public Class LesSalariésClasse
     Private maCol As System.Collections.Generic.List(Of Salarié)

     Public Sub New()
          maCol = New System.Collections.Generic.List(Of Salarié)    'cela crée une Collection de
 salariés
     End Sub


     Public Function Add(ByVal Sal As Salarié)
          maCol.Add(Sal)
          Add = Sal
     End Function
     Public ReadOnly Property Item(ByVal lIndex As Integer) As Salarié
          Get
               Item = maCol.Item(lIndex)  'On a directement un Objet Salarié: pas besoin de CType
          End Get
     End Property
….
End Class
```

On peut aussi créer des **Stack(Of…) Queue(Of…), Dictionnary(Of…) SortedList(Of…)** …

**Cette méthode utilisant une collection d'objets complètement encapsulés est une bonne méthode.**

## XV-G-2-b-v - Conclusion

**Si une classe doit contenir des objets**

- Si la classe a besoin d'utiliser et d'exposer **un petit nombre d'objets,** implémentez chaque objet en tant que propriété.
- Si **le nombre d'objets contenus est grand** (et que ce nombre n'est pas connu ou pas fixe) implémentez une propriété de **collection.**

Si vous ne voulez pas que des applications clientes puissent utiliser les objets contenus, définissez ces objets comme **Friend** ou **Private.**

## XV-H - Conservation (sauvegarde) d'objet, sérialisation


**Quand un objet est détruit (fin de programme), les valeurs de ses attributs (les variables) sont perdues !!**

**Si les valeurs de l'objet changent et doivent être retrouvées lors d'une utilisation ultérieure du programme, il faut les enregistrer.**

**On pourrait enregistrer chaque attribut dans un fichier séquentiel (FileOpen puis Print…).**

**On peut aussi utiliser la sérialisation.**

### XV-H-1 - La Sérialisation

La sérialisation est le processus de conversion d'un objet ou d'un groupe d'objets en séquence linéaire d'octets pour stockage ou transmission à un autre emplacement. La désérialisation est le processus consistant à accepter des informations stockées et à recréer des objets à partir de celles-ci.

**La sérialisation consiste donc à stocker les valeurs des attributs d'une instance d'un objet dans un fichier qui peut être au format binaire, xml ou Soap.**

- La **sérialisation binaire** concerne les **champs publics et privés** de l'objet et **le nom de la classe, y compris l'assembly** contenant la classe.
- La **sérialisation XML** ne sérialise **que les champs publics et les valeurs des propriétés d'un objet (si elles ne sont pas en lecture seule)** dans un flux XML. La sérialisation n'inclut pas d'informations de type.

Lors de la sérialisation, les champs et propriétés sont convertis en un flux d'octets, qui est alors écrit dans un flux de données enregistré sur le disque ou envoyé sur Internet.

Lorsque l'objet est ensuite désérialisé, le flux de données venant d'un fichier donne un flux d'octets qui donne une valeur aux champs et propriétés de l'objet, on obtient un objet identique à l'objet d'origine.

Vous pouvez aussi sérialiser un objet et le transporter sur Internet entre un client et un serveur à l'aide du protocole HTTP. À l'autre extrémité, la désérialisation reconstruit l'objet à partir du flux.

### XV-H-2 - Exemple 1 : Sérialisation binaire

Créons une mini Classe :

```
<Serializable()> Public Class Compta
Public Total As Double
Public Taux As Double
```

```
End Class
```

Notons que pour que la classe soit sérialisable, il faut ajouter :

```
<Serializable()>.
```

L'attribut Serializable indique donc au compilateur que tout ce que contient la classe peut être conservé dans un fichier.

- L'attribut **NonSerialized** peut être utilisé pour marquer les membres de la classe qui ne doivent pas être conservés.
- Pour empêcher la sérialisation d'un membre Customer par exemple :

```
<NonSerialized()> Public Customer As String
```

## XV-H-3 - Sérialisation

Dans le corps du programme, il faut mettre :

```
Imports System.IO
Imports System.Runtime.Serialization.Formatters.binary
```

Dans ce cas, vous utilisez un flux de sortie et un formateur binaire pour enregistrer l'objet dans un format binaire.

Dans l'entête du module créons un objet MyCompta :

```
Private myCompta As New Compta
```

Donnons des valeurs à ses membres :

```
myCompta.Taux = 2
myCompta.Total = 100
```

Si on quitte le programme, les valeurs sont perdues !!! On va donc les enregistrer dans un fichier "compta.bin" :

```
Dim myFileStream As Stream = File.Create("Compta.bin")
Dim serializer As New BinaryFormatter
serializer.Serialize(myFileStream, myCompta)
myFileStream.Close()
```

Et voilà un fichier compta.bin a été créé sur le disque, il contient :

```
Bin=  &#255;&#255;&#255;&#255; Kserialisation, Version=1.0.1994.38183, Culture=neutral,
 PublicKeyToken=null serialisation.MaClasse
Total Taux Y@ @
```

On a bien enregistré les valeurs des variables d'une instance dans un fichier.

## XV-H-4 - Désérialisation

Lors de la prochaine utilisation du logiciel, on crée de nouveau une instance de Compta :

```
Private myCompta As New Compta
```

Il faut ensuite 'récupérer' les valeurs de l'instance :

**Dans Form1_Load** par exemple :

```
Private Sub Form1_Load()
    If File.Exists("Compta.bin") Then
        Dim myFileStream As Stream = File.OpenRead("Compta.bin")
        Dim deserializer As New BinaryFormatter()
        myCompta = CType(deserializer.Deserialize(myFileStream), Compta)
        myFileStream.Close()
    End If
End Sub
```

À noter que vous devez d'abord vérifier que le fichier existe. S'il existe, créez une classe **Stream** pour lire le fichier binaire et une classe BinaryFormatter pour convertir le fichier. La méthode **CType** est utilisée pour la conversion du type d'objet Stream en type Compta.

Ça marche, on retrouve bien **MyCompta.Taux=2**

Bien sûr, si on sérialise une nouvelle fois, cela écrase le précédent fichier.

## XV-H-5 - Exemple 2 : Sérialisation XML

Pour les applications Web ou les services Web XML, vous souhaiterez peut-être conserver l'objet dans un fichier XML à l'aide d'un format SOAP, ce qui facilite le partage de l'objet.

il faut charger dans les références la dll .Net

```
System.Runtime.Serialization.Formatters.Soap.dll
Ensuite Imports System.Runtime.Serialization.Formatters.Soap
Dim deserializer As New SoapFormatter
```

Remplacez "SavedCompta.bin" par "SavedCompta.xml".

Cela donne :

```
Imports System.IO
Imports System.Runtime.Serialization.Formatters.Soap
Private MyCompta As New MaClasse
```

Sérialisation :

```
MyCompta.Taux = 3
MyCompta.Total = 100
Dim myFileStream As Stream = File.Create("SaveCompta.xml")
Dim serializer As New SoapFormatter
serializer.Serialize(myFileStream, MyCompta)
myFileStream.Close()
```

Déserialisation :

```
Dim myFileStream As Stream = File.OpenRead("saveCompta.bin")
Dim deserializer As New soapFormatter
MyCompta = CType(deserializer.Deserialize(myFileStream), MaClasse)
MsgBox(MyCompta.Taux.ToString)
myFileStream.Close()
```

Si on regarde le fichier SavedCompta.xml (il est dans le répertoire bin) on voit que c'est du XML :

```
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-ENV="http://
schemas.xmlsoap.org/soap/envelope/"
xmlns:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0" SOAP-ENV:encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<a1:MaClasse id="ref-1" xmlns:a1="http://schemas.microsoft.com/clr/nsassem/serialisation/
serialisation%2C%20Version%3D1.0.1995.
30938%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">
<Total>100</Total>
<Taux>3</Taux>
</a1:MaClasse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

On se rend compte que la sérialisation binaire produit un fichier plus petit.

## XV-H-6 - Exemple 3 : Sérialisation d'une collection

On peut sérialiser un objet, on peut donc sérialiser toutes sortes d'objets (dit sérialisables), une image, une collection, un tableau…

Une collection est un objet, pour enregistrer son contenu, on peut donc le sérialiser :

```
Imports System.IO
Imports System.Collections
Imports System.Runtime.Serialization.Formatters.Binary
Imports System.Runtime.Serialization


Module MonModule

' Creation d'une hashtable contenant des noms et adresses.
        Public addresses As New Hashtable
        addresses.Add("Phil", "12 grand rue,69872")
        addresses.Add("Bob", "98 petite rue, 196")
        addresses.Add("Marie", "BP 89, Paris, 75200")

    Sub Serialisation()


        ' Pour serialiser la hashtable (et les clé/valeur),
        Dim fs As New FileStream("MesAdresses.dat", FileMode.Create)
        Dim formatter As New BinaryFormatter
        Try
            formatter.Serialize(fs, addresses)
        Catch e As SerializationException
            Console.WriteLine("Echec serialisation. Cause: " &amp; e.Message)
            Throw
        Finally
            fs.Close()
        End Try
    End Sub



    Sub Deserialisation()
        ' Declaration de la HashTable.
        Dim addresses As Hashtable = Nothing
        Dim fs As New FileStream("DataFile.dat", FileMode.Open)
        Try
            Dim formatter As New BinaryFormatter

            addresses = DirectCast(formatter.Deserialize(fs), Hashtable)
        Catch e As SerializationException
            Console.WriteLine("Echec de deserialisation. Cause: " &amp; e.Message)
            Throw
        Finally
            fs.Close()
```

```
        End Try

    End Sub
End Module
```

## XV-H-7 - Exemple 4 : Sérialisation d'un tableau

```
Private MyCompta(10) As String
MyCompta(1) = "3"
MyCompta(2) = "100"
```

Sérialisation :

```
Dim myFileStream As Stream = File.Create("SaveCompta.xml")
Dim serializer As New SoapFormatter
serializer.Serialize(myFileStream, MyCompta)
myFileStream.Close()
```

Désérialisation :

```
Dim myFileStream As Stream = File.OpenRead("saveCompta.xml")
Dim deserializer As New soapFormatter
MyCompta = DirectCast(deserializer.Deserialize(myFileStream), String())
MsgBox(MyCompta(1).ToString)
myFileStream.Close()
```

Vous avez compris. Seule difficulté : le caste en String().

Bien sur, cela marche avec un tableau à plusieurs dimensions. Voyons les lignes à modifier

```
Private MyCompta(10,10) As String
MyCompta(1,1) = "3"
```

Dans la désérialisation:

```
MyCompta = DirectCast(deserializer.Deserialize(myFileStream), String(,))
```

## XV-H-8 - Exemple 5 : Sérialisation d'une collection générique

Ici nous enregistrons les données dans un fichier XML nommé "Meslivres.Xml" (il sera dans le répertoire bin/Debug lors de la conception, et dans le répertoire de l'exécutable si on installe le logiciel).

Les Sub SaveData et LoadData ont en paramètre un type de collection génériс list(Of ClasseLivre). C'est une collection d'objets typés ClasseLivre. Ce paramètre est passé avec ByRef.

(Pour l'exemple complet voir le chapitre architecture.)

```
Imports System.Xml.Serialization
Imports System.IO

Public Class AccesAuxDonnees

Public Sub SaveData(ByVal list As Collections.Generic.List(Of ClasseLivre))
' Déclaration
Dim serialXML As Xml.Serialization.XmlSerializer = Nothing
Dim streamIO As StreamWriter = Nothing
Try
serialXML = New Xml.Serialization.XmlSerializer(GetType(Collections.Generic.List(Of
 ClasseLivre)))
```

```
' Ouverture d'un flux en écriture sur le fichier XML des contacts
streamIO = New StreamWriter("Meslivres.Xml")
' Sérialisation de la liste des contacts
serialXML.Serialize(streamIO, list)
Catch ex As Exception
' Propagrer l'exception
Throw ex
Finally
' En cas d'erreur, n'oubliez pas de fermer le flux en écriture si ce dernier est toujours ouvert
If streamIO IsNot Nothing Then
streamIO.Close()
End If
End Try
End Sub

Public Sub LoadData(ByRef list As Collections.Generic.List(Of ClasseLivre))
' Déclaration
Dim streamIO As StreamReader = Nothing
Dim serialXML As Xml.Serialization.XmlSerializer = Nothing
Try
' Tester l'existence du fichier
If System.IO.File.Exists("Meslivres.Xml") = True Then
serialXML = New Xml.Serialization.XmlSerializer(GetType(Collections.Generic.List(Of
 ClasseLivre)))
' Ouverture d'un flux en lecture sur le fichier XML des contacts
streamIO = New StreamReader("Meslivres.Xml")
' Désérialisation de la liste des contacts
list = CType(serialXML.Deserialize(streamIO), Collections.Generic.List(Of ClasseLivre))
End If
Catch ex As Exception
' Propagrer l'exception
Throw ex
Finally
' En cas d'erreur, n'oubliez pas de fermer le flux en lecture si ce dernier est toujours ouvert
If streamIO IsNot Nothing Then
streamIO.Close()
End If
End Try
End Sub
End Class
```

Voila ce que donne le fichier Xml :

```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfClasseLivre xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<ClasseLivre>
<Titre>Livre1</Titre>
<Auteur>Auteur1</Auteur>
</ClasseLivre>
<ClasseLivre>
<Titre>Livre2</Titre>
<Auteur>Auteur2</Auteur>
</ClasseLivre>
<ClasseLivre>
<Titre>Titre3</Titre>
<Auteur>Auteur3</Auteur>
</ClasseLivre>
</ArrayOfClasseLivre>
```

# XV-I - Surcharge

**Quand on utilise une méthode avec des paramètres, il y a parfois possibilité d'utiliser, avec la même méthode, un nombre différent de paramètres ou des paramètres de nature différente : on appelle cela surcharger la méthode.**

Chaque manière d'écrire les paramètres s'appelle une **signature.**

Exemple

Voici une fenêtre MessageBox : pour ouvrir une fenêtre MessageBox il y a 12 signatures, en voici 2 :

```
Reponse= MessageBox.show(TexteAAfficher,Titre, TypeBouton etIcone, BoutonParDéfaut)
```

Ici on donne 4 paramètres.

```
Reponse= MessageBox.show(TexteAAfficher)
```

Ici 1 seul paramètre.

On voit qu'on peut appeler la méthode MessageBox.Show avec un nombre différent de paramètres.

Comme on ne peut pas connaitre toutes les signatures, VB nous aide.

Si on tape **R= MessageBox.show,** VB affiche dans un cadre une signature, de petites flèches permettent de faire défiler toutes les autres signatures.

Quand on crée une Classe, il est bien sûr possible d'écrire une property ou un constructeur qui accepte la surcharge.

## XV-I-1 - Surcharge en VB 2003

1 On peut surcharger un constructeur.
Pour cela il suffit de rajouter autant de procédures New que l'on veut avec pour chacune un nombre de paramètres différents (signatures différentes).
Exemple : on peut surcharger un constructeur :

```
Class Figure
Sub New()
    Bla Bla
End Sub

Sub New( ByVal X As Integer, ByVal Y As Integer)
     Blabla
End Sub.
End Class
```

On peut donc instancier la classe correspondante de 2 manières :

```
Dim A As New Figure    'Constructeur par défaut
```

ou

```
Dim A As New Figure(100,150)
```

2 On peut surcharger une property.
Pour cela il suffit de rajouter des procédures Property ayant le même nom de méthode avec pour chacune un nombre de paramètres différents (signatures différentes). On peut ajouter Overloads, mais c'est facultatif.
Exemple : surchargeons un membre :

```
Class Figure
Public Overloads Property Calcul()
    Bla Bla
End Sub

Public Overloads Property Calcul( ByVal X As Integer, ByVal Y As Integer)
     Blabla
End Sub.
End Class
```

*C'est un bon exemple de polymorphisme.*

## XV-I-2 - Surcharge en VB 2005 : nouveautés

- Exemple : surchargeons l'opérateur +

```
Public Structure height
    …
    Public Shared Operator +(ByVal h1 As height, ByVal h2 As height)As height
        Return New height(h1.feet + h2.feet, h1.inches + h2.inches)
    End Operator
End Structure
```

La routine doit être Shared, de plus si on surcharge certains opérateurs, il faut aussi surcharger leur inverse : si on surcharge '>' , il faut surcharger '<'.

**Surcharge de IsTrue, IsFalse CType**

Si on teste un boolean, il a la valeur True ou False.

Si par contre je crée une classe nommée 'Personne', je peux définir comment une instance sera considérée comme égale à True. Il faut surcharger l'opérateur IsTrue en lui indiquant dans quelle condition l'instance sera considérée comme =True.

Exemple

J'ai une instance **e** de type Personne, si e.Present =True, dans ce cas je veux que e soit considéré comme True, il faut écrire dans la Classe 'personne' :

```
Public Shared Operator IsTrue(ByVal e As Personne) As Boolean
If e Is Nothing Then
    Return False
Else
    Return e.Present
End If
End Operator
```

Pour définir l'opérateur IsFalse, c'est simple : c'est Not e

```
Public Shared Operator IsFalse(ByVal e As Personne) As Boolean
    Return Not e
End Operator
```

Ensuite je pourrai utiliser des instructions de la forme :

```
If e then…
```

### Surcharge de CType

Je peux définir dans une classe comment **CType** va fonctionner.

Pour cela dans la classe Personne, je vais définir les 3 possibilités :

```
Public Shared Widening Operator CType(ByVal e As Personne) As String
Return e.Nom + " " + e.Prenom
End Operator

Public Shared Widening Operator CType(ByVal  e As Personne) As Date
If e Is Nothing Then
    Return Nothing
Else
    Return e.DateDeNaissance
End If
End Operator

Public Shared Widening Operator CType(ByVal  e As Personne) As Boolean
    If e Is Nothing Then Return False Else Return e.Present
End Operator
```

Ainsi :

**CType(UnePersonne,String)** retourne Nom Prenom ;

**CType(UnePersonne,Date)** retourne la date de naissance ;

**CType(UnePersonne,Boolean)** retourne True ou False.

Les exemples sont des surcharges, car le type des paramètres est modifié.

## XV-J - Structure de programme : programmation à trois couches


### XV-J-1 - Introduction

Les programmes les plus fréquemment développés sont ceux utilisant une **interface utilisateur** permettant de travailler sur un ensemble de données, par exemple les clients d'une entreprise. Il faut pouvoir ajouter, supprimer, modifier les clients, en afficher la liste. Une **base de données** permet de stocker les données.

Il y a quelques années, dans l'interface utilisateur, du code lisait, modifiait la base de données. Très vite, sur un projet important cette approche, non architecturée, montrait ses limites.

Aussi très rapidement, en programmation procédurale, le besoin de travailler sur des couches est apparu.

**Pour afficher la liste des clients de l'entreprise :**

- **la couche présentation :** une windowsForm affichait la liste ; elle faisait appel à
- **la couche métier :** une routine ChargeListe située dans un module standard sélectionnait les clients, elle faisait appel à
- **la couche données** qui lisait la base de données.

Cela a été formalisé en programmation objet.

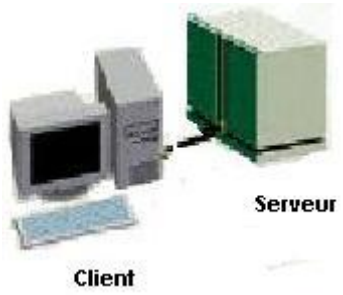
À noter que si les diverses couches sont sur les ordinateurs différents, on parle d'**architecture distribuée.**

## XV-J-2 - Architecture n-tiers

De l'anglais **tier** signifiant **étage** ou **niveau**.

Architecture logicielle impliquant plusieurs composants, chaque composant étant le client d'un et le serveur d'un autre.

Le cas le plus simple de ce type d'architecture est le **modèle Client/Serveur** qui est en **2-tiers.**

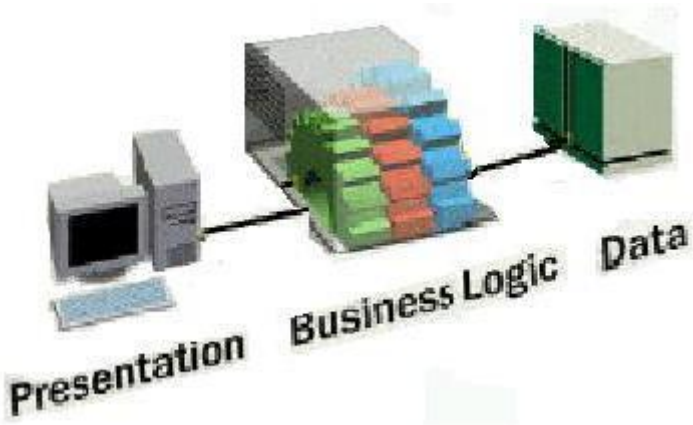



***Tier 'Client'*** - Contient l'interface utilisateur, adresse des requêtes au serveur.

***Tier 'Serveur'*** - Contient la base de données. Reçoit les requêtes, renvoie des données.

Dans les modèles **3-tiers** et plus, il existe des composants intermédiaires qu'on appelle également middleware.

Un exemple typique d'un système **3-tiers** est le suivant :



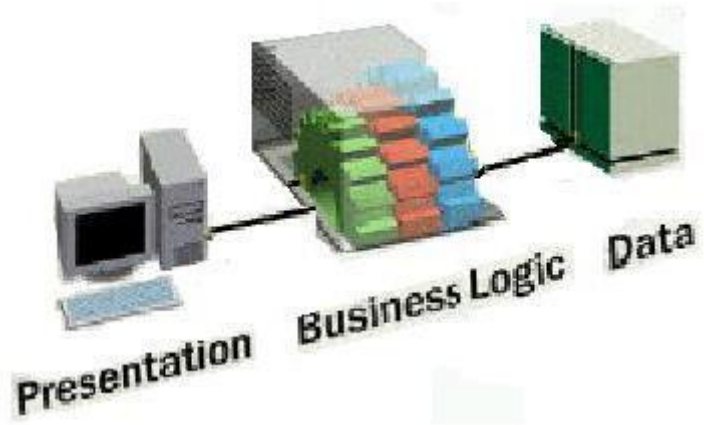

- **Tier de présentation** - c'est principalement l'interface utilisateur. contient les différents types de clients, léger (Web, ASP, JSP) ou lourd (Swing, WinForm).
- **Tier des règles de gestion :** couche métier : Business Logic.
- **Tier de base de données** - les programmes du deuxième tier font appel à ce dernier pour consulter ou mettre à jour les données relatives à l'entreprise.

Sur le même schéma, on peut ajouter des intermédiaires supplémentaires et obtenir une architecture 4, 5…, n-tiers.

**Un exemple d'architecture à 5 couches logicielles :**

- **Présentation :** contient tous les composants graphiques du module composant l'interface homme-machine (fenêtres, contrôles utilisateur…) avec le code propre à l'affichage de leur représentation et de leur contenu. Cette couche ne peut référencer que les couches "Référence" et "Application". Mettre le moins de code possible dans cette couche ;
- **Application :** contient tous les contrôleurs de cas d'utilisation du module. Cette couche assure le lien entre les composants graphiques et les composants métier. Cette couche ne peut référencer que les couches "Métier", "Persistance" et "Référence" ;
- **Métier :** contient tous les composants métier dont le module a la responsabilité. Ces composants métier ont en charge la gestion du cycle de vie des objets métier gérés par le module. Cette couche ne peut référencer que les couches "Référence" et "Persistance" ;
- **Référence :** cette couche contient les objets de données pures qui transitent entre toutes les autres couches. Ces objets sont aussi parfois nommés DataValues ou DataObjects.
- **Persistance :** contient les composants assurant le mapping entre les objets définis dans la couche Métier et les composants de stockage définis dans la base de données. Cette couche ne peut référencer que les couches "Référence" et la base de données. Concrètement, il s'agit de la seule couche ayant un lien avec la base de données.

## XV-J-3 - Architecture 3 tiers




- L'architecture 3-tiers (de l'anglais tier signifiant étage ou niveau) vise à séparer très nettement trois couches logicielles au sein d'une même application ou système, à modéliser et présenter cette application comme un empilement de trois couches, étages, niveaux ou strates dont le rôle est clairement défini :
- la **présentation** des données : correspondant à l'affichage, la restitution sur le poste de travail, le dialogue avec l'utilisateur,
- le traitement métier des données : correspondant à la mise en œuvre de l'ensemble des règles de gestion et de la logique applicative, c'est à ce niveau que se trouvent toutes les règles de gestion, et toute l'intelligence de la démarche
- et enfin l'**accès aux données** persistantes (persistancy en anglais) : correspondant aux données qui sont destinées à être conservées sur la durée.

**Relation entre les couches : l**es services d'une couche sont mis à disposition de la couche supérieure.

## XV-J-4 - Exemple 1 : Ma bibliothèque (en écrivant du code)

Créons une application permettant de saisir, d'enregistrer, de voir des fiches 'Livre' :


Il faut créer des objets 'Livre' puis un objet 'Livres' contenant une collection de tous les livres (**Couche métier**). Cette collection doit pouvoir être enregistrée puis lue sur disque (**Couche d'accès aux données**). Enfin on doit pouvoir afficher le détail d'un livre (**Interface utilisateur**).

**Couche métier :**

- Classe "ClasseLivre" : entité métier, classe permettant d'instancier un livre avec comme propriété : Titre, auteur…
- Classe "ClasseLivres" (avec un 's'): classe comprenant une collection d'objets "Livre". Elle possède des méthodes permettant d'ajouter un livre, de récupérer un livre dans la collection, de passer au précédent, au suivant…

**Couche d'accès aux données :**

- LoadData pour lire la collection de livres à partir d'un fichier (xml dans notre exemple).
- SaveData pour enregistrer la collection de livres dans un fichier (xml dans notre exemple).

**Interfaces graphiques :**

- un Formulaire principal permettant de saisir un nouveau livre et de faire défiler les livres.

## XV-J-4-a - Couche métier

Créons la ClasseLivre", elle doit contenir :

2 property public : 'Titre' et 'Auteur' ;

2 variables private : m_titre et m_Auteur.

Cela permettra d'instancier un objet livre et d'utiliser ses propriétés.

Exemple

Dim livre As New ClasseLivre : livre.Nom= "Cours VB"

Voici la classe :

```
Public Class ClasseLivre
```

```
Private m_Titre As String
Private m_Auteur As String

' Propriété Titre
Public Property Titre() As String
Get
Return m_Titre
End Get
Set(ByVal value As String)
m_Titre = value
End Set
End Property

' Propriété Auteur
Public Property Auteur() As String
Get
Return m_Auteur
End Get
Set(ByVal value As String)
m_Auteur = value
End Set
End Property

End Class
```

Créons la ClasseLivres" (avec un 's'), elle doit contenir :

une collection nommé ListLivre d'objet génerics : Collection.Generic.List (Of Classelivre).

C'est une collection typée, elle ne peut contenir que des 'Livre'.

On instancie aussi un objet ad d'accès aux données.

La méthode public LivrePointé retourne le livre en cours, les méthodes FirstLivre, LastLivre, NextLivre, PreviousLivre permettent de se déplacer dans les livres. On peut ajouter un livre avec AddLivre.

Une variable nommée m_numéro sert dans la classe de "pointeur" du livre courant.

Pour enregistrer ou charger la collection de livres sur disque, il y a les 2 méthodes SaveData et LoadData. Elle appelle des méthodes de la couche de données.

Quand cette classe est instanciée, elle charge les données (la procédure New appelle LoadData).

```
Imports System.Collections.Generic

Public Class ClasseLivres
Private ListLivre As New List(Of ClasseLivre) 'Collection de Livre
Private m_numero As Integer
Private ad As New AccesAuxDonnees

'Constructeur, charge la collection
Public Sub New()
Me.LoadData()
End Sub

Public Sub LoadData()
ad.LoadData(ListLivre)
m_numero = 0
End Sub

Public Sub SaveData()
ad.SaveData(ListLivre)
End Sub

'Retourne le livre courant
```

```
Public Function LivrePointé() As ClasseLivre
If ListLivre.Count <> 0 Then
If m_numero < ListLivre.Count Then
Return ListLivre.Item(m_numero)
Else
Return Nothing
End If
Else
Return Nothing
End If
End Function

'Ajouter un livre
Public Sub AddLivre(ByVal l As ClasseLivre)
ListLivre.Add(l)
End Sub

'Effacer le livre courant
Public Sub RemoveLivre()
ListLivre.RemoveAt(m_numero)
End Sub

'Mettre à jour un livre
Public Sub UpdateLivre(ByVal l As ClasseLivre)
ListLivre.Item(m_numero) = l
End Sub

'Livre suivant
Public Sub NextLivre()
If m_numero < ListLivre.Count Then
m_numero += 1
End If
End Sub

'Livre précédent
Public Sub PreviousLivre()
If m_numero > 0 Then
m_numero -= 1
End If
End Sub

'Premier Livre
Public Sub FirstLivre()
m_numero = 0
End Sub

'Dernier livre
Public Sub LastLivre()
m_numero = ListLivre.Count - 1
End Sub
End Class
```

Il aurait été plus élégant d'instancier l'objet d'accès aux données dès le début de la classe comme cela :

```
Public Class ClasseLivres
Private ListLivre As New List(Of ClasseLivre) 'Collection de Livre
Private m_numero As Integer

Dim ad As New AccesAuxDonnees
```

## XV-J-4-b - Couche d'accès aux données

Ici nous enregistrons les données dans un fichier XML nommé "Meslivres.Xml" (il sera dans le répertoire bin/Debug lors de la conception, et dans le répertoire de l'exécutable si on installe le logiciel). On utilise la sérialisation et les Stream. (Voir chapitre sur la sérialization pour plus de détails.)

Les Sub ont un paramètre : la collection de ClasseLivre. Ce paramètre est passé avec ByRef :

```
Imports System.Xml.Serialization
Imports System.IO

Public Class AccesAuxDonnees

Public Sub SaveData(ByVal list As Collections.Generic.List(Of ClasseLivre))
' Déclaration
Dim serialXML As Xml.Serialization.XmlSerializer = Nothing
Dim streamIO As StreamWriter = Nothing
Try
serialXML = New Xml.Serialization.XmlSerializer(GetType(Collections.Generic.List(Of
 ClasseLivre)))
' Ouverture d'un flux en écriture sur le fichier XML
streamIO = New StreamWriter("Meslivres.Xml")
' Sérialisation de la liste des contacts
serialXML.Serialize(streamIO, list)
Catch ex As Exception
' Propager l'exception
Throw ex
Finally
' En cas d'erreur, n'oubliez pas de fermer le flux en écriture si ce dernier est toujours ouvert
If streamIO IsNot Nothing Then
streamIO.Close()
End If
End Try
End Sub

Public Sub LoadData(ByRef list As Collections.Generic.List(Of ClasseLivre))
' Déclaration
Dim streamIO As StreamReader = Nothing
Dim serialXML As Xml.Serialization.XmlSerializer = Nothing
Try
' Tester l'existence du fichier
If System.IO.File.Exists("Meslivres.Xml") = True Then
serialXML = New Xml.Serialization.XmlSerializer(GetType(Collections.Generic.List(Of
 ClasseLivre)))
' Ouverture d'un flux en lecture sur le fichier XML des contacts
streamIO = New StreamReader("Meslivres.Xml")
' Désérialisation de la liste
list = CType(serialXML.Deserialize(streamIO), Collections.Generic.List(Of ClasseLivre))
End If
Catch ex As Exception
' Propager l'exception
Throw ex
Finally
' En cas d'erreur, n'oubliez pas de fermer le flux en lecture si ce dernier est toujours ouvert
If streamIO IsNot Nothing Then
streamIO.Close()
End If
End Try
End Sub
End Class
```

# XV-J-4-c - Couche de présentation : interface graphique


On instancie un objet Livres, cet objet contient la collection de livres.

Il suffit ensuite d'utiliser Livres.LivrePointé qui est le livre en cours, Livres.NextLivre() pour passer au livre suivant, Livres.PreviousLivre pour passer au précédent.

Quand on clique sur 'Enregister livre' on effectue Livres.AddLivre.

Quand on clique sur 'Enregister tous les livres' on effectue Livres.SaveData (qui fait appel à la couche de données).

Deux petites Sub LireLivre et AfficheLivre permettent de remplir ou de lire les TextBox à partir d'un objet livre.

```
Public Class InterfaceGraphique

Dim Livres As New ClasseLivres
Dim iFlagNewLivre As Integer

Sub LireLivre(ByVal l As ClasseLivre)
l.Titre = TextBoxTitre.Text
l.Auteur = TextBoxAuteur.Text
End Sub

Sub AfficheLivre(ByVal l As ClasseLivre)
If l IsNot Nothing Then
TextBoxTitre.Text = l.Titre
TextBoxAuteur.Text = l.Auteur
End If
End Sub

Private Sub ButtonNext_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles ButtonNext.Click
Livres.NextLivre()
AfficheLivre(Livres.LivrePointé)
iFlagNewLivre = False
End Sub


Private Sub ButtonPrevious_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles ButtonPrevious.Click
Livres.PreviousLivre()
AfficheLivre(Livres.LivrePointé)
iFlagNewLivre = False
End Sub

Private Sub ButtonNew_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
```

```
Handles ButtonNew.Click
TextBoxTitre.Text = ""
TextBoxAuteur.Text = ""
iFlagNewLivre = True
End Sub

Private Sub ButtonEnregister_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles ButtonEnregister.Click
Dim NouveauLivre As New ClasseLivre
If iFlagNewLivre = True Then
  LireLivre(NouveauLivre)
  Livres.AddLivre(NouveauLivre)
  Livres.LastLivre()
Else
  LireLivre(NouveauLivre)
  Livres.UpdateLivre(NouveauLivre)
End If
iFlagNewLivre = False
End Sub

Private Sub ButtonEnregistertous_Click(ByVal sender As System.Object, ByVal e As
 System.EventArgs)
_Handles ButtonEnregistertous.Click
    Livres.SaveData()
End Sub
End Class
```

## XV-J-5 - Exemple 2 : Bibliothèque (avec Binding et génération automatique de l'IU)

C'est le même programme, mais on va laisser le soin à VB de générer automatiquement l'interface utilisateur et les liens (Binding) entre cette interface et les données.

La couche d'accès aux données est la même.

On a une Form vide.

## XV-J-5-a - Couche métier

**ClasseLivre** est identique, par contre **ClassesLivres** est plus simple : elle contient uniquement la collection générique **ListLivre** contenant les livres.

(Et SaveData et Load Data.)

```
Imports System.Collections.Generic
Public Class ClasseLivre

Private m_Titre As String
Private m_Auteur As String
' Propriétés Titre
Public Property Titre() As String
Get
Return m_Titre
End Get
Set(ByVal value As String)
m_Titre = value
End Set
End Property

' Propriétés Auteur
Public Property Auteur() As String
Get
Return m_Auteur
End Get
Set(ByVal value As String)
```

```
m_Auteur = value
End Set
End Property

Public Function GetTitre() As String
Return m_Titre
End Function
End Class

Public Class ClasseLivres
Public ListLivre As New Collections.Generic.List(Of ClasseLivre)
Private ad As New AccesAuxDonnees

Public Sub LoadData()
ad.LoadData(ListLivre)
End Sub

Public Sub SaveData()
ad.SaveData(ListLivre)
End Sub

End Class
```

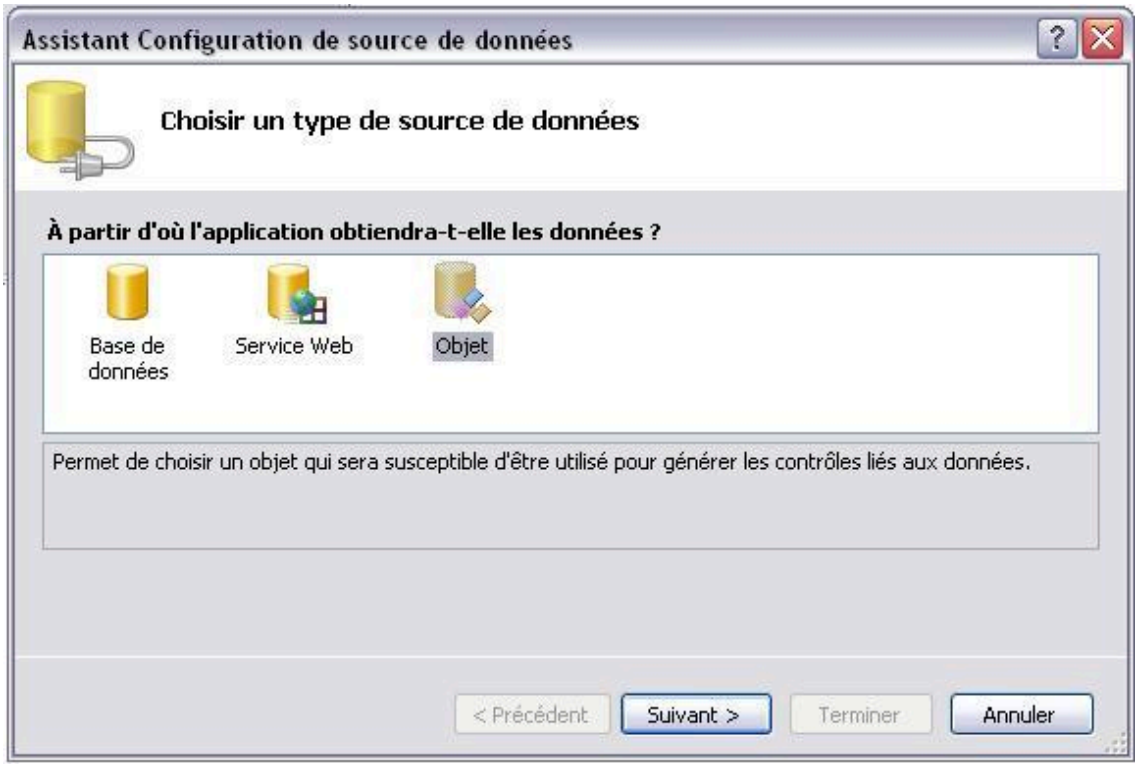
Ici ListLivre est Public.

## XV-J-5-b - Création de la source de données

Il faut ensuite créer une source de données.

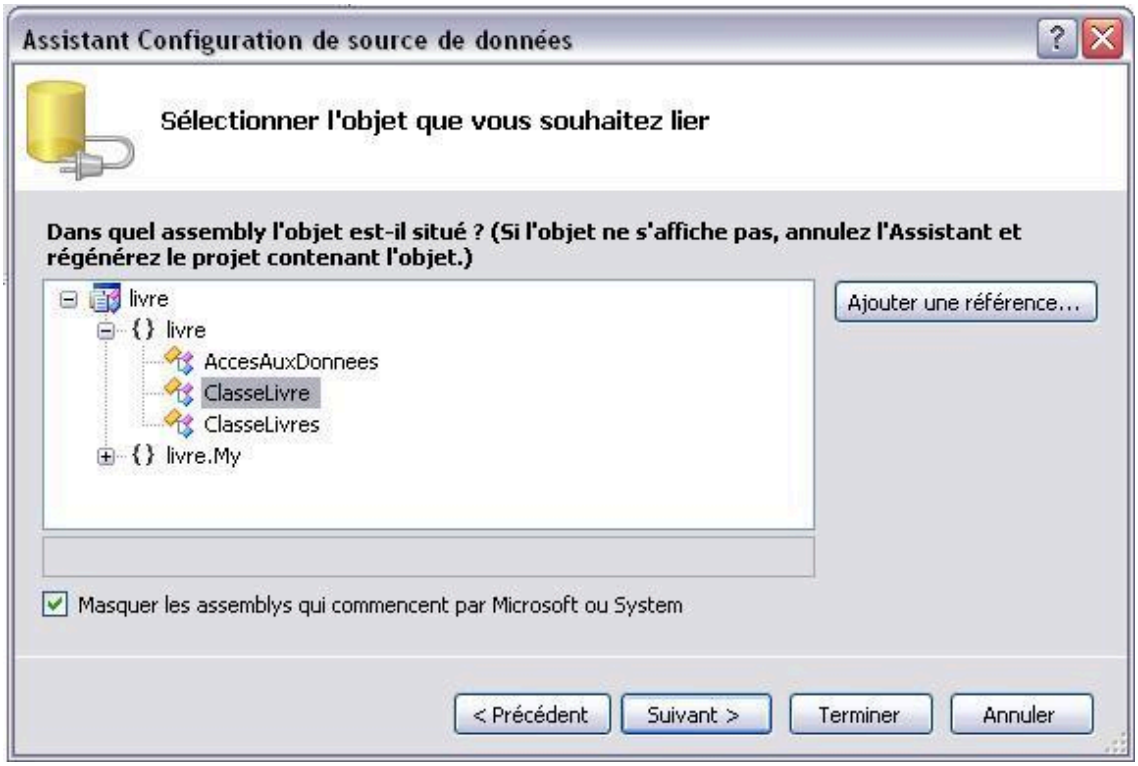
Comme on affiche des livres, la source c'est la ClasseLivre.


Menu 'Données'=> 'Ajouter une nouvelle source de données'




Ici la source de données n'est pas une base de données, mais un Objet :

On clique sur 'Objet' puis bouton 'Suivant'.




On déroule livre, qui est le nom de la solution, puis on clique sur 'ClasseLivre' et sur le bouton suivant.

Puis OK, la source de données est créée.

# XV-J-5-c - Génération automatique de l'interface utilisateur

Visual Studio dispose d'une fenêtre 'Sources de données' depuis laquelle vous pouvez faire glisser des éléments jusqu'à un formulaire pour créer automatiquement des contrôles liés aux données qui affichent des données.

Afficher les sources de données.

Menu 'Données' puis 'Afficher les sources de données'


Il apparait à droite la fenêtre 'Sources de données'.

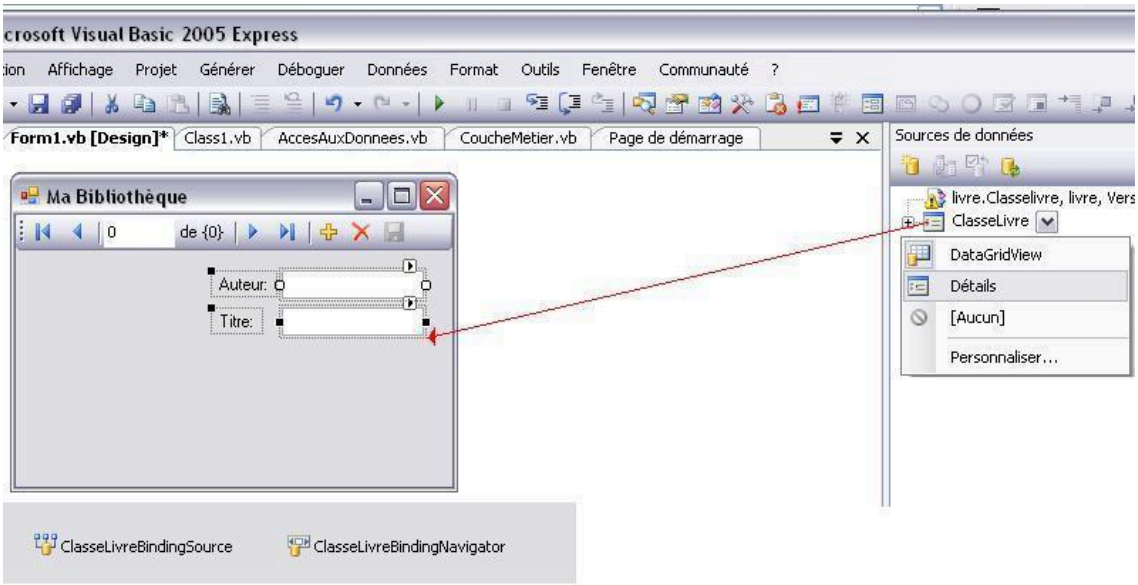
Dérouler 'livre' et cliquer sur 'ClasseLivre'.

