Visual Basic.NET

Par Hankerspace et Thunderseb

www.siteduzero.com

Dernière mise à jour le 18/09/2011

Sommaire

Visual Basic.NET

Vous trouvez le C et le C++ trop compliqués mais aimeriez concevoir des programmes fonctionnels, ergonomiques et facilement accessibles ?

Vous avez trouvé le langage qu'il vous faut : Visual Basic.

Il s'agit d'un langage excessivement simple permettant de :

- créer des programmes très simplement ;
- élaborer des interfaces graphiques sous Windows ;
- concevoir des formulaires ;
- gérer le temps ;
- écrire dans les fichiers ;
- accéder à une base de données ;
- et, par la suite, construire des sites web (oui, vous avez bien entendu !).

Ce tutoriel va vous initier aux bases du Visual Basic, ce qui est tout de même normal pour des Zéros.
Aucun prérequis n'est demandé : il n'est pas nécessaire de connaître un seul langage ; tout vous sera expliqué.

Voici quelques exemples de programmes réalisables en VB .NET et qui seront abordés dans le tutoriel.

Tout en essayant de rester le plus clair et concis possible, je vais vous expliquer, dans les grandes lignes, les principales fonctionnalités de base du langage, ainsi que la façon dont vous servir des outils que vous utiliserez par la suite pour réaliser des programmes. Ensuite, ce sera à vous de voler de vos propres ailes.

Informations sur le tutoriel

Auteur :

- Hankerspace

Difficulté :
Temps d'étude estimé : 15 jours
Licence :

Partie 1 : La théorie et les bases

Partie consacrée à l'apprentissage rapide et précis des concepts de base qui vont nous apprendre à programmer en BASIC.
Le basic n'est en fait pas réellement un langage, mais plutôt un style de programmation très simple et assez clair, sur lequel sont basés certains langages.

Nous allons ici parler de la partie "script" du langage créé par Microsoft. C'est la base de ce qu'il y a à connaître pour la suite 

Historique et Visual Basic Express 2010

Pour commencer, je vais vous présenter l'historique du Visual Basic. Ensuite, nous verrons ensemble comment télécharger et installer les outils nécessaires afin de poursuivre la lecture de ce tutoriel sans embûches.

Historique, naissance de Visual Basic

D'où vient le Visual Basic ?

Nous allons donc débuter par un petit morceau d'histoire, car il est toujours intéressant de connaître le pourquoi de l'invention d'un langage (il doit bien y avoir une raison ; sinon, nous serions encore tous à l'assembleur).

J'ai récupéré l'essentiel de Wikipédia et vous le résume brièvement.

Le BASIC

BASIC est un acronyme pour *Beginner's All-purpose Symbolic Instruction Code*. Le BASIC a été conçu à la base en 1963 par John George Kemeny et Thomas Eugene Kurtz au « Dartmouth College » pour permettre aux étudiants qui ne travaillaient pas dans des filières scientifiques d'utiliser les ordinateurs. En effet, à l'époque, l'utilisation des ordinateurs nécessitait l'emploi d'un langage de programmation assembleur dédié, ce dont seuls les spécialistes étaient capables.

Les huit principes de conception du BASIC étaient :

- être facile d'utilisation pour les débutants (*Beginner*) ;
- être un langage généraliste (*All-purpose*) ;
- autoriser l'ajout de fonctionnalités pour les experts (tout en gardant le langage simple pour les débutants) ;
- être interactif ;
- fournir des messages d'erreur clairs et conviviaux ;
- avoir un délai de réaction faible pour les petits programmes ;
- ne pas nécessiter la compréhension du matériel de l'ordinateur ;
- isoler l'utilisateur du système d'exploitation.

Tout ce qu'il nous faut, donc.

Le Visual Basic

De ce langage — le BASIC — est né le Visual Basic. Le VB est directement dérivé du BASIC et permet le développement rapide d'applications, la création d'interfaces utilisateur graphiques, l'accès aux bases de données, ainsi que la création de contrôles ou d'objets ActiveX.
Je pense qu'avec ces possibilités, on va déjà pouvoir créer de petites choses.
Le traditionnel « *hello world* » en Visual Basic :

Code : Autre

```
Sub Main()
    MsgBox("Hello World !")
End Sub
```

Ce code ouvre une ***MsgBox*** (comme un message d'erreur Windows) dans laquelle est contenu le message « Hello World ! ».

Nous allons rapidement résumer tout ce que Wikipédia nous a appris.

Il faut savoir que le BASIC, ancêtre du Visual Basic, est un langage de **haut niveau**. En programmation, les langages peuvent se trier par niveau : plus le niveau du langage est bas, plus celui-ci est proche du matériel informatique (le C est considéré comme un langage de bas niveau). Un développeur utilisant un langage de bas niveau devra, entre autres, gérer la mémoire qu'il utilise. Il peut même aller jusqu'à spécifier les registres matériels dans lesquels écrire pour faire fonctionner son programme.
Un langage de haut niveau fait abstraction de tout cela ; il le fait en interne, c'est-à-dire que le développeur ne voit pas toutes ces opérations. Après, tout dépend de votre envie et de votre cahier des charges : si vous devez développer une application interagissant directement avec les composants, un langage de bas niveau est requis. En revanche, si vous ne souhaitez faire que du graphisme, des calculs, du fonctionnel, etc., un langage de haut niveau va vous permettre de vous soustraire à beaucoup de

manipulations fastidieuses.

Le Visual Basic est donc un langage de haut niveau. Il a d'emblée intégré les concepts graphique et visuel pour les programmes que l'on concevait avec. Il faut savoir que les premières versions de VB, sorties au début des années 1990, tournaient sous DOS et utilisaient des caractères semblables à des lettres pour simuler une fenêtre.

Code : Console

```
|----------------------------|
|    Ma fenêtre en VB 1.0    |
|----------------------------|
|                            |
|                            |
|                            |
|                            |
|                            |
|                            |
|----------------------------|
```

Ce n'était pas la joie, certes, mais déjà une révolution !

Aujourd'hui, le VB a laissé place au VB .NET. Le suffixe **.NET** spécifie en fait qu'il nécessite le *framework* **.NET** de Microsoft afin de pouvoir être exécuté. À savoir qu'il y a également moyen d'exécuter un programme créé en VB sous d'autres plates-formes que Windows grâce à Mono.

M'sieur... qu'est-ce qu'un *framework* ?

Très bonne question. Un *framework* (dans notre cas, le *framework* .NET de Microsoft) est une sorte d'immense bibliothèque informatique contenant des outils qui vont faciliter la vie du développeur. Le *framework* .NET est compatible avec le Visual Basic et d'autres langages tels que le C#, le J#, etc.

Le *framework* .NET évolue : la version 2.0 en 2005, suivie de la 3.0 puis de la 3.5 en 2007, pour arriver aujourd'hui, en 2010, à la 4.0. Chaque version a apporté son lot de « livres » supplémentaires dans la bibliothèque. Au fil de ce tutoriel, j'évoquerai régulièrement ce *framework* qui va nous mâcher le travail dans bien des cas.

Après cette ignoble partie d'histoire, passons aux outils ! 

Sources :

- Visual Basic sur Wikipédia ;
- BASIC sur Wikipédia.

Notre outil : Visual Basic 2011 Express

L'environnement de développement

Eh oui, pour coder en Visual Basic, il nous faut des outils adaptés !

Comme je l'ai expliqué précédemment, nous allons utiliser du Visual Basic et non pas du BASIC. Cela signifie que nous créerons des interfaces graphiques et ergonomiques pour nos logiciels, et tout cela facilement.

Comment va-t-on procéder : utiliser un éditeur comme Paint et dessiner ce que l'on veut ?

Non, on ne va pas procéder de la sorte. Ce serait bien trop compliqué !
Sachez que des outils spécifiques existent, utilisons-les ! Bien, allons-y...

Visual Studio Express

Microsoft a créé une suite logicielle nommée « **Visual Studio** », qui rassemble Visual Basic, Visual C++, Visual C#, et j'en passe.

La suite provenant de Microsoft, on peut facilement deviner qu'elle coûte une certaine somme !
Heureusement, l'éditeur nous propose généreusement une version « express » gratuite de chaque logiciel de cette suite.

Nous allons donc utiliser **Visual Basic 2010 Express** (les étudiants peuvent toujours récupérer une version de Visual Studio 2010 sur la MSDN pour étudiants).

J'ai déjà installé une version de Visual Basic Express, mais celle de 2005 ou antérieure. Cela pose-t-il problème ?

Si vous êtes assez débrouillards, vous pouvez toujours conserver votre version. Je m'explique : Microsoft a sorti des versions différentes du *framework* (comme des librairies) pour chaque version de Visual Studio : **VS 2003 (Framework 1.1), VS 2005 (Framework 2.0), VS 2008 (Framework 3.5) et VS 2010 (Framework 4.0)**.

Vous l'avez donc certainement compris : si vous utilisez une autre version, vous aurez un ancien *framework*. De ce fait, certains objets ou propriétés évoqués ou utilisés dans le tutoriel sont peut-être différents voire inexistants dans les versions précédentes. Je vous conseille donc tout de même d'installer cette version *express* qui est relativement légère et vous permettra de suivre le tutoriel dans les meilleures conditions.

Cliquez sur l'image pour télécharger Visual Basic 2010 Express en français.

Installation de Visual Basic 2010 Express

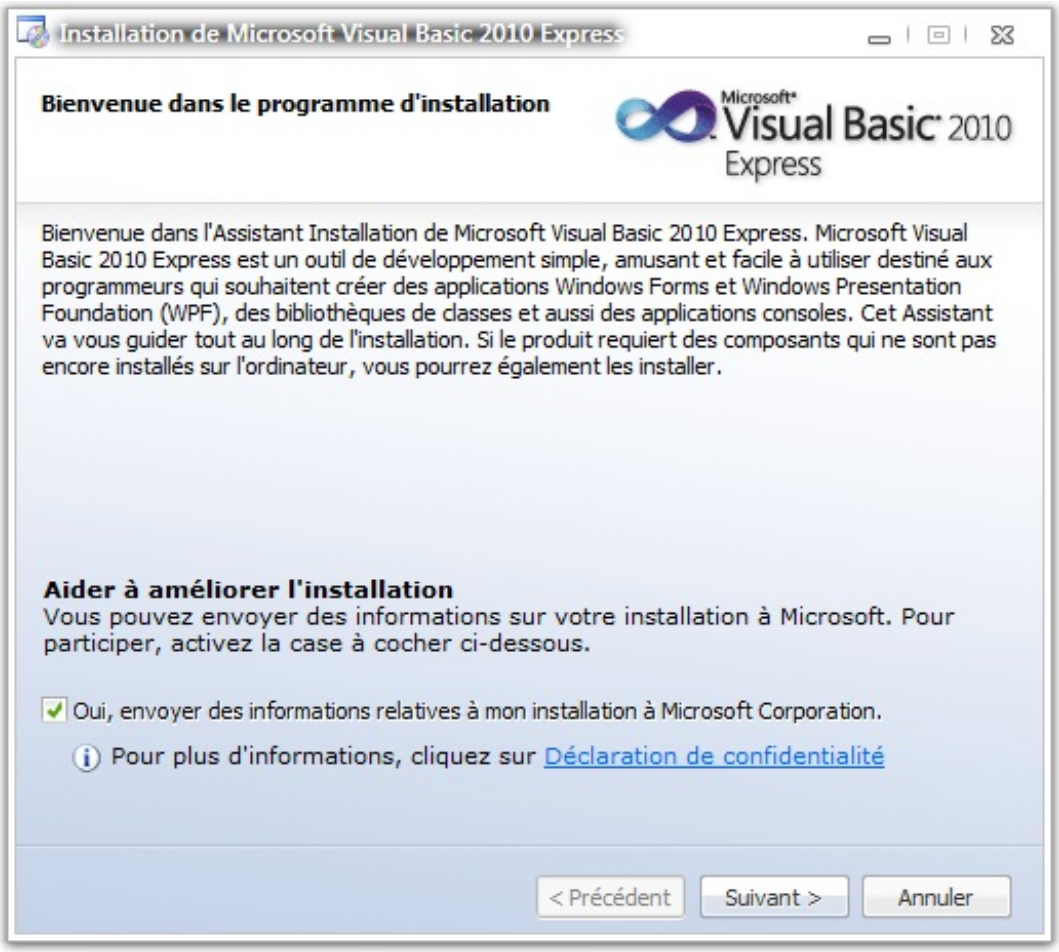
Étape par étape

Passons immédiatement au téléchargement du petit logiciel intermédiaire, qui ne pèse que quelques Mo et qui va télécharger Visual Basic 2010 Express.

Sachez que je travaillerai avec la version française du logiciel tout au long du tutoriel. Cela dit, rien ne vous empêche d'opter pour la version anglaise.

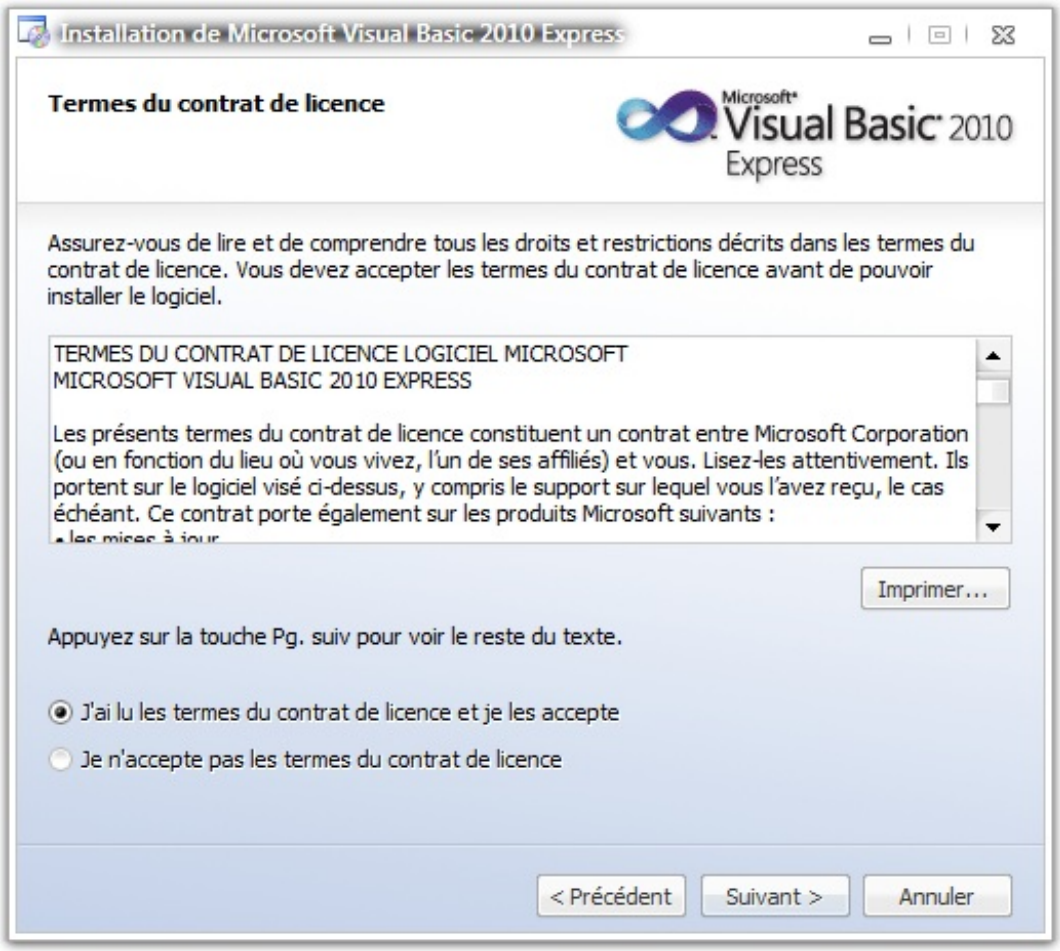
Accueil de l'installation

Vous lancez donc le programme, le laissez se préparer et arrivez à la première page.



Sur cette page, l'installateur vous propose déjà une case à cocher. Si vous autorisez Microsoft à récupérer des informations sur votre ordinateur et des statistiques pour ses bases de données, laissez comme tel. Dans le cas contraire, décochez la case. Cliquez ensuite sur le bouton « Suivant ».

Contrat de licence



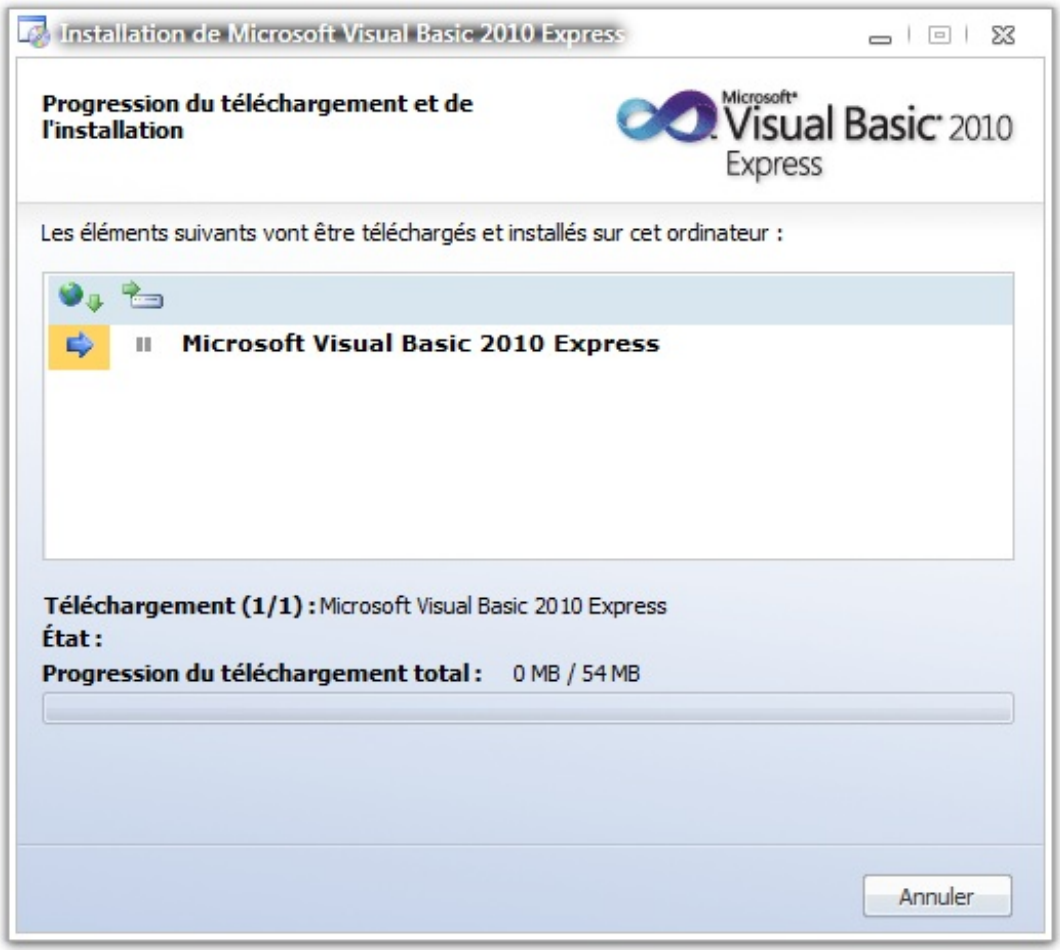
Lisez puis acceptez les termes du contrat de licence. Cela fait, appuyez une nouvelle fois sur « Suivant ».

Chemin d'installation

Comme pour n'importe quelle autre installation, choisissez le dossier dans lequel vous souhaitez que le logiciel s'installe. Cliquez ensuite sur « Installer ».

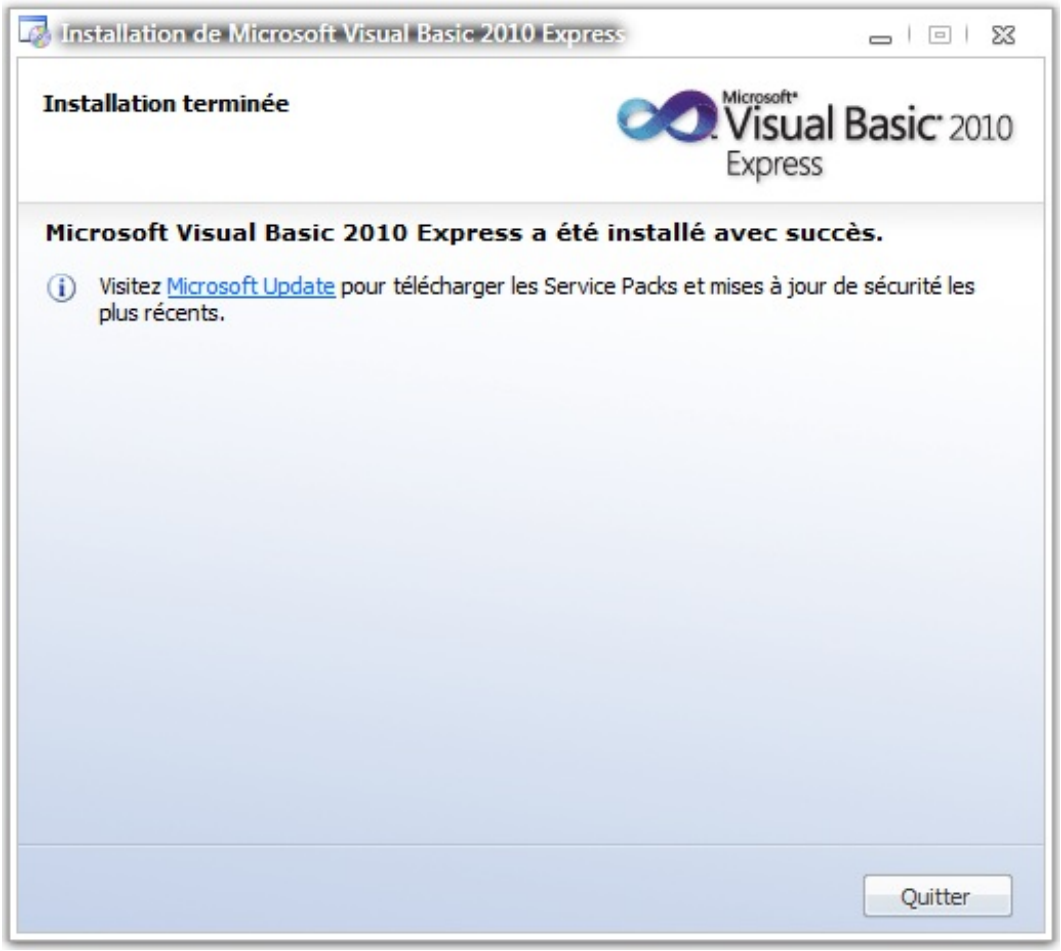
Téléchargement et installation

Une nouvelle page apparaît. Elle indique la progression du téléchargement du logiciel, le taux de transfert et la partie du programme en cours d'installation.



Il ne vous reste plus qu'à attendre la fin du téléchargement, suivi de l'installation. En attendant, faites un tour sur les articles de Wikipédia portant sur le Visual Basic.

Fini !



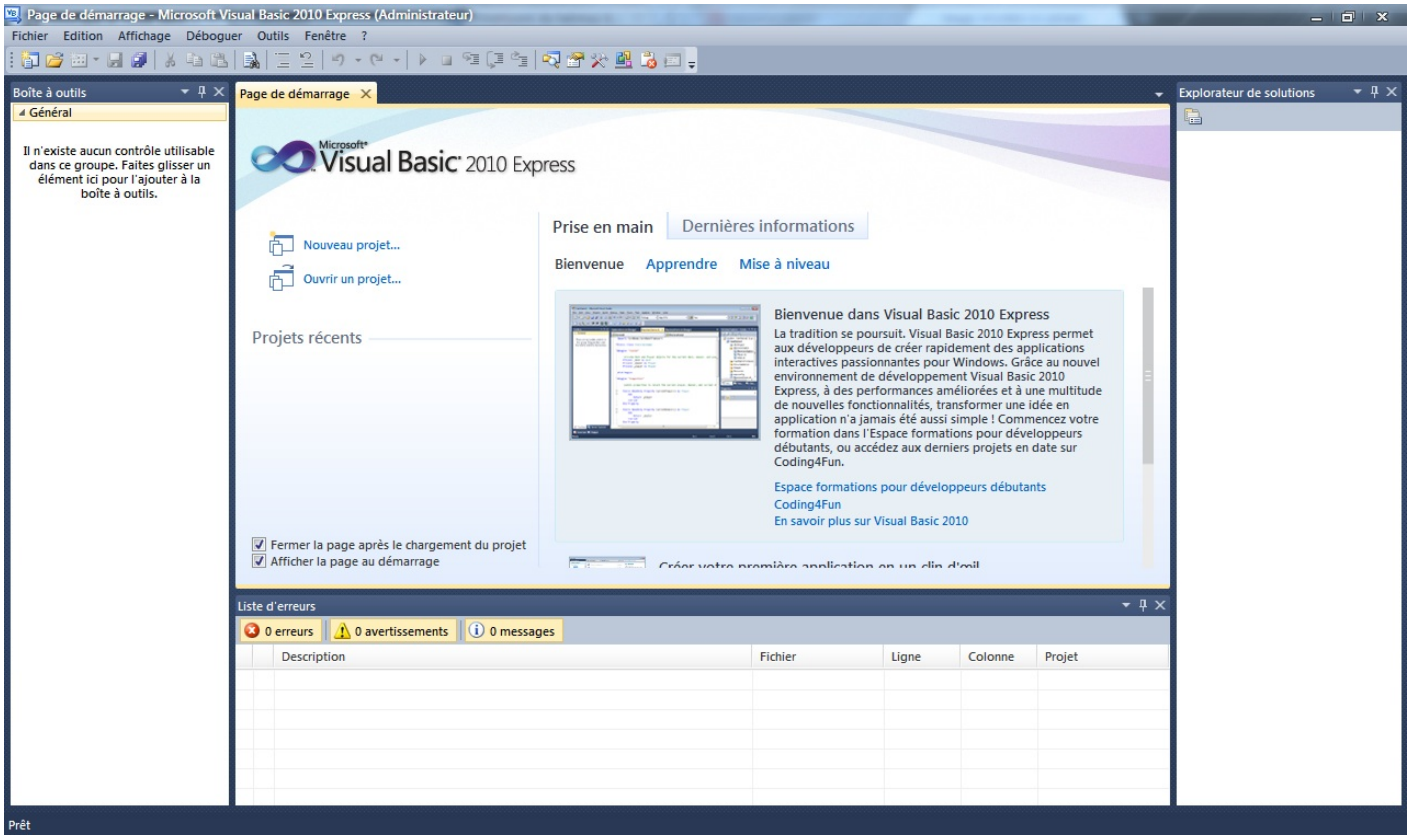
Nous voilà à présent avec Visual Basic 2010 Express installé ! Vous êtes désormais parés pour affronter cet abominable tutoriel.

Découverte de l'interface

L'interface de VB 2010 Express

Vous avez donc installé Visual Basic 2010 Express. En passant, sachez que ce dernier est un IDE (environnement de développement intégré) qui rassemble les fonctions de conception, édition de code, compilation et débogage. Lors du premier lancement, vous constatez qu'un petit temps de chargement apparaît : le logiciel configure l'interface pour la première fois.

Page d'accueil



Nous voici sur la page de démarrage du logiciel. Vous pouvez la parcourir, elle contient des informations utiles aux développeurs (vous) et conservera l'historique de vos projets récents.

Barre d'outils

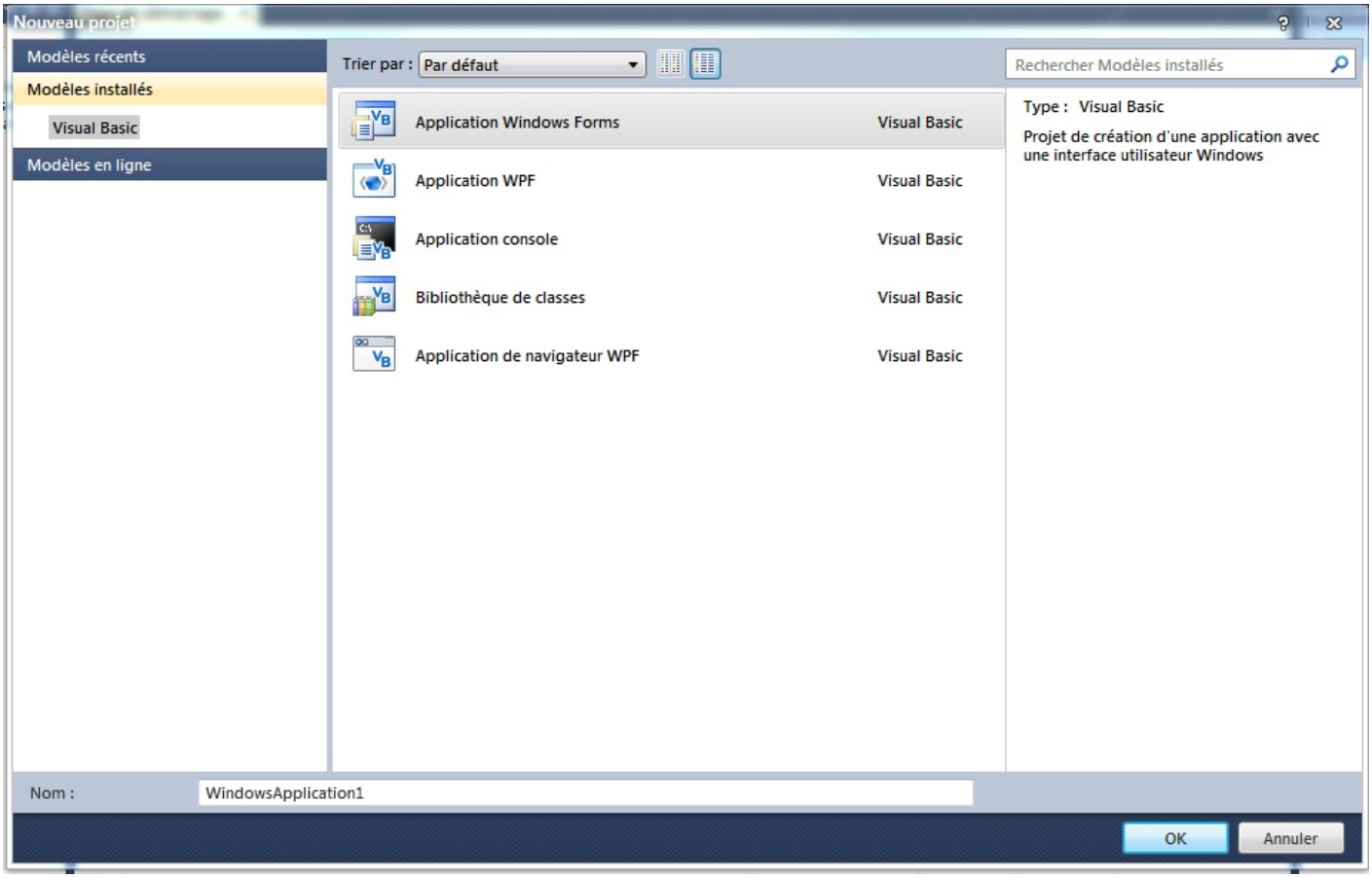
La barre d'outils vous sera indispensable afin de travailler en parfaite ergonomie. Je vais récapituler les boutons présents ci-dessus (de gauche à droite), actifs ou non durant vos travaux.

- Nouveau projet : crée un projet.
- Ouvrir un fichier : à utiliser pour ouvrir un projet existant ou une page simple.
- Ajouter un nouvel élément : disponible quand un projet est ouvert ; il permet d'ajouter des feuilles au projet.
- Enregistrer : raccourci CTRL + S.
- Enregistrer tout : raccourci CTRL + MAJ + S.
- Couper : raccourci CTRL + X.
- Copier : raccourci CTRL + C.
- Coller : raccourci CTRL + V.
- Rechercher : fort utile dans le cas de gros projets ; raccourci CTRL + F.

- Commenter les lignes : je reviendrai plus tard sur le principe des commentaires.
- Décommenter les lignes.
- Annuler : raccourci CTRL + Z.
- Rétablir : raccourci CTRL + MAJ + Z.
- Démarrer le débogage : expliqué plus tard.
- Arrêter le débogage : expliqué plus tard.
- Pas à pas détaillé : expliqué plus tard.
- Pas à pas principal : expliqué plus tard.
- Pas à pas sortant : expliqué plus tard.
- Explorateur de solutions : affiche la fenêtre de solutions.
- Fenêtre des propriétés : affiche la fenêtre des propriétés.
- Boîte à outils : permet d'afficher la boîte à outils.
- Gestionnaire d'extensions : permet de gérer les extensions que vous pouvez ajouter à Visual Basic Express.
- Liste d'erreurs : affiche la fenêtre d'erreurs.

Efforcez-vous de mémoriser les boutons importants, formatés en rouge dans la liste à puces ci-dessus. Il est également préférable de connaître les raccourcis clavier.

Nouveau projet



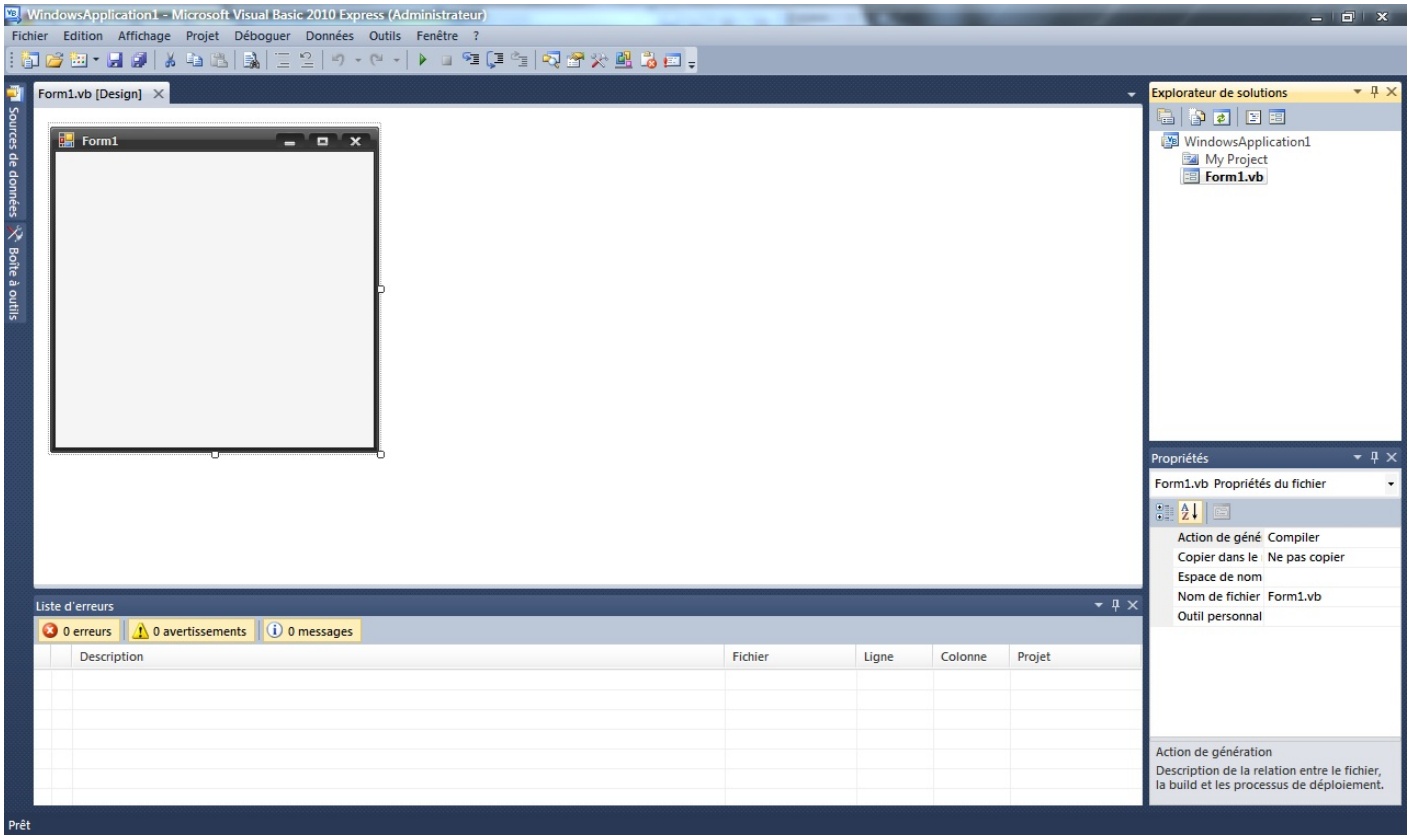
Je vous invite donc, seulement pour l'appréhension de l'interface, à créer un projet Windows Forms.
Pour ce faire, trois solutions s'offrent à vous : cliquer sur le bouton « Nouveau projet », se rendre dans le menu **Fichier → Nouveau projet** ou utiliser le raccourci clavier CTRL + N.

Cliquez donc sur l'icône

correspondant à **Application Windows Forms**.

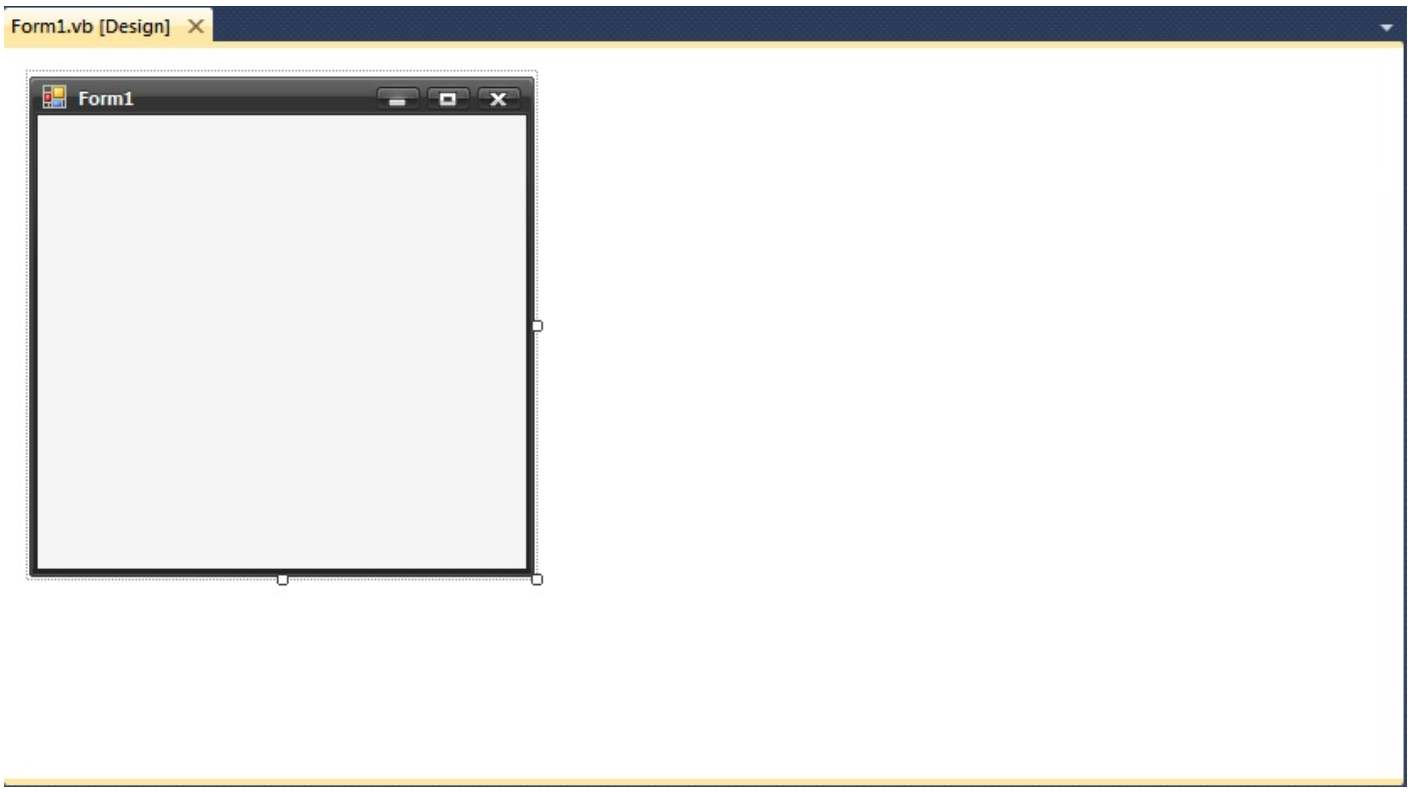
Saisissez un nom de projet dans la case « Nom ». Vous pouvez laisser le nom par défaut, ce projet ne sera pas utilisé.
Cliquez ensuite sur « OK », et vous voici dans un nouveau projet !

Vous remarquez que beaucoup plus de choses s'offrent à vous.



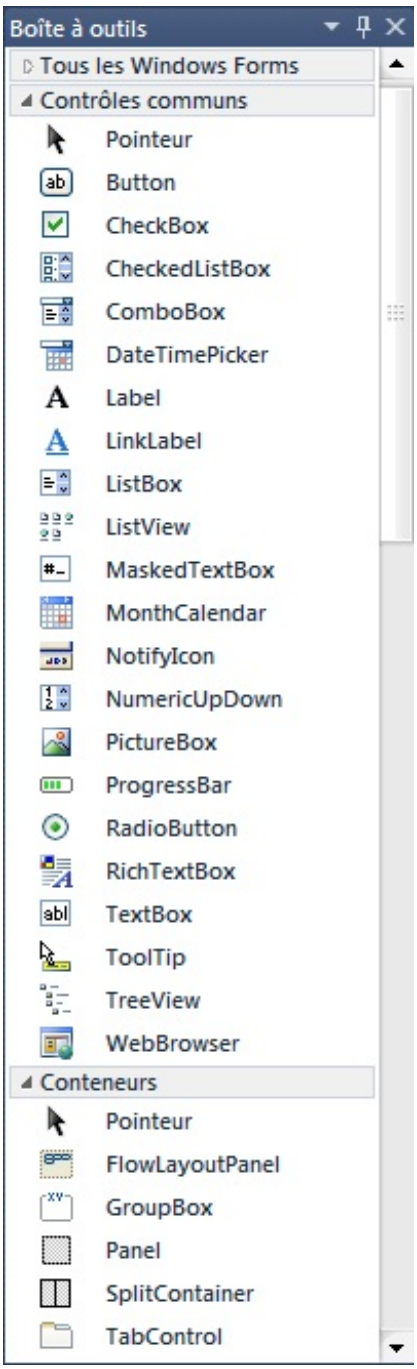
Nous allons tout voir en détail.

Espace de travail



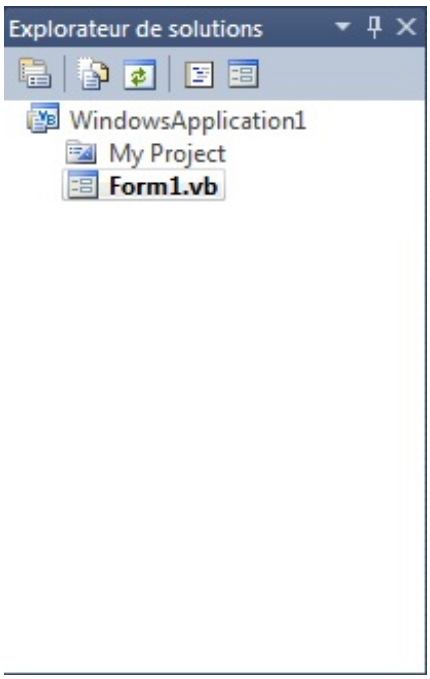
Cette partie correspond à notre espace de travail : c'est ici que nous allons créer nos fenêtres, entrer nos lignes de code, etc.

Boîte à outils



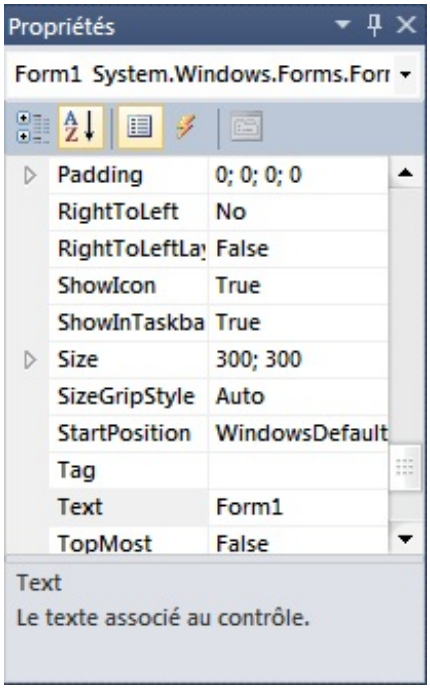
Sur la gauche de l'interface, nous avons accès à la boîte à outils. Pour afficher cette boîte, vous allez devoir cliquer sur le petit onglet qui dépasse sur la gauche. Une fois la boîte sortie, cliquez sur la punaise pour la « fixer » et la maintenir visible. La boîte à outils nous sera d'une grande utilité lorsque nous créerons la partie graphique de nos applications, mais inutile lors de l'écriture du code VB. Dès lors, si vous voulez la rentrer automatiquement, cliquez une nouvelle fois sur la punaise.

Fenêtre de solutions



De l'autre côté de notre écran, nous remarquons la fenêtre de solutions : elle récapitule l'arborescence de notre projet. Elle servira également à gérer les bases de données, mais plus tard.

Fenêtre des propriétés



Autre partie essentielle : la fenêtre des propriétés qui va nous permettre, en mode conception, de *modifier les propriétés* de nos objets. Vous n'avez rien compris ? Mettez ce terme dans un coin de votre tête, nous allons rapidement y revenir.

La dernière fenêtre est celle des erreurs. J'espère que vous n'en aurez pas l'utilité, mais elle saura se faire remarquer quand il le faudra, ne vous inquiétez pas.

En attendant, je vous laisse vous familiariser avec l'environnement : déplacez les boîtes, les fenêtres, et redimensionnez-les à votre guise.

Nous voici donc parés !

Premiers pas

Après cette petite découverte de notre IDE — qui signifie en français « Environnement de Développement Intégré » ; retenez bien ce terme car je vais l'utiliser par la suite —, nous allons immédiatement entrer dans le monde fabuleux de la programmation !

Hello World !

J'espère que vous connaissez et appréciez cette interface~~, car vous allez y travailler durant les dix prochaines années de votre vie~~ ! 

Tout d'abord, je tiens à m'excuser pour les termes que j'utiliserai dans ce tutoriel. Les puristes constateront immédiatement que les mots utilisés ne sont pas toujours exacts, mais je les trouve plus simples. Sachez que rien ne change : cela fonctionnera de la même façon.

Notre premier programme !

Nous allons donc aborder les principes fondamentaux du langage.

Pour cela, empressons-nous de créer un nouveau projet, cette fois en application console. 

Évitez d'utiliser des accents ou caractères spéciaux dans un nom de fichier ou de projet.

Sélectionnez cette icône : 

Ah ! Il y a déjà de nombreux mots de langue étrangère qui sont apparus !

Pas de panique, je vais vous expliquer.

Voici ce qui devrait s'afficher chez vous :

Code : VB.NET

```
Module Module1

    Sub Main()

    End Sub

End Module
```

Si ce n'est pas exactement ce code que vous voyez, faites en sorte que cela soit le cas, afin que nous ayons tous le même point de départ.

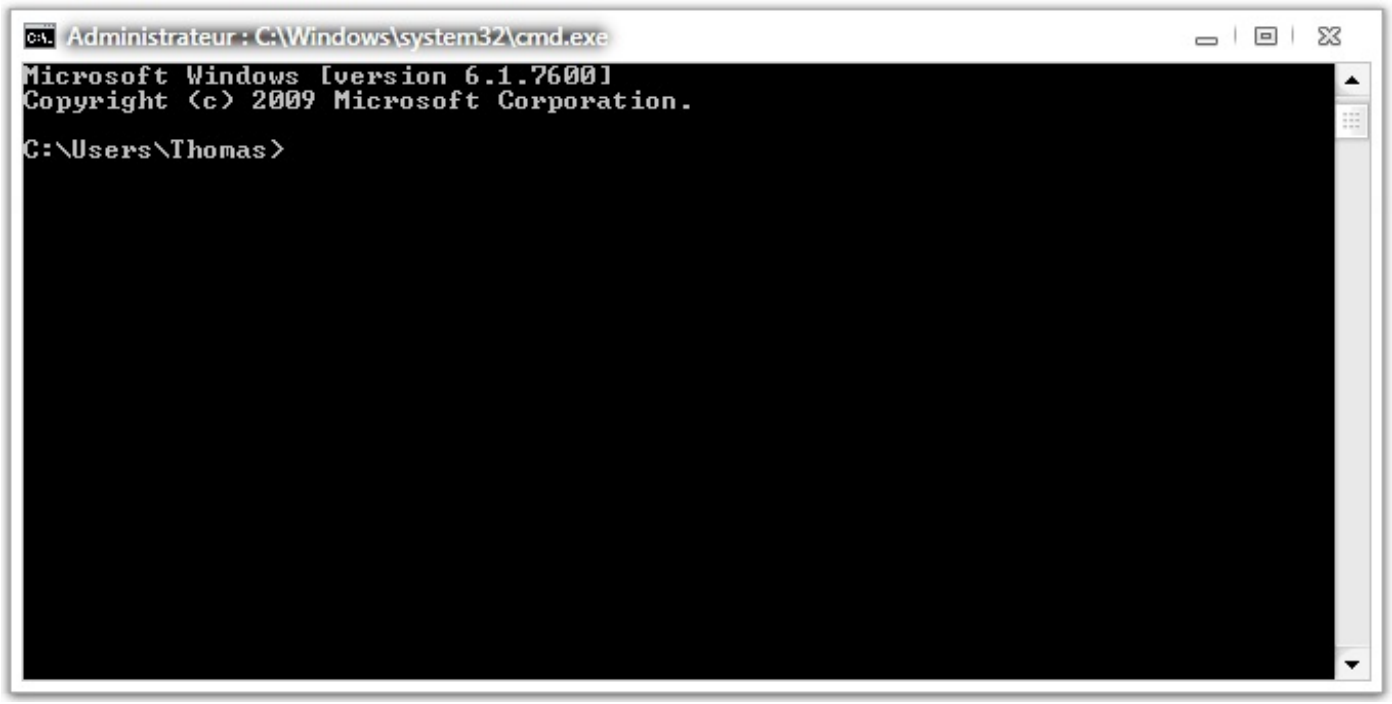
Ces mots barbares figurant désormais dans votre feuille de code sont indispensables ! Si vous les supprimez, l'application ne se lancera pas. 

Chaque grosse partie, telle qu'une *fonction*, un *module*, un *sub*, voire une *boucle conditionnelle* (désolé de parler aussi violemment dès le début ! ), aura une balise de début : ici, « **Module Module1** » et « **Sub Main()** », qui ont chacune une balise de fin (« **End Sub** » et « **End Module** »). Le mot « **Module1** » est le nom du module, que vous pouvez modifier si l'envie vous prend. Il nous sera réellement pratique lorsque nous utiliserons plusieurs feuilles.
Pour ce qui est du « **Main** », n'y touchez pas car, lorsqu'on va lancer le programme, la première chose que ce dernier va faire sera de localiser et de sauter dans la partie appelée « *Main* ». S'il ne la trouve pas, cela ne fonctionnera pas ! 

Les « parties » telles que **Main()** sont appelées des **méthodes** car elles sont précédées de « **Sub** ».

Tout d'abord, nous cherchons le moyen d'écrire quelque chose dans la console.

Ah, j'ai omis de vous expliquer en quoi consiste la console. Je suis confus !



Voici ma console. Je suis conscient que ce n'est visuellement pas exceptionnel, mais c'est plus simple pour apprendre les bases.

Mais pourquoi tant de haine ? Je souhaite plutôt faire Half-Life 3, moi ! Pas Space Invaders.

Du calme, les Zéros ! L'essentiel dans l'apprentissage de la programmation est d'y aller **progressivement**.
Cette console vous permettra d'apprendre les bases et les concepts fondamentaux du VB sans vous embrouiller directement l'esprit avec les objets qui orneront nos interfaces graphiques (c'est pour votre bien).

Nous allons donc créer un programme qui écrit dans cette console.

Je vais écrire pour vous la ligne qui va effectuer cette action, puisque mon travail est d'aider les Zéros que vous êtes !

Hello World !

Code : VB.NET

```
Console.Write("Hello World !")
```

Donc, pour ceux qui ont quelque peu suivi, où va-t-on placer cette ligne ? (Il n'y a pas cinquante possibilités !)

Secret (cliquez pour afficher)

Dans le **Main()**.

Une « ligne » est aussi appelée une **instruction**.

Eh bien oui, je l'ai dit plus haut : le programme va se rendre directement dans le **Main()**, autant donc y placer nos lignes (instructions) — c'est-à-dire entre « **Sub Main()** » et « **End Sub** ».

Pour lancer le programme, cliquez sur la petite flèche verte de la barre d'outils :

Ah ! je ne vois rien : la fenêtre s'ouvre et se referme trop rapidement !

Ah bon ?

Déroulement du programme

Excusez-moi, je vous explique : dans notre cas, le programme entre dans le *main* et exécute les actions de haut en bas, instruction par instruction. Attention, ce ne sera plus le cas lorsque nous aborderons des notions telles que les boucles ou les fonctions.

Voici nos lignes de code :

1. *Module Module1* : le programme entre dans son module au lancement. Forcément ; sinon, rien ne se lancerait jamais. La console s'initialise donc.
2. Il se retrouve à entrer dans le *main*. La console est ouverte.
3. Il continue et tombe sur notre ligne qui lui dit « affiche “Hello World !” », il affiche donc « Hello World ! » dans la console.
4. Il arrive à la fin du *main* (*end main*). Rien ne se passe, « Hello World ! » est toujours affiché.
5. Il rencontre le *End Module* : la console se ferme.

Résultat des courses : la console s'est ouverte, a affiché « Hello World ! » et s'est fermée à nouveau... mais tout cela en une fraction de seconde, on n'a donc rien remarqué !

La pause

La parade : donner au programme une ligne à exécuter sur laquelle il va attendre quelque chose. On pourrait bien lui dire : « Attends pendant dix secondes... », mais il y a un moyen plus simple et préféré des programmeurs : attendre une entrée. Oui, la touche **Entrée** de votre clavier (***return*** pour les puristes).
On va faire attendre le programme, qui ne bougera pas avant que la touche **Entrée** ne soit pressée.

Pour cela, voici la ligne de code qui effectue cette action :

Code : VB.NET

```
Console.Read()
```

Cette ligne dit à l'origine « lis le caractère que j'ai entré », mais nous allons l'utiliser pour dire au programme : « Attends l'appui sur la touche **Entrée**. »

Maintenant, où la placer ?

Secret (cliquez pour afficher)

Code : VB.NET

```
Module Module1
    Sub Main()
        Console.Write("Hello World !")
        Console.Read()
    End Sub
End Module
```

J'ai fourni l'intégralité du code pour ceux qui seraient déjà perdus. J'ai bien placé notre instruction après la ligne qui demande l'affichage de notre texte. En effet, si je l'avais mise avant, le programme aurait effectué une pause avant d'afficher la ligne : je l'ai dit plus haut, il exécute les instructions du haut vers le bas.

On clique sur notre fidèle flèche :

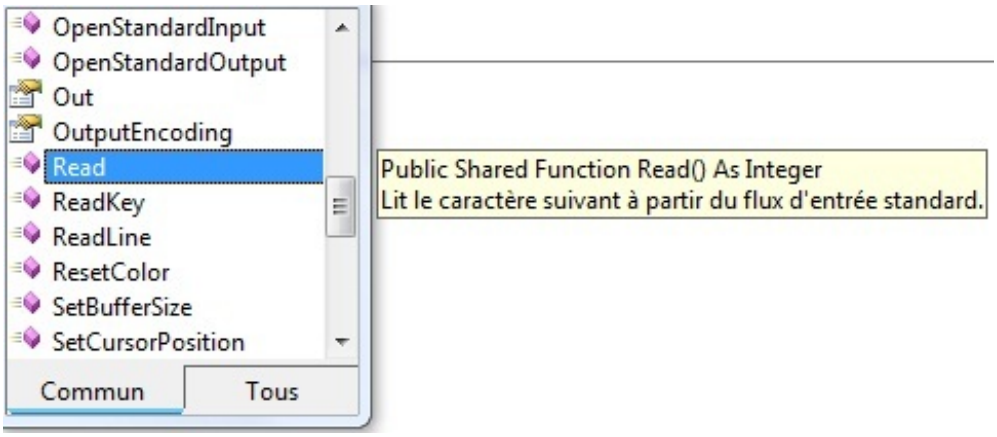
Code : Console

```
Hello World !
```

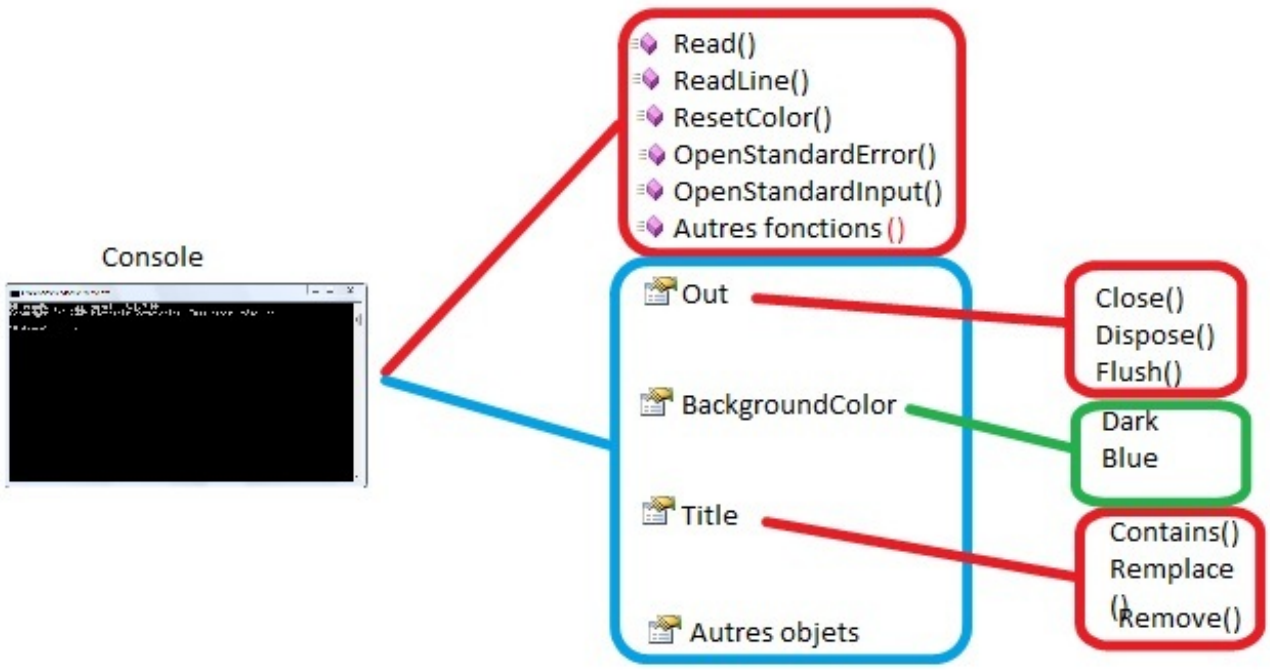
Victoire, notre « Hello World ! » reste affiché !
Si l'on presse la touche **Entrée**, la console se ferme : nous avons atteint nos objectifs !

Objets, fonctions ...

Vous l'avez peut-être remarqué : au moment où vous avez écrit « Console. », une liste s'est affichée en dessous de votre curseur. Dans cette partie, je vais vous expliquer l'utilité de cette liste.



Alors, encore une fois : un schéma !



J'ai essayé de faire le plus simple possible, mais accrochez-vous, vous allez recevoir beaucoup d'informations !
Nous avons donc notre console au début du schéma. Sous VB, la console est considérée comme un objet.
Cet objet possède des fonctions (en rouge) et d'autres objets (en bleu). Je vais déjà expliquer cela.

Fonctions

Une fonction est une séquence de code déjà existante et conçue pour obtenir un effet bien défini. Concrètement, cela nous permet de n'écrire qu'une seule fois ce que va faire cette séquence, puis d'appeler la fonction correspondante autant de fois que nous le voulons (qui exécutera bien entendu ce qu'on a défini au préalable dans la fonction... que des mots compliqués !).
Par exemple, nos deux lignes qui nous permettaient d'afficher « Hello World ! » et d'effectuer une pause auraient pu être placées dans une fonction séparée. Dans ce cas, en une ligne (l'appel de la fonction), on aurait pu effectuer cette séquence ; imaginez alors le gain de temps et les avantages dans des séquences de plusieurs centaines de lignes.

Un autre exemple : notre fonction **Write** avait pour but d'écrire ce que l'on lui donnait comme **arguments** (je vous expliquerai cela par la suite). La fonction **Write** a donc été écrite par un développeur qui y a placé une série d'instructions (et pas des

moindres !) permettant d'afficher du texte dans la console.

Objets

Pour faire simple, les objets permettent d'organiser notre code. Par exemple, notre fonction **Write** est, vous l'avez vu, liée à l'objet *Console*. C'est ainsi que le programme sait où effectuer le **Write**. Nous verrons plus en détail ce concept d'objets lorsque nous nous attaquerons au graphisme, mais vous venez de lire quelques notions de Programmation Orientée Objet (aussi appelée POO).

À noter : les « liens » entre les objets se font par des points (« **.** »). Le nombre d'objets liés n'est limité que si l'objet que vous avez sélectionné ne vous en propose pas. Sinon, vous pouvez en raccorder dix, si vous le voulez.

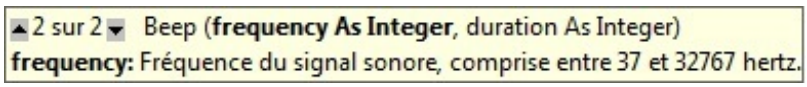
Fonctions, arguments

Pas de panique si vous n'avez pas compris ce concept de fonctions, d'objets, etc.

Nous allons justement nous pencher sur la structure d'un appel de fonction, car nous en aurons besoin très bientôt ; et pour cela, nous allons étudier une fonction simple : le **BEEP** (pour faire *bip* avec le haut-parleur de l'ordinateur).
Afin d'y accéder, nous allons écrire `Console.Beep` .

Ici, deux choix s'offrent à nous : le classique () ou **(frequency as integer, duration as integer)**.

Ouh là là, ça devient pas cool, ça !



Du calme, on y va doucement !
La première forme va émettre un *bip* classique lors de l'exécution.
La seconde demande des **arguments**. Il s'agit de paramètres passés à la fonction pour lui donner des indications plus précises. Précédemment, lorsque nous avons écrit `Write("Hello world")` , l'argument était « **"Hello world"** » ; la fonction l'a récupéré et l'a affiché, elle a donc fait son travail.

Pour certaines fonctions, on a le choix de donner des arguments ou non, selon la façon dont elles ont été créées (c'est ce qu'on appelle la **surcharge**, pour les personnes ayant déjà des notions d'orienté objet).

La seconde forme prend donc deux arguments, que vous voyez d'ailleurs s'afficher dès que vous tapez quelque chose entre les parenthèses, comme sur l'une des images au-dessus. Le premier sert à définir la fréquence du *bip* : entrez donc un nombre pour lui donner une fréquence. Le second, quant à lui, détermine la durée du *bip*. Les arguments sont délimités par une virgule, et si vous avez bien compris, vous devriez obtenir une ligne de ce genre :

Code : VB.NET

```
Console.Beep(500, 100)
```

Placez-la dans le programme comme nos autres lignes. Si vous la mettez avant ou après le *Console.Read()*, cela déterminera si le *bip* doit se produire avant ou après l'appui sur **Entrée**. Eh oui, le programme n'avancera pas tant que cette ligne ne sera pas exécutée.

Pourquoi n'y a-t-il pas de guillemets (doubles *quotes* : « " ») autour des nombres ?

Les nombres n'ont pas besoin de cette syntaxe particulière. Je m'explique : une variable ne peut pas avoir un nom composé uniquement de chiffres. Et donc, si vous écrivez des chiffres, le programme détectera immédiatement qu'il s'agit d'un nombre ; tandis que si vous écrivez des lettres, le programme ne saura pas s'il faut afficher le texte même ou si c'est le nom d'une variable. Donc, pour les noms de variables, il ne faut pas de guillemets, mais pour un simple texte, si.

Tenez, ça tombe bien, nous allons justement découvrir ce qu'est réellement une variable !

Allez, courage ! Certaines notions ou concepts vont être difficiles à assimiler, mais plus vous allez pratiquer, plus vous allez découvrir par vous-mêmes les actions que chaque élément peut effectuer. C'est cela qui fera de vous des programmeurs hors pair !

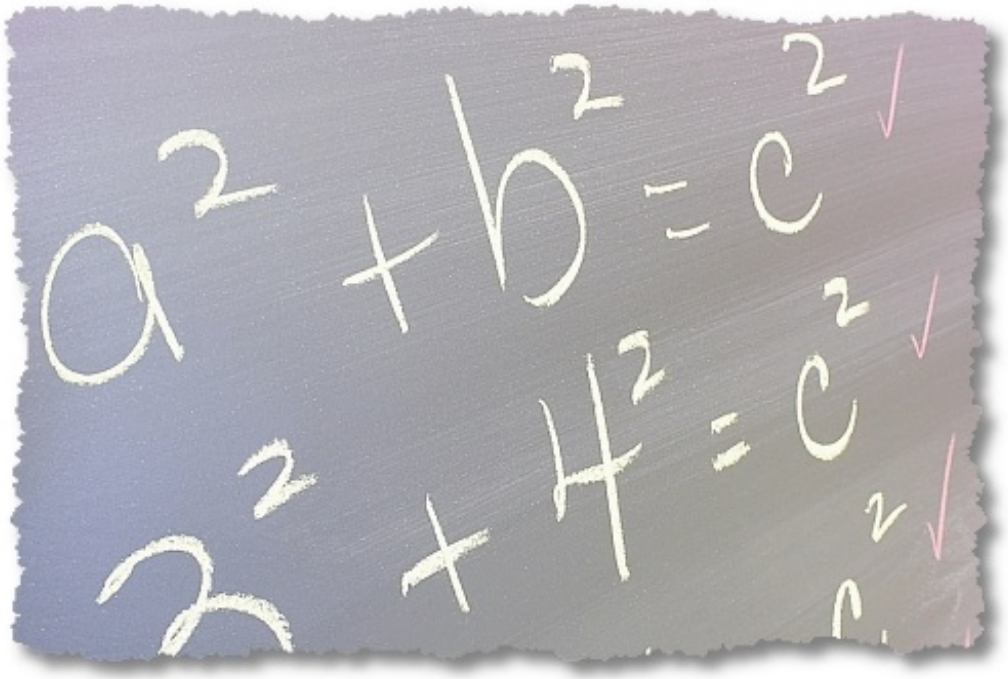
Les variables

Ce mot vous donne des boutons ? Il vous rappelle votre prof de maths du collège ?

N'ayez crainte ! Il s'agit d'un concept assez simple, qui ne requiert pas nécessairement des connaissances poussées en mathématiques (encore heureux !). Sur ce, je vais vous expliquer le concept de *variable*.

Qu'est-ce qu'une variable

Eh bien, comme son nom l'indique, c'est quelque chose qui varie. On peut y stocker pratiquement tout ce qu'on veut, comme par exemple des nombres, des phrases, des tableaux, etc.



N'est-ce pas ? Et c'est pour cette raison que les variables sont **omniprésentes** dans les programmes. Prenons comme exemple votre navigateur web favori : il stocke plein d'informations dans des variables, telles que l'adresse de la page, le mot de passe qu'il vous affiche automatiquement lorsque vous surfez sur votre site favori, etc.

Vous devez donc bien comprendre que ces variables vous serviront **partout** et dans tous vos programmes : pour garder en mémoire le choix que l'utilisateur a fait dans un menu, le texte qu'il a tapé il y a trente secondes... Les possibilités sont infinies.

Les types

Bien entendu, il y a tout de même des contraintes !
Les variables se déclinent sous différents types : il y a par exemple un type spécifique pour stocker des nombres, un autre pour stocker du texte, etc.
D'ailleurs, si vous tentez d'enregistrer du texte dans une variable créée pour contenir un nombre, l'ordinateur va vous afficher une petite erreur.

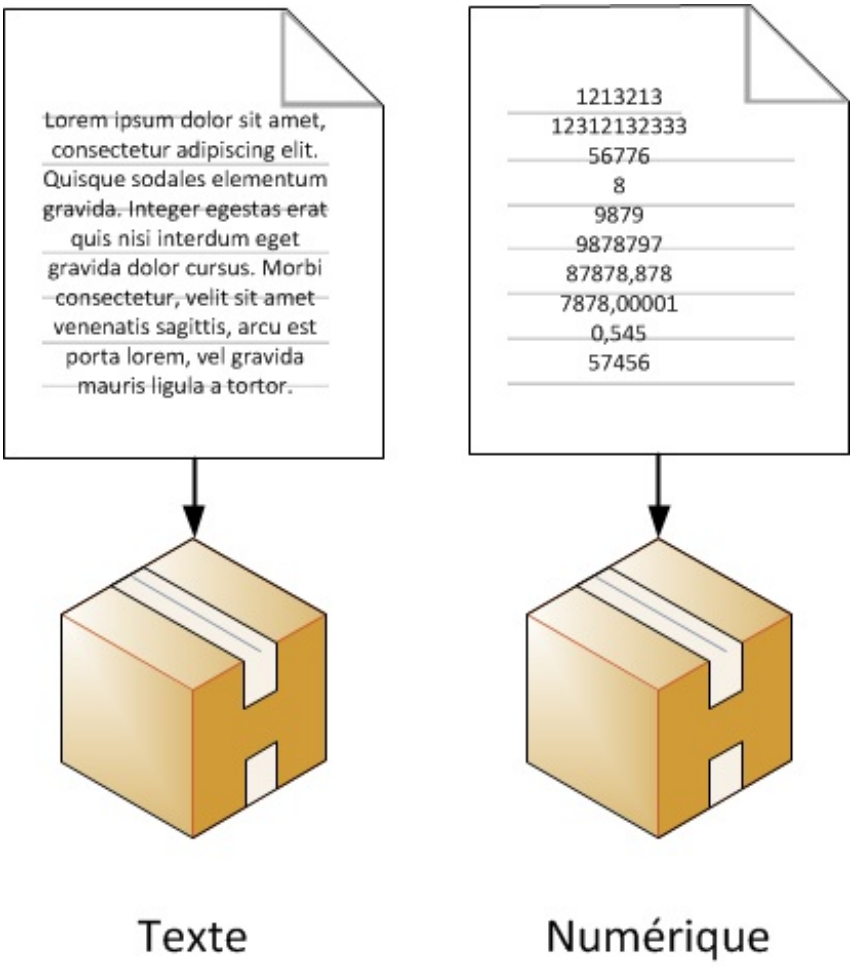


Tableau récapitulatif des types que nous allons utiliser

Nom	Explication
Boolean	Ce type n'accepte que deux valeurs : vrai ou faux. Il ne sert à rien, me direz-vous ; détrompez-vous.
Integer	Type de variable spécifique au stockage de nombres (existe sous trois déclinaisons ayant chacune une quantité de « place » différente des autres).
Double	Stocke des nombres à virgule.
String	Conçu pour stocker des textes ou des mots. Peut aussi contenir des nombres.
Date	Stocke une date et son heure sous la forme « 12/06/2009 11:10:20 ».

Il existe de nombreux autres types, mais ils ne vous seront pas utiles pour le moment.

J'ai précisé que le type ***integer*** (abrégé *int*) existait sous trois déclinaisons : ***int16***, ***int32*** et ***int64*** (le nombre après le mot ***int*** désigne la place qu'il prendra en mémoire). Plus le nombre est grand, plus votre variable prendra de la place, mais plus le nombre que vous pourrez y stocker sera grand. Pour ne pas nous compliquer la vie, nous utiliserons le ***integer*** (*int*) tout simple.

Si vous voulez en savoir plus sur l'espace mémoire utilisé par les variables, vous pouvez vous renseigner sur les « bits ». 

Pour ce qui est du texte, on a de la place : il n'y a pas de limite apparente. Vous pouvez donc y stocker sans souci un discours entier. Si le booléen, ce petit dernier, ne vous inspire pas et ne vous semble pas utile, vous allez apprendre à le découvrir. 

Les utiliser - la théorie

Comment allons-nous utiliser des variables ?

Telle est la question que nous allons éclaircir.

Que vous reste-t-il de vos cours de maths de 3e (sujet sensible) ?

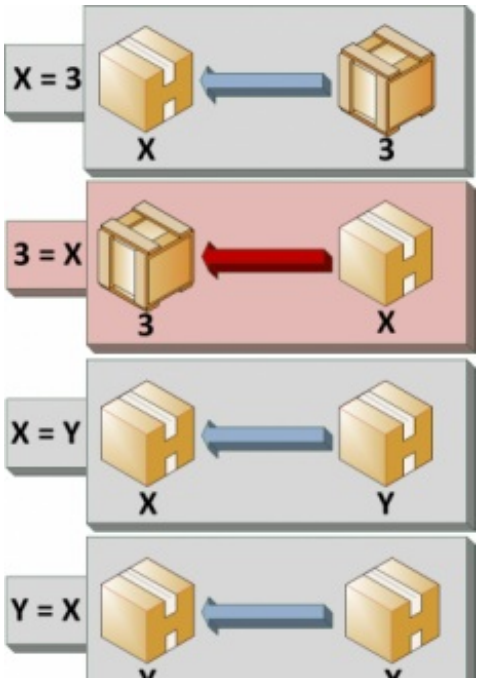
Bon... si j'écris ceci : $x^3 + 5x^2 - 3x + 1 = 0$, qu'est-ce qu'il se produit ?

1. Mon doigt se précipite pour éteindre l'écran.
2. Je ferme immédiatement le navigateur web.
3. Je prends une feuille de papier et résous cette équation.

Excusez-moi de vous avoir attaqués par derrière comme je l'ai fait, mais c'était dans le but de vous faire observer que l'attribution des variables est en de nombreux points similaire à notre vieil ami *x* en maths.
Comme le fait que pour attribuer une valeur à une variable, on place un « = » entre ces deux éléments.

Le sens

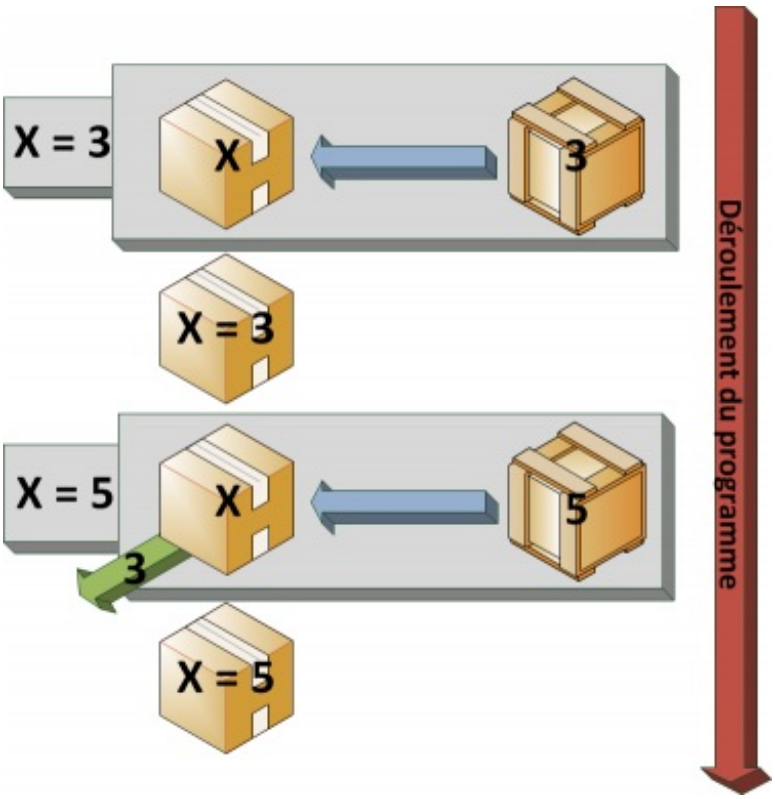
Ce n'est pas difficile : en VB, et même dans tous les langages de programmation, ce qui se situe à droite du « = » correspond à ce qui se trouve à gauche. C'est ainsi, cela ne changera pas !



Et donc, si vous regardez mes petits schémas :

- on entre le chiffre 3 dans la variable appelée X, pas de problème ;
- **ensuite, on souhaite mettre X dans 3 ! Aïe, cela ne va pas fonctionner ! Si vous écrivez quelque chose du genre, une erreur va se produire : comme si vous disiez « 3 = 2 », le compilateur va vous regarder avec des yeux grands comme ça et se demandera ce qu'il doit faire ;**
- ensuite, on met la variable Y dans la variable X ;
- et enfin, X dans Y.

Pas de problème pour le reste.
Je ne sais pas si vous l'avez remarqué, mais j'ai mis une variable dans une autre : c'est tout à fait possible, aucun problème à ce niveau-là. Par contre, l'affectation d'une valeur à une variable écrase l'ancienne valeur.



Revoilà donc mes boîtes. J'explique le schéma : vous ordonnez à votre programme « mets 3 dans X », ce qu'il va faire. Ensuite, vous lui dites « mets 5 dans X », mais il va oublier le 3 et écrire 5.

Attention, donc !

Mais en contrepartie, les variables offrent un stockage « nettoyable » à volonté. Je m'explique : vous pouvez les lire et y écrire autant de fois que vous le souhaitez. Lorsque vous lisez la valeur d'une variable, son contenu reste à l'intérieur (évident, me diront certains, mais sachez qu'il y a quelques dizaines d'années, les valeurs stockées dans la mémoire RAM s'effaçaient lors de leur lecture ; à l'époque, c'était des « tores » qui stockaient les bits et, lors de leur lecture, l'énergie se dissipait et faisait disparaître l'information).

Si vous avez bien compris, je pourrais écrire ceci (j'en profite pour vous montrer comment on initialise une variable, mais j'y reviendrai juste après) :

Code : VB.NET

```
Dim MaVariable As Integer
MaVariable = 5
MaVariable = 8
MaVariable = 15
MaVariable = 2
MaVariable = 88
MaVariable = 23
```

Que vaudra **MaVariable** à la fin de ces instructions ?

Secret (cliquez pour afficher)

23 !

Les utiliser - la pratique

Cette petite partie de lecture vous a ennuyé ? On va remédier à ce malaise.

Nous allons mettre en œuvre tout ce que je vous ai expliqué.

Tout d'abord, en VB, il faut **déclarer** une variable avant de l'utiliser. Autrement, l'ordinateur ne saura pas de quel type est la variable et ne saura donc pas comment réagir.

Nouvelle variable

Voici l'instruction servant à déclarer une variable, par exemple de type ***integer*** :

Code : VB.NET

```
Dim MaVariable As Integer
```

Pourquoi y a-t-il un terme appelé « MaVariable » ? Je pensais que le Visual Basic était conçu en anglais.

Effectivement, les mots que vous allez utiliser et qui serviront d'instructions dans vos programmes, comme par exemple « **Write** », « **If, Then** », etc., sont en anglais ; mais si l'on décortique la ligne que je viens de vous montrer, on obtient ceci :

Code VB	Dim	MaVariable	As	Integer
Français	Crée une variable	de nom « MaVariable »	en tant que	*integer*

En somme, le mot « MaVariable » est le nom attribué à la variable. C'est vous qui le choisissez !

Le nom d'une variable ne peut contenir d'espaces ; privilégiez plutôt un « _ » (***underscore***) ou une majuscule à chaque « nouveau mot », mais en liant le tout (comme dans mon exemple).

Autre chose à propos des noms : il y a des exceptions. En effet, une variable ne peut pas avoir comme nom un type ou le nom d'une boucle. Par exemple, si vous appelez votre variable « *date* », une erreur se produira car le type *date* existe déjà.

Bon, excusez-moi... j'avais dit qu'on allait pratiquer. Eh bien, on y va !

Retournez sur votre projet, qui doit encore être ouvert (du moins, je l'espère...). Si vous ne l'avez pas conservé, recréez-le (désolé).

Nous revoici donc avec nos lignes :

Code : VB.NET

```
Module Module1
    Sub Main()
        Console.Write("Salut")
        Console.Read()
    End Sub
End Module
```

J'ai retiré notre essai sur la fonction **BEEP**, car je pense que vous ne souhaitez pas entendre votre ordinateur *bipper* à chaque test.

MaVariable doit être égale à 5 !

Nous allons donc déclarer une variable et lui assigner une valeur. Je vous ai expliqué comment déclarer une variable.
Je vous ai aussi rapidement expliqué comment attribuer une valeur à une variable. Essayez donc de créer une variable de type ***integer*** appelée « **MaVariable** » et d'y entrer la valeur « 5 ».

Secret (cliquez pour afficher)

Code : VB.NET

```
Dim MaVariable As Integer
MaVariable = 5
```

Maintenant, où placer ces instructions ?

C'est la question fatidique ! Si vous vous rappelez le schéma sur l'ordre d'exécution des lignes dans un programme, vous devriez vous rappeler qu'une fois entrés dans un *sub* ou une fonction, sauf indications contraires (que nous étudierons dans un prochain chapitre), nous allons de haut en bas.

De ce fait, si vous avez besoin de votre variable à la ligne 4 de votre programme, il vous faut l'initialiser avant.
Même chose pour lui assigner une valeur : si vous l'affectez seulement à la ligne 6, la ligne 4 ne pourra pas lire ce résultat.

Dernière chose : je parie que vous souhaitez faire quelque chose de cette variable, ne serait-ce que l'afficher ?
J'ai expliqué comment afficher un texte avec le **Console.Write**. Pensez-vous être capables de faire en sorte d'afficher la valeur de la variable dans la console ?

Secret (cliquez pour afficher)

Code : VB.NET

```
Module Module1
    Sub Main()
        Dim MaVariable As Integer
        MaVariable = 5
        Console.Write(MaVariable)
        Console.Read()
    End Sub
End Module
```

Voici le résultat :

Code : Console

```
5
```

Voilà, vous pouvez tester : ce code affiche « 5 » dans la console.

Hop, hop, hop ! Pourquoi as-tu enlevé les doubles *quotes* (« " ») qui se trouvaient dans le **Write** ?

C'était le piège (sauf si vous avez bien lu précédemment) ! 

Si vous conservez les doubles *quotes*, la fonction **Write** affichera en dur le mot « MaVariable », et non sa valeur. Il faut donc enlever les doubles *quotes* pour que la fonction utilise le contenu de la variable **MaVariable**.

Si vous avez fait l'erreur, c'est normal : on va dire que je suis passé dessus trop rapidement. Mais après tout, c'est ainsi que vous apprendrez !

Vous êtes désormais capables de déclarer des variables et de leur affecter des valeurs. Vous en apprendrez plus durant l'exploration d'autres sujets. Rien de tel que de pratiquer pour s'améliorer. 

Dernière chose : il faut toujours essayer d'assigner une valeur à une variable dès le début ! Sinon, la variable n'est égale à rien, et des erreurs peuvent survenir dans certains cas. Donc, systématiquement : **une déclaration, une assignation**. 

Eh bien, je pense que vous savez désormais dans les grandes lignes à quoi servent les variables et comment les utiliser. Cela tombe bien car à partir de maintenant, elles seront **partout** !

Modifications et opérations sur les variables

Avant d'entamer ce nouveau chapitre, récapitulons et voyons où nous en sommes.

Nous avons installé Visual Basic 2010 Express et appris sommairement à l'utiliser ; nous avons vu comment créer un projet console, afficher du texte dans la console, mettre la console en « pause » et émettre un *bip*.

Ensuite, dans la partie « Les variables », nous avons appris à déclarer une variable, lui assigner une valeur et l'afficher.

Il est maintenant temps de s'amuser un peu avec nos nouvelles copines, les variables. Démarrons sans plus tarder !

Opérations sur une variable

Nous allons à présent apprendre comment modifier et effectuer des opérations sur ces variables.

Voici un exemple : vous souhaitez créer un programme qui calcule la somme de deux nombres ; pour ce faire, il vous faudra utiliser des opérations. Je vais vous expliquer la marche à suivre.

Reprenons notre programme, déclarons-y une variable « MaVariable » en *integer* et assignons-lui la valeur 5 (ce qui, normalement, est déjà fait).
Déclarons maintenant une seconde variable intitulée « MaVariable2 », de nouveau en *integer*, et assignons-lui cette fois la valeur 0.

Le nom de votre variable est unique : si vous déclarez deux variables par **le même nom**, une erreur se produira.

Si vous avez correctement suivi la démarche, vous devriez obtenir le résultat suivant :

Code : VB.NET

```
Module Module1
    Sub Main()
        Dim MaVariable As Integer
        Dim MaVariable2 As Integer
        MaVariable = 5
        MaVariable2 = 0
        Console.Write(MaVariable)
        Console.Read()
    End Sub
End Module
```

Dans le cas où vous avez plusieurs variables du même type, vous pouvez rassembler leur déclaration comme suit :

Code : VB.NET

```
Dim MaVariable, MaVariable2 As Integer
```

Second point : vous pouvez également initialiser vos variables dès leur déclaration, comme ci-dessous, ce qui est pratique pour les déclarations rapides.

Code : VB.NET

```
Dim MaVariable As Integer = 5
```

Attention toutefois, vous ne pouvez pas utiliser ces deux techniques ensemble ; une instruction du type `Dim MaVariable, MaVariable2 As Integer = 5` vous affichera une erreur ! C'est donc soit l'une, soit l'autre.

À l'attaque

Passons maintenant au concret !

On va additionner un nombre à notre variable « MaVariable ». Pour ce faire, rien de plus simple ! Démonstration.

Code : VB.NET

```
MaVariable + 5
```

Voilà ! Simple, n'est-ce pas ?
En résumé, vous avez additionné 5 à la variable « MaVariable ». Le programme a effectué cette opération. Seulement, le résultat n'est allé nulle part : nous n'avons pas mis le signe égal (« = ») !


Heu... tu me fais faire n'importe quoi ? 

Mais non, c'est pour vous montrer ce qu'il faut faire et ce qu'il ne faut pas faire.  *Imaginez un parent mettre ses doigts dans la prise et montrer à bébé l'effet que cela produit ; il comprendra tout de suite mieux !*  *(Mauvais exemple.)*

Pour y remédier, il faut ajouter le signe égal, comme lorsque nous initialisons nos variables.

Code : VB.NET

```
MaVariable2 = MaVariable + 5
```

Nous allons donc nous retrouver avec... 10, dans la variable « MaVariable2 ».

À noter que nous avons initialisé « MaVariable2 » avec 0. Si nous l'avions fait, par exemple, avec 7, le résultat aurait été identique puisque, souvenez-vous, l'entrée d'une valeur dans une variable écrase l'ancienne.


Il faut savoir que nous n'avons pas forcément besoin de deux variables. En effet, l'instruction `MaVariable = MaVariable + 5` aurait également affecté la valeur 10 à la variable « MaVariable ».

Plus en profondeur...

Vous savez à présent comment additionner un nombre à une variable.
Nous allons donc découvrir les autres opérations possibles.

Opération souhaitée	Symbole
Addition	+
Soustraction	-
Multiplication	*
Division	/
Division entière	\
Puissance	^
Modulo	Mod

J'explique ce petit tableau par un exemple : nous avons appris que, pour additionner 3 et 2, la syntaxe est `3+2` . C'est évident, me direz-vous... mais si je vous avais demandé de diviser 10 et 5, comment auriez-vous procédé ?

Eh bien, désormais, vous savez à quel « caractère » correspond chaque opération, la division de 10 et 5 aurait donc été : `10/5` .

Qu'est-ce que le modulo ?

Très bonne question. Le modulo est une opération spécifique en programmation, qui permet de récupérer le reste d'une division.

Exemples :

- `10 mod 5` correspond à `10/5` ; le résultat est 2, le reste est 0, donc `10 mod 5 = 0` .
- `14 mod 3` correspond à `14/3` ; le résultat est 4, le reste 2, donc `14 mod 3 = 2` .

x = 14 y = 3
x mod y = 2 x \ y = 4 x / y = 4.666666 x ^ y = 2 744

Nous allons immédiatement mettre en pratique ces informations.
Toutes les instructions que nous allons ajouter se feront dans le *main*.

Essayez d'attribuer des valeurs à vos variables et d'effectuer des opérations entre elles pour finalement stocker le résultat dans une troisième variable et afficher le tout.

Petite parenthèse : je vais en profiter pour vous expliquer comment écrire sur plusieurs lignes.
Si vous écrivez une fonction *Write*, puis une autre en dessous de façon à donner ceci :

Code : VB.NET

```
Console.Write("test")
Console.Write("test")
```

... vous allez vous retrouver avec le résultat suivant :

Code : Console

```
testtest
```

Afin d'écrire sur deux lignes, on va utiliser le procédé le plus simple pour le moment, qui est la fonction *WriteLine()*. Elle prend aussi comme argument la variable ou le texte à afficher mais insère un retour à la ligne au bout. Un code du genre...

Code : VB.NET

```
Console.WriteLine("test")
Console.WriteLine("test")
```

... produira le résultat suivant :

Code : Console

```
test
test
```

Avec ces nouvelles informations, essayez donc de multiplier 8 par 9 (chaque nombre mis au préalable dans une variable), le tout étant entré dans une troisième variable. En outre, un petit supplément serait d'afficher l'opération que vous faites.

Je vous laisse chercher ! 

Secret (cliquez pour afficher)

Code : VB.NET

```
Module Module1
    Sub Main()
        Dim MaVariable As Integer
        Dim MaVariable2 As Integer
        Dim MaVariable3 As Integer

        MaVariable = 8
        MaVariable2 = 9
        MaVariable3 = MaVariable * MaVariable2

        Console.Write("9 x 8 = ")
        Console.Write(MaVariable3)

        Console.Read()
    End Sub
End Module
```

Ce code, que j'ai tenté d'écrire de la façon la plus claire possible, nous affiche donc ceci :

Code : Console

```
9 x 8 = 72
```

Essayez de modifier les valeurs des variables, l'opération, etc.

Notre ligne `MaVariable3 = MaVariable * MaVariable2` aurait très bien pu être simplifiée sans passer par des variables intermédiaires : `MaVariable3 = 9 * 8` est donc également une syntaxe correcte.
Dans cette même logique, un `Console.Write(9 * 8)` fonctionnera également, car je vous ai expliqué que les arguments d'une fonction étaient séparés par des virgules ; donc, s'il n'y a pas de virgules, c'est le même argument.
Mais bon, n'allons pas trop vite.

Différentes syntaxes

Nous avons donc créé un code affichant `9 x 8 = 72`.

Ce code, comme vous l'avez certainement constaté, est très long pour le peu qu'il puisse faire ; pourtant, je vous ai donné quelques astuces.

Mon code peut donc être simplifié de plusieurs manières.

- Tout d'abord, l'initialisation lors de la déclaration :

Code : VB.NET

```
Dim MaVariable As Integer = 8
Dim MaVariable2 As Integer = 9
Dim MaVariable3 As Integer = 0
```

- Puis, un seul *Write* :

Code : VB.NET

```
Console.Write("9 x 8 = " & MaVariable3)
```

Wow, du calme ! À quoi sert le signe « & » ?

Bonne question. C'est ce qu'on appelle la **concaténation**, elle permet de « rassembler » deux choses en une ; ici, par exemple, j'ai rassemblé la chaîne de caractères `"9 x 8 = "` et le contenu de la variable, ce qui aura pour effet de m'afficher directement `9 x 8 = 72` (je parle de rassembler deux choses en une car, en faisant cela, on rassemble le tout dans le même argument).

- Dernière amélioration possible : la suppression d'une variable intermédiaire ; on se retrouve à faire l'opération directement dans le *Write*.

Code : VB.NET

```
Console.Write("9 x 8 = " & MaVariable * MaVariable2)
```

Ah, bah, autant effectuer directement le `9 * 8` en utilisant la concaténation !

Oui, effectivement. Mais dans ce cas, vos variables ne servent plus à rien et cette instruction ne sera valable que pour faire `9 * 8` ...

Grâce à ces modifications, notre code devient plus clair :

Code : VB.NET

```
Module Module1
    Sub Main()
        Dim MaVariable As Integer = 8
        Dim MaVariable2 As Integer = 9

        Console.Write("9 x 8 = " & MaVariable * MaVariable2)
```

```
            Console.Read()
        End Sub
    End Module
```

Attention toutefois en utilisant la concaténation : si vous en abusez, vous risquez de vous retrouver avec des lignes trop longues, et n'allez plus repérer ce qui se passe.

Pour cela, la parade arrive (eh oui, il y en a toujours une ; du moins, presque) ! 

Les commentaires

Les commentaires vont nous servir à éclaircir le code. Ce sont des phrases ou des indications que le programmeur laisse pour lui-même ou pour ceux qui travaillent avec lui sur le même code.

Une ligne est considérée comme commentée si le caractère « ' » (autrement dit, une simple *quote*) la précède ; une ligne peut aussi n'être commentée qu'à un certain niveau.

Exemples :

Code : VB.NET

```
'Commentaire
MaVariable = 9 * 6 ' Multiplie 9 et 6 et entre le résultat dans
MaVariable
```

Par exemple, voici notre programme dûment commenté :

Code : VB.NET

```
Module Module1
    Sub Main()
        'Initialisation des variables
        Dim MaVariable As Integer = 8
        Dim MaVariable2 As Integer = 9

        'Affiche "9 x 8 = " puis le résultat (multiplication de
MaVariable par MaVariable2)
        Console.Write("9 x 8 = " & MaVariable * MaVariable2)

        'Crée une pause factice de la console
        Console.Read()
    End Sub
End Module
```

Autre chose : si vous voulez commenter plusieurs lignes rapidement, ce qui est pratique lorsque vous testez le programme avec d'autres fonctions mais que vous souhaitez garder les anciennes si cela ne fonctionne pas, Visual Basic Express vous permet de le faire avec son interface. Sélectionnez pour cela les lignes souhaitées puis cliquez sur le bouton que j'ai décrit dans la barre d'outils, portant le nom « Commenter les lignes sélectionnées ».

Vous allez sûrement trouver cela long, fastidieux et inutile au début, mais plus tard, cela deviendra une habitude, et vous les insérerez sans que je vous le dise.

Il existe d'autres astuces pour expliquer et trier son code, que j'aborderai lorsque nous créerons nos propres fonctions.

Lire une valeur en console

Je vais immédiatement aborder ce passage, mais assez sommairement puisqu'il ne sera valable qu'en mode console.

Pour lire en mode console, par exemple si vous souhaitez que l'utilisateur saisisse deux nombres que vous additionnerez, il vous faut utiliser la fonction *ReadLine()*. Nous avions utilisé *Read*, mais cette fonction lit uniquement un caractère, elle est donc inutile pour les nombres supérieurs à 9.

Notre nouvelle fonction s'utilise de la manière suivante :

Code : VB.NET

```
MaVariable = Console.ReadLine()
```

Vous avez donc certainement déjà dû écrire ce code, qui multiplie les deux nombres entrés :

Secret (cliquez pour afficher)

Code : VB.NET

```
Module Module1
    Sub Main()
        'Initialisation des variables
        Dim MaVariable As Integer = 0
        Dim MaVariable2 As Integer = 0

        Console.WriteLine("- Multiplication de deux nombres -")

        'Demande du premier nombre stocké dans MaVariable
        Console.WriteLine("Veuillez entrer le premier nombre")
        MaVariable = Console.ReadLine()
        'Demande du second nombre stocké dans MaVariable2
        Console.WriteLine("Veuillez entrer le second nombre")
        MaVariable2 = Console.ReadLine()

        'Affiche "X x Y = " puis le résultat (multiplication de
MaVariable par MaVariable2)
        Console.WriteLine(MaVariable & " x " & MaVariable2 & " = "
& MaVariable * MaVariable2)

        'Crée une pause factice de la console
        Console.ReadLine()
    End Sub
End Module
```

Ce programme demande donc les deux nombres, l'un puis l'autre, et les multiplie.

Cette fonction ne formate et ne vérifie pas la réponse ; autrement dit, si votre utilisateur écrit « salut » et « coucou » au lieu d'un nombre, le programme plantera car il essaiera de saisir des caractères dans un type réservé aux nombres.

Ce qui nous amène à notre prochain chapitre : les boucles conditionnelles.

Conditions et boucles conditionnelles

Une quoi ?

Une boucle conditionnelle est quelque chose de fort utile et courant en programmation.

Je vous donne un exemple : imaginez que vous souhaitez que votre application effectue une action uniquement si le nombre entré est égal à 10, ou une autre tant qu'il est égal à 10. Eh bien, c'est précisément dans ce cas de figure que les boucles conditionnelles trouvent leur utilité.

Les boucles conditionnelles

Aperçu des différentes boucles

« If », mot anglais traduisible par « si »

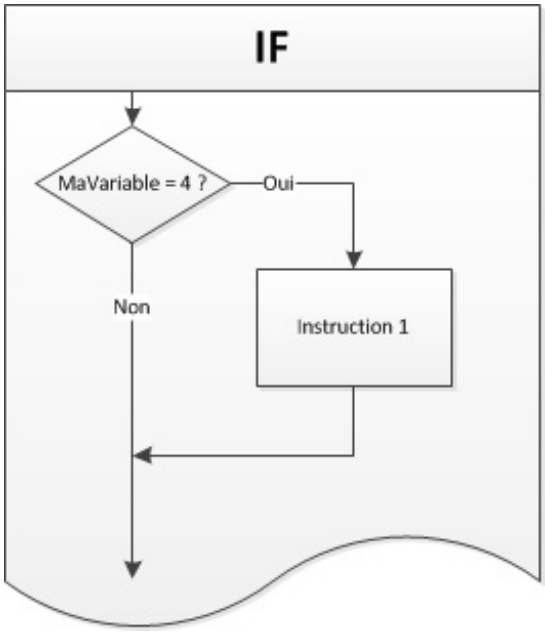
Retenez bien que les « mots » que le programme comprend et utilise sont anglais et ont donc une traduction qui peut vous aider à vous rappeler à quoi ils servent.

Attaquons avec la boucle la plus simple, mais non sans intérêt : *If* .

Une ligne commençant par ***If*** est toujours terminée par ***Then***, ce qui signifie « Si, alors ». C'est entre ces deux mots que vous placez la condition souhaitée.

Donc, si j'écris le code `If MaVariable = 10 Then` , ce qui se trouve en dessous ne sera exécuté que si la valeur de « MaVariable » est égale à 10.

Code VB	If	MaVariable	= 10	Then
Français	Si	« MaVariable »	est égale à 10	alors



Comment cela, tout ce qui se trouve en dessous ? Tout le reste du programme ?

Eh bien oui, du moins jusqu'à ce qu'il rencontre ***End If*** , traduisible par « Fin si ». Comme pour un *Sub* ou un *Module*, une boucle est associée à sa fin correspondante.

En clair, ***If*** , ***Then*** et ***End If*** sont indissociables !

Code : VB.NET

```
If MaVariable = 10 Then
      MaVariable = 5
End If
```

Si vous avez bien compris, vous devriez être capables de m'expliquer l'utilité du code ci-dessus.

Secret (cliquez pour afficher)

Si « MaVariable » est égale à 10, il met « MaVariable » à 5.

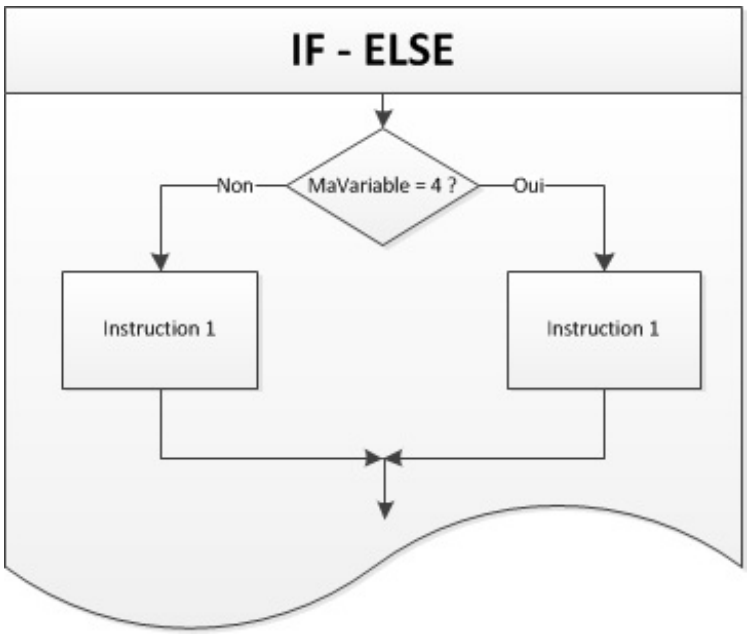
Exactement !

Mais si tu mets « MaVariable » à 5 dans la boucle, le programme ne va pas en sortir puisque ce n'est plus égal à 10 ?

Bonne observation. Eh bien, non, cela ne change rien : c'est en arrivant à la ligne du ***If*** que tout se joue. Ensuite, si la variable change, le programme ne s'en préoccupe plus.

« Else », mot anglais traduisible par « sinon »

« Sinon », il faut y penser des fois pour gérer toutes les éventualités.
Le ***Else*** doit être placé dans une boucle ***If*** , donc entre le ***Then*** et le ***End If*** .



La syntaxe est la suivante :

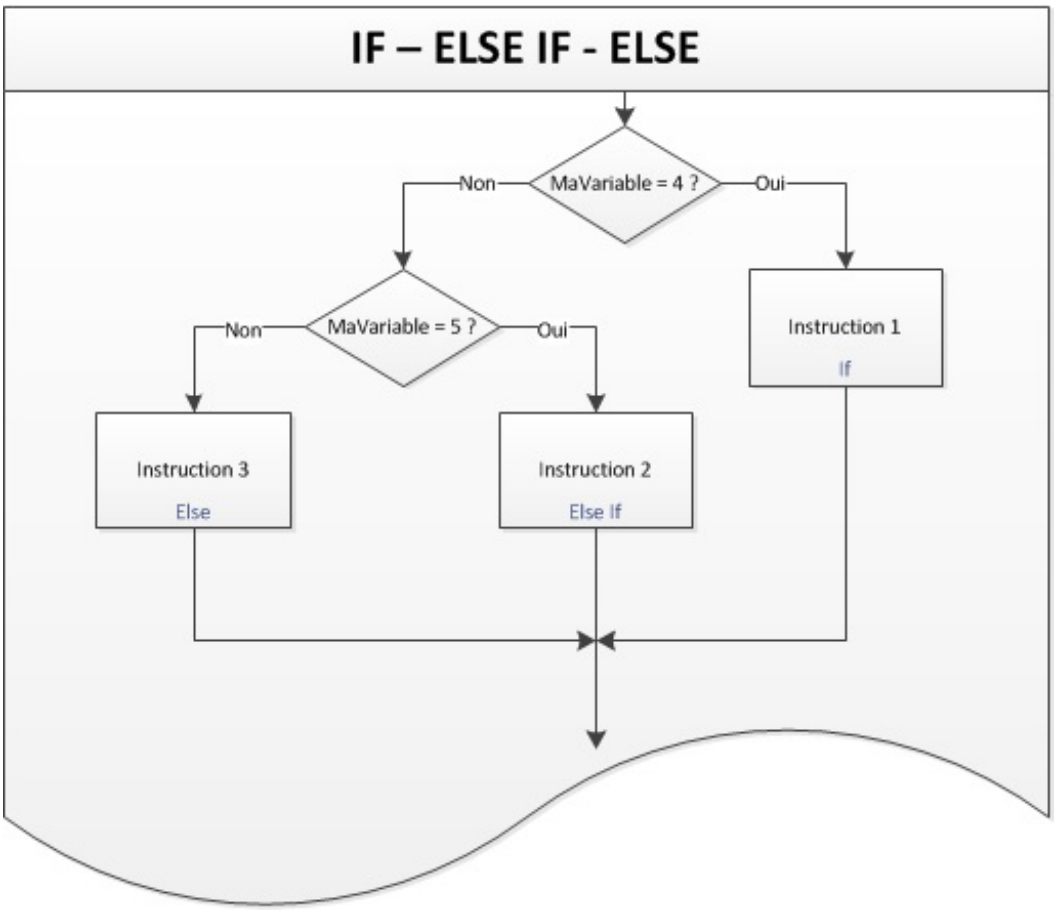
Code : VB.NET

```
If MaVariable = 10 Then
    'Code exécuté si MaVariable = 10
Else
    'Code exécuté si MaVariable est différente de 10
End If
```

Code VB	Else
Français	Sinon

Je vais en profiter pour vous signaler que le symbole *différent* en VB s'écrit « <> ». Autrement dit, un signe « inférieur » et un signe « supérieur » accolés.

ElseIf



Si vous voulez un cas particulier et non le reste des autres cas de votre condition, il existe le ***ElseIf*** .

Voici un exemple :

Code : VB.NET

```
If MaVariable = 10 Then
    'Code exécuté si MaVariable = 10
ElseIf MaVariable = 5 Then
    'Code exécuté si MaVariable = 5
Else
    'Code exécuté si MaVariable est différente de 10 et de 5
End If
```

Code VB	ElseIf
Français	Sinon, si

Dernière chose : les boucles *If* , *Then* et *ElseIf* peuvent s'imbriquer, ce qui signifie qu'on peut en mettre plusieurs l'une dans l'autre.

Code : VB.NET

```
If MaVariable = 10 Then
    If MaVariable2 = 1 Then
        'Code exécuté si MaVariable = 10 et MaVariable2 = 1
    Else
        'Code exécuté si MaVariable = 10 et MaVariable2 <> 1
    End If
ElseIf MaVariable = 5 Then
    If MaVariable2 = 2 Then
        'Code exécuté si MaVariable = 5 et MaVariable2 = 1
    End If
Else
        'Code exécuté si MaVariable est différente de 10 et de 5
End If
```

Select

Nous avons vu *If*, *ElseIf* et *Else*.

Mais pour ce qui est, par exemple, du cas d'un menu, vous avez dix choix différents dans votre menu. Comment faire ?

Une première façon de procéder serait la suivante :

Code : VB.NET

```
If Choix = 1 Then
     Console.WriteLine("Vous avez choisi le menu n° 1")
ElseIf Choix = 2 Then
     Console.WriteLine("Vous avez choisi le menu n° 2")
ElseIf Choix = 3 Then
     Console.WriteLine("Vous avez choisi le menu n° 3")
ElseIf Choix = 4 Then
     Console.WriteLine("Vous avez choisi le menu n° 4")
ElseIf Choix = 5 Then
     Console.WriteLine("Vous avez choisi le menu n° 5")
ElseIf Choix = 6 Then
     Console.WriteLine("Vous avez choisi le menu n° 6")
ElseIf Choix = 7 Then
     Console.WriteLine("Vous avez choisi le menu n° 7")
ElseIf Choix = 8 Then
     Console.WriteLine("Vous avez choisi le menu n° 8")
ElseIf Choix = 9 Then
     Console.WriteLine("Vous avez choisi le menu n° 9")
ElseIf Choix = 10 Then
     Console.WriteLine("Vous avez choisi le menu n° 10")
Else
     Console.WriteLine("Le menu n'existe pas")
End If
```

Il s'agit de la méthode que je viens de vous expliquer (qui est tout à fait correcte, ne vous inquiétez pas !).

Il faut néanmoins que vous sachiez que les programmeurs sont très fainéants, et ils ont trouvé sans cesse des moyens de se simplifier la vie. C'est donc dans le cas que nous venons d'évoquer que les ***Select*** deviennent indispensables, grâce auxquels on simplifie le tout. La syntaxe se construit de la forme suivante :

Code : VB.NET

```
Select Case MaVariable
      Case 1
          'Si MaVariable = 1
      Case 2
          'Si MaVariable = 2
      Case Else
          'Si MaVariable <> 1 et <> 2
   End Select
```

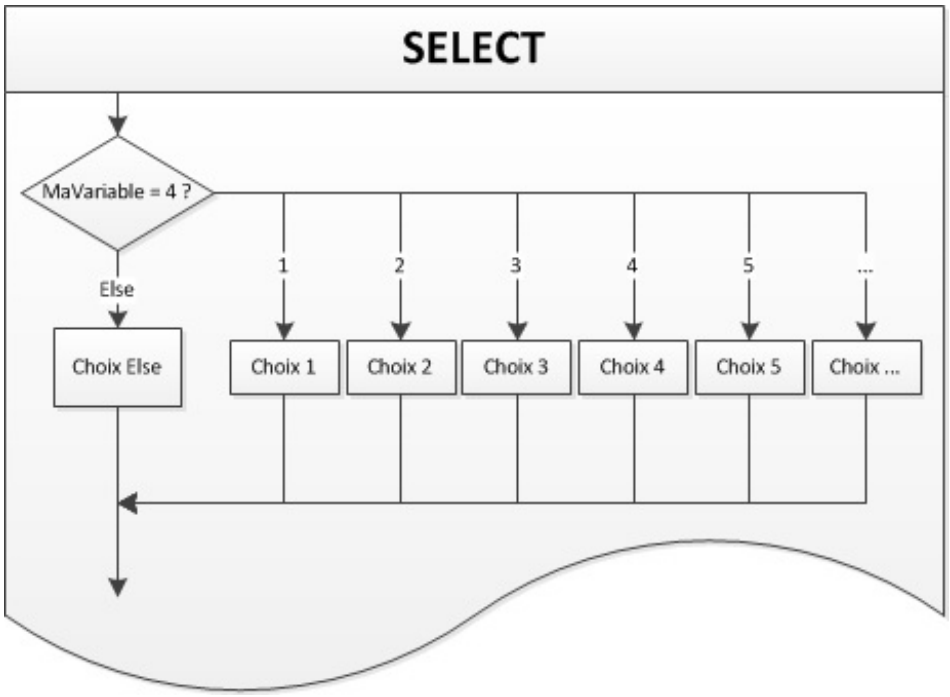
Code VB	Select	Case	MaVariable
Français	Sélectionne	dans quel cas	« MaVariable » vaut

Dans le même cas de figure, revoici notre menu :

Code : VB.NET

```
Select Case Choix
        Case 1
            Console.WriteLine("Vous avez choisi le menu n° 1")
        Case 2
            Console.WriteLine("Vous avez choisi le menu n° 2")
        Case 3
            Console.WriteLine("Vous avez choisi le menu n° 3")
        Case 4
            Console.WriteLine("Vous avez choisi le menu n° 4")
        Case 5
            Console.WriteLine("Vous avez choisi le menu n° 5")
        Case 6
            Console.WriteLine("Vous avez choisi le menu n° 6")
        Case 7
            Console.WriteLine("Vous avez choisi le menu n° 7")
        Case 8
            Console.WriteLine("Vous avez choisi le menu n° 8")
        Case 9
            Console.WriteLine("Vous avez choisi le menu n° 9")
        Case 10
            Console.WriteLine("Vous avez choisi le menu n° 10")
        Case Else
            Console.WriteLine("Le menu n'existe pas")
    End Select
```

Pour que vous compreniez bien, voici un petit schéma :



Ce code correspond exactement à celui qui se trouve plus haut. Le ***Case Else***, ici aussi, prend en compte toutes les autres possibilités.

Encore une fois : attention à bien penser à la personne qui fera ce qu'il ne faut pas faire !

Petites astuces avec Select

Si je souhaite que pour les valeurs 3, 4 et 5, il se passe la même action, dois-je écrire trois *Case* avec la même instruction ?

Non, une petite astuce du *Select* est de rassembler toutes les valeurs en un *Case*. Par exemple, le code suivant...

Code : VB.NET

```
Select Case Choix
       Case 3,4,5
            'Choix 3, 4 et 5
End Select
```

... est identique à celui-ci :

Code : VB.NET

```
Select Case Choix
       Case 3
            'Choix 3, 4 et 5
       Case 4
            'Choix 3, 4 et 5
       Case 5
            'Choix 3, 4 et 5
End Select
```

Astuce également valable pour de grands intervalles : le code suivant...

Code : VB.NET

```
Select Case Choix
       Case 5 to 10
            'Choix 5 à 10
End Select
```

... correspond à ceci :

Code : VB.NET

```
Select Case Choix
       Case 5
            'Choix 5 à 10
       Case 6
            'Choix 5 à 10
       Case 7
            'Choix 5 à 10
       Case 8
            'Choix 5 à 10
       Case 9
            'Choix 5 à 10
       Case 10
            'Choix 5 à 10
End Select
```

Voilà, j'espère que ces différentes formes vous seront utiles.

While

À présent, nous allons réellement aborder le terme de « boucle ».

Tu veux dire qu'on ne les utilisait pas encore ?

Non, ce ne sont pas à proprement parler des boucles ; en programmation, on appelle *boucle* un espace dans lequel le programme reste pendant un temps choisi, c'est-à-dire qu'il tourne en rond.

On va tout de suite étudier le cas de ***While***.

« While », mot anglais traduisible par « tant que »

Vu la traduction du mot « *while* », vous devriez vous attendre à ce que va faire notre boucle.
Elle va effectivement « tourner » tant que la condition est **vraie**.

Retenez bien ce « vrai ». Vous souvenez-vous du concept des booléens que nous avons étudié dans le chapitre sur les variables ? Eh bien voilà, dans ce cas-ci, le ***While*** va vérifier que le booléen est vrai.

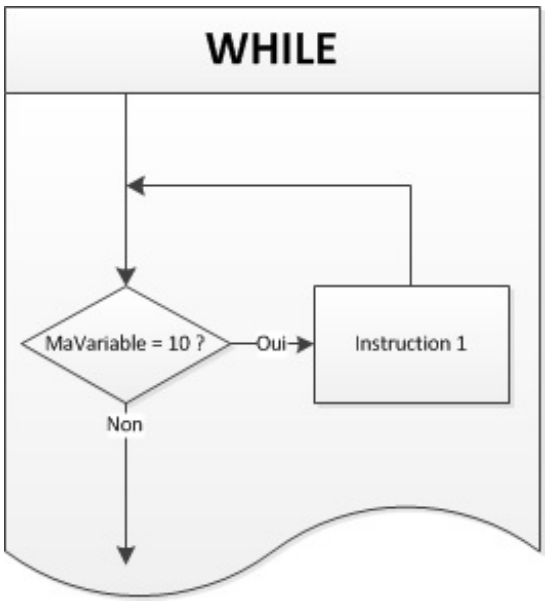
La syntaxe est similaire à celle du *If* de tout à l'heure. Voyons cela !

Code : VB.NET

```
While MaVariable = 10
    'Exécuté tant que MaVariable = 10
End While
```

Code VB	While	MaVariable	= 10
Français	Tant que	« MaVariable »	est égale à 10

Voici donc un schéma pour vous aider à comprendre.



En clair, le programme arrive au niveau de l'instruction *While*, vérifie que la condition est vraie et, si c'est le cas, entre dans le *While* puis exécute les lignes qui se trouvent à l'intérieur ; il arrive ensuite au ***End While*** et retourne au *While*. Cela tant que la condition est vraie.

Tu parlais de booléens...

Eh oui, lorsque vous écrivez `MaVariable = 10` , le programme va faire un petit calcul dans son coin afin de vérifier que la valeur de « MaVariable » est bien égale à 10 ; si c'est le cas, il transforme cela en un booléen de type *Vrai*.
Il s'agit du même principe que pour les autres boucles conditionnelles (*If*, *Else*, etc.).

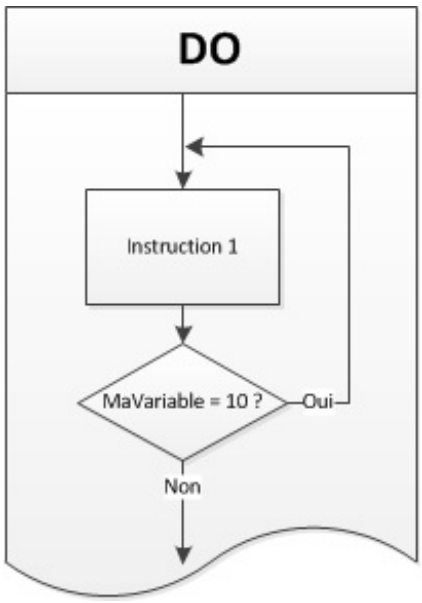
Grosse erreur possible : LES BOUCLES INFINIES !
C'est une erreur qui se produit si la condition ne change pas : le programme tourne dans cette boucle indéfiniment.
Pour y remédier, assurez-vous que la variable peut bien changer.

Mais m'sieur ! Si je veux passer au moins une fois dans la boucle même si la condition est fausse, comment dois-je faire ?

Oh, mais quelle coïncidence, une boucle spéciale existe pour un tel cas ! (C'est beau le hasard, parfois, n'est-ce pas ?)

Do While

À l'instar du *While*, le ***Do While*** (traduisible par « faire tant que ») passe au moins une fois dans la boucle.



Code : VB.NET

```
Do
    'Instruction exécutée au moins une fois
Loop While MaVariable = 10
```

Autre information : il existe un autre mot qui se met à la place de *While*. Ce mot est ***Until***. Il signifie : « passe tant que la condition n'est pas vraie » (le *While* est utilisé seulement tant que la condition est vraie).

Un code de ce type...

Code : VB.NET

```
Do
```

```
    Loop Until MaVariable = 10
```

... revient à écrire ceci :

Code : VB.NET

```
    Do

    Loop While MaVariable <> 10
```

J'espère ne pas avoir été trop brusque... 

Vous êtes désormais en mesure d'utiliser les boucles ***While***, ***If*** et ***Select***. Une dernière pour la route ?

For

« For », mot anglais traduisible par « pour »

For est indissociable de son ***To***, comme un *If* a son *Then* (sauf cas particuliers, tellement particuliers que vous ne les utiliserez pas dans l'immédiat).

Et tel *If*, ***For To*** a un ***Next*** (qui correspond à peu près au *End If*).

Je m'explique. Si je souhaite effectuer une instruction dix fois de suite, je vais écrire ceci :

Code : VB.NET

```
Dim x As Integer = 0

While x <> 10
    'Instruction à exécuter 10 fois
    'Augmente x de 1
    x = x + 1
End While
```

Je profite de cet exemple pour vous signaler que l'incrémentation d'une variable de 1 peut s'écrire `x += 1`. Pas besoin de « = », cette instruction seule remplace `x = x + 1`.
Tant que j'y suis, `x -= 1` remplace `x = x - 1`.

La boucle sera traversée à dix reprises. Eh bien, *For* remplace ce code par celui-ci :

Code : VB.NET

```
Dim x As Integer
For x = 1 to 10
    'Instruction à exécuter 10 fois
Next
```

Les deux codes effectueront la même chose. Le *Next* correspond à « ajoute 1 à ma variable ».

Code VB	For	MaVariable	= 1	To	10
Français	Pour	« MaVariable »	de 1	jusqu'à	10

Petites astuces du *For*...

- On peut déclarer les variables dans la ligne du *For*, de cette manière :

Code : VB.NET

```
For x As Integer = 1 to 10
     'Instruction à exécuter 10 fois
Next
```

Cela reviendra de nouveau au même.
- Si vous voulez ajouter 2 au *Next* à la place de 1 (par défaut) :

Code : VB.NET

```
For x As Integer = 1 to 10 step 2
     'Instruction à exécuter 5 fois
Next
```

Mieux comprendre et utiliser les boucles

Opérateurs

Vous savez maintenant vous servir des grands types de boucles. Rassurez-vous, tout au long du tutoriel, je vous apprendrai d'autres choses en temps voulu.

Je voulais vous donner de petits éclaircissements à propos des boucles. Pour valider la condition d'une boucle, il existe des opérateurs :

Symbole	Fonction
=	Égal
<>	Différent
>	Strictement supérieur
<	Strictement inférieur
=<	Inférieur ou égal
=>	Supérieur ou égal

Grâce ces opérateurs, vous allez déjà pouvoir bien exploiter les boucles.
Comment les utiliser ? C'est très simple.

Si vous voulez exécuter un *While* tant que « x » est plus petit que 10 :

Code : VB.NET

```
While x < 10
```

Voilà !

Explication des boucles

Second élément : une boucle est considérée comme vraie si le booléen correspondant est vrai (souvenez-vous du booléen, un type qui ne peut être que vrai ou faux).

En gros, si j'écris le code suivant :

Code : VB.NET

```
Dim x As Integer = 0
If x = 10 Then
End If
```

... c'est comme si j'écrivais ceci :

Code : VB.NET

```
Dim x As Integer = 0
Dim b As Boolean = false
b = (x = 10)
If b Then
End If
```

Eh oui, quelle découverte ! Si je place un *boolean* dans la condition, il est inutile d'ajouter `If b = true Then` .

J'espère vous avoir éclairés... et non enfoncés ! 

And, or, not

Nous pouvons également utiliser des **mots** dans les boucles !

Non, pas question !

Commençons donc par le mot clé ***not***, dont le rôle est de préciser à la boucle d'attendre l'inverse.
Exemple : un `While not = 10` correspond à un `While <> 10` .

Et puis ?

Un second mot permet d'ordonner à une boucle d'attendre plusieurs conditions : ce cher ami s'appelle ***And***. Il faut que toutes les conditions, reliées par *And*, soient vérifiées.

Code : VB.NET

```
While MaVariable > 0 And MaVariable =< 10
```

Ce code tournera tant que la variable est comprise entre 0 et 10.

Faites attention à rester logiques dans vos conditions :

Code : VB.NET

```
While MaVariable = 0 And MaVariable = 10
```

Le code ci-dessus est totalement impossible, votre condition ne pourra donc jamais être vraie...

Ou bien ?

Le dernier « mot » que je vais vous apprendre pour le moment est ***Or***.
Ce mot permet de signifier « soit une condition, soit l'autre ».

Voici un exemple dans lequel *Or* est impliqué :

Code : VB.NET

```
While MaVariable => 10 Or MaVariable = 0
```

Cette boucle sera exécutée tant que la variable est plus grande ou égale à 10, ou égale à 0.

Ensemble, mes amis !

Eh oui, ces mots peuvent s'additionner, mais attention au sens.

Code : VB.NET

```
While MaVariable > 0 And not MaVariable => 10 Or MaVariable = 15
```

Ce code se comprend mieux avec des parenthèses : `(MaVariable > 0 et non MaVariable => 10) ou MaVariable = 15` .
Donc, cela se traduit par « si MaVariable est comprise entre 1 et 10 ou si elle est égale à 15 ».

J'espère avoir été suffisamment compréhensible.

---

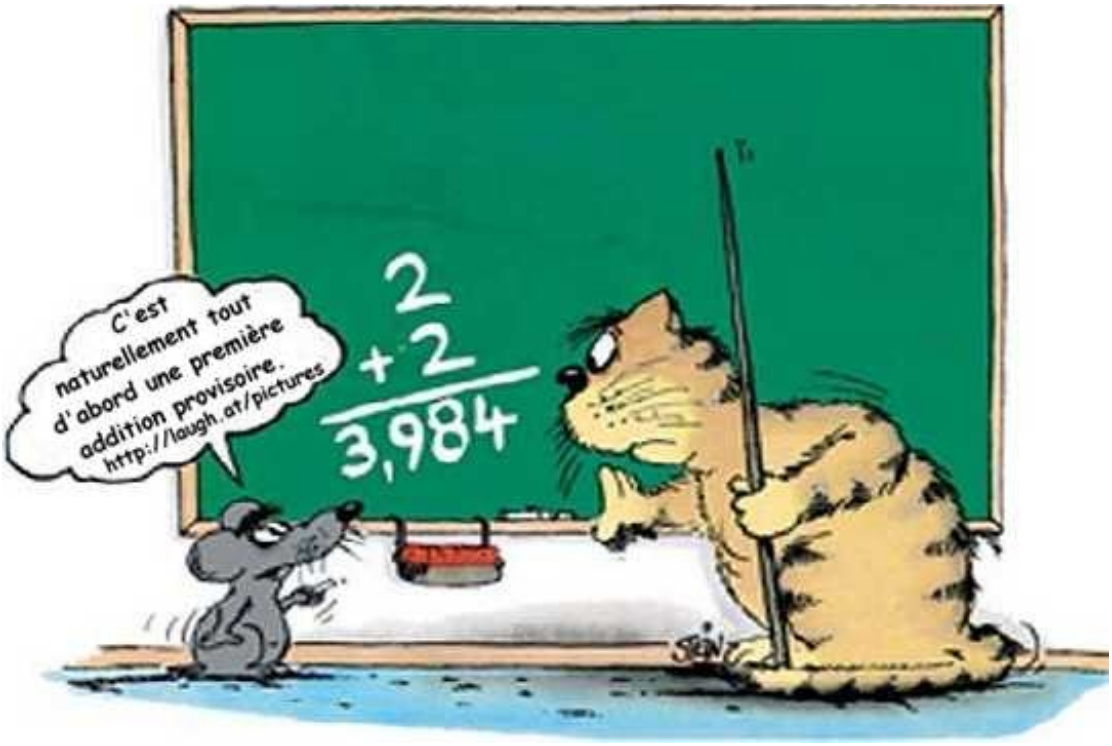
2 TPs d'apprentissage

Nous allons enchaîner avec deux TP. Sachez que pour ces TP, il est absolument inutile de sauter directement à la solution pour se retrouver avec un programme qui fonctionne, mais au final, ne rien comprendre. Je l'ai déjà répété à plusieurs reprises, c'est en pratiquant que l'on progresse.

Essayez donc d'être honnêtes avec vous-mêmes et de chercher comment résoudre le problème que je vous pose, même si vous n'y arriverez peut-être pas du premier coup. J'en profiterai également pour introduire de nouvelles notions, donc pas de panique : on y va doucement.

Addition

On va immédiatement passer à la partie au cours de laquelle ~~je vous laisse comme ça et je vais boire mon café~~ on va mettre la théorie en pratique.



Cahier des charges

Donc, c'est parti : **je veux** (j'ai toujours rêvé de dire ça !) un programme qui m'effectue l'addition de deux nombres demandés au préalable à l'utilisateur. Attention à prévoir le cas où l'utilisateur ne saisirait pas un nombre.

Vous connaissez déjà la marche à suivre pour demander des nombres, les additionner, afficher le résultat (je l'ai déjà indiqué, au cas où vous ne le sauriez pas), mais un problème subsiste : comment vérifier qu'il s'agit bel et bien d'un nombre ?

Code : VB.NET

```
IsNumeric()
```

Il vous faut faire appel à une fonction évoquée précédemment, qui prend en argument une variable (de toute façon, ce sera indiqué lorsque vous le taperez) et renvoie un booléen (*vrai* si cette variable est un nombre, *faux* dans le cas contraire).

Il va donc falloir stocker la valeur que la personne a entrée dans une variable de type *string*.

Et pourquoi pas dans une variable de type *integer* ? C'est bien un nombre, pourtant ?

Eh bien, tout simplement car si la personne entre une lettre, il y aura une erreur : le programme ne peut pas entrer de lettre dans un *integer*, à l'inverse d'un *string*.

Ensuite, vous allez utiliser la fonction ***IsNumeric*** sur cette variable.

Vous aurez sûrement besoin d'utiliser les boucles conditionnelles !

Je vous laisse, alors ?
Bonne chance !

Secret (cliquez pour afficher)

Code : VB.NET

```
Module Module1

    Sub Main()
        'Déclaration des variables
        Dim ValeurEntree As String = ""
        Dim Valeur1 As Double = 0
        Dim Valeur2 As Double = 0

        Console.WriteLine("- Addition de deux nombres -")
        'Récupération du premier nombre
        Do
            Console.WriteLine("Entrez la première valeur")
            ValeurEntree = Console.ReadLine()
            'Tourne tant que ce n'est pas un nombre
        Loop Until IsNumeric(ValeurEntree)
        'Écriture de la valeur dans un double
        Valeur1 = ValeurEntree

        'Récupération du second nombre
        Do
            Console.WriteLine("Entrez la seconde valeur")
            ValeurEntree = Console.ReadLine()
            'Tourne tant que ce n'est pas un nombre
        Loop Until IsNumeric(ValeurEntree)
        'Écriture de la valeur dans un double
        Valeur2 = ValeurEntree

        'Addition
        Console.WriteLine(Valeur1 & " + " & Valeur2 & " = " &
Valeur1 + Valeur2)

        'Pause factice
        Console.Read()
    End Sub

End Module
```

Le résultat

Code : Console

```
- Addition de deux nombres -
Entrez la première valeur
10
Entrez la seconde valeur
k
Entrez la seconde valeur
20
10 + 20 = 30
```

Des questions ?

Oui ! Je voulais utiliser *Do While* et non *Do Until* !

Ah ! Du genre... ceci ?

Code : VB.NET

```
Do
    Console.WriteLine("Entrez la première valeur")
    Valeur1 = Console.ReadLine()
    'Tourne tant que ce n'est pas un nombre
Loop While not IsNumeric(Valeur1)
```

C'est tout à fait juste.

Les variables utilisées sont des *double* pour que l'utilisation de nombres à virgule soit possible dans ces additions.

Tricheur ! Tu ne nous avais pas parlé du *not* !

Voilà l'occasion de vous en parler. Vous devez vous en douter : cela produit l'inverse.
Et de toute façon, je ne veux pas d'excuses : vous aviez *Until* à votre disposition !

Autre chose : pourquoi stockes-tu les résultats dans une autre variable, et n'as-tu pas tout de suite utilisé les mêmes variables ?

À cause des types : avec votre suggestion, il aurait fallu mettre « Valeur1 » et « Valeur2 » en *string*, on est d'accord ? Sauf qu'une addition sur un *string*, autrement dit une chaîne de caractères, même si elle contient un nombre, aura comme effet de « coller » les deux textes. Si vous avez essayé, vous avez dû récupérer un « 1020 » comme résultat, non ? 

Et pourquoi donc utiliser un *Do*, et non un simple *While* ou *If* ?

Parce qu'avec un *If*, si ce n'est pas un nombre, le programme ne le demandera pas plus d'une fois. Un simple *While* aurait en revanche suffi ; il aurait juste fallu initialiser les deux variables sans nombres à l'intérieur. Mais je trouve plus propre d'utiliser les *Do*.

Ne vous inquiétez pas : il s'agissait de votre premier TP, avec de nouveaux concepts à utiliser. Je comprends que cela a pu être difficile, mais vous avez désormais une petite idée de la démarche à adopter la prochaine fois.

Minicalculatrice

Nous venons donc de réaliser un programme qui additionne deux nombres.

Cahier des charges

Au tour maintenant de celui qui effectue toutes les opérations.

Pardon ?

Oui, exactement. Vous êtes tout à fait capables de réaliser ce petit module.

La différence entre les deux applications est simplement un « menu », qui sert à choisir quelle opération effectuer.

Je vous conseille donc la boucle *Select Case* pour réagir en fonction du menu.
Autre chose : pensez à implémenter une fonction qui vérifie que le choix du menu est valide.

Vous avez toutes les clés en main ; les boucles et opérations sont expliquées précédemment.

Bonne chance ! 

Secret (cliquez pour afficher)

Code : VB.NET

```
Module Module1

    Sub Main()
        'Déclaration des variables
        Dim Choix As String = ""
        Dim ValeurEntree As String = ""
        Dim Valeur1 As Double = 0
        Dim Valeur2 As Double = 0

        'Affichage du menu
        Console.WriteLine("- Minicalculatrice -")
        Console.WriteLine("- Opérations possibles -")
        Console.WriteLine("- Addition : 'a' -")
        Console.WriteLine("- Soustraction : 's' -")
        Console.WriteLine("- Multiplication : 'm' -")
        Console.WriteLine("- Division : 'd' -")

        Do
            Console.WriteLine("- Faites votre choix : -")
            'Demande de l'opération
            Choix = Console.ReadLine()
            'Répète l'opération tant que le choix n'est pas valide
        Loop Until Choix = "a" Or Choix = "s" Or Choix = "m" Or
Choix = "d"

        'Récupération du premier nombre
        Do
            Console.WriteLine("Entrez la première valeur")
            ValeurEntree = Console.ReadLine()
            'Tourne tant que ce n'est pas un nombre
        Loop Until IsNumeric(ValeurEntree)
        'Écriture de la valeur dans un double
        Valeur1 = ValeurEntree

        'Récupération du second nombre
        Do
            Console.WriteLine("Entrez la seconde valeur")
```

```
                ValeurEntree = Console.ReadLine()
                'Tourne tant que ce n'est pas un nombre
            Loop Until IsNumeric(ValeurEntree)
            'Écriture de la valeur dans un double
            Valeur2 = ValeurEntree

            Select Case Choix
                Case "a"
                    'Addition
                    Console.WriteLine(Valeur1 & " + " & Valeur2 & " =
    " & Valeur1 + Valeur2)
                Case "s"
                    'Soustraction
                    Console.WriteLine(Valeur1 & " - " & Valeur2 & " =
    " & Valeur1 - Valeur2)
                Case "m"
                    'Multiplication
                    Console.WriteLine(Valeur1 & " x " & Valeur2 & " =
    " & Valeur1 * Valeur2)
                Case "d"
                    'Division
                    Console.WriteLine(Valeur1 & " / " & Valeur2 & " =
    ")
                    Console.WriteLine("Valeur exacte : " & Valeur1 /
    Valeur2)
                    Console.WriteLine("Résultat entier : " & Valeur1 \
    Valeur2)
                    Console.WriteLine("Reste : " & Valeur1 Mod
    Valeur2)
            End Select

            'Pause factice
            Console.Read()
        End Sub

    End Module
```

J'ai choisi de faire appel à une méthode plutôt fastidieuse. En effet, dans la ligne `Loop Until Choix = "a" Or Choix = "s" Or Choix = "m" Or Choix = "d"` , j'ai réécrit toutes les valeurs possibles du menu, mais imaginez-vous dans le cas d'un menu de vingt choix...

Dans cette situation, l'astuce serait d'utiliser un menu à numéros et, carrément, d'exclure une plage avec un nombre supérieur à 10, par exemple.

Voici ce que j'obtiens lorsque je lance le programme :

Code : Console

```
-   Minicalculatrice   -
-  Opérations possibles -
-     Addition : 'a'    -
-   Soustraction : 's'  -
-  Multiplication : 'm' -
-     Division : 'd'    -
- Faites votre choix :  -
y
- Faites votre choix :  -
d
Entrez la première valeur
255
Entrez la seconde valeur
12m
Entrez la seconde valeur
36
255 / 36 =
```

```
Valeur exacte : 7,08333333333333
Résultat entier : 7
Reste : 3
```

Euh... m'sieur ! Pour ma part, j'utilise une variable intermédiaire, et je n'effectue pas directement l'opération dans le *WriteLine* ; mais dans le cas de la division, les résultats ne sont pas toujours justes... Pourquoi ?

Vous avez sûrement dû déclarer votre « variable intermédiaire » en *integer*.
Si c'est le cas, je vous explique le problème : le *integer* ne sert pas à stocker des nombres à virgule. Essayez de placer cette variable en *double* pour vérifier.

Idem pour les autres variables : si l'utilisateur veut additionner deux nombres à virgule, cela n'ira pas !

Alors, pas sorcier pour le reste ? Du moins, je l'espère.

Allez, on passe à la suite ! 

Jouer avec les mots, les dates

Vous êtes en mesure d'effectuer des opérations sur les chiffres et les nombres. Mais ne serait-il pas rigolo de faire de même avec les mots ?

Les chaînes de caractères

Remplacer des caractères

On va commencer par la fonction la plus simple : le ***Replace*** qui, comme son nom l'indique, permet de remplacer des caractères ou groupes de caractères au sein d'une chaîne.

La syntaxe est la suivante :

Code : VB.NET

```
Dim MonString As String = "Phrase de test"
MonString = MonString.Replace("test",
"test2")
```

Le premier argument de cette fonction est le caractère (ou mot) à trouver, et le second, le caractère (ou mot) par lequel le remplacer.
Dans cette phrase, le code remplacera le mot « test » par « test2 ».

Si vous avez bien assimilé le principe des fonctions, des variables peuvent être utilisées à la place des chaînes de caractères en « dur ».

Mettre en majuscules

La fonction ***ToUpper*** se rattachant à la chaîne de caractères (considérée comme un objet) en question permet cette conversion. Elle s'utilise comme suit :

Code : VB.NET

```
Dim MonString As String = "Phrase de test"
MonString = MonString.ToUpper()
```

Cette phrase sera donc mise en MAJUSCULES.

Mettre en minuscules

Cette fonction s'appelle ***ToLower*** ; elle effectue la même chose que la précédente, sauf qu'elle permet le formatage du texte en minuscules.