Pour générer, non pas une grid mais des zones de saisie, dérouler la liste de ClasseLivre et cliquer sur 'Détails'.

Enfin faire un drag and drop à partir de ClasseLivre et déposer-le sur la form de gauche (qui est vide au départ).

**Miracle : il apparait automatiquement :**

- des zones textbox pour chaque properties de la classe avec un label devant ;
- une barre de navigation (tout est généré automatiquement : les bitmap des boutons dans les ressources Setting…) ;
- un composant BindingSource. (Il fait le lien entre l'interface et la source de données) ;
- un composant BindingNavigator. (Il gère la barre de navigation)




On voit bien en dessous les 2 composants qui ont été ajoutés.

## XV-J-5-d - Création du Binding

Maintenant, il faut indiquer la source de données, le Datasource du BindingSource : c'est la collection **MesLivres.ListLivre** (c'est pour cela que la collection ListLivre est Public) :

```
Public Class Form1
Dim MesLivres As New ClasseLivres

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles MyBase.Load
 ClasseLivreBindingSource.DataSource = MesLivres.ListLivre
End Sub

'On peut ajouter dans la form 2 boutons permettant la sauvegarde ou la lecture sur disque.
Private Sub SauveLivres_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles Button1.Click
    MesLivres.SaveData()
End Sub

Private Sub LoadLivre_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles Button2.Click
    MesLivres.LoadData()
    ClasseLivreBindingSource.DataSource = MesLivres.ListLivre
End Sub
End Class
```

P… !! ça marche !!

**Ici, les différentes parties du logiciel sont dans la même solution. Rien n'empêche de :**

- mettre les classes dans un même fichier .vb ;
- mettre les classes dans différents fichiers .vb ;
- créer les Classes dans une solution, la générer, inclure dans les références du projet principal les dll ;
- enfin si diverses parties sont sur différentes machines, on parle d'architecture distribuée.

## XV-K - Utilisation de Patron (motif de conception, Design Pattern)


Un **motif de conception** (design pattern) est un concept destiné à résoudre les problèmes récurrents en POO. En français on utilise aussi les termes **modèle de conception, patron de conception ou de Patron.**

Pour ne pas avoir à réinventer la roue, les patrons décrivent des solutions standards pour répondre à des problèmes de conception ou d'architecture. C'est une formalisation de bonnes pratiques, ce qui signifie qu'on privilégie les solutions éprouvées.

Il ne s'agit pas de fragments de code, mais d'une méthode de conception.

Les auteurs des principaux design patterns (il y en a 23), plus connus sous le nom de Gang Of Four, sont Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides. Les DP qu'ils ont conçus sont considérés comme la base de tous les autres.

## XV-K-1 - Singleton

Parfois, on a besoin d'une classe avec laquelle on veut **créer une instance et une seule.**

**Il faut créer une classe :**

- où il ne soit pas possible de créer plusieurs instances.
  New , le constructeur, doit-être toujours private : cela empêche d'écrire 'Dim myobjet As New MyClasse' et empêche donc d'instancier.
  On utilise donc :

```
Private Sub New()
```

- Où une méthode de classe permet de créer une instance et une seule. Une méthode de classe nommée habituellement 'GetInstance' peut être appelée (sans avoir à instancier) directement sur le nom de la classe.
  Shared Function getInstance() As sing.
  Cette fonction qui s'appelle toujours getInstance va servir à instancier une fois la variable Instance.

Shared instance As sing

Cette variable est la base du Singleton. Elle s'appelle Instance (par convention) elle est du même type que la class et contient l'instance unique.

```
Public Class sing
'Déclaration de l'instance Singleton
Shared instance As sing

Private Sub New()'Pas oublier de mettre Private
    MyBase.New()
End Sub
Shared Function getInstance() As sing 'Voici la méthode de classe

If IsNothing(instance) Then 'Si le singleton n'est pas créé, alors faut le créer une et une seule
    instance = New sing
End If
Return instance
End Function
….
… 'Ici les autres membres de la classe
End Class
```

Comment utiliser cette Classe ?

```
Dim t As sing =  sing.getInstance
```

**Remarque**

- Si on fait ensuite Dim t1 As sing = sing.getInstance c'est la même instance qui est retournée. On ne peut en créer qu'une…
- Si on écrit **Dim t As New sing :**cela plante.

On peut ajouter une protection contre les multithread trop rapides avec **SyncLock GetType(sing)** :

```
    Shared Function getInstance() As sing

        If IsNothing(instance) Then
            SyncLock GetType(sing)
                 If IsNothing(instance) Then
                    instance = New sing
                end if
            End SyncLock
        End If
        Return instance
    End Function
```

# XV-K-2 - Itérateur

Un itérateur est un objet qui permet de parcourir tous les éléments contenus dans un conteneur.

Dans la programmation procédurale, on utilise souvent une boucle (for next) avec un index, pour accéder séquentiellement à tous les éléments d'un tableau. Cette méthode se nomme *Indexation.*

En programmation objet, le but d'un itérateur est de permettre à son utilisateur de *parcourir* le conteneur, c'est-à-dire d'accéder séquentiellement à tous ses éléments, de *consulter l'élément courant*. L'itérateur permet d'isoler l'utilisateur de la structure interne du conteneur, parfois complexe.

**Les itérateurs ont certains avantages**

- Une simple boucle n'est pas adaptée à toutes les structures de données, en particulier celles qui n'ont de méthode d'accès à un élément quelconque ou celles à accès à un élément quelconque très lent.
- Les vrais itérateurs peuvent être écrits pour toutes sortes de structures de données : liste arbre liste chainée… s'il y a changement dans l'organisation de la structure de données, celui qui utilise l'itérateur n'a pas à s'en soucier.
- Un vrai itérateur peut implanter des conditions additionnelles sur l'accès aux éléments, par exemple pour "sauter ou ne pas sauter".
- Un vrai itérateur peut dans certains cas permettre que le conteneur soit modifié, sans être invalidé pour autant. Par exemple, après qu'un itérateur s'est positionné derrière le premier élément, il est possible d'insérer d'autres éléments au début du conteneur avec des résultats prévisibles. Avec un index, on aurait plus de problèmes, parce que la valeur de l'index devrait elle aussi être modifiée en conséquence.

En VB, il existe des itérateurs 'clé en main' pour les collections par exemple, mais ils n'ont pas tous les avantages décrits ci-dessus.

La Classe **System.Array** et celles qui implémentent l'interface **IEnumerable** possèdent 2 méthodes pour itérer :

- une boucle For Each pour parcourir tous les éléments de l'Array ;
- l'utilisation d'un énumérateur.

Voyons comment faire.

On peut aussi utiliser dans ce cas **Getenumerator** pour créer un **énumérateur :** il permet de lire du premier élément au dernier.

Dans l'énumérateur l'élément courant est **Current.**

Pour passer au suivant, on utilise **MoveNext.**

**Reset** réinitialise.

Initialement le pointeur est avant le premier élément, aussi avant d'utiliser **Current**, il faut faire **MoveNext.**

Attention, on ne peut que lire les données dans l'ordre, si on modifie la collection, il faut redémarrer la lecture.

```
' Créer un tableau
Dim myArr(4) As String
myArr(0) = "toto"
myArr(1) = "lulu"
myArr(2) = "bibi"
myArr(3) = "tata"

Dim i As Integer = 0
```

```
'Créer un énumérateur
Dim myEnumerator As System.Collections.IEnumerator = myArr.GetEnumerator()

'Afficher sur la console successivement les éléments du tableau
'en utilisant MoveNext et Current
While myEnumerator.MoveNext() And Not (myEnumerator.Current Is Nothing)
  Console.WriteLine("[{0}] {1}", i, myEnumerator.Current)
  i += 1
End While
```

## XV-L - Linq dans les Classes

Lire le chapitre Linq dans la partie langage VB.

Exemple de Microsoft

Créer 4 étudiants ayant un nom, prénom et des notes. (On les met dans une ArrayList.) Rechercher les étudiants dont la première note est >95.

```
' Création de la classe Student

Public Class Student

    Public FirstName As String

    Public LastName As String

    Public Scores As Integer()

End Class


 Sub Main()

 'Création de 4 étudiants avec nom prénom et note

 Dim student1 As New Student With {.FirstName = "Svetlana", .LastName = "Omelchenko", _
  .Scores = New Integer() {98, 92, 81, 60}}

 Dim student2 As New Student With {.FirstName = "Claire",  .LastName = "O'Donnell", _
  .Scores = New Integer() {75, 84, 91, 39}}

 Dim student3 As New Student With {.FirstName = "Cesar",  .LastName = "Garcia", _
  .Scores = New Integer() {97, 89, 85, 82}}

 Dim student4 As New Student With {.FirstName = "Sven",  .LastName = "Mortensen", _
  .Scores = New Integer() {88, 94, 65, 91}}


 'Création d'une ArrayList dans laquelle on met les 4 étudiants

Dim arrList As New ArrayList()

arrList.Add(student1)

arrList.Add(student2)

arrList.Add(student3)

arrList.Add(student4)


 ' Requête: rechercher les étudiants dont la première note est >95
```

```
Dim query = From student As Student In arrList _
 Where student.Scores(0) > 95 _
 Select student


'Affichage des résultats
For Each student As Student In query
  Console.WriteLine(student.LastName & ": " & student.Scores(0))
Next


 ' referme la console.
Console.WriteLine("Press any key to exit.")
Console.ReadKey()


End Sub
```

Module Output : Omelchenko : 98 Garcia : 97

# XVI - Un peu de théorie pour en déduire de bonnes pratiques

## XVI-A - Diverses sortes de programmation

**Programmation impérative.**

**Programmation structurée.**

**Programmation fonctionnelle.**

**Programmation procédurale.**

**Programmation événementielle.**

**Programmation défensive.**

**Programmation-objet.**

### XVI-A-1 - Programmation impérative

On en fait sans le savoir.

Le programmeur spécifie explicitement l'enchainement des instructions devant être exécutées :

fais ceci, puis cela ;

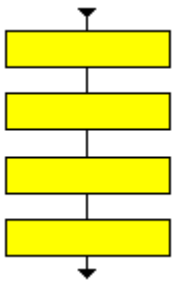
fais ceci, si cela est vrai ;

fais ceci, tant de fois.

Un programme **impératif** est composé de différentes instructions indiquant de manière explicite comment résoudre un problème.

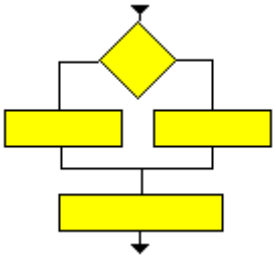
On l'a déjà vu, les instructions sont effectuées de manière séquentielle :

**Instruction1**
**Instruction2**
**Instruction3**
…



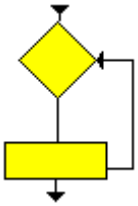
Il y a des choix :

**If… Then**



Il y a des boucles :

**For … next**



L'action de base dans la programmation impérative est l'affectation, c'est-à-dire la modification de la valeur associée à une variable du programme.

```
A=3
```

L'affectation va être soit effectuée en séquences successives, soit avec des choix, soit itérée selon les compositions définies par les algorithmes.

En programmation impérative, on travaille sur le modèle de la **machine de Turing**, avec une mémoire centrale et des instructions qui modifient son état grâce à des assignations successives.

Le langage machine et l'assembleur sont des langages impératifs. Le code en Visual Basic l'est aussi.

**Exemple :** réinitialiser un tableau en donnant la valeur 0 à tous ses éléments.

```
10 Dim tableau (10) as Integer
20 Dim compteur As Integer
30 compteur= 0
40 boucle:
50 tableau(compteur)=0
60 compteur= compteur + 1
70 If Compteur <11 Then Goto boucle
```

Ici on utilise un Basic très ancien avec un goto pour faire une boucle et des numéros de ligne, c'est pas très 'lisible'.

**Et c'est quoi la programmation non impérative ?**

C'est par exemple la **programmation déclarative,** elle consiste à énoncer les propriétés d'un système de résolution (à les déclarer) plutôt qu'à décrire les opérations à effectuer comme dans le cas de la programmation impérative. VB ne permet pas la programmation déclarative.

Un programme **déclaratif** définit (ou "déclare") différentes entités et leurs relations, à charge ensuite pour le programme d'utiliser ces relations pour résoudre le problème.

## XVI-A-2 - Programmation structurée

Pour éviter les programmes 'spaghetti', on a structuré et utilisé les procédés suivants :

### Découpage en fonction

L'approche structurée découpe le problème en fonctions.

L'analyse se fait de manière **descendante :** on découpe un problème complexe en problèmes plus simples qui sont eux-mêmes découpés en problèmes plus simples. On découpe jusqu'à ne plus avoir que des problèmes simples.

Il existe aussi l'analyse **ascendante :** ayant à sa disposition des procédures simples, on les assemble en les faisant appeler par des procédures plus complexes pour atteindre la solution.

Si le projet est entièrement nouveau, on fait plutôt une analyse descendante ; si on travaille sur un projet possédant déjà toutes les procédures simples, on raisonnera en analyse ascendante.

Rien n'empêche d'avoir une analyse mixte.

Les programmeurs doivent donc décomposer leur code en petites fonctions, assez petites pour être facilement comprises et claires.

### Utilisation de variables locales

En général les programmes doivent éviter d'utiliser des variables globales.

Au lieu de cela, les sous-programmes doivent utiliser des variables locales et agir sur des arguments ou paramètres qui leur sont envoyés.

### Organisation hiérarchique simple du code

La programmation structurée recommande une organisation hiérarchique simple du code. Pour cela on utilise des structures de contrôles *while, Do Loop, for, if … then … else.*

Il est également recommandé de n'avoir qu'un point d'entrée pour chaque boucle (et un point de sortie unique dans la programmation structurée originelle).

### Éviter les 'Goto'

L'instruction **Goto**, directement héritée des instructions de saut des langages machine (Jmp), était nécessaire dans les langages primitifs (Fortran, Assembleur) comme instruction de base permettant de réaliser des boucles et autres structures de contrôles (voir exemple sur la programmation impérative).

En programmation structurée (depuis les années 1970), l'instruction **Goto** n'est guère appréciée des programmeurs, car elle casse la structure séquentielle du programme et rend souvent les programmes plus difficiles à comprendre et à maintenir (on parle dans ce cas de programmation spaghetti). On utilise plus généralement des structures comme les sauts conditionnels (**si … alors … sinon …** ) ou les boucles (**pour, tant que**, etc.)

En VB, des instructions effectuent des sauts conditionnels ou inconditionnels (**If… Then**) et remplacent l'usage de l'instruction Goto, d'autres instructions (**For…Next, Do… Loop**) permettent d'élégantes boucles.

### Rupture de séquence

Il y a même des instructions qui 'cassent' élégamment les sorties de boucles, c'est le cas des instructions comme Continue ou Exit For. L'erreur majeure de Goto, se situe dans le fait que cette instruction renvoie vers une position précédente du code (aboutissant à ce que l'on appelle le "code spaghetti"), tandis que les deux autres renvoient (le plus souvent) vers un point du code situé logiquement après la boucle qu'elles interrompent.

```
While condition
```

```
    …Continue While
End While
```

Exemple : réinitialiser un tableau en donnant la valeur 0 à tous ses éléments.

```
initialisetableau:
Dim tableau (10) as Integer
Dim compteur As Integer
For compteur= 0 To 10
  tableau(compteur)=0
Next compteur
Return
```

On crée un sous-programme nommé **initialisetableau** qui a la fonction d'initialiser le tableau. Il n'utilise plus de GoTo mais une boucle For Next.

Ce **sous-programme** était appelé par un **Gosub** (Gosub n'existe plus en VB.Net).

## XVI-A-3 - Langage fonctionnel

C'est un langage généralisant l'usage des fonctions mathématiques. En "programmation fonctionnelle", les entités sont des fonctions au sens mathématique du terme. Une fonction en appelle d'autres qui en appelle d'autres.

Le programme principal est lui-même considéré comme une fonction qui fait appel à d'autres fonctions qui elles-mêmes…

**Les langages de programmation fonctionnelle dits "purs"** ne proposent ni affectation de variable, ni allocation de mémoire, ni boucles. Ces deux derniers procédés sont respectivement remplacés par les allocations automatiques et l'usage intensif de la récursivité.

Exemple de langage fonctionnel: LISP. (Pas VB.)

## XVI-A-4 - Programmation procédurale

VB en fait.

La programmation procédurale utilise des fonctions nommées **'procédure'**. Une procédure, aussi appelée *routine, sous-routine, **méthode*** ou ***fonction*** (**Sub** et **Function** en VB) contient simplement une portion de code qui effectue une fonction précise.
N'importe quelle procédure peut être appelée à n'importe quelle étape de l'exécution du programme, incluant d'autres procédures ou même la procédure elle-même (récursivité).

On peut, en appelant la procédure, envoyer des **paramètres.**

**Avantages**

- La possibilité de réutiliser le même code à différents emplacements dans le programme sans avoir à le retaper.
- Une façon plus simple de suivre l'évolution du programme.
- La création d'un code plus modulaire et structuré.

Exemple : réinitialiser un tableau en donnant la valeur 0 à tous ses éléments.

```
Sub InitialiseTableau ( tableau() As Integer)
Dim compteur As Integer
For compteur= 0 To Ubount (tableau,1)
```

```
    tableau(compteur)=0
Next compteur
End Sub
```

Ici on utilise une procédure, une Sub qui a pour seule fonction la réinitialisation du tableau.

## XVI-A-5 - Programmation défensive

Se dit d'une programmation où l'on considère que **le programme peut contenir des erreurs et que l'utilisateur est parfaitement malveillant** et fera tout pour faire planter le programme. Ainsi donc, il faut s'en défendre.

Elle consiste à ajouter du code vérifiant systématiquement l'état du système ainsi que la valeur des paramètres des fonctions et s'assurant que le changement d'état est consistant. Si une erreur est détectée, elle sera traitée.

En premier lieu donc on vérifiera que **toutes les entrées (saisie au clavier, lecture de fichier…) sont valides** pour le programme et ne contiennent pas, par exemple, une valeur pouvant provoquer une exception non gérée ultérieurement.

Pour se défendre des entrées invalides, on utilise la 'tolérance de faute'.

**Pour toutes entrées :**

- on teste les valeurs, on accepte uniquement les valeurs permises ;
- on gère les exceptions avec Try… Catch ;
- on utilise les assertions.

**Une entrée invalide entraine grâce à la programmation défensive :**

- soit l'arrêt du programme (programmation défensive forte) ;
- soit l'utilisation de valeur par défaut ou d'une ancienne valeur ;
- soit l'envoi à l'appelant l'indication qu'il y a une mauvaise entrée :
    - retour d'une valeur de diagnostic,
    - déclenchement d'une exception chez l'appelant (utilisation de Throw en VB dans une classe).

L'appelant, le client doit donc tester si la valeur de retour est valide ou bien gérer les exceptions qui sont retournées (solution qui semble préférable). Il devra donc traiter l'erreur.

Exemple

```
Try
SaveFile (maFile)
Catch E As Exception
    MsgBox (Exception.Message)
End Try
```

## XVI-A-6 - Programmation sécurisée

La **programmation sécurisée** va au-delà de la programmation défensive. Elle consiste à prendre en compte la sécurité informatique à tous les moments de la conception, de la réalisation et de l'utilisation d'un programme. Cela permet d'éviter au maximum les trous de sécurité et autres bugs.

## XVI-A-6-a - Conception

Lors de la conception, il s'agit de concevoir le programme de façon modulaire et nécessitant le moins de droits utilisateur possible. Il est préférable d'avoir plusieurs programmes de taille réduite qui collaborent entre eux, qu'un gros programme.

## XVI-A-6-b - Réalisation

Ensuite, lors de la réalisation, il faut penser à bien valider les données entrées par l'utilisateur. L'idée générale et la plus importante est de ne jamais faire confiance à l'utilisateur. Ne jamais faire des hypothèses sur les entrées sans les vérifier soi-même (par exemple taille de l'entrée, signe du nombre…). C'est **la programmation défensive.**

**Mais on peut généraliser ce processus :**

- en testant les entrées de toutes les procédures (**Préconditions**) ;
- en testant les sorties de toutes les procédures afin qu'elles soient conformes à ce qu'attend l'appelant (**Postconditions**).

On peut aussi faire de la **programmation par contrat :** l'appelant vérifie que les préconditions sont remplies et envoie à la procédure des 'paramètres valides'. La procédure effectue son code et vérifie que ce qu'elle retourne est valide. Il y a contrat entre l'appelant et la procédure appelée : l'appelant vérifie les préconditions seulement et sait qu'on lui retournera des informations valides. La procédure sait que les paramètres qu'elle recevra sont valides ; elle vérifiera la validité de ses résultats avant de les retourner.

**Invariants :** en plus des pré et post conditions, tous les objets inclus dans l'échange doivent être laissés dans le même état entre le début et la fin de celui-ci. Il faut s'assurer que le système entier conserve une certaine uniformité (et que l'on évite donc les **effets de bord** disgracieux).

## XVI-A-6-c - Exécution

Enfin lors de l'exécution, il faut penser par exemple à appliquer les différentes mises à jour de sécurité lorsqu'elles sortent. Pour ce faire, il peut être pratique de la part du concepteur de l'application de proposer un système de mise à jour simplifié de l'application.

**Effet de bord**

- Il peut arriver qu'une variable dispose d'une portée qui dépasse la procédure qui la contient, ceci afin d'être accessible à partir d'autres procédures. Certaines procédures peuvent ainsi modifier une variable dans le seul but de les maintenir dans un état donné, fixé par le développeur.
  Cette capacité d'une fonction de modifier l'état d'une valeur (variable globale ou statique, argument d'une autre fonction, affichage ou écriture des données) autre que celle qu'elle renvoie définit l'effet de bord.
- Ce mécanisme crée une sorte d'interdépendance entre les fonctions, rendant souvent plus complexe la compréhension d'un programme… D'autant que la modification d'une fonction peut dès lors avoir des conséquences inattendues sur le résultat d'autres fonctions "liées".

## XVI-A-7 - Programmation événementielle

**Avant VisualBasic 1**

Programme Basic **SANS programmation événementielle :**

```
10 PRINT "Donne ton nom";
20 INPUT N$
30 PRINT "Donne ton prénom";
```

```
40 INPUT P$
50 PRINT "Bonjour "; P$; " "; N$
60 END
```

L'exécution de ce programme conduit au résultat suivant :

```
C: > RUN
Donne ton nom ? LASSERRE
Donne ton prénom ? PHILIPPE
Bonjour PHILIPPE LASSERRE
C: >
```

Le programme affiche des informations à l'écran avec PRINT et utilise la commande INPUT lorsqu'il a besoin que l'utilisateur lui communique une information, au moyen du clavier.

Le programmeur indique la succession des lignes, **leur ordre est imposé.** Le programme s'arrête quand il attend une frappe au clavier puis redémarre quand l'utilisateur a validé sa réponse (en appuyant sur la touche 'Return'). On constate que l'ordre de saisie des informations est totalement déterminé par le programme. Il n'est pas question ici que l'utilisateur indique son prénom avant son nom.

Parfois pour assouplir les choses, on créait une boucle qui lisait à la volée, le clavier et qui en fonction de la touche appuyée permettait un choix.

**Depuis Visual Basic 1**

**En environnement graphique, l'interaction avec l'utilisateur est beaucoup plus élaborée :**

- il y a un environnement graphique ;
- la saisie d'informations peut s'effectuer au moyen du clavier, mais aussi de champs de saisie, boutons, listes, sélecteurs, cases à cocher, menus ;
- et surtout, l'ordre d'exécution des différentes opérations n'est pas strictement déterminé à l'avance ;
- toute action de l'utilisateur sur l'interface graphique déclenche des événements. Si l'utilisateur clique sur un bouton l'événement bouton.Click est déclenché ;
- pour chaque événement, on exécute une procédure (une Sub).

Du point de vue du développeur, cela change complètement la donne : ce n'est plus lui qui décide ce que va faire l'utilisateur ! Au contraire, il doit s'attendre a priori à n'importe quelle action de la part de ce dernier.

C'est ici qu'intervient la notion de **programmation événementielle :** le programme est structuré non plus pour exécuter une séquence d'instructions dans un ordre prédéfini, mais pour **réagir à des événements** qu'il consomme l'un après l'autre.

**Exemple pratique :** l'utilisateur saisit son nom et son prénom, il clique sur le bouton 'OK' ; une boite s'ouvre indiquant "Bonjour…'


Pour faire ce programme, il faut **dessiner l'interface utilisateur,** Vb fournit les événements et leur procédure. Il faut ensuite écrire dans la procédure correspondant à l'événement 'Bouton1-Click' le code qui affiche 'Bonjour…'.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
 Handles Button1.Click
    MsgBox("Bonjour " &amp; TextBox1.Text &amp; TextBox2.Text)
End Sub
```

Il existe d'autres événements (Form.Load, Button1.MouseDown…) qui peuvent ne pas être utilisés.

**Décortiquons le code**

Comment cela se passe en VB ?

Le composant (le bouton) doit se lier à un écouteur d'événement à qui il va **déléguer** le traitement de l'événement.

Le traitement de l'événement est **délégué** à l'écouteur et l'écouteur exécute une méthode spécifique qui prend en paramètre l'événement qui s'est produit.

Dans la Région "Code généré par le Concepteur Windows Form" il y a :

```
Friend WithEvents Button1 As System.Windows.Forms.Button
```

On crée une variable Button1 pour le bouton de commande, **WithEvents** indique que l'objet Button1 a des événements.

On voit dans le code la procédure événement Button1_Click :

```
Protected Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click

End Sub
```

Le terme **Handles** provoque l'association d'un événement (**Button1.Click** situé après Handles) à un gestionnaire d'événements (la **Sub Button1_Click**) il crée **une délégation.** Ainsi, à l'événement Button1.Click correspond la procédure **Sub Button1_Click.**

la Sub pourrait d'ailleurs se nommer différemment, cela n'a pas d'importance.

Plusieurs événements peuvent déclencher une seule Sub.

L'association d'événements aux gestionnaires d'événements se fait au moment de la compilation et ne peut pas être modifiée.

**En conclusion** avant la programmation événementielle on imposait :

```
Faire ceci.
Faire cela.
Lire le clavier
Faire ceci.
…
```

Avec la programmation événementielle, c'est plutôt :

```
Si l'utilisateur tape du texte faire ceci.
Si l'utilisateur clique sur le bouton faire cela.
….
```

## XVI-A-8 - Programmation-Objet

Les objets sont les choses physiques ou abstraites qui nous entourent. Typiquement, dans un programme de fiches de paie, le bulletin de salaire, l'employé, etc. sont des objets.

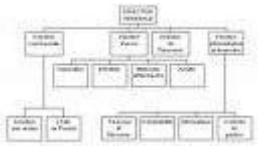
Un *objet* est une entité cohérente rassemblant des données et du code travaillant sur ses données.

Un logiciel est alors vu comme des d'objets communiquant par des méthodes.

Une *classe* peut être considérée comme un moule à partir duquel on peut créer des objets.

*"Ne commencez pas par demander ce que fait le système, demandez À QUOI il le fait !"*

# XVI-B - Programmation 'procédurale' ou 'objet' ?



**On a vu qu'on pouvait créer des programmes avec des Sub et des Functions, mais aussi avec des objets. Détaillons.**

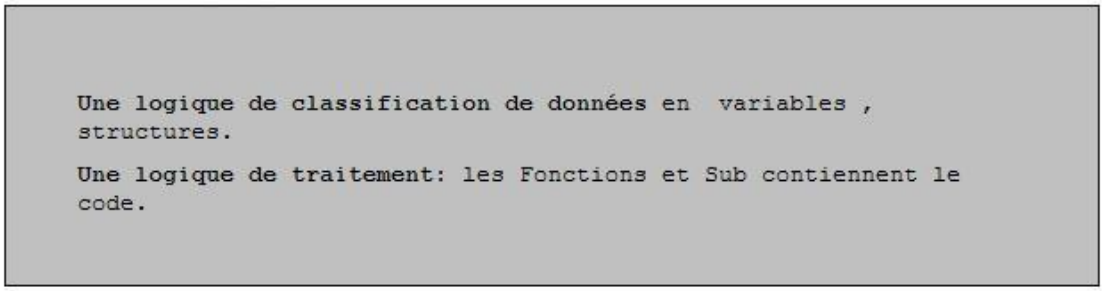
## XVI-B-1 - L'approche procédurale

Elle découpe le problème en fonctions.

Chaque fonction effectue une tâche précise. Avec l'aide de variables et de structures.

La base de la réflexion est effectuée autour des traitements. Le concepteur considère ainsi les tâches que doit accomplir le programme, en décomposant celui-ci en une série de tâches simples (approche descendante) ou en le construisant par composition de traitements (fonctions) disponibles (approche ascendante).

Il y a donc




Exemple

Calcul du salaire d'un employé. (Nombre d'heures * Taux horaire)

**Il faut écrire une Fonction CalculSalaire :**

```
Public Function CalculSalaire(Taux As Single, Heure As Single) As Single
    Return Taux*Heure
End Function
```

**Pour calculer un salaire, il faut appeler la fonction avec les bons paramètres :**

```
Dim TauxHoraire As Single
Dim HeuresTravaillées As Single
Dim Salaire As Single
TauxHoraire=30
HeureTravaillées=70
 Salaire=CalculSalaire(TauxHoraire,HeuresTravaillées)
```

Pour structurer le programme, on utilise des **'Modules Standards'** contenant les diverses fonctions Un module contient par exemple **Function CalculSalaire.**

**Le point d'entrée du programme** est une procédure Main() :

```
Module module1

Sub main()
……Créer une instance du formulaire de démarrage et l'ouvrir
End Sub


Public Function CalculSalaire(Taux As Single, Heure As Single) As Single
    Return Taux*Heure
End Function


End Module
```

Si on utilise des **variables globales visibles dans la totalité du programme (c'est à éviter)**, il faut les mettre dans un module standard et les définir comme Public.

Exemple nom et version du programme :

```
Module module1
Public nomProgramme As String= "Calcul Salaire"
Public versionProgramme As String= "1.2"
……

End Module
```

Ainsi le nom du programme et sa version sont accessibles partout.

Dans un formulaire on peut afficher le nom du programme dans la barre supérieure.

```
Me.Text= nomProgramme
```

Cette accessibilité semble un avantage, en fait c'est dangereux : n'importe quelle procédure peut modifier les données.

**Pour structurer les données**, on peut utiliser les **Structures.**

Exemple : créons une structure 'Salarié' :

```
Structure Salarié
Public Nom As String
Public TauxHoraire As Single
Public HeuresTravaillées As Single
End Structure
```

Ensuite pour avoir 100 salariés :

```
Dim Salaries(100) As Salarié
```

On pourra utiliser **Salaries(1).Nom** ou **Salaries(2).TauxHoraire**

On verra plus loin qu'utiliser des variables dites 'globales'(visibles partout) n'est pas une bonne chose, il vaut mieux créer des **procédures d'accès aux données.**

Plutôt que d'écrire **Salaries(1).Nom="Lulu"** il est préférable d'écrire une procédure **SalariesAdd()** qui ajoute un salarié avec son nom dans le tableau.

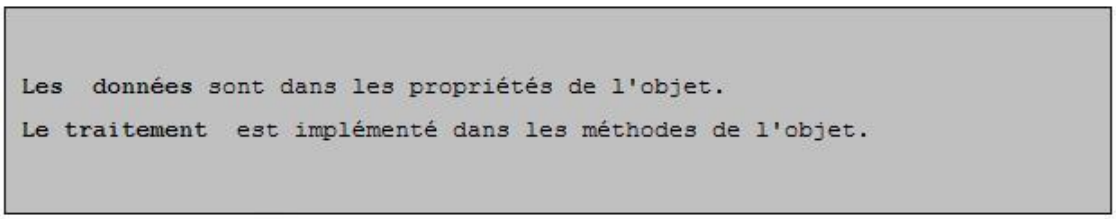
## XVI-B-2 - Approche Objet

Elle est centrée sur les Objets (et non sur les tâches).

Elle nécessite de créer une Classe (le moule).

Avec l'aide de la classe on peut déclarer des objets.

Chaque Objet a des propriétés, des méthodes.




Exemple

Calcul du salaire d'un employé. (Nombre d'heures * Taux horaire)

**Il faut écrire dans un module de Class une Class Employé :**

```
Public Class Employé
Private T As Single     'propriétés privées à la classe pour stocker les heures et taux horaires
Private H As Single

Public Property Taux As Single 'propriété Taux
Get
    Return T
End Get
Set(By Val Value)
    T=value
End Set
End Property

Public Property Heure As Single 'propriété heure
Get
    Return H
End Get
Set(By Val Value)
    H=value
End Set
End Property

Public Property Salaire As Single 'méthode Salaire
Get
    Return T*H
End Get
End Property
End Class
```

**Pour calculer un salaire, il faut créer un objet Employé, donner les bonnes valeurs aux propriétés et appeler la méthode salaire.**

```
Dim UnEmployé As new Employé
UnEmployé.Taux=30
UnEmployé.Heure=70
 Dim Salaire As Single =UnEmployé.Salaire
```

**On voit donc qu'il faut créer des 'Modules de Classe' pour y mettre les nouvelles classes.**

On évitera les modules standards qui ne sont pas dans l'esprit 'Objet'.

**Le point d'entrée du programme** pourrait être une Classe statique (Classe ne nécessitant pas d'instancier un objet) :

```
Public Class main2
Public Shared Sub main()
….
End Sub
End Class
```

Si on utilise des **variables qui doivent être accessibles,** il faut les mettre dans une Classe et les définir comme Shared. Ainsi une variable partagée d'une classe (Shared) a non seulement une valeur commune à toutes les instances de la classe, mais en plus on peut travailler directement sur la Classe (sans instancier).

Exemple nom et version du programme :

```
Class MonProgramme
Public Shared nomProgramme As String= "Calcul Salaire"
Public Shared versionProgramme As String= "1.2"
……

End Class
```

Ainsi le nom du programme et sa version sont accessibles partout.

Dans un formulaire on peut afficher le nom du programme dans la barre supérieure.

```
Me.Text= MonProgramme.nomProgramme
```

On peut créer une classe **qui hérite** des propriétés d'une autre classe.

Dans notre exemple en programmation-objet, on créera une Class 'Patron' qui hérite de la classe 'Employé', mais dont la méthode Salaire sera redéfinie **(Overrides)**. (En programmation procédurale, il faudra écrire une nouvelle fonction SalairePatron.)

```
Public Class Patron
Inherit Employé

Public Property  Overrides  Salaire As Single 'méthode Salaire
Get
    Return T*H
End Get
End Property
End Class
```

**Pour gérer un ensemble, un groupe de données,** on utilise une classe encapsulant une collection privée d'objets voir chapitre 5-7.

```
Public Class LesEmployes
     Private maCol As ArrayList

     Public Sub New()
          maCol = New ArrayList()     'cela crée une ArrayList
     End Sub


     Public Function Add(ByVal LEmploy As Employe) As Employe
      …
     End Function

     Public Property Item(ByVal lIndex As Integer) As Employe
     ….
     End Property


End Class
```

On rappelle que les classes peuvent contenir des méthodes, des variables publiques ou privées, mais elles sont 'PAR RÉFÉRENCE'.

Les partisans de la programmation-objet auront tendance à utiliser exclusivement les classes du Framework plutôt que les instructions de Microsoft.VisualBasic dans leur code.

## XVI-B-3 - Conclusion

La programmation fonctionnelle se focalise sur les **actions à effectuer**, alors que la programmation-objet se focalise sur **les données.**

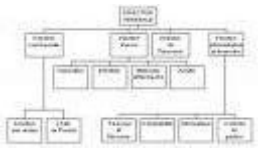
La méthode procédurale est plus intuitive, on a l'impression d'être plus proche de la réalité 'physique', le code est probablement plus rapide.

L'emploi d'objets permet une abstraction plus importante, une puissance inégalée grâce à l'héritable, aux surcharges.

On peut être puriste et ne programmer qu'en procédurale ou ne programmer qu'en objet.

Visual Basic permet de mélanger les 2 approches.

## XVI-C - Programmation 'procédurale' : faire de bonnes procédures



Quand on fait de la programmation procédurale : on n'utilise pas de classe, mais des modules standards, des Sub et des Functions.

Suivant quelles règles découper son programme ?

Si on programme depuis longtemps, on le sait de manière 'intuitive' (après de nombreuses erreurs), il est tout de même intéressant de connaitre les grandes règles à suivre. Très peu de sites ou d'ouvrages en parlent !!

Analyse **ascendante, descendante ?**

**Pourquoi découper** en procédures ?

La **cohésion** doit-elle être importante ?

Le **couplage** doit-il être modéré ?

Comment utiliser les **paramètres ?**

**Sub** ou **Function ?**

**Programmation défensive ?**

### XVI-C-1 - Approche procédurale, analyse 'descendante' ou 'ascendante'

L'approche procédurale découpe le problème en fonctions (ou procédures).

L'analyse se fait de manière **descendante :** on découpe un problème complexe en problèmes plus simples qui sont eux-mêmes découpés en problèmes plus simples. On découpe jusqu'à ne plus avoir que des problèmes simples.

Il existe aussi l'analyse **ascendante :** ayant à sa disposition des procédures simples, on les assemble en les faisant appeler par des procédures plus complexes pour atteindre la solution.

Si le projet est entièrement nouveau, on fait plutôt une analyse descendante, si on travaille sur un projet possédant déjà toutes les procédures simples, on raisonnera en analyse ascendante.

Rien n'empêche d'avoir une analyse mixte.

On rappelle :

chaque procédure effectue une tâche précise ;

les procédures sont composées des ' **Sub** ' et des ' **Function** ' ;

une **procédure est un ensemble d'instructions, de lignes de code**, un groupement d'instructions bien définies effectuant une tâche précise.

**Les procédures 'Sub'**

Elles débutent par le mot **Sub** et se terminent par **End Sub.**

Exemple :

```
Sub MaProcédure()
   A=1
End Sub
```

Pour appeler la Sub :

```
MaProcedure()
```

**Les procédures 'Function'**

Les 'Function' retournent une valeur.

Elles débutent par **Function** et se terminent par **End Function.**

Exemple :

```
Function SurfaceCercle( Rayon as Single)
    Return 3.14*Rayon*Rayon        'Return indique ce que la fonction doit retourner
End Function
```

Dans la procédure qui appelle, il faut une variable pour récupérer la valeur retourner par la Fonction :

```
S= SurfaceCercle()
```

*On rappelle que le nom de la procédure doit être **en minuscules avec la première lettre de chaque mot en majuscule.***

***SurfaceCercle()** est correct.*

## XVI-C-2 - Pourquoi utiliser des procédures ?

D'abord, dans certains cas, c'est obligatoire !! Il y a les procédures événements automatiquement crées par VB. Ensuite, dès que l'on veut écrire du code, il faut créer des procédures Sub ou Function dans les modules de formulaire ou dans les modules standard (voir 5-10).

**La programmation procédurale avec découpage en procédures a des avantages.**

- **On évite la répétition du même code**
  S'il faut 30 lignes de code pour faire un calcul et que le calcul est effectué 10 fois dans le programme, il est pratique de créer une **Sub CalculTotal()** contenant les 30 lignes de calcul et d'appeler 10 fois la Sub.
  Voici la Sub :

```
Public Sub CalculTotal()
À=B+C
D=E+3
```

```
        …
        End Sub
```

Voici les appels :

```
        CalculTotal()
        …
        CalculTotal()
```

- **On simplifie les modifications**
  S'il y a une modification à faire dans le calcul, la modification est effectuée une fois (Avec 10 fois le code du calcul disséminé dans le programme, on risque d'oublier de modifier un des codes).
- **On introduit une abstraction qui clarifie le code**
  **CalculTotal()** est plus compréhensible que **A=B+C….**
  On oublie le code interne à la procédure, pour ne plus se souvenir que de la fonction de la procédure : la procédure fait le Calcul du total…
  On masque ainsi l'implémentation.
- **On réduit la complexité**
  Chaque tâche complexe est décomposée en plusieurs procédures qui elles-mêmes sont décomposées en procédures plus simples.
- **On améliore la portabilité**
  On isole dans les procédures le code non portable et on le groupe.
  Si on doit effectuer un changement (changement de type de base de données par exemple en modifiant un programme pour qu'il fonctionne sous MySQL alors qu'il fonctionnait sous Acces), il suffit de modifier uniquement les procédures d'accès à la base de données.
- **On peut masquer certaines fonctions**
  Certaines fonctions sont complexes et leur code est peu intéressant, une fois écrites il vaut mieux ne plus la voir. (Fonctions booléennes complexes.)

## XVI-C-3 - La 'cohésion' de la procédure doit être importante

*La **'cohésion'** fait référence aux **liens établis entre les opérations de la procédure** DANS la procédure.*

Un exemple

La **Sub Carré()** à une forte cohésion : elle est entièrement consacrée à une tâche: le calcul du carré d'un nombre.

La **Sub CarréEtCube()** qui calcule le carré ou le cube d'un nombre a une cohérence faible !! (Il vaut mieux dans ce cas écrire une procédure Carré() et une procédure Cube() )

Compris ?

Si la cohésion est faible, il a été prouvé que les procédures contenaient plus d'erreurs de programmation.



*Donc **une procédure ne doit faire qu'une seule chose.***

- On parle ici de **cohésion fonctionnelle :** la fonction de la procédure est unique.
  **EffaceFichier(), CalculAge()** ont une bonne cohésion fonctionnelle, **SaisirDate&CalculAge()** a une moins bonne cohésion.

**Il existe d'autres sortes de cohésion.**

- La **cohésion temporelle**
  Différentes opérations qui n'ont rien à voir entre elles doivent être effectuées au même moment.
  Exemple : la **Sub Demarrage()** appelée au début d'un programme va ouvrir les fichiers, initialiser les variables, lire le fichier de configuration.
- La **cohésion séquentielle**
  Lorsque les opérations qui n'ont rien à voir entre elles doivent impérativement être effectuées dans un ordre précis.
  Exemple : la **Sub CalculAnnéeRetraite()** qui à partir de la date de naissance va calculer l'âge puis l'année de retraite. (On pourrait aussi dans ce cas écrire une routine CalculAge() qui serait appelée par CalculAnnéeRetraite() )

**Certaines cohésions sont à éviter**

- **La cohésion de communication**
  Les opérations travaillent sur les **mêmes données,** mais n'ont pas de relation.
  **ImprimeAndEffaceTableau()** me parait à éviter : écrire les procédures ImprimeTableau() et EffaceTableau().

**D'autres sont à bannir**

- La **cohésion logique**
  La sélection des opérations (sans relation entre elles) à effectuer est pilotée par un paramètre.
  Seul le paramètre est commun !!
  La Sub **ImprimerTableauAfficherResultat** (Flag) ou le Flag indique s'il faut imprimer ou afficher est à bannir.

## XVI-C-4 - Le 'couplage' entre procédures doit être modéré

*Le couplage traite des **connexions entre les différentes procédures, il doit être modéré**, ce qui signifie qu'il doit y avoir peu de connexions.*

- **Une procédure doit ne communiquer qu'avec un nombre "minimum" d'autres procédures** du logiciel.
  L'objectif est de minimiser le ***nombre*** d'interconnexions entre les procédures. L'intérêt de cette règle est de garantir un meilleur respect de la protection des procédures.
- Lorsque deux procédures communiquent entre elles, **l'échange d'information doit être minimal.** Il s'agit de minimiser la taille des interconnexions entre modules.
  Éviter le tout 'variables globales'.
  Une Sub ayant 2 paramètres c'est bien, une Sub en ayant 15 c'est trop.
- Lorsque deux procédures communiquent, l'échange d'information doit être explicite.
  Évitez qu'une variable soit visible dans une procédure, alors qu'elle est déclarée dans une autre.
  Si vous modifiez des variables globales utilisées par un autre module c'est dangereux !!
  Si vous utilisez un paramètre pour passer une donnée, c'est explicite, c'est OK.
- Toute information dans une procédure doit être répartie en deux catégories : **l'information privée et l'information publique.** Ce principe permet de modifier la partie privée sans que les clients (*fonctions utilisant cette procédure*) soient confrontés à un quelconque problème à cause de modifications ou de changements.
  Plus la partie publique est petite, mieux c'est…
  On déclarera en début de procédure des variables privées, et on travaillera sur ses variables privées, seul le résultat sera accessible de l'extérieur.

Une procédure doit donc être autonome.



*Une procédure est autonome et ne doit pas lire ni modifier directement les variables du programme qui l'appelle, sans passer par des paramètres.*

## XVI-C-5 - Squelette d'un programme

**Exemple simpliste de l'agencement des procédures**

De manière générale, le programme est composé de procédure, en appelant d'autres, qui en appelle d'autres…

'Squelette' de programme :

```
Sub Démarrage
    OuvrirFichier()
    LireConfig()
    InitialiserVariable()
    LireDonnées()
End Sub

Sub Quitter
        EnregistrerDonnée()
        FermerFichier()
End Sub
```

Voici la Sub Main(), procédure de démarrage du programme :

```
Sub Main()
    Démarrage()
    'puis affiche le formulaire principal
End Sub
```

Dans le formulaire principal, un bouton 'Quitter' déclenche la Sub suivante :

```
Sub BoutonQuitter_Click(…)
    Quitter()
End Sub
```

Un bouton 'Nouveau' (travail sur de nouvelles données) contient :

```
Sub Nouveau_Click(…)
       EnregistrerDonnée()
       InitialiserVariable()
End Sub
```

On remarque que **EnregistrerDonnée()** et **InitialiserVariable()** sont appelés de plusieurs endroits.

Ensuite il suffit d'écrire tout le code des Sub.

## XVI-C-6 - Les paramètres

**Dans quel ordre les mettre ?**

Une Sub reçoit 3 paramètres : P1 qui est une donnée qu'utilise la Sub, P2 qui est modifié par la Sub, P3 qui contient un résultat retourné par la Sub.

Et bien il est conseillé de les mettre dans l'ordre P1, P2, P3 (**Entrée, Modification, Sortie**).

Exemple **Sub CalculCarré(nombre, carré)** 'quand on appelle cette Sub on envoie un nombre, on récupère le carré.

**S'il y a une variable d'erreur ou d'état, la mettre en fin**

```
        Sub Divise(Nombre1, Nombre2, CalculImpossible)
```

Cette routine calcule Nombre1/Nombre2, mais retourne CalculImpossible=True si Nombre2 est égal à zéro.

**Nombre de paramètres ?**

7 maximum, au-delà de 7 on s'y perd !!

Dans les 7, il peut y avoir des tableaux.

**Paramètres pour plusieurs routines**

Si plusieurs routines utilisent les mêmes paramètres, les mettre dans le même ordre.

**AfficheText(LeTexte, Couleur)** et **ImprimeText(LeTexte, couleur)** sont cohérents.

**Ne pas utiliser de paramètre comme variable de travail**

```
Sub calcul (A, B)
A=A+2                    Est à éviter: le paramètre A ne contient plus la valeur d'entrée!!
…                       car on a utilisé A comme variable de travail.
End Sub
```

Faire plutôt :

```
Sub calcul (A, B)
Dim V As Integer= A      'On passe la valeur du paramètre dans une variable locale.
V=V+2                    'C'est mieux! A contient toujours la valeur d'entrée.
…
End Sub
```

Il faut donc déclarer en début de Sub des variables locales propres à la Sub et travailler avec elles dans la Sub.

Mettre des commentaires en début de sub :

```
Sub Divise(Nombre1, Nombre2,Resultat, CalculImpossible)
'***Division de 2 Nombres
'Entrée: Nombre1, Nombre2 de type Single
'Sortie: Résultat de type Single
'Teste Nombre2 si =0 retourne CalculImpossible=True
If Nombre2= 0 then
    CalculImpossible=True: Exit Sub
End If
Resultat=Nombre1/Nombre2
End Sub
```

Tester la validité des paramètres en début de Sub

Voir l'exemple précédent

```
If Nombre2= 0 then    'teste le paramètre Nombre2 et quitte la Sub s'il n'est pas valide.
```

C'est mieux de le faire au début pour tous les paramètres plutôt que de le faire dans toute la procédure.

## XVI-C-7 - Utiliser une 'Sub' ou une 'Function' ?

Les puristes diront qu'il faut utiliser une **fonction** quand on a **une valeur de sortie (une seule).**

Plusieurs paramètres d'entrée sont possibles par contre.

Exemple

La fonction **FileExist(),** avec comme entrée le nom d'un fichier, retourne uniquement un Booléen (True ou False), c'est bien une fonction qu'il fallait utiliser.

On peut élargir la définition en employant une fonction quand **l'objectif principal est de retourner la valeur indiquée dans le nom de la fonction :**

**MetAuFormat(In, Out)** transforme le texte 'In' en 'Out'; si la fonction a échoué, elle retourne False.

On l'utilisera comme cela :

```
If MetAuFormat(In, Out) = True Then  LeText.Text=Out
```

On aurait pu utiliser une Sub, avec comme troisième paramètre une variable indiquant le succès ou l'échec.