Code : VB.NET

```
Dim MonString As String = "Phrase de test"
MonString = MonString.ToLower()
```

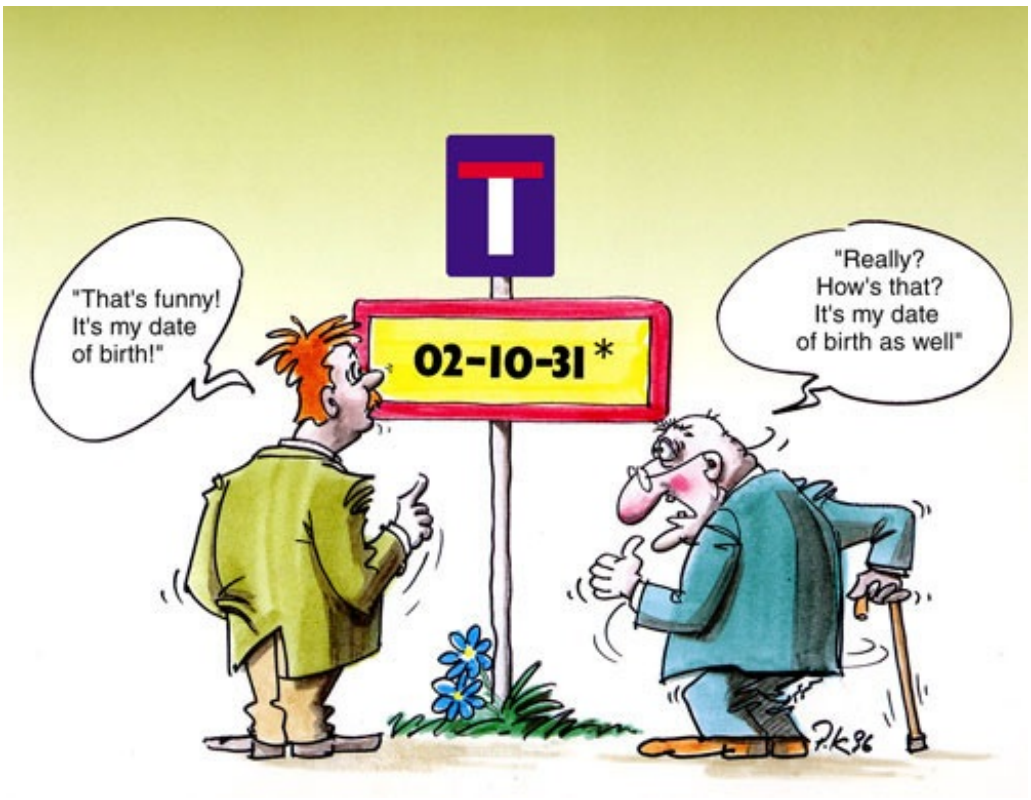
Ces petites fonctions pourront sûrement nous être utiles pour l'un de nos TP.

Les dates, le temps

Nous passons donc aux dates et à l'heure. Il s'agit d'un sujet assez sensible puisque, lorsque nous aborderons les bases de données, la syntaxe d'une date et son heure en base de données sera différente de la syntaxe lisible par tout bon francophone (âgé de plus de deux ans).

Tout d'abord, pour travailler, nous allons avoir besoin d'une date. Ça vous dirait, la date et l'heure d'aujourd'hui ?

Si cela ne vous dérange pas, nous allons utiliser l'instruction `Date.Now` , qui nous donne... la date et l'heure d'aujourd'hui, sous la forme suivante :



Code : Console

```
16/06/2009 21:06:
```

La première partie est la date ; la seconde, l'heure.

Nous allons ainsi pouvoir travailler. Entrons cette valeur dans une variable de type... ***date***, et amusons-nous !

Récupérer uniquement la date

La première fonction que je vais vous présenter dans ce chapitre est celle qui convertit une chaîne *date*, comme celle que je viens de vous présenter, mais uniquement dans sa partie « date ».

Je m'explique : au lieu de « 16/06/2009 21:06:33 » (oui, je sais, il est exactement la même heure qu'il y a deux minutes...), nous obtiendrons « 16/06/2009 ».

Code : VB.NET

```
ToShortDateString()
```

Cette fonction s'utilise sur une variable de type *date*. J'ignore si vous vous souvenez de mon petit interlude sur les objets et fonctions, au cours duquel je vous ai expliqué que le point (« . ») servait à « affiner la recherche ». Nous allons donc utiliser ce point pour lier le type (qui est également un objet dans notre cas) et cette fonction.

Cette syntaxe que vous avez, je pense, déjà écrite vous-mêmes (`MaVariableDate.ToShortDateString()`), convertit votre date en date sans heure, mais flotte dans les airs... Il faut bien la récupérer, non ? Pour ce faire, affichons-la !

Pour ma part, je me retrouve avec ceci :

Code : VB.NET

```
Module Module1

    Sub Main()
        'Initialisation des variables
```

```
            Dim MaVariableDate As Date = Date.Now
            'Écriture de la forme courte de la date
            Console.WriteLine(MaVariableDate.ToShortDateString)
            'Pause
            Console.Read()
        End Sub

    End Module
```

Voici le résultat qui est censé s'afficher dans la console :

Code : Console

```
16/06/2009
```

Hep, m'sieur ! Je ne comprends pas : j'ai stocké le résultat dans une variable intermédiaire de type *date* et je n'obtiens pas la même chose que toi !

Ah, l'erreur ! La variable de type *date* est formatée obligatoirement comme je l'ai montré au début, ce qui veut dire que si vous y entrez par exemple uniquement une heure, elle affichera automatiquement une date.

Comment dois-je faire, dans ce cas ?

Bah, pourquoi ne pas mettre cela dans un *string* ? (Vous n'aimez pas les *string* ? )

La date avec les libellés

Seconde fonction : récupérer la date avec le jour et le mois écrits en toutes lettres. Rappelez-vous l'école primaire où l'on écrivait chaque matin « mardi 16 juin 2009 » (non, je n'ai jamais écrit cette date à l'école primaire !).

Donc, pour obtenir cela, notre fonction s'intitule `ToLongDateString()` (je n'ai pas trop cherché ).

Le résultat obtenu est `Mardi 16 Juin 2009`.

L'heure uniquement

Voici la fonction qui sert à récupérer uniquement l'heure :

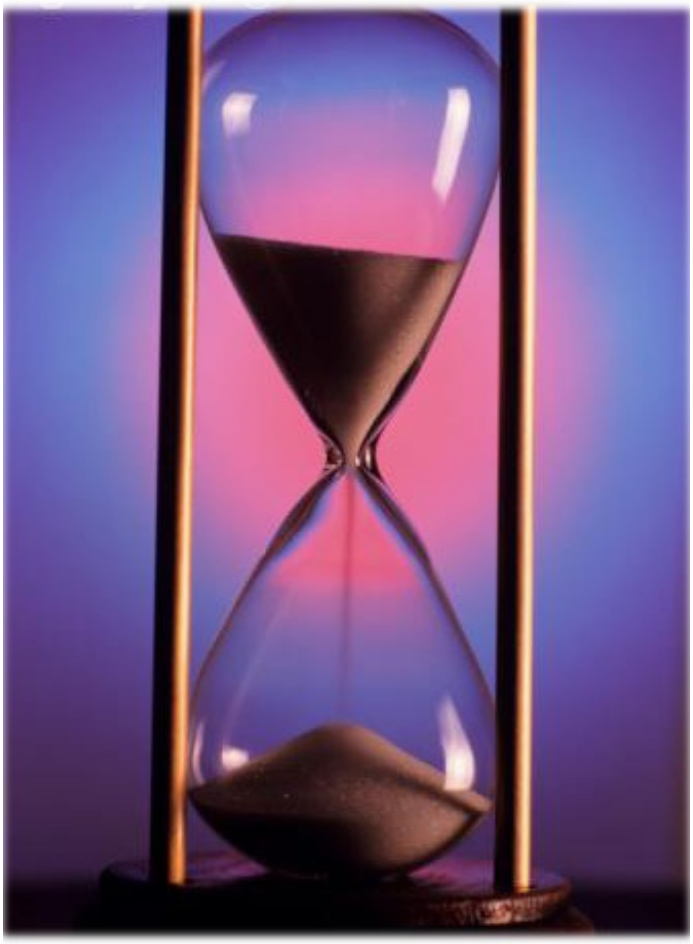
Code : VB.NET

```
ToShortTimeString()
```

Ce qui nous renvoie ceci :

Code : Console

```
21:06
```



L'heure avec les secondes

Même chose qu'au-dessus, sauf que la fonction se nomme :

Code : VB.NET

```
ToLongTimeString()
```

Cela nous renvoie :

Code : Console

```
21:06:33
```

TP sur les heures

Sur ce, j'espère que vous avez bien compris comment manipuler les dates, les heures et les chaînes de caractères.

Nous allons faire un mini-TP.

L'horloge

Eh oui, je ne suis pas allé chercher bien loin : ce TP aura pour but de mettre en œuvre une *horloge* ~~parlante~~ (heures:minutes:secondes).

Heureusement que vous avez lu ce qui est indiqué au-dessus, car vous aurez besoin de la fonction qui renvoie seulement les heures, les minutes et les secondes.

Je ne vais pas vous mâcher tout le travail, parce que vous devenez grands (dans le domaine du Visual Basic, en tous cas !) ; je vais donc me contenter de vous énumérer les fonctions dont vous aurez besoin pour mener à bien votre travail.

La première fonction consiste à mettre en « pause » le programme pendant une durée passée en argument. Attention : cette valeur s'exprime en *millisecondes*.

Code : VB.NET

```
System.Threading.Thread.Sleep()
```

La seconde servira à effacer l'écran de la console ; si vous avez déjà fait du **Bash**, c'est pareil :

Code : VB.NET

```
Console.Clear()
```

Avec ces deux fonctions et les connaissances du reste du chapitre, vous devez être capables de réaliser cette horloge.

Non, je ne vous aiderai pas plus !
N'insistez pas !

Vous avez terminé ? Bon, dans ce cas, faisons le compte rendu de ce que nous venons de coder.

Secret (cliquez pour afficher)

Code : VB.NET

```
Module Module1
    Sub Main()
        'Initialisation des variables
        Dim MaVariableDate As Date

        'Boucle infinie /!\
        While 1
            'Récupération de la date actuelle
            MaVariableDate = Date.Now
            'Affichage des heures, minutes, secondes
            Console.WriteLine("----------------")
            Console.WriteLine("--- " &
```

```
        MaVariableDate.ToLongTimeString & " ---")
                    Console.WriteLine("----------------")
                    'Pause de 1 seconde
                    System.Threading.Thread.Sleep(1000)
                    'Efface l'écran de la console
                    Console.Clear()
                End While
            End Sub
        End Module
```

Je vais vous expliquer mon code.

J'ai tout d'abord déclaré une variable de type *date*, ce que vous avez également dû faire. Ensuite, j'ai créé une boucle infinie avec un *While 1*.

Ce n'est pas bien ! C'était nécessaire dans ce cas pour réaliser un TP simple, et parce que le programme n'était censé faire que cela.

Le programme tourne donc jusqu'à ce qu'on l'arrête dans cette boucle.

Pourquoi *While 1* ?

Parce que le « 1 » est toujours vrai, cela signifie donc : « Tourne, mon grand, ne t'arrête pas ! »

Au début de cette boucle, je récupère la date actuelle et l'écris dans ma variable.
J'affiche cette variable en utilisant la fonction permettant d'en extraire l'heure, les minutes et les secondes.

Je fais une pause d'une seconde (`1000 ms = 1 s`).

J'efface ensuite l'écran, puis je recommence, et ainsi de suite.

On obtient donc ceci :

Code : Console

```
----------------
--- 21:10:11 ---
----------------
```

Notez que vous n'êtes pas obligés de saisir des petits tirets comme je l'ai fait.

Euh... pourquoi n'as-tu pas mis par exemple « 100 ms », pour que ce soit plus précis ?

Parce que, que j'utilise « 100 ms », « 1 s » ou même « 1 ms », cela aura la même précision. L'horloge change à chaque seconde, pourquoi donc aller plus vite qu'elle ?

Simplification du code

Pourquoi passer par une variable ? Pourquoi ne pas entrer l'instruction qui récupère l'heure actuelle et la formater en une seule ligne ?

Code : VB.NET

```
Module Module1
    Sub Main()
```

```
            'Boucle infinie /!\
            While 1
                'Affichage des heures, minutes, secondes
                Console.WriteLine(Date.Now.ToLongTimeString)
                'Pause de 1 seconde
                System.Threading.Thread.Sleep(1000)
                'Efface l'écran de la console
                Console.Clear()
            End While
        End Sub
    End Module
```

Voilà mon exemple de simplification du code. Je vous l'avais bien dit : les programmeurs sont fainéants !

Mais, tu ne nous avais pas expliqué ce raccourci !

Tatata, je l'ai expliqué lorsque j'ai parlé des fonctions et des objets. Il n'y a pas de limite d'objets que l'on peut relier, on a donc le droit de faire ça.

Attention : c'est plus simple mais pas toujours plus clair.

Les Tableaux

Vous avez acquis pas mal de connaissances en VB jusqu'à présent : vous savez notamment utiliser différents types de variables et interagir avec elles. Mais attention, après les variables, voici, Mesdames, Mesdemoiselles et Messieurs, les tableaux de variables !

Qu'est-ce qu'un tableau ?

Bah, quelle question ! C'est ceci, voyons.

Ah, vous voulez parler d'un tableau en Visual Basic ? Excusez-moi ! Je vais essayer de vous le décrire simplement.

Tableaux — Généralités

Donc, en gros, en quoi un tableau va-t-il nous être utile ?

Il va généralement nous servir à stocker plusieurs valeurs ; s'il s'agit seulement d'entrer un nombre à l'intérieur, cela ne sert à rien.
Par exemple, dans une boucle qui récupère des valeurs, si on demande dix valeurs et qu'on fait une boucle, on saisit les valeurs dans un tableau.

D'accord. Et concrètement, ça ressemble à quoi ?

On dirait mes questions !

Trêve de plaisanterie, voici un schéma de tableau à **une dimension**.

Les dimensions

Tableau à une dimension

Comme vous le voyez, c'est exactement comme sous Excel.

Pour déclarer un tableau de ce type en Visual Basic, c'est très simple : on écrit notre déclaration de variable, d'*integer* par exemple, et on place l'*index* du tableau entre parenthèses. Voici le code source de l'exemple que je viens de vous montrer :

Code : VB.NET

```
Dim MonTableau(4) As Integer
```

Voilà mon tableau !

Comme sur le dessin, tu disais ? Pourtant, sur ce dernier, il y a cinq cases. Or, tu n'en as inscrit que quatre. Comment cela se fait-il ?

Oui, sa longueur est de 4. Vous devez savoir qu'**un tableau commence toujours par 0**. Et donc, si vous avez compris, un tableau de quatre cases possède les cases 0, 1, 2, 3 et 4, soit cinq cases, comme sur le dessin.

Le nombre de cases d'un tableau est toujours « **indice + 1** ».
Réciproquement, l'index de sa dernière case est « **taille - 1** ».

Souvenez-vous de cela, ce sera utile par la suite.

Comment écrire dans un tableau ?
C'est très simple. Vous avez par exemple votre tableau de cinq cases (dimension 4) ; pour écrire dans la case 0 (soit, la première case), on écrit ceci :

Code : VB.NET

```
MonTableau(0) = 10
```

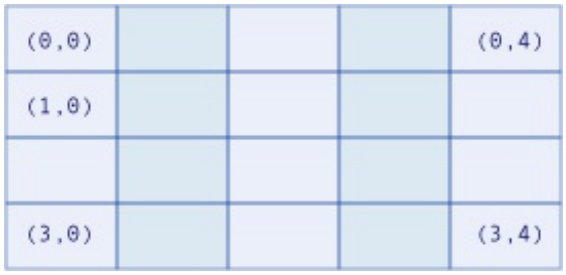
Eh oui, il s'utilise comme une simple variable ! Il suffit juste de mettre la case dans laquelle écrire, accolée à la variable et entre parenthèses.

Mais... c'est comme une fonction, je vais tout mélanger !

Eh bien, effectivement, la syntaxe est la même que la fonction ; le logiciel de développement vous donnera des indications lorsque vous allez écrire la ligne, pour que vous évitiez de confondre fonctions et tableaux.

Si vous comprenez aussi vite, passons au point suivant.

Tableaux à deux dimensions



Cette image représente un tableau à deux dimensions : une pour la hauteur, une autre pour la largeur.

Pour créer ce type de tableau, le code est presque identique :

Code : VB.NET

```
Dim MonTableau(3,4) As Integer
```

Cela créera un tableau avec quatre lignes et cinq colonnes, comme sur le schéma.

Pour ce qui est de le remplir, le schéma l'explique déjà très bien :

Code : VB.NET

```
MonTableau(0,0) = 10
```

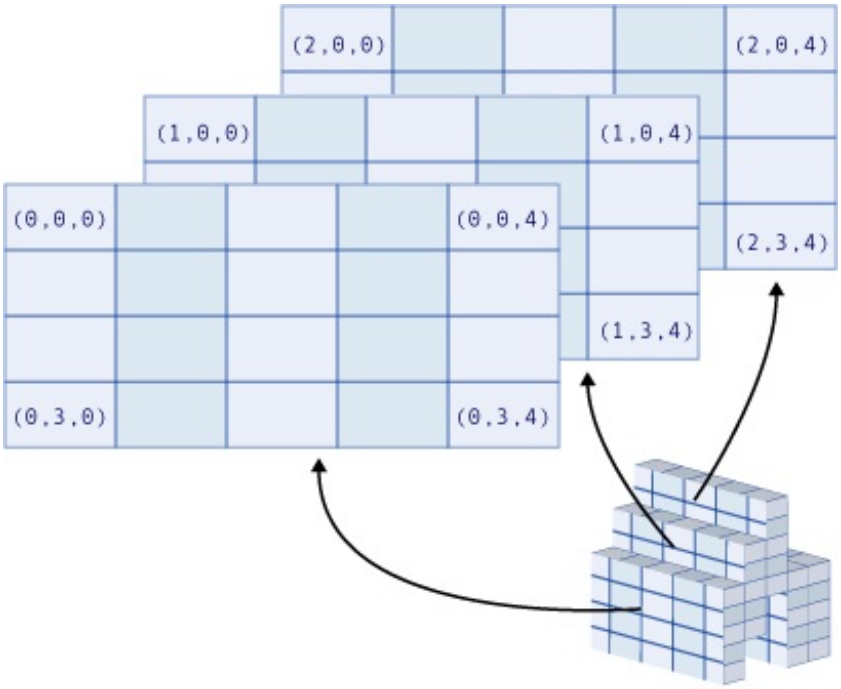
Cela attribuera « 10 » à la case en haut à gauche.

Il est bien important que vous vous représentiez mentalement de manière visuelle la façon dont laquelle est construit un tableau à plusieurs dimensions, pour comprendre comment agir.

Un tableau à deux dimensions peut servir comme tableau à doubles entrées, par exemple.

Tableaux à trois dimensions

Une fois n'est pas coutume : encore un schéma.



Comme vous le voyez, ce type de tableau est représentable par un « cube ».

Il peut être utile lors de représentations tridimensionnelles (du moins, je ne vois que cette utilisation).

Pour le déclarer, je pense que vous avez compris la marche à suivre.

Code : VB.NET

```
Dim MonTableau(2,3,4) As Integer
```

Soit un tableau de trois cases de profondeur (le premier nombre), de quatre lignes (le second nombre) et de cinq colonnes (le dernier nombre).

Idem pour lui attribuer une valeur :

Code : VB.NET

```
MonTableau(2,3,4) = 10
```

Voilà comment mettre « 10 » dans le coin inférieur droit et au fond du « cube » (pas vraiment cubique, d'ailleurs !).

Plus...

Bien qu'on puisse aller jusqu'à une trentaine de dimensions, les tableaux supérieurs à trois dimensions sont rarement utilisés. Si vous voulez stocker plus d'informations, je vous conseille de créer un tableau de tableau à dimensions (cela devient compliqué !).

Code : VB.NET

```
Dim MonTableau(1)(2,3,4) As Integer
```

Ce code crée deux tableaux de tableau à trois dimensions.

Pareil pour y accéder :

Code : VB.NET

```
MonTableau(0)(2,3,4) = 10
```

Je pense que vous avez compris comment les déclarer et les utiliser sommairement. Ça tombe bien : on continue !

Mini-TP : comptage dans un tableau.

Bon, on va pratiquer un peu : je vais vous donner un petit exercice. Tout d'abord, récupérez le code suivant :

Code : VB.NET

```
Dim MonTableau(50) As Integer
For i As Integer = 0 To MonTableau.Length - 1
     MonTableau(i) = Rnd(1) * 10
Next
```

J'explique sommairement ce code : il crée un tableau de 51 cases, de 0 à 50.
Il remplit chaque case avec un nombre aléatoire allant de 0 à 10.

En passant, vous pourrez vous servir assez souvent de la ligne `For i As Integer = 0 To MonTableau.Length - 1` puisqu'elle est, je dirais, universelle ; c'est d'ailleurs ce qu'il faut faire le plus possible dans votre code. Pourquoi universelle ? Parce que, si vous changez la taille du tableau, les cases seront toujours toutes parcourues.

L'instruction `MonTableau.Length` renvoie la taille du tableau, je lui retire 1 car le tableau va de 0 à 50, et la taille est de 51 (comme je l'ai expliqué plus haut).

Je veux donc que vous me comptiez ce tableau si fièrement légué !

Eh bien oui, je suis exigeant : je veux connaître le nombre de 0, de 1, etc.

Au travail, vous connaissez tout ce qu'il faut !

Secret (cliquez pour afficher)

Code : VB.NET

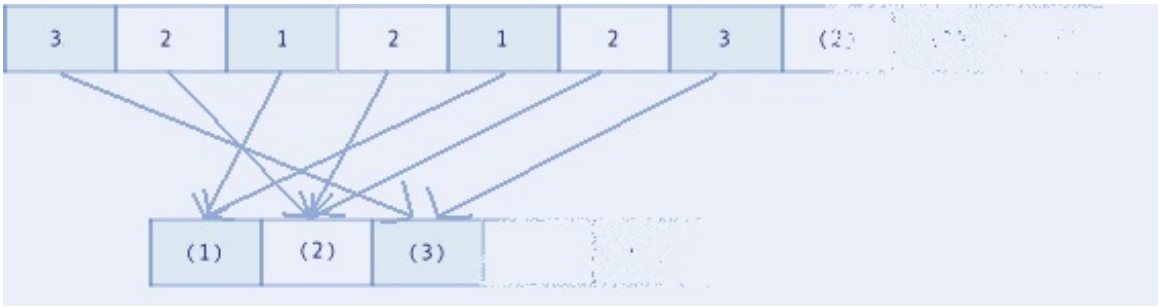
```
Module Module1

    Sub Main()
        'Initialisation des variables
        Dim MonTableau(50), Nombres(10), NumeroTrouve As Integer

        'Remplissage du tableau de nombres aléatoires
        For i As Integer = 0 To MonTableau.Length - 1
            MonTableau(i) = Rnd(1) * 10
        Next

        'Initialisation du tableau "Nombres" avec des 0
        For i = 0 To Nombres.Length - 1
            Nombres(i) = 0
        Next

        'Comptage
        For i = 0 To MonTableau.Length - 1
            'Entre la valeur trouvée dans une variable
intermédiaire
            NumeroTrouve = MonTableau(i)
            'Ajoute 1 à la case correspondant au numéro
            Nombres(NumeroTrouve) = Nombres(NumeroTrouve) + 1
        Next

        'Affichage des résultats
        For i = 0 To Nombres.Length - 1
            Console.WriteLine("Nombre de " & i & " trouvés : " &
Nombres(i))
        Next

        'Pause
```

```
            Console.Read()
        End Sub

    End Module
```

J'espère que vous avez réussi par vous-mêmes. Sachez que ce n'est pas grave si votre programme n'est pas optimisé, ou très long... ce n'est que le début !

J'explique donc ce que je viens de faire.

J'ai créé un tableau de onze cases appelé « **Nombres** » que j'ai initialisé avec des « 0 ».
Dans la boucle de comptage, je récupère le numéro présent dans la case actuelle et l'utilise comme indice de mon tableau **Nombres**.



En gros, si c'est un « 4 » qui est présent dans le tableau, il se positionne sur la quatrième case de **Nombres**, après quoi il ajoute « 1 » à cette valeur.

Enfin, j'affiche les résultats.

Petite remarque :

Code : VB.NET

```
NumeroTrouve = MonTableau(i)
Nombres(NumeroTrouve) = Nombres(NumeroTrouve) + 1
```

Sachez que le code ci-dessus peut se résumer en une ligne :

Code : VB.NET

```
Nombres(MonTableau(i)) = Nombres(MonTableau(i)) + 1
```

Mais attention : ne soyez pas radins sur les variables, cela devient très vite une usine à gaz dès que vous simplifiez tout, surtout lors de l'apprentissage ! Pensez également à toujours bien commenter vos codes.

Les résultats des tests sont les suivants :

Code : Console

```
Nombre de 0 trouvés : 4
Nombre de 1 trouvés : 5
Nombre de 2 trouvés : 1
Nombre de 3 trouvés : 2
Nombre de 4 trouvés : 4
Nombre de 5 trouvés : 4
```

```
Nombre de 6 trouvés : 5
Nombre de 7 trouvés : 5
Nombre de 8 trouvés : 1
Nombre de 9 trouvés : 3
Nombre de 10 trouvés : 6
```

Le fait d'avoir utilisé des `.Length` à chaque reprise me permet de changer uniquement la taille du tableau dans la déclaration pour que le comptage s'effectue sur un plus grand tableau.

Exercice : tri

Un petit exercice : le tri. Je vais vous montrer comment faire, parce que certains d'entre vous vont rapidement être perdus.

Nous allons utiliser le tri à bulles. Pour en apprendre plus concernant l'algorithme de ce tri, lisez le tutoriel rédigé par ShareMan en cliquant ici.

Je vais vous énumérer tout ce qu'il faut faire en français, ce que l'on appelle également un algorithme (un algorithme étant une séquence à accomplir pour arriver à un résultat souhaité).

1. Créer un booléen qui deviendra vrai uniquement lorsque le tri sera bon.
2. Créer une boucle parcourue tant que le booléen n'est pas vrai.
3. Parcourir le tableau ; si la valeur de la cellule qui suit est inférieure à celle examinée actuellement, les inverser.

J'ai expliqué ce qu'il fallait que vous fassiez en suivant le tutoriel du tri à bulles.

Le présent exercice demande un peu plus de réflexion que les autres, mais essayez tout de même.

Code : VB.NET

```
Module Module1

    Sub Main()
        'Initialisation des variables
        Dim MonTableau(20), Intermediaire, TailleTableau As Integer
        Dim EnOrdre As Boolean = False

        'Remplissage du tableau de nombres aléatoires
        For i As Integer = 0 To MonTableau.Length - 1
            MonTableau(i) = Rnd(1) * 10
        Next

        'Tri à bulles
        TailleTableau = MonTableau.Length
        While Not EnOrdre
            EnOrdre = True
            For i = 0 To TailleTableau - 2
                If MonTableau(i) > MonTableau(i + 1) Then
                    Intermediaire = MonTableau(i)
                    MonTableau(i) = MonTableau(i + 1)
                    MonTableau(i + 1) = Intermediaire
                    EnOrdre = False
                End If
            Next
            TailleTableau = TailleTableau - 1
        End While

        'Affichage des résultats
        For i = 0 To MonTableau.Length - 1
            Console.Write(" " & MonTableau(i))
        Next

        'Pause
        Console.Read()

    End Sub

End Module
```

Voilà donc mon code, que j'explique : le début, vous le connaissez, je crée un tableau avec des nombres aléatoires. J'effectue ensuite le tri à bulles en suivant l'algorithme donné. Enfin, j'affiche le tout !

Le résultat est le suivant :

Code : Console

```
0 0 0 1 2 2 2 3 3 5 5 5 5 6 7 8 8 9 9 10 10
```

Pourquoi as-tu mis `TailleTableau - 2` et pas que `- 1` ?

Parce que j'effectue un test sur la case située à la taille du tableau + 1. Ce qui signifie que si je vais jusqu'à la dernière case du tableau, ce test sur la dernière case + 1 tombera dans le **NÉANT** ; et là, c'est le drame : erreur et tout ce qui va avec (souffrance, douleur et apocalypse). 

J'espère que ce petit exercice vous a quand même éclairés concernant les tableaux !

Les fonctions

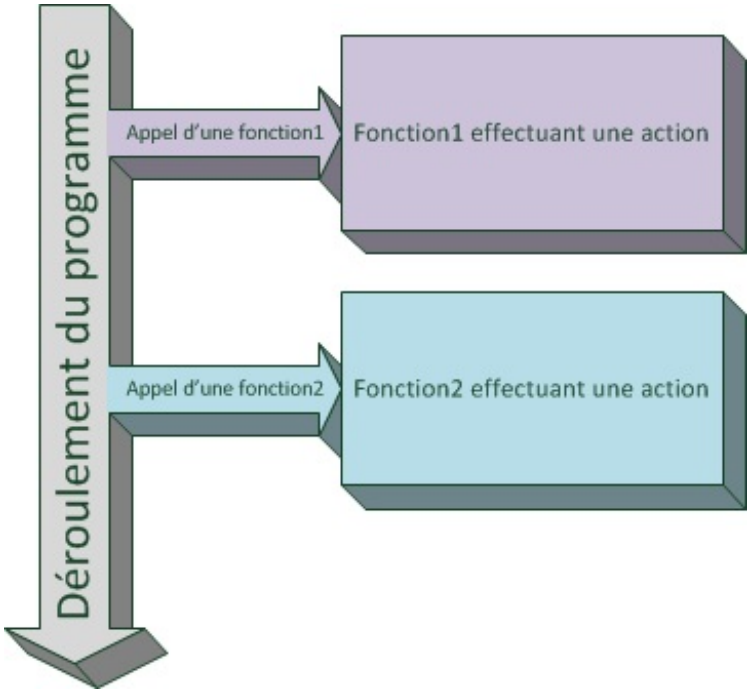
Eh bien, tant de choses à apprendre.
Attaquons-nous tout de suite à une notion importante et dont vous n'allez plus pouvoir vous passer : les fonctions !

Ça se mange ?

Alors oui, bonne question : à quoi vont nous servir les fonctions ? Tout d'abord, sachez que vous en utilisez déjà : je vous les ai expliquées, en plus ! Concrètement, une fonction répète une action bien précise.

Les fonctions que nous connaissons déjà sont, par exemple, le *BEEP* : il remplit la fonction de faire *bip* (sans blague !).
Il y a aussi la petite fonction *IsNumeric()*, qui vérifie que la valeur d'une variable est bien un nombre.

Tout ça pour dire que ces fonctions, des programmeurs les ont déjà créées et les ont intégrées dans les librairies, d'énormes fichiers qui les rassemblent toutes, que vous possédez sur votre ordinateur dès lors que vous avez installé Visual Basic Express. Nous allons donc à notre tour programmer une fonction et apprendre à l'utiliser.



En résumé, la fonction effectue une tâche bien précise.

Créons notre première fonction !

Nous allons donc créer notre première fonction, la plus basique qui soit : sans argument, sans retour.

Cette fonction, on va tout de même lui faire faire quelque chose... Pourquoi, par exemple, ne pas additionner deux nombres que l'on saisit à l'écran ?

Vous vous rappelez certainement le TP avec l'addition. Eh bien, on va factoriser l'addition avec la demande des nombres (*factoriser* signifie mettre sous forme de fonction).

Code : VB.NET

```
Module Module1

    Sub Main()
        Addition()
    End Sub

    Sub Addition()
        Dim ValeurEntree As String = ""
        Dim Valeur1 As Integer = 0
        Dim Valeur2 As Integer = 0

        Console.WriteLine("- Addition de deux nombres -")
        'Récupération du premier nombre
        Do
            Console.WriteLine("Entrez la première valeur")
            ValeurEntree = Console.ReadLine()
            'Tourne tant que ce n'est pas un nombre
        Loop Until IsNumeric(ValeurEntree)
        'Écriture de la valeur dans un integer
        Valeur1 = ValeurEntree

        'Récupération du second nombre
        Do
            Console.WriteLine("Entrez la seconde valeur")
            ValeurEntree = Console.ReadLine()
            'Tourne tant que ce n'est pas un nombre
        Loop Until IsNumeric(ValeurEntree)
        'Écriture de la valeur dans un integer
        Valeur2 = ValeurEntree

        'Addition
        Console.WriteLine(Valeur1 & " + " & Valeur2 & " = " &
Valeur1 + Valeur2)

        'Pause factice
        Console.Read()
    End Sub
End Module
```

Il n'y a plus rien dans le *main* : cela ne va plus marcher !

Bien au contraire : j'ai placé dans le *main* l'**appel de la fonction**.

Lorsque vous créez une fonction, ce n'est pas comme le *main*, elle ne se lance pas toute seule ; si je n'avais pas ajouté cet appel, le programme n'aurait rien fait !

La ligne `Addition()` appelle donc la fonction, mais pourquoi ?
Avez-vous remarqué le *Sub* que j'ai placé en dessous du *main* ?

Un *Sub*, contrairement à une *Function* (que nous verrons par après) ne renvoie aucune valeur.

C'est pour cela que j'ai écrit `Sub Addition()` , et non `Function Addition()` (*Function* étant le mot-clé déclarant une fonction).

Et donc, dans ce *Sub*, j'ai copié-collé exactement le code de notre TP sur l'addition.

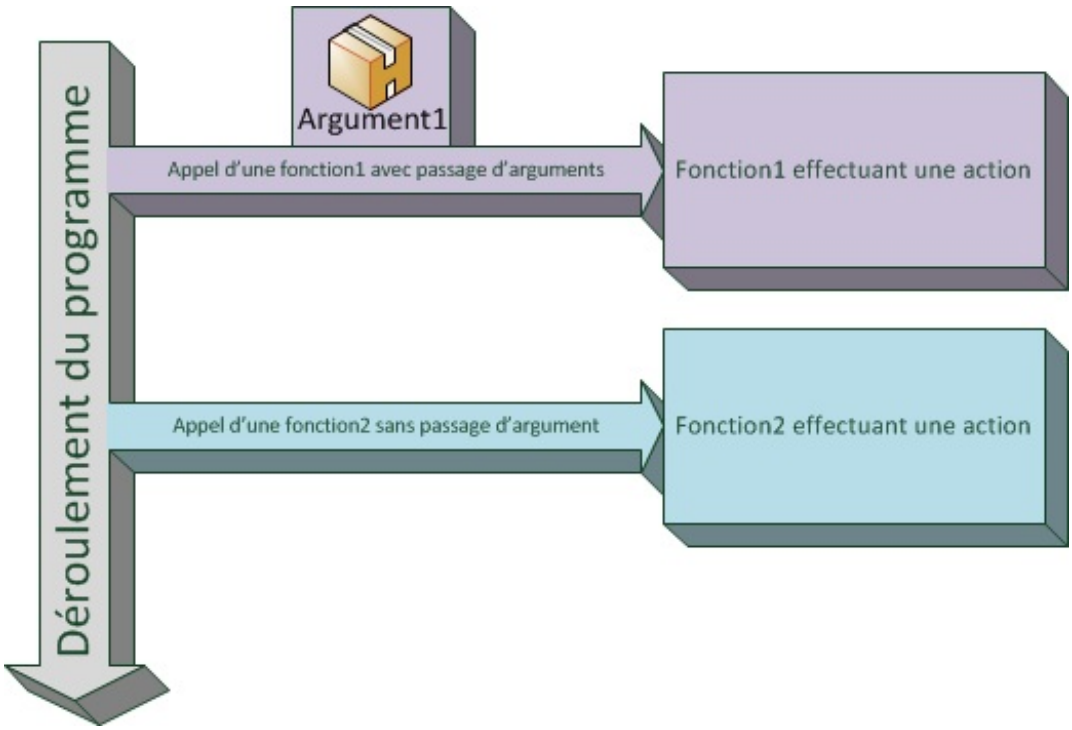
Vous pouvez tester : ça... fonctionne !

Ajout d'arguments et de valeur de retour

Vous avez vu comment créer une fonction que je qualifierais d'« indépendante », ce qui signifie que cette dernière n'influera pas sur le reste du programme, et exécutera toujours la même chose : demander des valeurs, les additionner et afficher le résultat.

Cependant, ce style de raisonnement va vite nous limiter. Heureusement, les créateurs des langages de programmation ont pensé à un truc génial : les arguments et le retour.

Les arguments



Vous savez déjà ce qu'est un argument ; je vous l'ai dit : par exemple, dans la fonction *Write()*, l'argument est la valeur placée entre parenthèses, et cette fonction effectue « Affiche-moi cette valeur ! ».

Vous l'avez donc sûrement compris, les arguments sont mis entre parenthèses... « les », oui, exactement, parce qu'il peut y en avoir plusieurs ! Et vous l'avez déjà remarqué : lorsque nous avons étudié le *BEEP*, les arguments étaient la fréquence et la durée. Et tous deux séparés par... une virgule (« , ») !

Dans une fonction, les différents arguments sont séparés par une virgule. Vous savez donc comment **passer** des arguments à une fonction, mais comment créer une fonction qui les reçoive ?

La ligne se présente sous la forme suivante :

Code : VB.NET

```
Sub Addition(ByVal Valeur1 As Integer, ByVal Valeur2 As Integer)
```

Vous remarquez bien les `ByVal Valeur1 As Integer` ; cette syntaxe est à utiliser pour *chaque* argument : le mot « *ByVal* », le nom de la variable, « *As* » et le type de la variable.

Ce qui nous donne, dans un cas comme notre addition :

Code : VB.NET

```
Sub Addition(ByVal Valeur1 As Integer, ByVal Valeur2 As Integer)
```

```
            'Addition des deux arguments
            Console.WriteLine(Valeur1 & " + " & Valeur2 & " = " &
    Valeur1 + Valeur2)

            'Pause factice
            Console.Read()
        End Sub
```

Voilà par exemple le *Sub* que j'ai écrit, et qui additionne deux valeurs passées en argument.

Pourquoi n'as-tu pas déclaré les variables *Valeur1* et *Valeur2* ?

Elles ont été automatiquement déclarées dans la ligne de création de fonction.

Si vous souhaitez appeler cette fonction, comment faut-il procéder ?

Code : VB.NET

```
Addition(Valeur1, Valeur2)
```

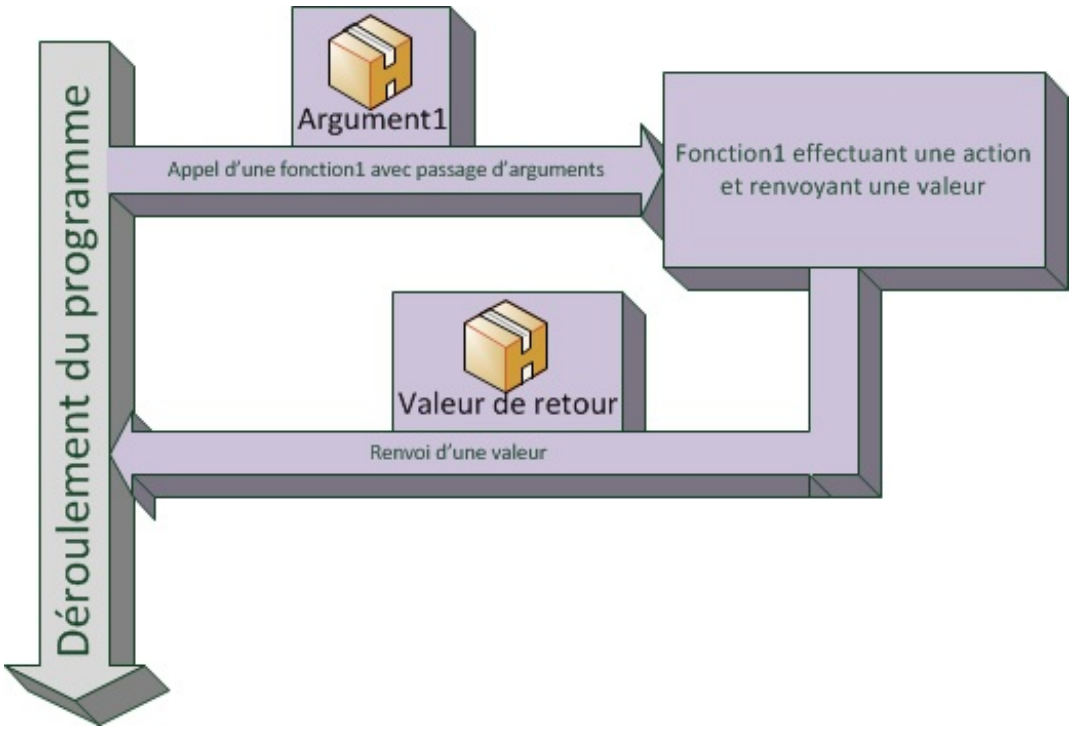
Vous avez bien compris ?

Vous n'êtes pas obligés de mettre les mêmes noms du côté de l'appel et du côté de la déclaration des arguments dans votre fonction ; la ligne `Addition(Valeur10, Valeur20)` aurait fonctionné, mais il faut bien sûr que « Valeur10 » et « Valeur20 » soient déclarées du côté de l'appel de la fonction.

Vous avez dû le remarquer, il faut obligatoirement utiliser tous les arguments lors de l'appel de la fonction, sauf une petite exception que je vous présenterai plus tard.

Les arguments doivent être passés dans le bon ordre !

Valeur de retour



Imaginez que vous ayez envie d'une fonction qui effectue un calcul très compliqué ou qui modifie votre valeur d'une certaine manière. Vous voudriez sans doute récupérer la valeur ? C'est ce qu'on appelle le retour :

Code : VB.NET

```
Function Addition(ByVal Valeur1 As Integer, ByVal Valeur2 As
Integer) As Integer
```

C'est le morceau du bout (`As Integer`) qui nous intéresse : c'est cette partie qui indiquera à la fonction le type de valeur qu'elle doit renvoyer. Dans le cas présent, il s'agit d'un type numérique, mais j'aurais très bien pu écrire `As String`.

Hop hop hop, pourquoi as-tu écrit « *Function* » au début et non plus « *Sub* » ?

Je vous l'ai dit tout en haut : les *Sub* ne renvoient rien, il faut donc passer par les fonctions si on veut une valeur de retour.

Code : VB.NET

```
Function Addition(ByVal Valeur1 As Integer, ByVal Valeur2 As
Integer) As Integer
        Dim Resultat As Integer
        'Addition des deux arguments
        Resultat = Valeur1 + Valeur2

        'Renvoie le résultat
        Return Resultat
    End Function
```

Cette fonction additionne donc les deux nombres passés en argument et renvoie le résultat.

La ligne `Return Resultat` est très importante car c'est elle qui détermine le retour : si vous n'écrivez pas cette ligne, aucun retour ne se fera.

Ce qui suit le *Return* ne sera pas exécuté ; la fonction est quittée lorsqu'elle rencontre cette instruction. Vérifiez donc


bien l'ordre de déroulement de votre programme.


La valeur retournée, « Resultat » dans ce cas-ci, doit être du même type que le type de retour indiqué dans la déclaration de la fonction.

Maintenant, comment appeler cette fonction ? La forme `Addition(Valeur1, Valeur2)` aurait pu fonctionner, mais où va la valeur de retour ? Il faut donc récupérer cette valeur avec un « = ».

Code : VB.NET

```
Resultat = Addition(Valeur1, Valeur2)
```

Une fois cet appel écrit dans le code, ce dernier additionne les deux valeurs. Je suis conscient que cette démarche est assez laborieuse et qu'un simple `Resultat = Valeur1 + Valeur2` aurait été plus simple, mais c'était pour vous faire découvrir le principe.

Cette fonction peut être directement appelée dans une autre, comme ceci par exemple :

Code : VB.NET

```
Console.WriteLine(Addition(Valeur1, Valeur2))
```

Sachez que les fonctions vont vous être **très** utiles. J'espère qu'à présent, vous savez les utiliser. 

Petits plus sur les fonctions

Les arguments facultatifs

Tout d'abord, une petite astuce que je vais vous expliquer : l'utilisation des arguments facultatifs. Je vous ai dit que tous les arguments étaient indispensables, sauf exception ; eh bien, voici l'exception.

Les arguments facultatifs sont des arguments pour lesquels on peut choisir d'attribuer une valeur ou non au moment de l'appel de la fonction. Pour les déclarer, tapez `Optional ByVal Valeur3 As Integer = 0 .`

Le mot clé « *Optional* » est là pour dire qu'il s'agit d'un argument facultatif, le reste de la syntaxe étant la même que pour les autres fonctions.

Un argument facultatif doit toujours être initialisé et se faire attribuer une valeur dans la ligne de déclaration de la fonction.

Code : VB.NET

```
Function Operation(ByVal Valeur1 As Integer, ByVal Valeur2 As
Integer, Optional ByVal Valeur3 As Integer = 0) As Integer
        Return Valeur1 + Valeur2 + Valeur3
    End Function
```

Voilà donc la nouvelle fonction. Dans l'appel de cette fonction, je peux maintenant écrire `Operation(1, 5)` , ce qui me renverra **6**, ou alors `Operation(10, 15, 3)` qui me renverra **28**. Les deux appels sont valides.

Commenter sa fonction

Je vous ai déjà résumé la marche à suivre pour commenter des lignes. Mais voilà, comment faire pour commenter des fonctions entières ?
Placez-vous sur la ligne juste avant la fonction.

Code : VB.NET

```
Function Operation(ByVal Valeur1 As Integer, ByVal Valeur2 As
Integer, Optional ByVal Valeur3 As Integer = 0) As Integer
        Return Valeur1 + Valeur2 + Valeur3
    End Function
```

Ajoutez ensuite trois *quotes* : « ''' ».
Des lignes vont s'afficher :

Code : VB.NET

```
''' <summary>
'''
''' </summary>
''' <param name="Valeur1"></param>
''' <param name="Valeur2"></param>
''' <param name="Valeur3"></param>
''' <returns></returns>
''' <remarks></remarks>
```

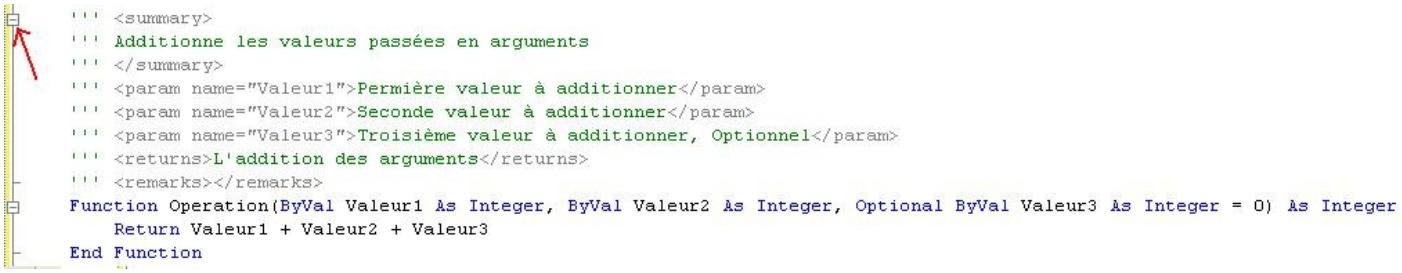
Ces lignes permettent de commenter la fonction : dans *summary*, expliquez brièvement le rôle de la fonction ; dans les paramètres, précisez à quoi ils correspondent ; et dans la valeur de retour, indiquez ce que la fonction retourne.

Par exemple, j'ai commenté ma fonction comme ceci :

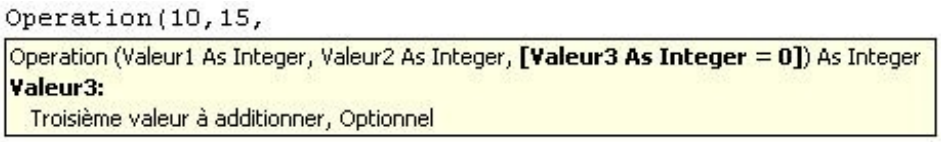
Code : VB.NET

```
''' <summary>
''' Additionne les valeurs passées en argument
''' </summary>
''' <param name="Valeur1">Première valeur à additionner</param>
''' <param name="Valeur2">Seconde valeur à additionner</param>
''' <param name="Valeur3">Troisième valeur à additionner,
Optionnel</param>
''' <returns>L'addition des arguments</returns>
''' <remarks></remarks>
```

Cliquez ensuite sur cette petite flèche pour « replier » cette zone :



À présent, à chaque endroit où vous allez écrire cette fonction, un cadre s'affichera vous indiquant ce qu'il faut lui donner comme arguments.



Cela est **très** utile lorsque vous avez beaucoup de fonctions.

Petit exercice

Pour clore ce chapitre, je vous propose un petit exercice.

Je vais vous demander de remplir un tableau de dix cases et d'additionner les valeurs, la récupération de ces valeurs devant se faire en toute sécurité, comme dans notre TP sur l'addition.
La partie qui demande la valeur et qui vérifie s'il s'agit d'un nombre devra être écrite dans une fonction séparée.

À vos claviers !

Secret (cliquez pour afficher)

Code : VB.NET

```
Module Module1

    Sub Main()
        Dim TableauDeValeurs(9) As Integer
        Dim Resultat As Integer = 0

        'Demande les valeurs en passant par la fonction
        For i As Integer = 0 To TableauDeValeurs.Length - 1
            TableauDeValeurs(i) = DemandeValeur(i)
        Next

        'Additionne les valeurs
        For i As Integer = 0 To TableauDeValeurs.Length - 1
            Resultat = Resultat + TableauDeValeurs(i)
        Next

        'Affiche le résultat
        Console.WriteLine(Resultat)

        'Pause
        Console.Read()
    End Sub

    Function DemandeValeur(ByVal Numero As Integer) As Integer
        Dim ValeurEntree As String

        'Demande la valeur
        Do
            Console.WriteLine("Entrez valeur " & Numero + 1)
            ValeurEntree = Console.ReadLine()
            'Tourne tant que ce n'est pas un nombre
        Loop Until IsNumeric(ValeurEntree)

        'Convertit la valeur en integer et la renvoie
        Return CInt(ValeurEntree)
    End Function
End Module
```

Clarifions un peu ce code.

J'entre dans une boucle dans laquelle j'appelle la fonction, en passant comme paramètre le numéro de la boucle. Mais pourquoi ?
Allons faire un tour du côté de la fonction : ce paramètre me permet de dire à l'utilisateur quel est le numéro qu'il entre (`Console.WriteLine("Entrez valeur " & Numero + 1)` ; j'ajoute « 1 » pour ne pas avoir de `Entrez valeur 0`).

Cette fonction, vous la connaissez : c'est la même que pour l'addition ; sauf que... la ligne `Return CInt(ValeurEntree)` vous inquiète...

J'explique : ce que je viens de faire s'appelle un ***Cast***, c'est-à-dire que l'on convertit un type en un autre.
Cette ligne fait appel à la fonction ***CInt()***, qui permet de convertir une variable de type *string*, par exemple, en *integer*.
Pourquoi ai-je fait cela ?

Parce que je renvoie un *integer*, et que la variable est pour le moment un *string*. Je peux le faire en toute sécurité puisque je vérifie que mon *string* contient bien une valeur numérique ; s'il avait contenu, par exemple, un mot, il y aurait eu une erreur.

Vous auriez très bien pu passer par une variable intermédiaire, comme on l'a vu précédemment.

Voilà qui clôt notre chapitre sur les fonctions.

Les inclassables

Quelques petits plus pour exploiter encore mieux les fonctionnalités de Visual Basic.

Les constantes

Pour commencer cette partie des inclassables, je vais vous apprendre ce qu'est une *constante* en programmation.

Je pense que vous savez ce qu'est une constante dans le langage normal : c'est une variable qui ne varie pas (rigolo !) ; elle garde **toujours** la même valeur.
Dans notre programme, ce sera pareil : une constante ne variera jamais au cours de notre programme, on ne peut donc pas lui assigner de valeur une fois sa déclaration effectuée.

C'est assez intéressant : imaginez un tableau dans lequel vous demandez dix valeurs à l'utilisateur. Vous allez le déclarer comme ceci :

Code : VB.NET

```
Dim MonTableau(9) As Integer
```

Et, dans le reste du code, vos boucles auront pour fin :

Code : VB.NET

```
to MonTableau.Length - 1
```

Si vous souhaitez changer et demander vingt valeurs au lieu de dix, vous allez devoir modifier cela dans la déclaration du tableau, ce qui, pour le moment, est simple si votre déclaration est faite au début de votre code.

Seulement, vous allez rapidement prendre l'habitude de déclarer vos variables en plein milieu du code, là où vous en avez besoin.
La joie pour retrouver son petit morceau de tableau dans tout ça...

Une autre solution serait de déclarer une constante, comme ceci :

Code : VB.NET

```
Const LONGUEURTABLEAU As Integer = 9
```

... et de déclarer votre tableau ainsi :

Code : VB.NET

```
Dim MonTableau(LONGUEURTABLEAU) As Integer
```

Eh oui, ça fonctionne ! Maintenant, vous rassemblez toutes vos constantes en haut de la page (ou dans une page à part pour les gros programmes), et voilà le moyen d'adapter facilement vos programmes sans trop de difficulté.

Les structures

Une autre chose qui pourra vous être utile dans certains programmes : les structures.

Alors, à quoi sert et comment utiliser une structure, ou plutôt un tableau de structure ? (Eh oui, on grille les étapes !)

Tout d'abord, une structure est un assemblement de plusieurs variables ; une fois n'est pas coutume, je vais vous donner un exemple.

Vous avez l'intention de créer des fiches de livre dans votre programme. Chaque fiche rassemble les informations d'un livre : titre, auteur, genre, etc. Eh bien, dans ce cas, un tableau de structure va vous être utile (je parle de tableau de structure, car si on n'utilise la structure qu'une seule fois, elle est presque inutile).

Maintenant, comment l'utiliser ?

Sa déclaration est simple :

Code : VB.NET

```
Structure FicheLivre
   Dim ID        As Integer
   Dim Titre     As String
   Dim Auteur    As String
   Dim Genre     As String
End Structure
```

Nous voici donc avec une structure définie (comme pour une fonction, il y a un ***End Structure*** à la fin). Comme vous pouvez le constater, je l'ai nommée « FicheLivre ».
En définissant cette structure, c'est comme si on avait créé un nouveau type de variable (symboliquement). Mais il faut à présent la déclarer et, pour ce faire, utilisons ce nouveau type !

C'est au moment de la déclaration que l'on décide si l'on souhaite un tableau de structure ou une simple structure :

Code : VB.NET

```
'Déclare une simple structure
Dim MaStructure As FicheLivre
'Déclare un tableau de structure
Dim MonTableauDeStructure(9) As FicheLivre
```

Je vais donc utiliser le tableau pour vous montrer comment on utilise cette structure.

Code : VB.NET

```
MonTableauDeStructure(0).ID = 0
MonTableauDeStructure(0).Titre = "Les Misérables"
'...
MonTableauDeStructure(9).Auteur = "Dan Brown"
MonTableauDeStructure(9).Genre = "Policier"
```

Voilà comment remplir votre structure ; cette méthode de programmation permet de se retrouver facilement dans le code.

Voici un exemple pour afficher cette structure :

Code : VB.NET

```
        For i As Integer = 0 To 10
            Console.WriteLine("ID : " & MonTableauDeStructure(i).ID)
            Console.WriteLine("Titre : " & MonTableauDeStructure(i).Titre)
            Console.WriteLine("Auteur : " & MonTableauDeStructure(i).Auteur)
            Console.WriteLine("Genre : " & MonTableauDeStructure(i).Genre)
        Next
```

Voilà encore un petit truc toujours utile.

Boucles supplementaires

Vous vous souvenez encore des boucles conditionnelles ? 

Eh bien, je vais vous en faire découvrir deux nouvelles : ***For Each*** et ***IIf*** .

Roh, tu es embêtant ! C'était tout à l'heure qu'il fallait nous expliquer ça, pas maintenant !

Désolé mais, tout à l'heure, vous ne pouviez pas vous en servir : vous n'aviez pas encore les connaissances requises.

Bon, je passe tout de suite à la première boucle !

For Each, traduisible par « pour chaque »

Vous vous souvenez des tableaux ?

Code : VB.NET

```
Dim MonTableau(9) As integer
```

Eh bien, la boucle ***For Each*** permet de parcourir ce tableau (un peu à la manière du *For* traditionnel) et de récupérer les valeurs.

Utilisons donc immédiatement cette boucle :

Code : VB.NET

```
For Each ValeurDeMonTableau As Integer In MonTableau
   If ValeurDeMonTableau < 10 Then
      Console.WriteLine(ValeurDeMonTableau)
   End If
Next
```

Ce qui se traduit en français par ceci : « Pour chaque *ValeurDeMonTableau* qui sont des entiers dans *MonTableau* ».

Ce code parcourt mon tableau et, pour chaque valeur, vérifie si elle est inférieure à 10 ; si c'est le cas, il l'affiche.

On ne peut pas assigner de valeurs dans un *For Each*, seulement les récupérer.

Très utile, donc, pour lire toutes les valeurs d'un tableau, d'un objet liste par exemple (que nous verrons plus tard).

Un *For Each* peut être utilisé pour parcourir chaque lettre d'un mot :

Code : VB.NET

```
Dim MaChaine As String = "Salut"
Dim Compteur As Integer = 0
For Each Caractere As String In MaChaine
   If Caractere = "a" Then
       Compteur = Compteur + 1
   End If
Next
```

Ce code compte le nombre d'occurrences de la lettre *a* dans un mot.

IIF

IIF est très spécial et peu utilisé : dans un certain sens, il simplifie un *If*. Voici un exemple de son utilisation dans le code précédent :

Code : VB.NET

```
Dim MaChaine As String = "Salut"
Dim Compteur As Integer = 0
For Each Caractere As String In MaChaine
   If Caractere = "a" Then
       Compteur = Compteur + 1
   End If
Next
Console.WriteLine(IIF(Compteur > 0, "La lettre 'a' a été trouvée
dans " & MaChaine, "La lettre 'a' n'a pas été trouvée dans " &
MaChaine))
```

En clair, si vous avez bien analysé : si le premier argument est vrai (c'est un booléen), on retourne le second argument ; à l'inverse, s'il est faux, on retourne le dernier.

Pour mieux comprendre :

Code : VB.NET

```
IIF(MonBooleen, "MonBooleen est true", "MonBooleen est false")
```

« MonBooleen » peut bien évidemment être une condition.

Les Cast

J'ai brièvement parlé des *Cast* dans un chapitre précédent : lorsque j'ai converti un *string* en un *integer* et que je vous ai dit que j'avais ***casté*** la variable.

Donc, vous l'avez compris, un *Cast* convertit une variable d'un certain type en un autre.

Attention lors des *Cast*, soyez bien sûrs que la variable que vous allez transformer peut être convertie : si vous transformez une lettre en *integer*, une erreur se produira.

Alors, il existe plusieurs moyens d'effectuer des *Cast* : une fonction universelle, et d'autres plus spécifiques.

Ctype()

La fonction universelle se nomme *Ctype*. Voici sa syntaxe :

Code : VB.NET

```
Ctype(MaVariableString, Integer)
```

Ce code convertira « MaVariableString » en *integer*.

Voici un exemple concret :

Code : VB.NET

```
Dim MonString As String = "666"
If Ctype(MonString, Integer) = 666 Then
'...
End If
```

Encore une fois, faites attention. Un code du style...

Code : VB.NET

```
Dim MonString As String = "a666"
If Ctype(MonString, Integer) = 666 Then
'...
End If
```

... produira une grosse erreur !

Les fonctions spécifiques

On a vu l'exemple de *Ctype()*, utile lorsqu'il s'agit de types peu courants. Mais pour les types courants, il existe des fonctions plus rapides et adaptées :

- *CBool()* : retourne un *Boolean*.
- *CByte()* : retourne un *Byte*.
- *CChar()* : retourne un *Char*.
- *CDate()* : retourne une date.
- *CDbl()* : retourne un *Double*.
- *CDec()* : retourne un nombre décimal.

- *CInt()* : retourne un *Integer*.
- *CLng()* : retourne un *Long*.
- *CSng()* : retourne un *Single*.
- *CStr()* : retourne un *String*.
- *CUInt()* : retourne un *Unsigned Integer*.
- *CULng()* : retourne un *Unsigned Long*.
- *CUShort()* : retourne un *Unsigned Short*.

Toutes ces fonctions ne prennent qu'un argument : la variable à convertir... c'est facile à retenir !
Les fonctions en rouge sont les plus utilisées.

Que sont ces « ***unsigned*** » ?

Ah, tenez... c'est une bonne occasion de vous en parler.
Vous connaissez le type numérique *integer* ? (Oui, évidemment ! )

Eh bien, le *unsigned* permet d'augmenter la capacité de vos variables : au lieu d'aller d'environ - 2 000 000 000 à 2 000 000 000 dans le cas d'un *int*, cette capacité s'étend plutôt de 0 à 4 000 000 000 (approximativement) ; « *unsigned* » signifiant « non signé », il n'y a plus de signe.

En quoi cela peut-il nous être utile ? Je n'ai pas encore trouvé d'utilisation particulière parce que, si j'ai besoin d'un nombre plus grand que quatre milliards, j'utilise *long* qui peut contenir des nombres beaucoup plus grands.

Il était surtout utilisé à l'époque où chaque bit de données comptait.

De retour à nos petites fonctions : leur utilisation.

Code : VB.NET

```
Dim MonString As String = "666"
If CInt(MonString) = 666 Then
'...
End If
```

Le type Object

Dernière chose pour cette partie « petits plus » : je vais vous présenter un nouveau type, appelé ***object***.

Ce type *object* (qui remplace le type ***variant*** en VB6) est utilisé lorsque l'on ne sait pas ce que va contenir notre variable. Il peut donc contenir des mots, des nombres, etc.

Exemple concret : vous vous souvenez de notre calculatrice ; les instructions dans lesquelles on demandait la valeur tout en vérifiant qu'il s'agissait d'un nombre étaient les suivantes.

Code : VB.NET

```
'Récupération du premier nombre
        Do
            Console.WriteLine("Entrez la première valeur")
            ValeurEntree = Console.ReadLine()
            'Tourne tant que ce n'est pas un nombre
        Loop Until IsNumeric(ValeurEntree)
        'Écriture de la valeur dans un double
        Valeur1 = ValeurEntree
```

Nous allons refaire cette partie en utilisant le type *object*.
Déclarons notre variable de type objet :

Code : VB.NET

```
Dim MonObjet As Object
```

Nous allons devoir tourner dans notre boucle tant qu'il ne s'agit pas d'un nombre.

Il est tout à fait possible d'utiliser la fonction *IsNumeric()* dans le cas d'un *object*, mais il existe aussi `TypeOf MonObjet Is Integer` qui renvoie un booléen.
Une fois placé dans une boucle, on retrouve notre programme sous une autre forme :

Code : VB.NET

```
Dim MonObjet As Object
Do
    Console.WriteLine("Entrez la première valeur")
    MonObjet = Console.ReadLine()
    'Tourne tant que ce n'est pas un nombre
Loop Until IsNumeric(MonObjet)
MonObjet = CInt(MonObjet)
```

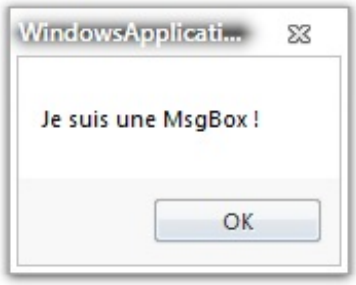
Cela revient au même que le code précédent, hormis que l'on n'utilise qu'une seule variable.

Lorsque, par exemple, vous *castez* un *object* en *integer*, vérifiez bien qu'il n'y a que des nombres à l'intérieur (comme les *string*, quoi).
En résumé, je ne vous conseille vraiment pas d'utiliser ce type, des erreurs de conversion de type pouvant très vite être oubliées.

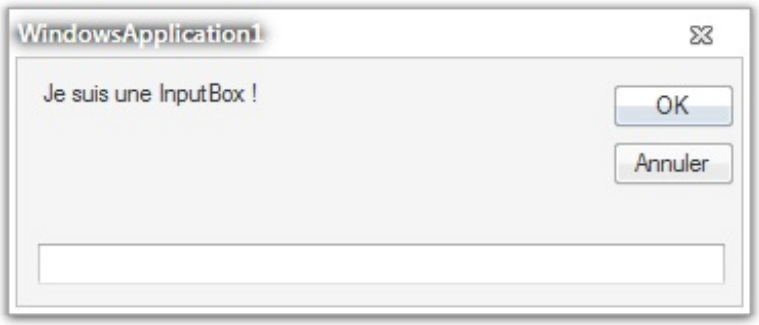
Les MsgBox et InputBox

Deux petites choses qui peuvent également vous aider : les ***MsgBox*** et les ***InputBox***.

Sur le côté droit, vous pouvez voir une *MsgBox* sous sa forme la plus basique.



Et, juste en dessous, une *InputBox*.



À quoi ça sert ?

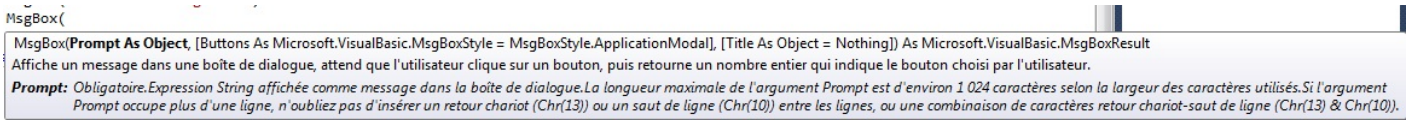
Première question... Eh bien, diverses utilisations peuvent en être faites, puisqu'elles seront utilisables du côté visuel également.

Les *MsgBox* peuvent signaler une erreur, demander une confirmation, etc. Les *InputBox*, quant à elles, peuvent être utilisées dans des scores par exemple, pour demander le nom du joueur.

Beaucoup d'arguments existent pour les paramétrer, je vais vous les expliquer.

La MsgBox

Les paramètres



Voilà la liste des arguments. Pas de panique, il n'y en a que trois !
Je vais vous les décrire :

1. *Prompt* : message qui apparaîtra dans la *MsgBox*.
2. *Buttons* : type de boutons à utiliser (style de la boîte).
3. *Title* : titre de la boîte.

Pour ce qui est du deuxième argument — les boutons à utiliser —, lorsque vous êtes sur le point d'entrer cet argument, une liste s'offre à vous : c'est cette liste qu'il vous faut utiliser pour trouver votre bonheur.

Voici divers exemples de style.

Membre	Valeur	Description
OKOnly	0	Affiche le bouton « OK » uniquement.
OKCancel	1	Affiche les boutons « OK » et « Annuler ».
AbortRetryIgnore	2	Affiche les boutons « Abandonner », « Réessayer » et « Ignorer ».
YesNoCancel	3	Affiche les boutons « Oui », « Non » et « Annuler ».
YesNo	4	Affiche les boutons « Oui » et « Non ».
RetryCancel	5	Affiche les boutons « Réessayer » et « Annuler ».

Critical	16	Affiche l'icône « Message critique ».
Question	32	Affiche l'icône « Requête d'avertissement ».
Exclamation	48	Affiche l'icône « Message d'avertissement ».
Information	64	Affiche l'icône « Message d'information ».
DefaultButton1	0	Le premier bouton est le bouton par défaut.
DefaultButton2	256	Le deuxième bouton est le bouton par défaut.
DefaultButton3	512	Le troisième bouton est le bouton par défaut.
ApplicationModal	0	L'application est modale. L'utilisateur doit répondre au message avant de poursuivre le travail dans l'application en cours.
SystemModal	4096	Le système est modal. Toutes les applications sont interrompues jusqu'à ce que l'utilisateur réponde au message.
MsgBoxSetForeground	65536	Spécifie la fenêtre de message comme fenêtre de premier plan.

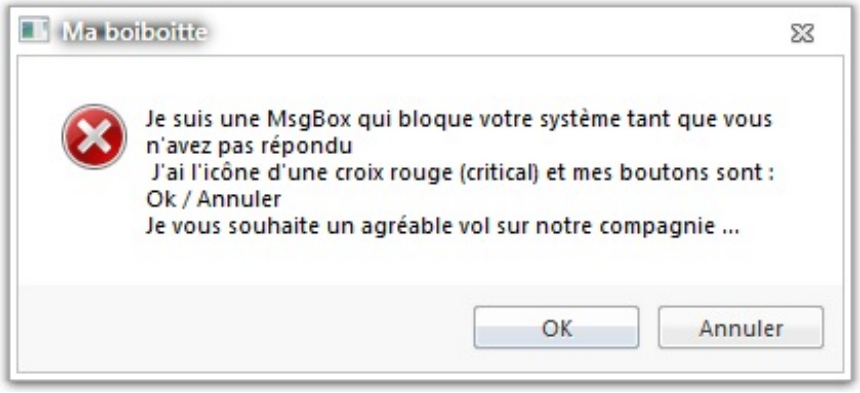
Les numéros indiqués correspondent aux *ID*, que vous pouvez cumuler. En gros, si vous souhaitez que votre boîte bloque le système et que l'on doive y répondre avant de continuer, avec une icône « Message critique » et des boutons « OK – Annuler », il faut que vous tapiez… 4096 + 1 + 16 = « **4113** ».

Voici donc le code correspondant, les **Chr(13)** représentant des retours à la ligne :

Code : VB.NET

```
MsgBox("Je suis une MsgBox qui bloque votre système tant que vous n'avez pas répondu" & Chr(13) & " J'ai l'icône d'une croix rouge (critical) et mes boutons sont : Ok / Annuler" & Chr(13) & "Je vous souhaite un agréable vol sur notre compagnie ...", 4113, "Ma boiboitte")
```

Cela donne ce résultat :



Pour le moment, c'est bon ? 

Le retour

Passons à la valeur de retour !

Les boutons sur lesquels on clique ne renvoient pas tous la même valeur :

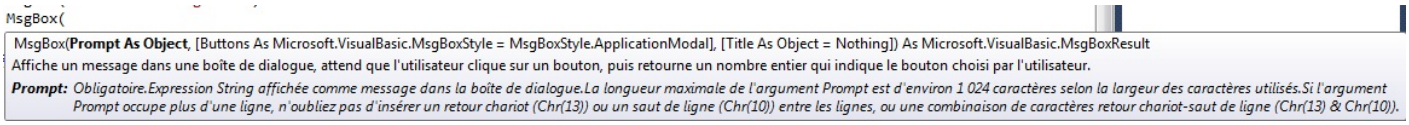
1. OK → 1
2. Cancel → 2
3. Abort → 3
4. Retry → 4
5. Ignore → 5
6. Yes → 6
7. No → 7

Un petit *If* devrait vous permettre d'effectuer une action précise en fonction de ce que l'utilisateur a entré !

InputBox

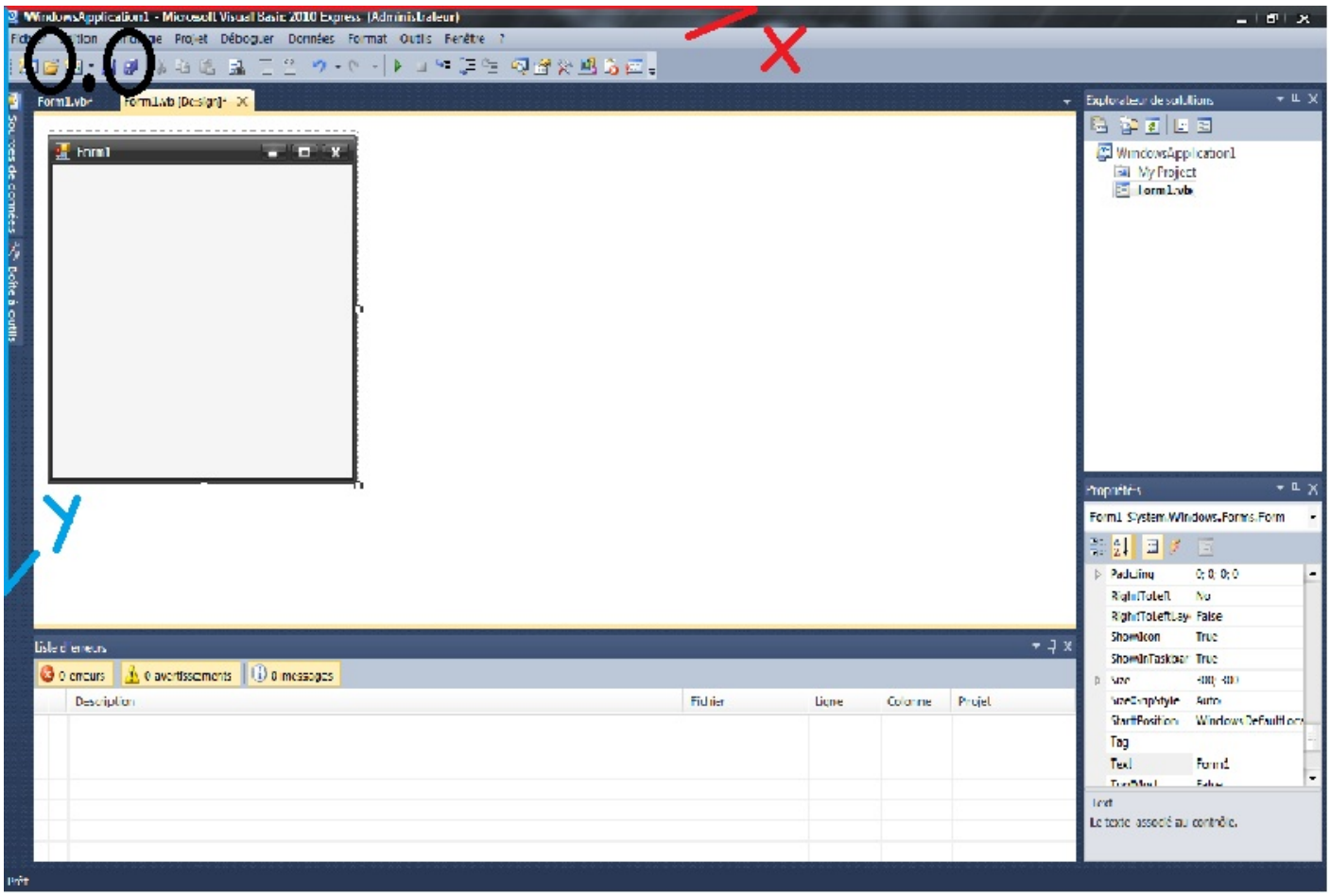
Les paramètres

Les arguments de l'*InputBox* sont un peu moins ennuyeux...



Les arguments ne sont donc pas difficiles à retenir :

1. Le *Prompt*, comme pour la *MsgBox*.
2. Le titre de la fenêtre.
3. La valeur par défaut entrée dans le champ à remplir.
4. La position de la boîte en X.
5. La position de la boîte en Y.



Comme vous pouvez le voir sur le dessin que j'ai fait (ou plutôt, essayé de faire...), l'origine du repère se situe en haut à gauche. Si vous entrez « 0 » pour les positions X et Y, alors la boîte se retrouvera en haut à gauche ; pour la laisser centrée, ne mettez rien.

Le retour

Cette fois, la valeur de retour n'est pas forcément un nombre : cela peut être une chaîne de caractères ou toute autre chose que l'utilisateur a envie d'écrire.

Voilà pour les *Box*, c'est fini !

Partie 2 : Le côté visuel de VB

Déprimés avec tout ce noir, voir la console pendant tout une partie ça désespère ?

Eh bien réjouissez-vous, on attaque la partie visuelle de Visual Basic !

Vous rêviez de pouvoir enfin commencer à concevoir des programmes concrets, qu'on à envie d'utiliser. Et tout cela sans avoir à ajouter des centaines de lignes supplémentaires à votre code ? Eh bien voilà la vraie force du Visual Basic.

Découverte de l'interface graphique

Des fenêtres, je veux des fenêtres ! A partir de maintenant, finit les essais au milieu du noir et du blanc de notre console. Nous allons donc commencer à aborder les nouveaux concepts du graphisme en commençant par placer des **contrôles** et découvrir les **évènements**.

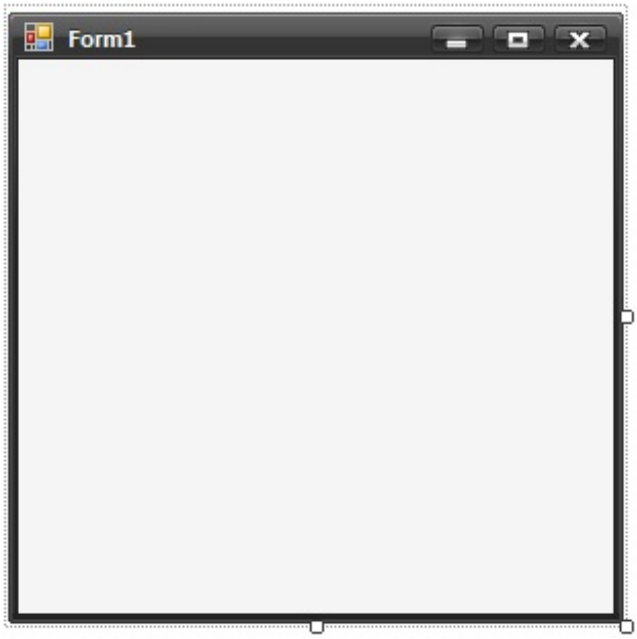
Allons y !

Les nouveautés

Jusqu'à maintenant je vous ai **obligés** (grrr) à rester sur la console. C'est moche d'accord, mais comprenez-moi, vous avez eu besoin d'uniquement 2 fonction jusqu'à maintenant : **Console.ReadLine()** pour l'entrée et **Console.WriteLine()** pour la sortie.

Ici, vous n'aurez plus besoin de l'objet **"Console"**, donc les "Console." on oublie !

Recréons un nouveau projet, **windows forms** cette fois-ci ; et découvrons !



Sur notre gauche nous retrouvons le panneau que je vous ai présenté tout au début de ce tutoriel : la boîte à outils.

Cette boite contient donc des outils, outils que nous allons déposer sur notre feuille. J'appelle feuille la petite fenêtre avec rien dedans au centre de votre écran, c'est un peu comme votre feuille et vous allez dessiner dedans.

Avantages par rapport à la console

Tout d'abord les avantages par rapport à la console sont immenses : c'est plus beau, c'est agréable de travailler dessus, c'est fonctionnel mais surtout, si vous vous amusez à lancer votre projet vide, sans aucune ligne de code ajoutée : votre fenêtre se lance et reste là. Elle restera jusqu'à ce que vous appuyiez sur la croix rouge en haut à droite.

Vous l'avez donc compris, si on écrit quelque chose dedans, ça reste !

Mais ce ne sera pas aussi simple qu'à la console. Il faudra passer par nos outils pour écrire et interagir avec l'utilisateur.

Donc, il faudra bien connaitre nos outils pour savoir lequel utiliser dans quelles situations.

Le Label par exemple nous servira principalement à écrire du texte dans cette interface, la textbox à demander à l'utilisateur d'écrire du texte et le bouton à déclencher un évènement.

Manipulation des premiers objets

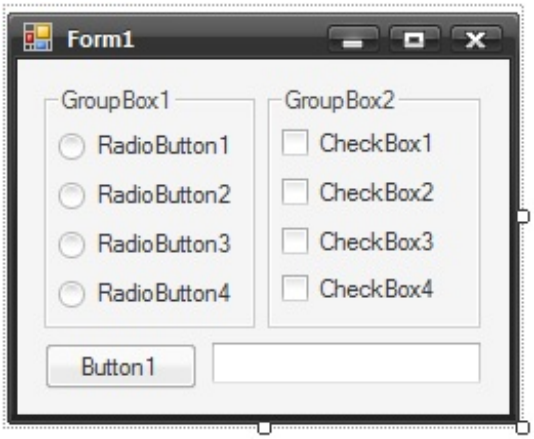
Retournons à notre feuille :

Rien qu'elle, vous pouvez déjà la manipuler : quand vous cliquez dessus des carrés blancs apparaissent sur les bords, ils vont nous permettre d'agrandir ou rapetisser la fenêtre, comme n'importe quelle autre fenêtre windows.

Une fois cette fenêtre à la hauteur de vos espérances, nous allons apprendre à ajouter des **objets** dedans, ces objets sont appelés des **contrôles**.

Alors pour ce faire, je vais vous laisser vous amuser avec les objets : prenez-les en cliquant dessus puis faites les glisser sans relâcher le clic jusqu'à la fenêtre.

Laissez libre court à votre imagination, essayez tous les objets que vous voulez .



J'aime pas les noms qu'il y a, je fais comment ?

Stop, pourquoi vouloir savoir courir avant de marcher ?

On va l'apprendre dans le prochain chapitre. Mais ce n'est pas une raison pour fermer ce chapitre et aller tout de suite au suivant !

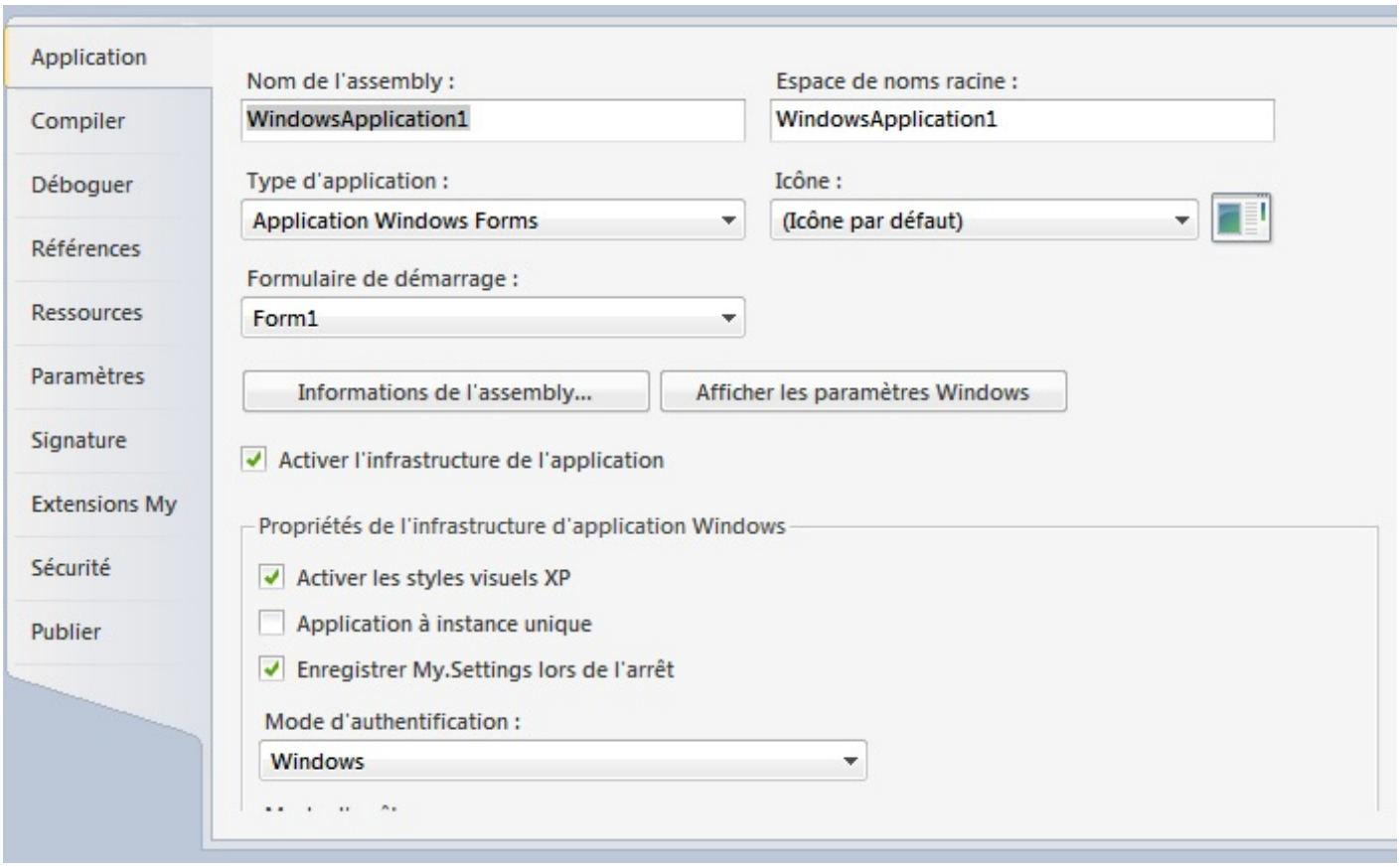
En attendant vous pouvez déjà lancer ce projet, votre fenêtre apparaitra. Bon, rien ne se passe quand vous appuyez sur les boutons, pourquoi ?

Eh bien nous n'avons pas encore codés **d'événements** !
Ça ne saurait tarder ...

Les paramètres de notre projet

Je vais quand même vous expliquer une dernière petite chose :

Cliquez dans projet -> propriétés de [nom de votre projet]



Voici cette fenêtre.

J'explique pourquoi elle va nous être utile. Elle permet tout d'abord de choisir un nom et une icône pour votre application (mais bon ce n'est pas la priorité), mais elle servira surtout à choisir sur quelle fenêtre votre projet va démarrer. Très utile lorsqu'on en aura plusieurs.

Les autres options étant plus techniques, et pas nécessaires actuellement.

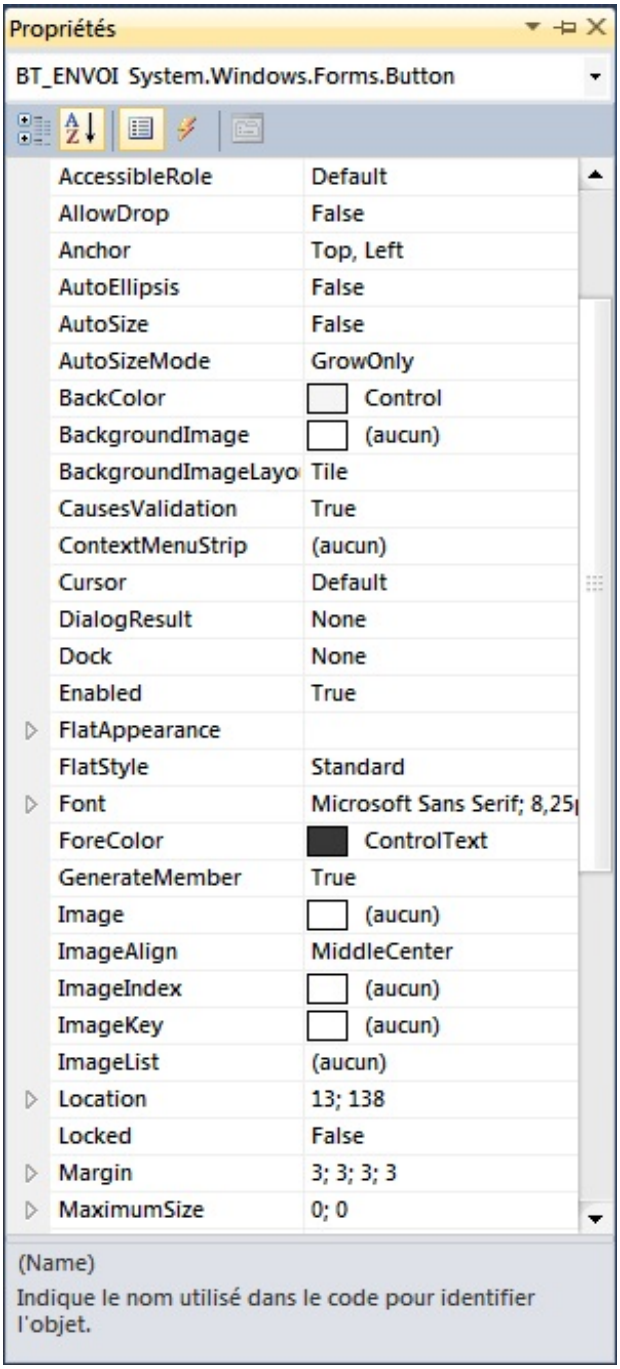
Les propriétés

Eh bien, déjà pas mal de nouveaux concepts à appréhender, mais ce n'est pas finit ! Vous ne voulez quand même pas en rester là ? Eh bien modifions les **propriétés** de nos contrôles !

A quoi ça sert ?

Alors vous vous demandez sûrement à quoi servent les propriétés, même si vous avez déjà vu la fenêtre les contenant.

La chose magnifique est que nous sommes sous Visual Basic Express, un module du grand IDE qu'est Visual Studio, cet IDE est, pour moi, **le meilleur** actuellement.



Votre IDE va vous permettre de : vous dire les erreurs en français la plupart du temps, la possibilité de les résoudre sans se poser de questions, **la description de toutes les propriétés des objets**, la description de toutes les fonctions, un système de debugg magique et j'en passe. Bref, Visual Basic Express va vous mâcher énormément le travail.

Tout ça pour vous dire : côté propriétés nous allons être grandement aidés, intuitif comme tout, un vrai plaisir.

Donc revenons à nos moutons : qu'est ce qu'une propriété sur un objet VB, attention ici je parle des objets graphique que nous voyons (boutons, labels, textbox ...).

Eh bien ces propriétés sont toutes la partie design et fonctionnelle de l'objet : vous voulez cacher l'objet, agissez sur la propriété "Visible" ; vous voulez le désactiver, la propriété "Enable" est là pour ça ; l'agrandir, lui faire afficher autre chose, le changer de couleur, le déplacer, le tagger ..., agissez sur ses propriétés.

Les propriétés nous seront accessibles côté feuille design, mais aussi côté feuille de code VB, on agit d'un côté ou de l'autre. Très utile lorsque vous voulez le faire apparaître, disparaître dynamiquement, l'activer le désactiver, etc.

Si vous voulez le placer, lui attribuer un nom, et le définir comme vous voulez : agissez côté design, ensuite si vous voulez le déplacer pendant l'exécution, les propriétés seront modifiées coté VB.

Ok, ok on te crois au fait comment ouvrir le code, je ne vois que le design ...

Oh, excusez-moi, pour ouvrir le code, allons dans notre fenêtre solutions, vous devez voir

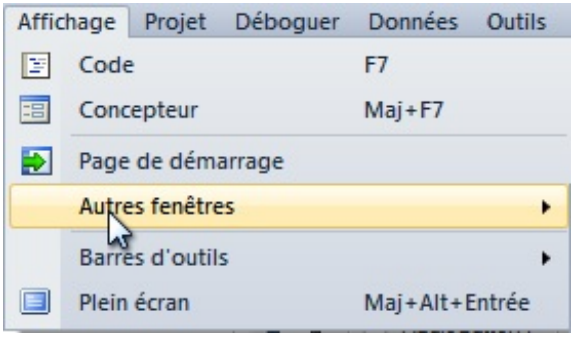
- une icône "myproject", elle correspond aux propriétés du projet entier, que je vous ai expliqué
- Le second est "form1.vb", c'est ce fichier qui nous intéresse.

Cliquez droit sur form1.vb vous avez :

- concepteur de vues pour le côté design
- et ... Afficher le code ! Voilà comment afficher le code.

Les utiliser

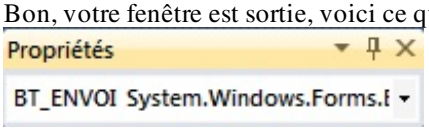
Alors vous savez désormais à quoi ça sert, mais comment se servir de ces magnifiques choses ?



Eh bien côté visuel, pas trop de mal : sortez la fenêtre propriétés.

Si vous ne savez plus comment on fait, au bout de la barre d'outils  cette icône sinon regardez sur votre droite ->

Bon, votre fenêtre est sortie, voici ce qu'il y a dans la partie supérieure



Le mot en gras est le nom de votre **contrôle** (que j'appelle également objet), ce nom est défini dans la propriété **(name)**, à noter que les propriété entre parenthèses ne peuvent plus être accédées côté VB.

Cette propriété est **FONDAMENTALE**, comme elle correspond au nom de l'objet (ou son ID), c'est ce nom qui sera utilisé Côté VB pour y accéder et le modifier. Utilisez un nom explicite !

Je vous explique ma manière de procéder : Mes contrôles sont toujours en MAJUSCULE.
Ils contiennent un préfixe en fonction de leur type :

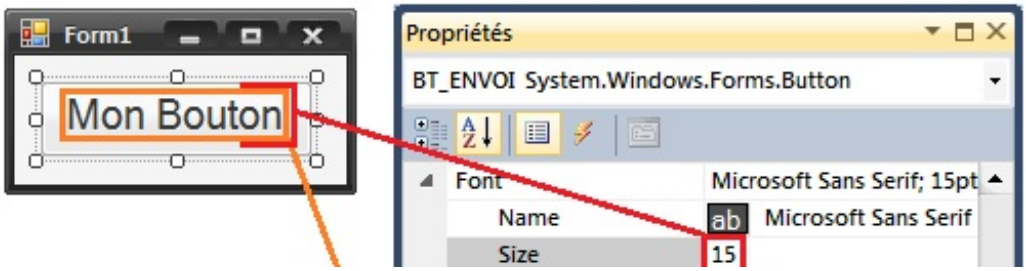
- BT pour les boutons
- LBL pour les labels
- TXT pour les txtbox
- LNK pour les labels link
- RB pour les radio boutons
- CHK pour les chkbox
- et j'en passe ...

Je ne vous oblige absolument pas mais je vous conseille quand même, après ce préfixe je place un **underscore** "_" puis la fonction rapide de l'objet. exemple pour ici : BT_ENVOI est le bouton pour envoyer.

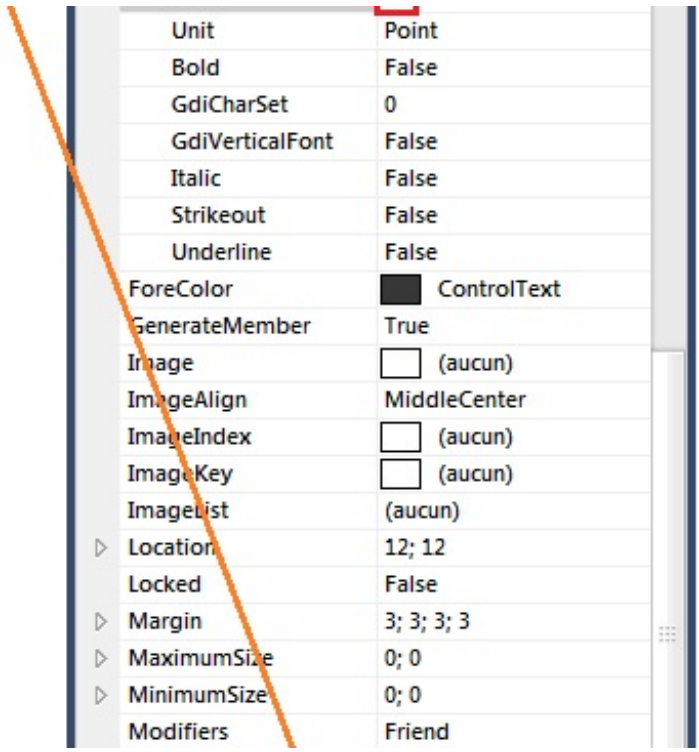
Bon après notre nom, nous avons le type d'objet que c'est ici, c'est un "button".

Faites attention à ces données en haut quand vous cliquez sur un contrôle, normalement Visual Basic Express doit vous afficher tout de suite ses propriétés mais j'ai déjà eu des cas où la propriété de l'ancien objet était resté et je modifiait le mauvais ... (anecdote )

Dans le reste de cette fenêtre, vous voyez la liste des propriétés de l'objet. Cliquez sur la case correspondante pour lui assigner une propriété. Dans le cas de mon bouton, vous le voyez à droite, j'ai modifié sa



propriété **"font"**, qui veut dire en anglais **police**, j'ai changé sa **"size"** autrement dit taille, et j'ai modifié également sa propriété **"Text"**, pour lui changer son nom.



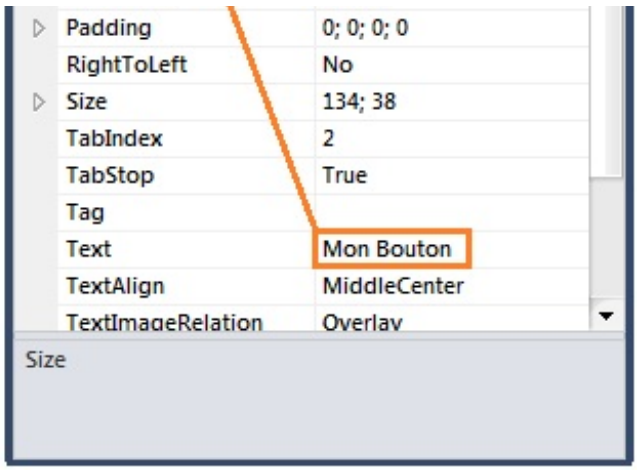
Pour qu'on travaille tous pareil, on va tous créer cette petite fenêtre :

Donc j'ai un bouton appelé ... BT_ENVOI, et une TextBox, que vous trouvez également sur le côté pour placer vos objets appelée TXT_RECOIT.

Et c'est parti pour l'aventure

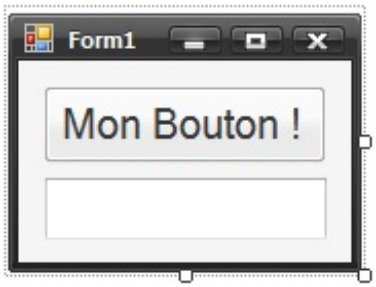
Les assigner et les récupérer côté VB

Nous allons donc modifier leurs propriétés côté VB.



Vous vous souvenez du Sub Main() quand nous étions en console ?

Ici, c'est à peut près pareil



sauf que ça s'appelle des **événements** (j'expliquerai plus tard pas de panique), et notre événement utilisé ici s'appelle **form_load**, c'est, comme son nom l'indique l'événement pénétré lorsque la fenêtre se lance (plus exactement durant son chargement).

Donc pour le créer il y a 2 manières : l'écrire mais comme vous ne savez pas les syntaxes des événements on ne va pas vous prendre la tête pour le moment, ou le générer automatiquement grâce à notre IDE.

Comment ?

Peut-être l'avez vous déjà fait par erreur : double cliquez sur n'importe quel point de la fenêtre que vous créer (pas un **contrôle** surtout !).

Vous atterrissez coté code VB

Code : VB.NET

```
Public Class Form1

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

    End Sub

End Class
```

Avec ce magnifique code. Alors pour le moment comprenez juste que ce sub sera pénétré au chargement de la fenêtre, j'expliquerai plus tard comment ça fonctionne. Donc on va écrire dedans.

Alors, comment attribuer les propriétés ? Eh bien c'est très simple : il faut tout d'abord dire sur quelle fenêtre on travaille, ici (et dans la majorité des cas) c'est la fenêtre actuelle appelée **"Me"** en VB (un peu comme le "this" en javascript ou en C++).

Donc nous avons "**Me**", il faut le lier au reste donc utilisons le "**.**" (nous allons donc accéder aux objets et contrôles de cette fenêtre), ici une liste s'affiche à nous, c'est tout ce que l'on peut utiliser avec l'objet **me** (autrement dit la fenêtre), spécifions autre chose : nous voulons accéder à notre textbox, donc on tape son nom : "TXT_RECOIT". A peine la partie "TXT" écrite, notre formidable IDE nous dit déjà le reste :

Un petit "TAB" nous permet de compléter automatiquement le mot (réflexe que vous prendrez par la suite pour coder de plus en plus rapidement), continuons notre avancement, je veux changer le texte présent dans la textbox, donc accédons à la propriété "**text**", pareil que pour avant, il nous affiche déjà le reste.

Nous voilà donc sur la propriété **text**, nous avons 2 choix : attribuer sa valeur ou l'utiliser ; pour l'utiliser le signe "=" se place avant, pour l'assigner, après.

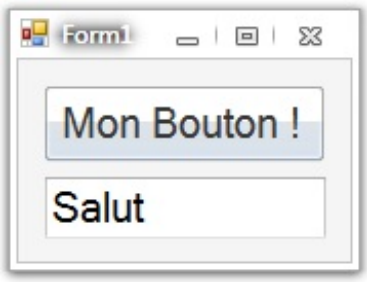
Assigner est utile pour entrer des valeurs, utiliser sa valeur est pratique pour des cas comme les textbox, récupérer ce que l'utilisateur a entré.

Nous allons donc l'assigner, et attribuons lui une valeur texte exemple de la ligne :

Code : VB.NET

```
Me.TXT_RECOIT.Text = "Salut"
```

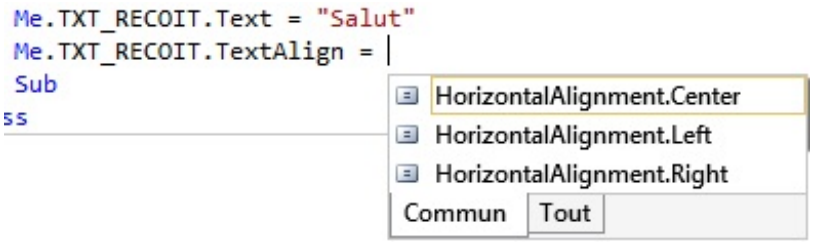
Voilà, lançons le programme pour essayer : un salut s'est affiché dans la textBox !



Nous avons réussi notre premier accès à une propriété coté VB.

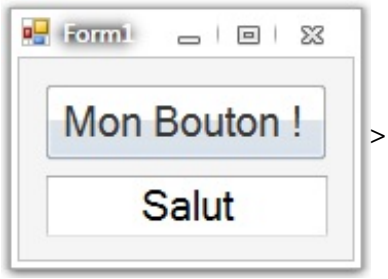
Pourquoi s'arrêter là, alignons ce texte !
Tapons donc "**Me.TXT_RECOIT.Tex**", et là, l'auto complétion me propose ... **textalign** !

Pourquoi s'en priver ! J'écris donc ma propriété textalign, un = pour lui assigner une propriété et là, notre magnifique IDE fait encore tout le travail !



Faites votre choix ! Centré, à gauche ou à droite ?
Je choisis *centré* chef ! On double clique donc sur ce choix !

Et au nouveau test de notre application ...



>

C'est magique, c'est le VB, c'est tout !

With

Voici un petit mot qui va changer votre vie :

With (autrement dit : **avec** en français).

Oui, bon, il va changer votre vie, mais comment ?

Eh bien il va vous arriver de vouloir assigner beaucoup de propriétés à un contrôle ou alors tout simplement définir toutes les composantes d'envoi de mail, de connection réseau, d'impression ...

Bon, restons dans le cas basique : j'ai un **Bouton** pour lequel je veux changer la couleur, le texte, la position, la taille ...

Avec ce que je vous ai expliqués vous allez écrire, en toute logique cela :

Code : VB.NET

```
Me.MonboutonPrefere.ForeColor = Color.Red
Me.MonboutonPrefere.Text = "Mon nouveau texte"
Me.MonboutonPrefere.Left = 10
Me.MonboutonPrefere.Top = 10
Me.MonboutonPrefere.Height = 50
Me.MonboutonPrefere.Width = 50
```

En passant, les information **Top** et **Left**, positionne le coin supérieur gauche de votre contrôle à la manière des **inputbox** et **Height** et **Width** respectivement la hauteur et la largeur de votre contrôle.

Bon, avec ce code, votre bouton aurait bien évidemment changé de position, de couleur, de texte, etc...

Mais un peu lourd comme notation ?

Eh bien le mot **With** va rendre tout ça plus lisible (enfin, plus lisible, ça dépend des goûts et habitudes de chacun.

Donc le code ci-dessus avec notre petit **With** (et son **End With** respectif) donnerait :

Code : VB.NET

```
With Me.MonboutonPrefere
            .ForeColor = Color.Red
            .Text = "Mon nouveau texte"
            .Left = 10
            .Top = 10
            .Height = 50
            .Width = 50
        End With
```

Eh oui, le **with** a fait disparaître tous les **Me.MonBoutonPrefere** devant chaque propriété.

Il faut garder le "." avant la propriété.

Vous pouvez bien sûr assigner des propriétés autre qu'au bouton durant le **with**. Un **MonLabel.Text = "Test"** aurait été bien sûr accepté.
Mais je ne vous le conseille tout de même pas, le **with** n'aurait plus son intérêt.

Eh bien, j'espère que ce mot vous aidera ! Bonne chance pour la suite.

Les évènements

Bon, attaquons maintenant réellement les évènements.

Je vous ai déjà expliqué un évènement, le form_load. Eh bien apprenons à les découvrir pour réagir a plein d'autres choses, un clic, une touche, une ouverture, une fermeture, que sais-je encore, les possibilités sont énormes.

Pourquoi ça encore !

Alors un événement s'écrit comme un sub : exemple l'évènement form load.

Code : VB.NET

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

Observez bien sa définition : nous avons tout d'abord le **private sub**,(sachez que le **private** ne nous intéresse pas pour le moment, je l'expliquerai plus tard en annexes). Nous avons donc cette définition que nous connaissons bien, puis le nom du **Sub** appelé ici **form1_load**.

Pourquoi ce nom ?

Tout simplement parce que la fenêtre s'appelle form1 et l'évènement load. =)

Je peux donc l'appeler autrement ?

Bien sûr mais je vous conseille de vous habituer a ces noms, ils sont plus pratiques, ce sont ceux que l'assistant (assistant que nous avons utilisé en double-cliquant sur la fenêtre) qui les crées automatiquement.

Continuons, nous avons entre parenthèses les **arguments** de ce **sub**, ces arguments sont indispensables ! Vous ne pouvez pas les supprimer ! Ce sont des arguments que la fenêtre passera automatiquement à ce **Sub** lorsqu'il sera appelé, ils nous seront inutiles pour le moment mais plus tard vous en verrez l'utilité.

Code : VB.NET

```
Handles MyBase.Load
```

Voilà notre salut ! Cette fin d'instruction avec ce mot clé : **Handles** . Ce mot clé peut se traduire par "écoute" suivi de l'évènement écouté. Ici il écoute le chargement de la fenêtre.

Donc si vous avez bien compris je résume : ce **sub** sera pénétré lors du chargement de la fenêtre, et maintenant nous savons pourquoi : car un événement attends que le chargement de la fenêtre s'effectue !

Créer nos évènements

Eh bien attelons nous de suite à la tâche :

Vous voulez peut-être réagir à d'autres occasions qu'au chargement de cette fenêtre, pourquoi ne pas réagir sur le clic du bouton ?

Eh bien allons-y ! Comme pour générer l'évènement **form load**, double cliquons sur notre bouton !

Automatiquement l'IDE me crée :

Code : VB.NET

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

End Sub
```

Comme pour le **form_load**, plaçons les instructions voulues dedans, feignant de nature, je déplace simplement celles que nous avions écrites dans le **form_load** :

Code : VB.NET

```
Private Sub BT_ENVOI_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BT_ENVOI.Click
        Me.TXT_RECOIT.Text = "Salut"
        Me.TXT_RECOIT.TextAlign = HorizontalAlignment.Center
End Sub
```

Testons voir : Avant, rien, après le clic, le message s'affiche, nous avons réussi !

Vous l'aurez compris, le double-clic sur un objet côté design crée son évènement le plus courant je dirais, pour un bouton : le clic, pour une textbox : le changement de texte etc...

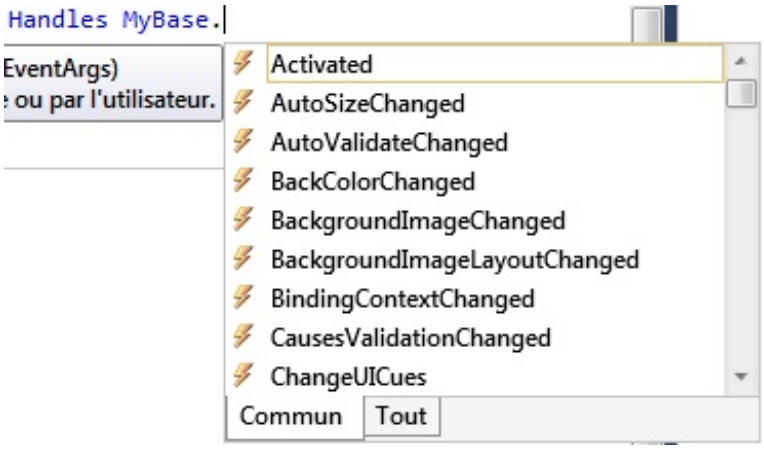
Mais pourquoi se limiter à cela puisque il existe des dizaines d'évènements différents pour chaque objet ?
Allons réagir manuellement à tous ces évènements !

Les mains dans le cambouis !

Créons donc nos évènements !

Enfin modifions-les plutôt ! Eh oui je vous l'ai dit je suis faignant, pourquoi les créer puisque un assistant le fait pour nous, enfin, fait la moitié du travail ! !

Double cliquons donc sur la fenêtre, l'évènement **form_load** se crée (s'il n'y était pas), intéressons-nous au **Handles**, supprimons le **.load** de la fin, plaçons-nous sur la fin du mot **MyBase** et écrivons un point : ".", voilà la liste d'évènements de l'objet fenêtre s'ouvrir à nous :



Eh bien, choisissons l'évènement **MouseClick**, qui (je pense que vous l'avez compris) se déclenche lors du clic de la souris sur la fenêtre.

Et renommons ce sub. Eh oui c'est bien beau d'avoir un nouvel événement mais il est toujours appelé en tant que **form_load**, si vous ne changez pas ce nom vous allez très vite ne plus penser qu'il réagit au clic de la souris, prenez donc l'habitude dès maintenant de le renommer.

Personnellement pour moi ce sera : **form1_MouseClick**

Replaçons maintenant le code que nous avons mis dans l'évènement du clic sur le bouton dans ce nouvel évènement.

Essayons, effectivement lors du clic de la souris sur la fenêtre (pas le bouton) le texte s'affiche ! Encore réussi !

Amusez-vous avec ces évènements, essayez-en, soyez amis avec .

Mini-TP : calcul voyage

Tout d'abord, petite dédicace à mon prof d'info, je le remercie pour ses efforts pour me supporter .

Nous passons à un mini-TP pour utiliser les évènements et ce que nous avons vu précédemment.

Cahier des charges

Bon, voici mes consignes : je voudrais que vous créiez un programme qui va calculer le coût de revient d'un voyage en voiture.

Il prendra en compte

- La consommation de la voiture (l/100km)
- le nombre de kilomètres
- le prix de l'essence (en euros)

Que l'utilisateur entrera dans des textbox et l'appui sur un bouton affichera le résultat.

Je ne sais pas quoi vous dire de plus ?
Non, n'insistez pas vous n'aurez pas la fonction qui calcule ce coût, c'est à vous de faire un peut marcher vos méninges, c'est aussi ça la programmation !

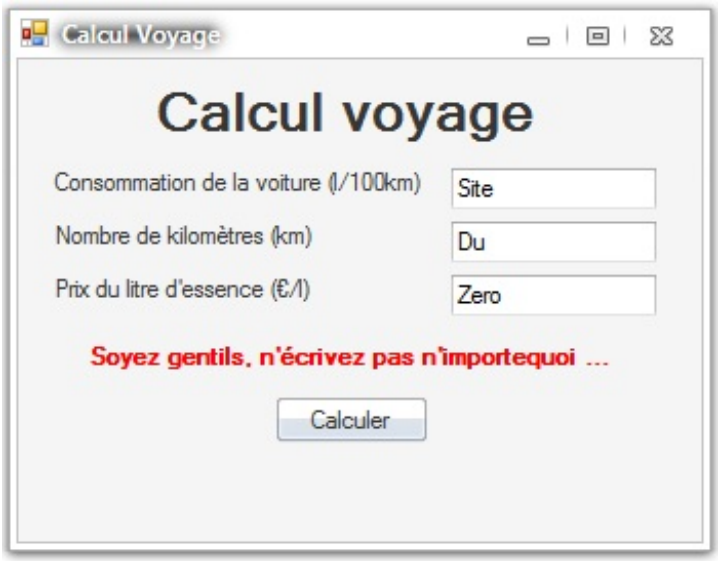
Et je veux que ce programme réagisse aussi aux utilisateurs qui s'amusent a entrer n'importe quoi .

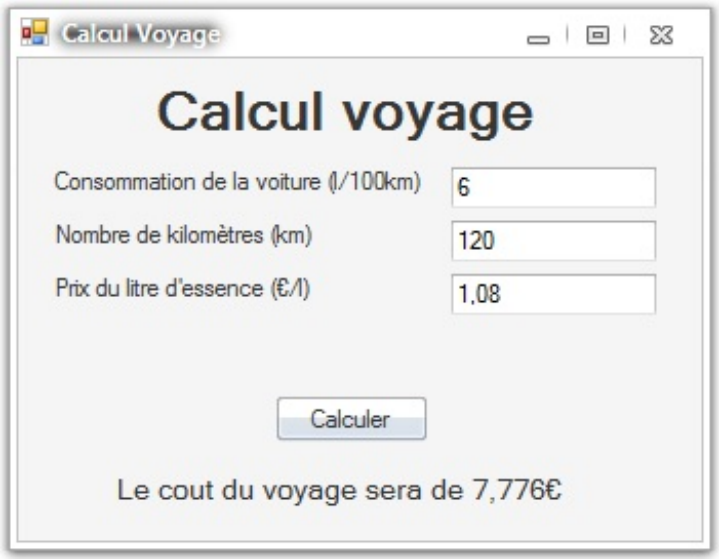
Bonne chance !

Correction

J'espère que vous avez trouvé tous seuls, je vous avais tout expliqués ! Bon je vous montre !

Tout d'abord les résultats :





Bon l'explication des objets : j'ai placé 3 textbox, une pour chaque valeur a entrer.
Leurs noms sont respectivement **TXT_CONSOMMATION, TXT_NBKM, TXT_PRIXESS**.

Puis des labels pour expliquer à quoi elles correspondent (je n'ai pas donné de noms particuliers aux labels puisque je n'agirais pas dessus pendant le programme, alors autant laisser comme ils sont.

Ensuite je leur ai attribués une propriété **text**, pour afficher le texte que vous voyez. Idem pour le titre sauf que j'ai modifié sa propriété **font size** pour le grossir.

Côté bouton, son nom est **BT_CALCUL**, j'ai écrit le texte "calculer" dedans.

Il reste 2 labels : un écrit en rouge qui est là pour les erreurs : j'ai caché ce label en utilisant la propriété **visible = false**, je ne le ferai apparaître que lors des erreurs.

Le dernier est celui qui contiendra le résultat, j'ai nommé ... **LBL_COUT**

Voilà pour ce qui est du design, passons au VB !

Secret (cliquez pour afficher)

Code : VB.NET

```
Public Class Form1

    Private Sub BT_CALCUL_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles BT_CALCUL.Click
        If Verification() Then
            Me.LBL_COUT.Text = "Le cout du voyage sera de " &
Calcul(Me.TXT_CONSOMMATION.Text, Me.TXT_NBKM.Text,
Me.TXT_PRIXESS.Text) & "?"
        Else
            Me.LBL_ERR.Visible = True
        End If
    End Sub

    ''' <summary>
    ''' Verifie les 3 textbox de la page, regarde si elles sont
remplies et si des nombres ont été entrés
    ''' </summary>
    ''' <returns>Vrai si pas d'erreur, faux si une
erreur</returns>
    ''' <remarks></remarks>
    Function Verification() As Boolean
```

```
            Dim Bon As Boolean = True
            If Me.TXT_CONSOMMATION.Text Is Nothing Or Not
    IsNumeric(Me.TXT_CONSOMMATION.Text) Then
                Bon = False
            End If
            If Me.TXT_NBKM.Text Is Nothing Or Not
    IsNumeric(Me.TXT_NBKM.Text) Then
                Bon = False
            End If
            If Me.TXT_PRIXESS.Text Is Nothing Or Not
    IsNumeric(Me.TXT_PRIXESS.Text) Then
                Bon = False
            End If
            Return Bon
        End Function

        ''' <summary>
        ''' Calcule le prix d'un voyage en fonction de la
    consommation, du prix de l'essence, et du nb de kilomètres
        ''' </summary>
        ''' <param name="Consommation">Consommation</param>
        ''' <param name="NbKm">Distance parcourue</param>
        ''' <param name="PrixEss">Prix du kérosène</param>
        ''' <returns>Le cout en double</returns>
        ''' <remarks></remarks>
        Function Calcul(ByVal Consommation As Double, ByVal NbKm As
    Double, ByVal PrixEss As Double) As Double
            Dim Cout As Double

            Cout = ((NbKm / 100) * Consommation) * PrixEss

            Return Cout
        End Function

    End Class
```

Examinons notre évènement : l'appui sur le bouton. Évènement que j'ai crée grâce à l'assistant, en double cliquant dessus.

Dans cet évènement j'utilise ma fonction Verification(), si le résultat est vrai, j'utilise ma fonction calcul() en lui passant comme arguments les valeurs des 3 textbox et j'écris le résultat sous la forme "Le cout du voyage sera de XX?".

Si la fonction Verification() renvoie faux, j'affiche le message d'erreur.

Passons donc aux fonctions :
La fonction Vérification(), cette fonction est spécifique à ce programme, je ne pourrai pas l'utiliser ailleurs, pourquoi ? Tout simplement car dedans j'accède à des objets qui sont sur cette feuille uniquement :

Code : VB.NET

```
Dim Bon As Boolean = True
If Me.TXT_CONSOMMATION.Text Is Nothing Or Not
IsNumeric(Me.TXT_CONSOMMATION.Text) Then
     Bon = False
End If
```

Ce code crée un booleen à true au début, il vérifie si le texte entré **is nothing** donc est nul ou **not isnumeric()** donc si ce n'est pas un numérique.

Si l'une de ces 2 conditions est présente (autrement dit une erreur lors de l'entrée des caractères), le booléen passe à false. Ce booléen est finalement retourné.

Passons à la fonction calcul, fonction qui effectue uniquement le calcul nécessaire, cette fonction pourra être réutilisée puisque elle a une forme universelle. Je m'explique : on lui passe les valeurs nécessaire et elle effectue le calcul, ce n'est pas elle qui va chercher les valeurs dans les textbox, donc on peut l'utiliser pour lui donner d'autres valeurs.

la ligne essentielle :

Code : VB.NET

```
Cout = ((NbKm / 100) * Consommation) * PrixEss
```

Pourquoi toutes les valeurs numériques que tu utilises sont en **double** ?

Eh bien parce que le type **integer** ne prends pas en compte les virgules et donc dans un programme comme celui-ci le **double** est nécessaire.

Voilà, j'espère que ce TP n'était pas trop dur !

Si vous n'avez pas le même code que moi, pas de panique ! Il y a une infinité de possibilité d'arriver au même résultat sans faire les mêmes choses.

Vous pourriez le faire évoluer ce programme par exemple

- Gérer un message d'erreur pour chaque txtbox
- Message d'erreur personnalisé (vide ou mauvaise valeur)
- Un bouton "effacer" qui remet à 0 toutes les valeurs et cache les message d'erreurs

Dites vous que ce programme est déjà très bien, il vous apprend à interagir avec les contrôles, utiliser les fonctions, arguments, retours et réaction à une possible erreur. Vous avancez vite !

Les contrôles spécifiques

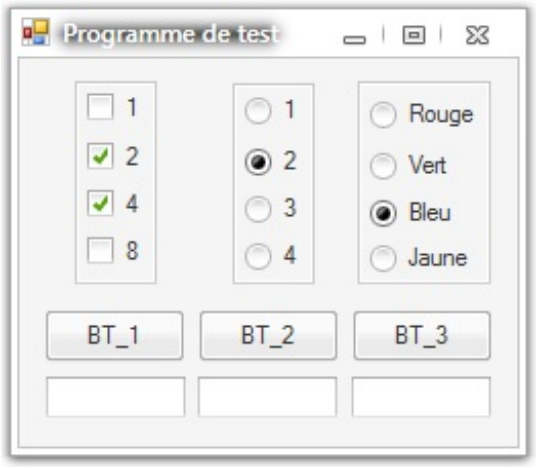
Vous savez désormais comment vous servir des contrôles basiques : les textbox, les labels, les boutons etc... Mais qu'en est-il pour les contrôles plus spécifique mais non sans être inintéressant.

Vous allez sûrement vouloir faire un peu plus que mettre des boutons et des TextBox dans tout votre programme.

Je parle des checkbox et des boutons radio entre autres. Comment s'en servir ?

CheckBox, BoutonRadio

Eh bien créons d'abord un nouveau projet (gardez le TP voyage dans un coin ca pourrait toujours vous servir), dans ce nouveau projet on va essayer de faire quelque chose comme ça :



Je vous donne les noms des composants que j'ai mis :

Des checkbox à gauche de haut en bas: • CHK_1 • CHK_2 • CHK_4 • CHK_8	Des RadioBoutons au centre de haut en bas • RB_1 • RB_2 • RB_3 • RB_4	Des RadioBoutons à droite de haut en bas : • RB_ROUGE • RB_VERT • RB_BLEU • RB_JAUNE
Bouton BT_1	Bouton BT_2	Bouton BT_3
Txtbox TXT_CHK	Txtbox TXT_RBNB	Txtbox TXT_RBCOL

Dictateur va !

Mais non, du calme ! Je vous donne les noms de mes contrôles pour que nous puissions tous travailler sur de bonnes bases, je ne me prend absolument pas pour dieu .

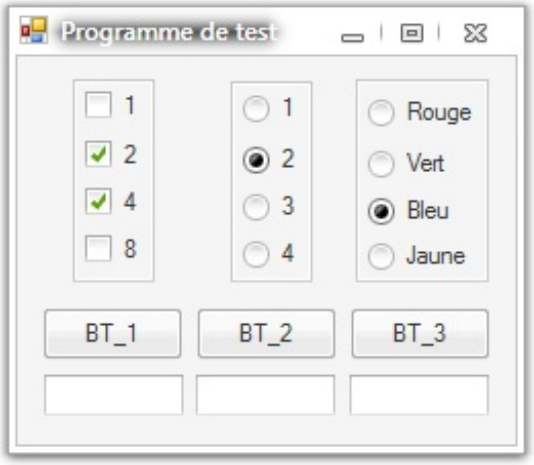
Bon. Si vous testez ce petit programme, vous pouvez cliquer sur les cases, elles s'allument bien seulement, problème du côté des boutons radios : cliquer sur n'importe lequel enlève l'autre même si ce dernier n'est pas dans la même colonne ... Eh oui, l'IDE n'est pas intelligent, il ne sait pas ce que nous voulons faire.

Comment faire ?

Eh bien retournez sur votre IDE et cherchez le contrôle **groupbox**. Entourez grâce à 2 **groupbox** vos 2 colonnes de **radioboutons** et allez dans les propriétés des 2 **groupbox** que vous venez de créer pour retirer le texte qu'elles contiennent : elles seront invisibles.

Lors de l'application de la propriété **Enabled** ou **Visible** sur la **GroupBox**, tous les éléments contenus à l'intérieur de celle-ci (appelés ses **enfants**) seront affectés par la propriété.

Une fois cela fait, retestez le programme et là : magie



On peut sélectionner un bouton dans chaque colonne.

La pratique

Bon le côté design fonctionne, on va passer à l'accès aux propriétés.

Donc allons côté code VB en double cliquant sur **BT_1**, ce qui nous créera notre évènement de clic sur le bouton.

Dans cet évènement je vais vous demander de faire la somme des **checkbox** cochées. Donc la propriété qui régit l'état d'une **checkbox** est ...

Checked ! (siii)

Bon, écrivons donc ce code :

Code : VB.NET

```
Me.CHK_1.Checked
```

Pour récupérer l'état de la première **checkbox**. Cette propriété est définie par **true** ou **false**. C'est donc un **Booléen**, vous avez dû vous en rendre compte lorsque vous avez inscrit cette ligne, l'IDE vous a affiché une infobulle.

Nous allons donc facilement pouvoir faire une boucle if :

Code : VB.NET

```
if Me.CHK_1.Checked then
```

Cette boucle sera pénétrée si la case 1 est cochée.

Donc, vous avez toutes les cartes en main. Écrivez dans la **txtbox TXT_CHK** la somme des cases cochées.

Secret (cliquez pour afficher)

Code : VB.NET

```
Private Sub BT_1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BT_1.Click
        Dim Somme As Integer = 0
        If Me.CHK_1.Checked Then
            Somme = Somme + 1
        End If
        If Me.CHK_2.Checked Then
            Somme = Somme + 2
        End If
        If Me.CHK_4.Checked Then
            Somme = Somme + 4
        End If
        If Me.CHK_8.Checked Then
            Somme = Somme + 8
        End If
        Me.TXT_CHK.Text = Somme
    End Sub
```

Et voilà le code permettant de faire cela.

C'était pas sorcier !

Je vous dis un secret : la propriété pour voir quel **boutonradio** est coché est la même !

Alors, à vos claviers ! Écrivez dans la seconde **txtbox** quel bouton a été coché et dans la dernière la couleur sélectionnée !

Je vous laisse quand même réfléchir !

Solution

Secret (cliquez pour afficher)

Code : VB.NET

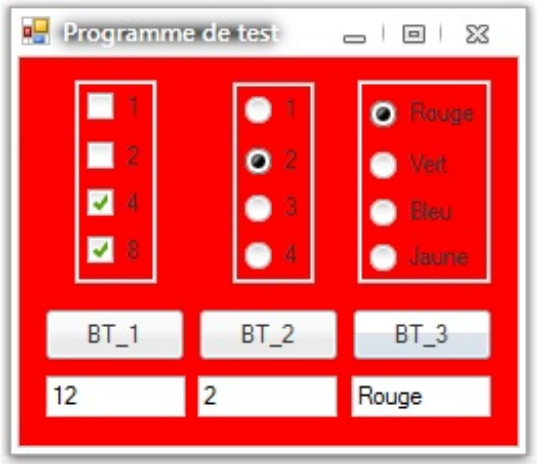
```
Public Class Form1

    Private Sub BT_1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles BT_1.Click
        Dim Somme As Integer = 0
        If Me.CHK_1.Checked Then
            Somme = Somme + 1
        End If
        If Me.CHK_2.Checked Then
            Somme = Somme + 2
        End If
        If Me.CHK_4.Checked Then
            Somme = Somme + 4
        End If
        If Me.CHK_8.Checked Then
            Somme = Somme + 8
        End If
        Me.TXT_CHK.Text = Somme
    End Sub

    Private Sub BT_2_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles BT_2.Click
        If Me.RB_1.Checked Then
            Me.TXT_RBNB.Text = Me.RB_1.Text
        End If
        If Me.RB_2.Checked Then
            Me.TXT_RBNB.Text = Me.RB_2.Text
        End If
        If Me.RB_3.Checked Then
            Me.TXT_RBNB.Text = Me.RB_3.Text
        End If
        If Me.RB_4.Checked Then
            Me.TXT_RBNB.Text = Me.RB_4.Text
        End If
    End Sub

    Private Sub BT_3_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles BT_3.Click
        If Me.RB_BLEU.Checked Then
            Me.TXT_RBCOL.Text = Me.RB_BLEU.Text
            Me.BackColor = Color.Blue
        End If
        If Me.RB_JAUNE.Checked Then
            Me.TXT_RBCOL.Text = Me.RB_JAUNE.Text
            Me.BackColor = Color.Yellow
        End If
        If Me.RB_ROUGE.Checked Then
            Me.TXT_RBCOL.Text = Me.RB_ROUGE.Text
            Me.BackColor = Color.Red
        End If
        If Me.RB_VERT.Checked Then
```

```
                Me.TXT_RBCOL.Text = Me.RB_VERT.Text
                Me.BackColor = Color.Green
            End If
        End Sub
    End Class
```

Donc voici mon code, ce n'est pas bien j'ai mis aucun commentaire !

Bon, il est un peu laborieux puisqu'il vérifie toutes les **chkbox** une par une...

Mais bon, il fonctionne et vous avez réussi à accéder et réagir aux checkbox et radiobuttons. Essayez donc de le simplifier à coup de **IIF** !



Petit plus : la couleur. Vous auriez dû vous douter que je ne mettais pas des couleurs juste comme ça xD, et la propriété vous auriez pu la trouver tout seul !

Code : VB.NET

```
    Private Sub BT_3_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles BT_3.Click
            If Me.RB_BLEU.Checked Then
                Me.TXT_RBCOL.Text = Me.RB_BLEU.Text
    Me.BackColor = Color.Blue
            End If
            If Me.RB_JAUNE.Checked Then
                Me.TXT_RBCOL.Text = Me.RB_JAUNE.Text
    Me.BackColor = Color.Yellow
            End If
            If Me.RB_ROUGE.Checked Then
                Me.TXT_RBCOL.Text = Me.RB_ROUGE.Text
    Me.BackColor = Color.Red
            End If
            If Me.RB_VERT.Checked Then
                Me.TXT_RBCOL.Text = Me.RB_VERT.Text
    Me.BackColor = Color.Green
            End If
        End Sub
```

Et l'IDE vous donne automatiquement la liste des couleurs disponibles quand vous écrivez le signe égal "=", il faut juste connaître les noms anglais.

Bon, vous savez désormais accéder et utiliser les **checkbox** et aux **radiobuttons** !

Les ComboBox

Bon, attaquons les **ComboBox** (aussi appelé DropDownList), sous ce nom barbare se trouve un élément que vous retrouvez partout, même sur le SDZ :

Eh oui, c'est cette petite boite déroulante une **ComboBox**. Nous allons apprendre à la remplir et à réagir avec.

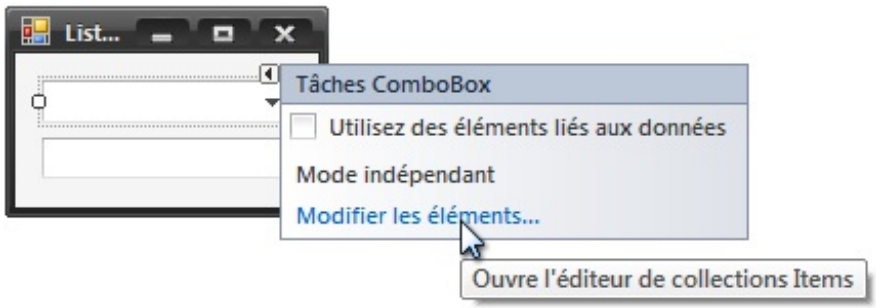
Créez donc cette petite fenêtre : une **ComboBox** nommée **CB_CHOIX**, et une **textbox** appelée **TXT_CHOIX**.

Cette fois, au lieu d'utiliser un bouton pour déclencher l'évènement, nous allons utiliser l'évènement propre de la **combobox**. Cet évènement se déclenche lors du changement de sélection.

Tout d'abord il faut attribuer des valeurs à la **combobox**, 2 choix s'offrent à nous : la manuelle (en dur dans le code) ou l'automatique (grâce a l'assistant de l'IDE). Je vais d'abord vous expliquer l'automatique puis la manuelle qui offre beaucoup plus de possibilités.

Méthode assistée

Pour la méthode avec assistant. Lors du clic sur la **combobox** (dans l'IDE), elle apparait en "sélectionnée" et une petite flèche apparait en haut à droite de cette sélection :



Cliquez maintenant sur **Modifier les éléments** pour lui en attribuer.

Méthode manuelle

La seconde méthode nous amène coté VB, double-cliquez sur la fenêtre pour créer l'évènement **onload**.

Une technique est de créer un tableau contenant les valeurs et de "lier" ce tableau à la combobox : créons tout d'abord notre tableau.

Code : VB.NET

```
Dim MonTableau(9) As Integer
 For i As Integer = 0 To 9
     MonTableau(i) = i + 1
 Next
```

Rempli ici avec des valeurs allant de 1 à 10.

L'instruction pour lier cette **combobox** (valable pour les **listbox** et autres) est :

Code : VB.NET

```
Me.CB_CHOIX.DataSource = MonTableau
```

Donc si l'on écrit tout ça dans le main, on obtient une liste déroulante avec des nombres allant de 1 à 10.

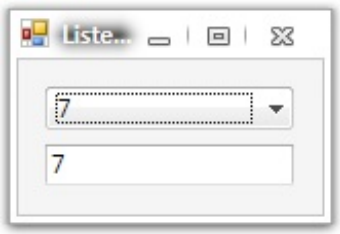
Nous allons écrire la valeur récupérée dans la **textbox** lors du changement de choix dans la **combobox**, la propriété utilisée pour récupérer la valeur sélectionnée est **SelectedValue** (je vous laisse faire cette modification).

Code : VB.NET

```
Private Sub CB_CHOIX_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
CB_CHOIX.SelectedIndexChanged
        Me.TXT_CHOIX.Text = Me.CB_CHOIX.SelectedValue
    End Sub
```

Et voilà !

Dernière chose avant le test : retournez côté design, recherchez et attribuez la propriété **DropDownList** à la propriété **DropDownStyle**. Pourquoi ? Cette propriété empêche l'utilisateur d'écrire lui-même une valeur dans cette **combobox**, il n'a que le choix entre les valeurs disponibles, dans le cas contraire, il aurait pu utiliser la ComboBox comme une textbox.



Après le test, nous voyons que tout fonctionne, nous avons réussi à accéder et à remplir une **combobox** !

MicroTP

Bon, pour vérifier vos connaissances sur les accès aux propriétés et l'utilisation de nouveau contrôles, je vais vous demander de réaliser un petit programme contenant : Une **ProgressBar** et une **TrackBar**.

Le déplacement de la **trackbar** par l'utilisateur se répercutera sur le remplissage de la **progressbar** : si la **trackbar** est au milieu, la **progressbar** aussi.

Ce petit TP vous apprendra a trouver tout seul les propriétés utiles des contrôles. Il va falloir se faire à cette pratique, c'est 50% du travail d'un développeur : trouver comment faire ce qu'il souhaite sans que personne ne lui montre. Ne vous inquiétez pas, l'IDE vous expliquera l'utilité de chaque propriété.

Bonne chance !

Résultat

Alors, une seule ligne côté VB a ajouter dans l'évènement de la **trackbar** :

Code : VB.NET

```
Private Sub TKB_IN_Scroll(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles TKB_IN.Scroll
        Me.PGB_OUT.Value = Me.TKB_IN.Value
    End Sub
```

Eh oui ! Les propriétés a utiliser étaient **value**. Vous avez dû avoir des surprises au premiers test, du genre la **progressbar** ne va pas jusqu'au bout alors que le **trackbar** y est ...

Alors, comment résoudre ce problème pour ceux qui n'ont pas trouvé !

Eh bien regardez un peu du côté de la propriété **Maximum** de ces 2 contrôles. Si elle n'est pas la même ça risque de ne pas aller . Autre chose : Je vous conseille de mettre la **tickfrequency** (autrement dit, le pas) de la **trackbar** à **0**, plus de "tirets" et donc la **progressbar** est mise à jour en temps réel.

Testez les propriétés, par exemple la propriété **style** de la **progressbar** peut être intéressante

Eh bien, pas trop dur !

Les timers

A quoi ca va nous servir ?

Vous devez bien vous demander à quoi ca va nous servir même si le nom explique déjà beaucoup.

Un timer nous sera très utile pour effectuer des actions temporelles et réagir à des évènements temporels.

Exemple : notre horloge que nous avons fait dans la partie 1 qui prenait la date actuelle et l'affichait, eh bien avec un timer on pourrait prendre la date actuelle et ajouter une seconde toutes les secondes. Même effet mais pas la même façon.

Autre exemple : faire déplacer une image ou un objet tout les x millisecondes (utile pour les jeux ou animations)

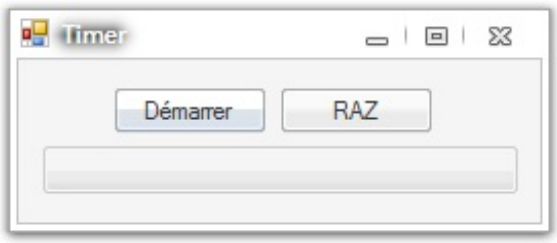
Ce **timer** est un contrôle comme n'importe quel bouton ou textbox, mais au lieu de pouvoir le placer ou l'on veut dans la fenêtre, il se met "en dehors" de cette fenêtre puisqu'il n'est pas visible à l'exécution.

Apprenons dès maintenant à l'utiliser.

Créer son premier timer

Créons notre premier **timer** : double-cliquons donc sur son contrôle pour le voir se placer en bas de notre fenêtre design.

Essayez de construire le reste de l'application comme moi :



Donc, nous avons 2 boutons : un démarrer nommé BT_DEMAR, un RAZ nommé ..... BT_RAZ.
Une progressbar : PGB_TIM, et un timer (non visible à l'image) nommé TIM_TIM.

J'explique ce que notre programme va faire : lors de l'appui sur le bouton démarrer, la progressbar va progresser jusqu'au bout de manière linéaire et à une certaine vitesse, sur RAZ, elle va retourner à 0.

Le **timer** contient 2 propriété essentielles : **enabled**, comme tout les autres contrôles détermine s'il est activé ou non, et la propriété **interval** (ce n'est pas une marque de cigarettes, non) cette propriété va déterminer l'intervalle entre 2 actions du **timer** (exprimée en ms).

Mettons donc pour ce tp **20ms** d'intervalle.

A chaque fois que ce temps sera écoulé, l'évènement du **timer** nommé **tick** se déclenchera. Pour créer cet évènement sur l'assistant, double-cliquez sur le **timer**, en bas. Faites de même pour les évènements des 2 boutons.

Nous avons donc les 3 évènements de crées dans notre code : le **timer** et les 2 boutons.

Je pense que vous êtes capables de faire cet exercice seul, avec tout ce que vous savez mais je vais quand même vous le faire.

Code : VB.NET

```
Public Class Form1
    Private Sub BT_DEMAR_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles BT_DEMAR.Click
        'Si le bouton démarrer est enfoncé, on active le timer, on
désactive ce bouton et on active RAZ
        Me.TIM_TIM.Enabled = True
        Me.BT_DEMAR.Enabled = False
        Me.BT_RAZ.Enabled = True
    End Sub

    Private Sub TIM_TIM_Tick(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles TIM_TIM.Tick
        'Si la progressbar est arrivée au bout, on désactive le
timer, on réactive le bouton démarrer
        If Me.PGB_TIM.Value = 100 Then
            Me.TIM_TIM.Enabled = False
            Me.BT_DEMAR.Enabled = True
        Else
            'Augmente de 1 la progressbar
            Me.PGB_TIM.Value = Me.PGB_TIM.Value + 1
        End If
    End Sub