```
MetAuFormat( In, Out, Succes)
```

Certains utilisent des noms de variable commençant par Is afin d'indiquer clairement que c'est une fonction et ce qu'elle retourne :

IsOpen() pour voir si un fichier est ouvert (retourne True si le fichier est ouvert). ce qui permet d'utiliser:

```
If IsOpen("monfichier") Then….
```

## XVI-C-8 - Programmation défensive

C'est comme quand on conduit une voiture, il faut se méfier des autres : bien regarder aux croisements et ralentir, même si on a la priorité !!

Il faut donc :

**Tester la validité des paramètres en début de Sub.**

On l'a déjà vu :

**If nombre2= 0 then** 'teste le paramètre Nombre2 et quitte la Sub s'il n'est pas valide ce qui évite une division par zéro si l'expression nombre1/nombre2 est utilisée.

C'est mieux de le faire au début pour tous les paramètres plutôt que de le faire dans toute la procédure.

Il faut le faire surtout si les données viennent de l'extérieur : **fichier, réseau.**

Il faut encore plus le faire si c'est **l'utilisateur** qui a saisi les données.

**Vérifier si la donnée est dans la plage attendue.**

Si on attend le numéro d'un jour du mois, vérifier que c'est un nombre, s'il est positif, compris entre 1 et 31, entier…

**Vérifier si la donnée est une String valide.**

Si on attend un nom de fichier, vérifier que c'est du texte, éliminer les espaces en début et fin, y a-t-il une extension ? Des caractères invalides ?

En VB il y a des fonctions qui font cela.

**Tester la validité des paramètres de sortie.**

**Prendre en charge les erreurs.**

Il y a des procédures 'tolérantes' :

- si une valeur n'est pas valide, on peut donner une valeur par défaut, redonner la précédente valeur… ;
- c'est le cas des prises de température à une fréquence importante. S'il manque une valeur, reprendre la précédente.

Il y a des procédures 'strictes', cela entraine une des actions suivantes :

- l'arrêt du programme !!;
- la procédure redemande la valeur ;
- elle retourne une variable indiquant qu'elle a échoué.

**Pendant le développement, utiliser les assertions**

C'est une manière de se contrôler soi-même **en cours de développement.**

On place des assertions dans le code :

si elles sont vraies, c'est que cela se passe comme prévu ;

si elles sont fausses, c'est qu'une erreur inattendue, inacceptable se produit.

**Debug.Assert** affiche une fenêtre Windows et stoppe le programme si une assertion (une condition) passe à **False.**

```
Debub.Assert(Assertion)
Debub.Assert(Assertion, Message1)
Debub.Assert(Assertion, Message1, Message2)
```

L'exemple suivant vérifie si le paramètre 'type' est valide. Si le type passé est une référence null (Nothing dans Visual Basic), Assert ouvre une boite de dialogue nommé 'Echec Assertion' avec 3 boutons 'Abandonner, Recommencer' 'Ignorer'… La liste des appels est affichée dans la fenêtre (procédure en cours en tête de liste, module et numéro de ligne en première ligne)

```
Public Shared Sub UneMethode (type As Type, Typedeux As Type)
Debug.Assert( Not (type Is Nothing), "Le paramètre Type est=Nothing ", _
 "Je ne peux pas utiliser un Nothing")
…
End Sub UneMethode
```

Il n'est pas souhaitable que l'utilisateur puisse voir les messages des assertions, elles disparaissent dans le code de production.

**En résumé**, une procédure doit avoir :

- **de la modularité :** un sous-programme réalise une tâche et une seule (par exemple, une fonction de calcul ne doit pas afficher de résultat) ;

- **de l'autonomie :** un sous-programme est autonome et ne doit pas lire ni modifier directement les variables du programme qui l'appelle, sans passer par des paramètres ;
- **de la transparence :** un sous-programme doit être conçu de façon à ce que le programmeur qui y fait appel (non nécessairement son concepteur) n'ait pas à tenir compte des choix du concepteur ;
- **de la convivialité :** l'appel du sous-programme doit être le plus évident possible.

## XVI-D - Programmation 'objet' : faire de bonnes Classes



*Ce chapitre tente de clarifier mes connaissances, ne pas hésiter à critiquer et me donner des conseils.*

Quand on programme avec une approche Objet, on crée des classes, on n'utilise par de modules standard ni de Sub et de Fonction.

Quelles règles suivre pour créer des objets ?

Si on programme depuis longtemps, on le sait de manière 'intuitive' (après de nombreuses erreurs), il est tout de même intéressant de connaitre les grandes règles à suivre. Très peu de sites ou d'ouvrages en parlent !!

Comment faire de bonnes Classes ?

Comment bien utiliser l'héritage ?

### XVI-D-1 - Rappel

On rappelle : une Classe sert à créer (instancier) des objets.

En programmation-objet, dans un module de Classe, on crée la Classe :

```
Classe MyClasse
…
End Class
```

puis dans le corps du programme, on crée un objet :

```
Dim MonObjet As New MyClasse.
```

### XVI-D-2 - Pourquoi utiliser 'Classe' et 'Objet' ?

Ils permettent :

**Modélisation des objets réels.**

Le programme travaille avec des objets réels : une facture, un client.

Créer une classe par objet puis :

- Y mettre les données liées à l'objet (le nom, l'adresse du client) ;
- Y ajouter les méthodes contenant le comportement de l'objet (Calcul de la facture).

**Modélisation des objets abstraits.**

Dans un programme de comptabilité, les classes 'technicien', 'ouvrier', dirigeant' sont concrètes, une classe 'Salarié' est plus abstraite. Les 3 premières pourront hériter de cette dernière.

Autre exemple classique : les classes Line, Triangle sont concrètes, la classe Shape (forme en français) est abstraite, ici aussi Line et Triangle hériteront de Shape.

Il existe même des **Classes abstraites:** une Classe abstraite est une Classe qu'on ne peut pas instancier, elle sert uniquement à définir des caractéristiques de base pour créer des classes qui seront dérivées de la Classe abstraite.

**Facilité pour réutiliser le code.**

Une fois créées, les Classes servent de briques pour construire un nouveau projet.

L'héritage permet la création de nouvelles Classes héritant d'une autre.

**Réduction et isolement de la complexité.**

Une tâche complexe est découpée en plusieurs Classes.

**Masquage de l'implémentation.**

Un fois la Classe créée, il faut oublier les détails et l'utiliser sans connaitre son fonctionnement interne.

C'est l'encapsulation.

**Masquage des données globales.**

Si vous devez absolument utiliser des données globales, vous les masquez derrière une interface de classe.

## XVI-D-3 - Identifier les objets

**Identifier les Objets pour définir ensuite les classes :**

1 **On identifiera les objets nécessaires et leurs attributs (méthodes, données)**
Exemple
Système de facturation:

| Objet : Client | Objet : Facture | Objet : FicheHoraire |
|---|---|---|
| NomClient<br>Adresse | Date<br>NomClient<br>Montant | Date<br>NonClient<br>NombreDheure<br>TauxHoraire |
| | CalculerFacture()<br>ImprimerFacture() | |

**Identifier les Objets pour définir ensuite les classes.**

2 **Déterminer ce que l'on peut faire avec chaque objet.**
Exemple
Modifier l'adresse du Client.

3 **Déterminer ce que chaque objet peut faire aux autres objets.**
Une facture peut contenir plusieurs fiches horaires.
Un Objet peut en hériter d'un autre.

4 **Déterminer les parties visibles (l'interface publique).**

5 **En conclusion, pour chaque objet il faut :**
6 Définir l'interface utilisateur :
7 Identifier les requêtes (information demandée à la classe). Ce sont les **Property,**
8 Identifier les commandes (procédures) : ce sont les méthodes. **Sub** ou **Function,**
9 Identifier les contraintes (pré, postcondition),
10 Définir les constructeurs. **Sub New(),**
11 Choisir une représentation de l'information (attributs),
12 Implanter les fonctions, procédures et opérateurs,
13 Définir (si nécessaire) le constructeur de copie, le destructeur.

## XVI-D-4 - Faire un 'couplage' modéré

*Le couplage traite des **connexions entre les différentes Classes, il doit être modéré**, ce qui signifie qu'il doit y avoir peu de connexions.*

- **Une Classe doit ne communiquer qu'avec un nombre "minimum" d'autres Classes** du logiciel. L'objectif est de minimiser le ***nombre*** d'interconnexions entre les Classes. L'intérêt de cette règle est de garantir un meilleur respect de la protection des Classes.
- Lorsque deux Classes communiquent entre elles, **l'échange d'information doit être minimal.** Il s'agit de minimiser la taille des interconnexions entre Classes.
  Une classe qui possède 4 méthodes Public c'est mieux qu'une Classe qui en possède 15.
  Il faut donc réduire l'accessibilité des classes et des membres.
- Lorsque deux Classes communiquent, **l'échange d'information doit être explicite.**
- Toute information dans une Classe doit être répartie en deux catégories : **l'information privée et l'information publique.** Ce principe permet de modifier la partie privée sans que les clients (fonctions utilisant cette classe) soient confrontés à un quelconque problème à cause de modifications ou de changements. Plus la partie publique est petite, mieux c'est…

## XVI-D-5 - Conserver une bonne abstraction et une bonne cohérence

Une classe qui :

- met en forme les données ;
- qui initialise le programme ;
- qui imprime un rapport

n'est pas cohérente.

Une classe qui :

- initialise une donnée ;
- charge une donnée ;
- sauve une donnée

est cohérente.

De plus l'abstraction est constante : elle se situe au même niveau (celui d'une donnée), si on rajoutait dans la classe une fonction effectuant une recherche dans une liste de données cela ne serait plus le cas.

## XVI-D-6 - Créer des méthodes par paires

Une méthode doit entrainer l'écriture de la méthode contraire.

Activation/désactivation.

Ajout/Suppression…

S'il existe la méthode 'AddLine' , on aura forcément besoin de 'RemoveLine'.

## XVI-D-7 - L'encapsulation doit être bonne

L'encapsulation est une barrière qui empêche l'utilisateur de voir l'implémentation…

**Il ne faut pas exposer les données.**

Éviter les variables Public, directement accessibles par l'utilisateur de la Classe.

```
Class MaClasse
Public X As Single
…
End Class
```

C'est pas génial !!

```
Class MaClasse
Private mX As Single
Public Property X() as Single
Get
    Return mX
End Get
Set(By Val Value)
    mX=value
End Set
End Property
…
End Class
```

C'est mieux, l'utilisateur n'a pas accès directement à 'mX', l'utilisateur voit 'X', il peut modifier X ce qui entraine une modification de mX, mais sous contrôle, on peut même rajouter des conditions qui seront testées avant de modifier mX.

## XVI-D-8 - Initialisez les données dans le constructeur d'une Classe

Il est important de ne pas oublier d'initialiser les variables dans une classe, le moment idéal est quand on crée une instance de cette Classe.

```
Class MaClasse
Private mX As Integer

Sub New()
    mX=2
End sub
End Class
```

## XVI-D-9 - Problèmes liés à l'héritage

**Ne pas multiplier les classes dérivées**

Si on a une Classe **'Animal'**, on peut créer les classes **'Cheval'** et **'Poisson'** qui dérivent de la Classe 'Animal'.

Par contre il ne faut pas créer les classes **'PoissonRouge'** ou **'ChevalNoir',** car la couleur est plus un attribut d'une classe qu'un déterminant de Classe.

**Éviter trop de classe intermédiaire**

Si **'Cheval'** dérive de **'Mammifèreà4Pattes'** qui dérive de **'Mammifère'** qui dérive d'**Animal**', c'est lourd !! surtout si 'Mammifèreà4Pattes' et 'Mammifère' sont des classes abstraites qui ne seront pas instanciées.

**Éviter l'héritage de construction**

Éviter de dériver une Classe en ajoutant des attributs qui modifient le concept de la classe mère.

Une classe 'Triangle' peut dériver de 'Polygone', la classe 'Carré' peut difficilement dériver de 'Triangle' (en y ajoutant les attributs d'un quatrième point !!).

Un Carré **est un** Polygone,oui. On ne peut pas dire qu'un Carré est un Triangle avec un sommet de plus !!

**Bien comprendre la différence entre héritage et agrégation**

Comme on l'a vu dessus, quand une classe dérive d'une autre, on peut dire que la classe dérivée 'est une' classe mère.

'Carré' est un 'Polygone'.

Par contre 'Carré' est composé de 4 'Point' (possède 4 points) : un objet 'Carré' contient 4 objets 'Point'. On parle d'agrégation (ou de composition pour certains).

# XVI-E - Faire du bon 'code'

## XVI-E-1 - Bonnes variables

### XVI-E-1-a - Utilisez Option Strict=On et Option Explicit=On

Travailler avec

```
Option Strict=On
```

Vous serez obligé de faire du 'cast' à longueur de temps (conversion d'une variable d'un type vers un autre), mais au moins le code sera rigoureux.

Les conversions seront évidentes, explicites :

```
Dim I As Integer
Dim J As Long
J= CType(I, Long)
```

et avec

```
Option Explicit=On
```

Vous serez obligé de déclarer les variables avant de les utiliser. C'est indispensable.

Voir chapitre 1-7 pour le détail.

## XVI-E-1-b - Donnez à chaque variable un seul rôle

À l'époque où il y avait très peu de mémoire, un octet était un octet, aussi on utilisait parfois une même variable pour différentes choses :

```
Dim total As Integer
total= nbArticle*3
Affiche (total)
total= nbFacture*2
Affiche (total)
```

Ici total contient le total des articles puis le total des factures !!

Il vaut mieux créer 2 variables et écrire :

```
Dim totalArticle As Integer
totalArticle= nbArticle*3
Affiche (totalArticle)
Dim totalFacture As Integer
totalFacture= nbFacture*2
Affiche (totalFacture)
```

Dans le même ordre d'idées, éviter les variables nommées '**temp**' (temporaire) qui serviront à plusieurs reprises pour des résultats temporaires successifs.

## XVI-E-1-c - Évitez les variables avec des significations non évidentes

Éviter les significations cachées.

Exemple à ne pas faire :

- **nbFichier** contient le nombre de fichiers sauf s'il prend la valeur -1 , dans ce cas cela indique qu'il y a eu une erreur de lecture !!
- **clientId** contient le numéro du client sauf s'il est supérieur à 100000 dans ce cas soustraire 10000 cela donne le numéro de la facture.

Dans le premier exemple, la solution est d'utiliser en plus de **nbFichier** une variable **isReadOk** indiquant si la lecture s'est bien déroulée.

## XVI-E-1-d - Initialisez les variables dès leur déclaration

Cela évite de l'oublier. Les erreurs d'initialisation sont très fréquentes.

```
Dim Taux As Integer= 12
```

Un truc : si vous initialisez ou réinitialisez une variable avec NoThing, elle prend la valeur obtenue après déclaration pour ce type. Pour une structure, c'est valable pour chaque variable.

Exemple

Soit une structure Adresse

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

Je déclare une variable Adresse

```
Dim MonAdresse As Adresse
```

J'initialise :

```
MonAdresse.Numero = 12
MonAdresse.Id= 15
MonAdresse.Ville = "lyon"
```

Maintenant si je fais :

```
MonAdresse = Nothing
```

Cela entraine:

```
MonAdresse.Numero = 0
MonAdresse.Id= 0
MonAdresse.Ville = Nothing
```

## XVI-E-1-e - Utilisez le principe de proximité

Déclarer, initialiser une variable le plus près possible de sa première utilisation, puis **grouper** les instructions dans lesquelles la variable est utilisée.

*Mauvais exemple :* (chaque variable est repérée par une couleur)

```
Dim Prixtotal As Integer
Dim Taux As Integer
Dim Prix As Integer
Dim Quantité As Integer
Dim  Ristourne As Integer
Taux=12
Ristourne=0
Prix=0
Prixtotal= Quantité*Prix
Ristourne=Taux*Quantité
```

Prixtotal n'est pas initialisé, on utilise les variables loin de leur déclaration…

Code dur à lire, source d'erreur.

Bonne manière de faire :

```
Dim Taux As Integer=12
Dim Prix As Integer=0
Dim Quantité As Integer=2
```

```
Dim Prixtotal As Integer=0
Prixtotal= Quantité*Prix

Dim  Ristourne As Integer=0
Ristourne=Taux*Quantité
```

## XVI-E-1-f - Travaillez sur des variables qui restent actives le moins de temps possible

Les instructions utilisant une même variable doivent être si possible regroupées.

Exemple : regardons la variable Acte.

***Mauvaise méthode :*** il y a plusieurs lignes entre 2 lignes contenant la même variable, le cheminement est difficile à suivre !!

```
Dim Acte As Integer
Dim Bib As Integer
Dim C As Integer
C=2
Call Calcul()
Call Affiche()
Acte=Acte+1
Call CalculTaux()
Call AfficheResultat (Acte)
```

***Bonne méthode :*** *le cheminement est clair, les lignes utilisant la variable Acte sont regroupées, la lecture est facile à suivre.*

```
Dim Acte As Integer
Acte=Acte+1
Call AfficheResultat (Acte)
Dim Bib As Integer
Dim C As Integer
C=2
Call Calcul()
Call Affiche()
Call CalculTaux()
```

## XVI-E-1-g - Si vous codez la valeur d'une variable en 'dur', utilisez plutôt des constantes

Une valeur codée en dur est une valeur imposée par le programmeur et écrite sous forme d'un littéral S :

```
I=123  'c'est codé en dur!! Que représente 123?
Écrire:
Const nbMaxLignes As Integer= 123
I=nbMaxLignes 'c'est plus signifiant
```

Autre exemple

Plutôt que :

```
For i= 0 To 120
Next i
```

Écrire :

```
Const  MAXELEMENT As Integer =120

For i= 0 To MAXELEMENT
Next i
```

Cela a tous les avantages : quand on regarde le code, MAXELEMENT est plus explicite que 120.

De plus si on change la valeur de MAXELEMENT, la modification est prise en compte partout dans le code et les boucles fonctionnent toutes bien.

On peut utiliser des '0' (pour démarrer les boucles: **For i= 0 to…)** ou des '1' pour incrémenter des compteurs **(i=i +1),** mais on évitera les autres littéraux.

Utilisez aussi les énumérations.

## XVI-E-1-h - Groupez les instructions liées

Regroupez les instructions faisant référence aux mêmes choses.

***Mauvaise méthode :*** *c'est un peu fouillis*

```
CalculNewTotal()
CalculOldTotal()
AfficheNewTotal()
AfficheOldTotal()
ImprimeNewTotal()
ImprimeOldTotal()
```

***Bonne méthode :***

```
CalculNewTotal()
AfficheNewTotal()
ImprimeNewTotal()

CalculOldTotal()
AfficheOldTotal()
ImprimeOldTotal()
```

## XVI-E-1-i - Réduisez la portée des variables (problème des variables globales)

Une variable '*globale*' est une variable **visible dans la totalité du programme.**

Exemple : créons une variable globale nommée Index

```
Module MonModule
Public Index As Integer
…
End Module
```

À l'inverse une variable '*locale*' est **visible uniquement dans une procédure ou une Classe :**

```
Class Form1
Private Dim Index As Integer
…
End Class
```

Avantages et inconvénients des 2 manières de faire

***Utilisation de variables globales***

Exemple :

```
Public maVariable As Integer

Sub Mescalculs()
… maVariable=12
End Sub
```

Toutes les procédures ont accès à la variable globale.

Les variables globales sont commodes, d'accès facile, il n'y a peu de paramètres à passer quand on appelle une fonction. Mais on prend des risques : n'importe quelle procédure peut modifier la variable globale.

***Utilisation de variables locales***

La variable a une portée limitée, pour qu'une procédure l'utilise, il faut lui fournir en paramètre.

Exemple :

```
Sub Main()
 Dim maVariable As Integer
 Call MesCalculs(maVariable)
End Sub

Sub Mescalculs(V As Integer)
… V=12
End Sub
```

La variable locale aide à la maniabilité, vous restez maître de ce à quoi chaque procédure a accès.

Chaque procédure est autonome. Cela conserve donc une bonne modularité et un bon masquage des informations.

**Autre danger des variables globales que l'on utilise comme paramètre**

Si on a des variables globales et en plus on utilise ces variables comme paramètres d'une procédure, la même variable peut être modifiée en utilisant 2 noms différents :

```
Public maVariable As Integer
Call Mescalculs (maVariable)

Sub Mescalculs(By Ref v As Integer)
… maVariable=12            'maVariable et v sont la même variable c'est dangereux!!
… v=15
End Sub
```

**Utiliser le maximum de variables locales**

(Il sera toujours possible d'augmenter la portée de la variable si nécessaire, plutôt que de la diminuer.)

**Éviter donc les variables globales qui sont visibles dans la totalité du programme.**

**Procédures d'accès aux variables**

Si vous tenez quand même à utiliser des variables globales, utilisez des 'procédures d'accès à ces variables'.

Le programme n'ira pas modifier directement la valeur de la variable globale, mais passera par une Sub ou une Classe qui ira modifier la variable.

Exemple avec un Sub :

```
Public maVariable As Integer
```

Au lieu d'écrire **maVariable=12** ce qui modifie directement l'accès à la variable globale, créer une Sub modifiant cette variable :

```
Sub SetVariable ( Valeur As Integer)
    maVariable=Valeur
End Sub
```

Cela a l'avantage de pouvoir ajouter un contrôle des données, on peut ajouter par exemple des conditions, interdire des valeurs…

Pour modifier la variable globale on écrira **SetVariable(12)**.

De manière plus générale plutôt que de créer un tableau de variables globales du genre :

```
Public Variables(100) As Variable
```

et d'écrire **Variables(50)=485**

créer une procédure **SetVariable()** une procédure **LetVariable()** une procédure **VariableCount()…**

En programmation-objet, on crée une classe contenant les données (tableau, collection), mais 'Privé', on crée aussi des membres de cette classe qui permettent de lire (les accesseurs) ou de modifier (les mutateurs) les données.

## XVI-E-1-j - Les Booléens sont des Booléens

Utiliser une variable Integer pour stocker un Flag dont la valeur ne peut être que 'vrai' ou 'faux' et donner la valeur 0 ou -1 est à proscrire.

Faire :

```
Dim Flag As Boolean
Flag=True
```

(Utiliser uniquement **True** et **False**).

Éviter aussi d'abréger à la mode Booléens ce qui n'en est pas.

```
Dim x,y As Integer
If x And y then  (pour tester si x et y sont = 0)  est à éviter.
```

Faire plutôt :

```
If x<>0 And y <>0
```

## XVI-E-1-k - Utiliser les variables Date pour stocker les dates

Ne pas utiliser de type Double.

```
Dim LaDate As Date
LaDate=Now
```

## XVI-E-2 - Règles de bonne programmation

**Pour faire un code solide, éviter les bugs, avoir une maintenance facile, il faut suivre quelques règles.**

Relire les chapitres :

sur les bonnes procédures ;

sur les bonnes Classes ;

sur le bon code : bonnes variables ;

sur les bons commentaires le code lisible ;

Voici un résumé des points forts.

## XVI-E-2-a - Séparer l'interface utilisateur et l'applicatif

Exemple : un formulaire affiche les enregistrements d'une base de données.

Créer :

- les fenêtres dont le code gère uniquement l'affichage. C'est l'interface utilisateur ou IHM (Interface Homme Machine) ;
- une Classe gérant uniquement l'accès aux bases de données.

Cela facilite la maintenance : si on désire modifier l'interface, on touche aux fenêtres et pas du tout à la Classe base de données.

**Architecture à 3 niveaux**

Elle peut être nécessaire dans certains programmes, les 3 niveaux sont :

- application, interface ;
- logique ;
- données.

Exemple : un formulaire affiche certains enregistrements d'une base de données.

- **l'interface** affiche les enregistrements ;
- les classes ou modules '**logiques**' déterminent les bons enregistrements ;
- les classes ou modules '**données**' vont chercher les données dans la base de données.

Si au lieu de travailler sur une base Access, je travaille sur une base SQLServer, il suffit de réécrire la troisième couche.

Utiliser des design pattern : ce sont des 'patrons', des modèles en programmation-objet.

## XVI-E-2-b - Utiliser le typage fort

Cela consiste à spécifier le type de données pour toute variable, en plus il faut utiliser des variables le plus typées possible.

Éviter le type 'Object'.

Le code en sera plus rapide, le compilateur interceptera des erreurs (cela évitera des erreurs à la compilation).

## XVI-E-2-c - Forcer la déclaration des variables et les conversions explicites

**Option Explicit** étant par défaut à **On**, **toute variable utilisée doit être déclarée.** Conserver cette option. Cela évite les erreurs liées aux variables mal orthographiées.

Si **Option Strict** est sur **On**, seules les conversions de type effectuées explicitement sur les variables seront autorisées. Le mettre sur On.

Voir Strict et Option Explicite.

## XVI-E-2-d - Utiliser des constantes ou des énumérations

**L'usage de constantes facilite les modifications.**

Exemple : un programme gère des utilisateurs.

Faire :

créer une constante contenant le nombre maximum d'utilisateurs :

```
Const NombreUtilisateur= 20
Dim VariableUtilisateur (NombreUtilisateur)  'on utilise NombreUtilisateur et non 20
For i=  0 To NombreUtilisateur-1
Next i
```

Plutôt que :

```
Dim VariableUtilisateur (20)
For i=  0 To 19
Next i
```

Si ultérieurement on veut augmenter le nombre d'utilisateurs possibles à 50, il suffit de changer une seule ligne :

```
Const NombreUtilisateur= 50
```

**Utiliser les constantes VB, c'est plus lisible**

Form1.BorderStyle=2 est à éviter

```
Form1.BorderStyle= vbSizable   c'est mieux
```

## XVI-E-2-e - Vérifier la validité des données que reçoit une Sub une Function ou une Classe

Vous pouvez être optimiste et ne pas tester les paramètres reçus par votre Sub. Les paramètres envoyés seront toujours probablement bons !! Bof un jour vous ferez une erreur, ou un autre n'aura pas compris le type de paramètre à envoyer et cela plantera !!

**Donc, il faut vérifier la validité des paramètres.**

On peut le faire au fur et à mesure de leur utilisation dans le code, **il est préférable de faire toutes les vérifications en début de Sub.**

Vérifier aussi ce que retourne la procédure.

## XVI-E-2-f - Se méfier du passage de paramètres 'par valeur' ou par 'référence'

Par défaut les paramètres sont envoyés 'par valeur' vers une procédure. Aussi, si la variable contenant le paramètre est modifiée, cela ne modifie pas la valeur de la variable de la procédure appelante.

Si on a peur de se tromper utilisons 'ByVal' et 'ByRef' dans l'entête de la Sub ou de la Fonction.

## XVI-E-2-g - Structurez le code, évitez les Goto

Faire de bonnes Sub ou de bonnes classes.

Découper les problèmes en sous-ensembles plus simples et ne faisant chacun qu'une tâche.

Structurer le code en évitant les GoTo.

## XVI-E-2-h - Ne faire aucune confiance à l'utilisateur du logiciel

Si vous demandez à l'utilisateur de saisir un entier entre 1 et 7.

**Vérifier :**

qu'il a tapé quelque chose !! ;

qu'il a tapé une valeur numérique ;

que c'est un entier ;

que c'est supérieur à 0 et inférieur à 8.

**Accorder les moindres privilèges :**

ne permettre de saisir que ce qui est valide.

## XVI-E-2-i - Rendre le code lisible par tous

S'il est lisible par tous, il sera lisible et compréhensible par vous-même dans un an.

S'il est lisible par tous, il sera débogable plus facilement.

Pour améliorer la lisibilité :

- bien choisir le nom des variables ;

- mettre des commentaires ;

- écrire les algorithmes compréhensibles par tous.

# XVI-E-3 - Rendre le code lisible : commentaires, noms de variables

**Pour faire un code lisible :**

- mettre des commentaires ;
- choisir de bons noms de variable ;
- aérer le code.

## XVI-E-3-a - Ajoutez des commentaires

Pour vous, pour les autres.

Au début de chaque routine, Sub, Function, Classe, noter en commentaire ce qu'elle fait et quelles sont les caractéristiques des paramètres :

- le résumé descriptif de la routine, la Sub ou Function ;
- une description de chaque paramètre ;
- la valeur retournée s'il y en a une ;
- une description de toutes les exceptions… ;
- un exemple d'utilisation ;
- une explication sur le fonctionnement de la routine.

Ne pas ajouter de commentaire en fin de ligne (une partie ne sera pas visible), mais plutôt avant la ligne de code. Seule exception où on utilise la fin de ligne : les commentaires après les déclarations de variable.

```
Dim i As Integer    'Variable de boucle
'Parcours du tableau à la recherche de…
For i=0 To 100
….
```

Paradoxalement, trop de commentaires tue le code autant que le manque de commentaires. Pour éviter de tomber dans le tout ou rien, fixons nous quelques règles.

- Commentez le début de chaque Sub, Fonction, Classe.
- Commentez toutes les déclarations de variables.
- Commentez toutes les branches conditionnelles.
- Commentez toutes les boucles.

**Commentaire en XML**

On peut ajouter des commentaires en XML dans VB2005. 


Exemple

*Pour une Sub :* sur une ligne blanche au-dessus de la Sub Calcul, tapez """" (3 """).

ou

*Pour une variable :* curseur sur la variable, bouton droit puis 'Insérer un commentaire' dans le menu.

Un bloc XML "Summary" se crée automatiquement, pour l'exemple de la Sub, ajouter 'Fonction calculant le total'

```
''' <summary>
''' fonction Calculant le total
''' </summary>
''' <param name="custName"></param>
''' <param name="amount"></param>
''' <remarks> </remarks>
Overloads Sub calcul(ByVal custName As String, ByVal amount As Single)
    ' Insert code to access customer record by customer name.
End Sub
```


Quand ensuite on tape le nom de la Sub, le commentaire s'affiche.

De plus Visual Basic génère automatiquement un fichier de documentation XML lorsque vous créez le projet. Ce fichier apparait dans le répertoire de sortie de l'application sous le nom AssemblyName.xml.

## XVI-E-3-b - Choisissez les noms de procédures et de variables avec soin

**On concatène plusieurs mots** pour former un nom de fonction, de variable de Classe…

Il y a 3 manières d'utiliser **les lettres capitales** dans ces mots (on appelle cela la capitalisation !)

- **Pascal Case :** la première lettre de chaque mot est en majuscule :

```
CalculTotal()
```

- **Camel Case :** la première lettre de chaque mot est en majuscule, sauf la première :

```
iNombrePatient
```

- **UpperCase :** toutes les lettres sont en majuscules :

```
MACONSTANTE
```

De plus le nom doit être explicite.

Microsoft propose quelques règles.

- **Sub , Fonctions**
  Utilisez la 'case Pascal' pour les noms de routine (la première lettre de chaque mot est une majuscule).
  Exemple: **CalculTotal()**
  Évitez d'employer des noms difficiles pouvant être interprétés de manière subjective, notamment Analyse() pour une routine par exemple.
  Utilisez les verbe/nom pour une routine : **CalculTotal().**
- **Variables**
  Pour les noms de variables, utilisez la 'case Camel' selon laquelle la première lettre des mots est une majuscule, sauf pour le premier mot.
  Exemple: **iNombrePatient**
  noter ici que la première lettre indique le type de la variable (Integer), elle peut aussi indiquer la portée(**gT**otal pour une variable globale).

Évitez d'employer des noms difficiles pouvant être interprétés de manière subjective, 'YYB8' ou 'flag' pour une variable.
Ajoutez des méthodes de calcul (Min, Max, Total) à la fin d'un nom de variable, si nécessaire.
Les noms de variable booléenne doivent contenir Is qui implique les valeurs True/False, par exemple **IsFileFound**
Évitez d'utiliser des termes tels que 'Flag' lorsque vous nommez des variables d'état (elles sont différentes des variables booléennes, car elles acceptent plus de deux valeurs). Plutôt que documentFlag, utilisez un nom plus descriptif tel que **documentFormatType**.
Même pour une variable à courte durée de vie utilisez un nom significatif. Utilisez des noms de variable d'une seule lettre, par exemple i ou j, pour les index de petite boucle uniquement.

- **Paramètre**
  Pour les noms de paramètres, utilisez la 'case Camel' selon laquelle la première lettre des mots est une majuscule, sauf pour le premier mot.
  Exemple: **typeName**
- **Constantes**
  Utiliser les majuscules pour les constantes: **MYCONSTANTE**
  N'utilisez pas des nombres ou des chaines littérales telles que For i = 1 To 7. Utilisez plutôt des constantes par exemple For i = 1 To DAYSINWEEK, pour simplifier la maintenance et la compréhension.
- **Objet, Classe**
  Pour les noms de Classe, utiliser le case Pascal :
  Exemple **Class MaClasse**
  Idem pour les événements, espace de noms, méthodes :
  Exemple: **System.Drawing, ValueChange…**
  Dans les objets, il ne faut pas inclure des noms de classe dans les noms de propriétés Patient.PatientNom est inutile, utiliser plutôt **Patient.Nom.**
  Les **interfaces** commencent par un I
  Exemple: **IDisposable**
  Pour les variables privées ou protégées d'une classe utilisez le case Camel :
  Exemple: **lastValue** (variable protégée)
  En plus pour les variables privées d'une classe mettre un "_" avant :
  Exemple: **_privateField**
  Pour une variable Public d'une classe :
  Exemple **TotalAge**
- **Tables**
  Pour les tables, utilisez le singulier. Par exemple, utilisez table 'Patient' plutôt que 'Patients'.
  N'incorporez pas le type de données dans le nom d'une colonne.
- **Divers**
  Minimisez l'utilisation d'abréviations,
  Lorsque vous nommez des fonctions, insérez une description de la valeur retournée, notamment **GetCurrentWindowDirectory().**
  Évitez de réutiliser des noms identiques pour divers éléments.
  Évitez l'utilisation d'homonymes et des mots qui entrainent souvent des fautes d'orthographe.
  Évitez d'utiliser des signes typographiques pour identifier des types de données, notamment $ pour les chaines ou % pour les entiers.
  Un nom doit indiquer la signification plutôt que la méthode.

## XVI-E-3-c - Éclaircir, aérer le code

Éviter plusieurs instructions par ligne.

Ajouter quelques lignes blanches.

Décaler à droite le code contenu dans une boucle ou une section If… End If :

Une mise en retrait simplifie la lecture du code, par exemple :

```
If… Then
    If… Then
        …
    Else
        …
    End If
Else
…
End If
```

# XVII - Les bases de données

## XVII-A - Notions sur les bases de données

Comment lire et écrire des informations complexes et structurées ?

### XVII-A-1 - Généralités

Pour travailler avec du texte, des octets, des données très simples (sans nécessité d'index, de classement…), on utilise les fichiers séquentiels, aléatoires, binaires.

Mais dès que les informations sont plus structurées, il faut utiliser les **bases de données** (Data Base en anglais).

Une base de données peut être :

- locale : utilisable sur un ordinateur par un utilisateur ;
- répartie : c'est-à-dire que la base est stockée sur des machines distantes et accessibles par réseau.

Plusieurs utilisateurs peuvent accéder à la base simultanément.

Exemples de type de bases de données :

**Dbase** Format très utilisé, qui date maintenant un peu, les fichiers contenant ces bases ont l'extension .dbf ;

**Paradox** ;

**FileMaker** ;

**FoxPro** ;

**Interbase** ;

**Access** : format très répandu, les fichiers contenant ses bases ont l'extension .mdb ;

**SQLServeur** : les fichiers contenant ses bases ont l'extension .dbo ;

**SyBase** ;

**MySql** ;

**Oracle…**

Pour pouvoir contrôler les données, l'accès à ces données et les utilisateurs utilisant une base de données, **un système de gestion** est nécessaire. La gestion de la base de données se fait grâce à un système appelé **SGBD**

**(système de gestion de bases de données)**, si la base de données est relationnelle (Existence de relation entre les tables) on parle de **SGBDR (système de gestion de bases de données relationnelles)**

**Un SGBD est un logiciel qui joue le rôle d'interface entre les utilisateurs et la base de données.**

Un SGBD permet de décrire, manipuler et interroger les données d'une 'Base de Données'.

## XVII-A-2 - Tables

Dans une base de données, il y a des **tables**.

Une table sert à stocker physiquement des données sous forme d'un tableau comportant des **lignes** (rows) et des **colonnes** (columns).

## XVII-A-3 - Exemple

Une base de données Access nommée Cabinet.mdb contient **les patients d'un cabinet, leurs consultations, les ordonnances, les médicaments…**

Dans cette base il y a plusieurs tables : **une table patient, une table consultation…**

Examinons la table patient.

Sur **chaque ligne (row)**, il y a un patient,.

**Chaque colonne (column)** représente un type de données (première colonne= civilité, seconde=nom, troisième=prénom, quatrième= numéro interne propre à chaque patient. )

L'ancienne terminologie parlait **d'enregistrements** (lignes) et de **champs** (colonnes).

| Champs Civilité | Champ Nom | Champ Prénom | Champ Numéro Interne |
|---|---|---|---|
| M. | Durand | Luc | 1 |
| Mme | Dupont | Josette | 2 |
| M. | Thomas | Guy | 3 |

Ici la seconde ligne (le 2e enregistrement, le second row) contient la civilité, le nom, le prénom, le numéro du patient Dupont Josette.

**Chaque colonne à un type bien défini :** dans notre cas la première colonne contient du texte, ainsi que la seconde, la troisième, la quatrième colonne contient un numérique long par exemple.

Examinons la table consultation

Sur **chaque ligne**, il y a une consultation.

**Chaque colonne** représente un type de données (première colonne= numéro correspondant au patient, seconde=date, troisième=texte de la consultation, quatrième= Courrier).

| Champs Numéro Interne | Champ Date | Champ Texte | Champ Courrier |
|---|---|---|---|
| 1 | 02/12/2003 | Angine | |
| 2 | 02/02/2004 | Hta | |
| 1 | 05/04/2004 | Bronchite | |

Il n'est pas question pour chaque consultation d'enregistrer de nouveau le nom et le prénom du patient, cela enregistrerait 2 fois la même information puisque le nom et le prénom du patient sont déjà dans la table 'patient'. On va donc, pour éviter les redondances, utiliser un numéro interne : chaque patient a un numéro unique (4e champ de la table 'nom'), il suffit de noter dans chaque consultation le numéro du patient.

Ensuite, si je consulte le patient Durand Luc, sachant que son numéro interne est '1', il suffit de rechercher dans la table consultation les consultations dont le premier champ est 1: Durand Luc a 2 consultations.

Le nom de la colonne est souvent nommé en utilisant le terme Id (pas ici) , 'IdPatient' par exemple, synonyme de 'numéro patient', cela permet de repérer les champs 'numéro interne'.

## XVII-A-4 - Type de colonne

Il existe des types de colonnes (de champs) alphanumériques :

- de longueur fixe (pour le champ 'nom', je prévois 30 caractères par exemple) ;
- de longueur variable (champ mémo dans la base Dbase par exemple).

Il existe aussi

- des champs numériques ;
- des champs dates ;
- et dans certaines bases de données des champs booléens, image…

## XVII-A-5 - Clé primaire

Quand il est nécessaire de **différencier chaque enregistrement de manière unique**, il faut définir un champ comme clé primaire.

Ce champ doit être unique pour chaque enregistrement (il ne doit pas y avoir de doublons : 2 enregistrements ne peuvent pas avoir la même clé primaire), et la valeur de la clé primaire ne peut pas être égale à null.

Dans notre exemple de la table patient, on ne peut pas utiliser le champ 'nom' comme clé primaire, car plusieurs patients peuvent avoir le même nom, il est judicieux de choisir le champ 'numéro interne' comme clé primaire, car chaque patient (donc chaque enregistrement) à un numéro interne unique.

Quand on enregistre une nouvelle fiche patient, il faut donc donner un nouveau 'numéro interne' qui n'a jamais été utilisé, en pratique :
il existe des champs numériques dont la valeur s'incrémente automatiquement d'une unité ; quand on crée un nouvel enregistrement, le champ 'numéro interne' prend automatiquement la plus grande valeur déjà présente dans la base +1.

## XVII-A-6 - Index

Un index permet d'**optimiser les recherches** dans une table, de **les rendre beaucoup plus rapides.**

Expliquons.

Si j'ai une table contenant les noms des médecins utilisateurs et que je veux chercher un nom, comme il y a au maximum 5 à 6 médecins dans un cabinet, pour rechercher un nom, il suffit de lire successivement chaque enregistrement et de voir si c'est celui recherché. C'est suffisamment rapide.

Par contre si je recherche dans la table patient un patient, comme il y a 4000 à 8000 enregistrements, je ne peux pas les lire un à un, c'est trop long, aussi je crée un index : c'est comme l'index d'un livre, le nom me donne directement l'endroit où se trouve l'enregistrement correspondant.

On peut combiner plusieurs champs pour en faire la base d'un index.

Pour ma table 'patient', je peux créer un index nommé IndexPatient qui sera indexé sur Nom +Prenom.

Il peut y a voir plusieurs index sur une même table.

Les index accélèrent les recherches, mais s'il y en a trop, cela alourdit le fonctionnement, on ne peut pas tout indexer !!

## XVII-A-7 - Relations entre les tables : différents types de relations

On a déjà vu que 2 tables peuvent être liées et avoir un champ commun présent dans les 2 tables.

Sur ce champ commun, il peut exister plusieurs types de relation :

relation 1 à N ;

relation 1 à 1 ;

relation N à M.

Voyons cela en détail.

## XVII-A-7-a - 1 à N (relation un à plusieurs)

Dans notre exemple la table 'patient' et la table 'consultation' ont chacune un champ numéro interne. Ce qui permet de lier à l'enregistrement du patient de numéro interne X toutes les consultations pour ce patient (elles ont dans leurs champs 'numéro interne' la valeur X).

Comme pour UN patient il peut y avoir N consultations, on parle de relation 1 à N.

**Un enregistrement unique est lié à plusieurs enregistrements de l'autre table par un champ présent dans les 2 tables.**

On remarque que le champ 'numéro interne' du coté patient est une clé primaire, pas du coté consultation.

Table 'patient'

| | | | Champ Numéro Interne |
|---|---|---|---|
| M. | Durand | Luc | 1 |
| Mme | Dupont | Josette | 2 |
| M. | Thomas | Guy | 3 |

Table 'consultations'

| Champ Numéro Interne | | | |
|---|---|---|---|
| 1 | 02/12/2003 | Angine | |
| 2 | 02/02/2004 | Hta | |
| 1 | 05/04/2004 | Bronchite | |

Le patient Durand Luc a 2 consultations : le 02/12/2003 et le 05/04/2004 (Le numéro interne de ce patient est 1, mais l'utilisateur final n'a pas à le savoir ni à le gérer: la relation utilisant le numéro interne est **transparente** pour l'utilisateur final).

## XVII-A-7-b - 1 à 1

**Un enregistrement unique est lié à un autre enregistrement unique par un champ présent dans les 2 tables.**

On peut imaginer dans notre exemple, créer une table Antécédents contenant aussi un champ numéro interne, pour chaque enregistrement de la table patient, il y a un enregistrement unique dans la table Antécédents, de même numéro interne contenant les antécédents du patient.

Table 'patient'

| | | | |
|---|---|---|---|
| M. | Durand | Luc | 1 |
| Mme | Dupont | Josette | 2 |
| M. | Thomas | Guy | 3 |

Table 'antécédents'

| Champ Numéro interne. | | |
|---|---|---|
| 1 | 02/01/2003 | appendicite |
| 2 | 02/02/2004 | Hta |
| 3 | 05/05/2004 | Cancer du colon |

## XVII-A-7-c - N à M

**Relation plusieurs à plusieurs. Plusieurs enregistrements de la première table peuvent être liés à plusieurs de la seconde table et vice versa.**

Exemple

J'ai une table 'ordonnances' qui peut contenir plusieurs médicaments, et une table 'médicaments' dont les médicaments peuvent être utilisés dans plusieurs ordonnances différentes.

Il faut dans ce cas avoir la table 'ordonnances' avec une clé primaire sur un numéro d'ordonnance (numéro d'ordonnance unique s'incrémentant à chaque nouvelle ordonnance), une table 'médicaments' avec une clé primaire sur le numéro unique du médicament, et **créer une troisième table gérant la relation ordonnance-médicament.**

**Table 'Ordonnances'**

| 'Numéro ordonnance' | 'Numéro Interne patient' | Champ' date' |
|---|---|---|
| 1 | 2 | 02/05/2002 |
| 2 | 3 | 02/04/2003 |
| 3 | 2 | 06/05/2004 |

**Table 'Médicaments'**

| 'Numéro médicament' | 'Libelle médicament' | Code CIP' |
|---|---|---|
| 1 | Amoxicilline | 65897 |
| 2 | Omeprazone | 66589 |
| 3 | Allopurinol | 78456 |

**Table supplémentaire 'Contenu ordonnance'**

| 'Numéro ordonnance' | 'Numéro médicament' |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 2 | 2 |

Ici le patient de numéro interne 2 (Dupont Josette) a une ordonnance visible dans la table 'Ordonnances' (numéro interne : 2, numéro de l'ordonnance : 1). Si on cherche dans la table 'Contenu ordonnance' (Index créé sur le numéro d'ordonnance) on retrouve 2 enregistrements (ayant un numéro d'ordonnance 1), on constate que l'ordonnance contient les médicaments 1 et 2 qui correspondent (table 'médicaments') à de l'amoxicilline et de l'oméprazone.

On remarque qu'une ordonnance peut avoir autant de médicaments que l'on veut.

## XVII-A-7-d - Relation N à M avec N fixe et petit

Dernier cas non décrit dans les livres.

J'explique : si chaque ordonnance a au maximum 3 médicaments (que la sécu serait contente si c'était vrai !!), il est possible de créer une table 'ordonnances' contenant 3 champs médicaments. Dans ce cas on se passe de la 3e table.

**Table 'Ordonnances'**

| 'Numéro ordonnance' | 'Numéro Interne patient' | Champ' date' | 3 champs médicaments | | |
|---|---|---|---|---|---|
| 1 | 2 | 02/05/2002 | 1 | 2 | 3 |
| 2 | 3 | 02/04/2003 | | | |
| 3 | 2 | 06/05/2004 | | | |

**Table 'Médicaments'**

| 'Numéro médicament' | 'Libelle médicament' | Code CIP' | |
|---|---|---|---|
| 1 | Amoxicilline | 65897 | |
| 2 | Omeprazone | 66589 | |
| 3 | Allopurinol | 78456 | |

## XVII-A-8 - Contraintes

Un champ peut avoir certaines contraintes :

- on peut **interdire la valeur Null :** cela empêche d'enregistrer un champ vide. On peut aussi donner une valeur par défaut ;
- on peut **empêcher les doublons** ;
- on peut exiger l'**intégrité référentielle :** la valeur d'un champ doit exister dans le champ d'une autre table. (On ne peut pas enregistrer une consultation pour le patient de numéro interne 2000 s'il n'existe pas de fiche patient ayant le numéro 2000.) ;
- on peut exiger des règles de validation pour un champ : interdire les valeurs négatives par exemple.

## XVII-A-9 - Serveur de fichier, Client serveur

Si plusieurs utilisateurs sont connectés à une base Access à travers un réseau, chaque utilisateur a sur son poste un 'moteur' Access, qui récupère l'ensemble des données à utiliser et qui les traite en local.

On parle de **serveur de fichier.**

Le moteur d'accès est présent sur chaque poste.

Si plusieurs utilisateurs sont connectés à une base SQLServer : la base est sur le serveur avec le logiciel SQLServeur.

Un logiciel utilisateur situé sur un autre ordinateur (le client) envoie au serveur une requête.

Le logiciel SQLServer traite la requête sur le serveur et retourne au logiciel client uniquement le résultat de la requête.

On parle d'**architecture Client-serveur.**

Le moteur d'accès est présent uniquement sur le serveur.

Si on cherche un enregistrement parmi 60 000 enregistrements, en serveur de fichiers, les 60 000 enregistrements sont envoyées par le réseau vers le moteur Access de l'utilisateur, le moteur les traite pour en sortir un.

En client serveur, le logiciel utilisateur envoie une requête au serveur, le logiciel serveur cherche sur le serveur dans la base l'enregistrement, il le trouve et envoie à travers le réseau vers le logiciel client uniquement un enregistrement.

## XVII-A-10 - Opérations sur les enregistrements

De manière générale, on peut :

**Ouvrir** une base de données (Open) ;

**Ajouter** un enregistrement (Add) ;

**Effacer** un enregistrement (Delete) ;

**Modifier** un enregistrement (Update) ;

**Chercher** un ou des enregistrements ;

**Fermer** la base (Close).

**Avant**

Il y a bien longtemps, on choisissait un index, on cherchait un enregistrement (avec Seek), on le lisait, le modifiait, on avançait (MoveNext) ou on reculait (MovePrevious) d'un enregistrement dans la base, mais c'est de l'histoire ancienne !!

Chaque type de base de données avait ses propres commandes.

Ensuite , il y a eu les RecordSet, sorte de tableau avec un curseur pointant un élément.

Avec ADO.NET

Maintenant quelle que soit la base de données, on utilise un langage unique : **le 'SQL' pour faire une requête** sur la base de données : extraction de certains enregistrements ou de certains champs **en fonction de critères**, le résultat (un ensemble d'enregistrements ou de champs) se retrouvant dans un **DataSet** (sorte de tableau situé en local sur lequel on lit on modifie, on ajoute ou on enlève des lignes, les modifications étant répercutées sur la base de données du départ).

## XVII-B - Généralités sur ADO.NET

Jusqu'en vb6 on utilisait DAO et ADO pour travailler sur les bases de données. En vb.Net on utilise ADO.NET.

**Comment donc travailler sur les Bases de données en VB.NET ? Avec ADO.NET**

## XVII-B-1 - Généralités

Pour avoir accès à partir de VB.NET aux bases de données, il faut utiliser ADO.NET.

ADO veut dire **Activex Database Objet**.

C'est la couche d'accès aux bases de données, le SGBD (Système de Gestion de Base de Données) de VB.

ADO.NET est ".NET" donc **managé** et géré par le CLR.

**Il est indépendant de la base de données :** alors qu'initialement chaque type de gestionnaire de base de données avait ses instructions, sa manière de fonctionner, ADO.NET à un langage unique pour ouvrir, interroger, modifier une base de données, quelle que soit la base de données.

**Le langage de requête est le SQL.**

En VB il y a **2 manières d'écrire un programme qui utilise une base de données :**

**- écrire du code** pour créer des objets Ado.net, écrire du code pour ouvrir la base, créer la liaison entre la base et un DataSet avec des critères de sélection écrits en SQL ;
**- utiliser l' 'Assistant de configuration de source de base de données'** qui crée les objets et le code à votre place. On peut même lier une table à une liste par exemple qui sera 'remplie' automatiquement par la table (on parle de Binding).

## XVII-B-2 - Les Managed Providers

Pour avoir accès aux données, il faut charger les **DRIVERS** (ou providers).

Comme d'habitude, il faut :

- charger les références des drivers (les Dll) ;
- importer les espaces de nom.

Ainsi on a accès aux objets Ado.Net correspondant.

Voyons cela.

- **OLE DB** Managed Provider est fourni dans 'System', après avoir importé le NameSpace System.Data.OLEDB, on peut travailler sur des bases Access par exemple.
- **SQL Server** Managed Provider est fourni dans 'System', après avoir importé le NameSpace System.Data.SqlClient, on peut travailler sur des bases SqlServeur.
- Un composant **ODBC** et un composant **ORACLE** sont disponibles sur le site MSDN, il faudra charger la référence de la Dll puis le NameSpace.
- Pour travailler sur une base **MySQL** lisez le très bon didacticiel **MySQLDotNet** (sur developpez.com bien sûr), il utilise le Managed Provider**ByteFX**.

Exemple, pour travailler sur une base Access, il faudra taper :

```
Imports System.Data.OLEDB
```

Avec Visual Basic Express 2010, vous pouvez accéder à trois types de bases de données : Microsoft SQL Server Compact Edition, Microsoft SQL Server Express ou Microsoft Access.

## XVII-B-3 - Les Objets ADO.NET

Il faut disposer d'un objet **Connexion** pour avoir accès à la base de données, on met dans la propriété ConnectionString les paramètres de la base de données (nom de la base de données, chemin, mot de passe…).

En fonction de la BD les paramètres sont différents. Un site nommé ConnetionStrings.com donne une mine de renseignements pour écrire les paramètres de connexion pour une BD.

Avec la méthode **Open** on ouvre la base.

On peut ensuite travailler de 2 manières:

*A- On envoie une requête Sql 'SELECT' à la base, on récupère le résultat dans un objet.*

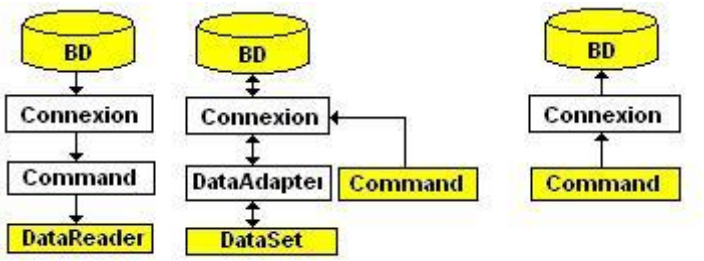
- Avec un objet **DataReader on extrait les données en lecture seule :** une requête SQL (sur un objet command) charge le DataReader. C'est rapide, on peut lire uniquement les données et aller à l'enregistrement suivant. Il travaille en mode connecté. Pour gérer un DataReader on a besoin d'un objet **Command.**
- Avec un objet **DataSet on manipule les données :** une requête SQL (sur un objet command) charge le DataSet avec des enregistrements ou des champs, on travaille sur les lignes et colonnes du DataSet en local, en mode déconnecté(une fois que le DataSet est chargé, la connexion à la base de données est libérée). Pour alimenter un DataSet on a besoin d'un objet DataAdapter qui fait l'intermédiaire entre la BD et le DataSet.

*B- On manipule directement la base (sans retour de résultats).*

- Avec un objet **Command** on peut manipuler directement la BD (en SQL avec UPDATE, INSERT, DELETE CREATE DROP…), on utilise la propriété **ExecuteNonQuery** pour cela.

Avec la méthode Close on ferme la base.

Résumons les différents objets nécessaires pour travailler sur une BD :




Noter bien le sens des flèches :

- le **DataReader** est en lecture seule, les données lues dans la BD sont accessibles dans le DataReader ;
- le **DataSet** peut lire et écrire des données dans la BD, il faut un DataAdapter en plus de la connexion ;
- l'objet **Command** peut modifier la BD.

Ce schéma souligne aussi les objets intermédiaires nécessaires :

- un objet **connexion** dans tous les cas ;
- un objet **Command** pour le **DataReader** :
- un objet **DataAdapter** plus un objet **Command** pour le **DataSet**.

L'objet **Command** permet d'envoyer des ordres en SQL à la BD et de la modifier, il permet aussi, quand on utilise un DataSet, d'envoyer une requête SELECT en SQL afin de remplir le DataSet avec le résultat de la requête.

Enfin certains contrôles comme les DataGrid, les ListBox par exemple peuvent afficher des données à partir d'un DataSet.

Pour mettre à jour la base après modification du DataSet ou de la Grid il faut un objet **CommandBuilder**.

Mode connecté ou déconnecté :

- le **DataReader** fonctionne en mode connecté,la connexion entre la BD et le DataReader est ouverte tant que le DataReader fonctionne ;
- le **DataSet** peut travailler en mode déconnecté : on ouvre la connexion, on charge le DataSet, on ferme la connexion (il faut le faire, ce n'est pas automatique), on travaille sur le DataSet, on peut le rouvrir plus tard pour les mises à jour.

Remarque : en fonction du provider, **le nom des objets change.**

Avec le provider OleDb, après Imports System.Data.OleDb

on utilisera OleDbConnexion, OleDbAdapter…

Avec le provider SQL, après Imports System.Data.SqlClient

on utilisera SqlConnexion, SqlAdapter…

Un site nommé **ConnetionStrings.com** donne une mine de renseignements pour écrire les paramètres pour une BD.

## XVII-B-4 - Le DataReader

Le DataReader permet donc de lire très rapidement une table, enregistrement par enregistrement, du début à la fin.

Il n'y a pas possibilité d'écrire dans la base.

## XVII-B-5 - Le DataSet

Le DataSet a la structure d'une base de données, mais en local, il contient :

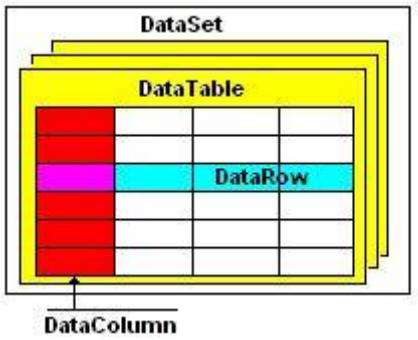
des DataTable qui contiennent des DataRow et des DataColumn.

Pour utiliser DataSet, DataTable, DataRow… il faut importer l'espace de noms Data :

```
Imports System.Data.
```

On peut créer un Dataset de toutes pièces, mais la plupart du temps, on charge le DataSet à partir d'une base de données.

Une requête SQL charge le DataSet, on travaille sur les lignes et colonnes du DataSet en local (sur des enregistrements ou des champs), en mode déconnecté (une fois que le DataSet est chargé, la connexion à la base de données peut être libérée).




La structure de données du DataSet reflétera automatiquement et exactement celle des données extraites. Si j'extrais 2 colonnes de données avec l'instruction Sql fournie à l'objet Command, le DataSet aura une table (DataTable) de 2 colonnes avec les données extraites.

Exemple

Avec ADO.NET je lance une requête SQL demandant toutes les fiches de la table 'nom' dont le champ 'prénom' est 'Philippe', je récupère un DataSet local contenant toutes les fiches (le DataColumn "Prénom" ne contient que des 'Philippe'). Je peux modifier en local le DataSet, (modifier une date de naissance par exemple) et mettre à jour automatiquement la base de données distante.

Pour être complet, il existe aussi les **DataView** qui représentent **une vue d'un DataTable**. (Un DataView peut contenir un champ d'une table d'un DataSet, ou les enregistrements répondant à un critère.)

# XVII-C - Syntaxe SQL (Généralités)

**Comment adresser une requête vers une Base de données de ADO.NET ? Avec SQL**

Ici on va voir des généralités sur le langage SQL, on utilisera ce langage dans **les requêtes** ADO.net . En Ado.Net on verra qu'on exécute ces requêtes par la méthode 'ExecuteNoQuery' d'un objet Command. On verra plus bas qu'il est aussi possible d'interroger une base de données grâce à LINQ qui permet d'interroger en VB avec une syntaxe proche de SQL.

## XVII-C-1 - Généralités

SQL veut dire **Structured Query Language : Langage d'interrogation structurée.**

SQL grâce au couplage avec un SGBD relationnelle permet un **traitement interactif des requêtes.**

SQL est le langage unique qui permet de décrire, manipuler, contrôler l'accès et interroger les bases de données relationnelles.

C'est un langage déclaratif, qui est régi par une norme (ANSI/ISO) qui assure la portabilité du langage sur différentes plateformes aussi bien matérielles que logicielles. Une commande SQL écrite dans un environnement Windows sous ACCESS peut, souvent sans modification, être utilisée directement dans un environnement ORACLE sous Unix…

SQL est utilisé dans ADO.NET pour manipuler **toutes** les bases de données.

## XVII-C-2 - Les commandes SQL

| Catégorie | Commandes SQL | |
|---|---|---|
| *Manipulation des tables* | CREATE | Création de tables |
| | ALTER | Modification de tables |
| | DROP | Suppression de tables |
| | | |
| *Manipulation des données* | INSERT | Insertion de lignes dans une table |
| | UPDATE | Mise à jour de lignes dans une table |
| | DELETE | Suppression de lignes dans une table |
| | | |
| *Contrôle des données* | GRANT | Attribution de droits d'accès |
| | REVOKE | Suppression de droits d'accès |
| | COMMIT | Prise en compte des mises à jour |
| | ROLLBACK | Suppression des mises à jour |
| | | |
| *Interrogation des données* | SELECT | Interrogations diverses |

# XVII-C-3 - SELECT : Interrogation

**Permet d'extraire, de sélectionner des données.**

Syntaxe simplifiée :

**SELECT champ FROM table WHERE condition**

**Dans la table 'table' sélectionner les enregistrements vérifiant la condition 'condition' et en afficher les champs 'champs'**

**Exemple**

| | |
|---|---|
| SELECT | Précise les colonnes qui vont apparaître dans la réponse |
| FROM | Précise la (ou les) table intervenant dans l'interrogation |
| WHERE | Précise les conditions à appliquer sur les enregistrements. On peut utiliser :<br>- Des comparateurs : =, >, <, >=, <=,<><br>- Des opérateurs logiques : AND, OR, NOT<br>- Les prédicats : IN, LIKE, NULL, ALL, SOME, ANY, EXISTS... |
| GROUP BY | Précise la (ou les) colonne de regroupement |
| HAVING | Précise la (ou les) conditions associées à un regroupement |
| ORDER BY | Précise l'ordre dans lequel vont apparaître les lignes de la réponse :<br>- ASC : En ordre ascendant (par défaut)<br>- DESC: En ordre descendant |

Exemple:

Soit la table patient:

| Champs 'Civilité' | Champ 'Nom' | Champ 'Prénom' | Champ Num Int | Datenais | Sexe |
|---|---|---|---|---|---|
| M. | Durand | Luc | 1 | 12/02/1952 | M |
| Mme | Dupont | Josette | 2 | 06/04/1936 | F |
| M. | Thomas | Guy | 3 | 08/02/1980 | M |

SELECT Nom FROM Patient

Cela signifie : dans la table Patient extraire les champs 'nom'

| |
|---|
| Durand |
| Dupont |
| Thomas |

SELECT Nom FROM Patient WHERE Prenom='Luc'

WHERE ajoute un critère de sélection.

Cela signifie : dans la table Patient extraire le champ Nom de tous les enregistrements dont le prénom est "Luc" .

| Durand |
|---|

SELECT Nom, Prenom FROM Patient WHERE Sexe='F'

Cela signifie : dans la table Patient extraire le champ Nom et prénom de tous les enregistrements dont le champ sexe est "F"( F comme féminin).

Dans l'exemple on obtient :

| Dupont | Josette |
|---|---|

SELECT * FROM Patient WHERE Datenais>=#01/01/1950#

Cela signifie : dans la table Patient extraire tous les champs de tous les enregistrements dont le champ date de naissance est supérieur ou égal à 01/01/1950 .

Dans l'exemple on obtient :

| | | | | | |
|---|---|---|---|---|---|
| M. | Durand | Luc | 1 | 12/02/1952 | M |
| M. | Thomas | Guy | 3 | 08/02/1980 | M |

On remarque que

*** signifie : extraire tous les champs.**

**Pour utiliser les dates, il faut les entourer de "#"**.

**Les dates sont au format mm/jj/aaaa**

SELECT * FROM Patient WHERE Datenais>= #01/01/1950# AND Datenais<= #01/01/1980#

Cela signifie : dans la table Patient extraire tous les champs de tous les enregistrements dont le champ date de naissance est supérieur ou égal à 01/01/1950 et inférieur ou égal à 01/01/1980.

On remarque qu'on peut utiliser avec Where, les opérandes

**AND OR NOT.**

**Il est bien sûr possible de combiner des conditions sur des champs différents :**

**Sexe='M' AND Prenom='Luc"**

SELECT * FROM Patient WHERE BETWEEN #01/01/1950# AND #01/01/1980#

Même signification que le précèdent, mais en utilisant BETWEN AND qui est plus performant.

SELECT Nom FROM Patient WHERE Prenom IN ('Luc', 'Pierre', 'Paul')

Cela signifie qu'il faut extraire les enregistrements dont le prénom est Luc, Pierre ou Paul .

SELECT Nom FROM Patient WHERE Prenom LIKE 'D%'

Cela signifie qu'il faut extraire les enregistrements dont le prénom commence par un 'D'.

LIKE recherche des chaines de caractères avec l'aide de caractères génériques :

% représente une chaine de caractères même vide.

_ représente un caractère.

On peut spécifier une série de caractères en les mettant entre ""

Exemple :

LIKE 'D%' commence par D

LIKE '%D%' contient D

LIKE '[DF]%' commence par D ou F

LIKE '___' contient 3 caractères

SELECT Nom FROM Patient WHERE SEXE IS NULL

Cela signifie qu'il faut extraire les enregistrements dont le sexe n'a pas été enregistré.

SELECT DISTINCT Nom FROM Patient WHERE SEXE IS NULL

DISTINCT permet d'éviter les doublons

Si dans les Noms extraits, il y a 2 fois le même (2 membres d'une même famille) , il n'en est gardé qu'un.

## XVII-C-4 - Tri des enregistrements

ORDER BY sert à trier les enregistrements.

Il est placé à la fin.

DESC sert à trier par ordre décroissant.

SELECT Nom, Prenom , Sexe, DateNais FROM Patient WHERE Sexe='F' ORDER BY DateNais

Trie les enregistrements de sexe 'F' par date de naissance

SELECT Nom, Prenom, DatNais, NumInt FROM Patient WHERE Sexe='F' ORDER BY DateNais DESC, NumInt

Trie les enregistrements de sexe 'F' par date de naissance, mais décroissante et pour une même date de naissance par numéro interne croissant.

## XVII-C-5 - Statistiques

SELECT COUNT(*) AS NombrePatient FROM Patient

Compte le nombre total d'enregistrements dans la table Patient et met le résultat dans le champ NombrePatient

On peut aussi utiliser :

MIN retourner la plus petite valeur ;

MAX retourner la plus grande valeur ;

SUM retourner la somme ;

AVG retourner la moyenne ;

VAR retourner la variance ;

STDEV retourner l'écart type.

SELECT Prenom ,COUNT(*) AS NombrePrenom FROM Patient GROUP BY Prenom

Extrait la liste des prénoms avec le nombre de fois que le prénom est utilisé.

GROUP BY regroupe les enregistrements par valeur.

SELECT Prenom ,COUNT(*) AS NombrePrenom FROM Patient GROUP BY Prenom HAVING CONT(*)>3

Extrait la liste des prénoms avec le nombre de fois que le prénom est utilisé. S'il est utilisé plus de 3 fois…

HAVING rajoute un critère au regroupement.

## XVII-C-6 - Extraction de données sur plusieurs tables

Parfois on a besoin d'extraire des champs de plusieurs tables différentes, mais ayant une relation (un champ commun), pour cela on utilise **une jointure.**

**Pour chaque enregistrement de la première table, on affiche en regard les enregistrements de la 2e table qui ont la même valeur de jointure.**

Exemple

**Soit la table patient**

| Champs 'Civilité' | Champ 'Nom' | Champ 'Prénom' | Numéro Ville | Datenais | Sexe |
|---|---|---|---|---|---|
| M. | Durand | Luc | 1 | 12/02/1952 | M |
| Mme | Dupont | Josette | 2 | 06/04/1936 | F |
| M. | Thomas | Guy | 3 | 08/02/1980 | M |

**Soit la table Ville:**

| Nom ville | Numéro ville |
|---|---|
| Paris | 1 |
| Lyon | 2 |
| Marseille | 3 |

Comment récupérer les champs Nom et ville (pas le numéro)?

SELECT Patient.Nom, Ville.NomVille From Patient INNER JOIN Ville ON Patient.NuméroVille= Ville.NuméroVille

On obtient:

| | |
|---|---|
| Durand | Paris |
| Dupont | Lyon |
| Thomas | Paris |

En ADO.Net, on verra qu'on passe la chaine SQL à un objet Command.

Ces données extraites à partir d'une base de données grâce à l'instruction SQL vont se retrouver dans un DataSet (sorte de Bd en local, en mémoire).

## XVII-C-7 - Ajout, suppression, modification d'enregistrement

Il est impossible d'insérer, de modifier ou de supprimer dans plusieurs tables simultanément.

Une requête de mise à jour (INSERT, UPDATE ou SELECT) est une transaction, tout doit être réalisé, sinon rien ne se passe (en particulier si une seule donnée viole une contrainte, toutes les opérations sont annulées).

Insertion d'enregistrement : syntaxe :

INSERT [INTO] nomdelatable [(listedescolonnes)]

{VALUES (listedesvaleurs) | requêteselect | DEFAULT VALUES }

Exemple

INSERT INTO TablePatient (Civilité, Nom, Prenom, NumeroVille, Datenais, Sex)

VALUES ('M', 'Dupont', 'Pierre', '2',' 02/12/51/, 'M')

Effacement d'enregistrement : syntaxe :

DELETE [FROM] nomtable [WHERE condition]

Exemple

DELETE FROM Patient WHERE Nom LIK'%d'

Mise à jour d'enregistrement : syntaxe :

UPDATE nomtable SET colonne1 = valeur1 , colonne2 = valeur2 … WHERE condition

WHERE condition est facultatif et permet de sélectionner les enregistrements correspondant

à un critère et de modifier ceux-là.

En Ado.Net on verra qu'il y a 2 méthodes pour modifier ajouter, supprimer un enregistrement:

- On peut effectuer directement les commandes en SQL : on exécute ces commandes par la méthode ExecuteNoQuery d'un objet Command.
- Il est plus simple de lier la base à un DataSet, de modifier le DataSet et de mettre à jour la base par un Update (du DataAdapter). Dans ce cas, on n'écrit pas de commande SQL.

## XVII-C-8 - Ajout de table

On utilise CREATE TABLE puis le nom de la table, on ajoute les différents champs (entre parenthèses et séparés par des virgules) avec pour chaque champ les conditions (NOT NULL…) et le type.

CREATE TABLE PARENT (CLI_ID INTEGER NOT NULL PRIMARY KEY, CLI_NOM CHAR(32) NOT NULL, CLI_PRENOM VARCHAR(32))

En Ado.Net on verra qu'on exécute ces commandes par la méthode ExecuteNoQuery d'un objet Command.

**Pour aller plus loin**

Série d'articles sur SQL chez developpez.com :

**https://sql.developpez.com/**

# XVII-D - ADO : Lire rapidement en lecture seule : le DataReader

**Comment lire rapidement des enregistrements ? Avec un DataReader.**

**Comment compter les enregistrements ? Avec ExecuteScalar.**

**Étudions en détail les objets Connexion, Command, DataReader.**

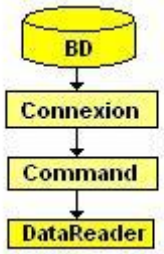
**Et s'il y a une erreur ?**

## XVII-D-1 - Généralités

Un objet **DataReader** fournit des données en lecture seule en un temps record. La seule possibilité est de se déplacer en avant.

En contrepartie de sa rapidité, il monopolise la connexion.

Il faut créer un objet **Connexion** puis un objet **Command**, ensuite on exécute la propriété **ExecuteReader** pour créer l'objet **DataReader ;** enfin on parcourt les enregistrements avec la méthode **Read.**



## XVII-D-2 - Exemple de DataReader avec une base Access

Il existe une base Access nommée 'consultation.mdb', elle contient une table 'QUESTIONS', dans cette table existe un champ 'NOM', je veux récupérer tous les noms et les afficher rapidement dans une ListBox.

Il faut **importer les NameSpaces** nécessaires.

```
Imports System.Data

Imports System.Data.OleDb
```

Il faut créer **un objet connexion :**

```
Dim MyConnexion As OleDbConnection = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
 source=" & _

"C:\consultation.mdb")
```

Il faut donner les paramètres Provider= et Data source=

Dans le cas d'une base Access le provider (le moteur à utiliser est le moteur OLEDB Jet 4.

Il faut créer **un objet Command :**

```
Dim Mycommand As OleDbCommand = MyConnexion.CreateCommand()
```

Il faut **donner dans la propriété CommandText la requête SQL** permettant d'extraire ce que l'on désire.

```
Mycommand.CommandText = "SELECT NOM FROM QUESTIONS"
```

Ici dans la table QUESTIONS, on extrait le champ NOM

Il faut **ouvrir la connexion :**

```
MyConnexion.Open()
```

Il faut **créer un objet DataReader :**

```
Dim myReader As OleDbDataReader = Mycommand.ExecuteReader()
```

On crée une boucle permettant de **lire les enregistrements les uns après les autres**, on récupère le champ (0) qui est une String, on la met dans la ListBox :

```
Do While myReader.Read()

ListBox1.Items.Add(myReader.GetString(0))

Loop
```

Remarquons que le champ récupéré est typé (ici une string grâce à GetString), il y a d'autres types : **GetDateTime, GetDouble, GetGuid, GetInt32, GetBoolean, GetChar, GetFloat, GetByte, GetDecimal etc. ;** il est possible de récupérer sans typage : on aurait écrit myReader(0) ou utiliser **GetValue** (on récupère un objet).

**Read** avance la lecture des données à l'enregistrement suivant, il retourne True s'il y a encore des enregistrements et False quand il est en fin de fichier, cela est utilisé pour sortir de la boucle Do Loop.

On ferme pour ne pas monopoliser.

```
myReader.Close()

MyConnexion.Close()
```

On aurait pu libérer la connexion automatiquement dès la lecture terminée en ajoutant un argument :

```
Dim myReader As OleDbDataReader = Mycommand.ExecuteReader(CommandBehavior.CloseConnection)
```

Simple, non !! (Je rigole !!)

**Cela donne :**

```
Imports System

Imports System.Data

Imports System.Data.OleDb

Imports Microsoft.VisualBasic

Public Class Form1

Inherits System.Windows.Forms.Form

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
 MyBase.Load

Dim MyConnexion As OleDbConnection = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
 source=" & _

"C:\consultation.mdb")

Dim Mycommand As OleDbCommand = MyConnexion.CreateCommand()

Mycommand.CommandText = "SELECT NOM FROM QUESTIONS"

MyConnexion.Open()

Dim myReader As OleDbDataReader = Mycommand.ExecuteReader()

Do While myReader.Read()

ListBox1.Items.Add(myReader.GetString(0))

Loop

myReader.Close()

MyConnexion.Close()

End Sub

End Class
```

Dans le cas d'une **base SQLSERVEUR**, on aurait les changements suivants :

```
Imports System.Data.SqlClient

Dim MyConnexion As SqlConnection = New SqlConnection("Data Source=localhost;" & _
"Integrated Security=SSPI;Initial Catalog=northwind")
Dim Mycommand As SqlCommand = MyConnexion.CreateCommand()

Dim myReader As SqlDataReader = Mycommand.ExecuteReader()
```

## XVII-D-3 - Comment compter ?

Avec ExecuteScalar de l'objet Command on peut récupérer les résultats d'une requête Sql qui contient une instruction COUNT (comptage) AVG (moyenne) MIN (valeur minimum) MAX (valeur maximum) SUM (somme).

Exemple : compter le nombre de questions :

```
Mycommand.CommandText = "SELECT COUNT(*) FROM QUESTIONS"

MyConnexion.Open()

Dim iResultat As Integer = Mycommand.ExecuteScalar()
```

Voyons en détail les objets utilisés.

## XVII-D-4 - L'objet Connection (détails)

Il permet de définir une connexion.

Il faut donner les paramètres 'Provider='indiquant le moteur de BD et 'Data source=' indiquant le chemin et le nom de la BD. Il peut être nécessaire de donner aussi 'Password=' le mot de passe, 'User ID=' Admin pour l'administrateur par exemple.

Il faut mettre ces paramètres avec le constructeur, ou dans la propriété **ConnectionString**.

Dans le cas d'une base Access le provider (le moteur à utiliser) est le moteur OLEDB Jet 4.

Il y a d'autres propriétés.

State état de la connexion (Open, Close, Connecting, Executing, Fetching, Broken)

**Open, Close**

**ConnectionTimeOut,**

**BeginTransaction,**

**…**

## XVII-D-5 - L'objet Command (détails)

Pour chaque provider il y a un objet Command spécifique :

**SqlCommand, OleDbCommand**, mais tous les objets 'Command' on la même interface, les mêmes membres ;

**CommandType** indique comment doit être traitée la commande **CommandText :**

- Text : par défaut (exécution directe des instructions SQL contenues dans CommandText),
- StoredProcedure : exécution de procédure stockée dont le nom est dans CommandText,
- Tabledirect.

Command permet de définir la commande à exécuter : SELECT UPDATE, INSERT, DELETE, en utilisant le membre CommandText.

(Seul SELECT retourne des données.)

**CommandTimeOut** indique la durée en secondes avant qu'une requête qui plante soit abandonnée.

**ExecuteReader** exécute le code SQL de CommandText et retourne un DataReader.

**ExecuteScalar** fait de même, mais retourne la première colonne de la première ligne.

**ExecuteNoQuery** exécute le code SQL de CommandText (généralement une mise à jour par UPDATE, INSERT, DELETE ou un ajout de table) sans retourner de données.

## XVII-D-6 - L'objet DataReader (détails)

Pour chaque provider, il y a un objet 'DataReader' spécifique :

**SqlDatReader, OleDbDataReader**

On a vu le membre **Read** qui avance la lecture des données à l'enregistrement suivant, il retourne True s'il y a encore des enregistrements et False quand il est en fin de fichier.

On a vu **GetDateTime, GetDouble, GetGuid, GetInt32, GetBoolean, GetChar, GetFloat, GetByte, GetDecimal** permettant de récupérer les valeurs typées des champs.

Il a bien d'autres propriétés :

**GetName** retourne le nom du champ (numéro du champ en paramètre) ;

**GetOrdinal** fait l'inverse : retourne le numéro du champ (nom en paramètre) ;

**FieldCount** retourne le nombre de colonnes ;

**GetDataTypeName** retourne le nom du type de données. ;

**IsDBNull** retourne True si le champ est vide ;

**…**

## XVII-D-7 - Exceptions

Chaque SGDB a des erreurs spécifiques. Pour les détecter, il faut utiliser les types d'erreurs spécifiques : **SqlException** et **OleDbException** par exemple :

```
Try

    MyConnection.Open()

Catch ex  As OleDbException

    MsgBox(ex.Message)