    Private Sub BT_RAZ_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles BT_RAZ.Click
        'Si le bouton RAZ est enfoncé, on remet la progressbar à 0,
on désactive le timer,
```

```
            'on active le bouton démarrer et on désactive le bouton RAZ
            Me.PGB_TIM.Value = 0
            Me.TIM_TIM.Enabled = False
            Me.BT_DEMAR.Enabled = True
            Me.BT_RAZ.Enabled = False
        End Sub
    End Class
```

J'ai fait l'effort de commenter le code pour une fois 

Bon, je pense que ce n'était pas si dur que ca, vous voyez que je me sers de l'évènement du **timer**, donc déclenché toutes les 20ms dans notre cas pour ajouter 1 à la **value** de la **progressbar**. Si la **value** arrive à 100, on l'arrête.

Je pense que vous avez compris que si je diminue l'intervalle, la **progressbar** progressera plus vite.

TP : la banderole lumineuse

Petit TP : la banderole lumineuse.

Bon le nom n'est pas très imaginatif ... Je sais.

Le but de ce TP va être d'allumer différents **RadioBoutons** (une dizaine) au rythme du **timer**, les faire défiler en gros. J'ai pris des **radioboutons** et pas des **chkbox**, parce que les **radiobouton** n'ont pas besoin d'être décochés, ils le font automatiquement lorsqu'un autre est coché.

Donc un bouton démarrer et arrêter la banderole seront nécessaires.

Et petit plus pour les rapides : une barre pour faire varier la vitesse de ce défilement.

Attention ce TP n'est pas aussi facile qu'il en a l'air !
Essayez de trouver une méthode pour pouvoir gérer aussi bien 10 boutons que 50.

Petite astuce : il va falloir faire un tableau ... Mais de quoi ? *That's the question*.

A vos claviers !

Solution

Bon alors, tout d'abord à quoi il ressemble mon programme ?

A ça :

Il y a bien les 10 Radioboutons.

Maintenant le code :

Secret (cliquez pour afficher)

Code : VB.NET

```
Public Class Form1
    Private Sub TIM_TIM_Tick(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles TIM_TIM.Tick
        Dim Tourne As Boolean = True
        Dim Bouton As Integer = 0

        'Rassemble tous les radioboutons dans un tableau
        Dim RB(9) As RadioButton
        RB(0) = Me.RB_1
        RB(1) = Me.RB_2
        RB(2) = Me.RB_3
        RB(3) = Me.RB_4
        RB(4) = Me.RB_5
        RB(5) = Me.RB_6
        RB(6) = Me.RB_7
        RB(7) = Me.RB_8
```

```
            RB(8) = Me.RB_9
            RB(9) = Me.RB_10

            While Tourne
                'Si on est arrivé au bout du tableau, on sort de cette
boucle
                If Bouton = 10 Then
                    Tourne = False
                Else
                    'Si le bouton actuellement parcouru est activé
                    If RB(Bouton).Checked Then
                        'Et si ce n'est pas le dernier
                        If RB(Bouton) IsNot RB(9) Then
                            'on active celui d'après et on sort de la
boucle
                            RB(Bouton + 1).Checked = True
                            Tourne = False
                        Else
                            'Sinon on reprends au premier
                            Me.RB_1.Checked = True
                        End If
                    End If
                    'On incrémente le compteur
                    Bouton = Bouton + 1
                End If
            End While
        End Sub

        Private Sub BT_PLAY_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles BT_PLAY.Click
            Me.TIM_TIM.Enabled = True
            Me.TIM_TIM.Interval = 501 - Me.TKB_VIT.Value * 50
        End Sub

        Private Sub BT_STOP_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles BT_STOP.Click
            Me.TIM_TIM.Enabled = False
        End Sub

        Private Sub TKB_VIT_Scroll(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles TKB_VIT.Scroll
            Me.TIM_TIM.Interval = 501 - Me.TKB_VIT.Value * 50
        End Sub
    End Class
```

Alors je vais expliquer le pricipal :

Vous voyez que dans l'évènement **tick** du **timer** j'ai crée un tableau, mais ce n'est pas un tableau de string ou de **integer**, non c'est un tableau de ... **Dim RB(9) As RadioButton** *RadioButton* !

Eh, oh je savais pas moi !

C'est pour ca que j'ai dit que ce TP était difficile, en cherchant un peu vous auriez pu avoir l'idée, ensuite la mettre en pratique aurait été faisable ...

Bon Ce n'est pas grave vous le saurez maintenant. Donc ce tableau de **radioboutons** ,je le remplit avec mes boutons !

Et donc si vous avec compris, la boucle en dessous est un petit algorithme qui parcours ces boutons et qui retourne au premier une fois arrivé au dernier.

Passons maintenant au changement de vitesse : **Me.TIM_TIM.Interval = 501 - Me.TKB_VIT.Value * 50.** Mais pourquoi ? tout d'abord ma **progressbar** a un **minimum** de 1 et un **maximum** de 10.
Donc, à 1 : 501 - 1 * 50 = 451 et à 10 : 501 - 10 * 50 = 1.

La vitesse change donc bien en fonction de cette barre.

Et pourquoi 501 et pas 500 ?

Parce que 500 - 10 * 50 = 0, et l'**interval** d'un **timer** ne doit jamais être égal à 0 !

Pour finir ce chapitre, je tien à dire que l'amélioration de ce TP peut être effectuée en de multiples points. Tout d'abord, le code lors du Tick du Timer est beaucoup trop conséquent, il faut au contraire qu'il soit le plus petit possible pour ne pas demander trop de mémoire au processeur. Donc les déclarations sont à effectuer au Load.
Et profitez-en pour factoriser ce petit algorithme qui fait défiler les RadioButtons =).

Les menus

Vous savez déjà faire un joli programme avec tout ça !

Mais joli c'est pas assez, il faut qu'il soit magnifique (je rigole, je rigole).
Bon, fini les plaisanteries, nous allons passer aux menus.

Vous savez, les menus, la barre qui est en haut de votre navigateur favori par exemple avec "Fichier", "Édition" ...
Et celle du dessous aussi avec les images (la barre d'outils) !

Eh bien oui, nous allons apprendre à faire cela !

Donc, tout d'abord pour les habitués du VB6 eh bien je peux vous dire que ça va être une partie de plaisir ! L'IDE nous mâche le travail ("*Pré-mâché et pré-digéré ...*").

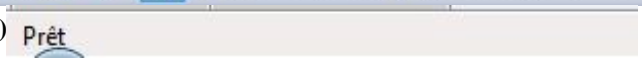
Mettons nous-y tout de suite !

Présentation des menus

Vous devez voir dans votre boîte à outils un sous-menu "**Menus et barres d'outils**", comme vous pouvez le constater, ces objets nous permettrons de créer : des menus **(1)**, une barre d'outils **(2)**, une barre de statut **(3)** et un menu contextuel **(4)** (menu que vous voyez s'afficher lors du clic droit sur la souris).

(1) 

(2)

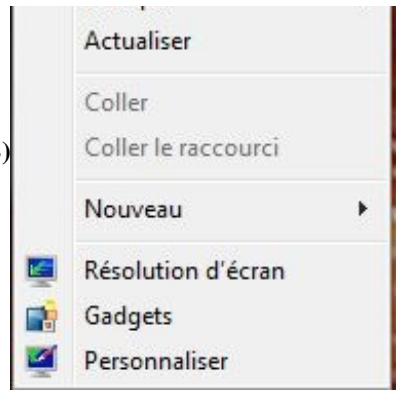
(3) 

Passons tout de suite au menu le plus intéressant : la barre de menu **(1)** !

La barre de menus

Création graphique

(4)

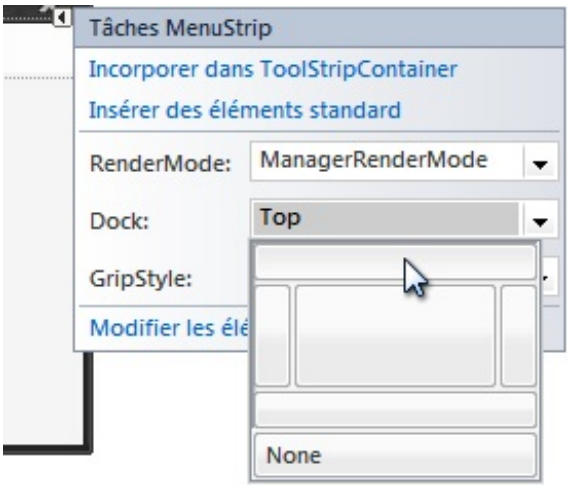


Comme je vous l'ai dit, L'IDE va grandement nous mâcher le travail : un assistant est fourni avec pour créer ces menus.

Mettons-nous y : prenez l'objet **MenuStrip** et insérez-le sur votre feuille (feuille vide de préférence )

Vous voyez donc que ce menu se place automatiquement en haut de votre feuille, vous ne le voulez pas en haut, ~~vous êtes pénibles !~~, pas de problème une propriété permet de choisir la position dans la feuille de ce menu (gauche, droite, etc...), ou un superbe objet : le **ToolStripContainer**.

Cette propriété est **Dock**, et comme notre IDE est gentil, il nous offre la possibilité de paramétrer cette propriété en cliquant sur la petite flèche en haut à droite de notre menu :



Bon, passons au remplissage de ce menu !

Comme vous le voyez, lorsqu'il est sélectionné, le menu vous affiche un petit "Tapez ici" (non, ne sortez pas votre marteau !), comme quoi on ne peut plus facile !

La première "ligne" correspond aux menus principaux (comme fichier, édition ...) Écrivez donc le nom de votre premier menu (pour moi ce sera fichier ). Vous devez voir lors de l'écriture de ce premier menu 2 cases supplémentaires (qui sont également masochistes apparemment), celle du dessous correspond au premier sous-menu de notre premier menu (*Fichier -> Nouveau* par exemple), la seconde est celle qui nous permet de créer un second menu.

Ne grillons pas les étapes, remplissons déjà notre premier menu !

Pour moi ce sera "**Reset**" et celui en dessous "**Quitter**".

Il y a encore des *Tapez* qui apparaissent, je fais quoi ?

Eh bien ces cases permettent de créer des sous-menus qui vous offrent plusieurs choix.

Comme vous allez le voir, la possibilité de créer notre menu entièrement personnalisé est bien réelle !

Bon, je crée un second menu, faites de même :

Puis, pour finir un petit label au centre de la feuille : **LBL_TEXTE**.

Evènements

Maintenant, attaquons la gestion des évènements !

Ces évènements seront créés grâce à l'assistant Visual studio comme le clic sur un bouton : un double clic sur le sous-menu que vous voulez gérer, le code s'ajoute automatiquement :

Code : VB.NET

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

End Sub
```

Faites cela pour tous les sous-menus (sinon à quoi ça sert de les créer).

Je peux le faire sur les menus comme "**Fichier**" aussi ?

Oui bien sûr, si vous en trouvez l'utilité !

Bon, voilà donc le code dûment rempli :

Code : VB.NET

```
Public Class Form1

    Private Sub ResetToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
ResetToolStripMenuItem.Click
        Me.LBL_TEXTE.Text = ""
    End Sub

    Private Sub QuitterToolStripMenuItem_Click(ByVal sender As
```

```
        System.Object, ByVal e As System.EventArgs) Handles
        QuitterToolStripMenuItem.Click
            End
        End Sub

        Private Sub BonjourToolStripMenuItem_Click(ByVal sender As
        System.Object, ByVal e As System.EventArgs) Handles
        BonjourToolStripMenuItem.Click
            Me.LBL_TEXTE.Text = "Bonjour !"
        End Sub

        Private Sub AuRevoirToolStripMenuItem_Click(ByVal sender As
        System.Object, ByVal e As System.EventArgs) Handles
        AuRevoirToolStripMenuItem.Click
            Me.LBL_TEXTE.Text = "Au revoir."
        End Sub

        Private Sub CiaoToolStripMenuItem_Click(ByVal sender As
        System.Object, ByVal e As System.EventArgs) Handles
        CiaoToolStripMenuItem.Click
            Me.LBL_TEXTE.Text = "Ciao."
        End Sub

        Private Sub ByeByeToolStripMenuItem_Click(ByVal sender As
        System.Object, ByVal e As System.EventArgs) Handles
        ByeByeToolStripMenuItem.Click
            Me.LBL_TEXTE.Text = "Bye bye."
        End Sub

        Private Sub AstalavistaBabyToolStripMenuItem_Click(ByVal sender
        As System.Object, ByVal e As System.EventArgs) Handles
        AstalavistaBabyToolStripMenuItem.Click
            Me.LBL_TEXTE.Text = "Astalavista baby !"
        End Sub

    End Class
```

Eh oui, tant de lignes pour si peu ! Je pense que vous avez compris ~~l'utilité~~ ce que doit faire le programme : lors du clic sur une sous-menu de "**Afficher**", il affiche ce texte, lors du clic sur **Reset**, il efface, et lors du clic sur **Quitter**, il quitte le programme (le **end** effectuant cette action).

Bon, vous vous souvenez des **MsgBox** ?

Eh bien elles vont nous être utiles ici : nous allons mettre une confirmation de sortie du programme.

Je pense que vous êtes capables de le faire tout seul mais bon, je suis trop aimable :

Code : VB.NET

```
        If MsgBox("Souhaitez-vous vraiment quitter ce magnifique programme ?
        ", 36, "Quitter") = MsgBoxResult.Yes Then
            End
        End If
```

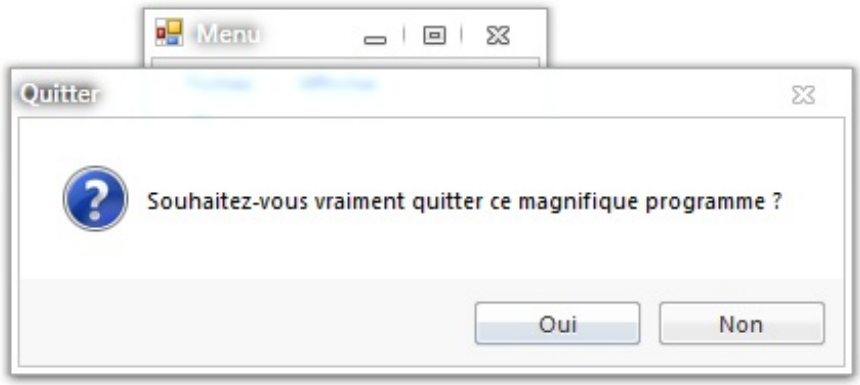
Pourquoi 36 en second argument ?

Vous ne vous en souvenez pas ? Je vous redonne le tableau :

Membre	Valeur	Description

OKOnly	0	Affiche le bouton OK uniquement.
OKCancel	1	Affiche les boutons OK et Annuler.
AbortRetryIgnore	2	Affiche les boutons Abandonner, Réessayer et Ignorer.
YesNoCancel	3	Affiche les boutons Oui, Non et Annuler.
YesNo	4	Affiche les boutons Oui et Non.
RetryCancel	5	Affiche les boutons Réessayer et Annuler.
Critical	16	Affiche l'icône Message critique.
Question	32	Affiche l'icône Requête d'avertissement.
Exclamation	48	Affiche l'icône Message d'avertissement.
Information	64	Affiche l'icône Message d'information.
DefaultButton1	0	Le premier bouton est le bouton par défaut.
DefaultButton2	256	Le deuxième bouton est le bouton par défaut.
DefaultButton3	512	Le troisième bouton est le bouton par défaut.
ApplicationModal	0	L'application est modale. L'utilisateur doit répondre au message avant de poursuivre le travail dans l'application en cours.
SystemModal	4096	Le système est modal. Toutes les applications sont interrompues jusqu'à ce que l'utilisateur réponde au message.
MsgBoxSetForeground	65536	Spécifie la fenêtre de message comme fenêtre de premier plan.

Et voilà votre programme qui affiche ce que vous voulez et qui vous demande une confirmation de fermeture :



Les différents contrôles des menus

Je viens de vous montrer un menu classique avec du texte comme contrôle, mais vous en voulez plus n'est-ce pas ?

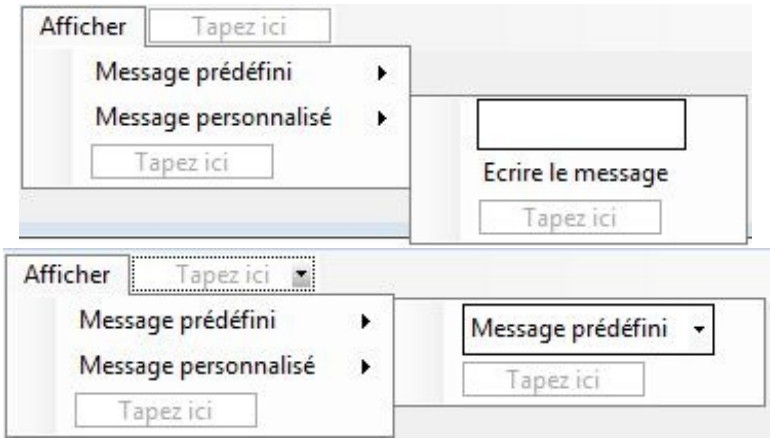
Eh bien, c'est parti : nous allons créer des **Combobox** (listes déroulantes) et des **TextBox**.

Dans le menu ??

Eh bien oui ! Oui, vous ne devez pas en voir souvent mais ça peut être utile !

Donc, pour avoir accès à ces contrôles supplémentaires il faut cliquez sur la petite flèche disponible à côté du "Tapez ici" :

Vous voyez que s'offre à vous les contrôles tant désirés !
Eh bien personnalisons un peu notre menu pour arriver à ça :



Sachant que dans la **ComboBox** (Message prédéfini), j'ai remis les messages d'avant (vous devez vous servir de la propriété "collection" de cette **combobox** pour en assigner les choix ou alors passer par le code VB, au choix).

Schématiquement :

- Fichier
  - Reset
  - Quitter
- Affichage
  - Message prédéfini
    - Combobox
      - Bonjour !
      - Au revoir.
      - Ciao.
      - Bye bye.
      - Astalavista baby !
  - Message personnalisé
    - TextBox
    - Ecrire

Ce qui est assez gênant avec cet assistant c'est que les noms qui sont entrés automatiquement sont assez cotons a repérer, avec une **textbox**, une **combobox**, ça passe mais au delà aie ! Alors prenez l'habitude de les renommer : un tour sur les propriétés et on change : **CB_MENU** et **TXT_MENU**.

Bon, ensuite on utilise notre fidèle assistant pour créer les évènements correspondants : sur le clic du bouton "**Ecrire**" et lors

du changement de la **ComboBox**.

Si vous avez utilisé l'assistant pour créer l'évènement de la **ComboBox**, lorsqu'elle est dans un menu, l'évènement est le **Clic**, il faut le changer :

Code : VB.NET

```
Private Sub CB_MENU_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
CB_MENU.SelectedIndexChanged
```

Supprimez les évènements relatifs aux anciens sous-menus (Bonjour ...) mais gardez ceux correspondant aux sous-menus **Reset** et **Quitter**.

Écrivons maintenant notre code : côté **combobox**, on veut afficher le texte correspondant à l'**item** de la **combobox** (je vous ai donné la solution là ), eh oui, l'évènement **SelectedItem** sera utilisé, le **selectedValue** n'étant pas disponible dans cette façon d'utiliser la **combobox**.

Ce qui nous donne :

Code : VB.NET

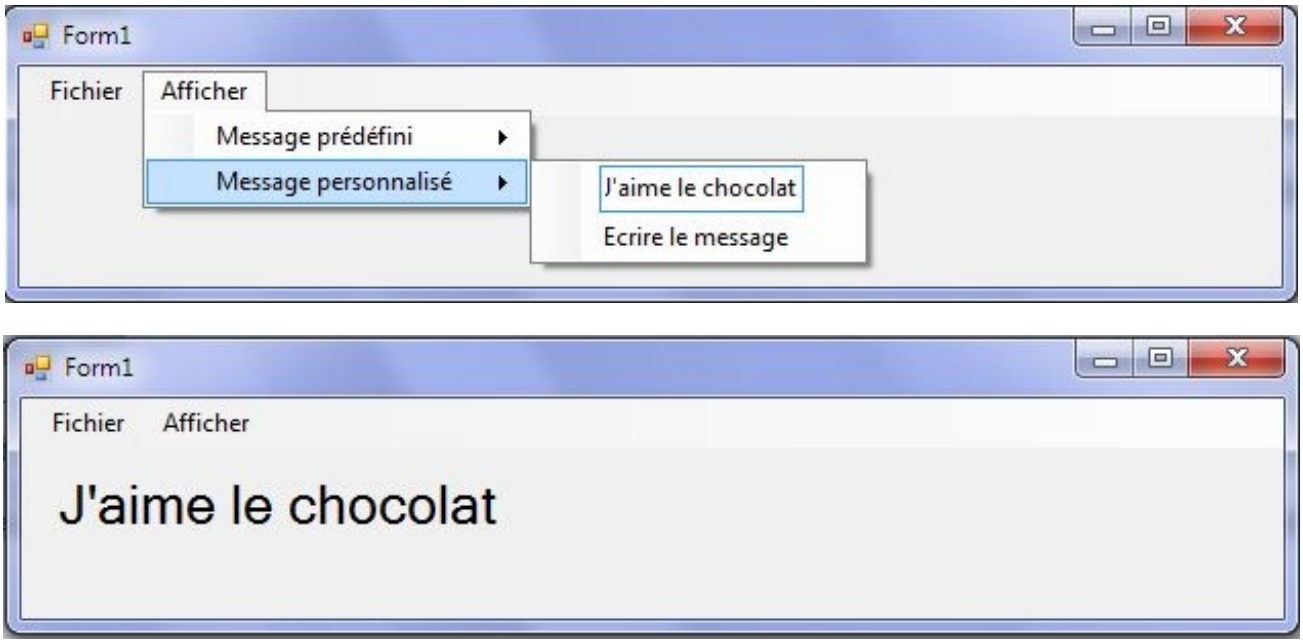
```
Private Sub CB_MENU_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
CB_MENU.SelectedIndexChanged
        Me.LBL_TEXTE.Text = Me.CB_MENU.SelectedItem
    End Sub
```

Bon, pour notre bouton **Ecrire**, c'est pas sorcier : on récupère la valeur de la **txtbox** et on l'affiche ; voilà le tout :

Code : VB.NET

```
Public Class Form1

    Private Sub ResetToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
ResetToolStripMenuItem.Click
        'Efface le label
        Me.LBL_TEXTE.Text = ""
    End Sub

    Private Sub QuitterToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
QuitterToolStripMenuItem.Click
        'Fermeture avec confimation
        If MsgBox("Souhaitez-vous vraiment quitter ce magnifique
programme ?", 36, "Quitter") = MsgBoxResult.Yes Then
            End
        End If
    End Sub

    Private Sub CB_MENU_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
CB_MENU.SelectedIndexChanged
        'Ecrit le texte de la combobox lros du changement d'index
        Me.LBL_TEXTE.Text = Me.CB_MENU.SelectedItem
    End Sub

    Private Sub EcrireToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
```

```
    EcrireToolStripMenuItem.Click
            'Ecrit le texte de la textbox lors de l'appui sur "ecrire"
            Me.LBL_TEXTE.Text = Me.TXT_MENU.Text
        End Sub
    End Class
```



Comme vous, le voyez VB est assez facile a utiliser dans différentes situations puisque les propriétés ne changent pas.

Bon, maintenant que vous savez ça, on ne se repose pas sur ses lauriers, on avance .

La barre de statut

Au tour de la barre de statut : c'est la barre qui vous indique le statut de l'application (si !).

Donc, à quoi va nous servir cette barre ?

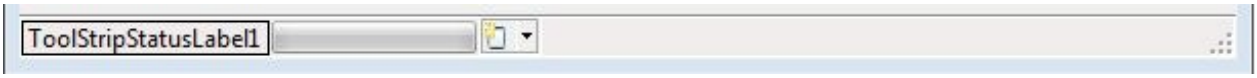
Eh bien à afficher, le statut de votre application par exemple, ou alors tout simplement mettre des boutons dessus !

Bon, je vais vous montrer une manière d'utiliser à bon escient cette barre, après, à vous de faire ce que vous voulez et de trifouiller toutes ses propriétés !

Bon, créons déjà la dite barre : toujours dans le menu "**Menus et barres d'outils**" vous choisissez "**StatusStrip**".

Vous l'intégrez à la feuille, elle se place en bas (normal), vous pouvez changer en agissant encore une fois sur la propriété "**Dock**".

Ajoutez 2 contrôles : un **label** et une **progressbar**. La **progressbar** est accessible, comme pour la **combobox** de la partie précédente, avec la petite flèche.



Renommez-les (bien sûr je vous dis de les renommer, mais si vous voulez vos noms à vous, ou tout simplement si vous voulez garder ceux d'origine, je ne vous oblige pas, ~~je ne suis pas votre dieu quand même ?~~) : **LBL_STATUT, PGB_STATUT**.

Nous allons nous servir de la **progressbar** comme indication d'avancement. Évidemment, ici, afficher un Label n'est pas sorcier, notre ordinateur ne va pas réfléchir plus d'une milliseconde (oui, même sous *windows* 3.1). Nous allons donc simuler une pause.

Pour utiliser cette **progressbar** comme indication voici une astuce. L'or d'un transfert comme un téléchargement, calculez la taille totale du fichier, le taux de transfert, ressortez le temps, et ajustez votre **progressbar** à ce temps, et voilà comment s'en servir comme source d'indication, mais bon ce n'est pas pour tout de suite.

Recréons donc un petit **timer** pour simuler le temps d'attente (**TIM_STATUT**) et utilisons le même procédé que le chapitre sur les **timers**.

Nous allons donc faire progresser la barre et afficher dans le label le statut.

C'est un exercice que vous pouvez évidemment faire.

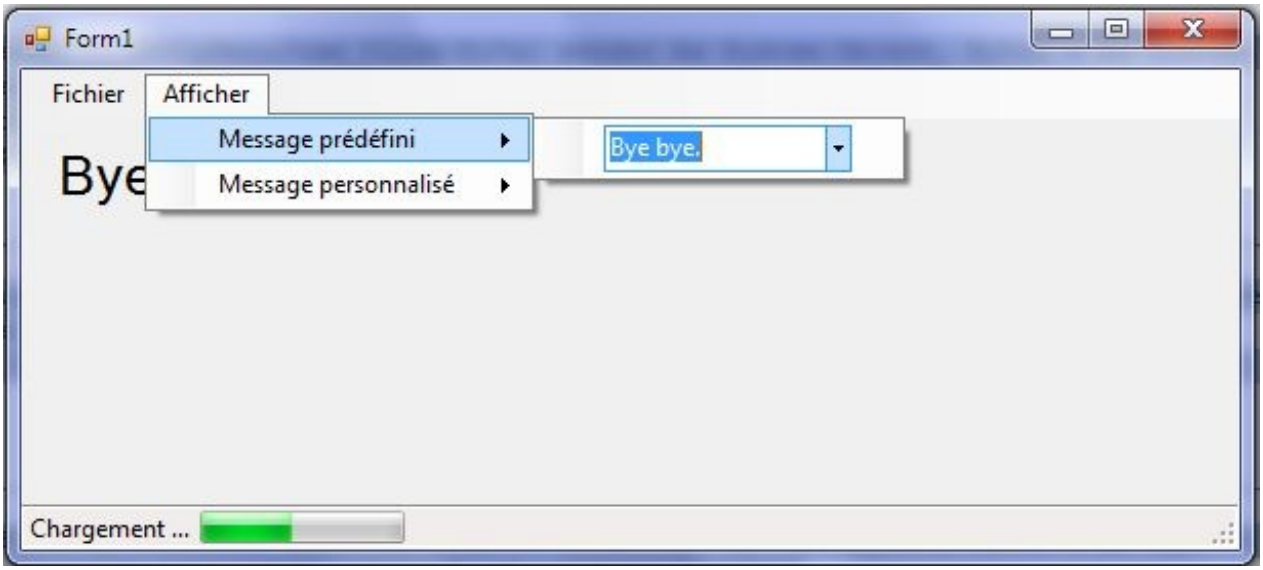
Voici donc une solution :

Code : VB.NET

```
Public Class Form1

    Private Sub ResetToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
ResetToolStripMenuItem.Click
        'Efface le label
        PauseFactice()
        Me.LBL_TEXTE.Text = ""

    End Sub

    Private Sub QuitterToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
QuitterToolStripMenuItem.Click
        'Fermeture avec confimation
        If MsgBox("Souhaitez-vous vraiment quitter ce magnifique
programme ?", 36, "Quitter") = MsgBoxResult.Yes Then
```

```
            End
        End If
    End Sub

    Private Sub CB_MENU_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
CB_MENU.SelectedIndexChanged
        'Ecrit le texte de la combobox lros du changement d'index
        PauseFactice()
        Me.LBL_TEXTE.Text = Me.CB_MENU.SelectedItem

    End Sub

    Private Sub EcrireToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
EcrireToolStripMenuItem.Click
        'Ecrit le texte de la textbox lors de l'appui sur "ecrire"
        PauseFactice()
        Me.LBL_TEXTE.Text = Me.TXT_MENU.Text

    End Sub

    Private Sub PauseFactice()
        LBL_STATUT.Text = "Chargement ..."
        PGB_STATUT.Value = 0
        TIM_STATUT.Enabled = True
    End Sub

    Private Sub TIM_STATUT_Tick(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles TIM_STATUT.Tick
        'Si la progressbar est arrivée au bout, on désactive le
timer
        If Me.PGB_STATUT.Value = 100 Then
            Me.TIM_STATUT.Enabled = False
            LBL_STATUT.Text = "Prêt"
        Else
            'Augmente de 1 la progressbar
            Me.PGB_STATUT.Value = Me.PGB_STATUT.Value + 1
        End If
    End Sub

End Class
```

Bon, pour ce code, je ne me suis pas trop fatigué : j'ai copié notre code dans le chapitre sur les timers. La "pause" n'est pas effectuée le texte s'affiche pendant la progression, oui c'est mal fait. Excusez-moi, mais bon c'est pour le principe ! L'idéal aurait été de placer un sémaphore (un flag) le tout avec une boucle while.



Alors ce code n'est pas dur a comprendre : j'ai mélangé le code de progression avec le code existant, en ajoutant des repères grâce au label : le "Chargement ..." et le "Prêt".

Comme d'habitude, essayez de modifier ce code pour le rendre plus efficace et comme vous le souhaitez.

Bon ... Cette barre n'était pas trop compliquée ? Eh bien ce n'est pas fini !

Le menu contextuel

Alors, le menu contextuel est, comme je vous l'ai expliqué, le menu visible lors du clic droit.

Comment ils font ceux sous *mac* avec un seul bouton à la souris ?

Je sais pas moi, il faudrait leur demander  .

Bon assez plaisanté.

Nous allons créer un **contextmenu**, toujours dans la suite de notre programme qui va déplacer le **Label** qui nous sert à afficher le texte.

Donc, toujours dans le menu de la boite à outils : "Menus et barres d'outils", vous prenez le **ContextMenuStrip** et vous l'intégrez à la feuille.

Une fois ce contrôle intégré, créez un élément avec comme texte : "Déplacer Le Label Ici".

Une fois cela fait, comme à l'accoutumée on crée son évènement correspondant.

Dans cet évènement, nous allons récupérer la position du curseur et changer la propriété **location** du **label** :

Code : VB.NET

```
Me.LBL_TEXTE.Location = Control.MousePosition
```

Control.MousePosition étant la propriété position de la souris (**control**), eh oui même la souris a des propriétés, vous en rêviez n'est-ce pas ?

Et donc voilà, une fois tout le code bien agencé :

Code : VB.NET

```
Public Class Form1

    Private Sub ResetToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
ResetToolStripMenuItem.Click
        'Efface le label
        PauseFactice()
        Me.LBL_TEXTE.Text = ""

    End Sub

    Private Sub QuitterToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
QuitterToolStripMenuItem.Click
        'Fermeture avec confimation
        If MsgBox("Souhaitez-vous vraiment quitter ce magnifique
programme ?", 36, "Quitter") = MsgBoxResult.Yes Then
            End
        End If
    End Sub

    Private Sub CB_MENU_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
CB_MENU.SelectedIndexChanged
        'Ecrit le texte de la combobox lros du changement d'index
        PauseFactice()
        Me.LBL_TEXTE.Text = Me.CB_MENU.SelectedItem
```

```
    End Sub

    Private Sub EcrireToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
EcrireToolStripMenuItem.Click
        'Ecrit le texte de la textbox lors de l'appui sur "ecrire"
        PauseFactice()
        Me.LBL_TEXTE.Text = Me.TXT_MENU.Text

    End Sub

    Private Sub PauseFactice()
        LBL_STATUT.Text = "Chargement ..."
        PGB_STATUT.Value = 0
        TIM_STATUT.Enabled = True
    End Sub

    Private Sub TIM_STATUT_Tick(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles TIM_STATUT.Tick
        'Si la progressbar est arrivée au bout, on désactive le
timer
        If Me.PGB_STATUT.Value = 100 Then
            Me.TIM_STATUT.Enabled = False
            LBL_STATUT.Text = "Prêt"
        Else
            'Augmente de 1 la progressbar
            Me.PGB_STATUT.Value = Me.PGB_STATUT.Value + 1
        End If
    End Sub

    Private Sub DéplacerLeLabelIciToolStripMenuItem_Click(ByVal
sender As System.Object, ByVal e As System.EventArgs) Handles
DéplacerLeLabelIciToolStripMenuItem.Click
        'lors d'un clic droit et du choix de déplacement du label,
on place le label aux positions de la souris.
        Me.LBL_TEXTE.Location = Control.MousePosition
    End Sub
End Class
```

Lors d'un clic et de la sélection, le label bouge (Waaaw ).

Bon, ce programme était là pour vous montrer les utilisations des différents menus, il est bien évidemment inutile (donc indispensable) !

Eh bien quelques connaissances en plus, une pierre en plus à l'édifice comme on dit !

TP : Navigateur WEB

Eh bien, pour moi, vous êtres prêts pour créer un vrai programme utilisable et fonctionnel, je pense aussi que ce petit exercice vous donnera envie de l'améliorer, le faire évoluer, bref le customiser à votre sauce.

Le cahier des charges

Eh bien c'est parti pour un TP, d'envergure cette fois !

Votre mission, si vous l'acceptez sera de créer un navigateur WEB en VB.net.

Wouaw ! Mais t'es pas bien, on sait à peine afficher 2-3 trucs et toi tu veux qu'on crée un navigateur WEB ?

Ah oui mince, vous ne connaissez ~~rien~~ pas beaucoup de choses, j'avais oublié ...

Mais c'est pas grave ! Pas besoin de s'y connaitre beaucoup pour le créer : avec ce que j'ai expliqué jusqu'à maintenant vous allez pouvoir déjà faire un joli truc, que nous améliorerons plus tard !

Un contrôle qui va nous être **indispensable** pendant ce TP est disponible sous VB : Le **WebBrowser**.

Si vous connaissez, un peu l'anglais, ça veut dire navigateur web. Ce contrôle va nous permettre de créer notre navigateur : vous lui entrez une adresse et il y va et affiche ce qu'il y a dans la page.

Il utilise le même moteur web qu'internet explorer (je sens que je vais me faire huer), bon le menu contextuel est donc déjà géré par ce contrôle, le téléchargement de fichiers aussi ...

Vous l'avez compris, nous allons créer l'interface.

Ce **webbrowser** est disponible dans les "**contrôles communs**".

Pour ce qui est des propriétés à utiliser pour naviguer, etc, eh bien à vous de trouver !

Ce ne sera pas sorcier, vous avez l'IDE qui vous affiche la liste des fonctions et propriétés disponibles sur le contrôle, après à vous de trouver celle qui sera utiliser et chercher comment l'utiliser en suivant la syntaxe donnée.

Oui ce TP, vous le ferez en *autonomie*, je n'interviendrais qu'à la fin, pour la correction.

Il va falloir chercher un peu c'est sûr, mais vous allez devoir le faire pour vos propres programmes, alors autant le faire tout de suite.

Bon, pour ce qui est de l'interface, donc : nous allons commencer doucement, je ne vais pas vous demander l'impossible : une barre d'adresses avec son bouton envoyer, précédent, suivant, arrêter, rafraichir.

Le statut de la page (terminé ...), le menu fichier : quitter (pas trop dur ).

Conseils : la méthode "url" du **webbrowser** sera sûrement utile  .

Après, tout dépendra de vos facilités, nous allons tout faire progressivement.

Bonne chance !

Les ébauches

Attention, la suite dévoile l'intrigue du film

Eh bien, j'espère que vous avez passé au moins quelques minutes a chercher (parce-que ça été mon cas).

Bon, nous allons progresser ensemble, voici donc mes premières ébauches, ce que je vous ai demandé de faire :

Code : VB.NET

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Les 2 lignes suivantes peuvent être remplacées par
Me.WB_NAVIGATEUR.Navigate("http://www.google.fr")
        Me.TXT_ADRESSE.Text = "http://www.google.fr"
        'Simule un clic en passant comme argument nothing (null)
        Me.BT_ADRESSE_Click(Nothing, Nothing)

        'Au démarrage, pas de possibilité de précédent, suivant,
stop
        Me.BT_SUIVANT.Enabled = False
        Me.BT_PRECEDENT.Enabled = False
        Me.BT_STOP.Enabled = False
    End Sub

    'Lorsque le chargement est fini
    Private Sub WB_NAVIGATEUR_DocumentCompleted(ByVal sender As
System.Object, ByVal e As
System.Windows.Forms.WebBrowserDocumentCompletedEventArgs) Handles
WB_NAVIGATEUR.DocumentCompleted
        'Affiche le nouveau statut, désactive le BT stop
        Me.LBL_STATUT.Text = WB_NAVIGATEUR.StatusText
        Me.BT_STOP.Enabled = False
        'on récupère l'adresse de la page et on l'affiche
        Me.TXT_ADRESSE.Text = Me.WB_NAVIGATEUR.Url.ToString
    End Sub

    'Lorsque le chargement commence
    Private Sub WB_NAVIGATEUR_Navigating(ByVal sender As
System.Object, ByVal e As
System.Windows.Forms.WebBrowserNavigatingEventArgs) Handles
WB_NAVIGATEUR.Navigating
        'On active le bouton stop
        Me.BT_STOP.Enabled = True
        'On met le statut à jour
        Me.LBL_STATUT.Text = WB_NAVIGATEUR.StatusText

        If Me.WB_NAVIGATEUR.CanGoForward Then
            Me.BT_SUIVANT.Enabled = True
        Else
            Me.BT_SUIVANT.Enabled = False
        End If
        If Me.WB_NAVIGATEUR.CanGoBack Then
            Me.BT_PRECEDENT.Enabled = True
        Else
            Me.BT_PRECEDENT.Enabled = False
        End If
    End Sub

#Region "Boutons de navigation"

    Private Sub BT_ADRESSE_Click(ByVal sender As System.Object,
```

```
        ByVal e As System.EventArgs) Handles BT_ADRESSE.Click
            'Si il existe une adresse, on y va
            If Not Me.TXT_ADRESSE Is Nothing Then
                Me.WB_NAVIGATEUR.Navigate(TXT_ADRESSE.Text)
            End If
        End Sub

        Private Sub BT_PRECEDENT_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles BT_PRECEDENT.Click
            'Va a la page précédente
            Me.WB_NAVIGATEUR.GoBack()
        End Sub

        Private Sub BT_SUIVANT_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles BT_SUIVANT.Click
            'Va a la page suivante
            Me.WB_NAVIGATEUR.GoForward()
        End Sub

        Private Sub BT_STOP_Click(ByVal sender As System.Object, ByVal e
    As System.EventArgs) Handles BT_STOP.Click
            'Desactive le bt stop et arrête le chargement du navigateur
            Me.BT_STOP.Enabled = False
            Me.WB_NAVIGATEUR.Stop()
        End Sub

        Private Sub BT_REFRESH_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles BT_REFRESH.Click
            'Raffraichit le navigateur
            Me.WB_NAVIGATEUR.Refresh()
        End Sub

    #End Region

        Private Sub QuitterToolStripMenuItem_Click(ByVal sender As
    System.Object, ByVal e As System.EventArgs) Handles
    QuitterToolStripMenuItem.Click
            If MsgBox("Souhaitez-vous vraiment quitter ce magnifique
    programme ?", 36, "Quitter") = MsgBoxResult.Yes Then
                End
            End If
        End Sub
    End Class
```

Ca y est, il y a plein de truc que je ne comprends pas, c'est quoi tes "**#region**" par exemple !?!

Pas de panique, votre code marche parfaitement sans, ça sert seulement à créer une zone rétractable ou l'on peut mettre les fonctions dont on est sûr du résultat pour plus de lisibilité.

Revenons à nos moutons ; la partie design :

(Très sobre, je sais, merci - Non je ne fais pas de pub pour Google)

Oui bon, vous l'avez compris, je ne me suis pas foulé côté visuel, nous rendrons tout cela plus beau plus tard !

Alors, quelques explications du code.

Les instructions directement liées au **webbrowser** sont nombreuses, vous auriez dû les trouver avec un peu de patiente, les plus pressés d'entre vous auront craqués et passé directement à cette partie je pense .

Je vais vous les lister avec mes noms d'objet (donc **WB_NAVIGATEUR** pour le **webbrowser**) :

- WB_NAVIGATEUR.StatusText = Récupère le statut du navigateur
- Me.WB_NAVIGATEUR.Url.ToString = récupère l'adresse actuellement parcourue par le navigateur
- Me.WB_NAVIGATEUR.CanGoForward = renvoie un booléen pour dire si le navigateur à une page suivante (si vous avez fait précédent avant)
- Me.WB_NAVIGATEUR.CanGoBack = pareil qu'au dessus mais pour dire si le navigateur peut faire précédent
- Me.WB_NAVIGATEUR.Navigate(TXT_ADRESSE.Text) = le navigateur va a l'adresse de la page passée en argument (ici le texte de TXT_ADRESSE)
- Me.WB_NAVIGATEUR.GoBack() = le navigateur va a la page précédente
- Me.WB_NAVIGATEUR.GoForward() = navigue vers la page suivante
- Me.WB_NAVIGATEUR.Stop() = arrête le chargement d'une page
- Me.WB_NAVIGATEUR.Refresh() = rafraichi la page

Comme vous l'avez remarqué dans le code j'ai deux évènements concernant le navigateur : un qui se déclenche quand la page commence à se charger (**Handles WB_NAVIGATEUR.Navigating**) et le second, celui d'origine du **webbrowser** : quand la page s'est totalement chargée (**Handles WB_NAVIGATEUR.DocumentCompleted**).

J'utilise ces 2 évènements pour activer ou non le bouton stop, changer le statut de la page, mettre à jour la nouvelle adresse, activer ou non les boutons précédent, suivant.

J'ai utilisé cette forme pour vous montrer comment nous allons améliorer ce programme en exploitant au mieux les évènements de notre contrôle (eh oui les fonctions c'est bien beau mais les évènements c'est magnifique).

Bien exploiter les évènements

Bon, je ne sais pas si vous avez prêtés attention à tous les évènements que nous offre ce petit **webbrowser** ...

En voici quelques uns qui vont nous être fort utiles :

- Handles WB_NAVIGATEUR.StatusTextChanged
- Handles WB_NAVIGATEUR.CanGoBackChanged
- Handles WB_NAVIGATEUR.CanGoForwardChanged
- Handles WB_NAVIGATEUR.ProgressChanged

Nous allons donc abondamment, ~~fortement, exponentiellement,~~ utiliser le petit "e", vous vous souvenez, ce petit argument dont j'ai parlé il y a quelques chapitres. Eh bien on va désormais l'utiliser.Il correspond à un objet qui va nous être utile, cet objet correspondra à différentes choses suivant le **handles** : par exemple pour le **handles ProgressChanged** il pourra nous fournir l'état d'avancement de la page, pour le cas du **statustextchanged**, le texte de statut, ainsi de suite ...

Améliorons notre navigateur en nous servant de ces évènements pour activer / désactiver les boutons précédent, suivant en fonction de la possibilité ou non d'avancer ou reculer dans l'historique, de mettre une barre de progression, un texte de progression, etc ...

Ce qui nous donne pour seulement, la gestion des évènements du navigateur :

Code : VB.NET

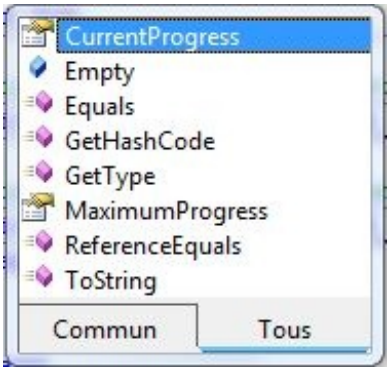
```
#Region "Evènements du WBroser"

    'a chaque changement d'état, on met à jour les boutons
    Sub WB_NAVIGATEUR_CanGoForwardChanged(ByVal sender As Object,
ByVal e As EventArgs) Handles WB_NAVIGATEUR.CanGoForwardChanged
        If Me.WB_NAVIGATEUR.CanGoForward Then
            Me.BT_SUIVANT.Enabled = True
        Else
            Me.BT_SUIVANT.Enabled = False
        End If
    End Sub

    'a chaque changement d'état, on met à jour les boutons
    Sub WB_NAVIGATEUR_CanGoBackChanged(ByVal sender As Object, ByVal
e As EventArgs) Handles WB_NAVIGATEUR.CanGoBackChanged
        If Me.WB_NAVIGATEUR.CanGoBack Then
            Me.BT_PRECEDENT.Enabled = True
        Else
            Me.BT_PRECEDENT.Enabled = False
        End If
    End Sub

    'Au changement de statut de la page
    Sub WB_NAVIGATEUR_StatutTextChanged(ByVal sender As Object,
ByVal e As EventArgs) Handles WB_NAVIGATEUR.StatusTextChanged
        'On met le statut à jour
        Me.LBL_STATUT.Text = WB_NAVIGATEUR.StatusText
    End Sub

    'Au changement de progression de la page
    Sub WB_NAVIGATEUR_ProgressChanged(ByVal sender As Object, ByVal
e As WebBrowserProgressChangedEventArgs) Handles
WB_NAVIGATEUR.ProgressChanged
        Me.PGB_STATUT.Maximum = e.MaximumProgress
        Me.PGB_STATUT.Value = e.CurrentProgress
    End Sub

    'Lorsque le chargement est fini
    Private Sub WB_NAVIGATEUR_DocumentCompleted(ByVal sender As
```

```
        System.Object, ByVal e As
        System.Windows.Forms.WebBrowserDocumentCompletedEventArgs) Handles
        WB_NAVIGATEUR.DocumentCompleted
                'desactive le BT stop
                Me.BT_STOP.Enabled = False
                'On cache la barre de progression
                Me.PGB_STATUT.Visible = False
                'on récupère l'adresse de la page et on l'affiche
                Me.TXT_ADRESSE.Text = Me.WB_NAVIGATEUR.Url.ToString
            End Sub

            'Lorsque le chargement commence
            Private Sub WB_NAVIGATEUR_Navigating(ByVal sender As
        System.Object, ByVal e As
        System.Windows.Forms.WebBrowserNavigatingEventArgs) Handles
        WB_NAVIGATEUR.Navigating
                'On active le bouton stop
                Me.BT_STOP.Enabled = True
                'au début du chargement, on affiche la barre de progression
                Me.PGB_STATUT.Visible = True
            End Sub
        #End Region
```

Bon, ce code, si vous avez pris la peine d'essayer de le comprendre fait ce que j'ai dit plus haut en s'aidant du "e" dans un cas pour l'instant : faire avancer la **progressbar**.

Alors comment ais-je fais pour réaliser cette prouesse xD !

Eh bien si vous avez tapé **"e."** (dans l'évènement du changement de progression), notre ide nous fournit les fonctions, objets, propriétés pouvant être utilisées.



Vous voyez 2 lignes qui s'écartent du lot :

- **CurrentProgress**
- **MaximumProgress**

En mettant le curseur dessus votre ide nous explique que ces valeurs renvoient chacune un long (donc un nombre que nous allons pouvoir exploiter) mais à quoi correspond-il ? Eh bien la réponse est déjà grandement fournie dans le nom mais bon, en dessous c'est marqué : le **maximumprogress** nous renvoie le nombre de **bytes** a télécharger pour avoir la page et le **currentprogress**, le nombre de **bytes** actuellement téléchargés.

Ensuite, il ne faut pas sortir de St-Cyr pour savoir ce qu'il faut faire : on attribue le nombre de bytes a télécharger en tant que maximum pour la **progressbar**, et on met comme valeur le nombre de bytes actuellement téléchargés.

Et on obtient notre premier évènement dans lequel on exploite les arguments transmis par "**e**".

Euh, quand j'ai voulu copier-coller la ligne :

Code : VB.NET

```
        Sub WB_NAVIGATEUR_StatutTextChanged(ByVal sender As Object,
        ByVal e As EventArgs) Handles WB_NAVIGATEUR.StatusTextChanged
```

et remplacer StatusTextChanged par ProgressChanged une erreur inconnue au bataillon est apparue ...

Alors, c'est normal : certains évènements utiliseront, comme ici **e as EventArgs** (ou **system.EventArgs**), alors que d'autres utiliseront des **e** de type spécifique : **WebBrowserProgressChangedEventArgs** (dans le cas du **handles progresschanged**). Et c'est également pour cette raison que dans certains évènements de propriétés supplémentaires s'offriront à notre "**e**", simplement car celui-ci n'est pas du même type ...

Bon, cette partie est très importante car ce petit **e** sera utilisé très souvent dans vos programmes, lorsque vous allez réagir avec des objets, c'est ce **e** qui gèrera les retours d'évènements.

Le design

Bon, évidemment, ça ne pousse pas trop à l'utiliser s'il reste comme ça notre programme, nous allons donc l'améliorer un peu côté visuel.

J'ai donc décidé d'utiliser des icônes et pictogrammes sous licence *creative commons for non commercial use* . Je vais vous les montrer ici mais le pack complet (plus de 1000 pictos) est disponible Ici.

Voici les dits pictos. Bon, je ne suis pas très créatif donc on va dire que le premier ce sera pour le bouton d'envoi, le second, le refresh, ensuite stop, suivant puis précédent.

Si vous êtes pas d'accord avec moi ~~voici mon adresse~~ vous n'avez qu'a prendre ceux qui vous plairont !

Bon, nous allons donc intégrer une image à nos boutons, pour ça il faut agir sur la propriété ... **Image** (sisi).

Bon, lorsque vous allez vouloir choisir une image, une fenêtre vous propose 2 choix : utiliser une ressource locale ou une fichier de ressources, la différence : le fichier de ressource rassemble toutes les ressource : images, sons, etc... alors que les ressources externes ne seront pas intégrées à la compilation du projet. Les ressources externes sont donc bonnes pour les tests.

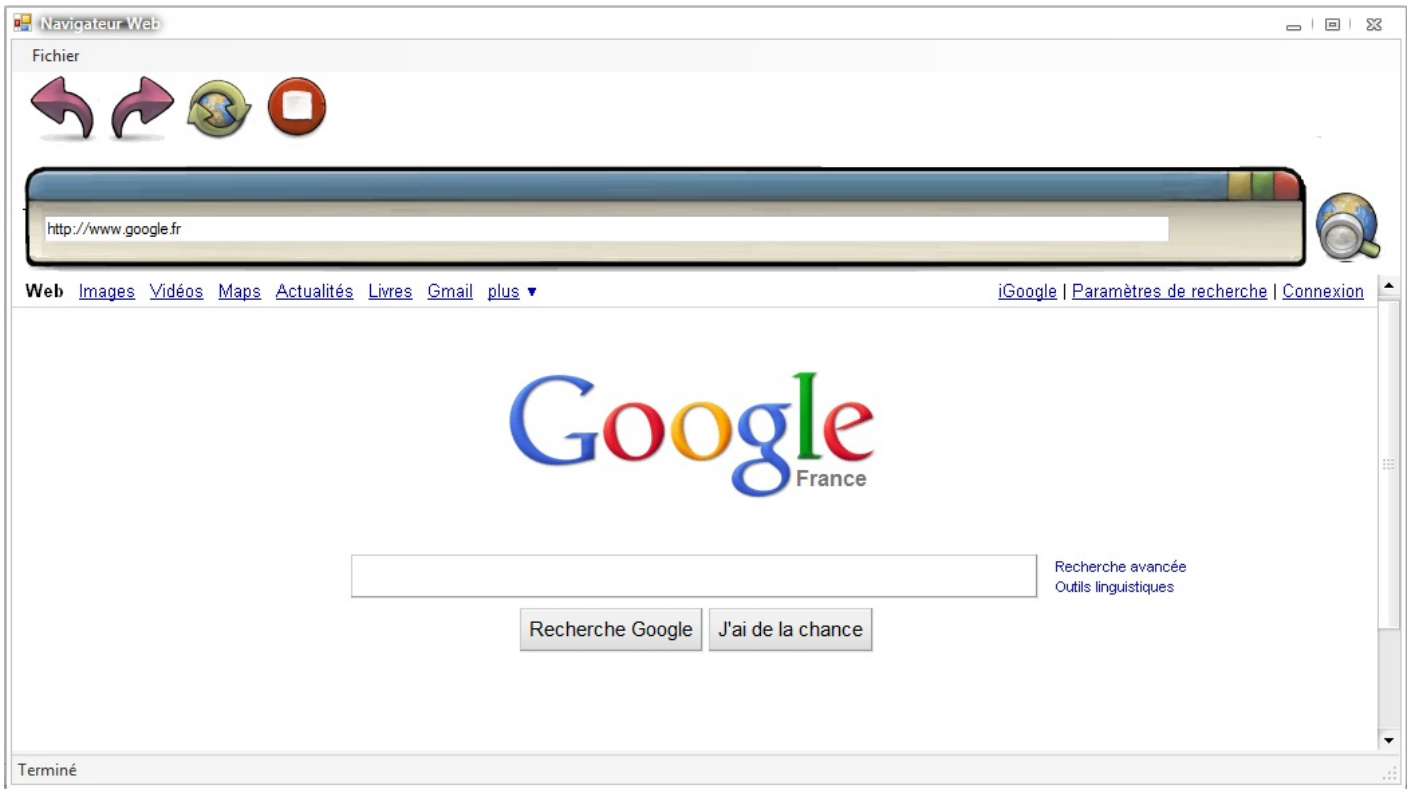
Nous allons tout de suite utiliser le fichier de ressource. Cliquez donc sur le petit "importer" en bas et choisissez vos images.

Attribuez les bonnes images au bons boutons (ce serait bête d'avoir un précédent avec une icône de suivant ..).

Pour un plus beau rendu, mettez la propriété **FlatStyle** à **Flat**, et dans **FlatAppearance**, **bordersize** à 0.

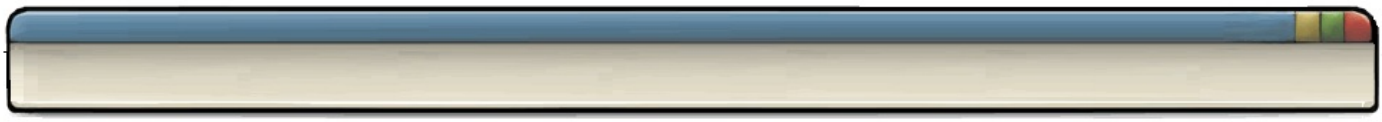
Bon, après à vous de toucher un peu les propriétés de la page, des éléments comme vous le sentez pour les adapter à vos goûts.

Avec un peu d'ennui et paint d'ouvert voilà a quoi j'arrive :



Bien sûr, pas de moqueries, je ne suis pas graphiste. Bon, je vais quand même vous donner mon image d'arrière-plan (si

certains osent la prendre).



Comme vous le voyez, j'ai changé la couleur de certains contrôles, modifié les styles etc ...

Cette partie sur le design n'avait pas la prétention de faire de vous des pros du design mais juste de vous faire découvrir une autre facette du développement d'un programme.

Bon, nous voilà avec une base de navigateur, gardez-le de côté, un prochain chapitre consacré a son amélioration viendra quand j'aurais apporté de nouveaux concepts et de nouvelles connaissances.

Fenêtres supplémentaires

Vous venez de faire un bien gros projet. Je suppose que vous allez vouloir en apprendre encore plus pour pouvoir agrémenter et améliorer vos programmes.

Eh oui, même avec tout ce que vous avez appris jusqu'ici, il vous reste encore bien des notions à acquérir.

Attaquons tout de suite les fenêtres supplémentaires.

Ajouter des fenêtres

Attaquons alors ! Je suis sûr que vous vous impatientez.

Alors nous allons commencer par ajouter des fenêtres supplémentaires puis nous allons apprendre à les faire communiquer entre elles !

Créerons tout de suite un nouveau projet de test Windows Forms avec le nom que vous souhaitez (pour moi ce sera FenetresSupplementaires).

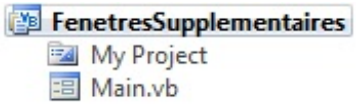
On se retrouve comme à l'accoutumée avec notre fenêtre **form1** gisant au beau milieu de notre feuille de design.

Vu que nous allons travailler avec plusieurs fenêtres, les noms de fenêtres vont être important maintenant.

Renommons donc cette fenêtre principale. Appelons-là "Main" (lorsque vous créerez un programme je vous suggère de nommer cette première fenêtre avec le nom du programme).

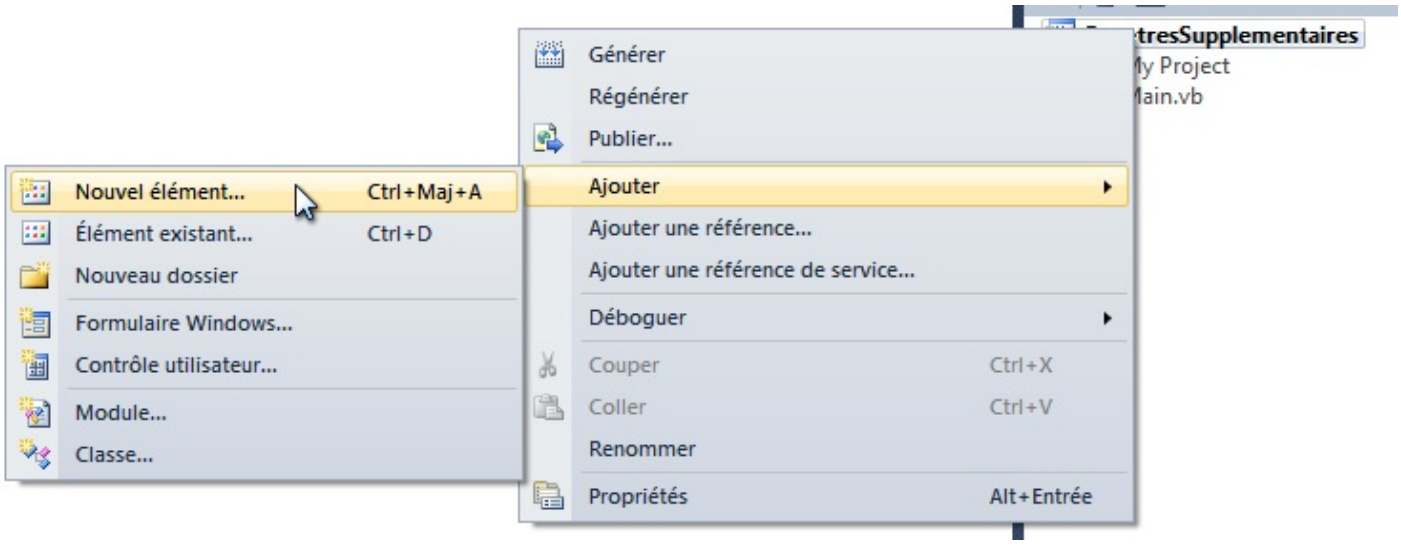
Pour ce faire, cliquons une fois sur elle dans la fenêtre design pour avoir accès à ses propriétés. Dans la valeur **(Name)** inscrivez donc **Main**, faites de même pour la valeur **Text**.

Puis renommons la feuille contenant cette fenêtre que nous voyons dans la fenêtre de solutions. Clic droit sur "**Form1.vb**" puis renommer. Inscrivez à la place "**Main.vb**".

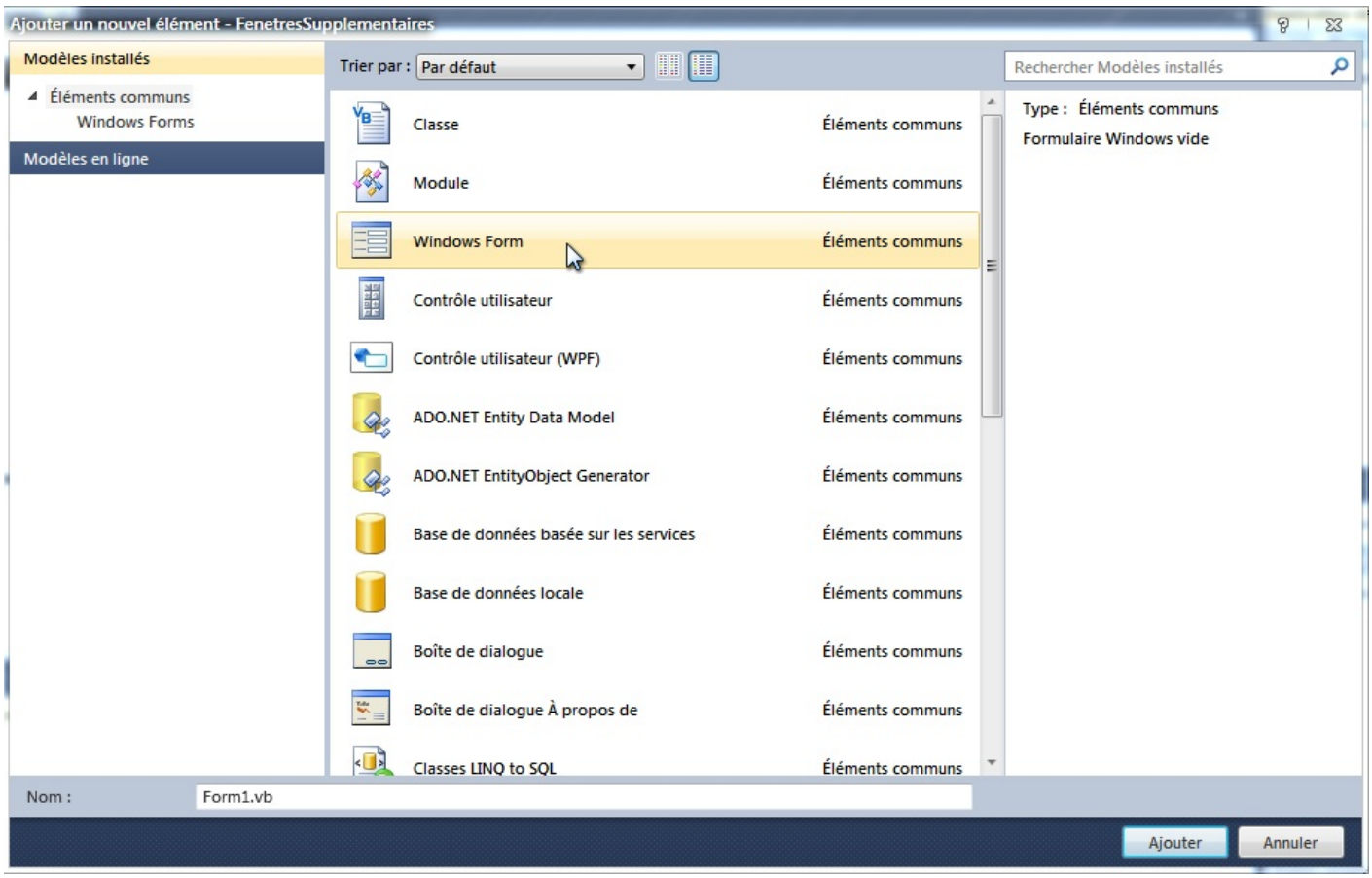


Voilà, vous venez de renommer entièrement votre fenêtre. Il faudra faire de même avec les supplémentaires.

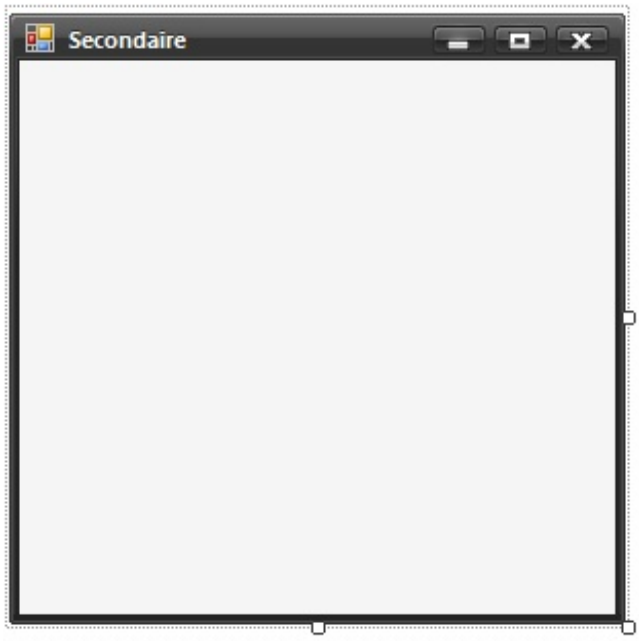
Ajoutons une seconde fenêtre maintenant. Un clic droit sur le nom de votre projet (pour moi : FenetresSupplementaires), puis déplacez votre souris sur le menu "Ajouter" qui nous donnera accès à "Nouvel élément", cliquez dessus.



Une fenêtre vous proposant quel type d'élément vous voulez ajouter à votre projet s'est ouverte. Nous voulons une fenêtre, il va donc falloir sélectionner "**Windows Form**" (retenez bien cette manipulation, elle va nous permettre d'ajouter d'autres types d'éléments à notre projet), pensez également à renommer cette fenêtre, essayez de trouver un nom adapté à sa fonction. Vu que notre projet est là pour l'apprentissage et n'a aucune fonction particulière à remplir je vais lui donner comme nom "**Secondaire**".



Vous voici avec votre seconde fenêtre "Secondaire" qui s'est automatiquement ouverte. Comme vous avez pu le constater, les modification de renommage que nous avons effectués sur la fenêtre Main ont été automatiquement effectuées sur celle-ci.



Vous voilà avec votre première seconde fenêtre (dur à suivre). Allons nous amuser avec elle !

Ouverture / Fermeture

Vous vous souvenez que je vous ai toujours appris à assigner des propriétés à vos contrôles en commençant la ligne par **"Me."**.

C'est dans ce chapitre que vous allez vous rendre compte de son utilité.

Créons tout de suite un contrôle sur notre seconde fenêtre, mettons un bouton **fermer**.

On vient à peine de la créer tu veux déjà nous apprendre à la fermer ?

Oui, on va effectuer simplement un programme avec un bouton qui l'ouvre et un bouton permettant de la fermer.

Donc, je crée mon bouton **Fermer** identifié *BT_FERMER*. Me voilà donc avec seulement ce bouton dans ma fenêtre secondaire.

Créons donc un bouton **Ouvrir** identifié *BT_OUVRIR* sur la fenêtre principale (Main). Vous pouvez accéder à la fenêtre main soit grâce au système d'onglets si vous ne l'avez pas fermée, soir grâce à la fenêtre solution en double-cliquant sur son nom.

Sur notre seconde fenêtre comme sur la première, si nous voulons accéder à des propriétés, il va falloir utiliser le Me. dans la page de code correspondante. En parlant d'elle, allons-y, créons l'évènement **BT_FERMER_Click** en double-cliquant sur le bouton.

L'évènement Load (ici Secondaire_Load) d'une fenêtre supplémentaire sera appelé à chaque fois que la demande d'ouverture de cette fenêtre sera demandé.

La fonction permettant de fermer une fenêtre individuelle est `Close()` .

Tu nous avais parlé de End dans les autres chapitres.

Oui, End permet de fermer le programme, dans notre cas nous voulons fermer la fenêtre seule, il faut donc utiliser la fonction close.

Maintenant l'objet sur lequel cette fonction va être exécutée va être important. La feuille de code dans laquelle je me trouve actuellement correspond à la fenêtre secondaire. En utilisant le préfixe **Me.** l'objet de cette fenêtre sera automatiquement pris en compte. Si vous avez suivis, notre fonction va se retrouver sous cette forme :

Code : VB.NET

```
Private Sub BT_FERMER_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles BT_FERMER.Click
    Me.Close()
```