End Try
```

# XVII-E - ADO : Travailler sur un groupe de données : le DataSet

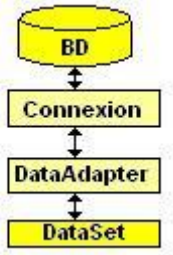
**Comment travailler (lire écrire, modifier, trier…) sur des enregistrements d'une base de données ?**

**Avec un DataSet.**

## XVII-E-1 - Généralités

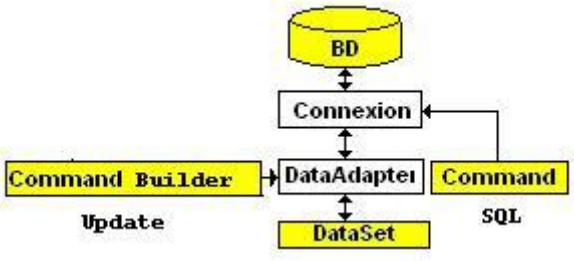
Le **DataSet** est une représentation en mémoire des données. On charge le DataSet à partir de la base de données. Une fois chargé on peut travailler en mode déconnecté. Pour effectuer une modification, on modifie le DataSet puis on met à jour la base de données à partir du DataSet.

Pour remplir un DataSet il faut une **Connexion** puis un **DataAdapter**.



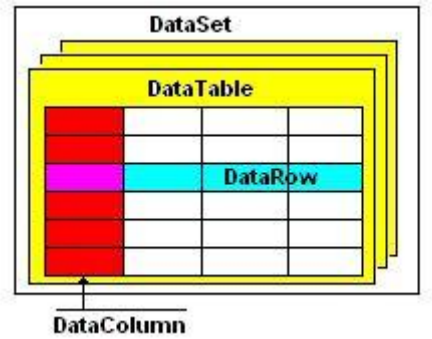

Il faut créer un objet **Connexion** puis un objet **DataAdapter** qui par sa propriété **Fill** charge le **DataSet.**

Les données sont extraites à l'aide de **requête SQL** sur l'objet **Command**, et on utilise un **CommandBluider** pour mettre à jour la base de données à partir du DataSet.




Le DataSet est organisé comme une base de données **en mémoire**, il possède :

- une propriété **Tables** qui contient des **DataTable** (comme les tables d'une BD) ;
- chaque **DataTable** contient une propriété Columns qui contient les **DataColomn** (les colonnes ou champs des BD) et une propriété Rows qui contient des **DataRow** (les lignes ou enregistrements des BD) :

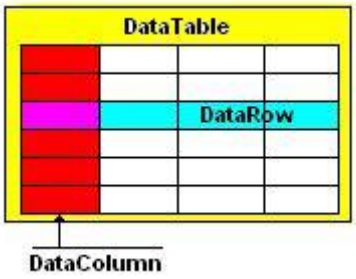



**DataColumn** contient des informations sur le type du champ.

**DataRow** permet d'accéder aux données.

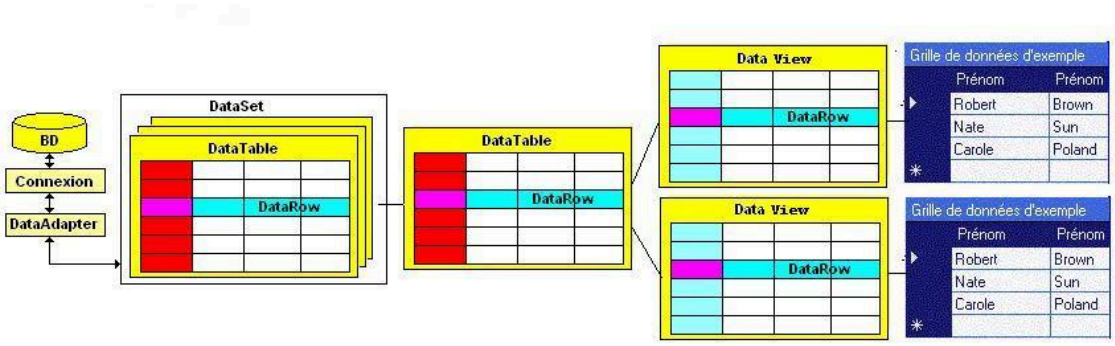
Un DataSet possède aussi la propriété **Constraints** qui contient les Constraint (clé primaire ou clé étrangère), et la propriété Relations qui contient les **DataRelations** (relation entre les tables).

On peut créer des DataTable 'autonomes' et y mettre une table provenant d'un dataSet :




On peut créer des DataView : Vue, représentation d'une table. À partir d'une table, on peut créer plusieurs DataView avec des représentations différentes : DataView trié, DataView donc les lignes ont été filtrées (RowFilter), DataView ne comportant qu'une colonne…

Enfin une DataTable ou un DataView peuvent être affichés dans un contrôle (grid par exemple).




**En conclusion :** on connecte la base de données au DataSet par l'intermédiaire de **Connexion** et **DataAdapter**. Ensuite, il y a 2 manières de 'travailler' :

- A - On utilise les membres du DataSet pour lire ou modifier les données ;
- B - On lie un DataSet, une DataTable ou un DataView à un contrôle DataGrid ou ListBox… c'est le Data Binding.

# XVII-E-2 - Utilisation du DataSet, du DataView : en pratique

Ici on utilise uniquement du code.
Soit une **base de données Access** nommée 'Nom.mdb', elle contient une table 'FichePatient' qui contient les champs (ou colonnes) 'Nom', 'Prenom'.

Je vais me connecter à cette base, extraire les enregistrements de la table 'FichePatient' et les mettre dans un DataSet. Chaque ligne du DataSet contient un patient. Je travaillerais sur ces lignes.

**En tête du module, il faut importer l'espace de noms permettant d'utiliser les DataSet et OleDB.**

```
Imports System.Data

Imports System.Data.OleDb
```

**Dans la zone Général, Déclarations du module, il faut déclarer les objets Ado :**

```
'  Déclaration Objet Connexion

Private ObjetConnection As OleDbConnection

' Déclaration Objet Commande

Private ObjetCommand As OleDbCommand

'  Déclaration Objet DataAdapter

Private ObjetDataAdapter As OleDbDataAdapter

' Déclaration Objet DataSet

Private ObjetDataSet As New DataSet() 'Attention au New

'String contenant la 'Requête SQL'

Private strSql As String

' Déclaration Objet DataTable

Private ObjetDataTable As DataTable

' Déclaration Objet DataRow (ligne)

Private ObjetDataRow As DataRow

'Numéro de la ligne en cours

Private RowNumber As Integer    'Numéro de l'enregistrement courant

'Paramêtres de connexion à la DB

Private strConn As String

'Pour recompiler les données modifiées avant de les remettre dans le

'"DataAdapter"

Private ObjetCommandBuilder As OleDbCommandBuilder
```

**Ouverture** :

```
'Initialisation de la chaine de paramètres pour la connexion

strConn = "Provider=Microsoft.Jet.OLEDB.4.0;"  _
```

```
"Data Source= c:\nom.mdb;"

'Initialisation de la chaine contenant l'instruction SQL

strSql = "SELECT FICHEPATIENT.* FROM FICHEPATIENT"

'Instanciation d'un Objet Connexion

ObjetConnection = New OleDbConnection()

'Donner à la propriété ConnectionString les paramètres de connexion

ObjetConnection.ConnectionString = strConn

'Ouvrir la connexion

ObjetConnection.Open()

'Instancier un objet Commande

ObjetCommand = New OleDbCommand(strSql)

'Instancier un objet Adapter

ObjetDataAdapter = New OleDbDataAdapter(ObjetCommand)

'initialiser l'objet Command

ObjetCommand.Connection() = ObjetConnection

'Avec l'aide de la propriété Fill du DataAdapter charger le DataSet

ObjetDataAdapter.Fill(ObjetDataSet, "FICHEPATIENT")

'Mettre dans un Objet DataTable une table du DataSet

ObjetDataTable = ObjetDataSet.Tables("FICHEPATIENT")
```

Ici le code est simplifié, mais dans la réalité, il est **indispensable d'user de Try Catch** pour capter les exceptions surtout pour ObjetConnection.Open() et pour l'update.

Les extraits (snippets) de VB 2005, donnent un exemple de code plus condensé pour ouvrir un DataSet :

```
Dim conn As String = "Provider=Microsoft.Jet.OLEDB.4.0; _
 Data Source=AccessFile.mdb;Persist Security Info=False"

        Dim cmd As String = "Select * from Topics"

        Dim adapter As New OleDbDataAdapter(cmd, conn)

        Dim topics As New Data.DataSet

        adapter.Fill(topics)
```

### Voir un enregistrement

Routine affichant un enregistrement : une TextBox nommée 'Nom' et une TextBox 'Prenom' afficheront les nom et prénom des patients.

On rappelle que RowNumber est une variable contenant le numéro de la ligne en cours (allant de 0 à Rows.Count-1) :

```
If RowNumber < 0 Then Exit Sub

'Lors de l'ouverture de la BD, s'il n'y a aucun enregistrement

If RowNumber > ObjetDataTable.Rows.Count - 1 Then Exit Sub
```

```
'ObjetTable.Rows(Numéro de lignes).Item( Nom de colonne) donne le contenu
'd'un champ dans une ligne donnée

Me.Nom.Text = ObjetDataTable.Rows(RowNumber).Item("Nom").ToString

Me.Prenom.Text = ObjetDataTable.Rows(RowNumber).Item("Prenom").ToString

'Item peut avoir en paramètre le nom de la colonne ou sont index
```

### Pour se déplacer:

Voir la ligne suivante par exemple :

```
RowNumber += 1 incrémente le numéro de la ligne en cours puis on affiche par:

Me.Nom.Text = ObjetDataTable.Rows(RowNumber).Item("Nom").ToString

Me.Prenom.Text = ObjetDataTable.Rows(RowNumber).Item("Prenom").ToString

RowNumber -= 1 pour la précédente.

RowNumber = 0  pour la première.

RowNumber =  ObjetDataTable.Rows.Count - 1  pour la dernière.
```

On remarque qu'**il n'y a pas d'enregistrement courant**, pas de pointeur sur un enregistrement, c'est vous-même qui gérez 'RowNumber' une simple variable qui contient le numéro de l'enregistrement (l'index du DataRow) sur lequel vous travaillez.

### Modifier un enregistrement :

```
' Extraire l'enregistrement courant

ObjetDataRow = ObjetDataSet.Tables("FICHEPATIENT").Rows(RowNumber)


'Modifier les valeurs des champs en  récupérant le contenu des TextBox

ObjetDataRow("Nom") = Me.Nom.Text

ObjetDataRow("Prenom") = Me.Prenom.Text


'Pour modifier les valeurs changées dans le DataAdapter

ObjetCommandBuilder = New OleDbCommandBuilder(ObjetDataAdapter)


'Mise à jour

ObjetDataAdapter.Update(ObjetDataSet, "FICHEPATIENT")


'On vide le DataSet et on le 'recharge' de nouveau.

ObjetDataSet.Clear()

ObjetDataAdapter.Fill(ObjetDataSet, "FICHEPATIENT")

ObjetDataTable = ObjetDataSet.Tables("FICHEPATIENT")
```

**Attention : quand la commande Update est effectuée, si l'enregistrement ne répond pas aux spécifications de la base (doublon alors que c'est interdit, pas de valeur pour une clé primaire, Champ ayant la valeur Null alors que c'est interdit…), une exception est levée, si vous ne l'avez pas prévue cela plante !!**

Il faut donc mettre la ligne contenant l'Update dans un Try Catch.

### Ajouter un enregistrement :

```
ObjetDataRow = ObjetDataSet.Tables("FICHEPATIENT").NewRow()

ObjetDataRow("Nom") = Me.Nom.Text

ObjetDataRow("Prenom") = Me.Prenom.Text

ObjetDataSet.Tables("FICHEPATIENT").Rows.Add(ObjetDataRow)

'Pour modifier les valeurs changées dans le DataAdapter

ObjetCommandBuilder = New OleDbCommandBuilder(ObjetDataAdapter)

'Mise à jour

ObjetDataAdapter.Update(ObjetDataSet, "FICHEPATIENT")

'On vide le DataSet et on le 'recharge' de nouveau.

ObjetDataSet.Clear()

ObjetDataAdapter.Fill(ObjetDataSet, "FICHEPATIENT")

ObjetDataTable = ObjetDataSet.Tables("FICHEPATIENT")
```

il peut y avoir génération d'une exception au niveau de la ligne Update si l'Objet Datarow présente un 'défaut', par exemple si un champ de l'ObjetDatarow a une valeur null (il n'a pas été renseigné) alors qu'il sert l'index.

### Effacer l'enregistrement en cours :

```
ObjetDataSet.Tables("FICHEPATIENT").Rows(RowNumber).Delete()

ObjetCommandBuilder = New OleDbCommandBuilder(objetDataAdapter)

ObjetDataAdapter.Update(objetDataSet, "FICHEPATIENT")
```

### Fermer :

```
'Objet connecté

ObjetConnection = Nothing

ObjetCommand = Nothing

ObjetDataAdapter = Nothing

'Objet déconnecté

ObjetDataSet = Nothing

ObjetDataTable = Nothing

ObjetDataRow = Nothing
```

### Trier,Filtrer, rechercher

Il existe une méthode Select pour les DataTable, mais qui retourne des DataRow :

```
Dim expression As String = "NOM='LASSERRE'"   'expression à rechercher

Dim sortOrder As String = "Nom DESC"

Dim foundRows() As DataRow     'résultat dans des DataRow

foundRows = ObjetDataTable.Select(expression, sortOrder)

Dim objetDatat1 As New DataTable

For Each r As DataRow In foundRows

objetDatat1.ImportRow(r)

Next

'MsgBox(objetDatat1.Rows.Count)

Dim dv As New DataView(objetDatat1)

DataGrid1.DataSource = dv
```

### Travailler sur une DataTable

Comment ajouter une ligne (Row) :

```
Dim LeNewRow As DataRow = ObjetDataTable.NewRow()  'On crée un DataRow

LeNewRow("NOM") = "Smith"  'On remplit la première cellule

ou
LeNewRow(1) = "Smith"

ObjetDataTable.Rows.Add(LeNewRow) 'On ajoute la Row au DataTable
```

### Utiliser un DataView

Mais on peut aussi passer la table dans un DataView et utiliser Find Sort RowFilter sur le DataView.

Le DataView est un objet qui ressemble à une table, mais qui correspond à une représentation et permet les tris ou l'utilisation d'un filtre :

```
'On crée un DataView, on y met une table.

Dim dv As DataView

dv.Table =  ObjetDataSet.Tables("FICHEPATIENT")
```

ou

```
Dim dv As New DataView(ObjetDataSet.Tables("FICHEPATIENT")) 'on met une table dans le dataview
```

ensuite on peut **trier** le DataView :

```
dv.Sort = "Nom DESC"  'on trie  sur le champ Nom en ordre décroissant (ASC pour un tri
 croissant).
```

On peut **filtrer** le DataView :

```
dv.RowFilter = " PreNom = 'Philippe'"  'bien respecter les  '. et les majuscules/minuscules.
```

Le DataView ne contient plus que les rows dont la col ="Philippe". On peut afficher dans une grid :

```
DataGrid1.DataSource = objetDataView
```

On peut **rechercher** dans le DataView avec Find :

```
    Dim dv As DataView = New DataView(Mytable)
    dv.Sort = "CustomerID"

    ' Cherche le customer named "DUPONT" dans la première colonne
    Dim i As Integer = dv.Find("DUPONT")
    Console.WriteLine(dv(i))'Affiche sur la console
```

Il existe 'FindRows' qui retourne des DataRow.

Ensuite on peut lier le DataView à une Grid.

## XVII-E-3 - Remplir un DataGrid ou une ListBox avec un DataSet

Une fois que le dataSet existe, en UNE ligne de code, on peut l'afficher dans un DataGrid.(Grille avec des lignes et des colonnes comme un tableur)

DataGrid1.SetDataBinding(ObjetDataSet, "FICHEPATIENT") en VB2003 on associe le dataset à la grid.

On peut aussi passer par l'intermédiaire d'une DataTable en VB 2003 et VB 2005 :

```
Dim matable As New DataTable

matable = ObjetDataSet.Tables("FICHEPATIENT")

DataGrid1.DataSource = matable
```

Remplir une Listbox ligne par ligne :

```
Dim matable As New DataTable

matable = ObjetDataSet.Tables("FICHEPATIENT")

Dim Ligne As DataRow()

For Each Ligne In Matable.Rows

          List1.Items.Add(ligne.Item(" Nom "))

Next
```

## XVII-E-4 - Étudions en détail un DataSet

**Un DataSet est un cache de données en mémoire. On a vu qu'on pouvait le remplir avec une base de données, mais on peut imaginer le créer de toute pièce et le remplir en créant des tables, des colonnes, des données.**

Dans un DataSet on peut donc ajouter des tables, dans les tables des colonnes, des enregistrements.

L'exemple suivant crée un objet DataTable, qui sera ajouté au DataSet :

```
            Private myDataSet As DataSet


' Créer une nouvelle DataTable.
```

```
Dim myDataTable As DataTable = new DataTable("ParentTable")
' Declaration de variables DataColumn et DataRow objects.
Dim myDataColumn As DataColumn
Dim myDataRow As DataRow

' Créer un nouveau DataColumn, lui donner un DataType, un nom, diverses valeurs pour ses
 propriétés
'et l'ajouter à la DataTable.
myDataColumn = New DataColumn()
myDataColumn.DataType = System.Type.GetType("System.Int32")     'Type de la colonne
myDataColumn.ColumnName = "id"                                  'Nom de la colonne
myDataColumn.ReadOnly = True                                    'Colonne ReadOnly
myDataColumn.Unique = True                                      'Évite les doublons
myDataTable.Columns.Add(myDataColumn)

' Créer une seconde column.
myDataColumn = New DataColumn()
myDataColumn.DataType = System.Type.GetType("System.String")
myDataColumn.ColumnName = "ParentItem"
myDataColumn.AutoIncrement = False
myDataColumn.Caption = "ParentItem"
myDataColumn.ReadOnly = False
myDataColumn.Unique = False
myDataTable.Columns.Add(myDataColumn)

'La colonne id doit être une clé primaire.
Dim PrimaryKeyColumns(0) As DataColumn
PrimaryKeyColumns(0)= myDataTable.Columns("id")
myDataTable.PrimaryKey = PrimaryKeyColumns


'la colonne peut être une colonne auto incrémentée: (la valeur du champ "CustomerID" démarre à
 100 ,
's'incrémente de 2 automatiquement à chaque création de DataRow.

Dim MyDataColumn2 As DataColumn = MyDataTable.Columns.Add("CustomerID", typeof(Int32))
MyDataColumn2.AutoIncrement = true
MyDataColumn2.AutoIncrementSeed = 100
MyDataColumn2.AutoIncrementStep = 2
```

Il est bon de la mettre en Readonly :

```
' Créer un objet DataSet
myDataSet = New DataSet()
' Ajouter la Table au DataSet.
myDataSet.Tables.Add(myDataTable)

' Créer 3 DataRow objects (3 lignes)  et les ajouter à la DatTable
Dim i As Integer
For i = 0 to 2
myDataRow = myDataTable.NewRow()
myDataRow("id") = i
myDataRow("ParentItem") = "ParentItem " + i.ToString()
myDataTable.Rows.Add(myDataRow)
Next i
End Sub



'parcourir dans la table 'ParentTable' toutes les lignes et lire le premier élément (1re colonne)


For compteur=0 To myDataSet.Tables("ParentTable").Rows.Count -1

    Element= myDataSet.Tables("ParentTable").Rows.(compteur).Item(0)

Next compteur
```

## Travailler avec la Base MySQL

Pour travailler sur une base MySQL lisez le très bon didacticiel **MySQLDotNet** (sur developpez.com bien sûr).

## XVII-F - Liaison DataGrid, ListBox et base de données : le "DataBinding"

**A - Comment remplir des listBox avec des tables venant de base de données ?**

**B - Comment remplir des DataGrid avec des tables venant de base de données ?**

**C - Comment avec quelques clics et sans une ligne de code, créer une application permettant d'afficher le contenu d'une base de données ? (en VB 2005)**

**Cette liaison de données entre une source de données et un contrôle se nomme 'DATABINDING'.**

### XVII-F-1 - Remplir une ListBox avec une colonne d'une table d'une BD

**Ici, on va tout faire avec du code.**

**Exemple**

**Dans une base de données Accès nommée 'BaseNom', j'ai une table 'NomPatient' avec plusieurs colonnes, je veux afficher la colonne des noms dans une listBox :**

| Champs Civilité | Champ Nom | Champ Prénom | Champ Numéro Interne |
|---|---|---|---|
| M. | Durand | Luc | 1 |
| Mme | Dupont | Josette | 2 |
| M. | Thomas | Guy | 3 |

**On peut utiliser un 'DataReader'.**

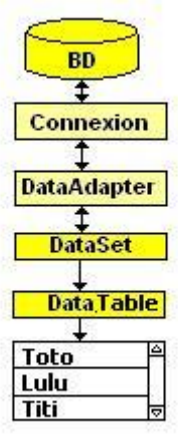
Voir le chapitre sur le DataReader.

C'est de la 'lecture seule' très rapide.

**On peut utiliser un 'DataSet', créer un 'DataTable' et la lier au contrôle.**

Le DataSet est une représentation en mémoire des données. Une fois chargé on peut travailler en mode déconnecté.

Pour le remplir, il faut un **DataAdapter**.




Bien sûr il faut importer des espaces de noms :

```
Imports System

Imports System.Data

Imports System.Data.OleDb
```

Dans la zone déclaration de la fenêtre :

```
'Déclarer la connexion

Private ObjetConnection As OleDbConnection

' Déclaration l'Objet Commande

Private ObjetCommand As OleDbCommand

' Déclaration Objet DataAdapter

Private ObjetDataAdapter As OleDbDataAdapter

' Déclaration Objet DataSet

Private ObjetDataSet As New DataSet

'String contenant la 'Requête SQL'

Private strSql As String

' Déclaration Objet DataTable

Private ObjetDataTable As DataTable

'Paramêtres de connexion à la DB

Private strConn As String
```

Dans une routine Button1_Click créer les divers objets et remplir la listbox :

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
 Button1.Click

'Initialisation de la chaine de paramètres pour la connexion

strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" & "Data Source= c:\Basenom.mdb;"

'Initialisation de la chaine contenant l'instruction SQL

strSql = "SELECT FICHEPATIENT.* FROM FICHEPATIENT"
```

```
'Instanciation d'un Objet Connexion

ObjetConnection = New OleDbConnection

'Donner à la propriété ConnectionString les paramètres de connexion

ObjetConnection.ConnectionString = strConn

'Ouvrir la connexion

ObjetConnection.Open()

'Instancier un objet Commande

ObjetCommand = New OleDbCommand(strSql)

'Instancier un objet Adapter

ObjetDataAdapter = New OleDbDataAdapter(ObjetCommand)

'initialiser l'objet Command

ObjetCommand.Connection() = ObjetConnection

'Avec l'aide de la propriété Fill du DataAdapter charger le DataSet

ObjetDataAdapter.Fill(ObjetDataSet, "FICHEPATIENT")

'Mettre dans un Objet DataTable une table du DataSet

ObjetDataTable = ObjetDataSet.Tables("FICHEPATIENT")


'Indiquer au ListBox d'afficher la table "fichepatient" (indiquer la source)

ListBox1.DataSource = ObjetDataSet.Tables("FICHEPATIENT")

'Indiquer quelle colonne afficher

ListBox1.DisplayMember = "NOM"

End Sub
```

Voilà, cela affiche la liste des noms.

**Bien respecter les minuscules et majuscules dans les noms de tables et de champs+++**

**Récupérer la valeur d'un autre champ.**

On a vu que dans la table, chaque enregistrement avait un champ 'Nom', mais aussi un champ 'NumInt' (numéro interne).

Dans les programmes, on a souvent besoin de récupérer le numéro interne (un ID) quand l'utilisateur clique sur un des noms, le numéro interne servira à travailler sur ce patient.

Exemple : comment récupérer le numéro interne 3 quand l'utilisateur clique sur 'Thomas' ?

Il faut indiquer à la ListBox que la Value de chaque ligne est 'NumInt' en utilisant la propriété ListBox1.ValueMember.

Quand l'utilisateur clique sur une ligne de la ListBox, cela déclenche l'événement ListBox1_SelectedIndexChanged, là on récupère la valeur de NumInt, elle se trouve dans ListBox1.SelectedValue (c'est un Int32).

**Attention**

Les exemples de Microsoft ne fonctionnent pas !! car, on n'explique nulle part l'importance de l'ordre des lignes remplissant la ListBox.

**C'est DataSource qui semble déclencher le chargement de la ListBox avec la prise en compte de DisplayMember et ValueMember.**

Si on fait comme Microsoft (renseigner ListBox1.DataSource en premier) :

```
ListBox1.DataSource = ObjetDataSet.Tables("FICHEPATIENT")

ListBox1.DisplayMember = "NOM"

ListBox1.ValueMember = "NUMINT"
```

ValueMember ne fonctionne pas bien, et ListBox1.SelectedValue retourne un objet de type DataRowView impossible à utiliser.

**Il faut donc renseigner le DataSource en dernier.**

```
ListBox1.DisplayMember = "NOM"

ListBox1.ValueMember = "NUMINT"

ListBox1.DataSource = ObjetDataSet.Tables("FICHEPATIENT")

'Dans ce cas ListBox1.SelectedValue contient bien un Int32 correspondant au 'NutInt' selectionné.

'Ensuite on peut récupérer sans problème NumInt (et l'afficher par exemple dans une TextBox)

Private Sub ListBox1_SelectedIndexChanged(ByVal sender As Object, ByVal e As System.EventArgs)
_Handles ListBox1.SelectedIndexChanged

Dim o As New Object

If ListBox1.SelectedIndex <> -1 Then

TextBox1.Text = CType(ListBox1.SelectedValue, String)

End If

End Sub
```

## XVII-F-2 - Remplir un DataGrid avec une base de données via un DataSet

**Avec mise à jour de la base de données,** si on fait une modification dans le Datagrid. Oui ça marche (depuis que j'ai ajouté le CommandBluider).

**Ici, on va tout faire avec du code.**

Il faut créer une 'Connection', un DataAdapter et un DataSet, **puis en UNE ligne de code**, on peut l'afficher dans un DataGrid (grille avec des lignes et des colonnes comme un tableur), ne pas oublier le CommandBuilder sinon Update ne marche pas.

**Dans la zone de déclaration :**

```
' Déclaration Objet Connection

Private ObjetConnection As OleDbConnection

' Déclaration Objet Commande

Private ObjetCommand As OleDbCommand
```

```
' Déclaration Objet DataAdapter

Private ObjetDataAdapter As OleDbDataAdapter

' Déclaration Objet DataSet

Private ObjetDataSet As New DataSet

' Déclaration Objet DataTable

Private ObjetDataTable As New DataTable

'String contenant la 'Requête SQL'

Private strSql As String

'Paramêtres de connexion à la DB

Private strConn As String

' Déclaration d'un  OleDbCommandBuilder

Private ObjetCB As OleDbCommandBuilder
```

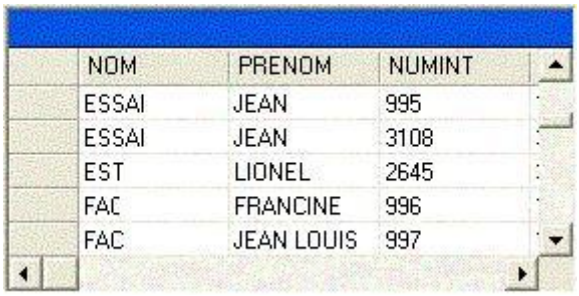
### Le bouton ButtonAfficheGrid remplit le DataGrid avec nom.mdb table 'FICHEPATIENT'

```
Private Sub ButtonAfficheGrid_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
_Handles ButtonAfficheGrid.Click

'Initialisation de la chaine de paramètres pour la connexion

strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" & "Data Source= c:\nom.mdb;"

'Initialisation de la chaine contenant l'instruction SQL

strSql = "SELECT FICHEPATIENT.* FROM FICHEPATIENT"

'Instanciation d'un Objet Connexion

ObjetConnection = New OleDbConnection

'Donner à la propriété ConnectionString les paramètres de connexion

ObjetConnection.ConnectionString = strConn

'Ouvrir la connexion

ObjetConnection.Open()

'Instancier un objet Commande

ObjetCommand = New OleDbCommand(strSql)

'Instancier un objet Adapter

ObjetDataAdapter = New OleDbDataAdapter(ObjetCommand)

'initialiser l'objet Command

ObjetCommand.Connection() = ObjetConnection

'initialiser l'objet OleCBComandBuilder (sinon pas d'update)

ObjetCB = New OleDbCommandBuilder(ObjetDataAdapter)

'Avec l'aide de la propriété Fill du DataAdapter charger le DataSet

ObjetDataAdapter.Fill(ObjetDataSet, "FICHEPATIENT")

'Créer une datatable à partir du dataset
```

```
ObjetDataTable = ObjetDataSet.Tables("FICHEPATIENT")

'Mettre dans le DataGrid une table  DataTable

DataGrid1.DataSource= ObjetDataTable

End Sub
```

**Le bouton ButtonMiseA jour met à jour nom.mdb si on a fait une modification dans le DataGrid.**

```
Private Sub ButtonMiseAJour_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
_Handles ButtonMiseAJour.Click

    'Mettre à jour

    ObjetDataAdapter.Update(ObjetDataSet, "FICHEPATIENT")

End Sub
```



| | NOM | PRENOM | NUMINT |
|---|---|---|---|
| | ESSAI | JEAN | 995 |
| | ESSAI | JEAN | 3108 |
| | EST | LIONEL | 2645 |
| | FAC | FRANCINE | 996 |
| | FAC | JEAN LOUIS | 997 |

Les lignes, colonnes, nom des champs, ascenseurs… sont crées automatiquement avec la table "FICHEPATIENT" !! Génial.

On peut cliquer dans une cellule, la modifier…rajouter ou enlever des lignes.

On peut aussi remplir le Datagrid avec :

```
DataGrid1.DataSource = ObjetDataSet

DataGrid1.DataMember = "FICHEPATIENT"
```

En mode Design, on peut modifier l'aspect du DataGrid dans la fenêtre de propriété ou en utilisant la mise en forme automatique (lien en bas de la fenêtre de propriétés) en VB 2003 !!

## XVII-F-3 - Remplir un contrôle avec une source de données avec le moins de code possible(VB 2005 2008)

**Ici, on va créer une source de données (sans code, avec l'IDE).** Puis on fera un binding sur le contrôle d'abord avec du code puis sans code.

On a une base de données Access (Nom.MDB), on veut afficher le contenu d'une des tables (la table 'FICHEPATIENT') dans une grille.

- Il faut créer une source de données : la BD Nom.MDB.

- Il faut créer l'interface utilisateur.

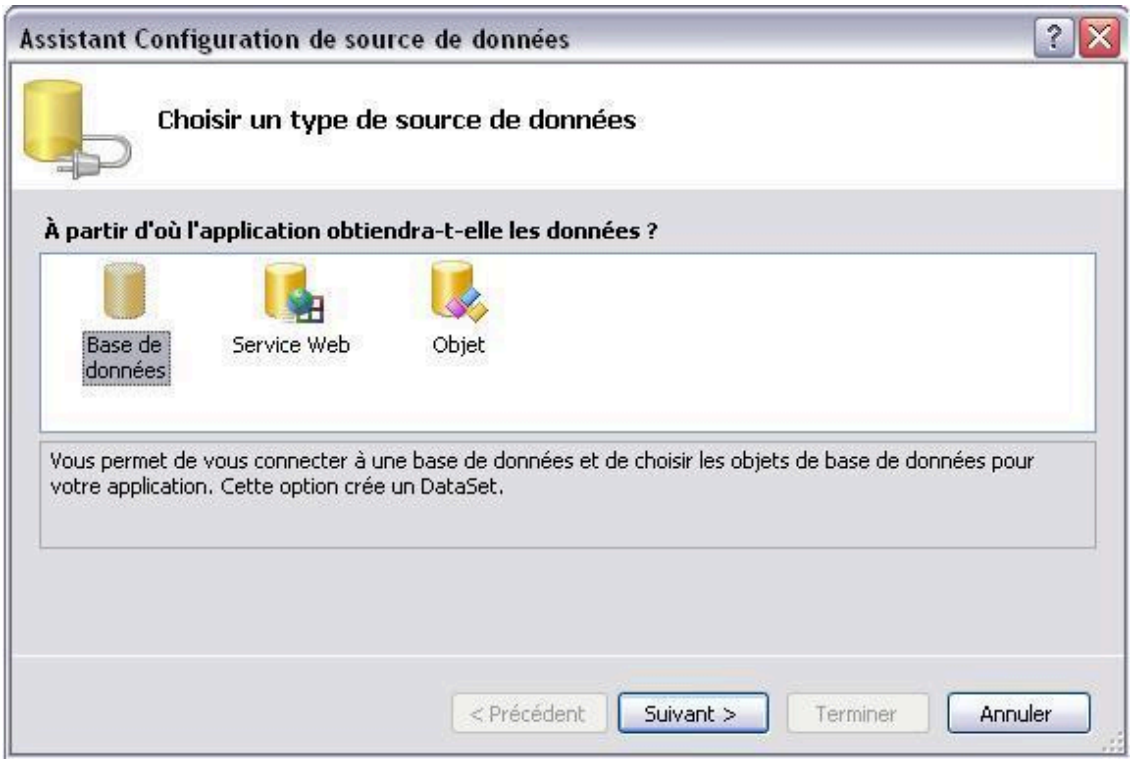
- Il faut créer les liens entre cette interface et une table de la BD.

## XVII-F-3-a - Création de la source de données

On veut créer sans code une **source de données.**

Menu 'Données'=> 'Ajouter une nouvelle source de données'


Ici la source de données est une base de données :

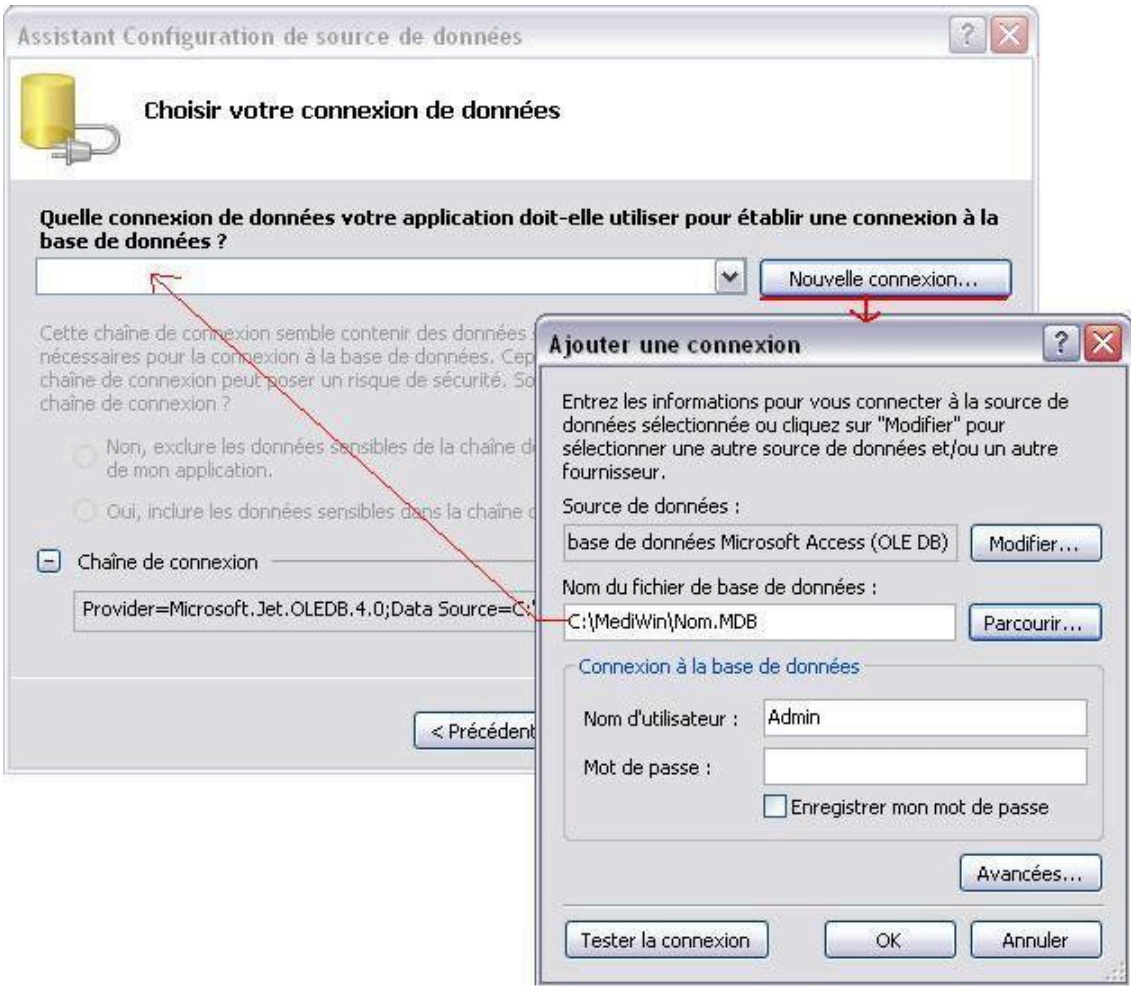



On clique sur 'Base de données' puis bouton 'Suivant'.

Ensuite il faut faire le choix de la connexion (cela correspond au choix d'une base de données existante).

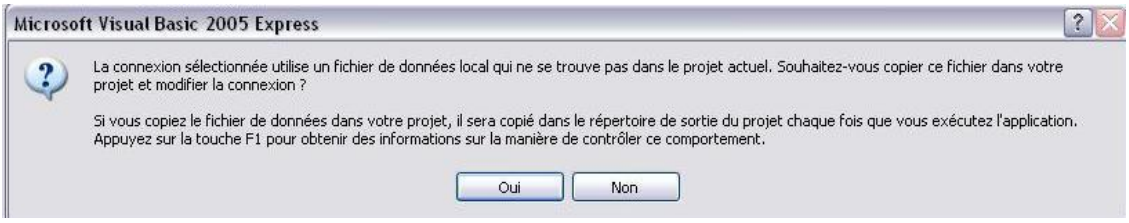
Cliquer sur 'Nouvelle connexion'. Une nouvelle fenêtre nommée 'Ajouter une connexion' s'ouvre.

Indiquer le type de source de données (ici Microsoft Access (OLEDB), puis le nom de la base de données, enfin cliquer sur "OK".

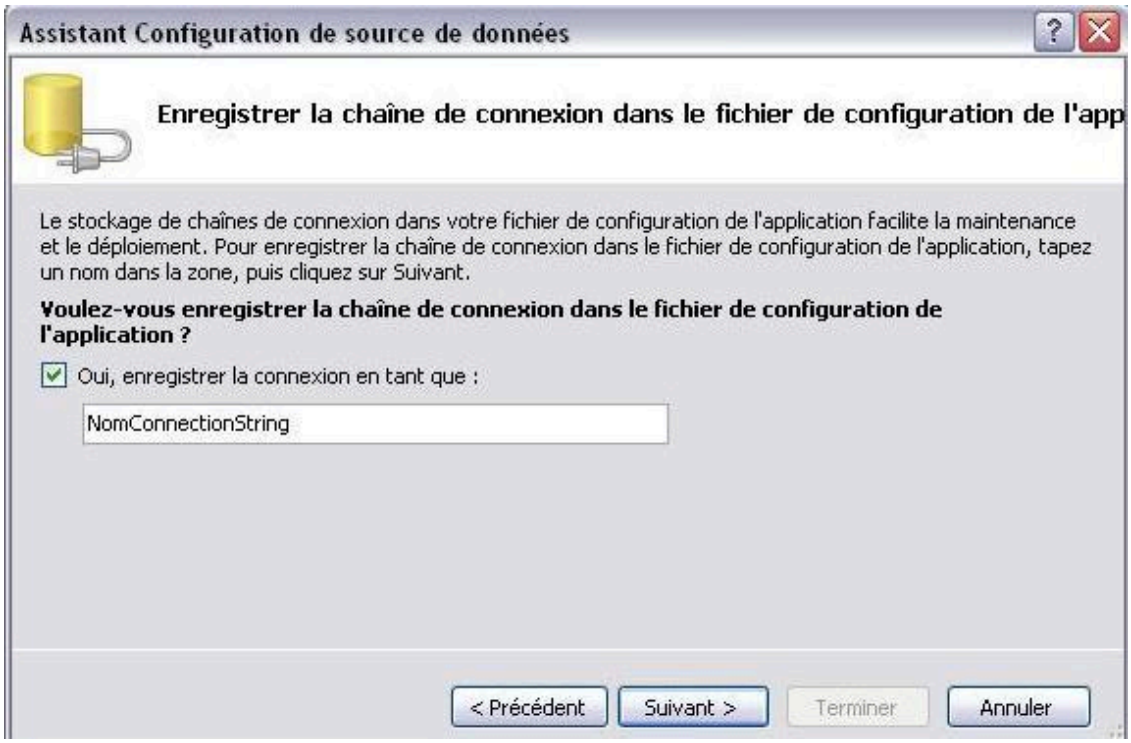



On retrouve le nom de la bd dans la zone connexion, cliquer sur "Suivant".

Ensuite VB vous demande s'il faut copier la BD dans le projet : si oui la BD sera copiée dans le répertoire de travail quand vous installerez votre application. À vous de voir.




Ensuite VB vous propose d'enregistrer les chaines de connexion (nom de la base…) dans le fichier de configuration afin qu'elles soient stockées et lues lors de l'utilisation ultérieure (automatiquement bien sûr). À vous de voir.

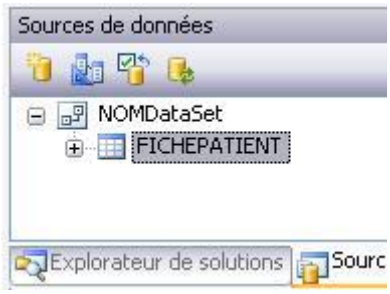



Ensuite il faut choisir dans la base les objets qui seront chargés dans le dataset.

C'est habituellement les Tables. Cocher 'Tables'.


Cliquer sur "Terminer".

Pour voir les sources de données, cliquer sur le menu 'Données' puis 'Voir les sources de données'.
On voit bien dans notre exemple 'NOMDataSet' le DataSet de la source de données qui contient FICHEPATIENT.




## XVII-F-3-b - Liaison source de données-Grid avec du code

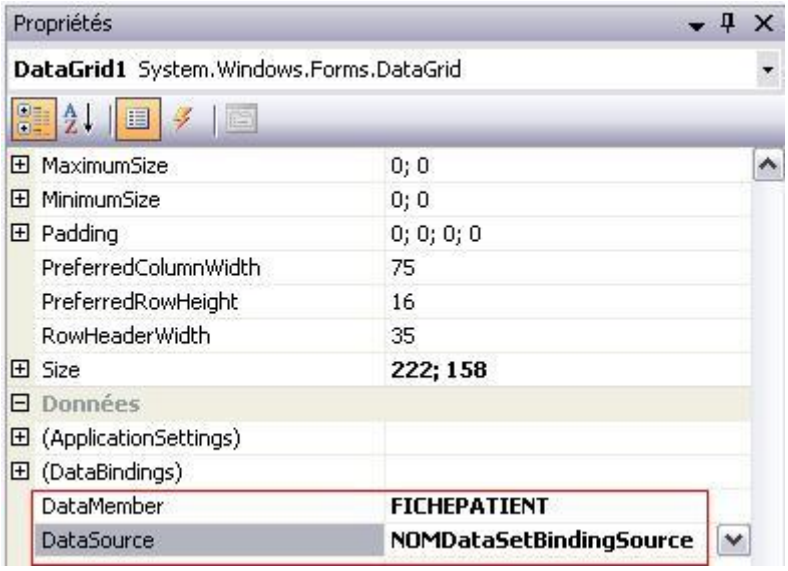
Maintenant qu'on a la source de données, on va créer un DataAdapter et une Grid qu'on va remplir grâce à sa propriété 'DataSource'.

**On utilisera la méthode Fill du TableAdapter.**

```
Private Sub Form1_Load

'Création d'un TableAdapter

Dim FPTableAdapter = New NOMDataSetTableAdapters.FICHEPATIENTTableAdapter

'Extraction des données et chargement du DataSet

FPTableAdapter.Fill(NOMDataSet.FICHEPATIENT)

End Sub
```

**Il faut ensuite ajouter une grid puis faire le lien DataSet-Grid.**

Dans les propriétés de DataGrid il faut enseigner DataSource et DataMember.




On exécute, la grille se remplit avec les données.

**Comment enregistrer les modifications de la grid effectuées par l'utilisateur dans la base de données ?**

Pour que les modifications de la grille soient répercutées dans le DataSet, il faut enregistrer le DataSet dans la base grâce au TableAdapter et à sa méthode Update.

```
Private Sub Enregistrer_Click

Dim PatTableAdapter = New NOMDataSetTableAdapters.FICHEPATIENTTableAdapter

NOMDataSetBindingSource.EndEdit()

'Vérifiez que des modifications ont eu lieu

If NOMDataSet.HasChanges Then

    'Appliquer les changements dans la base de données

    PatTableAdapter.Update(NOMDataSet.FICHEPATIENT)

End If

End Sub
```

## XVII-F-3-c - Génération automatique de l'interface utilisateur

Plutôt que de faire le binding 'à la main' et taper du code comme ci-dessus, on peut tout faire faire par VB.