```
    End Sub
```

Et donc avec cette méthode nous sommes certains que c'est cette fenêtre qui va être affectée par le **Close()** et donc fermée.

Retournons dans la fenêtre **Main** et double-cliquons sur le bouton "Ouvrir" pour créer son évènement correspondant.

Et insérons dans cet évènement le code nécessaire pour ouvrir une autre fenêtre qui est ...
La fonction Show()

La propriété **Visible** de la fenêtre supplémentaire peut aussi être utilisée pour afficher ou faire disparaître cette dernière. Cependant je vous la déconseille car c'est une fenêtre fantôme qui est toujours présente en mémoire. Les fonctions Show() et Close() permettent d'ouvrir et fermer proprement ces nouvelles fenêtres.

Alors, si vous avez suivi mon monologue sur les *Objets*, sur quel objet va-t-il falloir appliquer cette fonction ?

Eh bien c'est sur l'objet de la fenêtre supplémentaire. Autrement dit l'objet "Secondaire".

Ce qui nous donne :

Code : VB.NET

```
    Private Sub BT_OUVRIR_Click(ByVal sender As System.Object, ByVal e
    As System.EventArgs) Handles BT_OUVRIR.Click
        Secondaire.Show()
    End Sub
```

Pourquoi pas Me. ?

Eh bien toi tu n'as pas dû écouter ce que j'ai dit sur les objets !  L'utilisation du préfixe "Me." force l'utilisation de la fenêtre actuelle comme objet, vous ne voulez pas afficher une fenêtre déjà ouverte  .

Si je test, tout fonctionne, lors du clic sur Ouvrir la seconde fenêtre s'ouvre, lors du clic sur Fermer, la seconde fenêtre se ferme.

Eh bien vous avez déjà réussi à faire apparaître et fermer votre nouvelle fenêtre.

Notions de parent/enfant

Pourquoi tu veux nous parler de famille ?

Non, non. Le concept de parent/enfant est aussi utilisé en informatique.

Je vais justement vous l'exposer sommairement ici.

Partie théorique, allez vous chercher un café et détendez-vous.

Programmation orientée objet

La notion de parent/enfant est à l'origine utilisée en Programmation Orientée Objet (abréviée POO).

La POO est un style de programmation de plus en plus répandu. Certains langages ne sont pas orientés objets, d'autres y sont totalement. Notre langage, VB.NET utilise énormément la notion d'orienté objet.

Le langage VB6, quand à lui ne prenait pas en charge les concepts de la POO

Concrètement que nous apporte cette notion de POO ?

Eh bien si vous voulez avoir plus de détail concernant la POO je vous renvoie sur un chapitre du tutoriel sur le C++ de M@teo21 qui explique très bien les concepts d'objets.

Pour faire simple : on a introduit la notion d'objet pour pouvoir gérer plus facilement les gros programmes. Par exemple, dans nos programmes, les fenêtres sont toutes des objets bien distincts. Lorsque nous voudrons agir sur une fenêtre en particulier, il nous suffira de manipuler son objet.

Bon, j'essaie de suivre mais pourquoi nous racontes-tu ça ?

Eh bien maintenant que je vous ai fait peur avec les Objets  , je vais vous parler des relations parent/enfant qui s'applique sur les objets.

L'héritage

Cette notion a été introduite avec la notion d'**héritage**.

L'héritage existait bien avant l'informatique ...

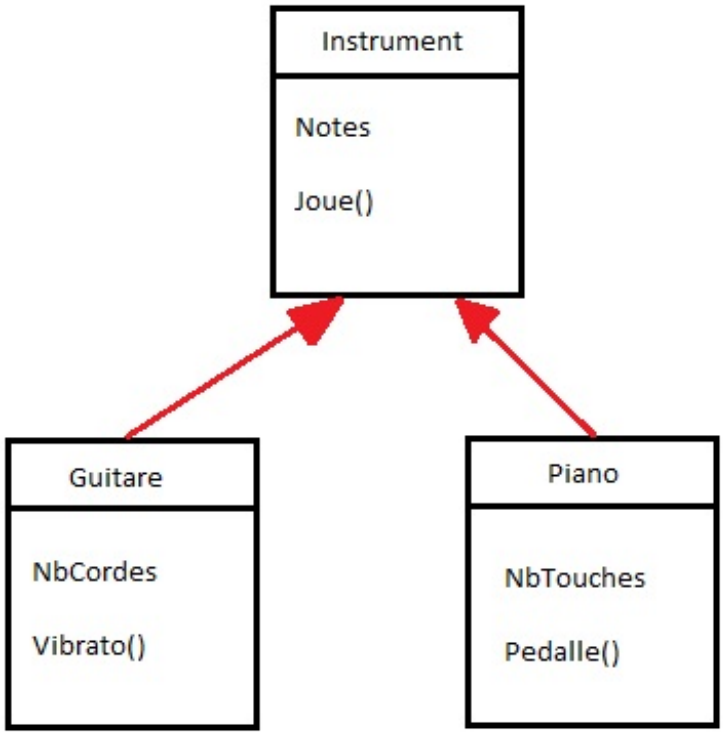
Mais moi je vous parle de l'héritage en informatique. C'est un concept qui s'applique aux objets.

Imaginez que vous avez un objet de type *instrument* (eh oui, un objet peut être n'importe quoi du moment que vous le codez), cet objet va avoir des variables et des fonctions qui lui seront spécifiques (les notes qu'il est capable de jouer, la fonction "joue", etc ...).
Si je crée un autre objet de type *guitare*, vous voyez tout de suite qu'une guitare est un instrument, donc au lieu de recréer

toutes les fonctions et variables qui existaient pour l'objet *instrument*, je vais faire **hériter** ma *guitare* de *instrument*.

Ouch ?

Voici un petit schéma :



Donc, les rectangles sont chacun des objets. Vous voyez l'objet *Instrument*, l'objet *Guitare* et l'objet *Piano*.
L'objet *Guitare* et *Piano* héritent tout les deux de l'objet *Instrument*.

Donc, la *Guitare* aura en plus de ses possibilités originelles (qui sont NbCordes et Vibrato()) celles de *Instrument* (qui sont Notes et Joue()). Pareil côté *Piano*.

Parent/enfant

Je crois comprendre. Pourquoi nous expliques-tu cela ?

Eh bien, vous voyez que si je détruis l'objet *Instrument* les objets qui en héritent (autrement dit Guitare et Piano) seront également détruits. Dans ce cas *Instrument* est le **parent** et *Guitare* et *Piano* sont les **enfants**.

Donc lorsqu'un **parent** est détruit ses **enfants** le sont également.

J'en viens donc à notre problème actuel : les fenêtres.

Lorsque je lance mon programme, la première fenêtre, ici le Main est

considéré comme la fenêtre **parent**. et donc toutes les fenêtres supplémentaires crées seront ses **enfants**.
Si vous avez suivis ce que j'ai expliqué, si je ferme donc la fenêtre Main, les autres fenêtres se fermeront également.

Donc il va falloir bien faire attention à ça dans nos programmes. Ne pas fermer la fenêtre principale !

Et si je ne veux pas la voir ?

Et bien il vous suffit d'effectuer un *Visible = false* sur cette dernière (j'ai dit que c'est pas bien mais ici vous êtes obligés). Mais attention avec ça, c'est pas le tout de cacher la fenêtre et de ne jamais pouvoir la ré-afficher.

Bon, avec toutes ces nouvelles notions nous allons pouvoir attaquer la communication entre fenêtres.

Vous pouvez ré-ouvrir les yeux, le cauchemar est terminé.

Communication entre fenêtres

Bon, après cette lourde partie théorique, attelons nous a faire communiquer nos fenêtres entre elles.

Pourquoi diable aurais-je envie de faire ça ?

Eh bien vous avez vus comment déclarer des variables. Vous ne voulez pas aller modifier les variables de la fenêtre d'à côté ? Écrire dans une textbox présente sur une autre fenêtre ?

Et puis même si vous n'avez pas envie, je vais quand même vous l'expliquer.

Manipulation de contrôles

Commençons par manipuler les contrôles, le plus facile.

Créons un label dans notre fenêtre principale nommée LBL_MAIN et un dans la fenêtre secondaire nommée LBL_SECOND. Enlevez leur leurs textes pour ne laisser que du blanc.

Nous allons écrire un message dans le label de la fenêtre secondaire à son ouverture mais à partir de la feuille de code de la fenêtre main. Puis inversement lors de la fermeture de la fenêtre secondaire.

Si vous avez bien appréhendé toutes les notions d'objet, vous devriez être capables de le faire vous même.

Eh bien pour manipuler un contrôle d'une autre fenêtre, il suffit d'inscrire le nom de la fenêtre souhaitée à la place du préfixe "Me."

Soit pour les deux évènements présents sur les deux fenêtres.

Fenêtre **Main**

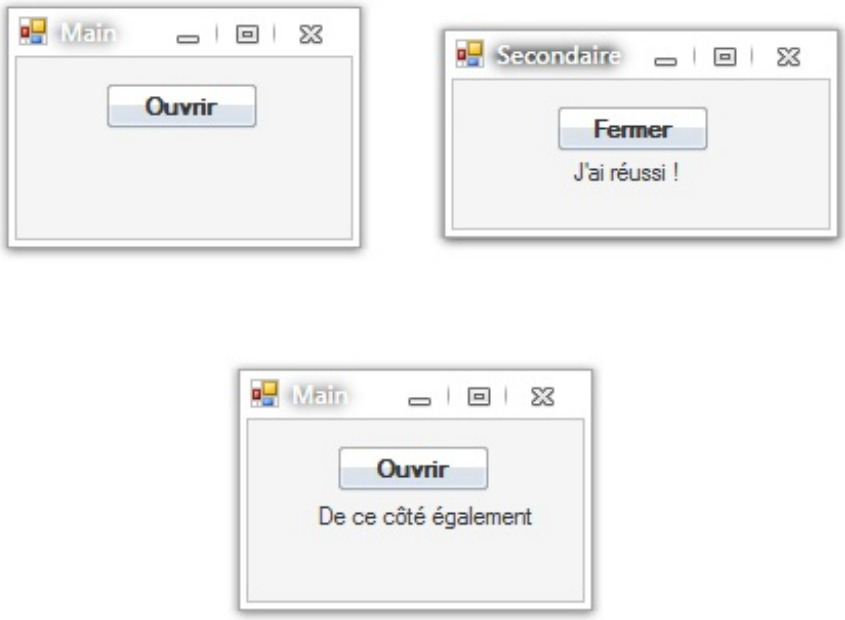
Code : VB.NET

```
Public Class Main

    Private Sub BT_OUVRIR_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles BT_OUVRIR.Click
        Secondaire.Show()
        Secondaire.LBL_SECOND.Text = "J'ai réussi !"
    End Sub

End Class
```

Fenêtre **Secondaire**

Code : VB.NET

```
Public Class Secondaire

    Private Sub BT_FERMER_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles BT_FERMER.Click
        Main.LBL_MAIN.Text = "De ce côté également"
        Me.Close()
    End Sub

End Class
```



Nous avons réussi à manipuler des contrôles distants !

Manipulation des variables

Attaquons nous donc tout de suite aux variables.

Une petite précision supplémentaire va être requise pour cette partie.

Pour le moment nous déclarions nos variables avec un Dim à l'intérieur d'une fonction ou alors directement déclarées dans les arguments de la fonction. Nos variables avaient donc une "durée de vie" limitée. Une fois la fonction terminée, toutes les variables déclarées à l'intérieur cessaient d'exister.

Nous allons donc créer des variables **globales** de manière à ce qu'elles soient accessibles de partout.

Pour déclarer une variable **globale** il faut placer son instruction de déclaration juste après l'ouverture du module.

Si elles sont crées comme je vous l'ai appris (Dim X as Integer), ces variables sont accessibles uniquement à partir de la fenêtre les ayant crées. Pour pouvoir y accéder d'ailleurs il va falloir les rendre publiques. Je vais donc vous apprendre un nouveau mot (programmatique) : **Public**

Donc, si vous voulez des variables accessibles de "l'extérieur" il va falloir les déclarer ainsi.

On reprend donc le code de notre fenêtre secondaire en ajoutant cette variable **globale** et en assignant au label sa valeur lors du chargement de la fenêtre.

Code : VB.NET

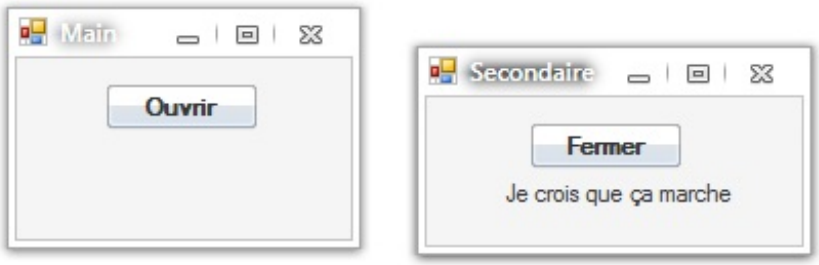
```
Public Class Secondaire

    Public MonString As String = ""

    Private Sub Secondaire_Load(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles MyBase.Load
        Me.LBL_SECOND.Text = MonString
    End Sub

    Private Sub BT_FERMER_Click(ByVal sender As System.Object, ByVal
```

```
        e As System.EventArgs) Handles BT_FERMER.Click
            Main.LBL_MAIN.Text = "De ce côté également"
            Me.Close()
        End Sub

    End Class
```

Et le code de la fenêtre Main dans lequel j'accède à la variable MonString. J'ai volontairement retiré la ligne où je modifiais directement le label.

Code : VB.NET

```
    Public Class Main

        Private Sub BT_OUVRIR_Click(ByVal sender As System.Object, ByVal
    e As System.EventArgs) Handles BT_OUVRIR.Click
            Secondaire.MonString = "Je crois que ça marche"
            Secondaire.Show()
        End Sub

    End Class
```



Et voilà, votre première variable **Globale Publique** et vous y avez déjà accédé à partir d'un autre objet !

Les fichiers - Partie 1/2

Je suppose que vos connaissances ne vous suffisent pas ! Eh bien tant mieux, on va attaquer une partie pour le moins intéressante.

Elle concerne les fichiers.

Vous avez bien entendu, vous allez pouvoir commencer à enregistrer des données pour les récupérer même si le programme s'est fermé entre temps (ce qui n'était pas possible avec les variables si vous m'avez suivi).

Donc tout cela va nous permettre de créer des fichiers de configuration, sauvegarder des textes, des images, des scores, que sais-je encore...

Votre imagination est la seule limite de la programmation.

Introduction sur les fichiers

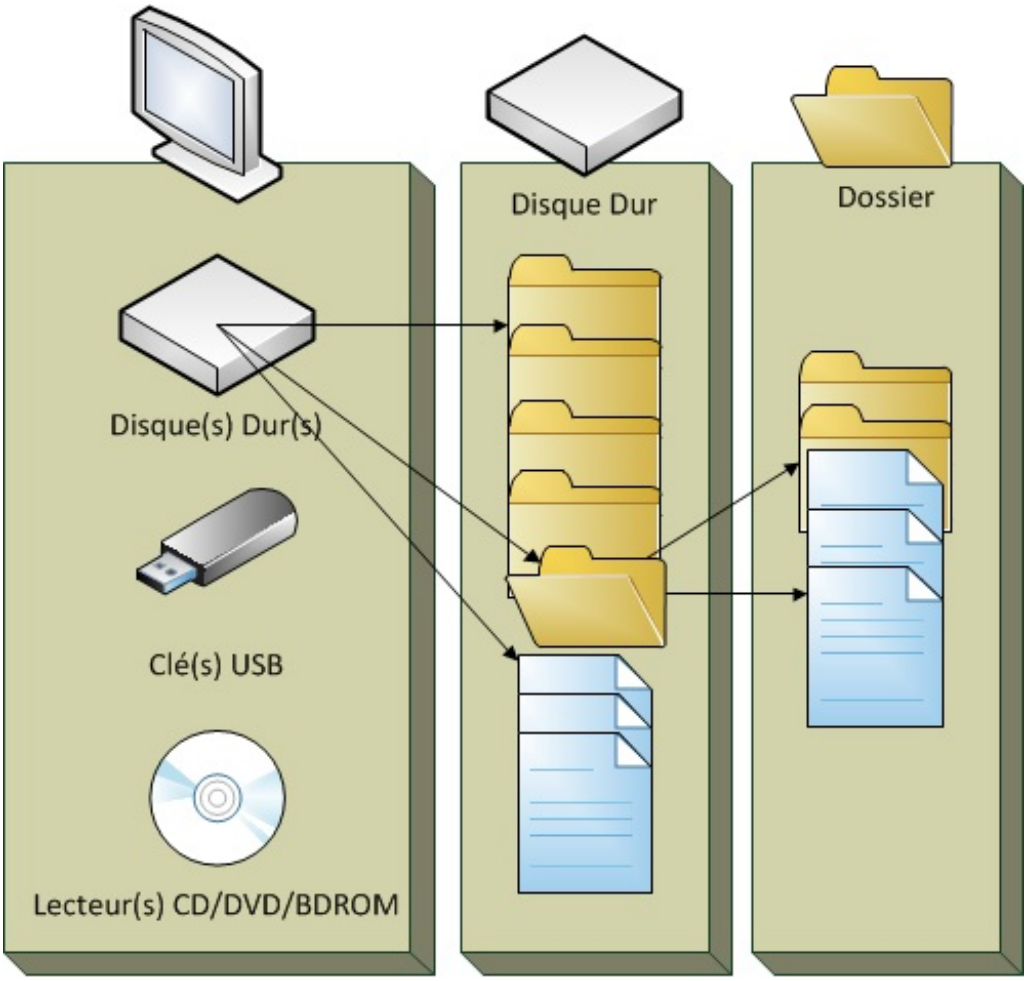
Zéros, bonsoir.

Nous allons commencer par une rapide introduction sur les fichiers. Je suppose que pour nombre d'entre vous c'est un concept acquis mais pour d'autres il va falloir quelques éclaircissements.

Commençons. Amis *Windowsiens*, vous pouvez apercevoir dans votre poste de travail des lecteurs et des disques durs.

Tu nous prends pour qui ?

Du calme, ça va se compliquer.



Donc dans le(s) disque(s) dur(s), vous pouvez naviguer dedans suivant un système d'**arborescence** : Le disque dur contient des dossiers, ces dossiers contiennent des fichiers (récursif : les disques durs peuvent contenir des fichiers et les dossiers d'autres dossiers).

Résumons : le disque dur contient toutes vos données ; le dossier permet de gérer, organiser et hiérarchiser votre disque dur ; *les fichiers quand à eux contiennent des données pures et dures*.

Les fichiers : des 0 et des 1

Les fichiers contiennent des données donc. Ces données sont représentées (côté machine de votre PC) par des 0 et des 1 (des bits), le système binaire qu'ils appellent ça .

Nous, pauvres petits mortels ne comprendrions rien à ce langage propre à la machine, c'est pourquoi des *codages* ont été mis en place pour convertir ces groupements de 0 et de 1 (généralement groupés par 8, ce qui donne **8 bits** autrement appelé **un**

octet).

Donc individuellement, vous vous apercevez que ces 0 et ces 1 ne sont pas reconnaissables, on ne peut rien en tirer. Mais une fois en groupe, ces petits bits peuvent être transcrits différemment en fonction du codage.

Exemples :

Octet en binaire	Nombre décimal	Nombre hexadécimal
01100011	99	63
10010010	146	92

Alors, les cases dans chaque ligne de ce tableau ont la même valeur, seulement le codage utilisé n'est pas le même.

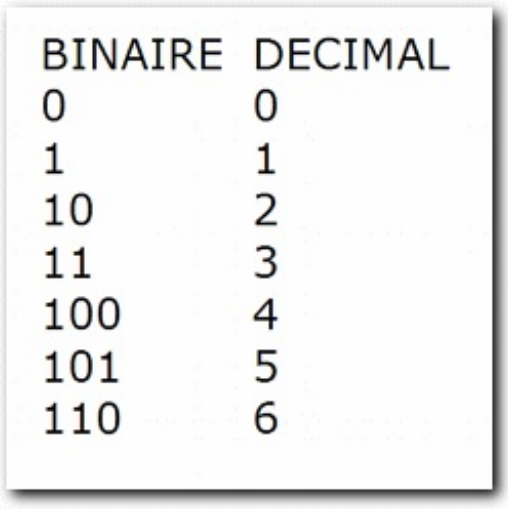
Le nombre **décimal** résultant de ces 0 et ces 1 vous le connaissez, pour peu que vous soyez allés à l'école primaire. Par contre j'ai été méchant, j'ai rajouté une colonne avec à l'intérieur un nombre **Hexadécimal**.

Sans m'étendre sur le sujet, le système **Hexadécimal** est de base **16** (où le décimal est de base 10), il a été inventé par des informaticiens principalement pour des informaticiens. Il permet de transcrire rapidement des nombres binaires (car un groupement de 4 chiffres binaire correspond à un chiffre hexadécimal).

Mais pourquoi tu nous dis tout ça ... ?

Ça vient, ça vient. Donc vous avez compris que les données sont stockées sous forme de 0 et de 1, que des codages existent pour les transcrire en quelque chose de compréhensible par un humain. Pour le moment on se retrouve avec des nombres.

Mais moi dans mes fichiers je vois pas ça, il y a des lettres !



Oui mais la base de tous vos fichiers sont des 0 et des 1 ! Maintenant découvrons comment ils deviennent des caractères grâce à **la norme ASCII**.

La norme ASCII

La norme ASCII est la norme de codage de caractères standardisée. Autrement dit on l'utilise désormais dans tous les systèmes d'exploitation. Ici, c'est un groupement de 8 bits qui est converti en un caractère grâce à une table ASCII. Lien vers une table ASCII.

Exemple : la première suite de bits du tableau plus haut (01100011) correspond au caractère 'c'

Donc on retient que un caractère = 1 octet = 8 bits.

Bref, je ne vous demande pas d'apprendre la table ASCII par cœur, notre IDE se chargera d'effectuer les codages tout seul. Tout ça pour vous sensibiliser un peu quant à la taille de vos fichiers. Windows a l'habitude de noter les tailles en **ko** pour les petits fichiers jusqu'aux **Mo** voire **Go**.

Ces acronymes correspondent à Kilo octet, Mega octet et Giga octet.
Respectivement 1024 octets, 1 048 576 octets et 1 073 741 824 octets.

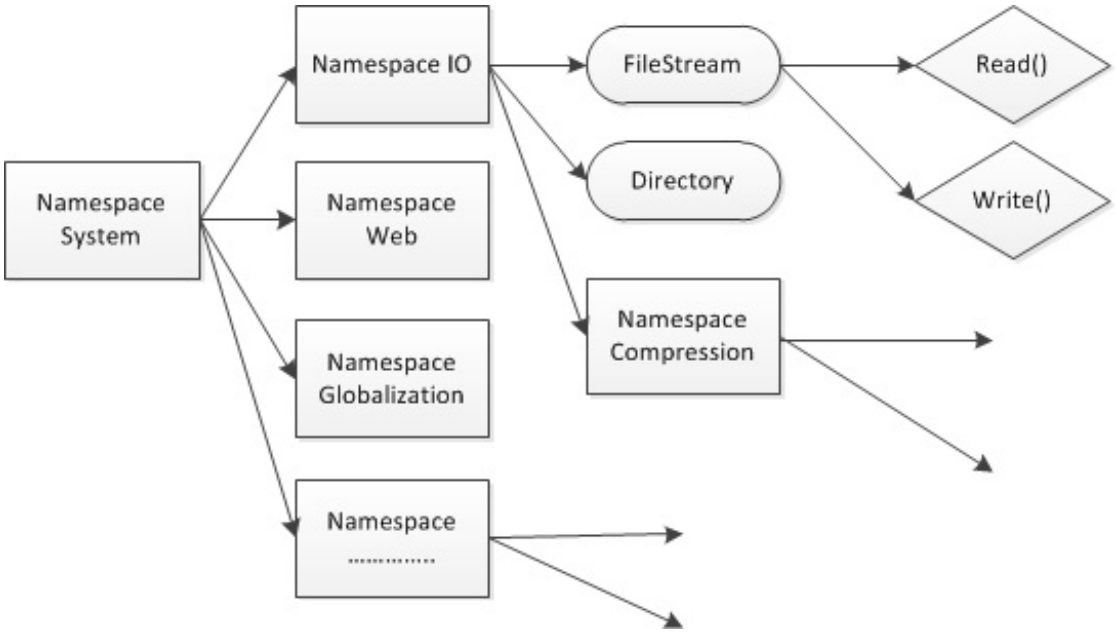
Donc **1024** caractères équivaudra à **un ko**.

Bien, on voit un peu mieux le fonctionnement des fichiers ? On est prêt à les modifier ? C'est parti !

Le namespace IO

Je vous ai peut être fait peur avec mes notions se rapprochant de la machine mais ne vous inquiétez pas, c'était un peu de culture générale.

Microsoft, au travers de son Framework (qui est une librairie contenant des centaines de classes, fonctions, objets) a développé tout les outils nécessaires pour travailler facilement sur les fichiers.



Ce namespace (un namespace est un sorte de dossier contenant des classes / fonctions spécifiques) est le namespace **IO**.
Comme vous le voyez sur le schéma, ces namespaces permettent d'organiser le contenu du Framework.

Pourquoi un nom comme ca ? **Fichiers** aurait été mieux ...

Eh bien **IO** correspond à **Input Output**. En français : **Entrée Sortie**.

Ce namespace ne va pas contenir que les fonctions et objets pour manipuler les fichiers, elle va nous permettre également d'effectuer de la communication inter-processus, de la communication série et de la compression. Mais n'allons pas trop vite.

Donc ce namespace fait lui-même partie du namespace System (comme dans votre ordinateur, on a plusieurs niveaux de dossiers pour mieux classer vos fichiers et bien ici c'est pareil avec les namespace et les objets/fonctions).

Et dans ce namespace se situe la classe **FileStream** qui va nous permettre de créer un objet de type **FileStream** et de le manipuler. La classe **File** quant à elle ne nous permettra pas de créer un objet mais seulement de manipuler notre **FileStream**.
Vous allez très vite comprendre.

Il existe aussi une classe **Directory** permettant de manipuler les répertoires.

Une petite information supplémentaire avant de passer à la pratique :
Nous allons devoir créer un **Objet** et la manipuler pour cela nous allons découvrir un nouveau mot récurrent en programmation : **new**.

Notre premier fichier

Let's go !

Créez un nouveau projet et rendez-vous dans l'évènement du **form load** (je ne vous explique plus la démarche, vous êtes grands), je vous attends.

Donc créons notre objet et entrons le dans une variable.

De quel type ma variable ?

Très bonne question, on va créer un objet permettant de manipuler les fichiers, ce serait malpropre de l'insérer dans un **integer** voire un **string** ...

Eh bien, j'ai dit que nous allons créer un objet **FileStream**, pourquoi ne pas l'entrer dans une variable de type **FileStream** ?

J'ai donc crée mon objet et je l'ai entré dans une variable :

Code : VB.NET

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Création d'un objet de type FileStream
        Dim MonFichier As IO.FileStream = New
IO.FileStream("Zero.txt", IO.FileMode.OpenOrCreate)

    End Sub

End Class
```

Plein de choses à vous expliquer en une seule ligne, j'aime ça !

Allons-y. Vous reconnaissez tous le `Dim MonFichier As IO.FileStream`, une simple déclaration de variable de type **FileStream**.

Pourquoi **IO.** avant ?

Je l'ai expliqué, on est dans le *namespace IO*, il faut donc faire **IO.** avant de pouvoir accéder aux membres du **namespace**.

Petite astuce : inscrivez `Imports System.IO` tout en haut de votre programme, avant la déclaration du module. Cette ligne va permettre de s'affranchir de cet **IO.** avant nos fonctions utilisant ce namespace.

Bon, déjà une chose de faite, continuons : `= New IO.FileStream` permet d'assigner une valeur à notre variable dès la déclaration, ça aussi nous avons vu. Le mot clé **New**, j'ai dit qu'il servait à créer un nouvel objet (ici de type **FileStream**).

Instanciation de notre premier objet

Oui j'avais compris mais pourquoi tu as mis des choses entre parenthèses, ce n'est pas une fonction quand même !

Eh bien pas exactement. Lorsque nous instancions un objet (le mot instanciation fait peur mais il signifie juste que nous en

créons un nouvel objet grâce à **New**), la classe utilisée pour déclarer l'objet va faire appel à son **Constructeur**. Le mot **constructeur** est spécifique à la *POO*, nous reviendrons dessus plus tard. Le fait est que ce **constructeur** est appelé et des fois ce constructeur nécessite des **Arguments**.

Dans notre cas, le **constructeur** de **FileStream** accepte plein de "*combinaisons*" d'arguments possible (ce que l'on appelle la **surcharge**, j'expliquerai aussi plus tard).

J'ai choisi les plus simples : le **Path** du fichier (en **String**) avec lequel nous allons travailler et un argument qui va nous permettre de déterminer comment ouvrir ou créer le fichier (de type **IO.FileMode**).

Le Path

Je vais faire une rapide parenthèse sur le Path. Tout d'abord le mot Path signifie le **chemin** du fichier.
Ce chemin (je préfère parler de Path) peut être de deux types :

- Absolu : le chemin n'a pas de référence mais n'est pas exportable (ex : C:\Windows\System32 ... est un chemin absolu) ;
- Relatif : le chemin prend comme point de repère le dossier d'exécution de notre programme (ex Zero.txt sera placé dans le même dossier que le programme que nous créons).

Il est donc préférable d'utiliser des chemins relatifs dans nos programmes à moins que vous soyez certains de l'emplacement des fichiers que vous voulez manipuler.

FileMode

Dans notre cas, j'ai inscrit un Path relatif, le fichier Zero.txt sera crée s'il n'existe pas, sinon il sera ouvert. Et tout cela grâce à l'argument **IO.FileMode.OpenOrCreate**.

Cet argument peut prendre quelques autres valeurs :

Nom VB	Valeur	Description
FileMode.CreateNew	1	Crée le fichier spécifié, s'il existe déjà une erreur se produira.
FileMode.Create	2	Crée le fichier s'il n'existe pas. S'il existe, le remplace.
FileMode.Open	3	Ouvre un fichier existant, une erreur se produira s'il n'existe pas.
FileMode.OpenOrCreate	4	Ouvre un fichier existant, s'il n'existe pas ce dernier sera crée puis ouvert.
FileMode.Truncate	5	Ouvre le fichier spécifié et le vide entièrement, la lecture de ce fichier n'est pas possible dans ce mode
FileMode.Append	6	Ouvre le fichier spécifié et se place à sa fin.

Comme vous le voyez, l'argument que j'ai utilisé **FileMode.OpenOrCreate** (aussi remplaçable par le chiffre 4), permet d'adapter notre programme. Imaginez en utilisant l'argument **FileMode.CreateNew**, le premier lancement du programme se déroulera bien mais lors du second lancement une erreur se produira parce que le fichier existe déjà. A moins que vous ne gériez toutes ces éventualités. Mais nous sommes des zéros, allons au plus simple.

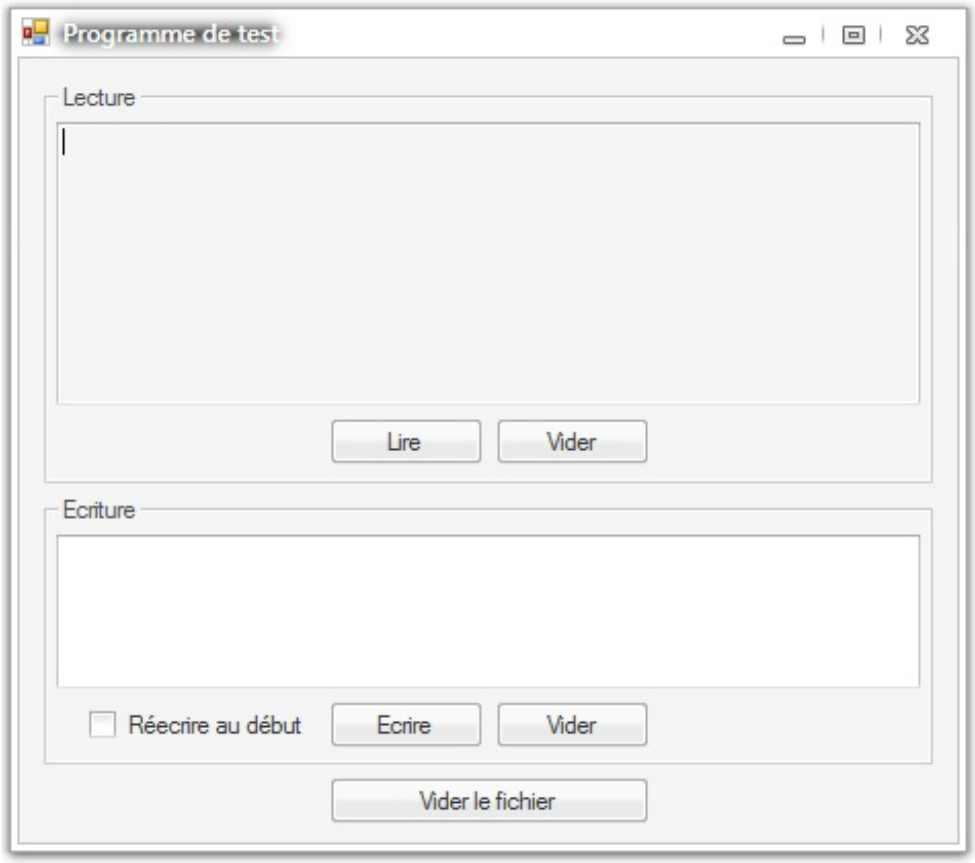
Résumé

Résumons ce que cette instruction a fait : on a ouvert le fichier **Zero.txt** (créé s'il n'existait pas) et on l'a dans la variable **MonFichier**. Bien bien.

Nos premières manipulations

Programme de base

Bon continuons, créons une petite interface basique permettant de lire / écrire dans le fichier. J'aimerais donc que vous créiez quelque chose qui ressemble à ça :



Alors, pour ce qui est des noms des contrôles je pense que vous êtes grands maintenant, ils ne vont plus poser problème.
Mes deux textBox (TXT_LECTURE, TXT_ECRITURE) ont la propriété **Multiline** à true, celle du haut a **ReadOnly** à true.
Des boutons (BT_LIRE, BT_CLEARLIRE, BT_ECRIRE, BT_CLEARECRIRE et BT_CLEAR tout en bas) et une checkbox (CHK_DEBUT).

Voici pour ce qui est du design. Pour le code je vais vous montrer le mien et on va détailler le tout. Attention, je reprends pas mal de concepts abordés avant tout en intégrant des nouveaux, accrochez-vous !

Code : VB.NET

```
Imports System.IO

Public Class Form1

    Dim MonFichier As IO.FileStream

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Création d'un objet de type FileStream
        MonFichier = New IO.FileStream("Zero.txt",
IO.FileMode.OpenOrCreate)
    End Sub

    Private Sub Form1_FormClosing(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.FormClosing
```

```
            'Libère la mémoire
            MonFichier.Dispose()
        End Sub

    #Region "Gestion des boutons"

        Private Sub BT_LIRE_Click(ByVal sender As System.Object, ByVal e
    As System.EventArgs) Handles BT_LIRE.Click

            If MonFichier.CanRead() Then
                'Crée un tableau de Byte
                Dim Contenu(1024) As Byte
                'Lit 1024 bytes et les entre dans le tableau
                MonFichier.Position = 0
                MonFichier.Read(Contenu, 0, 1024)
                'L'affiche
                Me.TXT_LECTURE.Text = ""
                For Each Lettre As Byte In Contenu
                    Me.TXT_LECTURE.Text += Chr(Lettre)
                Next
            End If

        End Sub

        Private Sub BT_ECRIRE_Click(ByVal sender As System.Object, ByVal
    e As System.EventArgs) Handles BT_ECRIRE.Click
            If MonFichier.CanWrite Then
                Dim Contenu(1024) As Byte
                Dim Compteur As Integer = 0
                'Parcours la txtbox
                For Each Lettre As Char In
    Me.TXT_ECRITURE.Text.ToCharArray
                    'Convertit une lettre en sa valeur ascii et l'entre
    dans compteur
                    Contenu(Compteur) = Asc(Lettre)
                    Compteur += 1
                Next
                'Ecrit dans le fichier
                If Me.CHK_DEBUT.Checked Then
                    MonFichier.Position = 0
                End If
                MonFichier.Write(Contenu, 0, Compteur)
            End If
        End Sub

        Private Sub BT_CLEARLIRE_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles BT_CLEARLIRE.Click
            Me.TXT_LECTURE.Text = ""
        End Sub

        Private Sub BT_CLEARECRIRE_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles BT_CLEARECRIRE.Click
            Me.TXT_ECRITURE.Text = ""
        End Sub

        Private Sub BT_CLEAR_Click(ByVal sender As System.Object, ByVal
    e As System.EventArgs) Handles BT_CLEAR.Click
            'Je ferme le fichier actuel
            MonFichier.Dispose()
            'Je le réouvre en écrasant ses données
            MonFichier = New IO.FileStream("Zero.txt", FileMode.Create)
        End Sub

    #End Region

    End Class
```

Explications

Bien bien, vous voilà avec des codes de plus en plus conséquents. Prenons le problème par étapes. Tout d'abord nous avons les boutons de vidage des TextBox qui ne sont pas sorciers, une simple instruction pour remplacer leur contenu.

Alors commençons à étudier le voyage de notre fichier. Je déclare en variable **Globale** le fichier, de façon à ce qu'il soit accessible dans toutes les fonctions. Lors du **load** j'ouvre mon fichier comme nous l'avons vu dans la partie d'avant.

Et, chose importante, j'ai réagi à l'évènement **FormClosing** (traduisible par *fenêtre en cours de fermeture*, à ne pas confondre avec **FormClosed** : *fenêtre fermée*). Lorsque cet évènement se produit, je **Dispose()** le fichier.

La fonction **Dispose** permet de vider les ressources mémoire que prenait le fichier. En résumé, cela le **ferme**.

Donc, fichier ouvert et chargé à l'ouverture du programme, fermé à la fermeture. Parfait !
Travaillons.

Nous arrivons aux deux boutons *Lire* et *Ecrire*.

L'écriture

Bien, commençons par l'écriture (on ne va pas lire avant d'avoir écrit ).

Code : VB.NET

```
Private Sub BT_ECRIRE_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles BT_ECRIRE.Click
        If MonFichier.CanWrite Then
            Dim Contenu(1024) As Byte
            Dim Compteur As Integer = 0
            'Parcours la txtbox
            For Each Lettre As Char In
Me.TXT_ECRITURE.Text.ToCharArray
                'Convertit une lettre en sa valeur ascii et l'entre
dans compteur
                Contenu(Compteur) = Asc(Lettre)
                Compteur += 1
            Next
            'Ecrit dans le fichier
            If Me.CHK_DEBUT.Checked Then
                MonFichier.Position = 0
            End If
            MonFichier.Write(Contenu, 0, Compteur)
        End If
    End Sub
```

- Alors, première instruction, déjà une nouvelle chose : *MonFichier.CanWrite*
  C'est une propriété de l'objet. Elle nous informe sur la possibilité d'écriture dans notre fichier. Si c'est *true*, c'est parfait, on continue (ces petites vérification sont souvent inutiles mais il ne coute rien de les faire et elles peuvent parfois éviter des erreurs, pensez aussi à gérer les cas d'erreur aussi).
- Je crée ensuite un tableau de **Byte**, 1025 cases (je prévois grand !). Sachant que chaque **byte** (je n'ai pas expliqué mais un **byte** est aussi de 8 bits dans notre cas soit ... **Un octet** soit ... **Un caractère** !) peut contenir un caractère, nous avons une possibilité d'écriture de 1025 caractères.
- Un petit compteur, il va nous servir après.
- Puis un **For Each** grâce auquel je parcours tous les caractères contenus dans ma textbox : *Me.TXT_ECRITURE.Text.ToCharArray*. La fonction **ToCharArray** permet, comme son nom anglais l'indique, de

convertir en tableau de char. Pour chaque caractère donc, ce caractère est entré dans la variable **Lettre**.
- Je rentre chaque lettre dans mon tableau de **Byte**. Mais attention, les **Bytes** et les **Char** ne sont pas homogènes, il faut passer par une fonction qui va récupérer la valeur binaire de notre caractère (j'ai expliqué au début de ce chapitre =), **transformation ASCII => 8 Bits** grâce à la fonction **Asc**()) de façon a pouvoir l'inscrire dans le **Byte**.
- Viens ensuite l'incrémentation du compteur pour pouvoir écrire chaque caractère dans une case différente.
- Ensuite, si la case est cochée on déplace le curseur au début du fichier. Je vais parler des curseurs juste après.
- Puis on écrit le contenu de notre tableau en indiquant combien de **Bytes** écrire (avec **Compteur**)

Eh bien, je sais qu'il y a pas mal de notions d'un coup. Reprenez-le tout lentement en essayant de comprendre chaque ligne individuellement.

Les curseurs

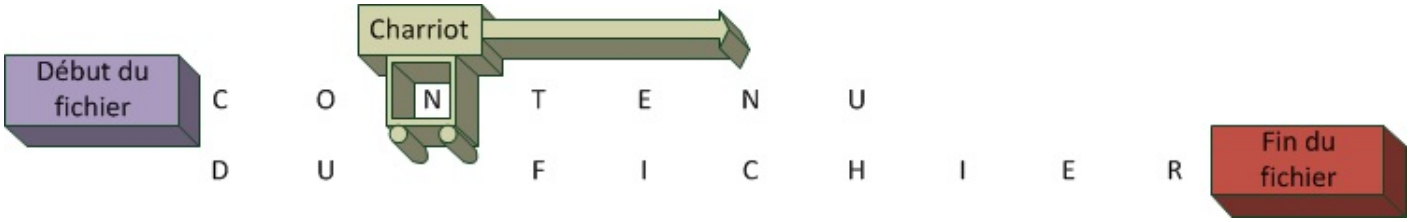
Petit aparté sur les curseurs.

Alors je viens de parler de curseur dans notre fichier mais qu'est-ce que cela ?

Non, n'y pensez même pas, ce n'est pas un curseur de souris qui bouge dans notre fichier mais c'est comparable :
Un curseur doit être représenté mentalement. C'est un petit charriot qui avance dans notre fichier. Lorsqu'on lui demande de lire ou d'écrire, ce petit charriot va se déplacer de caractère en caractère et l'écrire (ou le lire). Donc lorsqu'on lit un fichier entier, le curseur se retrouve tout à la fin. Si on ne lit que la moitié, à la moitié.

Quelques schémas :

Ici le charriot est au début du fichier, une demande de lecture a été effectué, il lit ce caractère puis se déplace au suivant.



Ainsi de suite.

Arrivé à la fin du fichier, il s'arrête et se met en attente d'une commande lui demandant de se déplacer.

Bref, tout ça pour dire que ce petit charriot ne bouge pas tout seul si on ne lui en donne pas l'ordre. Si je lis mon fichier, le curseur va se retrouver à la fin, lors d'une écriture sans bouger le curseur, l'écriture s'effectuera au début.

Pareil pour la lecture, si le curseur est à la fin et qu'on demande une lecture, il n'y aura rien à lire. Donc la propriété **Position** permet de spécifier l'index de ce curseur. Ici je le replace au début à chaque fois (0).

Mais attention, si je reprends l'écriture au début, le curseur ne s'occupe pas de ce qu'il y a avant, lorsqu'il va rencontrer un caractère déjà écrit dans le fichier il va purement et simplement le remplacer.

Faites bien attention donc et représentez-vous mentalement le trajet du curseur dans votre fichier pendant votre programme.

La lecture

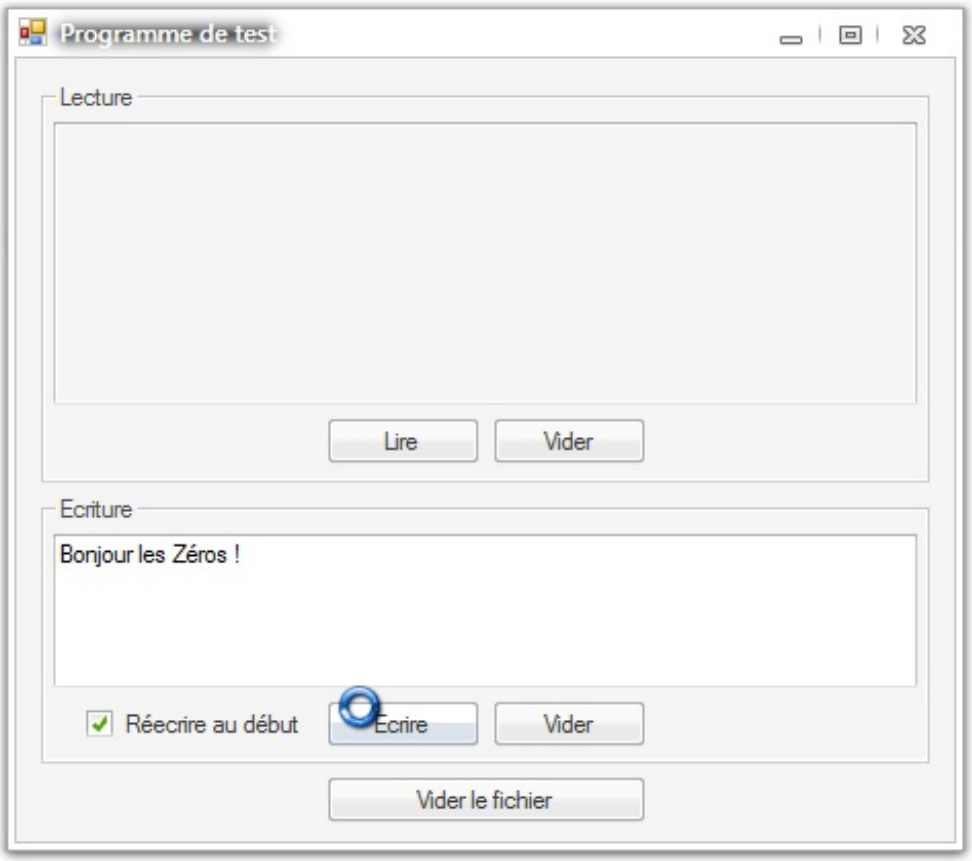
Reprenons l'évènement qui s'effectue lors du clic sur le bouton lire :

Code : VB.NET

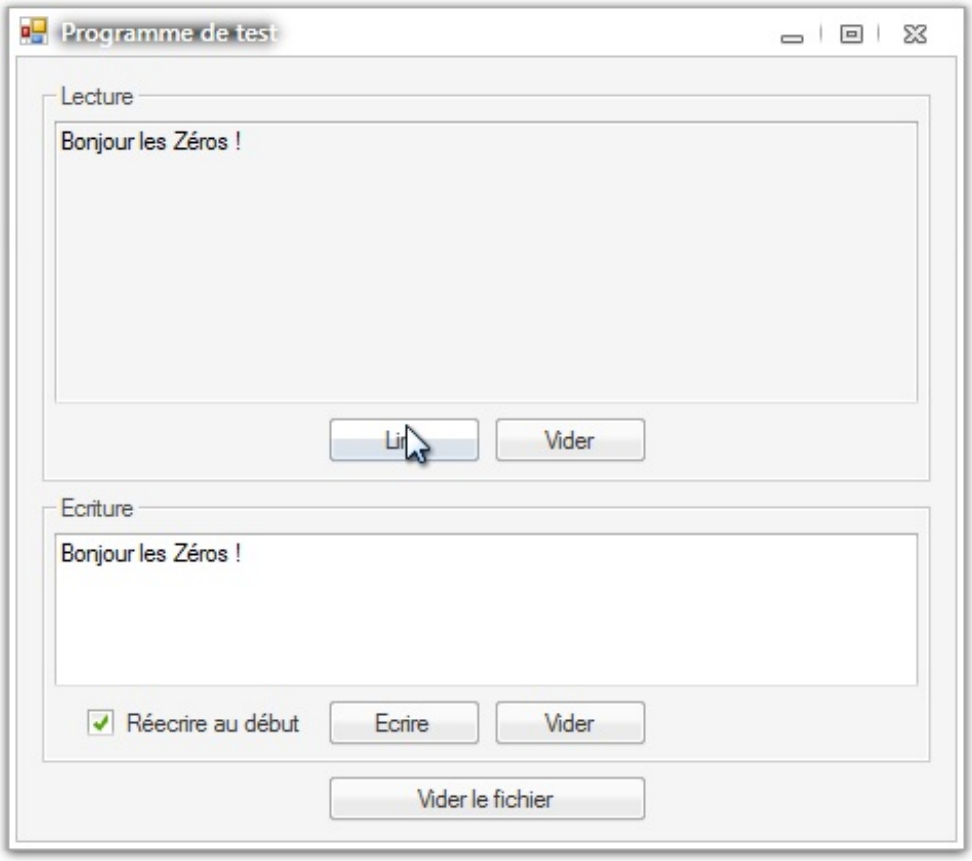
```
Private Sub BT_LIRE_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BT_LIRE.Click

        If MonFichier.CanRead() Then
            'Crée un tableau de Byte
            Dim Contenu(1024) As Byte
            'Lit 1024 bytes et les entre dans le tableau
            MonFichier.Position = 0
            MonFichier.Read(Contenu, 0, 1024)
            'L'affiche
            Me.TXT_LECTURE.Text = ""
            For Each Lettre As Byte In Contenu
                Me.TXT_LECTURE.Text += Chr(Lettre)
            Next
        End If

    End Sub
```

- Première ligne, le même principe que pour l'écriture, on effectue un petite vérification pour savoir si l'on peut effectuer notre lecture.
- On crée un tableau de **Byte** (comme l'écriture : 1025 cases)
- On place le *curseur* à la position 0 (début de mon fichier).
- On lit sur 1024 **bytes** (si le curseur rencontre la fin du fichier, la lecture s'arrête), et on place ce qui a été lu dans le tableau de **Bytes** déclaré juste avant.
- Puis un traditionnel For Each afin de parcourir toutes les cases de ce tableau de Bytes.
- On effectue une conversion **Numerique => Caractère** (soit Byte => ASCII grâce à la fonction **Chr()**) sinon vous ne liriez que des chiffres dans votre résultat ! On place le tous dans la textBox (grâce à **+=** on ajoute les caractères à la suite).

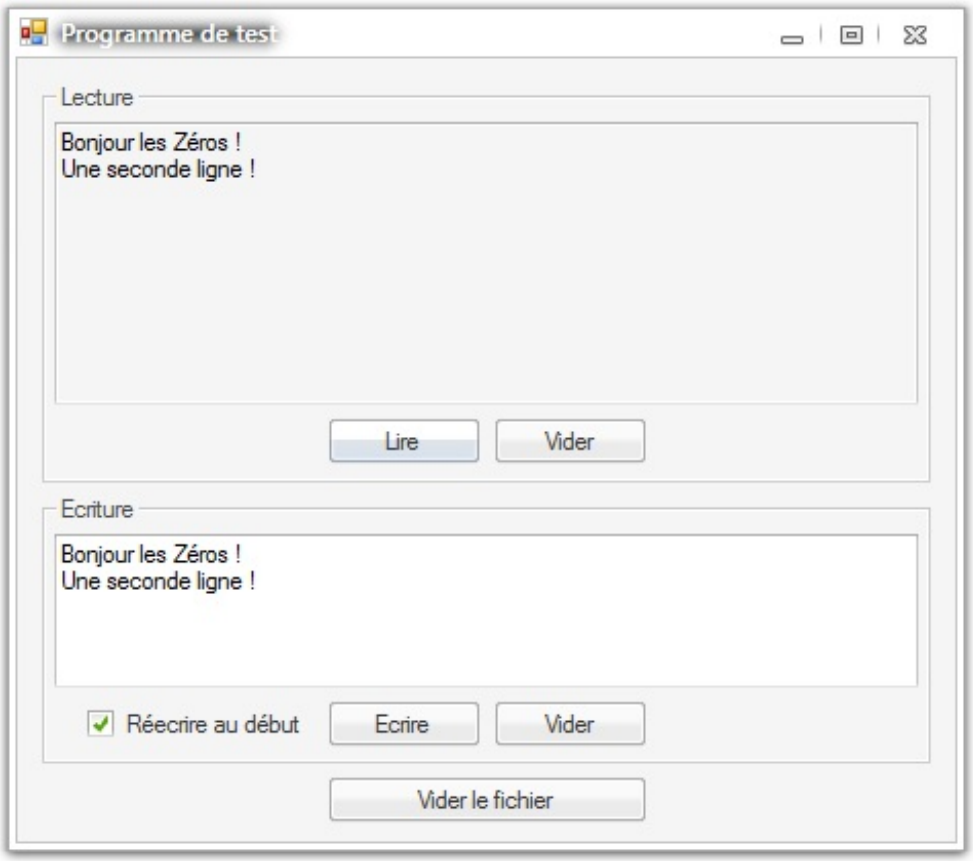
Eh bien voilà, ce qui nous donne en résultat de tests :



Une demande d'écriture dans notre fichier.
Résultat dans le fichier : "Bonjour les Zéros !"



Une demande de lecture, le fichier n'a pas changé, son contenu est toujours le même.



L'écriture d'une seconde ligne, le contenu du fichier est devenu :
"Bonjour les Zéros !
Une seconde ligne !"

J'en profite pour vous dire que les caractères permettant de matérialiser le retour à la ligne sont contenus dans la chaîne que vous récupérez de votre textBox, donc lorsqu'on demande une écriture de tout son contenu, le caractère est également écrit dans le fichier, le retour à la ligne s'effectue donc également dans le fichier sans manipulations supplémentaires.

Déjà une bonne chose de faite, ne partez pas ! On va apprendre de nouvelles fonctions et manipulations sur nos nouveaux amis les fichiers dans la partie suivante.

Les fichiers - Partie 2/2

La suite sur les fichiers :

Des notions supplémentaires qui peuvent être utiles.

Mais attention, le niveau monte d'un cran, accrochez-vous !

Plus loin avec nos fichiers

Allez, continuons donc sur notre lancée. Voyons les fonctions spécifiques aux fichiers plus en détail.

Tout d'abord la technique que je vous ai montrée utilise le principe du Stream. Autrement dit : Flux.
Dans le principe : le fichier est intégralement ouvert et inséré dans un objet de type Stream. Pendant le temps que le stream est ouvert (fichier ouvert par le programme), son écriture par une autre instance (un autre programme) est impossible.

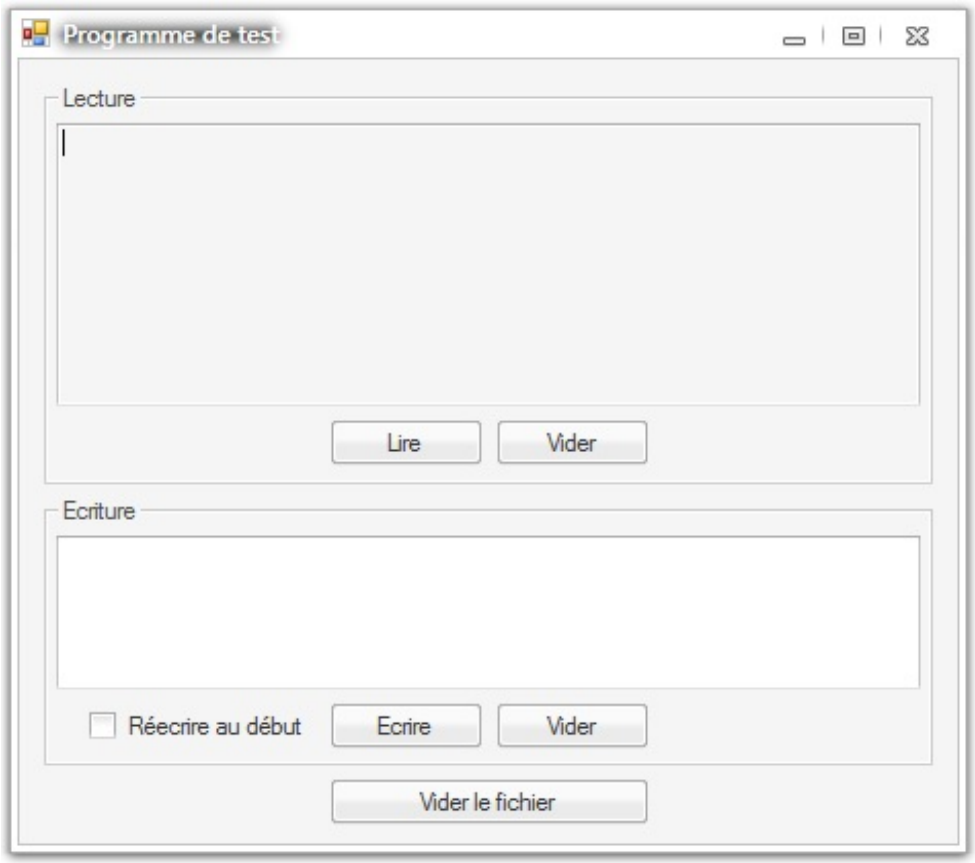
Cette technique comporte des avantages et des inconvénients : on peut être certain que le fichier ne sera pas modifié pendant le déroulement du programme, mais par contre, il est bloqué et donc plusieurs programmes ne peuvent pas travailler dessus en même temps.

Bref, je parie que vous voulez une autre technique.

La **classe File** vient à votre secours !

La classe File

Cette classe est pré-implémentée dans le Framework. On va créer le même programme de lecture / écriture avec cette classe.



La même interface donc, le code va légèrement changer :

Code : VB.NET

```
Imports System.IO

Public Class Form1

    Const PATHFICHIER As String = "Zero.txt"

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

```
            Me.TXT_LECTURE.Text = ""
        End Sub

        Private Sub BT_CLEARECRIRE_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles BT_CLEARECRIRE.Click
            Me.TXT_ECRITURE.Text = ""
        End Sub

        Private Sub BT_CLEAR_Click(ByVal sender As System.Object, ByVal
    e As System.EventArgs) Handles BT_CLEAR.Click
            File.WriteAllText(PATHFICHIER , "")
        End Sub

        Private Sub BT_LIRE_Click(ByVal sender As System.Object, ByVal e
    As System.EventArgs) Handles BT_LIRE.Click
            Me.TXT_LECTURE.Text = File.ReadAllText(PATHFICHIER )
        End Sub

        Private Sub BT_ECRIRE_Click(ByVal sender As System.Object, ByVal
    e As System.EventArgs) Handles BT_ECRIRE.Click
            If Me.CHK_DEBUT.Checked Then
         'Depuis le début
                File.WriteAllText(PATHFICHIER , Me.TXT_ECRITURE.Text)
            Else
         'A la suite
                File.AppendAllText(PATHFICHIER , Me.TXT_ECRITURE.Text)
            End If

        End Sub
    End Class
```

Il n'y a plus rien !?!

Eh bien oui, si on veut. La classe File a les outils nécessaires pour effectuer les actions que nous avions besoin.

Mais tu es stupide ! Pourquoi tu nous as ennuyé avec tes 500 lignes au chapitre précédent ?

Eh bien, je vous aurais montré qu'on aurait pu le faire ainsi, vous auriez réellement pris le temps de comprendre tout ce qui a été introduit au chapitre précédent ? (Les objets, le stream, les conversions de caractères).

Bon, cette classe nous permet de lire / écrire rapidement dans nos fichiers. Examinons quand même ces lignes.

Tout d'abord une variable constante de déclarée pour spécifier le Path que je vais utiliser pendant tout le programme : `Const PATHFICHIER As String = "Zero.txt"` . Path relatif bien évidemment.

`File.WriteAllText(PATHFICHIER , Me.TXT_ECRITURE.Text)` : La méthode WriteAllText de la classe File permet d'écrire du texte dans un fichier en **redémarrant du début** . Donc effacement du contenu précédent (ce que j'ai utilisé pour l'effacement du fichier).

`File.AppendAllText(PATHFICHIER , Me.TXT_ECRITURE.Text)` : LA méthode AppendAllText quant à elle écrit **à la suite** du fichier, donc je l'ai utilisé lorsque la checkbox est cochée.

Il nous reste finalement la lecture : `Me.TXT_LECTURE.Text = File.ReadAllText(PATHFICHIER )` . Une fonction cette fois-ci qui lit depuis le début et entre le tout dans un string. String que j'affiche directement via ma textbox.

Quelle simplification quand même, je vous rassure, par la suite nous utiliserons cette classe, nous nous concentrerons plus sur le fonctionnel des fichiers que sur comment effectuer nos manipulations dessus.

Découvrons d'autres manipulations

Bien, tout d'abord le légendaire **move**, autrement dit déplacement du fichier.

Code : VB.NET

```
File.Move(Source as string, Destination as string)
```

Vous pouvez bien évidemment utiliser des chemins relatifs / absolus ou mélanger les deux =).
Cette méthode est également utilisée pour **renommer** les fichiers, il suffit simplement d'effectuer le move avec deux noms différents, mais sur le même Path.

La copie :

Code : VB.NET

```
File.Copy(Source as string, Destination as string)
```

Même principe que la méthode précédente, vous n'avez cependant pas le droit d'attribuer le même nom à la source et à la destination.

La vérification de la présence du fichier :

Code : VB.NET

```
File.Exists(Fichier as string)
```

Fonction **très importante** ! Lorsque l'on va effectuer des manipulations, toujours vérifier la présence du fichier avant d'effectuer une action dessus ! Vous ne voulez pas vous retrouver avec une grosse erreur qui tache ! Renvoie une Boolean : True si présence du fichier, False dans le cas contraire.

Les répertoires

Bien, nous savons manipuler les fichiers assez bien je dois dire, du moins suffisamment pour les utilités que nous allons leur donner.

Attaquons alors maintenant les répertoires.

Bon, cette fois pas de stream ou autres, la classe Directory est la seule dans le namespace IO (Directory : répertoire).

Fonctions de modification

On va commencer par la fonction à utiliser avant toute chose :

La vérification :

Code : VB.NET

```
Directory.Exists(Path As String)
```

Renvoie un booléen encore une fois, bien évidemment très important ! On l'utilisera systématiquement !

La création de dossiers :

Code : VB.NET

```
Directory.CreateDirectory(Path As String)
```

Alors, cette méthode est assez magique. Elle va créer entièrement le Path spécifié. Je m'explique :
Parlons en path relatif : il n'y a actuellement aucun dossier dans le répertoire d'exécution de votre programme. Si en argument de la méthode je passe "Dossier1/SousDossier1/SousSousDossier1", il y aura 3 dossiers de créés, suivant l'arborescence définie :
Le dossier "Dossier1" sera créé directement dans le répertoire, le dossier "SousDossier1" sera crée dans "Dossier1" et finalement le dossier "SousSousDossier1" sera créé dans "SousDossier1". Le tout pour dire à quel point cette méthode peut se révéler pratique.

La suppression :

Code : VB.NET

```
Directory.Delete(Path As String, Recursif As Boolean)
```

Alors, ici nous avons un second argument en plus du path du dossier à supprimer ; il correspond à la récursivité. Si vous activez la récursivité, les dossiers et fichiers en "dessous" (dans l'arborescence des fichiers) du path que vous avez indiqués seront également supprimés, sinon si la récursivité n'est pas activée et que vous tentez de supprimer un dossier qui n'est pas vide, une erreur surviendra.
En résumé : la récursivité supprime le répertoire plus **l'intégralité de son contenu !**

Le légendaire Move :

Code : VB.NET

```
Directory.Move(PathSource As String, PathDest As String)
```

Même principe que pour les fichiers, avec les répertoires cette fois-ci : déplace le dossier et son contenu vers le nouveau Path.

Fonctions d'exploration

Bien, vous savez maintenant manipuler les fichiers ET les répertoires, mais il va falloir associer les deux pour pouvoir rendre vos programmes exportables et adaptables aux environnements.

Nous allons donc apprendre à chercher dans un dossier spécifié les sous-dossiers et les fichiers qu'il contient.
Bref, cela va nous permettre de pouvoir nous représenter notre arborescence. Nous allons également créer un petit programme permettant de représenter l'arborescence de notre disque.

Commençons donc avec les fonctions :

Rechercher tous les dossiers contenus dans le dossier spécifié :

Code : VB.NET

```
Directory.GetDirectories(Path as String)
```

Renvoie un **tableau** de string contenant le path de tous les dossiers qui sont contenus dans le dossier spécifié.

Cette fonction revoit un path absolu si vous lui en avez fourni un au départ, un path relatif dans le cas contraire. Attention de toujours utiliser le même type de Path !

Rechercher tous les fichiers contenus dans un dossier spécifié :

Code : VB.NET

```
Directory.GetFiles(Path as String)
```

Comme pour au-dessus, même remarque le path renvoyé correspond à celui que vous avez passé en argument. Renvoie les fichiers avec leur extension.

Un rapide bout de code permet de lister les fichiers présents en utilisant cette fonction :

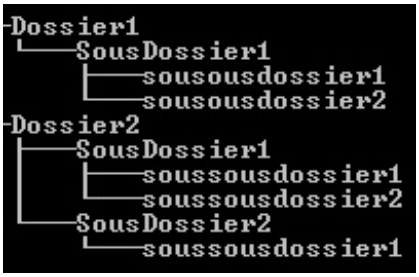
Code : VB.NET

```
        For Each Fichier As String In Directory.GetFiles("c:/")
                    MsgBox(Fichier)
                Next
```

Eh bien voilà.

Mini-TP : Lister notre arborescence

Tout d'abord, explorons notre arborescence avec une commande tout faite dans notre invite de commande Windows.
La commande shell (commande spécifique à Windows) s'appelle **Tree**. Elle donne un résultat de ce genre :



Vous n'avez pas besoin d'utiliser cette commande c'est pour vous montrer l'arborescence du dossier dans lequel nous allons faire notre mini-tp.

Nous allons donc retrouver notre arborescence de manière a se retrouver avec le même schéma, le tout grâce a un algorithme. Je vous ai déjà parlés du principe d'un algorithme. Eh bien nous allons devoir en trouver un pour pouvoir effectuer ce listage. Nous récupérerons les informations et les afficherons dans un TreeView (ça vous donnera l'occasion de découvrir un nouveau contrôle), spécifiquement conçu pour effectuer des arborescences (avec des parents et des enfants).

Pour résumer, dans le TreeView : un dossier correspondra à un noeud principal (on peut cliquer dessus pour le "déplier") et un fichier sera un noeud simple, pas de possibilité de le "dérouler".

C'est un programme très basique, sa base pourra être utilisée dans d'autres programmes qui nécessitent une exploration des répertoires.

Donc passons à l'algorithme. Je ne suis pas là pour vous apprendre les rudiments et normes de l'algorithmie, j'aimerai juste un peu de logique de votre part, peu importe comment vous vous représentez ce qu'il va y avoir à faire (schéma, texte, dessin, ...).

Le tout est de comprendre ce qu'on va devoir effectuer comme action, appeler comme fonctions.

Un algorithme version texte tout simple :

Parcourir le répertoire, pour chaque dossier ajouter un noeud principal, pour chaque fichier ajouter un noeud simple.
Répéter cette action pour chaque répertoire

(Attention, cet algorithme ne respecte pas les normes de l'algorithme, si vous voulez en savoir plus, de très bon tutos existent sur le SDZ)

Maintenant il va falloir l'adapter pour le rentrer dans notre code.

Le code VB :

Code : VB.NET

```
Imports System.IO

Public Class Form1

    Const RepertoireALister As String = "."