Visual Studio dispose d'une fenêtre 'Sources de données' depuis laquelle vous pouvez faire glisser des éléments jusqu'à un formulaire pour créer automatiquement des contrôles liés aux données qui affichent des données.

Afficher les sources de données.

Menu 'Données' puis 'Afficher les sources de données'

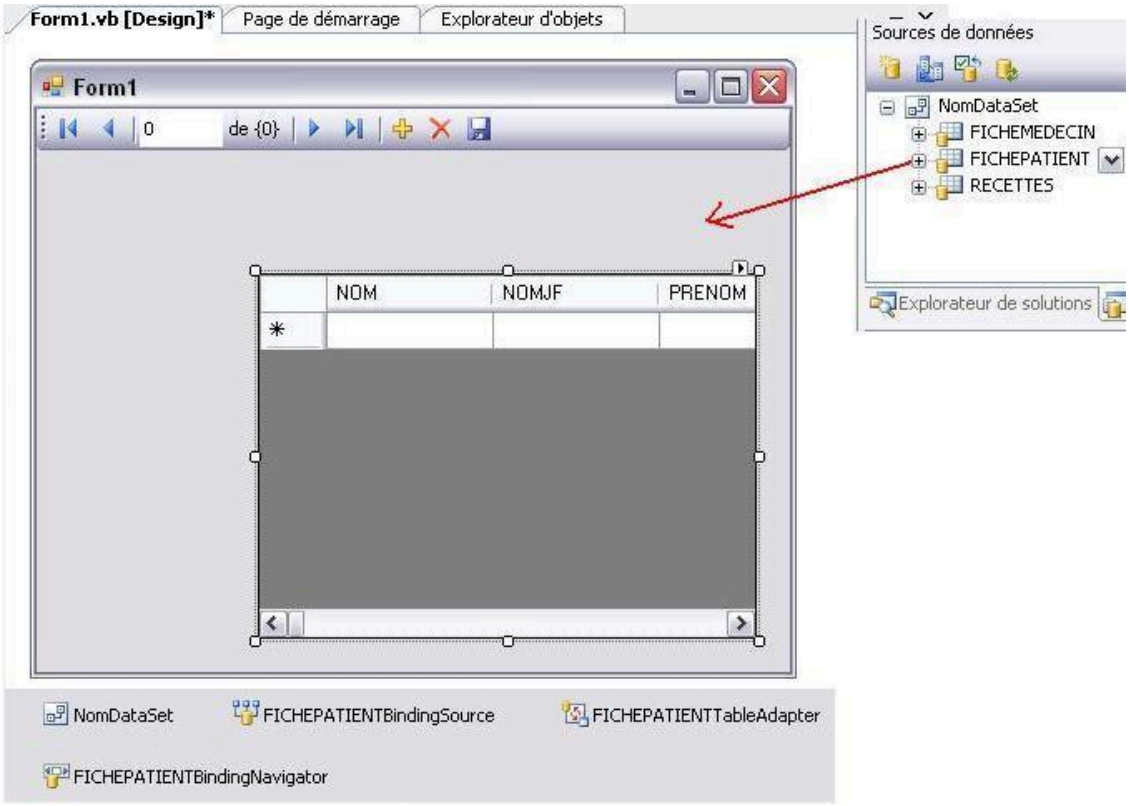

Il apparait à droite la fenêtre 'Sources de données'.

Dérouler 'NomDataSet' (c'est le nom donné par VB à la connexion) et cliquer sur 'FICHEPATIENT' (c'est le nom d'une table de la BD).

Enfin faire un drag and drop à partir de FICHEPATIENT et le déposer sur la form de gauche (qui est vide au départ).

Miracle : il apparait automatiquement :

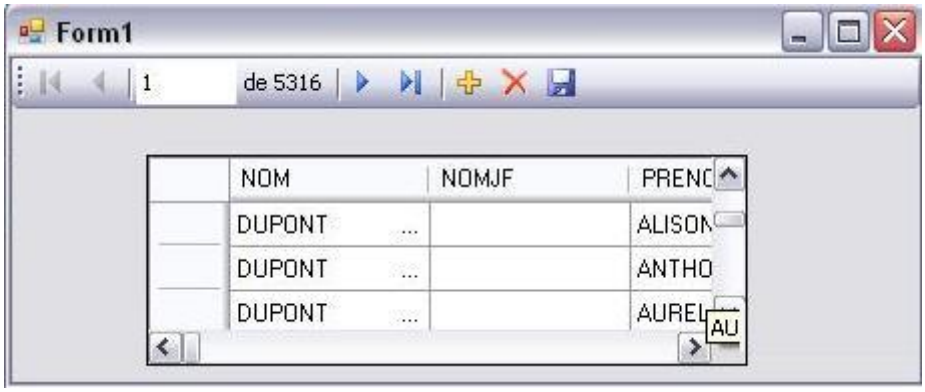
- un datagrid ;
- une barre de navigation (tout est généré automatiquement : les bitmap des boutons dans les ressources Setting…) ;
- un DataSet, un TableAdapter ;
- un composant BindingSource (il fait le lien entre l'interface et la source de données) ;
- un composant BindingNavigator (il gère la barre de navigation).




On voit bien en dessous les 4 composants qui ont été ajoutés.

C'est terminé !!

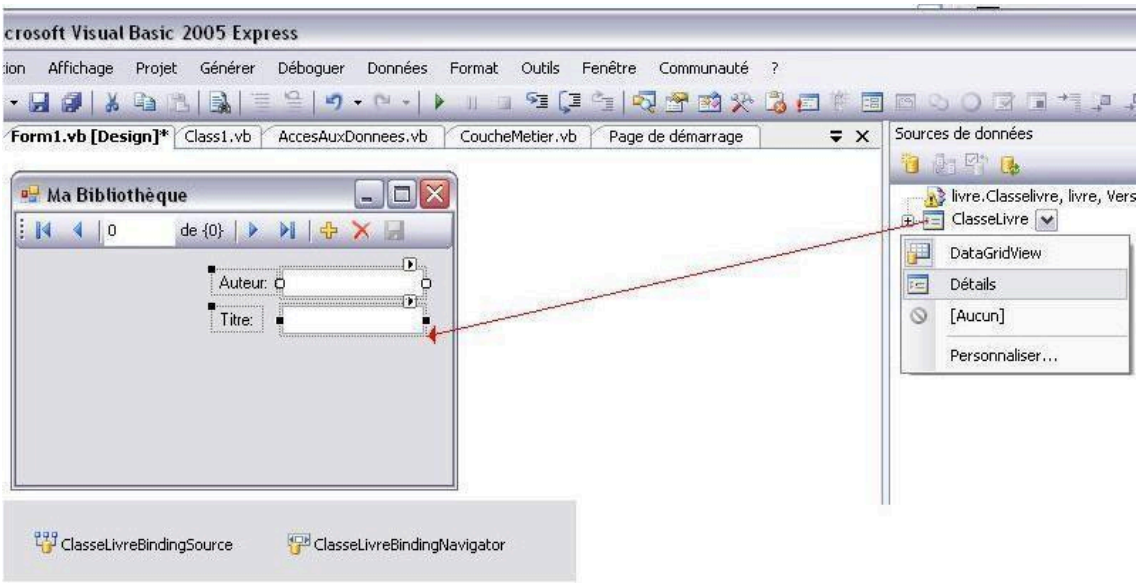
Il suffit de lancer le programme, et on voit la bd dans la grille, on se ballade dedans avec le curseur ou la barre de navigation, on peut ajouter, effacer une ligne, enregistrer les modifications :




## XVII-F-3-d - Binding et TextBox

Pour générer, non pas une grid, mais des zones de saisie textbox pour chaque champ, avant de faire le drag and drop, dérouler la liste contre la Table dans les sources de données et cliquer sur 'Détails'.

Il y a un exemple dans la section sur les classes, Programme à 3 couches qui donne ceci :




## XVII-G - Créer une BD, ajouter une table à une base de données

**Comment créer une base de données ?**

**Comment ajouter une table ?**

Pour **consulter une base de données,** vous pouvez utiliser l'explorateur de serveurs.

Pour accéder à l'Explorateur de serveurs, sélectionnez Explorateur de serveurs dans le menu Affichage et regarder la doc.

## XVII-G-1 - Créer une base de données

Généralement la base de données est fournie à l'utilisateur avec l'exécutable.

**Créer une Base Access**

Avec un Provider OleDB (en ADO) : (System.Data.OLEDB) impossible de créer une BD Access par du code.

On rappelle qu'on peut créer rapidement une BD Access 'à la main'.

Sur le fond d'écran : Bouton droit->Nouveau->Application Microsoft Access. Il faut ensuite la fournir.

Si on veut absolument créer une BD Access par code, il faut passer par DAO.

```
Référence: ajouter le composant COM 'Microsoft DAO 3.6 Library 5'

Imports DAO

Imports DAO.LanguageConstants

Dim result As Boolean = False
```

```
Dim dbe As New DBEngine
Dim db As Database
Try
db = dbe.CreateDatabase("c:\test.mdb", dbLangGeneral)
If Not (db Is Nothing) Then result = True
Catch ex As Exception : MsgBox(ex.Message)
Finally : If Not (db Is Nothing) Then db.Close()
End Try
```

Ensuite, on peut travailler dessus avec ADO en OleDB.

(Merci les forums de developpez.com).

**Base SqlServer**

Avec un Provider SqlServer: (System.Data.SqlClient) **on peut** créer une BD Sql Server par du code.

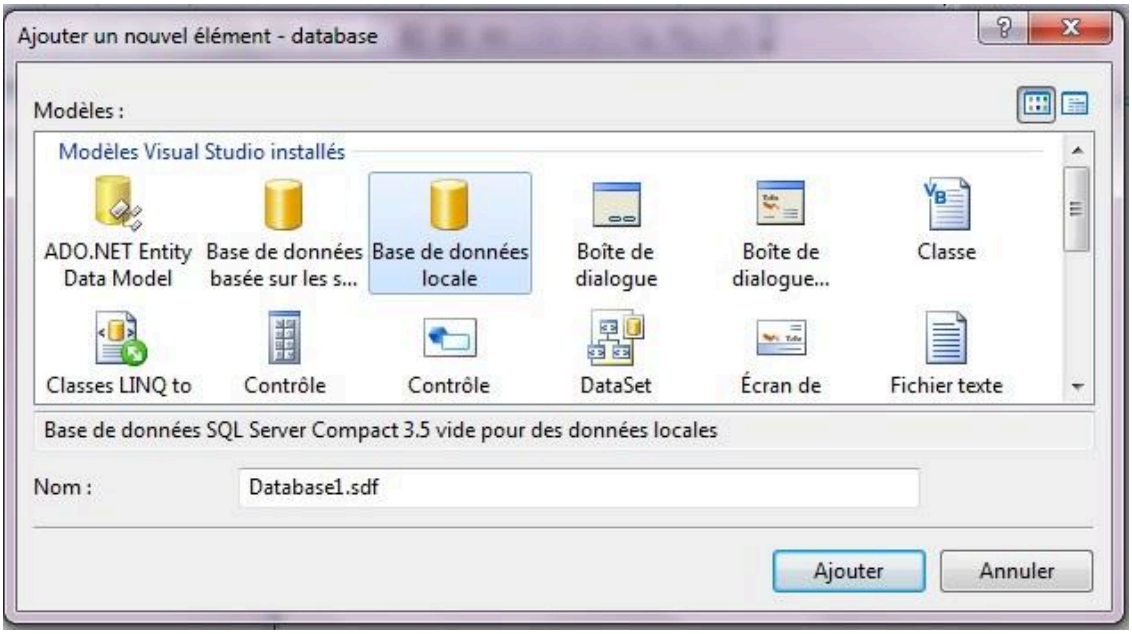
Exemple :

```
Protected Const SQL_CONNECTION_STRING As String = _
"Server=localhost;" & _
"DataBase=;" & _
"Integrated Security=SSPI"

Dim strSQL As String = _
"IF EXISTS (" & _
"SELECT * " & _
"FROM master…sysdatabases " & _
"WHERE Name = 'HowToDemo')" & vbCrLf & _
"DROP DATABASE HowToDemo" & vbCrLf & _
"CREATE DATABASE test"

Dim northwindConnection As New SqlConnection(connectionString)
Dim cmd As New SqlCommand(strSQL, northwindConnection)
northwindConnection.Open()
cmd.ExecuteNonQuery()
northwindConnection.Close()
```

Code non testé.

Il est aussi possible (en VB 2008) de **créer sa base de données SQL Server dans l'IDE** (base de données SQL Server Compact 3.5 à l'aide de Visual Basic Express).

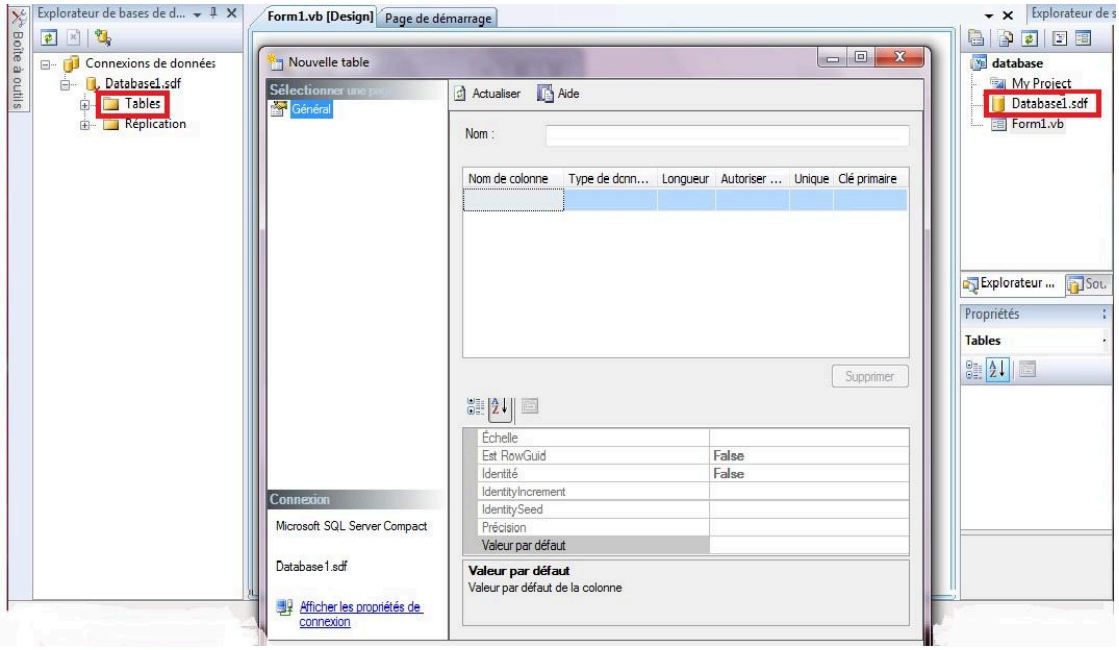
Menu 'Projet'=>'Ajouter un nouvel élément' :




Cliquez sur 'Base de données locale'. Dans la zone Nom, tapez le nom de votre base (DataBase1), puis cliquez sur 'Ajouter'.

Dans l'assistant de configuration qui s'ouvre, cliquez sur annuler.

Pour ajouter des tables dans la base qui a été créée, double-cliquez sur DataBase1.sdf dans l'explorateur de solution à droite (voir ci-dessous).
La structure de la base de données apparait à gauche dans l'explorateur de base de données. Cliquez sur Table avec le bouton droit de la souris, dans le menu qui apparait cliquez sur ajouter une table :



La fenêtre qui apparait permet de saisir un nom de table et d'ajouter les champs et leur type ainsi que la clé primaire.

Notez que la base de données (sous Windows 7) est dans Document/Visual Studio/Projects/Database/Database (Database étant le nom du programme).

## XVII-G-2 - Ajouter une table par code

Dans un Dataset, on peut ajouter des **données** puis par un update mettre à jour la base de données.

On peut aussi ajouter des tables en local dans le Dataset, mais updater la base ne rajoute pas les tables dans la base.

**Comment donc ajouter une table à la base ?**

En fait pour ajouter une table à la base, il faut le faire avec l'objet Command et sa méthode **ExecuteNoQuery** et une requête **SQL.**

On crée une connexion et un objet Command, on indique à la propriété TypeCommand que l'on envoie du texte (pas une procédure stockée), on prépare le texte SQL avec CREATE puis on exécute avec ExecuteNoQuery.

Exemple

Il existe une BD Acces nommée NOM.MDB , je veux y ajouter une Table nommée **PARENT** avec 3 champs CLI_ID CLI_NOM CLI_PRENOM :

```
Imports System

Imports System.Data

Imports System.Data.OleDb
```

Dans la zone déclaration de la fenêtre :

```
'Déclarer la connexion

Private ObjetConnection As OleDbConnection

' Déclaration l'Objet Commande

Private ObjetCommand As OleDbCommand



'Paramètres de connexion à la DB

Private strConn As String
```

Dans une routine Button1_Click créer une table avec 3 champs dans la base de données.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
 Handles Button1.Click


strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" & "Data Source= c:nom.mdb;"

ObjetConnection = New OleDbConnection

'Donner à la propriété ConnectionString les paramètres de connexion

ObjetConnection.ConnectionString = strConn

'Ouvrir la connexion
```

```
ObjetConnection.Open()

'Instancier un objet Commande

ObjetCommand = New OleDbCommand

'Lier Commande et Connexion

ObjetCommand.Connection = ObjetConnection

'Indiquer le type de commande

ObjetCommand.CommandType = CommandType.Text

'Donner le texte de la commande SQL

ObjetCommand.CommandText = "CREATE TABLE PARENT (CLI_ID INTEGER NOT NULL
 _PRIMARY KEY, CLI_NOM CHAR(32) NOT NULL, CLI_PRENOM VARCHAR(32))"

'Ici on crée une table PARENT avec 3 champs ; CLI_ID est un entier non nul qui sert de clé
 primaire,
' CLI_NOM  CLI_PRENOM sont des chaines de 32 caractères.

'on exécute la commande

ObjetCommand.ExecuteNonQuery()

'Fermer la connexion

ObjetConnection.Close()

End Sub
```

## XVII-H - LINQ et les bases de données

Comment interroger une base de données ?

Jusqu'à VB 2005, on mettait les instructions SQL dans une chaine de caractères et on créait un objet Command ADO avec cette chaine pour interroger la base.

**Avec VB 2008 et le Framework 3.5, on utilise LINQ** un langage de requêtes (permettant d'interroger une source de données) directement dans le code Visual Basic et à l'aide de mots-clés familiers (issues du SQL, le langage d'interrogation des bases de données).

### A - Linq c'est quoi ?

'Language-Integrated Query' (LINQ), veut dire "langage de requête intégré".

On l'utilise dans VB à partir de VB2008 (Framework 3.5).

C'est un **langage de requêtes** (permettant d'interroger une source de données) directement dans le code Visual Basic et à l'aide de mots-clés familiers (issues du SQL, le langage d'interrogation des bases de données).

Cette source de données peut être un objet ArrayList, un tableau, une chaine de caractères (voir Linq to Objects chapitre 1.18), mais aussi, c'est ce qui nous intéresse ici, DataSet ou une Base de données SQL.

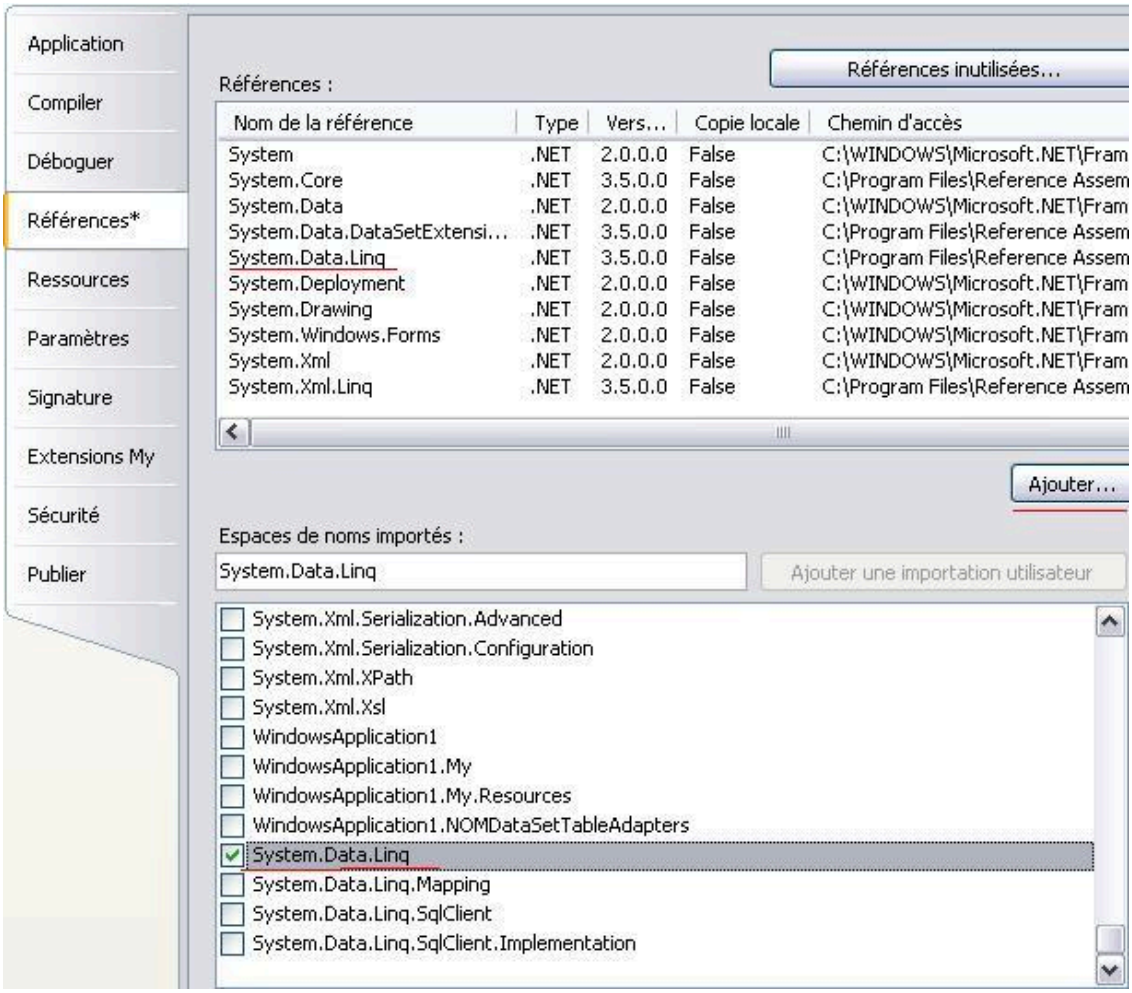
**Pour que LINQ soit pris en compte il faut :**

-utiliser le framework 3.5 ;

-dans les propriétés, onglet compile que **Option Infer=On** ;

-ajouter **Imports System.Data.Linq**.

Si vous créez un nouveau projet dans VB 2008, toutes les conditions sont effectives par défauts, si vous modifiez un ancien projet, il faut rajouter certaines références.
Dans l'Explorateur de solutions (Projet, Propriétés…), cliquez sur 'Références', puis cliquez sur 'Ajouter une référence'. Cliquez sur .NET, sur l'assembly System.Data.Linq, puis sur OK, cela ajoute la référence.




Il faut ajouter l'espace de noms : dans l'explorateur de solution, cocher Systel.Data.Link comme ci-dessus ou ajouter les directives suivantes en haut du Module1 : Imports System.Data.Linq

### Principe d'une requête Linq

À titre d'exemple simpliste, on a des données dans MyData et chaque donnée a un champ 'Nom'. Comment chercher les enregistrements ayant comme nom "toto"?

```
Dim Resultat = From Element In MyData _

Where Element.Nom = "Toto" _

Select Element
```

On crée **une variable de requête** (ici 'Dim Resultat') qui contient les résultats,

puis l'expression de requête composée de :

- **From :** dans quoi chercher ? Dans chaque Element de MyData ;

- **Where :** précise les conditions à appliquer ;

- **Select**: précise les éléments qui vont apparaitre dans 'Resultat'.

Remarquons que **Dim From In Where Select doivent être sur une seule unique et même ligne ;** pour la lisibilité, on écrit sur plusieurs lignes en ajoutant des continuateurs de lignes "_".

Remarquons aussi qu'initialement on connait MyData et on sait que chaque élément de MyData a un champ 'Nom', c'est tout !! On utilise dans la requête les nouvelles variables 'Resultat' et 'Element' sans avoir à déclarer leurs types (on aurait pu le faire). 'Element' est une variable de portée déduite comme élément de MyData.

'Et pour afficher les noms dans une ListBox :

```
For Each P In Resultat

   ListBox1.Items.Add(P.NOM )

Next
```

**Order By** permet de trier les résultats.

```
Dim Resultat = From Element In MyData _

 Order By Element.Price Descending, Element.Nom _

 Select Element.Nom, Element.Price
```

Ici on trie par prix décroissant, puis à prix égal sur le nom croissant.

Remarquons qu'on sélectionne seulement 2 'colonnes'.

## B - Linq To DataSet

En plus de System.Linq, il doit y avoir une référence à **System.Data** et **System.Data.DataSetExtentions** pour le DataSet.

Exemple : on a une base de données Access (Nom.MDB), on veut charger la table 'FICHEPATIENT' qui contient les colonnes NOM, PRENOM, SEXE… dans un DataSet puis interroger le DataSet avec Linq.

### 1 - Création de la source de données

Il faut créer une source de données.

Menu 'Données'=> 'Ajouter une nouvelle source de données'

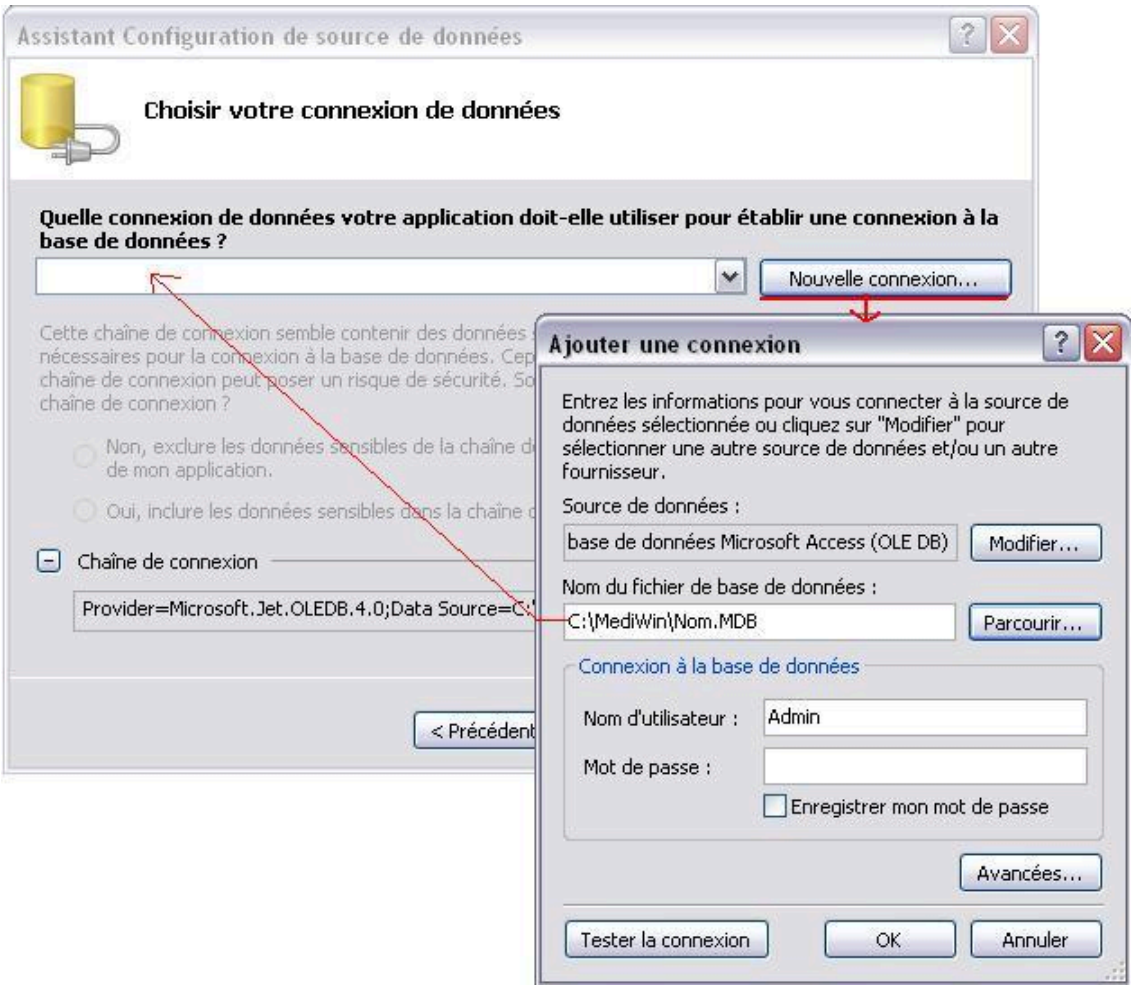

Ici la source de données est une base de données :


On clique sur 'Base de données' puis bouton 'Suivant'.

Ensuite il faut faire le choix de la connexion (cela correspond au choix d'une base de données existante).

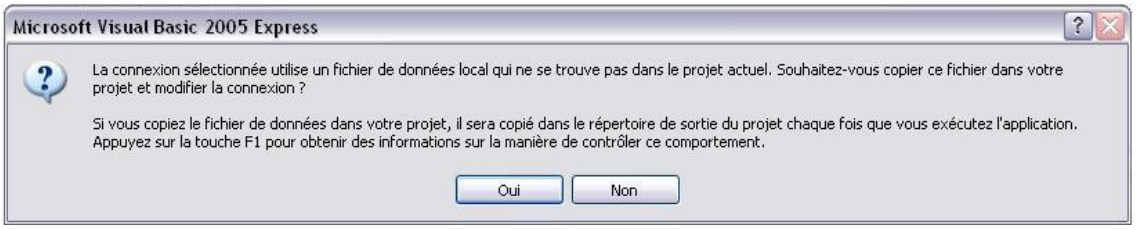
Cliquer sur 'Nouvelle connexion'. Une nouvelle fenêtre nommée 'Ajouter une connexion' s'ouvre.

Indiquer le type de source de données (ici Microsoft Access (OLEDB), puis le nom de la base de données, enfin cliquer sur "OK".

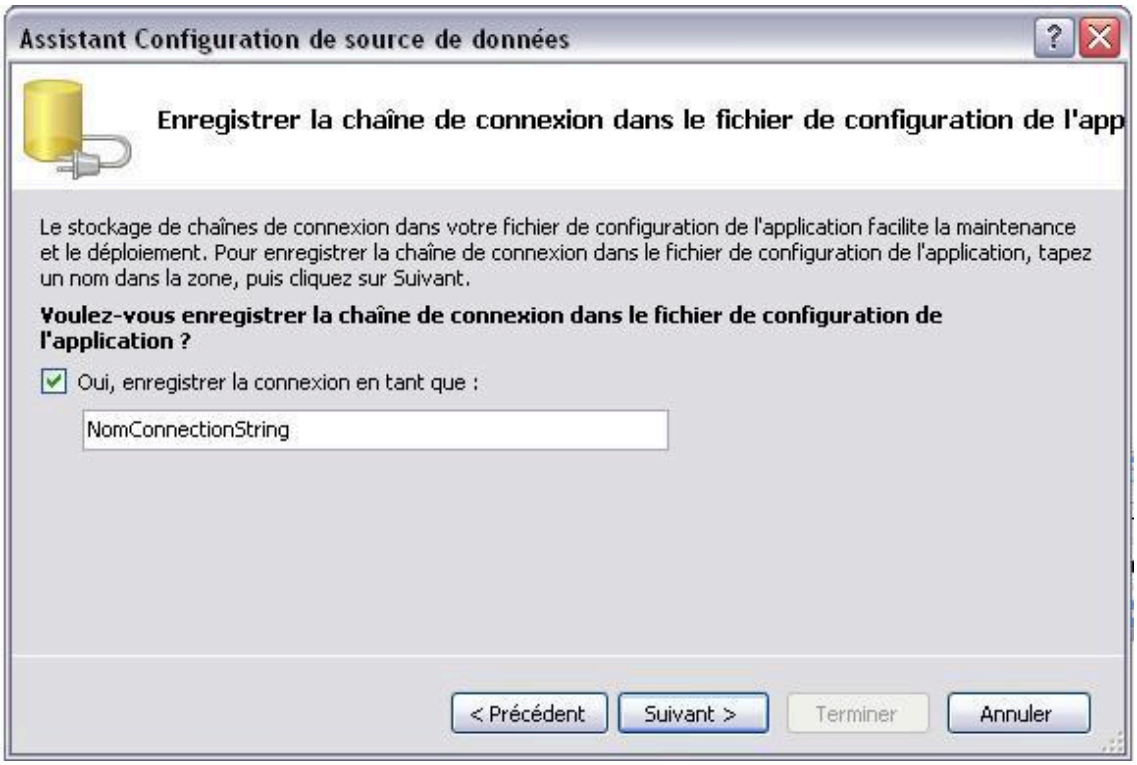



On retrouve le nom de la bd dans la zone connexion, cliquer sur "Suivant".

Ensuite VB vous demande s'il faut copier la BD dans le projet : si oui la BD sera copiée dans le répertoire de travail quand vous installerez votre application. À vous de voir.

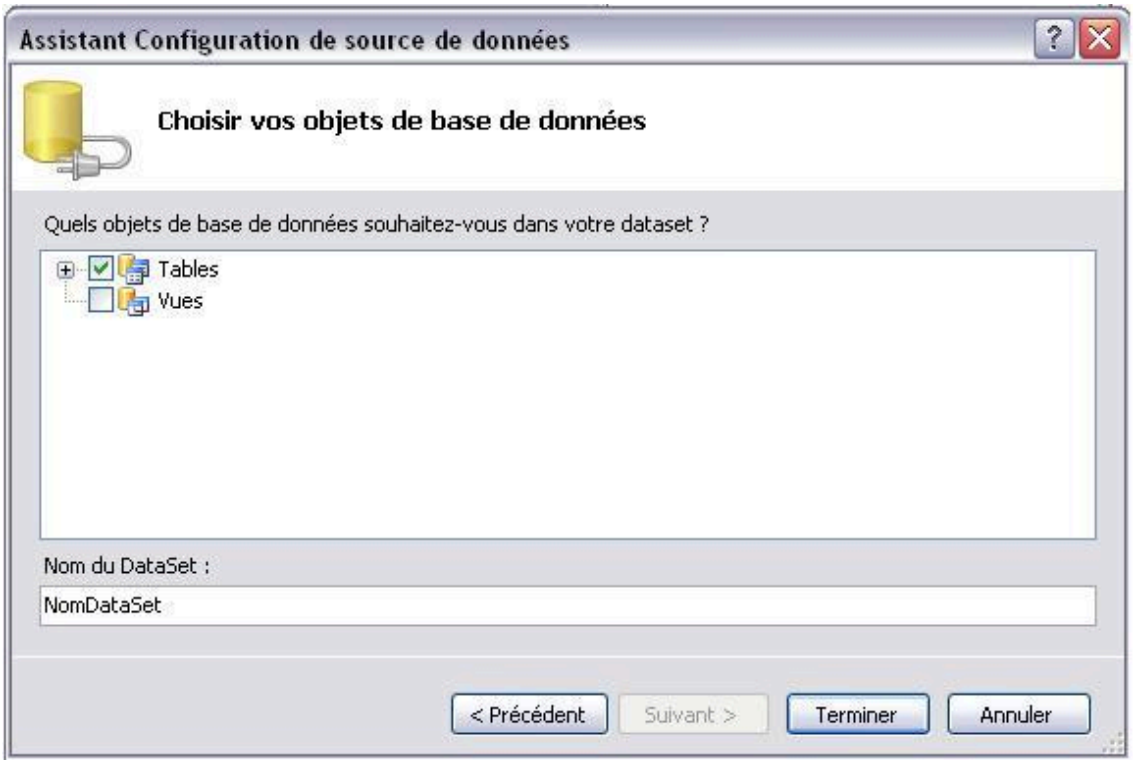



Ensuite VB vous propose d'enregistrer les chaines de connexion (nom de la base…) dans le fichier de configuration afin qu'elles soient stockées et lues lors de l'utilisation ultérieure. (automatiquement bien sûr). À vous de voir.



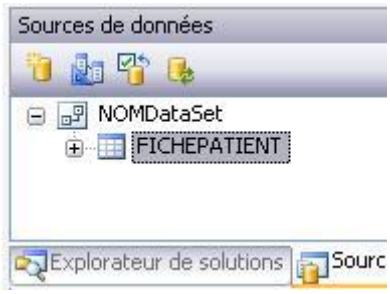

Ensuite il faut choisir dans la base les objets qui seront chargés dans le DataSet.

C'est habituellement les Tables. Cocher 'Tables' dérouler puis cocher FICHEPATIENT.




Cliquer sur "Terminer", la source de données a été créée.

Pour voir la source de données, cliquer dans le menu 'Données', 'Voir les sources de données'.




On voit bien le DataSet qui a été créé (NOMDataSet) et qui contient une table FICHEPATIENT.

**2 - Remplir le DataSet**

On utilisera la méthode Fill du TableAdapter pour remplir le DataSet.

```
Private Sub Form1_Load

'Création d'un TableAdapter

Dim FPTableAdapter = New NOMDataSetTableAdapters.FICHEPATIENTTableAdapter

'Extraction des données et chargement du DataSet

FPTableAdapter.Fill(NOMDataSet.FICHEPATIENT)

End Sub
```

Voila, le DataSet est chargé puis déconnecté.

**3 - Interroger le DataSet avec Linq**

Afficher tous les garçons.

Le champ SEXE contient 'M' ou 'F' le 'filtre' Where est donc simple et utilise '=':

LePatient.SEXE = "M"

Cela donne :

```
Dim query = _

From LePatient In NOMDataSet.FICHEPATIENT.AsEnumerable() _

Where LePatient.SEXE = "M" _

Select LePatient
```

Et pour afficher les noms et prénoms dans une ListBox :

```
For Each P In query

ListBox1.Items.Add(P.NOM.ToString & P.PRENOM.ToString)

Next
```

FICHEPATIENT étant une Table (qui n'a pas l'interface IEnumerable), il faut ajouter .AsEnumerable().

Afficher tous les "PHILIPPE"

Comme notre champ PRENOM est de 20 caractères et qu'il contient des espaces, on ne peut utiliser '='; on va donc utiliser **Like** qui recherche si LePatient.PRENOM contient le modèle "PHILIPPE*".

```
Dim query = _

From LePatient In NOMDataSet.FICHEPATIENT.AsEnumerable() _

Where LePatient.PRENOM Like "PHILIPPE*" _

Select LePatient
```

Attention, dans le modèle il y a des caractères génériques :

? veut dire 1 caractère quelconque ;

* veut dire 0 ou plusieurs caractères quelconques.

(Ce n'est pas les caractères'_' et '%' comme dans SQL!!!)

Afficher tous les "PHILIPPE", mais en créant une nouvelle variable de requête.

```
Dim query =  From LePatient In NOMDataSet.FICHEPATIENT.AsEnumerable() _

Where LePatient.PRENOM Like "PHILIPPE*" _

Select New With { _

.LeNom = LePatient.Field(Of String)("NOM"), _

.LeSexe = LePatient.Field(Of String)("SEXE")}


For Each P In query

ListBox1.Items.Add(P.LeNom.ToString & P.LeSexe.ToString)

Next
```

Ici la structure de la variable de requête (qui contient les résultats) est créée de toutes pièces par Select New With {}, on s'est contenté de créer des champs avec un nouveau nom.

## C - Linq To SQL

DataContext permet une connexion à la base.

```
Dim db As New DataContext("&#8230;\Northwnd.mdf") '
```

Récupérer une table :

```
Dim Customers As Table(Of Customer) = db.GetTable(Of Customer)()


 ' Interrogation sur Customer en recherchant 'London'.

Dim Query = _

  From cust In Customers _

  Where cust.City = "London" _
```

```
   Select cust


 'Affichage résultat sur la console
For Each cust In Query
      Console.WriteLine("id=" & cust.CustomerID & _ ", City=" & cust.City)
Next
```

On peut aussi exécuter des commandes sur le DataContext (exemple : augmenter de 100 le prix unitaire) :

```
db.ExecuteCommand ("UPDATE Products SET UnitPrice = UnitPrice + 100")
```

# XVIII - Optimisation en vitesse

## XVIII-A - Comparaison VB6 VB.Net

**VB.NET est-il rapide ?**

### XVIII-A-1 - Comment VB.NET 2003 est situé en comparaison avec les autres langages de programmation ?

Le site OsNews.com publie les résultats d'un petit benchmark comparant les performances d'exécution sous Windows de plusieurs langages de programmation.

Les langages .NET - et donc le code managé en général - n'ont pas à rougir devant Java, pas plus que face au langage C compilé grâce à GCC.

**- Nine Language Performance Round-up: Benchmarking Math & File I/O [OsNews.com]**

Article publié sur **www.DotNet-fr.org**

### XVIII-A-2 - Vitesse de VB6, VB.NET 2003, 2005, 2008, 2010

**En mode Design, pour écrire un programme, l'IDE VB 2008 est beaucoup plus rapide que VB 2005.**

**En exécution par contre…**

Exemple No1

**Sur une même machine P4 2.4 G faisons tourner un même programme : 2 boucles imbriquées contenant une multiplication, l'addition à un sinus et l'affichage dans un label.**

- **En VB6**

```
Private Sub Command1_Click()
Dim i As Long
Dim j As Long
Dim k As Long
For i = 0 To 100
For j = 0 To 1000
Label1.Caption = Str(k * 2 + Sin(4)): Label1.Refresh
k = k + 1
Next
Next
End Sub
```

9 secondes dans l'IDE, **7 secondes avec un exécutable** après compilation.

- **En Vb.Net 2003**

```
Imports System.Math

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
 Handles Button1.Click
Dim i As Integer
Dim j As Integer
Dim k As Integer
For i = 0 To 100
For j = 0 To 1000
Label5.Text = (k * 2 + Sin(4)).ToString : Label5.Refresh()
k = k + 1
Next
Next
End Sub
```

35 secondes dans l'IDE, **25 secondes avec un exécutable** après compilation.

En utilisant des 'Integer' ou des 'Long', il y a peu de différence.

- **En Vb.Net 2005 en Vb 2008**


Même code.

55 secondes dans l'IDE, **45 secondes avec un exécutable** après compilation.

**Dur, dur !!!.**
Pratiquement la même vitesse en Vb 2008.

- **En Vb.Net 2008 et vb 2010**


En faisant tourner le même programme sur une autre machine :
Vb 2008=47 s
Vb 2010=43 s
Vb 2010 parait donc légèrement plus rapide que Vb 2008.

Exemple No2

**Sur une même machine P4 2.4 G faisons tourner un même programme : on crée un tableau de 10 000 String dans lequel on met des chiffres. Puis on trie le tableau.**

- **En VB6**

```
Private Sub Command1_Click()

Dim i As Integer
Dim A(10000) As String
Dim j As Integer
Dim N As Integer
Dim Temp As String
N = 9999
'remplir le tableau
For i = 9999 To 0 Step -1
    À(i) = Str(9999-i)
Next i
'trier
For i = 0 To N - 1
    For j = 0 To N - i - 1
```

```
        If A(j) > A(j + 1) Then
             Temp = À(j): À(j) = À(j + 1): À(j + 1) = Temp
         End If
 Next j
Next i
End Sub
```

**35 secondes.**

- **En Vb.Net 2003**

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
 Handles Button1.Click
Dim i As Integer
Dim A(10000) As String
For i = 9999 To 0 Step -1
A(i) = (9999 - i).ToString
Next i
Array.Sort(A)
Label1.Text = "ok"
End Sub
```

**< 1 seconde**

- **En Vb.Net 2005 beta2**


Même code.

**< 1 seconde**

Moins d'une seconde avec VB.NET, 35 secondes en VB6.

**La méthode 'Sort' est hyper plus rapide que la routine de tri.**

**(Pour être honnête, il faut dire que mon test n'est pas rigoureux, car le tri VB.NET est probablement un tri 'rapide' alors qu'en VB6 la routine basic n'est pas optimisée, je ne compare donc pas les mêmes routines.)**

Plusieurs remarques (merci Clement)

1. Le tableau employé n'est pas représentatif : il n'est pas constitué d'éléments à répartition aléatoire.

2. Ce ne sont pas les mêmes routines. Vous utilisez une version optimisée du bubble sort, le tri à bulle, de complexité $O(n^2)$. Dès lors, pour trier mille éléments, avec une complexité de comparaison de deux string en O(1), il faut approximativement un million d'opérations dans le pire des cas.

3. Le Framework emploie un tri fusion, c'est-à-dire un tri de complexité O(n*log(n)), soit environ 10 000 opérations (le log est un log binaire en réalité). Il ne s'agit pas d'optimiser la routine de tri, mais de la changer.

4. Si on code effectivement un tri fusion en VB 6, alors la routine 'Sort' en VB.net n'est pas "hyper plus rapide" que la routine de tri, mais la routine de tri est fausse.

## En conclusion

La couche du Framework semble ralentir **considérablement la vitesse du code.**

Mais, en VB.net, **il faut raisonner différemment** et utiliser judicieusement les classes et les méthodes au lieu de taper de longues routines.

**Cela fait qu'en VB.Net :**

**le code est plus court et compact (moins de temps de développement) ;**

**le code est plus rapide.**

**Vitesse comparée Windows Forms WPF**

Thomas Lebrun dans son blog teste le remplissage de 10 000 objets dans une ComboBox ListView Treeview et les trie en vb 2008.

"D'une manière générale, les applications WPF sont plus performantes que les applications WindowsForms. Seul le contrôle ListView semble déroger à ce constat, mais la différence est sans doute suffisamment infime pour qu'elle soit ignorée."

Un bémol : la position d'un contrôle dans une grid un panel peut influencer fortement la vitesse s'il y a une gestion d'affichage différente.

Enfin l'usage immodéré d'effets visuels dans une interface ralentit et consomme du temps CPU.

## XVIII-B - Chronométrer le code, compteur temps mémoire…

**On veut comparer 2 routines et savoir laquelle est la plus rapide.**

On peut pour compter le temps écoulé :

utiliser un Timer ;

utiliser l'heure système ;

utiliser la Classe Environment.Tickcount ;

appeler QueryPerformanceCounter, une routine de Kernel32 ;

utiliser System.Diagnostics.Stopwatch (Framework 2 Vb 2005).

### XVIII-B-1 - Pour chronométrer un événement long

Entendons par événement long, plusieurs secondes ou minutes.

Trois solutions :

- on utilise un **Timer**, (dans l'événement Ticks qui survient toutes les secondes, une variable s'incrémente comptant les secondes).**(4-5)** ;

- on peut utiliser **l'heure Système :**

```
Dim Debut, Fin As DateTime

Dim Durée As TimeSpan
```

```
Debut=Now

….Routine….

Fin=Now

Durée=Fin-Debut
```

(Utiliser Durée.ToString pour afficher.) ;

- on peut utiliser la variable **Environment.tickcount()**

```
Dim Debut, Fin As int32

Debut=Environment.tickcount()

….Routine….

Fin=Environment.tickcount()
Durée=Fin-Debut
```

Cela donne un temps écoulé en ms, sachant toutefois que le compteur de ticks (1tick=100 nanosecondes) n'est mis à jour que toutes les 16 ms environ…

## XVIII-B-2 - Créer un compteur pour les temps très courts (Framework 1, VB2003)

C'est le cas pour chronométrer des routines de durée bien inférieure à une seconde.

Cela semblait à première vue facile !!!

J'ai en premier lieu utilisé un **Timer**, (dans l'événement Ticks un compteur de temps s'incrémente), mais les intervalles de déclenchement semblent longs et aléatoires.

J'ai ensuite utilisé l'heure système.

Mais 'Durée' est toujours égal à 0 pour les routines rapides, car il semble que Now ne retourne pas les millisecondes ou les Ticks.

Le temps écoulé est en ms, sachant toutefois que le compteur de ticks (1tick=100 nanosecondes) n'est mis à jour que toutes les 16 ms environ…(15,62468 ms) Now utilise la même horloge. En ce qui concerne le timer, tout intervalle inférieur à 15 donnera le même résultat, 15,6 ms d'attente. À noter également pour le timer que le délai d'attente ne commence pas au top précédent, mais à la fin de la procédure appelée par ce timer, ce qui peut augmenter sa période.

J'ai trouvé la solution chez Microsoft.

Utilisation d'une routine de Kernel32 qui retourne la valeur d'un compteur (QueryPerformanceCounter).

De plus QueryPerformanceFrequency retourne le nombre de fois que le compteur tourne par seconde.

Exemple : comparer 2 boucles, l'une contenant une affectation de variable tableau (b=a(2)) l'autre une affectation de variable simple (b=c), on gagne 33 %.

```
Declare Function QueryPerformanceCounter Lib "Kernel32" (ByRef X As Long) As Short

Declare Function QueryPerformanceFrequency Lib "Kernel32" (ByRef X As Long) As Short
```

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

Dim debut As Long

Dim fin As Long

Dim i As Long

Dim a(5) As String

Dim b As String

Dim c As String

Dim d1 As Long

Dim d2 As Long

'**********première routine

QueryPerformanceCounter(debut)

For i = 0 To 10000

b = a(2)

Next

QueryPerformanceCounter(fin)

d1 = fin - debut

Label1.Text = d1.ToString

'**********seconde routine

QueryPerformanceCounter(debut)

c = a(2)

For i = 0 To 10000

b = c

Next

QueryPerformanceCounter(fin)

d2 = fin - debut

Label2.Text = d2.ToString


Label5.Text = "Gain 2e routine:" & 100 - Int(d2 / d1 * 100).ToString

End Sub
```

C'est cette routine qui est utilisée pour étudier l'**optimisation du code**.

**Elle n'est pas parfaite, car sujette à variation :** les valeurs sont différentes d'un essai à l'autre en fonction des processus en cours !

Avec VB .Net (version antérieure au Build 1.1.4322.SP1) le problème est catastrophique : 2 routines identiques ne donnent pas les mêmes temps (bug du compilateur VB ?).

Il reste que le problème de la variabilité des performances est absolument inévitable puisque Windows est un système préemptif qui peut prendre le contrôle de l'ordinateur à tout moment.

C'est pourquoi la seule façon fiable de faire des tests consiste à avoir une **approche expérimentale :** on répète la mesure un certain nombre de fois et on fait une moyenne des essais individuels.

Pour des routines très très rapides, une autre façon de faire consiste à **effectuer des tests sur de longues durées** (en répétant des milliers de fois la routine rapide à l'aide d'une boucle) pour arriver à des temps de plusieurs dizaines de secondes, de manière à diluer les interventions de Windows dans le processus.

Il y a peut-être d'autres solutions ?

## XVIII-B-3 - Créer un compteur pour les temps très courts (Framework 2, VB2005)

Le Framework 2.0 comporte une classe qui encapsule l'utilisation de l' API QueryPerformanceCounter : **System.Diagnostics.Stopwatch.**

```
Dim Stopwatch As System.Diagnostics.stopWatch = System.Diagnostics.Stopwatch.StartNew()

' Code à mesurer
MsgBox(stopWatch.ElapsedMilliseconds.ToString())
```

La première ligne crée une instance d'un Stopwatch et démarre le compteur.

La propriété ElapsedMilliseconds nous donne le temps mesuré en millisecondes. Attention, pour mesurer le temps de fonctionnement d'une routine, les millisecondes c'est trop long !!

On a également la propriété ElapsedTicks pour compter les Ticks (1 Tick : 100 nanosecondes).

```
Dim S As System.Diagnostics.stopWatch = System.Diagnostics.Stopwatch.StartNew()

'routine à mesurer

S.Stop()
Dim t1 As Decimal = S.ElapsedTicks
        MsgBox( t1.ToString)
```

Enfin si l'on veut compter en TimeSpan, utilisons la propriété Elapsed.

On peut arrêter et redémarrer le compteur plusieurs fois.

```
stopwatch.Stop     'arrête le compteur

stopwatch.Start    'fait redémarrer le compteur

stopwatch.Reset()  'remet le compteur à zéro

stopwatch.Restart()  'redémarre le compteur (à partir de VB 2010)
```

## XVIII-B-4 - Compteur de performance

On généralise aux compteurs de performance qui peuvent calculer des temps, mais aussi de la mémoire…

Un compteur de performance surveille le comportement d'un élément de performance mesurable, sur un ordinateur. (Composants physiques tels que processeurs, disques et mémoire, objets système : processus et threads).
Il faut créer une instance de la classe **PerformanceCounter**, puis indiquer la catégorie avec laquelle le composant doit interagir, puis choisir un compteur de cette catégorie.

Par exemple, dans la catégorie Memory des compteurs système assurent le suivi des données en mémoire, telles que le nombre d'octets disponibles et le nombre d'octets mis en cache.
Les catégories les plus utilisées sont : Cache, Memory, Objects, PhysicalDisk, Process, Processor, Server, System et Thread.

Ici on va utiliser la catégorie 'Process' (processus, programme), le compteur 'octet' et on va surveiller le programme 'Explorer'.
À chaque fois qu'on exécute ce code, il donne la quantité de mémoire utilisée.

```
Dim PC As New PerformanceCounter()
PC.CategoryName = "Process"
PC.CounterName = "Private Bytes"
PC.InstanceName = "Explorer"
MessageBox.Show(PC.NextValue().ToString())
```

NextValue retourne le contenu du compteur.

Il est possible de compter du temps, mais la doc avec les noms des compteurs est très difficiles à trouver !! Si vous avez une adresse !!

## XVIII-C - Comment accélérer une application VB.net ?

**L'optimisation** est la pratique qui consiste généralement à réduire le temps d'exécution d'une fonction, l'espace occupé par les données et le programme. Elle ne doit intervenir qu'une fois que le programme fonctionne et répond aux spécifications. L'expérience montre qu'optimiser du code avant revient le plus souvent à une perte de temps et s'avère néfaste à la clarté du code et au bon fonctionnement du programme.

### XVIII-C-1 - Utilisation des nouvelles fonctionnalités

**Il faut raisonner différemment** et **utiliser judicieusement les classes et les méthodes** au lieu de taper de longues routines.

Exemple

**La méthode 'Sort' d'un tableau est hyper plus rapide que la routine de tri écrite en code.**

```
Array.Sort(A)
```

est hyper plus rapide que :

```
For i = 0 To N - 1
    For j = 0 To N - i - 1

        If A(j) > A(j + 1) Then
            Temp = À(j): À(j) = À(j + 1): À(j + 1) = Temp
        End If

    Next j
Next i
```

**La méthode BinarySearch de la Classe Array est hyper plus rapide que n'importe quelle routine écrite en code pour rechercher un élément dans un tableau trié.**

```
I=Array.BinarySearch(Mois, "Février")
```

## XVIII-C-2 - Choix des variables

Les types de données les plus efficaces sont ceux qui emploient la largeur de données native de la plateforme d'exécution. Sur les plateformes courantes, la largeur de données est 32 bits, pour l'ordinateur et le logiciel d'exploitation.

Sur les ordinateurs actuels et en VB la largeur de donnée native est donc de 32 bits.

Pour les entiers les **Integer sont donc les plus rapides,** car le processeur calcule en Integer. Viennent ensuite les **Long, Short,** et **Byte**.

Dans les nombres en virgule flottante, les **Double** sont les plus rapides, car le processeur à virgule flottante calcule en Double, ensuite ce sont les **Single** puis les **Decimal**.

Les Calculs en Decimal sont 10 fois plus lents que les calculs en Single.

**Si c'est possible, utiliser les entiers plutôt que les nombres en virgules flottantes, car le travail sur les nombres entiers est beaucoup plus rapide.**

**Bon choix des unités**

Exemple : pour stocker les dimensions d'une image, on utilisera les pixels : l'image aura un nombre entier de pixels et on peut ainsi utiliser une variable Integer, alors que si on utilise les centimètres on devra travailler sur des fractionnaires donc utiliser par exemple des Single ce qui est plus lent.

**L'usage de constantes est plus rapide que l'usage de variable**, car la valeur d'une constante est directement compilée dans le code.

Pour stocker une valeur, une variable est plus rapide qu'une propriété d'objet.

**Les variables 'par valeur' peuvent être plus rapides que celles 'par référence'**. Les premières sont stockées directement dans la pile, les secondes sur le 'tas'.

Si vous utilisez une valeur entière, créer une variable Integer et non une variable Object.

**Typer le plus possible les variables**
Fuyez le plus possible les variables objets : si vous affectez un Integer à une variable objet, le CLR doit faire des opérations de boxing, avec recopie de la valeur et enregistrement du type de la variable ce qui prend du temps, et l'inverse pour UnBoxing !
Vous pouvez éliminer le boxing/unboxing par inadvertance par la mise en Option Strict On.

Une variable est à liaison tardive si elle est déclarée en tant que type objet et non d'un type de données explicite. Lorsque votre code accède à des membres d'une telle variable, le Common Language Runtime est obligé d'effectuer la vérification de type et de recherche de membres au moment de l'exécution.

## XVIII-C-3 - Tableau

**Le CLR est optimisé pour les tableaux unidimensionnels.** Employer le moins de dimensions possible dans un tableau.

**L'usage des tableaux de tableaux 'A(9),(9)' est plus rapide** que les tableaux multidimensionnels 'A(9,9)'.

**Tableau ou Collections ?**

Pour rechercher un élément dans un ensemble d'éléments à partir de son index, utilisez un tableau (l'accès à un élément d'index i est plus rapide dans un tableau que dans une collection).

**L'accès a une variable simple est plus rapide que l'accès à un élément d'un tableau.**

**Si vous utilisez de nombreuses fois à la suite le même élément d'un tableau, le mettre dans une variable simple, elle sera plus rapide d'accès :**

```
Dim a As Integer= P(6)

b=a*3

c=a+2

….

z=a*5
```

Est plus rapide que :

```
b=P(6)*3

c=P(6)+2

….

z=P(6)*5
```

**L'usage d'un tableau est plus rapide qu'une multitude de SelectCase ou de If Then.**

Exemple : obtenir le nom du mois en fonction de son numéro d'ordre.

```
Dim Mois() As String ={Janvier,Février,Mars,Avril,Mai,Juin,Juillet}

nomDuMois=Mois(i-1)
```

Est plus rapide que :

```
Select Case i

    Case 1

        nomDuMois="Janvier"

    Case 2

        nomDuMois="Février"

    ……

End Select.
```

**Pour rechercher rapidement un élément dans un tableau**

Utiliser la méthode Binaryscearch plutôt que IndexOf.

Pour la méthode Binaryscearch, le tableau doit être trié. (Le trier avant la recherche).

**On peut utiliser des méthodes génériques (VB 2005) pour travailler sur les tableaux. c'est beaucoup plus rapide.**

Exemple recherche dans un tableau de short nommé monTab l'élément 2 :

index= Array.indexOf (Of Short)(monTab, 2) est **hyper plus rapide que**

index= Array.indexOf (monTab, 2), car la première version avec généric est directement optimisée pour les Short.

Il est est de même pour Binarysearch et Sort.

Cela est valable pour les types 'valeur' (peu d'intérêts pour les strings par exemple).

## XVIII-C-4 - Collections

Si on ne connait pas le nombre d'éléments maximum et que l'on doit ajouter, enlever des éléments, il vaut mieux utiliser une collection (ArrayList) plutôt qu'un tableau avec des Dim Redim Preserve. Mais attention une collection comme ArrayList est composée d'objets, ce qui implique une exécution plus lente.

Pour rechercher un élément dans un ensemble d'éléments à partir d'un index, utilisez un tableau.

Pour rechercher un élément dans un ensemble d'éléments à partir d'une clé (KeyIndex), utilisez une collection (l'accès à un élément ayant la clé X est plus rapide dans une collection que dans un tableau, dans un tableau il faut en plus écrire la routine de recherche).

En VB2005 on peut utiliser les Collections génériques plus rapides, car typées (les éléments ne sont pas des Objets).

## XVIII-C-5 - Éviter la déclaration de variables 'Objet' et les liaisons tardives, les variables non typées

**Éviter de créer des variables Objet.**

Pour créer une variable et y mettre une String :

```
Dim A  crée un 'Objet' A
```

Il est préférable d'utiliser :

```
Dim A As String
```

La gestion d'un objet est plus lente que la gestion d'une variable typée.

Il faut aussi éviter **les liaisons tardives :** une liaison tardive consiste à utiliser une variable Objet. À l'exécution, donc tardivement, on lui assigne un type, une String ou un Objet ListBox par exemple. Dans ce cas, à l'exécution, VB doit analyser de quel type d'objet il s'agit et le traiter, alors que si la variable a été déclarée d'emblée comme une String ou une ListBox, VB a déjà prévu le code nécessaire en fonction du type de variable. **Utilisez donc des variables typées.**

**Utilisez donc des variables le plus typées possible.**

Si une variable doit être utilisée pour une assignation de Button, ListBox… plutôt que la déclarer en Objet, il est préférable de la déclarer en System.Windows.Forms.Control

**Utilisez donc des variables ByVal plutôt que ByRef.** Les types ByVal sont gérés sur la pile, les types ByRef sur 'le tas' c'est plus long.

De manière générale, si le compilateur sait quel type de variable il utilise, il fait des contrôles lors de la compilation, s'il ne sait pas, il fait des contrôles lors de l'exécution et cela prend du temps à l'exécution.

## XVIII-C-6 - Utiliser les bonnes 'Options'

Option Strict On permet de convertir les variables de manière explicite et accélère le code. De plus, on est poussé à utiliser le bon type de variable.

Si on affecte une valeur 'par référence' à un objet, le CLR doit créer un objet, transformer la valeur, la mettre dans l'objet et gérer le pointeur (on nomme cela 'boxing'), c'est très long. L'inverse c'est du 'unboxing'.

Si on utilise Option Strict Off le boxing se fait automatiquement et systématiquement et c'est long.

Si on utilise Option Strict On, on a tendance (ce qui est bien) à moins utiliser des variables de types différents, ce qui diminue le nombre de boxing-unboxing, et cela oblige si on utilise des variables différentes à caster à l'aide d'instructions qui sont plus rapides.

**Donc utiliser Option Strict On et choisir des variables du même type dans une routine afin de réduire au minimum les conversions.**

Choisir les méthodes de conversion, quand elles sont nécessaires, les plus typées possible.

Pour les conversions en entier par exemple CInt est plus rapide que CType, car CInt est dédié aux entiers.

Option Compare Binary accélère les comparaisons et les tris (la comparaison binaire consiste à comparer les codes Unicode des chaines de caractères).

## XVIII-C-7 - Pour les fichiers, utiliser System.IO

L'utilisation des **System.IO** classes accélère les opérations sur fichiers (en effet, les autres manières de lire ou d'écrire dans des fichiers comme les FileOpen font appel à System.IO : autant l'appeler directement !!)

Utiliser donc :

- **Path, Directory,** et **File ;**
- **FileStream** pour lire ou écrire ;
- **BinaryReader** and **BinaryWriter** pour les fichiers binaires ;
- **StreamReader** and **StreamWriter** pour les fichiers texte.

Utiliser des buffers entre 8 et 64 K.

## XVIII-C-8 - If Then ou Select Case ?

Plutôt qu'un If Then et une longue série de ElseIf, il est préférable d'utiliser un SelectCase qui en Vb est plus rapide (20 %).

Dans les Select Case **mettre les 'case' fréquents et qui reviennent souvent en premier**, ainsi il n'y a pas besoin de traiter tous les Case :

```
Select Case Variable

Case Valeur1

…. Valeur1 revient souvent
```

```
Case Valeur2

….

….

Case Valeur25

…. Valeur25 survient rarement

End Select
```

Plutôt qu'un If B= True Then… il est préférable d'utiliser un If B Then… cela va 2 fois plus vite en VB. B étant une opération booléenne quelconque. En effet, entre If et Then l'expression est de toute façon évaluée pour savoir si elle est = True.

## XVIII-C-9 - Utiliser les bonnes 'Opérations'

**Si possible :**

**Utiliser : "\"**

Pour faire une vraie division, on utilise l'opérateur '/'

Si on a seulement besoin du quotient d'une division (et pas du reste ou du résultat fractionnaire) on utilise '\', c'est beaucoup plus rapide.

**Utiliser : "+="**

A+= 2 est plus rapide que A= A+2

**Utiliser : AndAlso et ElseOr**

AndAlso et ElseOr sont plus rapides que And et Or.

(Puisque la seconde expression n'est évaluée que si nécessaire.)

**Arrêter le test lorsqu'on connait la réponse :**

```
if x<3 And y>15 then
```

Les 2 expressions sont évaluées x<3 et x>15 puis le And est évalué alors que dès que x<3 on pourrait arrêter de tester.

Solution :

```
if x<3 then

    If y>15 then

…
```

**Réduire les opérations gourmandes**

Remplacer une multiplication par une addition quand c'est possible.

Les fonctions trigométriques (Sinus Cosinus…), Log, Exp… sont très gourmandes.

(Je me souviens d'un programme, en QuickBasic !! qui affichait de la 3D, plutôt que de calculer plein de sinus, on allait chercher les sinus stockés dans un tableau, cela entrainait un gain de temps phénoménal.)

**Calculer des expressions à l'avance**

- Log(2) est très long à calculer surtout s'il est dans une boucle.

```
For i=1 to 100000

    R=i*P+ Log(2)

next i
```

utiliser plutôt le résultat calculé à la main :

```
For i=1 to 100000

    R=i*P+ 0.693147

next i
```

De même si on utilise un membre d'une classe :

```
For i=1 to 100000

    R=i*P+ Myobjet.MyPropriété

next i
```

mettre la valeur de la propriété dans une variable simple, c'est plus rapide :

```
Dim valeur As Integer = Myobjet.MyPropriété

For i=1 to 100000

    R=i*P+ valeur

next i
```

- L'accès à un élément d'un tableau est lent :

```
For i=1 to 100000

    R=i*P+ MyTableau (2 ,3 )

next i
```

mettre la valeur du tableau dans une variable simple, c'est plus rapide si on utilise cette valeur 10 000 fois :

```
Dim valeur As Integer= MyTableau (2 ,3)

For i=1 to 100000

    R=i*P+ valeur

next i
```

**Pour les conversions, utilisez DirectCast plutôt que CType**

CType est moins rapide.

### Utiliser les conversions typées plutôt que CType

Faire d=Cdbl(i) plutôt que d= CType(i, Double)

Cdbl est fait pour convertir en Double alors de CType qui convertit 'tout' en 'tout' doit analyser en quel type, il faut convertir puis appeler les routines correspondantes.

## XVIII-C-10 - Utiliser : With End With

With… End With accélère le code :

```
With Form1.TextBox1
    .BackColor= Red
    .Text="BoBo"
    .Visible= True
End With
```

est plus rapide que :

```
Form1.TextBox1.BackColor= Red
Form1.TextBox1.Text="BoBo"
Form1.TextBox1.Visible= True
```

car Form1.TextBox1 est 'évalué' 1 fois au lieu de 3 fois.

## XVIII-C-11 - Optimiser les boucles

### En mettre le moins possible dans les boucles

Soit un tableau J(100,100) d'entiers:

Soit un calcul répété 100 000 fois sur un élément du tableau, par exemple :

```
For i=1 to 100000
    R=i*J(1,2)
next i
```

On va 100 000 fois chercher un élément d'un tableau, c'est toujours le même !

Pour accélérer la routine (c'est plus rapide de récupérer la valeur d'une variable simple plutôt d'un élément de tableau), on utilise une variable intermédiaire P :

```
Dim P as Integer
P=J(1,2)
For i=1 to 100000
    R=i*P
next i
```

c'est plus rapide.

De la même manière, si on utilise une propriété (toujours la même) dans une boucle, on peut stocker sa valeur dans une variable, car l'accès à une variable simple est plus rapide que l'accès à une propriété.

**Les opérations qui ne sont pas modifiées dans la boucle doivent donc être mises à l'extérieur.**

**Éviter les On Error dans des grandes boucles**, qui ralentissent considérablement, par contre, contrairement à ce qu'on entend, le Try Catch Finally dans une très grande boucle ralentit très peu.

Dans une boucle tournant 1 000 000 000 de fois :

5 s sans gestion d'erreur ;

6 s si la boucle contient Try Catch ;

2 mn si la boucle contient on error resume next !!

**Fusionner plusieurs boucles si nécessaire**

Au lieu de faire 2 boucles :

```
For i=1 to 100000
    P(i)=i
Next i
For i=1 to 100000
    Q(i)=i
Next i
Faire:
For i=1 to 100000
    P(i)=i
    Q(i)=i
Next i
```

C'est possible quand les 2 boucles ont même valeur initiale et finale.

**En cas de boucles imbriquées, placer la boucle la plus grande à l'intérieur :**

```
For i=1 to 100    '100 itérations
For j=1 to 10     '100 X 10 itérations
    ….
Next j
Next i
100+(100X10) = 1100 itérations de compteur
```

```
For j=1 to 10      '10 itérations

For i=1 to 100     '100 X 10 itérations

    ….

Next j

Next i

10+(100X10) = 1010 itérations de compteur
```

Attention : le nombre de boucles est le même (1000), par contre le nombre d'itérations de compteur (à chaque fois qu'on passe par une instruction 'For') n'est pas le même.
Comptons les itérations :

```
Dim iteration, boucle, i, j As Integer

 Private Sub Button1_Click
        For i = 1 To 100     '100 iterations
            iteration = itération + 1
            For j = 1 To 10      '100 X 10 iterations
                iteration = iteration + 1
                boucle = boucle + 1

            Next j

        Next i
        MsgBox(iteration)
        '100+(100X10) = 1100 iterations de compteur

        iteration = 0 : boucle = 0

        For j = 1 To 10      '10 iterations
            iteration = iteration + 1
            For i = 1 To 100     '100 X 10 iterations
                iteration = iteration + 1
                boucle = boucle + 1

            Next i

        Next j
        MsgBox(iteration)
        '10+(10X100) = 1010 iterations de compteur

    End Sub
```

En conclusion :

(1100-1010)/1100 gain 8 % d'itérations de compteur en moins (le nombre de boucles restant égal.)

**Sortir avec exit for dès qu'on a trouvé ou qu'il n'y a plus rien à chercher**

Exemple : rechercher un élément dans un tableau avec une boucle.

Dès que l'élément a été trouvé ou que le tableau ne contient plus rien, on quitte la boucle avec un Exit For.

## XVIII-C-12 - Appel de procédure

**Si on appelle une petite procédure dans une grande boucle, on a parfois intérêt à mettre le contenu de la procédure directement dans la boucle. Cela évite les appels et retours. C'est plus rapide.**

Si on a :

```
For i=1 to 100000

        resultat= Calcul(i,j)

    Next i


    Functin Calcul (i As Integer, j As Integer)

        i=i+3

        Return i*j

    End Sub

    C'est plus rapide d'écrire:

    For i=1 to 100000

        i=i+3

        resultat= i*j

    Next i
```

Si la procédure est longue et complexe et surtout si on a besoin de l'utiliser à différents endroits dans le programme, il est préférable d'utiliser une procédure.

Ne multipliez pas les petites procédures courtes ce qui entraine dans une boucle un grand nombre d'appels, preférez l'appel d'une procédure unique.
Exemple : plutôt que d'appeler Calcul1 puis Calcul2 puis Calcul3 puis Calcul4, il est préférable de créer une procédure unique nommée Calculs qui fait tous les calculs, le découpage excessif s'il n'est pas absolument nécessaire diminue les performances.

## XVIII-C-13 - Usage de threads

**Il peut être judicieux d'utiliser des threads, pour accélérer certaines applications.**

## XVIII-C-14 - Comment accélérer quand on utilise des 'String' ?

**Utiliser & pour la concaténation de chaine plutôt que +.**

Utiliser :

```
    s &= "mon" & "ami"
```

plutôt que :

```
    s += "mon" + "ami"
```

**Utiliser les StringBuilder**

Exemple d'une opération coûteuse en temps :

```
Dim s As String = "bonjour"


    s += "mon" + "ami"
```

En réalité le Framework va créer 3 chaines en mémoire avec toutes les pertes en mémoire et en temps que cela implique.

Pour effectuer des opérations répétées sur les string, le framework dispose donc d'une classe spécialement conçue et **optimisée** pour ça : System.Text.StringBuilder.

Pour l'utiliser, rien de plus simple :

```
Dim sb As new System.Text.StringBuilder()

    sb.Append("bonjour")

    sb.Append("mon ami")

    Dim s As String

    s = sb.ToString()
```

La méthode *ToString* de la classe StringBuilder renvoie la chaine qu'utilise en interne l'instance de StringBuilder.

Pour comparer 2 StringBuilder utiliser la méthode Equals plutôt que =.

## XVIII-C-15 - Comment accélérer l'affichage ?

**Formater le plus vite possible le texte**

Pour mettre en forme des nombres et les afficher Format est puissant (Prise en charge de la culture…), mais si on peut utiliser ToString c'est plus rapide (ToString est aussi plus rapide que Cstr).

ChrW utilisé pour afficher un caractère et AscW sont plus rapides que Chr et Asc, car ils travaillent directement sur les Unicodes.

**Précharger les fenêtres et les données**

Quand une fenêtre en ouvre une autre, le temps de chargement est long, l'utilisateur attend !

Solution

En début de programme précharger les fenêtres en les rendant invisibles. Lors de l'utilisation de ces fenêtres, il suffira de les rendre visibles, ce qui est plus rapide que de les charger.

Certaines données (liste…) doivent être chargées une fois pour toutes, le faire en début de programme, lors de l'affichage de la fenêtre 'Splash' par exemple.

**Afficher les modifications en une fois dans un TextBox**

À chaque fois que l'on fait une modification de propriété (couleur, taille…) ou de contenu (texte dans un TextBox) Vb met à jour chaque modification. Si on modifie tout et que l'on réaffiche tout, cela va plus vite.

Pour le cas du TextBox ne pas faire :

```
TextBox1.Text = TextBox1.Text + "Bonjour"

TextBox1.Text = TextBox1.Text + ""Monsieur"
```

Faire :

```
Dim T as string

T = "Bonjour"

T &= "Monsieur"

TextBox1.Text = T
```

Le texte est affiché en une fois, en plus, cela ne 'clignote' pas.

**Rendre l'affichage de l'objet inactif, faire toutes les modifications puis réactiver l'affichage**

Cas d'affichage dans une grid (MsFlexGrid).

**Afficher en 2 fois dans une ListBox**

À l'inverse pour ne pas faire attendre un affichage très long, afficher le début (l'utilisateur voit apparaitre quelque chose à lire) il est occupé un temps, ce qui permet d'afficher le reste.

Exemple : remplir une listBox avec un grand nombre d'éléments : en afficher 5 rapidement puis calculer et afficher les autres. L'utilisateur à l'impression de voir la ListBox se remplir immédiatement.

Pour les images, utiliser le **DoubleBuffer**.

**Pour faire patienter l'utilisateur lors d'une routine qui dure longtemps** (et lui montrer que l'application n'est pas bloquée) :

- transformer le curseur en sablier en début de routine, remettre un curseur normal en fin de routine,
- utiliser une ProgressBar (pour les chargements longs par exemple).

## XVIII-C-16 - Utiliser les tableaux en mémoire plutôt que la lecture de fichiers sur disque

On a un fichier de 300 noms, plutôt que de lire 300 enregistrements sur disque pour rechercher le bon, charger en mémoire le fichier dans un tableau (en début de programme), la recherche sera ensuite, dans le tableau en mémoire, extrêmement rapide.

## XVIII-C-17 - Ce qui n'influence pas la rapidité du code

Les boucles For, Do, While ont toutes une vitesse identique.

Quand on appelle une procédure, c'est aussi rapide si elle est dans le même module ou un autre module.

## XVIII-C-18 - Compilation DLL

Le chargement de dll est gourmand en temps, moins il y en a, plus c'est rapide.

Les dll ne sont chargées qu'une fois.

La compilation en mode Debug fournit un code plus lent.

## XVIII-C-19 - En conclusion

Une optimisation sur une ou deux instructions apporte un gain de temps négligeable.

L'optimisation dans les grandes boucles est perceptible.

Le travail en mémoire plutôt que sur disque accélère considérablement.

**Les goulots d'étranglement sont les longues boucles et l'accès aux bases de données.**

# XIX - Bonnes adresses, bibliographie du site

## XIX-A - Mes livres

**Mes livres de chevet**

**Visual Basic .Net.** de Gilles Nicot MicroApplication.

Complet, traite de l'IDE, Windows Forms et Web Forms.

Bien pour ceux qui viennent de VB6, traite de VB 2003.

1092 pages, 31 euros, date un peu (2003).

**Tout sur le code.** de Steve McConnell Microsoft Press 2e édition.

Concevoir un logiciel de qualité dans tous les langages (VB, C, Java…). Ce n'est pas un cours de programmation, mais un ouvrage qui aide le programmeur à mieux écrire.

Pavé de 893 pages (2005), un peu cher (56 euros). Le meilleur ouvrage sur la programmation.

**Visual Basic 2005 Manuel de référence** de Microsoft de Francesco Balena , Microsoft Press 2006.

Haut niveau. Pas pour débutant.

## XIX-B - VB 2003 sur le Net

**Comprendre la Plateform NET** mémoire de JP Bobier de 2001 très pointu.

( http://www.developpez.biz/downloads/dotnet/MemoireDotNet.pdf )

**http://www.pise.info/vb-net**Cours très plaisant, plein d'humour, avec des exemples de code. Il faut déjà avoir des notions de programmation, mais les explications sont très claires.

(http://www.pise.info/vb-net)

**Un bon cours VB.Net Par S.Tahé.**

(http://tahe.developpez.com/dotnet/vbnet)

Cours très rigoureux, très bon niveau, bon complément. Sur developpez.com

À lire après le mien, en français. Débutant s'abstenir.

**De vb6 à VB.Net Par J-M Rabilloud** Sur developpez.com

(http://bidou.developpez.com/tutoriels/dotnet/migration)

3 pdf , en fait un cours sur vb.net, haut niveau. À lire, en français.

Débutant s'abstenir.

**Site VisualBasic.Net en français** VB 2003

(http://www.microsoft.com/france/vbasic/plan.mspx)

**MSDN 1 de Microsoft** La bible en français!! VB 2003

(http://msdn.microsoft.com/library/fre/)

**Exemples de petites applications VB 2003 par Microsoft:**

**101 exemples de programmes Vb 2003: une mine.**

## XIX-C - VB 2005

**Portail Vb 2005 en anglais**

(http://lab.msdn.microsoft.com/vs2005/default.aspx)

**Visual Studio 2005**chez Microsoft en français.

(http://www.microsoft.com/france/msdn/vbasic/default.mspx)

**MSDN 2** VB 2005

(http://msdn2.microsoft.com/en-us/library/ms123401(en-us,MSDN.10).aspx)

**WebCast VB 2005++** avec présentation,exemple de code commenté à vive voix ou filmé.

(http://www.microsoft.com/france/msdn/vbasic/decouvrez/coach.mspx)

**Exemples de petites applications VB2005 par Microsoft:**

**101 exemples de programme Vb 2005: une autre mine**

( http://msdn.microsoft.com/fr-fr/vbasic/ms789075.aspx)

**Vidéo Microsoft par le 'coach'** 

(http://www.microsoft.com/france/msdn/vbasic/decouvrez/coach.mspx)

**Recherche dans la base de connaissance de Microsoft (en Français)**

(http://support.microsoft.com/search/?adv=1)

**Centre de ressources de Microsoft pour développeur VisualBasic en français**

(http://www.microsoft.com/france/vbasic/default.mspx)

**MSDN, les Api Windows (Anglais)**

(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winprog/winprog/overview_of_the_windows_api.asp)

FAQ

**Windows Forms Faq de George Sheperd's en anglais très très bien**

(http://64.78.52.104/FAQ/WinForms/default.asp#92)

## XIX-D - VB 2008

**Téléchargement VB 2008 Express**

(http://www.microsoft.com/express/vb/Default.aspx)

**Msdn Framework 3.5**

(http://msdn.microsoft.com/fr-fr/library/aa139616.aspx)

**Msdn WPF**

(http://msdn.microsoft.com/fr-fr/library/ms754130.aspx)

## XIX-E - VB 2010

Vb 2010 utilise le Framework 4: **Voir la documentation Msdn du Framework 4**

**Exemples de code en VB2010**

## XIX-F - Sites dédiés au Visual Basic

**Developpez.Com** le meilleur site de développeur, une mine: Cours didacticiels, forum…


(http://dotnet.developpez.com)

**Faq Dot Net**

**Faq Visual Basic.Net**

**Poser une question dans le forum**

**C2i.fr**

(http://www.c2i.fr)

Les articles sont tous bons.

**MoteurPrg.com**

En anglais:

**VbNet**

**codeproject**

(http://www.codeproject.com)

**devcity.net**

## XIX-G - Convertisseur C# -> VB

**SharpDevelop** possède un convertisseur C#=>vb et l'inverse.

Cela énerve, dans certains sites Dot.Net, les exemples sont en C# et c'est pratique de les convertir!!

**Convertisseur C#->VB** (coller le code C# puis cliquer sur le bouton 'Generate VB Code')

(http://www.kamalpatel.net/ConvertCSharp2VB.aspx)

Un autre:

**Convertisseur C#->VB** (coller le code C# puis cliquer sur le bouton 'Generate VB Code')

(http://www.developerfusion.com/tools/convert/csharp-to-vb/)

**Convertisseur chez developpez.com**

(http://convertisseur.developpez.com/converter.aspx)

## XIX-H - SQL

Pour aller plus loin:

Série d'articles sur SQL chez developpez.com:

**https://sql.developpez.com/**

(http://sql.developpez.com/)

**Voir site pour le paramètre de connexions (SQL et autres bases)**

(http://www.connectionstrings.com/)

## XIX-I - Glossaire

**Glossaire informatique anglais->français**

(http://www.glossaire.be)

## Dictionnaire de développeur. developpez.com

(http://dico.developpez.com/html/0.php)

# XX - Annexes

## XX-A - Le codage de caractères ASCII ANSI UNICODE et UTF

Ici on parle de l'informatique en général.

**Chaque caractère possède un code caractère numérique**. Quand on utilise une chaine de caractères sur un ordinateur ou dans une page Web, ce sont les codes des caractères qui sont enregistrés ou transmis. Quand on affiche, les caractères correspondant aux codes sont affichés.

Ce code peut être d'une **longueur de 7, 8, ou 16 bits**.

**A chaque code correspond un glyphe** qui est une représentation graphique du caractère, le glyphe fait partie d'une font.

Exemple:

En ASCII (codage ASCII ),le caractère 97 correspondant au glyphe 'a' dans la font Courier New.

### XX-A-1 - Codage sur 7 bits : ASCII

L'American Standard Code for Information Interchange (**ASCII**) est en vigueur depuis les années 1980.

C'est un ancien codage des caractères sur 7 bits (ASCII pur) au départ correspondant au clavier américain sans accents.

Il comporte 128 caractères.

Les codes 0 à 31 correspondent aux caractères de contrôle. (Non imprimable : 9 = tabulation, 13 = retour chariot.)

Les codes 32 à 64 correspondent aux opérateurs et aux chiffres.( 32 =' ', 43 ='+', 49 ='1')

Les codes 65 à 90 représentent les majuscules. (65='A')

Les codes 91 à 122 représentent les minuscules. (91='a')

Il a été normalisé sous le nom d'ISO 646.

### XX-A-2 - Codage sur 8 bits

On est ensuite passé sur 8 bits (un octet) pour **l'ASCII étendu** contenant les accentués. On peut donc coder 256 (0 à 255) caractères.

Il a été normalisé sous le nom d'ISO 8859.

L'ISO 8859 a plusieurs cartes de caractères, ou **jeu de caractères**, ou **page de codes** (CharSet).

- La norme ISO 8859-1, dont le nom complet est ISO/CEI 8859-1 est appelée **Latin-1** ou alphabet latin numéro 1. Elle permet de représenter la plupart des langues de l'Europe occidentale : l'albanais, l'allemand, l'anglais, le catalan, le danois, l'espagnol, le féroïen, le finnois, le français, le galicien, l'irlandais, l'islandais, l'italien, le néerlandais, le norvégien, le portugais et le suédois.


- Le jeu ISO 8859-2 est appelé **Latin 2** et permet de représenter les langues slaves et d'Europe centrale qui utilisent l'alphabet latin. L'allemand, le croate, le hongrois, le polonais, le roumain, le slovaque, le slovène et le tchèque.

- Le jeu de caractères ISO 8859-3 est utilisé pour l'espéranto, le galicien, le maltais et le turc.

…

- Le jeu ISO-8859-15 ou **Latin 9** est une légère modification du Latin 1 qui ajoute le symbole monétaire de l'euro, ainsi que quelques lettres accentuées qui manquaient pour le français et le finnois. Il est destiné aux mêmes langues que le Latin-1.

- Le jeu **ISO-8859-1** (remarquez le tiret supplémentaire) a été validé par l'IANA (Internet Assigned Numbers Authority), pour une utilisation sur Internet, c'est un sur-ensemble de l'ISO/CEI 8859-1.Ce jeu attribue des caractères de contrôle aux valeurs 00 à 1F, 7F, et 80 à 9F. Pour compliquer un peu les choses, le nom du jeu a souvent de nombreux synonymes et équivalents (exemple : ISO-8859-1= ISO_8859-1:1987, ISO_8859-1, ISO-8859-1, iso-ir-100, csISOLatin1, latin1, IBM819, CP819).

**Windows-1252** ou **CP1252** est un jeu de caractères disponible sur Microsoft Windows aux États-Unis, et dans certains pays de l'Union européenne.

Windows-1252 est une extension de l'ISO 8859-1 : il en diffère par l'utilisation de caractères imprimables, plutôt que des caractères de contrôle, dans la plage 80-9F. Windows appelle ce jeu l'ANSI, mais il peut y avoir un autre nom, comme par exemple CP1252.

**MacRoman** ou Mac OS ROMAN est un jeu utilisé sur les ordinateurs Mac qui diffère légèrement du jeu Latin.

Pour mémoire il existait le jeu **OEM** sur les premiers PC.

**En codage 8 bits, pour que les caractères apparaissent correctement, il faut utiliser la bonne page de code (CharSet). Il n'y a qu'une page de code par texte.**

## XX-A-3 - Codage sur 16 bits ou plus : Unicode

On a ensuite créé un code de 16 bits ou plus pour pouvoir coder tous les caractères du monde.

**L'UCS** "jeu de caractères codés universel" (Universal Coded Character Set, souvent abrégé en UCS).

Existe depuis l981.

Il y a :

- l'**UCS-2** qui ne contient que 65535 codes :PMB ou 'Plan Multilingue de Base' codé sur 2 octets contenant les 'principaux' caractères.

- l'**UCS-4** codé sur 4 octets contenant tous les caractères.

- **L'Unicode** ou 'Unicode Standard' depuis 1991 crée par le Consortium Unicode qui collabore avec l'Iso.

Il a été normalisé sous le nom d'ISO/CEI 10646-1:1993 puis ISO/CEI 10646-1:2000.

Contient les mêmes caractères que l'UCS, mais avec quelques règles en plus.

USC-4 = Unicode en simplifiant !!

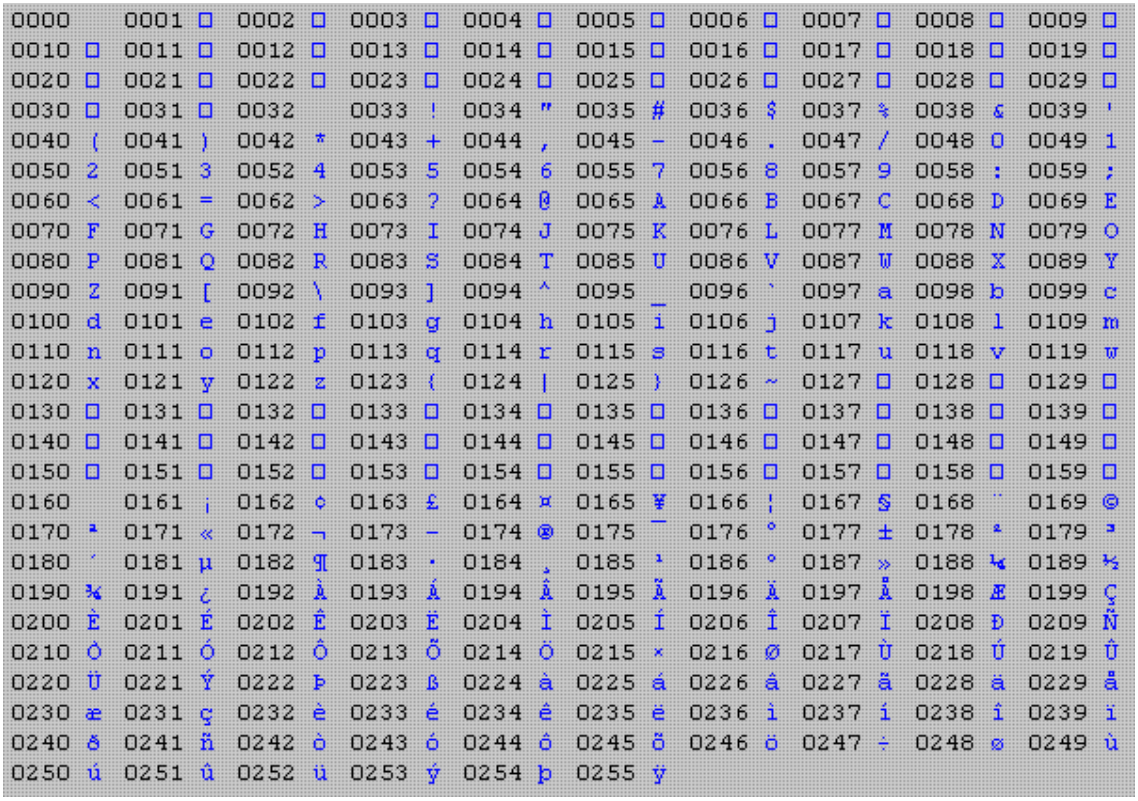
L'Unicode dans sa version la plus récente (4.1.0) contient 245 000 codes différents, représentant 245 000 caractères, symboles, etc.

Il contient tous les caractères d'usage courant dans les langues principales du monde. Il permet de mettre plusieurs langues dans une seule page, ce que ne permet pas le codage 8 bits.

Il a été normalisé sous le nom d'ISO/CEI 10646.

Les premiers 128 codes (0-127) Unicode correspondent aux lettres et aux symboles du clavier américain standard. Ce sont les mêmes que ceux définis par le jeu de caractères ASCII (ancien codage sur un octet). Les 128 codes suivants (128-255) représentent les caractères spéciaux, tels que les lettres de l'alphabet latin, les accents, les symboles monétaires et les fractions. Les codes restants sont utilisés pour des symboles, y compris les caractères textuels mondiaux, les signes diacritiques, ainsi que les symboles mathématiques et techniques.

Voici les 255 premiers




Le petit carré indique un caractère non imprimable (non affichable), certains caractères sont des caractères de contrôle comme le numéro 9 qui correspondant à tabulation, le numéro 13 qui correspond au retour à la ligne…

La valeur d'un caractère Unicode est appelée **point de code**. Ainsi le caractère 'f' a un point de code égal à 102.

L'Unicode regroupe ainsi la quasi-totalité des alphabets existants (arabe, arménien, cyrillique, grec, hébreu, latin…) et est compatible avec le code ASCII.

Encoding

**Pour écrire de l'Unicode dans un fichier (web, html, xml…), cela serait trop simple de mettre 2 ou 3 octets pour chaque caractère. Aussi pour des problèmes de compatibilité entre plateformes les codes Unicode sont transformés en UTF.**

**UTF** = UCS Transformation Format. (Format de transformation d'Unicode.)

Il existe plusieurs méthodes de transformation l'UTF-8, l'UTF-16, l'UTF-32 qui sont des manières d'écrire de l'Unicode dans un fichier.

**UTF-8**

Taille variable : 1 à 4 octets, plus le nombre est grand, plus il y a d'octets.

Les caractères de numéro 0 à 127 sont codés sur un octet dont le bit de poids fort est toujours nul.

0xxxxxxx 1 octet codant 7 bits maximum

'A'= 65 en Unicode = 01000001 en UTF-8 soit 1 octet.

On remarque que dans ce cas ASCII= UTF-8

Les caractères de numéro supérieur à 127 sont codés sur plusieurs octets. Dans ce cas, les bits de poids fort du premier octet forment une suite de 1 de longueur égale au nombre d'octets utilisés pour coder le caractère, les octets suivants ayant 10 comme bits de poids fort.