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Définit le premier noeud
        Me.TV_ARBORESCENCE.TopNode =
Me.TV_ARBORESCENCE.Nodes.Add(RepertoireALister, RepertoireALister)
```

```
        'Arborescence du premier noeud
        For Each Repertoire As String In
Directory.GetDirectories(RepertoireALister)
            Me.TV_ARBORESCENCE.TopNode.Nodes.Add(Repertoire,
Path.GetFileName(Repertoire))
            'Récursif
            ListeArborescenceDossier(Repertoire,
Me.TV_ARBORESCENCE.TopNode)
        Next
        'Fichiers du premier noeud
        For Each Fichier As String In
Directory.GetFiles(RepertoireALister)

Me.TV_ARBORESCENCE.TopNode.Nodes.Add(Path.GetFileName(Fichier))
        Next
    End Sub

    Sub ListeArborescenceDossier(ByVal RepertoireActuel As String,
ByVal NodeActuel As TreeNode)
        'Recupère le node dans lequel on est
        Dim Node As TreeNode = NodeActuel.Nodes(RepertoireActuel)
        'Répertoires de ce noeud
        For Each Repertoire As String In
Directory.GetDirectories(RepertoireActuel)
            Node.Nodes.Add(Repertoire, Path.GetFileName(Repertoire))
            'Récursif
            ListeArborescenceDossier(Repertoire, Node)
        Next
        'Fichiers de ce noeud
        For Each Fichier As String In
Directory.GetFiles(RepertoireActuel)
            Node.Nodes.Add(Path.GetFileName(Fichier))
        Next
    End Sub

End Class
```

Expliquons un peu le tout. Tout d'abord Node en anglais signifie Nœud.
Tout d'abord, le répertoire que je dois explorer en constante, vous pouviez bien évidemment créer une textbox demandant à l'utilisateur quel dossier lister. Le path que j'ai utilisé est "." cela signifie le dossier courant, c'est un **Path Relatif**.

Vient le load, je crée d'office un **TopNode** autrement dit "le nœud le plus haut", le nœud principal de notre **treeview**. J'en profite pour créer un noeud avec : `Me.TV_ARBORESCENCE.Nodes.Add()` . En premier argument de cette fonction la "clé" pour identifier le nœud dans le treeview, cette clé doit avoir un nom **unique**, et en second le texte qui sera affiché sur mon nœud.

Ensuite, la petite boucle que je vous ai montrée plus haut : je parcours tout les répertoires dans le répertoire à lister, je les ajoute en tant que nœud principal avec comme clé leur path entier (exemple : "./Dossier1/SousDossier1") donc un nom qui est unique, et en texte le nom du dossier simplement. Nom de dossier que j'ai récupéré en utilisant la classe **Path** qui donne des méthodes et fonction pour manipuler les chemins. J'ai utilisé la fonction **GetFileName** qui renvoie le nom du fichier ou le nom d'un dossier contenu dans un Path.

Puis j'appelle une méthode que je vous exposerai juste après.

Quand il n'y a plus de dossiers on passe aux fichiers, sur le même principe sauf que ces nœuds n'auront pas de nœud enfants. Donc pas dépliables.

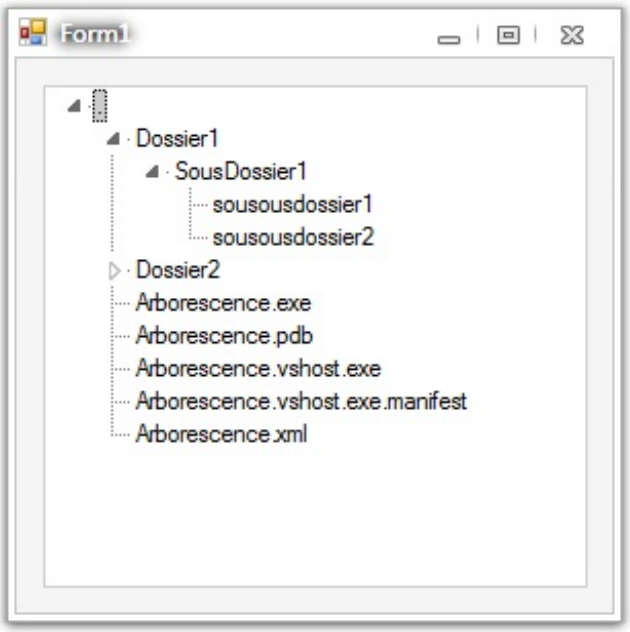
La méthode que j'ai mise juste après, eh bien c'est elle qui va nous permettre d'effectuer la récursivité de notre algorithme à travers tous les sous dossiers.

Sans elle, on aurait seulement le niveau 0 de notre arborescence de listé (les dossiers et fichiers du répertoire principal) et pas

plus loin.

Eh bien pour ce qui est de la fonction, elle effectue exactement la même chose que ce que je viens d'expliquer mais où le nom du répertoire et le nœud dans lequel on se trouve actuellement sont passés en paramètres de façon à permettre de la rappeler dynamiquement et pour qu'elle puisse s'adapter aux différents niveaux de l'arborescence.

Résultat :



Je tien juste à vous dire que si vous prenez un peu de temps, essayez de comprendre le fonctionnement de ce programme étape par étape (commencez par un seul niveau d'arborescence), vous allez comprendre la démarche qu'il effectue et ce sera un premier et un grand pas vers de notions de programmation plus complexe que nous allons aborder dans la partie 3 de ce tutoriel.
Autre conseil pour vous éclairer le programme : créez des variables intermédiaires dans lesquelles vous vous habituerez à trouver le bon type de variable à employer, les méthodes disponibles sur ce type, pour finalement arriver à tout rassembler tout en le laissant clair à vos yeux.

Un fichier bien formaté

Bien, passons aux fichiers de configuration.

Peut-être que quelques-uns d'entre vous ont déjà vu les fichiers de configuration standard de Windows : les Fichiers .ini

Ces fichiers ont été utilisés par Windows pour définir les paramètres de configuration. C'est de simples fichiers contenant du texte mais au lieu d'avoir l'extension basique de texte : .txt, ils ont une extension .ini

Petite parenthèse sur les extensions : elles ne définissent pas obligatoirement le contenu du fichier, les fichiers .jpg contiennent habituellement des images et ont l'habitude d'être ouverts par des logiciels de dessin ou de visualisation d'image mais ils peuvent très bien contenir du texte et être ouvert avec le bloc-notes.

Les fichiers .ini contiennent donc du texte mais formaté d'une certaine manière, nous allons l'étudier ici.

Exemple de mon fichier Win.ini :

Code : Autre

```
; for 16-bit app support
[fonts]
[extensions]
[mci extensions]
[files]
[Mail]
MAPI=1
[MCI Extensions.BAK]
3g2=MPEGVideo
3gp=MPEGVideo
3gp2=MPEGVideo
3gpp=MPEGVideo
aac=MPEGVideo
adt=MPEGVideo
adts=MPEGVideo
m2t=MPEGVideo
m2ts=MPEGVideo
m2v=MPEGVideo
m4a=MPEGVideo
m4v=MPEGVideo
mod=MPEGVideo
mov=MPEGVideo
mp4=MPEGVideo
mp4v=MPEGVideo
mts=MPEGVideo
ts=MPEGVideo
tts=MPEGVideo
```

Explications :

Le contenu d'un fichier de configuration .ini contient 3 types de lignes :

- Les lignes commençant par ";" sont des commentaires, elles ne sont pas prises en compte pendant le traitement du fichier.
- Les lignes où il y a des crochets : "[" "]" définissent une nouvelle section. Cela permet d'organiser un minimum notre fichier .ini
- Finalement les lignes de clé, les plus importantes, elles contiennent les variables que nous stockons. Par exemple "MAPI=1" signifie que la variable (ou clé) MAPI est égale à 1.

Bon, vous voyez maintenant le principe d'un fichier de configuration. A quoi diable va-t-il nous servir ? Eh bien simplement à garder des paramètres du programme même s'il y a eu un arrêt de se dernier.

Bien, vous voilà donc avec une petite norme à respecter pour stocker vos informations de configuration (ça ne fait pas de mal de temps en temps).

Nous allons passer à un TP conséquent et qui va vous demander de réviser vos notions sur les fichiers. C'est juste après.

TP : ZBackup

Eh bien chers amis zéros, on va attaquer un TP de taille.

Ce TP aura pour but de nous faire développer un programme (nommé ZBackup par mes soins) qui aura pour but d'effectuer des sauvegardes périodiques de dossiers spécifiés.

Je ne vous en dis pas plus, on attaque tout de suite.

Le cahier des charges

Eh bien c'est parti pour le cahier des charges !

En premier lieu je vais vous décrire ce que notre programme sera susceptible de faire.

Donc, tout d'abord ce programme est un programme d'auto backup, autrement dit : sauvegarde automatique.

Ce programme sera capable de sauvegarder périodiquement un ou des dossiers que nous spécifierons dans une liste.

Pour commencer nous n'allons pas chercher bien loin : nous allons juste créer un répertoire dans lequel nous entasserons nos sauvegardes (répertoire spécifié par l'utilisateur). Essayez de ranger et de trier les différentes sauvegardes, pourquoi pas avec la date et l'heure.

Attention à ce point, les dossiers n'ont pas de fonction permettant leur copie, vous allez devoir copier les fichiers individuellement, essayez de trouver un algorithme, servez-vous de notre mini-tp sur l'arborescence.

Je vous laisse libre court à votre imagination, à vous de voir si une seconde fenêtre est préférable pour spécifier la configuration, etc ...

En parlant de configuration, après nos deux chapitres sur les fichiers, j'aimerais que vous sauvegardiez les paramètres de configuration de ce petit programme dans un fichier .ini. Je vous laisse également choisir la structure qu'aura ce fichier, les choses que vous allez avoir à insérer dedans, etc ...

Pour le choix des dossiers, je ne vous en avais pas parlés avant mais un petit module très pratique existe : le **FolderBrowserDialog**.

Dans la boite à outils, section boites de dialogue. Ce module ouvre une boite de dialogue, demandant à l'utilisateur de sélectionner un dossier. Il a également la possibilité d'en créer un par la même occasion. Vous pouvez récupérer le dossier sélectionné avec **FolderBrowserDialog.SelectedPath** où **FolderBrowserDialog** est le nom de votre contrôle.

Pour ce qui est de la liste des dossiers à sauvegarder, vous pouvez les insérer dans une listbox ou une textbox multilignes, au choix.

Pour le reste, je vous laisser agrémenter au choix et selon vos goûts..

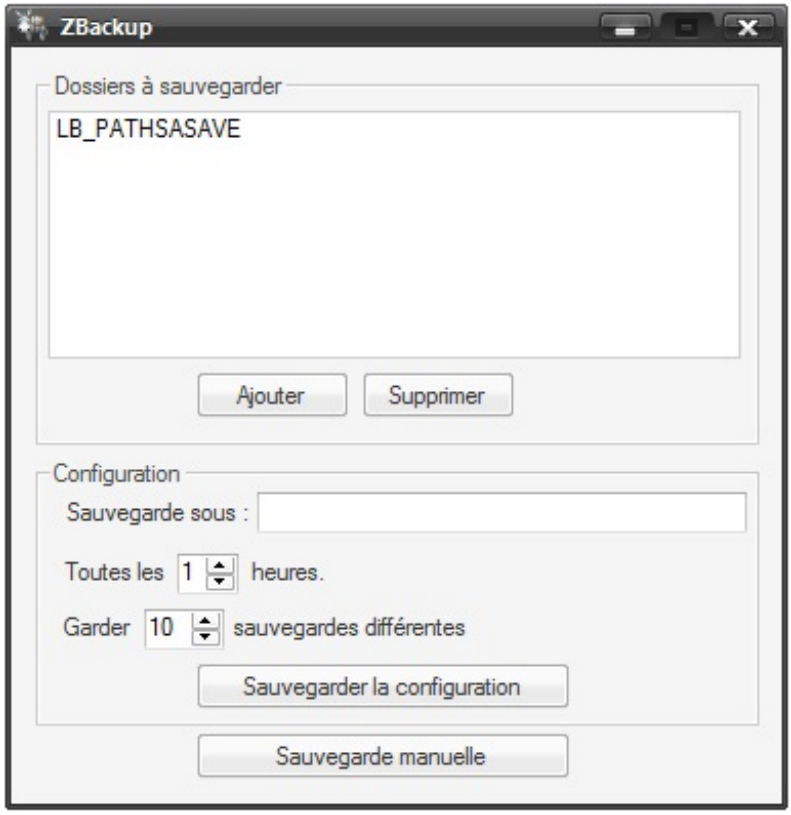
Je dois dire que vous avez toutes les compétences et les méthodes de réflexion (savoir trouver les propriétés qui vous seront utiles) requise. Essayez de ne pas vous décourager trop rapidement et ne pas aller trop vite à la correction.

Eh bien, amis zéros, au travail !

Correction

Alors, en espérant que vous avez donnés le meilleur de vous même.

Commençons par mon interface et par les fonctionnalités que mon programme contient.



Comme vous le voyez, j'ai une **listbox** qui me permet de lister les répertoires dont je veux la sauvegarde.
Viens ensuite un bouton d'ajout et de suppression des répertoires, l'ajout se fait par **FolderBrowserDialog**, la suppression par lignes sélectionnées dans la **listbox**.
Un petit menu de configuration dans lequel on spécifie le dossier où placer les sauvegardes. Lors du clic sur la **textbox** un **folderbrowserdialog** s'ouvre et c'est sa sélection qui remplira la **textbox**.
Des **NumericUpDown** (un contrôle) permettant de spécifier un nombre avec le clavier ou grâce à des boutons haut et bas.

Puis un bouton pour enregistrer la configuration et un second pour effectuer une sauvegarde manuelle.

Bon je vais vous montrer le code tout de suite, on va développer section par section.

Secret (cliquez pour afficher)

Code : VB.NET

```
Imports System.IO

Public Class ZBackup

    'Définit les constantes
    Const FichierIni As String = "Zbackup.ini"
    Const LignesFichierIni As Integer = 6
    Const CleSavePath As String = "SavePath"
    Const CleTempSave As String = "TempSave"
    Const CleNbSaves As String = "NbSaves"
    Const ClePaths As String = "Paths"

    Private Sub ZBackup_Load(ByVal sender As System.Object, ByVal
```

```
e As System.EventArgs) Handles MyBase.Load

        'Configure le timer
        Me.TIM_SAVE.Interval = Me.NUM_SAVETIME.Value * 3600000
'Convertir une heure une milisecondes
        Me.TIM_SAVE.Enabled = True

        'Recupère la configuration enregistrée
        If RecupereInfosFichierIni() Then
            'Effectue d'office une sauvegarde
            Sauvegarde()
        End If

    End Sub

#Region "Interface"

    Private Sub TXT_SAVEPATH_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles TXT_SAVEPATH.Click

        'Ajoute la ligne seulement si un dossier a été selectioné
dans le dialogue
        If Me.BD_DOSSIER.ShowDialog() Then
            Me.TXT_SAVEPATH.Text = Me.BD_DOSSIER.SelectedPath
        End If

    End Sub

    Private Sub BT_AJOUT_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles BT_AJOUT.Click

        'Ajoute la ligne seulement si un dossier a été selectionné
dans le dialogue
        If Me.BD_DOSSIER.ShowDialog Then

Me.LB_PATHSASAVE.Items.Add(Me.BD_DOSSIER.SelectedPath)
        End If

    End Sub

    Private Sub BT_SUPPR_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles BT_SUPPR.Click

        'Vérifie si une ligne est sélectionnée dans la listbox
        If Not Me.LB_PATHSASAVE.SelectedItem Is Nothing Then

Me.LB_PATHSASAVE.Items.Remove(Me.LB_PATHSASAVE.SelectedItem)
        Else
            MsgBox("Selectionnez un path à supprimer")
        End If

    End Sub

    Private Sub BT_SAVECFG_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles BT_SAVECFG.Click
        SauvegardeFichierIni()
    End Sub

    Private Sub TIM_SAVE_Tick(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles TIM_SAVE.Tick
        'Sauvegarde avec le timer
        Sauvegarde()
    End Sub

    Private Sub BT_MANUSAVE_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles BT_MANUSAVE.Click
        'Sauvegarde manuelle
        Sauvegarde()
```

```
        End Sub

    #End Region

    #Region "FichierIni"

        Sub SauvegardeFichierIni()
            'Verification sur le path de sauvegarde
            If Me.TXT_SAVEPATH.Text = "" Then
                MsgBox("Veuillez selectionner un path de sauvegarde")
            ElseIf Not Directory.Exists(Me.TXT_SAVEPATH.Text) Then
                MsgBox("Path de sauvegarde invalide")
            Else
                'La fonction recrée le fichier quoi qu'il arrive
                File.WriteAllLines(FichierIni,
CreeStructureFichierIni(Me.TXT_SAVEPATH.Text,
Me.NUM_SAVETIME.Value, Me.NUM_NBSAVE.Value,
Me.LB_PATHSASAVE.Items))
            End If
        End Sub

        Function CreeStructureFichierIni(ByVal SavePath As String,
ByVal TempsSave As Integer, ByVal Nbsaves As Integer, ByVal
PathsASave As ListBox.ObjectCollection) As String()

            'Crée un tableau du nombre de lignes requises
            Dim FichierIni(LignesFichierIni + PathsASave.Count) As
String
            'Replit la structure du fichier ini
            FichierIni(0) = ";Fichier de configuration du Zbackup"
            FichierIni(1) = "[configuration]"
            FichierIni(2) = CleSavePath & "=" & SavePath
            FichierIni(3) = CleTempSave & "=" & TempsSave
            FichierIni(4) = CleNbSaves & "=" & Nbsaves
            FichierIni(5) = ""
            FichierIni(6) = "[paths]"
            'Rempli dynamiquement les paths souhaités
            Dim Compteur As Integer = LignesFichierIni
            For Each Path As String In PathsASave
                Compteur += 1
                FichierIni(Compteur) = ClePaths & Compteur -
LignesFichierIni & "=" & Path
            Next

            Return FichierIni

        End Function

        Function RecupereCleFichierIni(ByVal Cle As String) As String

            For Each Ligne As String In File.ReadAllLines(FichierIni)
                'Découpe la ligne au niveau de "=" (s'il existe),
                'compare la premiere partie de la ligne (soit la clé)
                If Ligne.Split("=")(0) = Cle Then
                    'Recupère la seconde partie de la ligne (soit la
valeur)
                    Return Ligne.Split("=")(1)
                End If
            Next
            'Sinon ne retourne rien
            Return ""

        End Function

        Function RecupereInfosFichierIni() As Boolean

            'Verification de la présence du .ini
            If File.Exists(FichierIni) Then
```

```
            If File.Exists(FichierIni) Then
                'Recuperation
                Dim SavePath As String =
RecupereCleFichierIni(CleSavePath)
                Dim TempSAve As String =
RecupereCleFichierIni(CleTempSave)
                Dim NbSaves As String =
RecupereCleFichierIni(CleNbSaves)
                Dim Paths(100) As String
                Dim i As Integer = 0 '0 car le premier path est à 1 et
on incrémente avant
                Do
                    i += 1
                    Paths(i - 1) = RecupereCleFichierIni(ClePaths & i)
                Loop While Paths(i - 1) <> ""
                'Donc nombre de paths valides : i-1

                'Verification
                If (SavePath <> "") And (TempSAve <> "") And (NbSaves
<> "") And (i - 1 > 0) Then
                    'Attribution
                    Me.TXT_SAVEPATH.Text = SavePath
                    Me.NUM_NBSAVE.Value = NbSaves
                    Me.NUM_SAVETIME.Value = TempSAve
                    'Nettoie le LB puis le remplit
                    Me.LB_PATHSASAVE.Items.Clear()
                    For j As Integer = 0 To i - 1
                        Me.LB_PATHSASAVE.Items.Add(Paths(j))
                    Next
                Else
                    'Sinon notification
                    MsgBox("Le fichier " & FichierIni & " est
corrompu, utilisation des paramètres par défaut")
                    Return False
                End If
            Else
                'Sinon notification
                MsgBox("Le fichier " & FichierIni & " n'a pas été
trouvé, utilisation des paramètres par défaut")
                Return False
            End If

            Return True

        End Function

    #End Region

    #Region "Sauvegarde"

        Sub Sauvegarde()

            'Vérifie les paramètres
            If Directory.Exists(Me.TXT_SAVEPATH.Text) Then
                'Vérifie le nombre de sauvegardes
                'Supprime la plus vieille si limite atteinte
                NettoieNbSaves()

                'Si le dernier caractère de la chaine est un "\" on le
supprime
                If Me.TXT_SAVEPATH.Text.EndsWith("\") Then
                    Me.TXT_SAVEPATH.Text.Trim("\")
                End If
                'Crée le dossier de sauvegarde avec un nom spécifié
                'Supprime les / et : de la date et de l'heure
                Dim DossierSave As String = Me.TXT_SAVEPATH.Text &
"\Sauvegarde du " & Date.Now.ToShortDateString.Replace("/", "-") &
" a " & Date.Now.ToShortTimeString.Replace(":", "-")
```

```
            If Not Directory.Exists(DossierSave) Then 's'il
exciste : 2saves dans la même minute, on ne la fait pas
                Directory.CreateDirectory(DossierSave)
                'Pour chaque path demandé, copie son dossier
                For Each PathASave As String In
Me.LB_PATHSASAVE.Items
                    If Directory.Exists(PathASave) Then
                        CopieDossier(New DirectoryInfo(PathASave),
New DirectoryInfo(DossierSave & "\" &
Path.GetFileName(PathASave)))
                    End If
                Next
            End If
        Else
            MsgBox("Sauvegarde échouée : le path de sauvegarde est
invalide, veuillez le redéfinir")
        End If

    End Sub

    Sub NettoieNbSaves()

        Dim Compteur As Integer = 1
        For Each Repertoire As String In
Directory.GetDirectories(Me.TXT_SAVEPATH.Text)
            'Si le repertoire est un répertoire de sauvegarde
            If Path.GetFileName(Repertoire).Contains("Sauvegarde")
Then
                'Incrementation du compteur
                Compteur += 1
            End If
        Next

        If Compteur > Me.NUM_NBSAVE.Value Then
            'Détermination du plus ancien
            Dim PlusAncien As DirectoryInfo
            Dim DatePlusAncienne As Date = Date.Now
            For Each Repertoire As String In
Directory.GetDirectories(Me.TXT_SAVEPATH.Text)
                'Si le repertoire est un répertoire de sauvegarde
                If
Path.GetFileName(Repertoire).Contains("Sauvegarde") Then
                    'Si le répertoire est plus ancien de que le
précédent
                    If (Directory.GetCreationTime(Repertoire) <
DatePlusAncienne) Then
                        'On le place en plus ancien
                        DatePlusAncienne =
Directory.GetCreationTime(Repertoire)
                        PlusAncien = New DirectoryInfo(Repertoire)
                    End If
                End If
            Next

            'Supprime le plus vieux
            If PlusAncien.Exists Then
                PlusAncien.Delete(True)
            End If
        End If

    End Sub

    Sub CopieDossier(ByVal DossierSource As DirectoryInfo, ByVal
DossierDesination As DirectoryInfo)

        'Crée le dossier
        DossierDesination.Create()
```

```
            'Copie les fichiers
            For Each Fichier As FileInfo In DossierSource.GetFiles()

        Fichier.CopyTo(Path.Combine(DossierDesination.FullName,
        Fichier.Name))
            Next

            'Recommence pour les sous-repertoires
            For Each SousRepertoire As DirectoryInfo In
        DossierSource.GetDirectories()
                CopieDossier(SousRepertoire,
        DossierDesination.CreateSubdirectory(SousRepertoire.Name))
            Next

        End Sub

        #End Region

        End Class
```

Eh bien, ça devient conséquent.

Comme vous pouvez le voir dès la première ligne, j'ai essayé de rendre le programme "flexible". Autrement dit, il me suffit de changer les constantes pour changer le nom du fichier ini par exemple, c'est cette constante qui est utilisée à chaque fois qu'une fonction demande le nom de ce fichier.

Trois grandes sections se distinguent :

- L'interface, contenant la réaction aux boutons, etc ...
- Le fichier ini, contenant tout ce qui touche à la configuration.
- Finalement la sauvegarde.

Une rapide vue d'ensemble du fonctionnement :

- Récupération de la configuration
- - Si elle n'existe pas création du fichier ini
  - Si elle est corrompue, recréation du fichier ini
- A chaque tick de timer (timer configuré sur le temps souhaité entre 2 sauvegardes), on effectue la sauvegarde
- Avec le bouton sauvegarde manuelle la même action est réalisée
- Le sauvegarde consiste a créer un dossier sous la forme "**Sauvegarde du DD-MM-AAAA a HH-MM**"
- Puis copie l'intégralité des dossiers en respectant leur arborescence.

L'interface

Voilà donc, commençons par analyse le plus simple : l'interface.

Première information : l'ouverture de la **folderbrowserdialog** lors du clic sur la textbox.
Eh bien, rien de sorcier, l'évènement clic de la textbox !

Code : VB.NET

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

```
            'Ajoute la ligne seulement si un dossier a été selectioné
    dans le dialogue
            If Me.BD_DOSSIER.ShowDialog() Then
                Me.TXT_SAVEPATH.Text = Me.BD_DOSSIER.SelectedPath
            End If

        End Sub
```

Eh bien lors du clic on se sert du contrôle "BD_DOSSIER" qui est mon **FolserBrowserDialog**. Vu que c'est une boite de dialogue, c'est la fonction **ShowDialog** qui est appellée. Puis on récupère la sélection avec le propriété **SelectedPath**.

Le bouton d'ajout à le même principe mais il agit sur la listbox, sur le système que le treeview que nous avons étudiés auparavant, il faut créer, non plus des nodes, mais des items.

Code : VB.NET

```
Me.LB_PATHSASAVE.Items.Add(Me.BD_DOSSIER.SelectedPath)
```

C'est dans le membre "**Items**" que les fonctions spécifiques a ces objets sont trouvables. La méthode **Add()** permet d'ajouter un item, avec comme valeur le dossier sélectionné.

Pour la suppression :

Code : VB.NET

```
    'Vérifie si une ligne est sélectionnée dans la listbox
            If Not Me.LB_PATHSASAVE.SelectedItem Is Nothing Then

    Me.LB_PATHSASAVE.Items.Remove(Me.LB_PATHSASAVE.SelectedItem)
            Else
                MsgBox("Selectionnez un path à supprimer")
            End If
```

Vous vous apercevez qu'une vérification est faite pour voir si une ligne est sélectionnée avec `If Not Me.LB_PATHSASAVE.SelectedItem Is Nothing Then` . Vous vous apercevez que je n'utilise pas le symbole "**<>**" pour dire différent mais "**not ... is nothing**" c'est une autre notation plus littérale, tout dépend des goûts de chacun.
Ensuite on supprime avec **Items.Remove** en passant comme paramètre la ligne selectionnée.

Pour les autres boutons, la sauvegarde des paramètres appelle la méthode **SauvegardeFichierIni()** que nous allons étudier. Le timer et la sauvegarde manuelle appellent la méthode **Sauvegarde()** que nous allons aussi étudier.

Sauvegarde en fichier .ini

Attaquons tout de suite la sauvegarde.

Pour ce qui est de cette sauvegarde, je vérifie la présence d'un dossier dans la textbox et si ce dossier est valide. Ensuite j'appelle la fonction `File.WriteAllLines(FichierIni, CreeStructureFichierIni(Me.TXT_SAVEPATH.Text, Me.NUM_SAVETIME.Value, Me.NUM_NBSAVE.Value, Me.LB_PATHSASAVE.Items))` qui s'occupe de créer un fichier et d'entrer dedans un tableau de string (une case de tableau pour une ligne).
En premier paramètre, le fichier de destination, c'est notre constante avec le nom du fichier ini. Le second, autrement dit le tableau de string, c'est une fonction que nous allons étudier tout de suite :

Code : VB.NET

```
Function CreeStructureFichierIni(ByVal SavePath As String, ByVal
TempsSave As Integer, ByVal Nbsaves As Integer, ByVal PathsASave As
ListBox.ObjectCollection) As String()

        'Crée un tableau du nombre de lignes requises
        Dim FichierIni(LignesFichierIni + PathsASave.Count) As
String
        'Replit la structure du fichier ini
        FichierIni(0) = ";Fichier de configuration du Zbackup"
        FichierIni(1) = "[configuration]"
        FichierIni(2) = CleSavePath & "=" & SavePath
        FichierIni(3) = CleTempSave & "=" & TempsSave
        FichierIni(4) = CleNbSaves & "=" & Nbsaves
        FichierIni(5) = ""
        FichierIni(6) = "[paths]"
        'Rempli dynamiquement les paths souhaités
        Dim Compteur As Integer = LignesFichierIni
        For Each Path As String In PathsASave
            Compteur += 1
            FichierIni(Compteur) = ClePaths & Compteur -
LignesFichierIni & "=" & Path
        Next

        Return FichierIni

    End Function
```

Les valeurs passées en paramètres auraient pu être remplacées par des récupérations directement à l'intérieur de la fonction.
Bref. Un tableau est crée avec comme taille le nombre de lignes initiales plus le nombre de paths à insérer.
Pour les premières lignes, j'écris manuellement dedans les premières clés. Ce qui nous donne dans le fichier ini une fois crée :

Code : Autre

```
;Fichier de configuration du Zbackup
[configuration]
SavePath=C:\Save
TempSave=1
NbSaves=3

[paths]
Paths1=C:\ASave
```

Deux sections donc : une configuration, une autre paths.
La section configuration contient le répertoire où sauvegarder, le temps entre 2 saves et le nombre de saves max.
La section paths contient les différents paths, tous avec un numéro différent. Les techniques peuvent différer, il aurait été possible de mettre tous les paths sur la même ligne, séparés par des ";".

Bref. La création du fichier n'est pas sorcier, le tableau de variables "FichierIni()" est renvoyée et est écrite dans le fichier.

Maintenant que vous avez vus comment le remplir, voyons comment récupérer les valeurs.

Donc pour cela une petite fonction à laquelle on passe en paramètre la clé à récupérer.

Code : VB.NET

```
    Function RecupereCleFichierIni(ByVal Cle As String) As String

            For Each Ligne As String In File.ReadAllLines(FichierIni)
                'Découpe la ligne au niveau de "=" (s'il existe),
                'compare la premiere partie de la ligne (soit la clé)
                If Ligne.Split("=")(0) = Cle Then
                    'Recupère la seconde partie de la ligne (soit la
valeur)
                    Return Ligne.Split("=")(1)
                End If
            Next
            'Sinon ne retourne rien
            Return ""

        End Function
```

Principe de cette fonction : on parcours toutes les lignes du ini, à chaque ligne on la découpe grâce à la fonction **Split()**.
La fonction split s'applique sur une chaine de caractères, elle permet de "découper" cette chaine à chaque occurrence du caractère ou de la chaine passée en argument. Donc exemple :
Pour une chaine de caractère sous la forme "Cle1=Valeur1" un `Split("=")` donnera un tableau de string sous la forme.
Tableau(0) = Cle1
Tableau(1) = Valeur1

Donc un bête et méchant test sur le tableau(0) qui est retourné avec la clé souhaitée nous indique la ligne contenant la clé voulue. Une fois cette ligne atteinte, le tableau(1), celui contenant la valeur est retourné.
Si la clé n'est pas trouvée, on retourne "".

Il fallait y penser.

Sauvegarde

Attaquons tout de suite le principe de la sauvegarde.

La méthode **Sauvegarde()** effectue diverses vérification sur la présence des dossiers, elle crée le dossier dans lequel la sauvegarde va être placée et lance la méthode **CopieDossier()** que voici :

Code : VB.NET

```
    Sub CopieDossier(ByVal DossierSource As DirectoryInfo, ByVal
    DossierDesination As DirectoryInfo)

            'Crée le dossier
            DossierDesination.Create()

            'Copie les fichiers
            For Each Fichier As FileInfo In DossierSource.GetFiles()
                Fichier.CopyTo(Path.Combine(DossierDesination.FullName,
    Fichier.Name))
            Next

            'Recommence pour les sous-repertoires
            For Each SousRepertoire As DirectoryInfo In
    DossierSource.GetDirectories()
                CopieDossier(SousRepertoire,
    DossierDesination.CreateSubdirectory(SousRepertoire.Name))
            Next

        End Sub
```

Cette méthode prend comme arguments des variables de type DirectoryInfo. Ce n'est pas des variables communes : ce sont des objets. Il faut donc les instancier avec un **new**.

Lors de l'appel de la ligne avec `CopieDossier(New DirectoryInfo(PathASave), New DirectoryInfo(DossierSave & "\" & Path.GetFileName(PathASave)))` , j'instance deux objets avec comme paramètre les chemins des dossiers voulus.

Une fois dans la méthode de copie, un dossier est tout d'abord crée, les fichiers contenus y sont copiés également puis cette action est répétée pour tous ses sous répertoires. De la même manière que le treeview du chapitre précédent.

Il y a finalement la méthode de nettoyage des sauvegardes (si on demande qu'un certain nombre de sauvegardes). Elle parcourt les répertoires crées, récupère la date de création de chacun, identifie le plus vieux et le supprime en utilisant la récursivité de la méthode Directory.Delete.

Eh bien voilà. Le code décortiqué.

Allons un peu plus loin.

Récapitulatif fichier ini

Bien, procédons à un récapitulatif des fonctions que vous allez pouvoir utiliser dans vos futurs programmes pour créer et gérer un fichier .ini, je ne pense pas revenir dessus par la suite, autant tout résumer tout de suite.

La création

Tout d'abord pour la création du fichier ini.

Deux manières de procéder : la création manuelle en utilisant une fonction du genre :

Code : VB.NET

```
Function CreeStructureFichierIni(ByVal SavePath As String, ByVal
TempsSave As Integer, ByVal Nbsaves As Integer, ByVal PathsASave As
ListBox.ObjectCollection) As String()

        'Crée un tableau du nombre de lignes requises
        Dim FichierIni(LignesFichierIni + PathsASave.Count) As
String
        'Replit la structure du fichier ini
        FichierIni(0) = ";Fichier de configuration du Zbackup"
        FichierIni(1) = "[configuration]"
        FichierIni(2) = CleSavePath & "=" & SavePath
        FichierIni(3) = CleTempSave & "=" & TempsSave
        FichierIni(4) = CleNbSaves & "=" & Nbsaves
        FichierIni(5) = ""
        Return FichierIni

    End Function
```

Cette fonction est appelée manuelle car vous voyez que chaque ligne doit être écrite côté programmatique.
Très pratique et très visuel pour le programmeur, mais beaucoup moins agréable lorsque vous avez 100 clés de configuration à entrer.

Une autre méthode consiste a passer un tableau à deux dimensions de string, deux colonnes et autant de lignes que de clés.
La première colonne contenant les clés, la seconde les valeurs.

un rapide algorithme vous construit le même tableau de lignes que vous écrirez dans votre fichier avec WriteAllLines().

Code : VB.NET

```
Function CreeStructureFichierIni(ByVal ClesValeurs(, ) As String) As
String()

 Dim Ligne(10) As String
 'Par exemple ClesValeurs a deux dimensions sous la forme : (1, 10)
        'On divise la taille par 2 car elle correspond à l'ensemble
des cellules et on a deux colonnes donc cellules / 2 = nbligne
 For i As Integer = 0 To te.Length / 2
            Ligne(i) = ClesValeurs(0,i) & "=" & ClesValeurs(1,i)
        Next

    End Function
```

Mais ce genre d'algorithme est à faire par vos soins, il n'est pas très compliqué mais demande un léger travail de recherche.

La récupération de valeurs

Passons tout de suite à la récupération des valeurs.

Ma fonction faite dans ce TP devrait amplement suffir :

Code : VB.NET

```
    Function RecupereCleFichierIni(ByVal Cle As String) As String

            For Each Ligne As String In File.ReadAllLines(FichierIni)
                'Découpe la ligne au niveau de "=" (s'il existe),
                'compare la premiere partie de la ligne (soit la clé)
                If Ligne.Split("=")(0) = Cle Then
                    'Recupère la seconde partie de la ligne (soit la
valeur)
                    Return Ligne.Split("=")(1)
                End If
            Next
            'Sinon ne retourne rien
            Return ""

        End Function
```

On passe la clé souhaitée en argument, on récupère sa valeur.

Eh bien je pense que vous avez les éléments en main pour créer les fichier ini de tous nos prochains TP  .

Pour aller plus loin

Bon, je ne vais pas continuer l'évolution car il me suffit amplement ainsi.

Cette petite idée de TP m'étant venu lors d'un aprem de programmation, j'ai l'habitude de sauvegarder régulièrement mon travail mais après une fausse manip tout mon projet s'est retrouvé converti et irrécupérable, impossible de faire machine arrière.

Ce petit programme effectuant des sauvegardes périodiques du travail m'aurait pu éviter cette erreur.

Bien, passons aux améliorations possibles.

Tout d'abord une sauvegarde plus propre et moins lourde.

Comment faire ?

Eh bien je suppose que vous avez déjà entendus parler de la compression zip. Elle convertit des dossiers et des fichiers en un seul fichier zip. On dit alors que nos fichiers sont compressés sous zip.
La même méthode utilise avec l'archivage et les rar.

Je ne vais pas vous aider plus sur cette voie car elle est réservée à ceux qui souhaitent effectuer un peu de recherche. Je vais juste vous donner quelques voies.

La première étant l'utilisation du namespace **Compression** contenu dans **IO**. Assez difficile à utiliser à mon avis, très fastidieux à mettre en place.

La seconde étant l'utilisation de l'utilitaire 7zip, utilitaire open source donc gratuit.
Voici sa fiche clubic : 7zip

Cet utilitaire dispose d'une interface graphique mais aussi d'une utilisation en ligne de commande.

Les commandes (arguments) possibles avec l'exécutable 7z.exe sont :

```
Usage: 7z <command> [<switches>...] <archive_name> [<file_names>...]
       [<@listfiles...>]

<Commands>
  a: Add files to archive
  b: Benchmark
  d: Delete files from archive
  e: Extract files from archive (without using directory names)
  l: List contents of archive
  t: Test integrity of archive
  u: Update files to archive
  x: eXtract files with full paths
<Switches>
  -ai[r[-¦0]]{@listfile¦!wildcard}: Include archives
  -ax[r[-¦0]]{@listfile¦!wildcard}: eXclude archives
  -bd: Disable percentage indicator
  -i[r[-¦0]]{@listfile¦!wildcard}: Include filenames
  -m{Parameters}: set compression Method
  -o{Directory}: set Output directory
  -p{Password}: set Password
  -r[-¦0]: Recurse subdirectories
  -scs{UTF-8 ¦ WIN ¦ DOS}: set charset for list files
  -sfx[{name}]: Create SFX archive
  -si[{name}]: read data from stdin
  -slt: show technical information for l (List) command
  -so: write data to stdout
  -ssc[-]: set sensitive case mode
  -ssw: compress shared files
  -t{Type}: Set type of archive
  -v{Size}[b¦k¦m¦g]: Create volumes
  -u[-][p#][q#][r#][x#][y#][z#][¦newArchiveName]: Update options
  -w[{path}]: assign Work directory. Empty path means a temporary directory
  -x[r[-¦0]]]{@listfile¦!wildcard}: eXclude filenames
  -y: assume Yes on all queries
```

Cette manipulation est réservée aux plus expérimentés d'entre vous, il va falloir combiner commande et path de fichiers dans une fonction VB nommée : **Shell()** permettant l'éxécution shell de programmes.

Exemple : si votre 7z.exe est dans le dossier de votre programme, il faudra utiliser `Shell("7z.exe a MonArchive.zip MonFichierAZipper").`

Je vous laisse explorer cette voie qui semble prometteuse.

Reste ensuite comme améliorations possible un écran supplémentaire listant les sauvegardes effectuées, la possibilité de restaurer l'une d'entre elle.

Egalement une exécution de ce programme en arrière plan voire en tant que service.

Pour l'arrière plan il faudra déjà s'employer à rendre la forme principal à Visible = false.
Puis créer une icône contextuelle, un contrôle tout fait existe, cherchez dans votre boîte à outils.
Puis récupérer l'évènement clic ou doubleclic sur cette icone pour faire repasser votre forme à visible = true.

Finalement le lancement au démarrage : il faut créer un raccourci de votre programme ou placer votre programme dans le dossier :
C:\Users\@VOTREUSER@\AppData\Roaming\Microsoft\Windows\Start Menu\Programs

Que d'amélioration possible, en y passant un peu de temps votre programme peut devenir une véritable sauvegarde périodique de vos données vitales en restant discret et rapide. Et puis lors de notre partie concernant le réseau, une sauvegarde sur un FTP ou un serveur sera envisageable.

Voilà amis zéros, bonne chance !

Partie 3 : La programmation orientée objet

Les concepts de la POO

Eh bien mes chers amis zéros, nous allons passer à une partie qui va ~~changer votre vie~~ changer votre conception de la programmation.

Vous pensiez avoir déjà vu pas mal de choses en programmation, eh bien c'est loin d'être fini.

Vous vous souvenez que nous utilisons des objets, classes et autres méchantes choses qui ont hanté vos nuits.

Nous allons approfondir encore plus le concept d'objets, et nous allons apprendre a concevoir vos propres objets, ca vous dirait de construire votre propre voiture ?

Bref, je rigole mais attaquons tout de suite.

Pourquoi changer ?

Alors, comme beaucoup je suppose qu'arrivés à cette partie vous vous demandez pourquoi vous changeriez de méthode de programmation.

C'est vrai après tout, celle que nous utilisions fonctionnait très bien jusqu'à maintenant et pourquoi ne pas continuer ?

Eh bien d'un point de vue, la méthodologie de programmation que nous avons vus à présent est très bonne, nous avons même vus quels genre de programmes nous étions capables de réaliser en suivant ce concept.

Mais d'un autre côté ... Il y a plein de choses qui sont impossible ou très difficilement réalisables en programmant de cette façon.

Vous imaginez créer un jeu comme ça ?

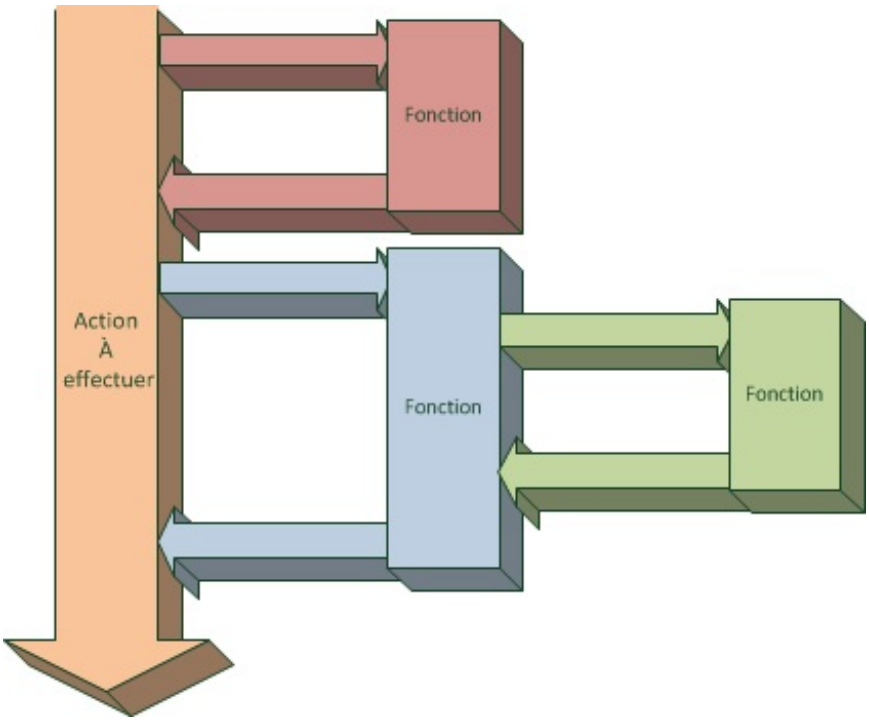
Même le plus basique des jeux de rôles vous prenait des heures de travail pour un résultat que la POO vous apporterait sur un plateau.

Bref, je m'égare, je vais essayer de vous expliquer plus en détail ce qu'est le fabuleux monde de l'Orienté Objet.

Mesdames, Messieurs, Sa Majesté POO.

Je rappelle pour ceux qui ont tendance à sauter des chapitres entiers : POO = Programmation Orientée Objet.

Jusqu'à maintenant nous avons fait de la programmation **Procédurale**, pour faire simple, ce principe se base sur les procédures et fonctions, vous avez remarqués que pendant tous nos TPs, chaque "action" à effectuer était souvent décomposée en un sous-ensemble de fonctions et procédures (sub). C'est donc cela la programmation procédurale.



Comme vous le voyez sur le schéma, ces fonction s'imbriquent les unes aux autres. Pour le moment, aucun problème. Lorsque nous attaquerons de gros projet, cette structure va devenir un véritable pêle-mêle.

C'est pourquoi ~~dieu~~ Alan Kay se décida à révolutionner la façon de programmer en élaborant la Programmation Orientée Objet.

Comme son nom l'indique, nous allons créer des objets. Mais tout d'abord qu'est-ce qu'un objet programmatiquement parlant ?

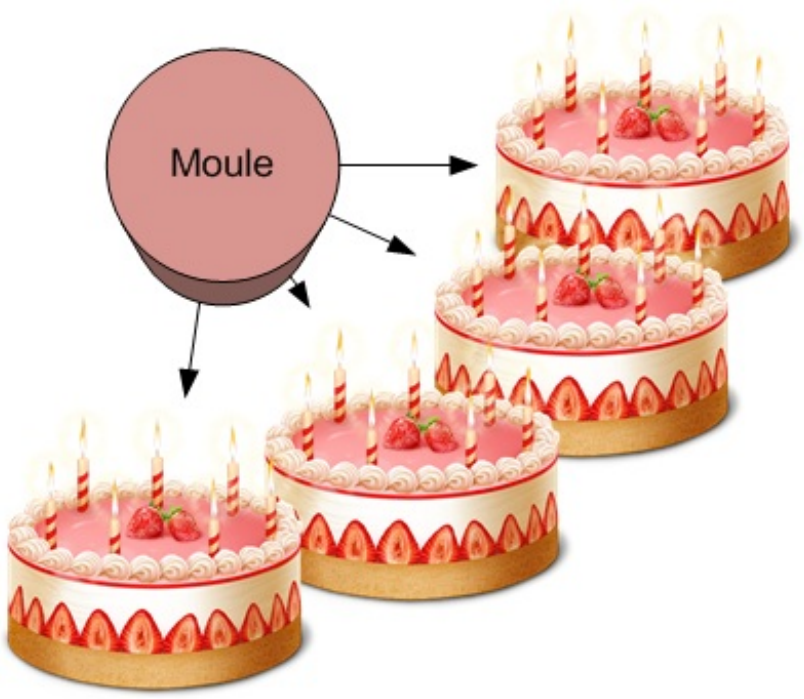
Nous nouveaux amis : les objets

Dans la vie de tous les jours, vous voyez ce qu'est qu'un objet ? Eh bien en programmation le concept reste le même.

Je m'explique :

Nous allons créer des objets qui vont nous permettre de rassembler des groupes de procédures ou fonctions appartenant à la même famille. En gros, si nous voulons contrôler une voiture, nous pouvons, avec nos connaissances actuelles, créer des dizaines de fonctions pour faire avancer, reculer, tourner, freiner notre voiture. Mais si nous en voulons une seconde nous allons être obligés de recommencer.

Avec le concept d'objet, nous programmions des fonctions qui seront liées à l'objet "Voiture" et après peu importe que l'on décide d'en faire 1 ou 100, il n'y aura pas à recommencer.



Ici, sur le principe d'un gâteau. Nous allons coder le "moule" du gâteau, une fois ce moule bien bâtit, il sera capable de nous créer des dizaines de gâteaux.

Vous avez utilisés pas mal d'objet jusqu'à maintenant, exemple : la class File.

Le moule de "File" nous a permis de créer des dizaines de "File" et de les manipuler séparément.

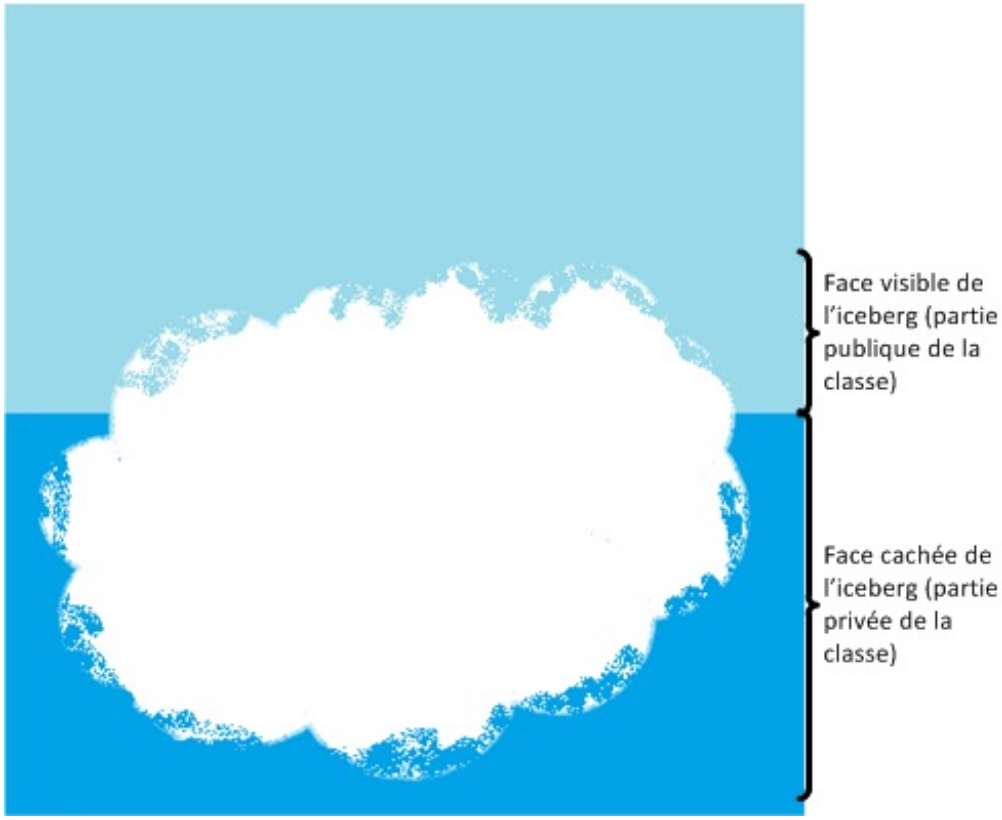
Retournons à notre voiture.

Notre moule, en Visual Basic se nomme la Classe. La classe contient un Contructeur (ce qui se produit lorsque l'on crée notre objet, en l'occurence notre voiture) et possibilité de mettre un destructeur (je pense que vous avez compris son utilité).

Ces méthodes seront présentes dans le fichier de la Class. On peut également ajouter beaucoup d'autres fonctions ou sub à notre classe. Une fonction pour faire avancer notre voiture, une autre pour la faire reculer, nous pouvons également déclarer des variables qui seront utilisables seulement par cette classe.

Les accessibilités

Ce qu'il va bien falloir comprendre et essayer d'appréhender c'est ce qui se passe en "interne" de la classe et ce qui va se passer à l'extérieur.



(ceci est censé être un iceberg ...)

Comme vous pouvez le constater, dans la classe il y a une partie considérée comme privée, qui ne sera accessible que par elle-même, des fonctions ou des variables internes, pour faire avancer la voiture, vous imaginez qu'il va falloir un énorme travail en interne pour que au final on aie juste à appuyer sur une touche pour qu'elle bouge. C'est le travail du privé cela.
Vient ensuite le public, c'est ce qui sera visible par le reste du programme. Sur notre voiture la fonction "avancer" sera publique, on pourra l'appeler de l'extérieur sous ma forme MaVoiture.Avancer (où MaVoiture est la voiture crée avec le moule auparavant).

Mais pourquoi on ne met pas tout en public ?

Eh bien, en faisant ça, c'est le principe fondamental de l'orienté objet auquel vous vous attaquez ... L'encapsulation.

L'encapsulation est le terme employé pour dire "ce que la classe fait en interne c'est du domaine du privé".

Pour la voiture, le moteur, la boite de vitesse, etc ... Tout ce qui fait fonctionner en "interne" cette petite automobile ne doit pas être accessible à l'utilisateur, sinon il n'y a plus aucun intérêt. L'utilisateur doit seulement avoir accès aux pédales et au levier de vitesses.

Les fonctions se passant en interne sont les **attributs**, celles visibles depuis l'extérieur sont les **méthodes**.

Il existe d'autre mots "préfixe" que nous pourrons employer avant nos déclaration. Il y a **Shared**, **Protected**, ... Mais pour le moment intéressons nous uniquement à **Private** et **Public**.

Les fichiers de classe

Donc, théoriquement parlant je comprends tout à fait que c'est très dur à appréhender cette notion d'objet, je ne vais pas plus vous brusquer pour le moment.

Je vais juste vous expliquer comment nous allons créer nos objets.

Personnellement je crée un nouveau fichier par objet, mais si vos objets sont petits, vous pouvez les rassembler en un.

Une déclaration de classe (notre moule) s'effectue avec :

Code : VB.NET

```
Public Class MaClasse

End Class
```

Vous voyez que même sur la déclaration de la classe on peut spécifier publique ou privée. Notre classe doit créer des objets qui seront accessibles depuis tout le programme, laissons en public.

Je vous expliquerai plus tard dans quels cas de figure le Private sera de mise pour déclarer une classe.

Notre classe va contenir des variables et des fonctions qu'elle poura utiliser. Des membres et des attributs en langage technique. Pour les créer ce sera comme ce que nous avons vus jusqu'à présent. Un préfixe d'accessibilité (public, private ...), un **dim** pour les variables, **sub** pour les fonctions ne renvoyant rien et **function** pour les fonctions.

Le constructeur

Je vous rappelle avant que vous n'oubliez ce qu'est que le contructeur.

C'est la méthode qui va être appellée à l'instanciation de l'objet, au moment où nous ferons "`= New MaClasse `" ce sera des arguments que l'on poura spécifier de cette manière : "`= New MaClasse(ArgumentConstructeur1, ArgumentConstructeur2) `".
Comme une fonction qui demande des arguments, le constructeur réagira pareil. Il récupèrera les informations passées en argument et en fera ce qu'il veut, les attribuer à des membres privés par exemple.

Je vous expliquerai dans la partie suivant comment le coder et l'utiliser.

Le destructeur

Je vais vous parler brièvement du destructeur.

Le destructeur est particulier, il est surtout utilisé pour libérer les ressources mémoires allouées à l'objet juste avant sa destruction.
Lorsque nous utiliserons les bases de données par exemple, il faudra se servir du destructeur pour fermer et libérer la connection si elle a été établie pour cette classe.

Bref, sinon il ne sera pas utile : les variables qui sont crées pour la classe sont automatiquement libérées de la mémoire quand cetter dernière "meurt".

L'objet sera détruit si la valeur Nothing lui est affecté ou si il arrive à la fin de sa portée (fin d'une fonction dans laquelle il a été crée) par exemple, comme pour les variables normales.

Vous avez là les prérequis théorique pour attaquer les classes et les objets.

Dans le chapitre suivant nous allons tout de suite commencer à créer notre classe, en tirer nos objets, les manipuler, faire de la

surcharge, de l'héritage, de l'écoute d'évènements, de l'attribution de propriété, de ... 

Calmons nous, la POO est fabuleuse mais on va y aller petit à petit, ne vous inquiétez pas, tout ces mots font peur sur le papier mais une fois les concepts aquis ca ira tout seul.

Courage !

Notre première classe

Notre première classe

Pour commencer notre entrée dans le monde des Classes je vous propose d'attaquer tout de suite sur un petit thème.

Pourquoi pas Mario ?

Eh bien nous allons essayer de faire bouger Mario pour commencer.

Logiquement, puisque nous sommes sur la partie traitant de la POO, notre Mario sera un ... Objet.

La classe

Créons tout de suite notre "moule" de Mario.

Créez donc un nouveau projet, toujours Windows Forms. Ajoutez ensuite un nouvel élément, comme ce que nous avons fait pour une fenêtre mais cette fois choisissez :

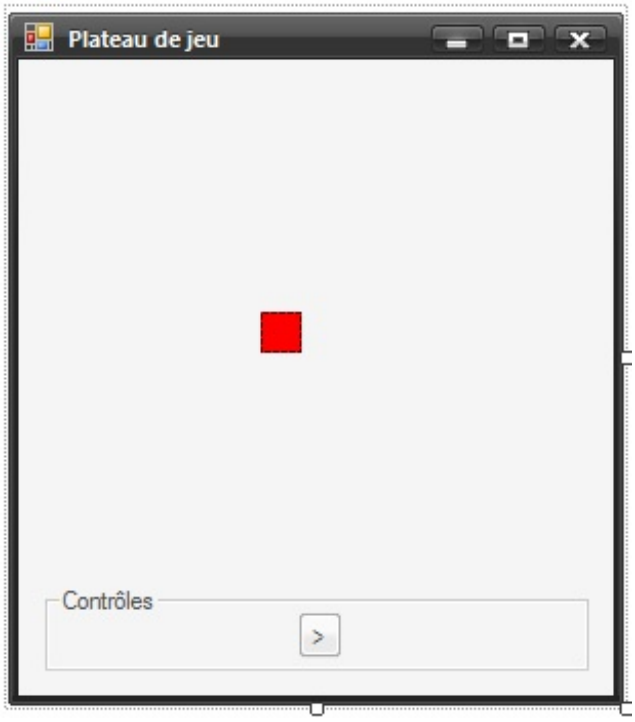
C'est donc un fichier de classe.

Nommez tout de suite ce fichier en Mario par exemple.

Nous voilà donc avec notre classe "Mario" totalement vide, juste la déclaration de début et de fin.

L'interface

Retournons dans notre fenêtre et ajoutons-y quelques contrôles pour pouvoir commencer.



Donc j'ai juste ajouté un petit PANEL dans lequel j'ai spécifié la taille : 20x20 et la couleur de fond : Rouge.

J'ai ajouté en bas un petit bouton qui va nous permettre de déplacer ce panel. Pour le moment uniquement vers la droite.

Vous vous doutez que coder l'évènement qui va permettre au bouton de déplacer ce panel ne va pas être sorcier : quelques location à définir et le tour est joué.

Oui mais nous allons définir une classe qui va pouvoir s'adapter à d'autres situations (eh oui vous pourrez utiliser vos classes dans d'autres programmes.)

Réfléchissons un peu

Un léger moment de réflexion.

Pour le moment je vous ne demande pas de créer une classe vous-même, nous n'avons pas encore vu comment faire, nous allons le faire ensemble.

Je vous demande juste d'essayer de trouver comment rendre cet objet adaptable.

Mario doit être capable de se déplacer par "cases" pour le moment la taille de la case est définie par la taille du panel.

Donc si mon panel devient de 50x50, une case sera égale à 50x50 (les cases sont dans votre imagination mais vous pouvez placer des lignes de manière à faire une grille.

Notre classe ne pourra pas agir directement sur notre panel, c'est toujours le code qui l'a appelé qui va devoir le modifier, notre classe va donc devoir nous renvoyer des coordonnées. Les coordonnées de la nouvelle position de Mario.

Donc si vous avez suivi ma (fastidieuse) réflexion, nous allons devoir travailler avec des variables de type Point, nous aurions pu nous envoyer des int contenant la position en X et en Y mais la variable point rassemble le tout.

Nous allons aussi devoir passer à la classe la taille de notre panel, pour qu'elle puisse travailler dynamiquement, quelle que soit la taille de notre Mario.

Bon, trêves de commentaires, allons-y.

Notre classe

Dans notre fichier classe il va falloir tout d'abord définir un constructeur.

Rendez-vous dans le fichier Mario.vb (la classe) et insérez la déclaration du constructeur (avec New).

On se place dans le "Class Mario" et on inscrit :

Code : VB.NET

```
Sub New()

    End Sub
```

Voilà notre premier constructeur.

Ce constructeur de demande aucun argument. Nous ferions tout de même bien de spécifier à la classe la position originelle de notre Mario et sa taille.

Pour cela commençons par déclarer des variables en **Privé** car elles ne seront accessibles qu'à partir de la classe.

Juste en dessous de Public Class Mario, déclarez :

Code : VB.NET

```
    Private _CoordonneesActuelles As Point
        Private _Taille As Size
```

J'ai donc les coordonnées actuelles de notre Mario de type **Point**, et la taille de type **Size**.

Pourquoi "_" devant tes variables ?

C'est une vieille habitude, lorsque je déclare des membres ou des attributs privés, je les précède d'un "_", vous n'êtes pas obligés de faire comme moi.

Ces variables sont déclarées en **Global** elles seront donc accessibles de partout dans la classe.

Changeons notre constructeur de manière à demander en arguments ces valeurs :

Code : VB.NET

```
    Sub New(ByVal PositionOriginelle As Point, ByVal TailleMario As
    Size)
            _CoordonneesActuelles = New Point(PositionOriginelle)
            _Taille = New Size(TailleMario)
        End Sub
```

Et entrons les dans les variables.

Vu que point et size sont eux même des objets, il faut les instancier avec "New".

Nous voilà donc avec la taille et les coordonnées de notre Mario.

Des méthodes et des attributs

Je rappelle rapidement ce que ces mots signifient :

Les méthodes sont des fonctions de type privées, leur nécessité est interne, rien n'est visible depuis l'extérieur.
Les attributs quand à eux sont publics, visibles depuis l'extérieur, accessibles.

Eh bien codons quelques fonctions qui vont nous permettre de déplacer notre Mario.

A première vue, rien de problématique, on va jouer sur la propriété .X de nos coordonnées pour déplacer Mario en horizontal, .Y pour le vertical.

Mais de combien allons-nous le bouger ?

C'est là que notre notion de "cases" et de taille intervient. On va le faire bouger d'une case.

Créons une nouvelle fonction de type Public.

Cette fonction sera destinée à faire avancer Mario d'une case. Appelons là donc **Avance**.

Code : VB.NET

```
Public Sub Avance()
        _CoordonneesActuelles.X = _CoordonneesActuelles.X + _PasX()
    End Sub
```

C'est quoi _PasX ?

Une petite fonction qui nous renvoie la taille en longueur de Mario (pour savoir quelle est la valeur de la case en longueur).
Eh oui, je ne lésine pas sur les fonctions. Elle est précédée de "_" car c'est un attribut, elle est private :

Code : VB.NET

```
#Region "Fonctions privées"

    Private Function _PasX()
        Return _Taille.Width
    End Function

    Private Function _PasY()
        Return _Taille.Height
    End Function

#End Region
```

Je renvoie simplement la longueur, pour le pasY, c'est la hauteur.

Si vous avez suivi, cette méthode publique va déplacer Mario d'une case sur la droite.

Oui mais ce n'est pas une fonction, on a aucun retour, comment est-ce que notre panel va bien pouvoir se déplacer ?

Il va falloir se servir des propriétés.

Les propriétés

Vous savez déjà ce que c'est que les propriétés, vous en assignez tout le temps quand vous modifiez vos contrôles.

On va apprendre à faire de même pour nos objets.

La syntaxe de déclaration d'une propriété est assez particulière.

Il va falloir gérer lorsqu'on **assigne** cette propriété et lorsqu'on la **récupère**.

Pour commencer voici la syntaxe :

Code : VB.NET

```
Public Property Position()
        Get

        End Get
        Set(ByVal value)

        End Set
    End Property
```

Vous voyez qu'elle fonctionne comme une fonction sauf qu'à l'intérieur on a deux nouveaux mots clés :

- Set : sera appelée lorsque l'on assignera une valeur à cette propriété
- Get : sera appelée lorsque l'on demandera une valeur à cette propriété

Pour le moment l'argument fournit au Set n'a pas de type défini, ce qui est renvoyé non plus, commençons par les définir :

Code : VB.NET

```
Public Property Position() As Point
        Get

        End Get
        Set(ByVal value As Point)

        End Set
    End Property
```

On attend donc un point et on renvoie un point.

Notre propriété va seulement assigner la valeur à la variable _CoordonneesActuelles et renvoyer sa valeur, rien de bien sorcier (vous pouvez même le faire tout seul) mais on aurait pu effectuer beaucoup d'autre choses en même temps. Assigner plusieurs variables, incrémenter un compter, les possibilités sont infinies =).

Voici donc la propriété au final :

Code : VB.NET

```
Public Property Position() As Point
        Get
            Return _CoordonneesActuelles
```

```
            End Get
            Set(ByVal value As Point)
                _CoordonneesActuelles = value
            End Set
        End Property
```

On assigne la valeur, on la retourne.

Voici notre première propriété.

Avec cette nouvelle propriété et la fonction précédente on peut désormais retourner sur notre code relatif à la fenêtre et faire bouger Mario simplement en quelques lignes :

Code : VB.NET

```
Public Class PlateauDeJeu

    'Mario déclaré en global
    Dim MonMario As Mario
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Un nouveau Mario
        MonMario = New Mario(Me.PAN_MARIO.Location,
Me.PAN_MARIO.Size)
    End Sub

    Private Sub BT_AVANCE_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles BT_AVANCE.Click
        'On le fait avancer
        MonMario.Avance()
        'On recupère la nouvelle position
        Me.PAN_MARIO.Location = MonMario.Position
    End Sub

End Class
```

Voici mon code relatif à la fenêtre.

Je déclare un Mario en variable globale, donc il sera accessible et utilisable pendant toute la durée de vie de la fenêtre.
Je l'instancie au load de la fenêtre en passant la position du panel et ses dimensions en paramètres au constructeur.

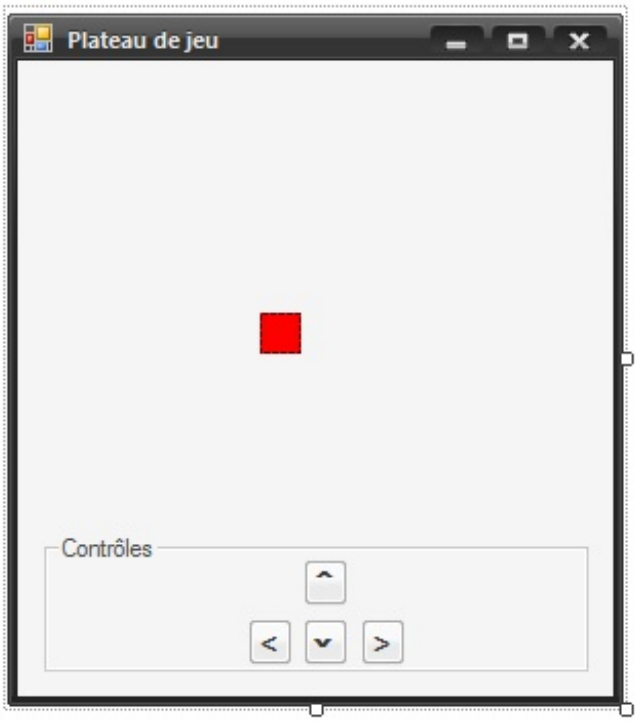
Puis lors du clic sur le bouton, on fait avancer Mario et on récupère sa nouvelle position pour l'affecter au panel.

Vous pouvez essayer.

Notre petit Mario

Une petite partie qui n'a aucun rapport mais c'est pour vous mettre le reste du code de notre déplacement de Mario, on s'en servira sûrement plus tard.

Voici donc a quoi ressemble ma fenêtre finale (on mettra une image de Mario plus tard =)) :



Ma classe :

Code : VB.NET

```
Public Class Mario

    Private _CoordonneesActuelles As Point
    Private _Taille As Size

    Sub New(ByVal PositionOriginelle As Point, ByVal TailleMario As
Size)
        _CoordonneesActuelles = New Point(PositionOriginelle)
        _Taille = New Size(TailleMario)
    End Sub

    Public Sub Avance()
        _CoordonneesActuelles.X = _CoordonneesActuelles.X + _PasX()
    End Sub

    Public Sub Recule()
        _CoordonneesActuelles.X = _CoordonneesActuelles.X - _PasX()
    End Sub

    Public Sub Monte()
        _CoordonneesActuelles.Y = _CoordonneesActuelles.Y - _PasY()
    End Sub

    Public Sub Descend()
        _CoordonneesActuelles.Y = _CoordonneesActuelles.Y + _PasY()
    End Sub

    Public Property Position() As Point
        Get
```

```
            Return _CoordonneesActuelles
        End Get
        Set(ByVal value As Point)
            _CoordonneesActuelles = value
        End Set
    End Property

#Region "Fonctions privées"

    Private Function _PasX()
        Return _Taille.Width
    End Function

    Private Function _PasY()
        Return _Taille.Height
    End Function

#End Region

End Class
```

J'ai rajouté des déplacements.

Et la code final avec la gestion des touches :

Code : VB.NET

```
Public Class PlateauDeJeu

    'Mario déclaré en global
    Dim MonMario As Mario

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Se met en écoute des touches
        Me.KeyPreview = True
        'Un nouveau Mario
        MonMario = New Mario(Me.PAN_MARIO.Location,
Me.PAN_MARIO.Size)
    End Sub

    Sub Form1_KeyDown(ByVal sender As Object, ByVal e As
KeyEventArgs) Handles Me.KeyDown
        Select Case e.KeyCode
            Case Keys.Z
                MonMario.Monte()
            Case Keys.S
                MonMario.Descend()
            Case Keys.Q
                MonMario.Recule()
            Case Keys.D
                MonMario.Avance()
        End Select
        Me.PAN_MARIO.Location = MonMario.Position
    End Sub


#Region "Boutons de l'interface"

    Private Sub BT_AVANCE_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles BT_AVANCE.Click
        'On le fait avancer
        MonMario.Avance()
```

```
            'On recupère la nouvelle position
            Me.PAN_MARIO.Location = MonMario.Position
        End Sub

        Private Sub BT_RECULE_Click(ByVal sender As System.Object, ByVal
    e As System.EventArgs) Handles BT_RECULE.Click
            'On le fait reculer
            MonMario.Recule()
            'On recupère la nouvelle position
            Me.PAN_MARIO.Location = MonMario.Position
        End Sub

        Private Sub BT_DESCEND_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles BT_DESCEND.Click
            'On le fait descendre
            MonMario.Descend()
            'On recupère la nouvelle position
            Me.PAN_MARIO.Location = MonMario.Position
        End Sub

        Private Sub BT_MONTE_Click(ByVal sender As System.Object, ByVal
    e As System.EventArgs) Handles BT_MONTE.Click
            'On le fait monter
            MonMario.Monte()
            'On recupère la nouvelle position
            Me.PAN_MARIO.Location = MonMario.Position
        End Sub