```
110xxxxx 10xxxxxx 2 octets codant 8 à 11 bits
1110xxxx 10xxxxxx 10xxxxxx 3 octets codant 12 à 16 bits
11110xxx 10xxxxxx 10xxxxxx 10xxxxxx 4 octets codant 17 à 21 bits
```

Les autres bits (notés 'x' ) codent le nombre.

Exemple

€= 8364= 11100010 10000010 10101100

Il est utilisé sur Unix et le WEB.

il est économique en termes d'octet, car il permet de stocker la majorité des caractères sur un seul octet. Pour les caractères dont le point de code a une forte valeur valeurs, l'UTF-8 se chargera de déterminer s'il est nécessaire d'utiliser 1, 2, 3 ou 4 octets pour représenter sa valeur.

**UTF-16**

Taille fixe : 2 ou 4 octets

Il existe l'UTF-16LE et l'UTF-16BE

ils ont la même taille.(Accès rapide au caractère de position x.)

L'UTF-16 est utile lorsque la place en mémoire n'est pas un facteur très important. Il facilite la manipulation de textes, car le processeur sait qu'il doit analyser les deux premiers octets dans la plupart des cas (parfois 4).

Il est utilisé en Java et dans les API Windows pour la manipulation des chaines.

**UTF-32**

Taille fixe : 4 octets.

Mais peu économique en mémoire, il gaspille des octets.

Tous les caractères de la terre y sont, ils ont la même taille. (Accès rapide au caractère de position x.)

Gourmand en mémoire, peu utilisé.

## XX-A-4 - Représentation graphique des caractères : Glyphe, Font, Police

Un glyphe est une représentation graphique (un dessin) d'un signe typographique, autrement dit d'un caractère (glyphe de caractère) ou d'un accent (glyphe d'accent).

Une fonte de caractères est un ensemble de glyphes, c'est-à-dire de représentations visuelles de caractères d'une même famille, de même style, corps et graisse.

La place d'un glyphe dans une fonte est déterminée par le codage des caractères dans la fonte. Ainsi, dans une fonte encodée en ASCII le glyphe du caractère "a" se trouvera en 97e position.

Une police regroupe tous les corps et graisses d'une même famille.

Le glyphe présente le dessin, alors que le caractère (son code en informatique) représente le sens.

En informatique on utilise les caractères (leur code) pour stocker du texte en mémoire, les glyphes pour imprimer ou afficher ces caractères.

**Police BitMap, 'True Type', 'Open Type'.**

Avant les années 1990, il y avait des polices au format BitMap (image matricielle). Elles ne sont plus utilisées par VB.Net.

Depuis 1980 existe **True Type**®, un format de police multiplateforme vectorielle, développé par Apple et vendu ensuite à Microsoft. (Concurrent du format Type1 de PostScript d'Adobe.)

On utilise depuis 2001 **OpenType**® qui est un nouveau format de police multiplateforme vectorielle, développé conjointement par Adobe et Microsoft. Adobe propose plusieurs milliers de polices OpenType.

Les deux principaux atouts du format OpenType résident dans sa compatibilité multiplateforme (un seul et même fichier de polices exploitable sur les postes de travail Macintosh et Windows) et sa prise en charge de jeux de caractères et de fonctions de présentation très étendus, qui offrent de meilleures capacités linguistiques et un contrôle typographique évolué.

Une police Open Type est basée sur le numéro de caractères Unicode et peut contenir jusqu'à 65 000 glyphes.

Le format OpenType est une extension du format TrueType SFNT qui gère également les données des polices Adobe® PostScript® et des fonctions typographiques inédites.

Les noms de fichier des polices OpenType contenant des données PostScript possèdent l'extension .otf, tandis que les polices OpenType de type TrueType portent l'extension .ttf.

**Notion de police proportionnelle**

Il existe des **fonts proportionnelles**, comme l'Arial, où les différents caractères n'ont pas la même largeur (le i est plus étroit que le P) c'est plus joli.

Par contre dans les **fonts non proportionnelles**, comme le 'Courier New', tous les caractères ont même largeur. C'est parfois plus pratique quand on veut afficher des lignes qui concordent sur le plan alignement vertical (et qu'on ne veut pas utiliser les tabulations).


## XX-A-5 - Sur le Web

Un document contient le nom du type de caractères utilisés dans le document : le type d'UTF (codage en 16 bits) ou le nom de la table de caractères (une seule) associée à ce document (codage 8 bits).

Par exemple pour un document HTML une balise indique que le document utilise une table des caractères. (Caractères codés sur 1 octet, c'est de l'ASCII étendu dans cet exemple.)

```
<meta http-equiv="content-Type" content="text/html; charset=iso-8859-1" />
```

ou

```
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
```

S'il n'y a pas de charset spécifié, c'est la table de caractères du système d'exploitation qui est utilisée. Si elle ne correspond pas à celle du document l'affichage est incohérent.

Le problème est aussi (si on utilise un codage 8 bits) l'impossibilité de mélanger dans un même document des alphabets qui ne sont pas définis par la même table. Par exemple le français et l'hébreu.

La solution est de passer à Unicode.

**Par exemple pour un mail**

Content-Type: text/plain; charset=ISO-8859-1

ou

Content-Type: text/plain; charset="UTF-8"

**Par exemple pour un fichier XML :**

```
<?xml version="1.0" encoding="utf-8"?>
```

## XX-A-6 - En VB

Les variables string sont stockées sous la forme de séquences de 16 bits (2 octets) non signés dont les valeurs sont comprises entre 0 et 65 535. Chaque nombre de 16 bis représente le point de code (son numéro et non l'UTF) du caractère Unicode.

Les variables Char sont stockées sous la forme d'un mot de 16 bits (2 octets).

```
Dim monUnicode As Short = Convert.ToInt16 ("B"c) ' le code Unicode de B est 66.
```

En Net, il existe des fonctions d'encodage et de décodage à bas niveau sur les points de code (byte) et les caractères (char), on peut ainsi transformer une chaine de Latin-1 en UTF-8, par exemple.

Fonctions d'encodage :

GetBytes() convertit une chaine ou un caractère en un tableau de bytes dans le charset choisi.

Comment obtenir un tableau de Bytes contenant le code ASCII d'un texte ?

```
Dim  ASCII As System.Text.Encoding = System.Text.Encoding.ASCII

Dim MyString As String= "mon texte"

Dim  MesByte() As Byte = ASCII.GetBytes( MyString )
```

(les caractères non gérés en ASCII sont transformés en "?", portant la valeur numérique 63.)

Fonctions de décodage :

GetChars() convertit un tableau de bytes en un tableau de char.

## XX-B - Nommage des objets visuels, variables et objets

### Nommage des variables et Objets

**Notation hongroise**

La notation hongroise est une convention de nommage qui met en avant, par l'utilisation d'un préfixe, des informations sur l'objet nommé.

La qualification de "hongroise" vient du pays d'origine de Simonyi, le créateur de ce nommage qui a travaillé chez Xerox puis Microsoft.

Il y a 2 sortes de notation hongroise.

*Notation Apps*

On préfixe le nom des variables de manière à indiquer leur utilisation.

rwPosition : est une variable représentant la position d'une ligne (rw=row en anglais)

*Notation Systems*

Cette notation consiste à faire précéder le nom de la variable d'un préfixe reflétant son type. Ce préfixe est toujours écrit en minuscules puis la première lettre suivante en majuscule.

Par exemple, la variable booléenne 'ouvert' est préfixée par un b pour indiquer un booléen : 'bOuvert'.

On peut préfixer avec le type et (ou) la portée.

Type :

i Integer (entier) ;

n Short int (entier court) ;

l Long (entier long) ;

f float= Single (nombre a virgule flottante) ;

d Double (float double) ;

c Char (caractère) ;

by Byte (caractère non signe) ;

b Boolean (booleen true/false) ;

s String (chaine de caractères) ;

h Handle ;

file File (fichier).

On peut aussi utiliser k pour les constantes.

Et non utilisé en VB :

v void ;

w word (mot = double octet) ;

dw double word (double mot) ;

sz zero-terminated string (chaine de caractères terminée par un char zéro) ;

str string object (objet String) ;

pt point ;

rgb rgb triplet ;

Portée :

g : variable globale à une application ;

s : variable locale à un module ;

p : variable passée en paramètre d'une fonction ou d'une méthode de classe ;

Si une variable est un tableau, on ajoute le préfixe "a".

**Le nom de la variable commence par une majuscule.**

Exemple

Préfixe avec le type : variable nommée Ouvert de type boolean : bOuvert.

Préfixe avec la portée, variable nommée Nom de portée globale : gNom ou g_Nom.

Variable nommée Position qui est une variable globale et une Integer : g_iPosition

On voit qu'on peut utiliser ou non le caractère""_".

Cette notation hongroise a ses partisans et ses adversaires (nom trop complexe).

## Choisir les noms de procédures et de variables avec soins

**On concatène plusieurs mots pour former un nom de fonction, de variable de Classe…**

Il y a 3 manières d'utiliser les lettres capitales dans ces mots (on appelle cela la capitalisation !)

**- Pascal Case : la première lettre de chaque mot est en majuscule :**

CalculTotal()

**- Camel Case : la première lettre de chaque mot est en majuscule, sauf la première :**

iNombrePatient

**- UpperCase : toutes les lettres sont en majuscules :**

MACONSTANTE

De plus le nom doit être explicite.

Microsoft propose quelques règles :

**Sub , Fonctions**

Utilisez la 'case Pascal' pour les noms de routine (la première lettre de chaque mot est une majuscule).

Exemple : CalculTotal()

Évitez d'employer des noms difficiles pouvant être interprétés de manière subjective, notamment Analyse() pour une routine par exemple.

Utilisez les verbe/nom pour une routine : CalculTotal().

**Variables**

Pour les noms de variables, utilisez la 'case Camel' selon laquelle la première lettre des mots est une majuscule, sauf pour le premier mot.

Exemple: iNombrePatient

Noter ici que la première lettre indique le type de la variable (Integer), elle peut aussi indiquer la portée (gTotal pour une variable globale).

Évitez d'employer des noms difficiles pouvant être interprétés de manière subjective, 'YYB8' ou 'flag' pour une variable.

Ajoutez des méthodes de calcul (Min, Max, Total) à la fin d'un nom de variable, si nécessaire.

Les noms de variable booléenne doivent contenir Is qui implique les valeurs True/False, par exemple fileIsFound.

Évitez d'utiliser des termes tels que Flag lorsque vous nommez des variables d'état, qui différent des variables booléennes, car elles acceptent plus de deux valeurs. Plutôt que documentFlag, utilisez un nom plus descriptif tel que documentFormatType.

Même pour une variable à courte durée de vie, utilisez un nom significatif.

Utilisez des noms de variable d'une seule lettre, par exemple i ou j, pour les index de petite boucle uniquement.

**Paramètre**

Pour les noms de paramètres, utilisez la 'case Camel' selon laquelle la première lettre des mots est une majuscule, sauf pour le premier mot.

Exemple : typeName

**Constantes**

Utiliser les majuscules pour les constantes : MYCONSTANTE

N'utilisez pas des nombres ou des chaines littérales telles que For i = 1 To 7. Utilisez plutôt des constantes par exemple For i = 1 To DAYSINWEEK, pour simplifier la maintenance et la compréhension.

**Objet, Classe**

Pour les noms de Classe, utiliser le case Pascal:

Exemple Class MaClasse

Idem pour les événements, espace de noms, méthodes :

Exemple : System.Drawing, ValueChange…

Dans les objets, il ne faut pas inclure des noms de classe dans les noms de propriétés Patient.PatientNom est inutile, utiliser plutôt Patient.Nom.

Les interfaces commencent par un I

Exemple : IDisposable

Pour les variables privées ou protégées d'une classe utilisez le case Camel :

Exemple : lastValue (variable protégée)

En plus pour les variables privées d'une classe mettre un "_" avant :

Exemple : _privateField

Pour une variable Public d'une classe utiliser le 'case Pascale' :

Exemple TotalAge

**Tables**

Pour les tables, utilisez le singulier. Par exemple, utilisez table 'Patient' plutôt que 'Patients'.

N'incorporez pas le type de données dans le nom d'une colonne.

**Divers**

Minimisez l'utilisation d'abréviations.

Lorsque vous nommez des fonctions, insérez une description de la valeur retournée, notamment GetCurrentWindowDirectory().

Évitez de réutiliser des noms identiques pour divers éléments.

Évitez l'utilisation d'homonymes et des mots qui entrainent souvent des fautes d'orthographe.

Évitez d'utiliser des signes typographiques pour identifier des types de données, notamment $ pour les chaines ou % pour les entiers.

**Un nom doit indiquer la signification plutôt que la méthode.**

## Nommage des objets visuels

Il est conseillé de commencer le nom d'un objet visuel par un mot évoquant sa nature.

Microsoft conseille :

btn pour les Boutons

lst pour les ListBox

chk pour les CheckBox

cbo pour les combos

dlg pour les DialogBox

frm pour les Form

lbl pour les labels

txt pour les Textbox

tb pour les Toolsbar

rb pour les radiobutton

mm pour les menus

tmr pour les timers

Exemple :

btnOk par exemple pour un bouton sur lequel il y a 'OK'.

## XX-C - Couleurs disponibles dans VB

Le plus simple est, pour modifier la couleur d'un objet par du code, d'utiliser les constantes VB qui contiennent le code d'une couleur toute faite (en RGB sans composante Alpha) :

```
Color.Back,

Color.Fuchsia

Color.Chocolate

Color.Red…
```

Voici toutes les couleurs à votre disposition :


Elles font partie de System.Drawing.

## XX-D - Format de fichier texte : le RTF

**Qu'est-ce que RTF ?**

Un texte peut être enregistré en brut (en ASCII sans enrichissement en '.txt' par exemple) en RTF ('.Rtf') , dans un format propriétaire : format Word (.doc)…

RTF= Rich Text Format = Format de Texte Enrichi

Le RTF est un format de fichier texte assez universel. Il permet de mettre dans un fichier du texte, mais aussi d'indiquer l'enrichissement de ce texte : texte en gras, italique, souligné, en couleur, en Arial…

Les fichiers Rtf ont l'extension '.Rtf'. Ils sont lisibles dans la plupart des traitements de texte (Word, Open Office, NotePad…).

Le format du texte que l'on peut mettre dans une RichTextBox est le format RTF.

**Les bases du codage RTF**

Le texte doit débuter par '{' et se terminer par '}'.

Il peut aussi débuter par "{\rtf1\ansi" et se terminer par '}'.

Cela indique que le texte est en rtf et le codage des caractères est en ansi.

Ensuite les enrichissements s'effectuent par des balises qui indiquent le début et la fin de l'attribut.

Une balise commence par le caractère '\' .

Toujours mettre un espace après la balise.

Entre \b et \b0 le texte sera en gras (Bold)

Exemple :

Ajoute le texte "Ce texte est en gras." à un contrôle RichTextBox existant.

RichTextBox1.Rtf = "{\rtf1\ansi Ce texte est en \b gras\b0 .}"

Voici les principaux attributs :

```
\b   \b0      ce qui est entre les 2 balises est en gras
\i    \i0     ce qui est entre les 2 balises est en italique
\par          fin paragraphe (passe à la ligne)
\f            font    \f1 … \f0  font numéro 1 entre les 2 balises
\plain        ramène les caractères par défaut
\tab          caractère de tabulation
\fs           taille de caractère   \fs28 = taille 28
```

Mettre un espace après la balise :

Écrire: \b bonjour \b0 et non \bbonjour \b0

**Mettre un texte en couleurs, utiliser plusieurs polices :**

Mettre la table des couleurs en début de texte :

```
{ \colortbl \red0\green0\blue0;\red255\green0\blue0;\red0\green255\blue0;}
```

Après Colortbl (Color Table) chaque couleur est codée avec les quantités de rouge vert et bleu.

Les couleurs sont repérées par leur ordre : couleur 0 puis 1 puis 2… et séparées par un ';'

Dans notre exemple couleur 0=noir; couleur 1=rouge; couleur 2=vert

Pour changer la couleur dans le texte on utilise \cf puis le numéro de la couleur :

```
"\cf1 toto   \cf0 }" 'toto est affiché en rouge.
```

Pour modifier les polices de caractères, le procédé est similaire avec une Font Table :

```
{\fonttbl
{\fo\froman Symbol;}
{\f1\fswiss Arial;}
}
```

Pour passer en Arial \f1 …\f0

Exemple complet :

```
"{\rtf1\ansi
 { \colortbl
\red0\green0\blue0;
\red255\green0\blue0;
\red0\green255\blue0;}
{\fonttbl
{\fo\froman Symbol;}
{\f1\fswiss Arial;}
}
Ce qui suit est en \f1 \cf1 \i Arial Rouge Italique \f0 \cf0 \i0
}"
```

Cela donne :


N. B. Si vous voulez copier-coller l'exemple pour l'essayer, enlever les sauts à la ligne.

## XX-E - Format XML

Qu'est-ce que le XML ?

XML est l'abréviation de e**X**tensible **M**arkup **L**anguage( langage de balisage extensible).

XML a été mis au point par le XML Working Group sous l'égide du W3C dès 1996. (http://www.w3.org/XML/)

XML est un sous-ensemble, une simplification de SGML (Standard Generalized Markup Language)pour le rendre utilisable sur le web !

**XML décrit le contenu plutôt que la présentation** (contrairement À HTML). Ainsi, XML permet de séparer le contenu de la présentation… ce qui permet par exemple d'afficher un même document sur des applications ou des

périphériques différents sans pour autant nécessiter de créer autant de versions du document que l'on nécessite de représentations.

**Qu'est-ce qu'une balise ?**

'Élément Sémantique de base' des langages de balisage.

Une balise est un 'mot-clé', un élément, comprise entre crochets (< et > ) qui possède un nom et parfois des attributs.

Toutes les balises doivent être ouvertes puis refermées (contrairement à l'HTML ou certaines balises n'ont pas besoin d'être refermées). On retrouvera donc toujours une balise de début et une balise de fin. La balise de fin porte le même nom que la balise de début à l'exception du nom de la balise qui est précédé du signe /.

Exemple :

```
<titre> cours de XML </titre>
```

Si une balise est vide on peut combiner balise de début et de fin :

```
<reponse />
```

Le nom de la balise et le nom des attributs (contenu entre les crochets) doivent être en majuscules ou minuscules. Les noms de balise peuvent comporter des lettres, des chiffres, des tirets, des traits de soulignement, des deux-points ou des points. Le caractère deux-points (:) ne peut être utilisé que dans le cas particulier où il sert à séparer des espaces de noms. Un nom de balise ne doit pas commencer par xml ou XML.

Les caractères < et & ne peuvent pas être utilisés dans le texte, car ils sont utilisés dans le balisage. Si vous devez employer ces caractères, utilisez < à la place de < et < à la place de &.

Sinon utiliser la balise CDATA , dans ce cas on peut mettre les caractères interdits:

```
<![CDATA[ Texte avec des < des &  ]]>
```

**Un attribut** est le nom d'une propriété de la balise souvent associé à une valeur, il est mis dans la balise d'ouverture après le nom de la balise, il est en minuscules ou majuscules, la valeur est entre " ou entre ' :

```
<élément attribut="valeur"/>
```

Exemple :

```
<reponse = "oui" />
```

La balise ne sera pas vue si on affiche le document dans un navigateur.

Il n'y a aucun nom d'élément réservé.(quelques noms d'attributs le sont).

Les noms d'élément sont choisis librement par l'auteur du document.

**Un document XML se compose :**

a- d'un prologue, facultatif, il contient des déclarations :

b- d'un arbre d'éléments (contenu proprement dit du document) ;

c- de commentaires ou d'instructions de traitement facultatifs.

### a- PROLOGUE XML

Les documents doivent commencer par une déclaration indiquant **la version de l'XML**.

```
<?xml version="1.0"?>
```

Ce document respecte la spécification XML 1.0.

On peut ajouter **le type de codage des caractères :**

```
<?xml version="1.0" encoding="UTF-8"?>
```

'Ici les caractères sont codés en UTF-8

```
Les caractères >, " , et ' peuvent également être remplacés par &gt; , &quot; et &apos;
 respectivement
```

On peut aussi ajouter **une déclaration de type de document (DTD)**.

Si le document doit se conformer à une structure particulière, on doit l'indiquer.

Exemple : le document est de type rapport médical et sa structure doit se conformer à la structure type définie dans la ressource "rapportmedical.dtd" :

```
<!DOCTYPE rapportmedical SYSTEM "rapportmedical.dtd" [  déclarations ]>
```

### b- Structure en arbre

Il peut y avoir plusieurs balises intriquées, mais attention à l'ordre des balises de fin qui doivent être dans l'ordre inverse (première balise ouverte, dernière fermée).

```
<article>
   <para>
        …

     </para>
 </article>
```

Tout élément enfant est inclus dans l'élément parent.

Il doit y avoir un **élément racine** qui englobe tous les autres :

```
<livre>

  <chapitre>Item 1</chapitre>

  <chapitre>Item 2</chapitre>

  <chapitre>Item 3</chapitre>

</livre>
```

Ici l'élément racine est livre, peu importe le nom, pourvu qu'il englobe tout.

On remarque qu'il y a plusieurs balises chapitre.

### c- Commentaires et instructions de traitement

Il existe des instructions de traitement qui contiennent des instructions destinées aux applications.

```
<?perl lower-to-upper-case  ?>
```

On peut mettre des commentaires :

```
<!-- Commentaire -->
```

### Dissocier la forme et le fond

Les balises XML indiqueront le contenu, **la signification (la sémantique)** des éléments de la page et non pas comment les afficher.

Le HTML lui comportait des informations sur la forme, la manière dont était affichée la page, l'apparence du site.

Soit un document :

- soit un **document HTML**, les balises définissent pour ce document ce qui est en gras(balise b), en italique (balise i)…

- soit un **document XML**, les balises définissent le contenu. Ainsi une œuvre est contenue entre les balises 'œuvre', de même les balises 'date' et 'auteur' permettent de dresser une liste des auteurs et des dates. La présentation sera, si nécessaire, indiquée dans une feuille de style CSS.

Ce document XML est enregistré dans un fichier .xml

Vous pouvez avoir des noms de balise propre à votre document (en HTML, les noms sont prédéfinis).

### DTD

On peut aussi ajouter une déclaration de type de document (DTD).

Si le document doit se conformer à une structure particulière, on doit l'indiquer.

Exemple : le document est de type rapport médical et sa structure doit se conformer à la structure type définie dans la ressource "rapportmedical.dtd"

```
<!DOCTYPE rapportmedical SYSTEM "rapportmedical.dtd" [  déclarations ]>
```

### Feuille de style

La feuille de style, ou CSS est l'abréviation de Cascading Style Sheets. Cette feuille nous sert uniquement à présenter la page web. C'est en CSS que l'on dira : "Mes titres sont en vert et sont soulignés, le nom doit être affiché en gras…

La feuille de style est dans un fichier .css

### XML dans VB

À partir de VB 2008, on peut créer directement du XML dans le code.

On peut créer un élément XML :

```
Dim contact1 As XElement = _
```

```
    <contact>
      <name>Patrick Dupont</name>
      <phone type="home">206-555-0144</phone>
      <phone type="work">425-555-0145</phone>
    </contact>
```

On peut créer un document XML :

```
Dim contactDoc As XDocument = _
    <?xml version="1.0"?>
    <contact>
      <name>Patrick Dupont</name>
      <phone type="home">206-555-0144</phone>
      <phone type="work">425-555-0145</phone>
    </contact>
```

On peut mettre dans le XML des expressions qui sont évaluées au cours de l'exécution, ces expressions sont de la forme :

```
<%= expression %>.
```

Exemple: Inclure dans l'élément XML un nombre et une date :

```
Dim MyNumber As String = "12345"

Dim MyDate As String = "3/5/2006"

Dim livre As XElement = _

<livre category="fiction" isbn=<%= MyNumber %>> <TheDate><%= MyDate %></TheDate> </livre>
```

On peut aussi créer une String et la transformer en XML :

```
Dim MyString = "<Cours id=""1"">" & vbCrLf & _

 " <Author>Philippe</Author>" & vbCrLf & _

 " <Title>Cours VB</Title>" & vbCrLf & _

" <Price>0</Price>" & vbCrLf & _

 "</Cours>"

Dim xmlElem = XElement.Parse(MyString)
```

On peut charger un document à partir d'un fichier :

```
Dim books = XDocument.Load("books.xml")
```

ou à partir d'un Stream :

```
Dim reader = System.Xml.XmlReader.Create("books.xml")

reader.MoveToContent()

Dim inputXml = XDocument.ReadFrom(reader)
```

# XX-F - Migration VB6=>VB.NET


**Cela concerne surtout ceux qui passent de VB6 à VB.NET, pour les autres c'est une révision.**

Les petits nouveaux qui ne connaissaient pas VB6 (précédente version de VB) ne doivent pas lire ce qui est en vert.

## XX-F-1 - Les objets

En VB.NET **tout est objet :** les fenêtres, les contrôles, les variables…

Set et Let ne sont plus pris en charge.

Les objets peuvent être assignés par une simple opération d'assignation :

Object1 = Object2

Pour définir une propriété par défaut d'un objet, vous devez désormais référencer explicitement la propriété. Exemple :

Object1.Text = Object2.Text

Vous pouvez définir vous-même un nouveau type d'objet, une **Classe.**

Puis instancier des objets à partir de cette Classe.

VB.NETpermet une **vraie programmation-objet :** héritage, polymorphisme, surcharge, Interface…

### XX-F-1-a - Les Classes du Framework

Il existe toujours **les mots-clés de Visual Basic** (Len, Trim, Rnd…), mais en plus le Framework met à disposition **une multitude de Classes qui possèdent des méthodes** permettant de faire une multitude de choses sans programmer.

Exemple

La Classe Array (les tableaux) possède une méthode Sort qui trie les tableaux :

```
Dim T() As Integer
    T(0)=78
    …
    Array.Sort(T)
```

## XX-F-1-b - Les formulaires ou fenêtres

On se rend compte que quand on dessine une fenêtre Form2 par exemple, VB crée une nouvelle classe **'Class Form2' :**

```
Public Class Form2
Inherits Systeme.Windows.Forms.Form

End Class
```

Elle hérite de **System.Windows.Forms.Form :** on le voit bien dans le code.

Pour utiliser cette fenêtre, il faut créer une instance de cette fenêtre à l'aide de la Classe.

On tape Dim f As New Form2(), on crée une instance de la Class Form2.

Enfin la fenêtre sera ouverte grâce à la méthode ShowDialog ou Show.

Comme pour un Objet en général, la fenêtre créée sera visible dans sa portée. Si une fenêtre est instanciée dans une procédure, l'objet fenêtre f ne sera visible que dans cette procédure.

**Comment passer de VB6 à VB.net ?**

- Avant en **VB6**, on avait :

```
Form2.Load

Form2.Show
```

- **Avec le programme de conversion** VB6=>VB.Net on a :

```
Form2.DefInstance.Show()
```

car il ajoute une routine qui créée automatiquement une instance de form2 :

#Region "Prise en charge de la mise à niveau"

```
Private Shared m_vb6FormDefInstance As form2
Private Shared m_InitializingDefInstance As Boolean
Public Shared Property DefInstance() As form2
    Get
        If m_vb6FormDefInstance Is Nothing OrElse m_vb6FormDefInstance.IsDisposed Then
            m_InitializingDefInstance = True
            m_vb6FormDefInstance = New form2()
            m_InitializingDefInstance = False
        End If
            DefInstance = m_vb6FormDefInstance
    End Get
    Set
            m_vb6FormDefInstance = Value
    End Set
End Property
#End Region
```

- En fait il faut mieux avec **VB.net** écrire :

```
Dim F As New Form2

F.Show
```

On remarque que Load n'existe plus, par contre, le Dim crée le formulaire sans l'afficher, c'est à peu près équivalent…

Les Forms ont **2 contrôles menu :**

les MainMenu ;

les ContextMenu.

Visual Basic .NET ne prend pas en charge la méthode Form.PrintForm

## XX-F-1-c - Les Contrôles

La plupart des objets ne possèdent **plus de propriétés par défaut.**

En VB6 :

```
Dim str As String = TextBox1
```

Maintenant il faut écrire :

```
Dim str As String = TextBox1.Text
```

Visual Basic .NET **ne prend pas en charge le contrôle conteneur OLE.**

Il n'existe **pas de contrôle carré rectangulaire ligne.** On peut les remplacer sous la forme d'étiquettes (Label), tandis que **les ovales et les cercles qui n'existent pas** non plus ne peuvent pas être mis à niveau.

Visual Basic .NET est doté d'un nouvel ensemble de commandes graphiques faisant partie de GDI+. Circle, CLS, PSet, Line et Point n'existent plus. Étant donné que le nouveau modèle objet est assez différent de celui de Visual Basic 6.0, il faut tout réécrire.

Visual Basic .NET **ne prend pas en charge l'échange dynamique de données (DDE**, Dynamic Data Exchange).

Bien que Visual Basic .NET prenne en charge la fonctionnalité de glisser-déplacer, le modèle objet est différent de celui de Visual Basic 6.0. Il faut tout réécrire.

Le .NET Framework est doté d'un objet **Clipboard** amélioré (**System.Windows.Forms.Clipboard**) qui offre plus de fonctionnalités et prend en charge un plus grand nombre de formats de presse-papiers que l'objet **Clipboard** de Visual Basic 6.0. Il faut tout réécrire.

Les groupes de contrôle n'existent plus. Il y a des moyens de les remplacer.

## XX-F-1-d - Les Variables

**Option Explicit** étant activé par défaut, toutes les variables doivent être déclarées avant d'être utilisées.

Le type de données Variant n'existe plus. Celui-ci a été remplacé par le type **Object.**

Le type de données **Integer** est désormais de 32 bits ; le type de données **Long** est de 64 bits.

On peut utiliser les **types booléens** qui ne peuvent prendre que 2 valeurs : **True et False** (pas 0 et -1).

En VB.NET Option Strict étant activé, il faut convertir explicitement une donnée d'un type vers un autre si c'est nécessaire.

Response.Write("Le total est" & CStr(total))

On attend des String, la variable total qui est numérique est donc transformée en String par CStr.

Les variables créées dans la même instruction **Dim** seront du même type. Par exemple, dans VB.NET, l'instruction Dim i, j, k As Integer crée chacun des trois objets (i, j, et k) comme **Integer.** Les versions précédentes de VB créaient i et j comme Variants et k comme Integer (c'était nul !!).

Il existe un **type Char** qui contient un seul caractère.

Le type Currency est remplacé par le type **Decimal.**

Les String de longueur fixe n'existent plus. Il y a quelques ficelles pour contourner cela, mais bonjour la simplicité !!

Les String et Char sont en **Unicode (chaque caractère est codé sur 2 octets).**

Une variable peut avoir une portée locale, publique ou, et c'est nouveau, **une portée au niveau d'un bloc :**

```
For i=0 to 100
    Dim Str As String 'Str est visible uniquement entre For et Next
    ….
    next i
```

## XX-F-1-e - Les Tableaux

**Le premier élément d'un tableau a l'indice 0** obligatoirement, plus d'Option Base.

On peut initialiser un tableau en même temps qu'on le déclare :

Dim Tableau(3) As Integer ={7,2,3,5}

À noter que ce tableau contient 4 éléments d'index 0, 1, 2, 3.

Dim S(4 to 15) n'est plus accepté.

Dim est utilisé pour la déclaration initiale, **Redim pour le redimensionnement** uniquement.

Les tableaux font partie de **la Classe Array**, ce qui autorise l'utilisation de méthodes bien pratiques : Sort par exemple trie automatiquement le tableau.

## XX-F-1-f - Les Collections

Elles sont omniprésentes. C'est fondamental de bien comprendre leur fonctionnement : ce sont des listes ayant un nombre d'éléments non défini, on peut y ajouter des éléments, en retirer, il y a des méthodes pour trier, rechercher…

Cela peut être :

des listes d'objets : ArrayList ;

des listes de booléens : BitArray ;

des listes Clé-Valeur :HashTable ;

des Queue ;

des Piles : Stack.

La notion de collection est généralisée et utilisée dans beaucoup d'objets : une ListBox possède une collection d'Items (les éléments de la listBox).

## XX-F-1-g - Les Structures

Elles remplacent les "Types définis par l'utilisateur".

```
Structure MaStructure
    Public i As Integer
    Public c As String
End Structure
```

## XX-F-1-h - Les Fonctions et Sub

Les parenthèses sont désormais requises autour de la liste de paramètres dans tous les appels de méthodes y compris pour les méthodes qui ne prennent pas de paramètres. Exemple :

```
If Flag = False Then
       AfficheMessage()
End If
```

Par défaut, les arguments sont passés **par valeur**, et non pas référence. Si vous voulez passer des arguments par référence, vous devez utiliser le mot-clé **ByRef** devant l'argument comme dans l'exemple suivant :

Call MaFunction(argbyvalue, ByRef argbyref)

Il peut y avoir des **paramètres optionnels.**

**Return** est un nouveau mot-clé qui permet à une fonction de retourner une valeur.

```
Private Function MaFonction (Byval Chaine As String)
    Return Chaine.ToLower()
End Function
```

## XX-F-1-i - Dans le code

Simplification d'écriture :

A +=2 est équivalent à A=A+2

Nouveau type de boucle

While condition

End While

Boucle tant que condition est vraie.

Wend n'existe plus.

Le **Multithreading** est possible.

## XX-F-1-j - Gestion des erreurs

La gestion des erreurs est structurée.

Elle utilise

Try

Code à tester

Catch

interception de l'erreur

Finally

suite

End Try

**On error goto** reste utilisable.

## XX-F-1-k - Les graphiques

En GDI (VB6) on utilisait les handles(HDC) pour spécifier un image.

En GDI+ (VB.Net), on travaille sur les Graphics et leurs méthodes. Graphics.DrawLine…

L'unité de mesure est le pixel. (Plus de Twips.)

## XX-F-1-l - Les bases de données

Visual Basic .NET contient une version améliorée des objets de données actifs (ADO, Active Data Objects) appelée **ADO.NET.**

DAO, RDO et ADO peuvent toujours être utilisés dans du code Visual Basic .NET, avec toutefois quelques petites modifications. Toutefois, Visual Basic .NET ne prend pas en charge la liaison de données DAO et RDO aux contrôles ou contrôles de données ni la connexion utilisateur RDO.

## XX-F-1-m - Les Classes

La syntaxe des propriétés de classe a changé et ne contient plus **Property Let, Property Get,** et **Property Set.** La nouvelle syntaxe des propriétés est analogue à celle de C#.

```
Public Property ThisProperty As String
    Get
          ThisProperty = InternalValue
    End Get
    Set
```

```
        InternalValue = value
    End Set
End Property
```

Les classes sont totalement objet et acceptent le polymorphisme, la surcharge, l'héritage…

## XX-F-1-n - GOSUB et ON GOSUB n'existent plus

Il faut remplacer une routine qui était appelée par gosub par une sub ou une **fonction.**

Remplacer

```
Sub Mon Programme
    …
        Gosub Routine
    …
End SuB
```

```
Routine:
        Code de la routine
Return
```

Par

```
Sub Mon Programme
    …
        Call Routine()
    …
        End Sub

Sub Routine()

    code de la routine

End Sub
```

Il faudra simplement faire attention aux variables, les variables locales à Mon Programme ne seront pas accessibles dans la routine.

Pour **On Gosub**, il faut remplacer par un **SelectCase.**

## XX-F-1-o - Les Timers

S'agissant du contrôle Timer, le fait d'affecter à la propriété **Interval** la valeur 0 ne désactive pas la minuterie ; l'intervalle est réinitialisé à 1. Dans vos projets Visual Basic 6.0, vous devez affecter à la propriété **Enabled** la valeur **False** plutôt que d'affecter à **Interval** la valeur 0.

## XX-F-1-p - Outil de conversion VB6 vers VB.NET

Il existe un **outil de conversion** (Menu Fichier, Ouvrir, Convertir, Assistant de mise à niveau de VB.NET) pour convertir un source VB6 en VB.NET.

Le problème est qu'il donne un code, à mon avis, inutilisable avec :

- conversion des instructions VB6=>VB.NET quand il le peut ;

- conversion en utilisant une **classe de compatibilité** contenant des instructions spécifiques à VB6 (qui ne sont PAS du VB.NET) produisant un code hybride. Cette classe de compatibilité disparaitra probablement dans les prochaines versions ;
- conversion en utilisant des ficelles avec ajout de plein de code : voir l'exemple des Forms au-dessus ou l'outil de conversion crée une complexe fonction nommée 'DefInstance' permettant de créer des formulaires ;
- des instructions impossibles à convertir automatiquement et qui seront à réécrire à la main.

Il faut convertir les programmes VB6 avec l'outil de conversion **pour voir ce que cela donne:** c'est instructif de voir le nouveau code.

Mais il faut réécrire totalement une bonne partie du code : l'appel des fenêtres en particulier.

**Il faut rapidement ne pas utiliser du tout la classe de compatibilité VB 6 , éviter les instructions héritées de VB6, privilégier l'usage des classes du FrameWork.**