    #End Region

    End Class
```

Voilà. Amusez-vous bien.

Concepts avancés

La POO est un monde fabuleux, pour le moment elle doit vous sembler un peu trouble mais avec un peu de pratique, vous n'allez plus pouvoir vous en passer.

Mais bon, en attendant que tout ça devienne limpide, je vais vous apporter quelques notions nouvelles affin de vous permettre d'employer au mieux ces nouvelles connaissances.

Dans ce chapitre nous allons aborder quelques notions plus poussées et non pour le moins utiles.

Au menu : l'héritage, le polymorphisme, les collections, la surcharge d'opérateurs, les propriétés par défaut,les bibliothèques de classes et comment les utiliser, et les classes abstraites, soyons fous !

Bon appétit

L'héritage

Premier concept important : l'héritage.

Même très important je dois dire. J'ai déjà tenté de vous exposer cette notion dans la partie sur la fenêtre mais ce n'était pas très judicieux, vous ne connaissiez pas la POO.

On va donc tout recommencer ici.

Bon, j'espère que les principes de création de classe et de programmation orientée objet sont acquis. Je sais que c'est une rude partie mais allez chercher un café et reprenez tout ça calmement.

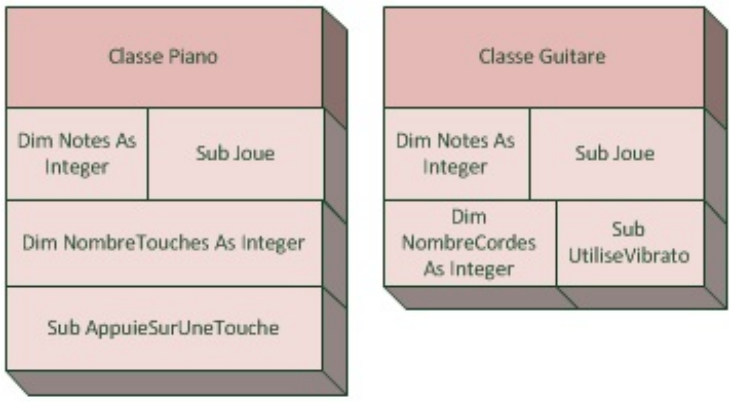
Vous connaissez tous les mots Français "Héritage" et "Hériter", on peut le résumer par :

Citation : Wikitionnaire

> Devenir propriétaire d'une chose par droit de succession.

Eh bien ce concept est quasiment le même en programmation à une petite nuance près. Dans la vraie vie un héritage transmet simplement une chose, en programmation l'héritage d'une classe va dupliquer cette dernière et donner ses caractéristiques à la classe qui hérite.

Un petit schéma pour être plus clair



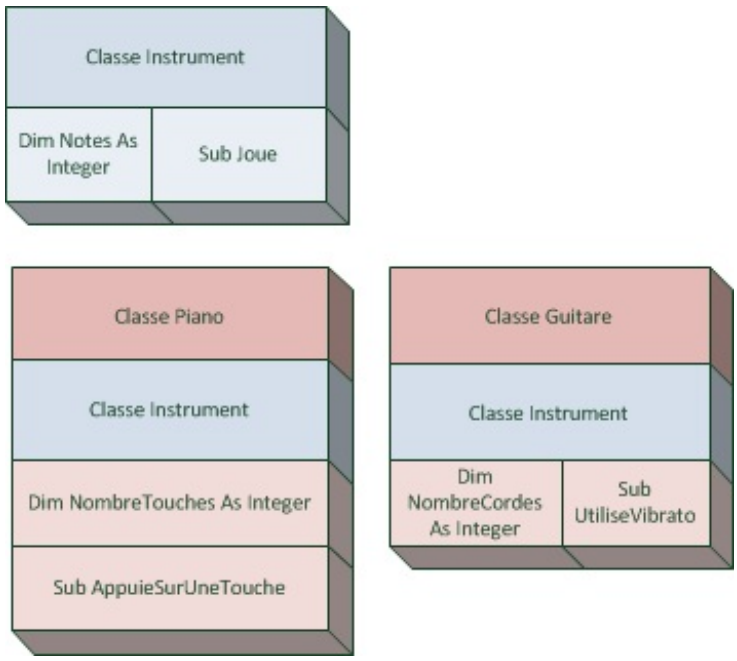
Admettons que je veuille créer 2 classes comme ceci. Une classe guitare et une classe piano. Vous remarquez qu'elles présentent des similitudes, elles possèdent toutes les deux un attribut "Notes" qui va contenir le panel de notes que cet instrument peut jouer et un **Sub** "Joue" qui va lui permettre de produire un son. (Cas purement hypothétique, je ne vais pas vous demande de créer un orchestre en Visual Basic).

En plus de ces deux éléments communs elles présentent des particularités spécifiques à leur type : la guitare aura en plus le nombre de cordes qu'elle possède et une fonction permettant d'utiliser le Vibrato. Le piano quand à lui, contiendra une variable comptant le nombre de touches qu'il possède et un **Sub** pour appuyer sur une touche spécifique.

Vous remarquez tout de suite que ces "similitudes" vont devoir être écrites en double ... Beaucoup de travail et de lignes pour rien.

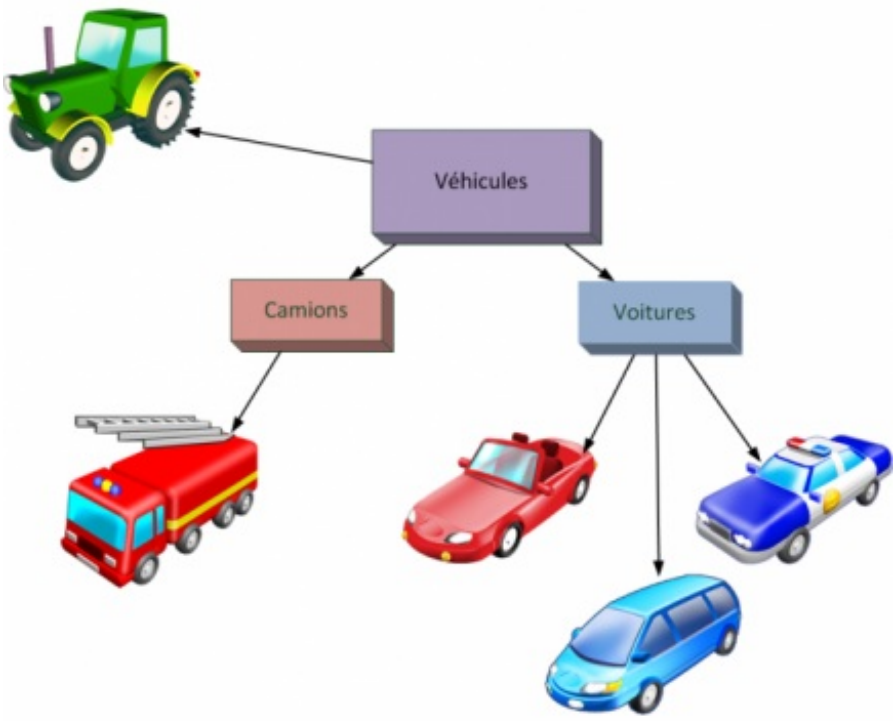
C'est pour cela que ~~dieu~~ les programmeurs ont introduit le concept de l'héritage.

Regardez, si nous créons une troisième classe nommée "instrument" qui contiendra des membres communs à tous les instruments, comme l'attribut "Notes". Il serait si simple d'inclure cette classe dans les autres, de façon à bénéficier de ses caractéristiques.



C'est justement là toute la puissance de l'héritage. En une ligne de programmation nous allons pouvoir faire hériter nos classes Guitare et Piano de Instrument de façon à leur donner la possibilité d'utiliser ses membres.

Un héritage peut se faire sur plusieurs niveaux, il n'y a pas de limite. La classe instrument pouvait elle même hériter d'une classe de type "Chose" et ainsi de suite ...



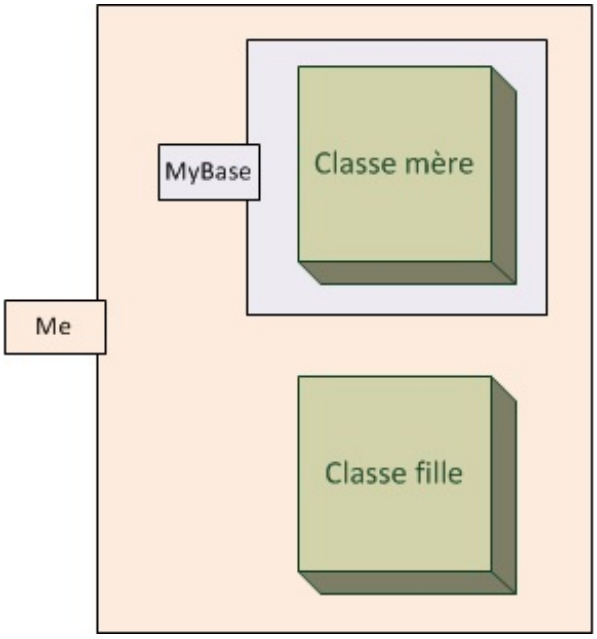
Ici vous voyez l'héritage sur plusieurs niveaux : la voiture de police hérite de la classe "Voitures" qui hérite elle même de la classe "Véhicules".

Alors qu'est-ce que cette modification implique concrètement pour nos classes ?

Eh bien d'un point de vue extérieur à la classe, une fois instanciée par exemple, eh bien c'est transparent. C'est à dire qu'on ne saura pas si le membre de la classe auquel on va accéder appartient à la classe mère ou fille.

Par contre d'un point de vue interne à la classe, ça se complique. Nous allons devoir apprendre à jongler entre les membres appartenant a la classe fille où à la classe mère au travers de préfixes (du même type que Me).

Ce mot est **MyBase**.



Dans ce schéma, on se place du point de vue de la classe fille. Si depuis cette classe fille on débute une ligne par **Me.**, les membres auxquels nous pourrons accéder seront ceux de de la classe fille et de toutes les classes dont elle hérite. En revanche, en utilisant **MyBase.** nous accèderons uniquement aux membres de la classe mère.

Cette informations va nous être très précieuse, surtout lorsque nous allons faire appel à des classes héritées qui ont besoin d'être instanciées.

Vous avez deux choix possible dans notre exemple : créer un constructeur dans "instrument" ou non. Si vous décidez de ne pas en mettre cette classe va être considérée comme abstraite (le chapitre d'après).
En revanche si vous décidez de mettre un constructeur, il va falloir instancier la classe mère au même moment que la classe fille.

Bon, arrêtons la théorie attaquons tout de suite la pratique pour que vous puissiez voir concrètement à quoi ça ressemble.

Code : VB.NET

```
Public Class Instrument

    Private _Notes() As Integer

    Sub New(ByVal Notes() as Integer)
        _Notes = Notes
    End Sub

End Class

Public Class Guitare
    Inherits Instrument 'Hérite d'instrument

    Private _NbCordes As Integer

    Sub New(ByVal Notes() As Integer, ByVal NbCordes As Integer)
```

```
            MyBase.New(Notes) 'On instancie la mère
            _NbCordes  = NbCordes
        End Sub

    End Class
```

Dans ce petit bout de code j'ai crée deux classes : "Instrument" et "Guitare".

La classe Instrument est la classe mère, elle a un attribut **_Notes** et un constructeur.
La classe Guitare est la classe fille, elle a également un attribut **_NbCordes** et un constructeur.

La ligne `Inherits Instrument` indique que la classe hérite d'"Instrument". Et lors de l'instanciation de la classe fille, le constructeur de la classe mère est lui aussi appelé via `MyBase.New(Notes)` .

Les classes abstraites

Eh bien une classe abstraite est une classe ne pouvant pas être instanciée, autrement dit on ne peut pas créer d'objet à partir de ce moule.

Alors à quoi va-t-elle nous servir ?

Justement, nous allons lui trouver une utilité, et non des moindres.
Vous vous souvenez du principe de l'héritage ? (On vient de l'aborder ).
Elle va permettre de créer des classes dérivées, en clair, cette classe va seulement servir de base (une classe mère) pour des classes qui vont lui dériver.

Comme notre exemple sur les instruments, au début de cette partie. Une guitare, un piano, bref des instruments concrets, nous allons les instancier et les utiliser. Cependant la classe "instrument", la classe mère de tous les autres, nous aurions pu la définir en classe abstraite.

C'est vrai après tout, vous vous voyez créer un objet "instrument"  .

Donc notre précédent code va devenir :

Code : VB.NET

```
Public MustInherit Class Instrument

    Private _Notes() As Integer

    Public Property Notes As Integer()
        Set(ByVal value() As Integer)
            _Notes = value
        End Set
        Get
            Return _Notes
        End Get
    End Property

End Class

Public Class Guitare
    Inherits Instrument 'Hérite d'instrument

    Private _NbCordes As Integer

    Sub New(ByVal Notes() As Integer, ByVal NbCordes As Integer)
        MyBase.Notes = Notes 'On spécifie la propriété Notes de la
mère
        _NbCordes = NbCordes
    End Sub

End Class
```

Vous voyez que ici le mot clé **MustInherit** spécifié dans la déclaration de la classe (qui signifie en français "doit hériter" ou plutôt à traduire ici par "doit être hérité"), spécifie que cette classe sera abstraite, elle ne pourra pas être utilisée telle qu'elle.

Ma classe Guitare est donc ici pour hériter d'Instrument. Une fois la classe Guitare instanciée, on peut très bien accéder aux membres d'Instrument. `MaGuitare.Notes(0)` est tout à fait correct (où MaGuitare est mon objet crée).

Il existe toutefois une seconde utilisation possible pour les classes abstraites. En tant que bibliothèque de fonctions.

Je m'explique, plutôt que de placer toutes vos fonctions dans le fichier qui contient la fenêtre, il vous est possible de créer un fichier de classe, nommé "Fonctions" par exemple, et y inscrire des fonctions qui pourront être utilisées à partir de n'importe où dans votre programme.

Code : VB.NET

```
Public Class Fonctions

    Shared Function Somme(ByVal X As Integer, ByVal y As Integer) As
Integer
        Return X + y
    End Function

    Shared Function Difference(ByVal X As Integer, ByVal y As
Integer) As Integer
        Return X - y
    End Function

End Class
```

Ma classe Fonctions, j'ai retiré le mot MustInherit.

Vous vous apercevez que mes fonctions ne sont pas publiques ou privées mais **Shared**.

Le mot clé **Shared** signifiant **Partagé** en Français, indique que cette fonction peut être utilisée sans avoir besoin d'instancier la classe. Autrement dit, dans le programme on pourra utiliser `Fonctions.Addition(1,2)` où l'on veut.

Les avantages : la possibilité de regrouper des fonctions utiles dans le même fichier.
Les inconvénients : puisque la classe n'est pas instanciée, il n'est pas possible d'accéder à des membres externes à la fonction. Des variables globales ne pourront pas être utilisées par exemple.

Les évènements

Vous vous souvenez des évènements dans nos contrôles ?

Eh bien ici c'est le même principe.

Un évènement est une fonction appelée lorsque quelque-chose se produit. Ce quelque chos-, dans un contrôle commun est prédéfini le clic, le changement de texte ... Et on "écoute" cet évènement avec handles. Ici c'est pareil, on va écouter un évènement sur cet objet avec Handles.

Pour commencer, il faut spécifier ce qui va déclencher l'évènement à l'intérieur de notre classe.

Dans ma classe je déclare un timer en global avec WithEvents, ce qui signifie que je vais pouvoir écouter les évènements de cet objet. Vous avez déjà utilisé un timer en tant que contrôle (les contrôles étant d'office avec WithEvent) cette fois il ne sera visible que côté code.

Et je l'instancie puis le lance dans le constructeur avec une seconde en interval, j'utilise la fonction **Start()** pour le démarrer plutôt que **Enable = true**.

Dans l'évènement Tick du timer, j'incrémente un compteur, une fois arrivé à 10 je déclenche l'évènement.

Côté code ca nous donne ça dans notre classe :

Code : VB.NET

```
Private WithEvents _Tim As Timer
    Private _Compteur As Integer

    Sub New()
        _Tim = New Timer
        _Tim.Interval = 1000
        _Tim.Start()
        _Compteur = 0
    End Sub

    Public Event DixSecondes()

    Sub _Tim_Tick() Handles _Tim.Tick
        _Compteur += 1
        If _Compteur = 10 Then
            RaiseEvent DixSecondes()
        End If
    End Sub
```

Vous voyez pour commencer la déclaration des variables (compteur et timer). Puis le constructeur initialise, instancie et démarre tout ça.

Vient l'événement **Tick** du timer, je compte et je déclenche l'événement avec le mot clé **RaiseEvent**.

L'évènement déclenché doit être déclaré : **Public Event DixSecondes()**, en public pour pouvoir "l'écouter" de l'extérieur.

Allons du côté de notre fenêtre qui va instancier notre objet.

La classé déclarée en globale doit être faite avec le mot clé WithEvents également.

Code : VB.NET

```
Dim WithEvents MaClasse1 As MaClasse

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
```

```
System.EventArgs) Handles MyBase.Load
        MaClasse1 = New MaClasse1()
    End Sub

    Sub AttendsLesDixSecondes() Handles MaClasse1.DixSecondes
        MsgBox("Dix secondes que l'objet est crée")
    End Sub
```

Puis je l'instancie dans le form.

Pour finir j'ai écouté l'évènement avec `Sub AttendsLesDixSecondes() Handles MaClasse1.DixSecondes`. Lorsque les 10 secondes sont écoulées, je déclenche une msgBox.

Avec des arguments

Et on peut passer des arguments avec nos évènements :

Côté classe on déclare l'évènement avec

Code : VB.NET

```
Public Event DixSecondes(ByVal Message As String)
```

On voit qu'il attend un argument de type string.

DDonc lorsqu'on va l'appeler :

Code : VB.NET

```
RaiseEvent DixSecondes("Dix secondes que l'objet est crée")
```

Je lui passe mon argument.

Et finalement côté fenêtre, le handles de mon évènement s'effectue ainsi :

Code : VB.NET

```
Sub AttendsLesDixSecondes(ByVal Message As String) Handles
MaClasse1.DixSecondes
        MsgBox(Message)
    End Sub
```

Et voilà, ca va vous être utile je pense, cette petite notion d'évènements.

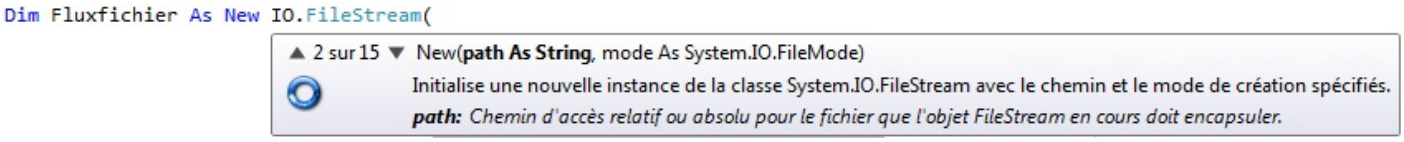
La surcharge

Passons à la surcharge.

Encore une particularité de la POO qui va vous être fort utile.

Même si vous ne savez pas ce que ce mot signifie, je peux vous dire que vous y avez déjà été confrontés.

Souvenez-vous, lorsque vous passez des arguments à une fonction et que l'assistant de Visual Basic vous propose plusieurs possibilités de passer ces arguments :



Sur ce screen, vous voyez que l'infobulle spécifie "**2 sur 15**", j'ai choisi la seconde possibilité de donner les arguments sur 15 possibilité différentes. C'est cela la surcharge, pour la même fonction, avoir plusieurs "résultats" possible en fonction du passage des arguments.

Créons tout de suite un constructeur surchargé pour vous monter ce que cela implique :

Code : VB.NET

```
Public MustInherit Class Instrument

    Private _Notes() As Integer

    Public Property Notes As Integer()
        Set(ByVal value() As Integer)
            _Notes = value
        End Set
        Get
            Return _Notes
        End Get
    End Property

End Class

Public Class Guitare
    Inherits Instrument 'Hérite d'instrument

    Private _NbCordes As Integer

    Sub New()
        _NbCordes = 0
    End Sub

    Sub New(ByVal Notes() As Integer)
        MyBase.Notes = Notes
        _NbCordes = 0
    End Sub

    Sub New(ByVal Notes() As Integer, ByVal NbCordes As Integer)
        MyBase.Notes = Notes
        _NbCordes = NbCordes
    End Sub

End Class
```

Ici j'ai donc déclaré dans ma classe Guitare, 3 constructeurs différents, j'ai donc surchargé le constructeur.

Ce qui me donne le droit à cette infobulle lorsque je veux l'instancier :

Deux fonctions avec des arguments de type différent apportent aussi une surcharge :

Code : VB.NET

```
    Sub New(ByVal Notes() As Integer, ByVal NbCordes As Integer)
        MyBase.Notes = Notes
        _NbCordes = NbCordes
    End Sub

    Sub New(ByVal Notes() As Integer, ByVal NbCordes As String)
        MyBase.Notes = Notes
        _NbCordes = NbCordes
    End Sub
```

Ici le nombre de cordes est une fois un string, une autre fois un integer, la fonction appelée va dépendre du type passé lors de l'instanciation.

Surcharger la classe mère

Pour surcharger une méthode de la classe mère, la technique est presque la même. Il va juste falloir rajouter le mot clé `Overloads` devant la méthode de la classe fille. Ce qui nous donne :

Code : VB.NET

```
Public MustInherit Class Instrument

    Private _Notes() As Integer

    Public Property Notes As Integer()
        Set(ByVal value() As Integer)
            _Notes = value
        End Set
        Get
            Return _Notes
        End Get
    End Property

    Sub Joue()
        'Ding
    End Sub

End Class

Public Class Guitare
    Inherits Instrument 'Hérite d'instrument

    Private _NbCordes As Integer

    Sub New(ByVal Notes() As Integer, ByVal NbCordes As Integer)
        MyBase.Notes = Notes
        _NbCordes = NbCordes
    End Sub

    Overloads Sub Joue(ByVal Note As Integer)
```

```
            'Ding
        End Sub

    End Class
```

J'ai bien rajouté `Overloads` devant la méthode **Joue** de la classe fille, et lors de son appel j'ai le choix entre les deux possibilités : avec ou sans argument.

Dernière chose : on peut "Bypasser" une méthode mère. Autrement dit, créer une méthode de la même déclaration dans l'enfant et spécifier que c'est cette dernière qui a la priorité sur l'autre. Cela grâce à `Overrides` .

Code : VB.NET

```
    Public MustInherit Class Instrument

        Private _Notes() As Integer

        Public Property Notes As Integer()
            Set(ByVal value() As Integer)
                _Notes = value
            End Set
            Get
                Return _Notes
            End Get
        End Property

        Overridable Sub Joue()
            'Ding
        End Sub

    End Class

    Public Class Guitare
        Inherits Instrument 'Hérite d'instrument

        Private _NbCordes As Integer

        Sub New(ByVal Notes() As Integer, ByVal NbCordes As Integer)
            MyBase.Notes = Notes
            _NbCordes = NbCordes
        End Sub

        Overrides Sub Joue()
            MyBase.Joue() 'Ding de la mère
            'Ding
        End Sub

    End Class
```

La méthode Joue de la fille prime sur la mère. Le mot clé `Overrides` dans la déclaration de la méthode fille est nécessaire, tout comme le mot clé `Overridable` dans la déclaration de la méthode mère.

Et j'ai pu utiliser `MyBase.Joue()` pour que la méthode mère soit quand même appelée.

La surcharge d'opérateurs et les propriétés par défaut

Dernière section de ce premier chapitre sur les notions avancées : la surcharge d'opérateurs et les propriétés par défaut.

Paramètres aux propriétés

Avant de comprendre les propriétés par défaut il faut juste que je vous montre comment utiliser un paramètre dans un propriété, ça me semble logique mais quelques lignes d'éclaircissement ne feront pas de mal. Admettons que je veuille accéder à un certain index dans un tableau à partir d'une propriété, je vais devoir passer un argument à cette dernière :

Code : VB.NET

```
Module Module1

    Sub Main()
        Dim MaClasse As New Classe
        Console.WriteLine(MaClasse.Variable(0))
        Console.Read()
    End Sub

End Module

Public Class Classe

    Dim _Variable() As String

    Sub New()
        _Variable = {"a", "b", "c", "d"}
    End Sub

    Property Variable(ByVal Index As Integer) As String
        Get
            Return _Variable(Index)
        End Get
        Set(ByVal value As String)
            _Variable(Index) = value
        End Set
    End Property

End Class
```

Ici (je ne devrais même plus avoir à vous expliquer le code je pense ), je demande lors de l'appel de la propriété, un paramètre spécifiant l'index, même principe qu'une fonction demandant des arguments : `Property Variable(ByVal Index As Integer) As String` .

Les propriétés par défaut

Les propriétés par défaut, vont vous permettre de vous soustraire à quelques lignes dans votre code source.
Ce concept a pour but d'attribuer à une certaine propriété la particularité d'être par "défaut".

Lorsque vous voudrez utiliser cette propriété vous n'autrez plus besoin d'écrire `MaClasse.Variable(0)` mais seulement `MaClasse(0)` .

A utiliser avec précaution si vous ne voulez pas vite être embrouillé  .

Un simple mot suffit dans le code que je viens de faire, pour la spécifier en défaut :

Code : VB.NET

```
Module Module1

    Sub Main()
        Dim MaClasse As New Classe
        Console.WriteLine(MaClasse(0))
        Console.Read()
    End Sub

End Module

Public Class Classe

    Dim _Variable() As String

    Sub New()
        _Variable = {"a", "b", "c", "d"}
    End Sub

    Default Property Variable(ByVal Index As Integer) As String
        Get
            Return _Variable(Index)
        End Get
        Set(ByVal value As String)
            _Variable(Index) = value
        End Set
    End Property

End Class
```

Le mot clé **Default** spécifie quelle propriété doit être considérée comme celle par défaut.

Deux précautions à prendre : les propriétés par défaut doivent au moins attendre un argument. Et il ne peut y avoir qu'une seule propriété par défaut par classe (logique).

Surcharge d'opérateurs

Comme son nom l'indique, cette surcharge va être spécifique aux opérateurs : +, -, /, *, &, =, <, >, Not, And, et j'en passe ...

Vous savez déjà qu'ils n'ont pas la même action en fonction des types que vous utilisez.

Entre deux integers : 10 + 10 = 20
Entre deux strings : "Sal" + "ut" = "Salut"
Entre deux date : CDate("20/10/2010") + CDate("20/10/2010") = 20/10/201020/10/2010

Bref, rien à voir.

Apprenons a surcharger un opérateur pour notre classe pour la faire réagir avec ce dernier.

La ligne de déclaration d'une surcharge d'opérateur est un peu plus spécifique :

Code : VB.NET

```
Shared Operator +(ByVal Valeur1 As Classe, ByVal Valeur2 As Classe)
As Classe
```

Tout d'abord, une surcharge d'opérateur doit être en **Shared**. Ensuite le mot Operator est suivi de l'opérateur que l'on souhaite surcharger. Ici c'est le "+". Suivi de deux paramètres (un de chaque côté du "+"). Et le type qu'il retourne.

Exemple dans un petit programme :

Code : VB.NET

```
Module Module1

    Sub Main()

        Dim MaClasseBonjour As New Classe("Bonjour")
        Dim MaClasseSDZ As New Classe(" SDZ")
        Dim MaClasseBonjourSDZ As Classe = MaClasseBonjour +
MaClasseSDZ

        Console.WriteLine(MaClasseBonjourSDZ.Variable)
        Console.Read()
    End Sub

End Module

Public Class Classe

    Dim _Variable As String

    Sub New(ByVal Variable As String)
        _Variable = Variable
    End Sub

    Property Variable As String
        Get
            Return _Variable
        End Get
        Set(ByVal value As String)
            _Variable = value
        End Set
    End Property

    Shared Operator +(ByVal Valeur1 As Classe, ByVal Valeur2 As
Classe) As Classe
        Return New Classe(Valeur1.Variable + Valeur2.Variable)
    End Operator

End Class
```

J'ai donc surchargé l'opérateur "+" qui me permet d'additionner les valeurs de l'attribut **Variable**. Vous pouvez bien évidemment inventer d'autres choses à faire qu'une simple addition.

Puisque notre opérateur est Shared, on ne peut pas accéder aux attributs internes à la classe pendant son utilisation, il faut donc agir uniquement sur les paramètres qu'il récupère.

Les collections

Eh bien attaquons les collections.

Tout d'abord, et comme d'habitude, qu'est-ce qu'une collection ? A quoi ça va nous servir ?

Eh bien je vais d'abord vous exploser un problème. Vous avez un tableau que vous initialisez à 10 cases. Une case pour un membre par exemple. Si un membre veut être ajouté après la déclaration du tableau, vous allez devoir redéclarer un tableau avec une case de plus (on ne peut normalement pas redimensionner un tableau).

Une collection est sur le même principe qu'un tableau mais les éléments peuvent être ajoutés ou supprimés à souhait. Pour les zéros connaissant les listes chaînées, c'est le même concept.

Vous vous souvenez que nous déclarions un tableau en ajoutant accolé au nom de la variable deux parenthèses contenant le nombre d'éléments dans le tableau. Eh bien ici, ce n'est pas plus compliqué, mais ce n'est pas vraiment un tableau que l'on crée, c'est un objet de type collection.

La syntaxe d'instanciation sera donc :

Code : VB.NET

```
Dim MaListeDeClasses As New List(Of Classe)
```

Où **Classe** est une classe que j'ai crée pour les tests.
Le mot clé est **List(Of** TypeSouhaité).

Du même principe qu'un tableau qu'on remplissait à l'instanciation avec "= {1, 2, 3}", la liste peut se remplit manuellement ainsi :

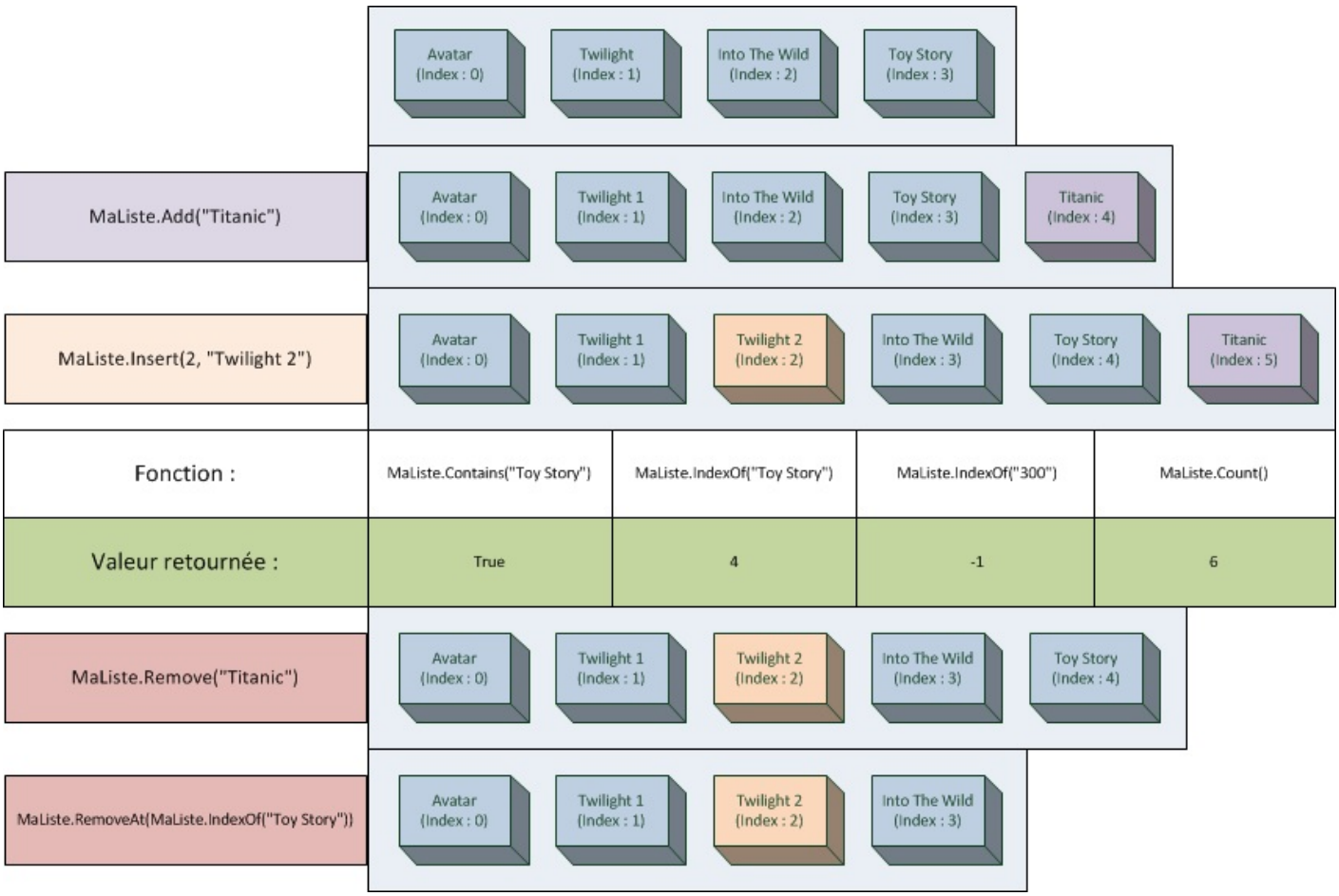
Code : VB.NET

```
Dim MaListeDeClasses As New List(Of Classe) From {New Classe("1"),
New Classe("2")}
```

Avec le mot clé **From**.

Cette collection va être vraiment utile, par de simples fonctions on va pouvoir ajouter un élément au bout où à un index spécifié, en retirer un, trouver un élément.

Exemple ici par un schéma :



- J'initialise une liste de String. Cette liste va contenir des noms de films. Elle contient au début 4 films.
- J'utilise la fonction **Add** sur cette liste, elle a pour effet d'ajouter au bout de la liste "Titanic".
- J'utilise la fonction **Insert**, le premier argument est l'index où ajouter l'objet que l'on passe en second argument. Ici je le place en index 2. Sachant que L'index 0 l'aurait ajouté au début de la liste.
- Puis j'utilise quelques fonctions que je vais vous détailler :
  - La fonction **Contains** effectue une recherche dans la liste pour trouver l'élément passé en argument. Si il est présent, elle renvoie True, sinon False.
  - **IndexOf** se présente de la même manière que **Contains**. Si elle ne trouve pas l'élément elle renvoie -1 sinon elle retourne l'index de l'élément. La fonction **LastIndexOf** existe aussi. Si des éléments sont présents en double, **IndexOf** retourne le premier, **LastIndexOf** le dernier.
  - **Count** quand à elle renvoie le nombre d'éléments dans la liste. A la même manière que **Lenght** sur un tableau.

    Le dernier index est donc **MaListe.Count - 1**

- Puis j'utilise la fonction **Remove** pour supprimer l'élément Titanic.
- Et la fonction **RemoveAt** sert aussi à supprimer un élément mais cette fois c'est l'index qui est passé en paramètre. J'aurais pu entrer l'index de Toy Story en dur (4) ou alors combiner la fonction **IndexOf** et **RemoveAt** comme fait ici.

Il existe beaucoup d'autres fonctions possibles sur les collections, je ne peux pas toutes les lister mais vous allez vite les découvrir grâce au listing qu'effectue Visual Studio lorsque vous écrivez quelque-chose.

Regardons un peu côté programmations :

Code : VB.NET

```
Module Module1
```

```
        Sub Main()

            Dim MaListeDeClasses As New List(Of Classe)
            MaListeDeClasses.Add(New Classe("Avatar"))
            MaListeDeClasses.Add(New Classe("Twilight 1"))
            MaListeDeClasses.Insert(0, New Classe("Titanic"))

            For Each MaClasse As Classe In MaListeDeClasses
                Console.WriteLine(MaClasse.Affiche)
            Next
            Console.Read()

        End Sub

    End Module

    Public Class Classe

        Private _Variable As String

        Sub New(ByVal Variable As String)
            _Variable = Variable
        End Sub

        Function Affiche() As String
            Return _Variable
        End Function

    End Class
```

J'insère des éléments dans ma liste, et grâce à **For Each**, je parcours ces éléments.

J'espère que vous allez préférer ça aux tableaux =)

Bien évidemment vos listes peuvent être de tous les types, même des objets (comme ici dans l'exemple). Les avantages des listes sont multiples et très modulaires.

Les bibliothèques de classes

Lorsque vous créerez de gros objet, avec des dizaines de fonctions complexes à l'intérieur et qui peuvent être utilisés dans de multiples autres situations, vous voudrez sûrement sauvegarder ces derniers.

Vous avez donc deux possibilités. La première étant de simplement copier le fichier .vb contenant la classe et le coller dans votre nouveau projet.
La seconde étant de créer une bibliothèque de classes.

La bibliothèque de classes étant un nouveau projet qui aura pour but de créer une DLL, un fichier simple et compilé dans lequel toutes les classes que vous aurez développés dedans seront compilées et seront facilement réutilisables.

Un hic : vous ne pouvez plus modifier les classes une fois le fichier DLL compilé.

Les créer

Donc pour commencer votre bibliothèque, créer un nouveau projet -> Bibliothèque de clases.

Je vais l'appeler **MesClasses**

A l'intérieur de crée une classe :

Code : VB.NET

```
Public Class MaClasse

        Private _Variable As String

        Sub New(ByVal Variable As String)
            _Variable = Variable
        End Sub

        Function Affiche() As String
            Return _Variable
        End Function

    End Class

End Namespace
```

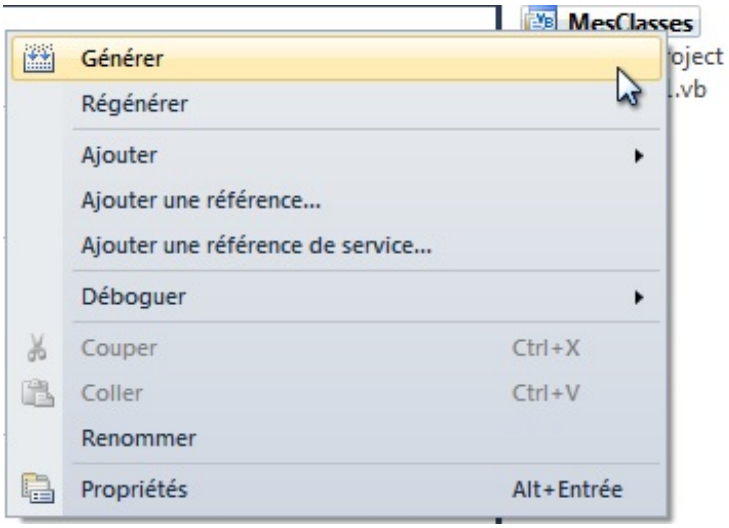
Remarquez le "Namespace", il permet de placer notre classe dans un namespace, ici celui de "MesClasses". On devra donc importer ce dernier à la même manière que System.IO par exemple.

Bien sûr si vous avez plusieurs classes a créer, soit vous les insérez dans le même fichier, soit vous créez un fichier par classe.

Ce type de projet n'est pas exécutable, il ne contient ni de Main, si de Load, il doit être obligatoirement utilisé par un autre projet.

Une fois votre classe crée et vérifiée, il va falloir générer le projet.

Pour ce faire, un clic droit sur le projet dans l'explorateur de solutions et "Générer".

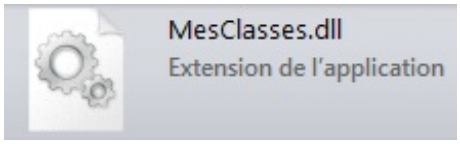


La DLL est maintenant compilée.

Pour la retrouver, direction **VosDocuments\Visual Studio 2010\Projects\MesClasses\MesClasses\bin\Debug**.

Où MesClasses est le nom de votre projet. Si vous avez modifié la configuration de la génération, il est possible que la DLL se situe dans **Release** plutot que **Debug**.

Une fois dans ce répertoire donc, le fichier DLL s'est compilé et vous le retrouvez, gardez le bien au chaud, dans un répertoire contenant toutes vos bibliothèques, pourquoi pas.

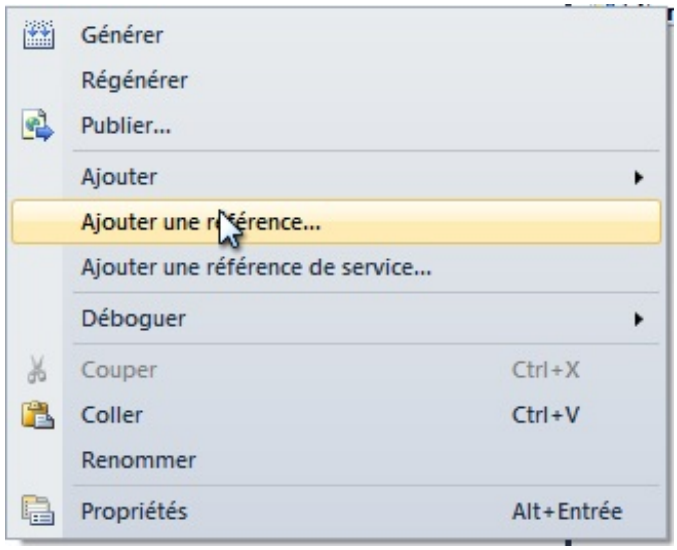


Les réutiliser dans un projet

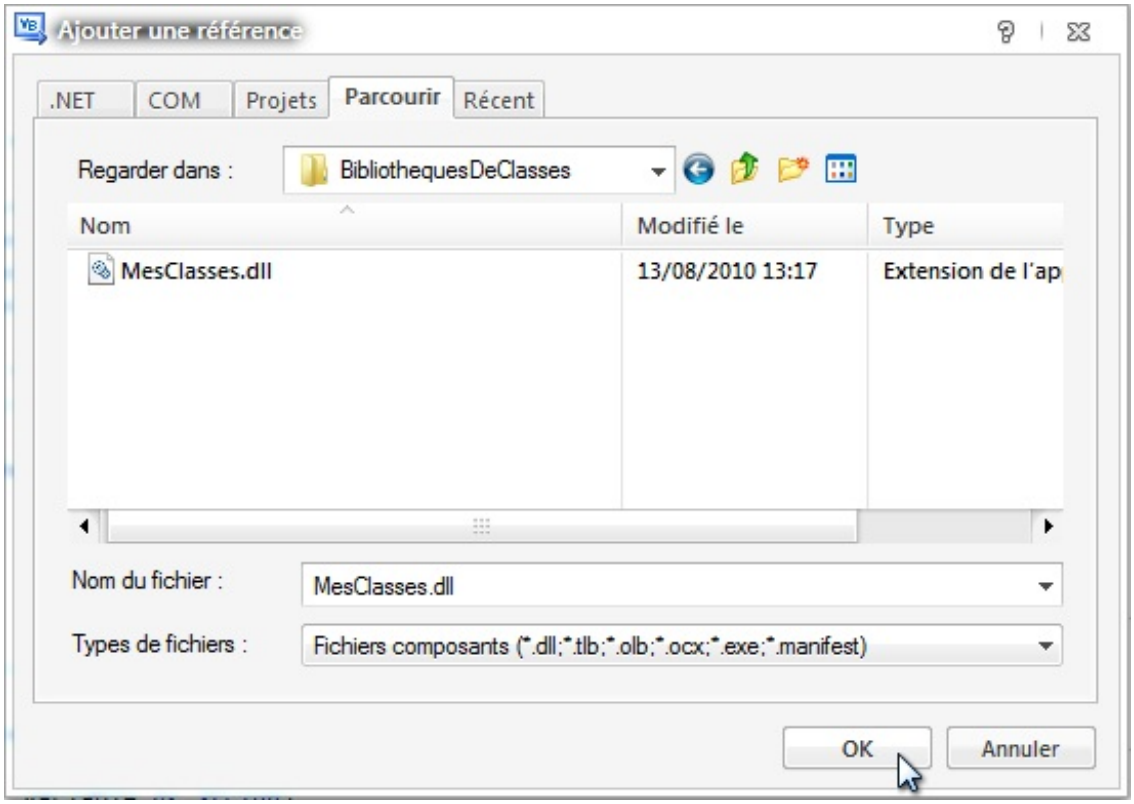
Alors, maintenant pour pouvoir réutiliser nos classes dans un projet il va falloir effectuer une petite manipulation, ajouter une référence.

Nous allons spécifier au projet d'utiliser en plus du framework qui est préincorporé par défaut, notre DLL contenant notre bibliothèque.

Un clic droit sur le projet (où l'on veut utiliser la bibliothèque cette fois puis "Ajouter une référence".



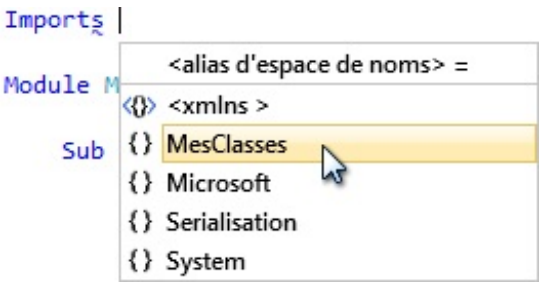
Dans l'onglet "Parcourir", recherchez votre DLL.



Puis ok.

Maintenant il va falloir importer le namespace.

Vous voyez que Visual Studio nous aide :

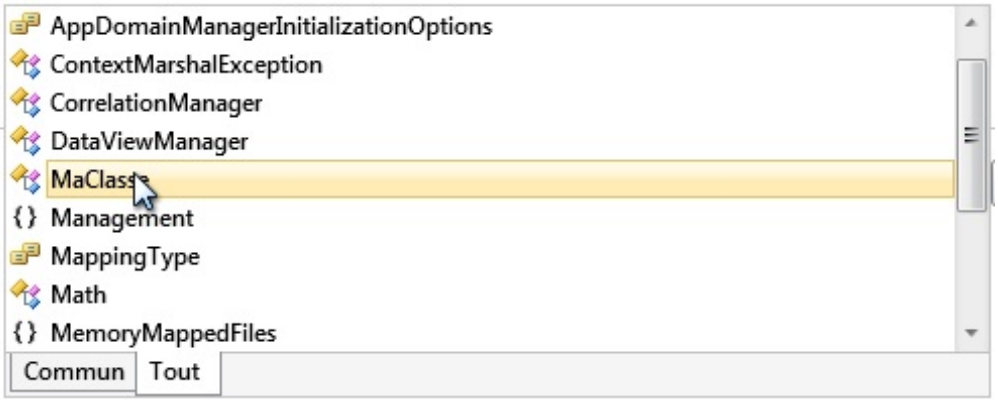


Code : VB.NET

```
Imports MesClasses
```

Et voilà votre bibliothèque est totalement utilisable.

```
Imports MesClasses

Module Module1

    Sub Main()

        Dim UneClasse As New Ma

    End Sub

End Module
```



En espérant que cela va pouvoir trouver une certaine utilité parmi vos projets .

Beaucoup de notions qui sont particulièrement spécifiques à la POO et peuvent vous être d'une grande aide dans certains cas particuliers.

Ne sous-estimez pas ces chapitres, même s'ils sont compliqués à appréhender ils sont très importants.

La sauvegarde d'objets

Attaquons désormais la sauvegarde d'objets.

Vous avez vu comment créer vos objets, vous avez aussi vus comment sauvegarder des données dans des fichiers. Mais on ne peut pas simplement écrire un objet dans un fichier comme s'il s'agissait d'une simple chaîne de caractères, il faut passer par une méthode particulière.

Mesdames, Messieurs les Zeros, je vous présente la sérialisation.

La sérialisation, c'est quoi ?

Vous vous souvenez du cycle de vie d'une variable normale, dés que la boucle, fonction, classe dans laquelle elle a été crée est terminée, la variable est détruite et libérée de la mémoire. Les données qu'elle contient sont donc perdues.

Vous avez dus vous douter que le même principe s'appliquait pour les objets. Et là ce n'est pas une simple valeur de variable qui est perdue mais toutes les variables que l'objet contenant (des attributs). Pour sauvegarder notre variable pour une utilisation ultérieure (un autre lancement du programme), nous avons vus comment stocker cette variable dans un fichier.

Les integers, strings, dates, bref les variables basiques sont facilement stockables mais les objets le sont plus difficilement.

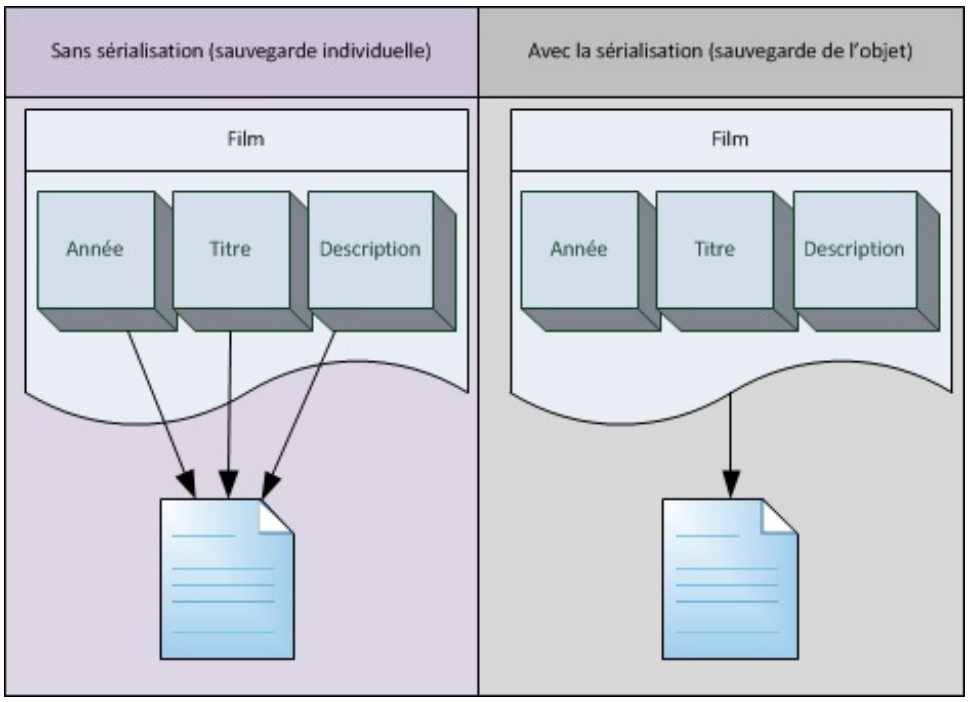
Je vais créer une classe basique contenant des informations concrètes. Le projet que j'ai crée est de type console car l'interface graphique est superflue pour le moment.

Code : VB.NET

```
Public Class Film

    Public Titre As String
    Public Annee As Integer
    Public Desciption As String

    Sub New()

    End Sub

    Sub New(ByVal TitreFilm As String, ByVal AnneeFilm As Integer,
ByVal DescriptionFilm As String)
        Titre = TitreFilm
        Annee = AnneeFilm
        Desciption = DescriptionFilm
    End Sub

End Class
```

Ma classe s'appelle Film, elle va contenir des informations pour créer un objet Film auquel on spécifiera son nom, son année de sortie et une petite description.

Cette classe est très basique, seulement trois attributs. Mais si je veux la sauvegarder il va déjà falloir écrire 3 variables différentes dans un fichier. Donc imaginez avec plusieurs dizaines d'informations (attributs).



Avec la sérialisation (le stockage d'objet) on va pouvoir facilement écrire tous les attributs d'un objet dans un fichier. Ce fichier sera automatiquement formaté. Ce formatage va dépendre de la méthode de sérialisation utilisée.

Il existe donc deux méthodes.

La méthode binaire, elle permet la sauvegarde des attributs Publics et Privés tout en incluant également le nom de la classe dans le fichier.
La seconde méthode, la sérialisation XML est très utile car le format soap (le type de formatage d'un fichier XML) est très répandu, pour communiquer via des Webservices et en général sur l'internet. Ses inconvénients sont que les attributs privés ne sont pas pris en compte et les types des attributs ne sont enregistrés nulle part.

Commençons avec la sérialisation Binaire :

La sérialisation binaire.

Pour commencer je modifie notre classe Film pour spécifier au programme qu'il peut la sérialiser grâce à `<Serializable()>` inscrit dans sa déclaration :

Code : VB.NET

```
<Serializable()>
Public Class Film

    Public Titre As String
    Public Annee As Integer
    Public Desciption As String

    Sub New()

    End Sub

    Sub New(ByVal TitreFilm As String, ByVal AnneeFilm As Integer,
ByVal DescriptionFilm As String)
        Titre = TitreFilm
        Annee = AnneeFilm
        Desciption = DescriptionFilm
    End Sub

End Class
```

Ensuite, dans la page contenant le code qui va permettre de sérialiser l'objet on va faire deux Imports, l'un permettant d'utiliser la sérialisation, le second permettant l'écriture dans un fichier :

Code : VB.NET

```
Imports System.Runtime.Serialization.Formatters.binary
Imports System.IO
```

Je ne sais pas si vous vous souvenez de la partie 1 sur les fichiers mais elle était fastidieuse car je vous faisais travailler sur des flux. La sérialisation reprend le même principe, nous allons ouvrir un flux d'écriture sur le fichier et la fonction de sérialisation va se charger de le remplir.

Donc pour remplir un fichier bin avec un objet que je viens de créer :

Code : VB.NET

```
'On crée l'objet
            Dim Avatar As New Film("Avatar", 2009, "Avatar, film de
James Cameron sorti en décembre 2009.")
            'On crée le fichier et récupère son flux
            Dim FluxDeFichier As FileStream =
File.Create("Film.bin")
            Dim Serialiseur As New BinaryFormatter
            'Serialisation et écriture
            Serialiseur.Serialize(FluxDeFichier, Avatar)
            'Fermeture du fichier
            FluxDeFichier.Close()
```

Et pourquoi .bin l'extension du fichier ?

Ce n'est pas obligatoire de lui assigner cette extension, c'est juste une convention. Comme lorsque nous créions des fichiers de configuration, nous avions tendance à les nommer en .ini, même si nous aurions pu leur donner l'extension .bla.

Le BinaryFormatter est un objet qui va se charger de la sérialisation Binaire. C'est lui qui va effectuer le formatage spécifique à ce type de sérialisation.

L'objet Avatar étant un film est désormais sauvegardé, si je veux récupérer les informations il faut que je le désérialise.

La désérialisation est l'opposé de la sérialisation. Sur le principe de la lecture / écriture dans un fichier. La sérialisation écrit l'objet, la désérialisation le lit. Nous allons utiliser le même BinaryFormatter que lors de la sérialisation.

Programmatiquement :

Code : VB.NET

```
If File.Exists("Film.bin") Then
            'Je crée ma classe "vide"
            Dim Avatar As New Film()
            'On ouvre le fichier et recupere son flux
            Dim FluxDeFichier As Stream = File.OpenRead("Film.bin")
            Dim Deserialiseur As New BinaryFormatter()
            'Désérialisation et conversion de ce qu'on récupère dans
le type "Film"
            Avatar = CType(Deserialiseur.Deserialize(FluxDeFichier),
Film)
            'Fermeture du flux
            FluxDeFichier.Close()
        End If
```

Je vérifie avant tout que le fichier est bien présent. La ligne de désérialisation effectue deux opérations : la désérialisation et la conversion avec Ctype de ce qu'on récupère dans le type de notre classe (ici Film). Et on entre le tout dans "Avatar". Si vous analysez les attributs, vous remarquez que notre film a bien été récupéré.

VB.Net est "intelligent", si vous n'aviez pas effectué de Ctype, il aurait tout de même réussi à l'insérer dans l'objet Avatar. La seule différence est que le Ctype offre une meilleure vue et compréhension de notre programme. La même remarque peut être faite lors de renvoie de valeurs via des fonctions, le type de retour n'est pas forcé d'être spécifié, une variable de type **Object** (spécifique à Visual Basic) sera alors retourné mais pour un programmeur, un code avec des "oublis" de ce type peut très vite devenir incompréhensible et ouvre la porte à beaucoup de possibilité d'erreur.

Bon, je ne vous montre pas le contenu du fichier résultant, ce n'est pas beau à voir. Ce n'est pas fait pour ça en même temps, la sérialisation XML quand à elle va nous donner un résultat plus compréhensible et modifiable manuellement par un simple mortel.

Voyons ça tout de suite.

La sérialisation XML

La sérialisation XML quand à elle respecte le protocole SOAP (Simple Object Access Protocol), le fichier XML que vous allez générer va pourvoir être transporté et utilisé plus facilement sur d'autres plateformes et langages de programmations qui respecteront eux aussi le protocole SOAP.

Le format XML (je vous renvoie à sa page Wikipedia pour plus d'informations) formate le fichier sous une méthode bien spécifique composée de sections et sous-sections.

Je vais reprends l'exemple de Wikipedia pour vous détailler rapidement sa composition :

Code : XML

```
<?xml version="1.0" encoding="UTF-8"?>
  <!-- '''Commentaire''' -->
  <élément-document xmlns="http://exemple.org/" xml:lang=";fr">
    <élément>Texte</élément>
    <élément>élément répété</élément>
    <élément>
      <élément>Hiérarchie récursive</élément>
    </élément>
    <élément>Texte avec<élément>un élément</élément>inclus</élément>
    <élément/><!-- élément vide -->
    <élément attribut="valeur"></élément>
  </élément-document>
```

La première ligne spécifiant une **instruction de traitement**, elle est nécessaire pour des logiciels traitant ce type de fichier, une convention encore une fois.
Viens une ligne de commentaire, elle n'est pas censée nous intéresser car notre sérialisation ne génèrera pas de commentaire.
Par contre le reste est intéressant. La sérialisation va convertir la classe en un noeud principal (élément racine) et ses attributs seront transformés en sous-éléments.

Un exemple simple du résultat de la sérialisation :

Code : VB.NET

```
Public Class ClasseExemple
    Public ValeurNumerique
As Integer
End Class
```

Code : XML

```
<ClasseExemple>

<ValeurNumerique>23</ValeurNumerique>
</ClasseExemple>
```

Vous arrivez à distinguer à l'oeil nu les corrélations qui sont présentes entre la classe et sa sérialisation, ce qui va être beaucoup plus facile pour les modifications manuelles.

Bon, passons à la programmation.

On va changer un l'import que nous utilisions pour la sérialisation binaire.

Ce qui nous amène à écrire :

Code : VB.NET

```
Imports System.IO
Imports System.Xml.Serialization
```

Où `System.IO` contient toujours de quoi interagir avec les fichiers et `System.Xml.Serialization` les classes nécessaires à la sérialisation XML.

Et dans notre code, presque aucun changement :

Code : VB.NET

```
Dim Avatar As New Film("Avatar", 2009, "Avatar, film de James
Cameron sorti en décembre 2009.")
'On crée le fichier et récupère son flux
Dim FluxDeFichier As FileStream = File.Create("Film.xml")
Dim Serialiseur As New XmlSerializer(GetType(Film))
'Serialisation et écriture
Serialiseur.Serialize(FluxDeFichier, Avatar)
'Fermeture du fichier
FluxDeFichier.Close()
```

Eh oui, le principe de sérialisation reste le même, c'est juste un objet de type **XmlSerializer** qu'on instancie cette fois-ci. Il faut passer un argument à son constructeur : le type de l'objet que nous allons sérialiser. Ici c'est ma classe film donc la fonction `GetType(Film)` convient parfaitement.

Une fois la sérialisation effectuée, le fichier en résultant contient :

Code : XML

```
<?xml version="1.0"?>
<Film xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Titre>Avatar</Titre>
  <Annee>2009</Annee>
  <Desciption>Avatar, film de James Cameron sorti en décembre
2009.</Desciption>
</Film>
```

Un beau fichier bien formaté !

Pour la désérialisation, même principe :

Code : VB.NET

```
If File.Exists("Film.xml") Then
    'Je crée ma classe "vide"
    Dim Avatar As New Film()
    'On ouvre le fichier et recupere son flux
    Dim FluxDeFichier As Stream = File.OpenRead("Film.xml")
    Dim Deserialiseur As New XmlSerializer(GetType(Film))
    'Désérialisation et convertion de ce qu'on récupère dans le type
"Film"
    Avatar = CType(Deserialiseur.Deserialize(FluxDeFichier), Film)
    'Fermeture du flux
    FluxDeFichier.Close()
End If
```

Juste un petit mot sur la taille du fichier résultant.

Même si le fichier XML semble plus long leur taille ne diffère que du type de formatage qu'utilise XML.

Si votre objet contient 1Ko de données programmatiquement parlant, la sérialisation n'est pas là pour réduire cette taille. Au contraire, le formatage qu'apporte la sérialisation (binaire ou XML) ne tend qu'à l'alourdir.

La sérialisation multiple

Bon, jusqu'à présent aucun problème pour ce qui est de sérialiser notre petit film "Avatar".

Mais immaginez que nous voulons enregistrer toute une bibliothèque de films (une idée de TP ?).

Déjà, avant que vous vous enfonciez je vous dis qu'avec notre méthode ça ne va pas être possible.

Et pourquoi ça ?

Eh bien vous avez vus que le noeud principal de notre document XML est <Film>, et j'ai dit qu'il ne pouvait avoir qu'un seul nœud principal par document XML. autrement dit, qu'un seul film.
Certes il reste l'idée de créer un fichier XML par Film à sauvegarder mais je pense que ce n'est pas la meilleure méthode .

Rappelez vous de nos amis les tableaux. Un tableau de string, vous vous souvenez, et dans une TP je vous ai tendu un piège en faisant un tableau de RadioButtons.

Ça va être le même principe ici, un tableau de films.

Oui, il fallait juste y penser. Ce qui nous donne en création d'un tableau de films et sérialisation du tout :

Code : VB.NET

```
Dim Films(1) As Film
Films(0) = New Film("Avatar", 2009, "Avatar, film de James Cameron
sorti en décembre 2009.")
Films(1) = New Film("Twilight 3", 2010, "Troisième volet de la
quadrilogie Twilight")
'On crée le fichier et récupère son flux
Dim FluxDeFichier As FileStream = File.Create("Films.xml")
Dim Serialiseur As New XmlSerializer(GetType(Film()))
'Serialisation et écriture
Serialiseur.Serialize(FluxDeFichier, Films)
'Fermeture du fichier
FluxDeFichier.Close()
```

Pour la création du tableau, je ne vous fais pas d'explication je pense. Un tableau de 2 cases, Avatar dans la première, Twilight 3 dans la seconde (il va falloir que je mette à jour ce tuto dans le futur, avec les films du moment).

La seule particularité est le type que l'on fournit au XmlSerializer. Ce n'est pas un simple `GetType(Film)` , autrement dit le type de ma classe Film mais c'est `GetType(Film())` , le type d'un tableau de ma classe Film.

Une fois la sérialisation effectuée, notre fichier se compose ainsi :

Code : XML

```
<?xml version="1.0"?>
<ArrayOfFilm xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Film>
    <Titre>Avatar</Titre>
    <Annee>2009</Annee>
    <Desciption>Avatar, film de James Cameron sorti en décembre
2009.</Desciption>
  </Film>
  <Film>
    <Titre>Twilight 3</Titre>
```

```
    <Annee>2010</Annee>
    <Desciption>Troisième volet de la quadrilogie
Twilight</Desciption>
  </Film>
</ArrayOfFilm>
```

Le nœud principal est désormais un **ArrayOfFilm**, soit un **Tableau de Film**. Et chaque nœud secondaire correspond à un film.

La désérialisation est pratiquement la même elle aussi :

Code : VB.NET

```
If File.Exists("Film.xml") Then
   'On ouvre le fichier et recupere son flux
   Dim FluxDeFichier As Stream = File.OpenRead("Film.xml")
   Dim Deserialiseur As New XmlSerializer(GetType(Film()))
   'Désérialisation et insertion dans le tableau de Film()
   Dim Films() As Film = Deserialiseur.Deserialize(FluxDeFichier)
   'Fermeture du flux
   FluxDeFichier.Close()
End If
```

Ici, même remarque que pour la sérialisation, le type qu'on fournit est bien un tableau de Film() et non plus un Film simple.
Seconde remarque : j'ai effectué la déclaration de mon tableau sur la même ligne que la désérialisation. Cela me permet de m'affranchir de la déclaration fixe du la longeur de mon tableau.
Autrement dit, le programme va se charger tout seul de déterminer combien il y a de films présents et construira un tableau de taille nécessaire. J'ai fait ceci en écrivant **Films()** au lieu de **Films(1)**. Mais le résultat sera le même, et au moins pas de risque d'erreurs de taille =)

Et en ce qui concerne les collections, que nous venons d'aborder ; la méthode est la même :

Code : VB.NET

```
Imports System.IO
Imports System.Xml.Serialization

Module Module1

    Sub Main()

        Dim MaListeDeClasses As New List(Of Classe)
        MaListeDeClasses.Add(New Classe("Avatar"))
        MaListeDeClasses.Add(New Classe("Twilight 1"))
        MaListeDeClasses.Insert(0, New Classe("Titanic"))

        'On crée le fichier et récupère son flux
        Dim FluxDeFichier As FileStream =
File.Create("C:\Classes.xml")
        Dim Serialiseur As New XmlSerializer(GetType(List(Of
Classe)))
        'Serialisation et écriture
        Serialiseur.Serialize(FluxDeFichier, MaListeDeClasses)
        'Fermeture du fichier
        FluxDeFichier.Close()

    End Sub

End Module
```

```
Public Class Classe

    Private _Variable As String

    Sub New()

    End Sub

    Sub New(ByVal Variable As String)
        _Variable = Variable
    End Sub

    Public Property Variable As String
        Get
            Return _Variable
        End Get
        Set(ByVal value As String)
            _Variable = value
        End Set
    End Property

End Class
```

Sur le même principe, le GetType s'effectue sur une List(Of Classe). Et le fichier XML résultant a le même schéma qu'un tableau :

Code : XML

```
<?xml version="1.0"?>
<ArrayOfClasse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Classe>
    <Variable>Titanic</Variable>
  </Classe>
  <Classe>
    <Variable>Avatar</Variable>
  </Classe>
  <Classe>
    <Variable>Twilight 1</Variable>
  </Classe>
</ArrayOfClasse>
```

Il serait même préférable à présent d'utiliser des collections, qui sont plus modulaires et qui représentent mieux les concepts de la POO que des tableau (archaïques) .

En conclusion, la sérialisation est vraiment très pratique, une simple ligne pour sauvegarder tout un objet.

Sur ce principe, une configuration peut aisément être sauvegardé dans un objet fait par vos soins recensant toutes les valeurs nécessaires au fonctionnement de votre programme puis une sérialisation XML vous donnera un fichier clair et formaté, au même titre qu'un fichier .ini. Même si ce n'est pas son utilisation principale, ça peut être une bonne alternative.

Rappelez vous toutefois que la sérialisation XML n'enregistre pas les attributs privés ! Source d'erreur.

Pour combler cette lacune deux solutions : passer tout ses arguments en public, mais cette technique "tue" le principe de la POO ou alors utiliser des propriétés. Les **property**. Si votre attribut est privé et que vous avez créé la property publique correspondante, il sera sérialisé.

Le XML est donc un format générique et, il a été conçu pour stocker des données. Lorsque nous aborderons le chapitre sur les bases de données, les premières notions se feront sûrement avec des documents XML, c'est une base de donnée comme une autre ...

Bref, n'allons pas trop vite, il nous reste a finir cette méchante partie d'orienté objet avant d'attaquer ces autres concepts  .

TP : ZBiblio, la bibliothèque de films

Pour mettre tout ce chapitre en pratique j'ai trouvé comme idée de TP une petite bibliothèque de films. Cette bibliothèque va bien sûr travailler surtout avec un objet de votre cru qui contiendra un film.

Bon je ne vous en dis pas plus, passons au cahier des charges.

Le cahier des charges

Comme dans chaque TP, je vais quand même vous laisser faire ça vous même.

Donc je vais vous donner un cahier des charges, et quelques informations. Tout cela à pour but de vous guider et d'axer vos recherches et vos idées.

Eh oui parce-que les TP qu'on commence à attaquer sont d'une certaine envergure, vous allez vous confronter à des problèmes auxquels vous ne trouverez sûrement pas de solutions dans ce TP (tout dépend de la manière dont vous abordez la situation). Il va donc sûrement falloir (si vous êtes un minimum courageux et que vous ne sautez pas directement à la solution) que vous fassiez quelques recherches sur Google (je ne fais pas de pub) ou alors la MSDN de Microsoft, une bibliothèque recensant toutes les fonctions intégrées au Framework (une annexe ne tardera pas à sortir la concernant). Sinon, si votre problème n'a aucune solution, vous pouvez toujours demander une petite aide sur le forum du SDZ, section "Autres langages" en précédant le titre de votre topic par [VB.NET], ça aide à les distinguer.

Bien bien, excusez moi, je m'égare, le cahier des charges donc.

Ce TP à pour but de vous faire développer une bibliothèque de films (vous pouvez bien sûr transformer des films en musiques, images, ...). Je vais vous laisser libre court concernant le design, les méthodes de programmation. Je vous donne juste quelques lignes de guidage :

- Les films dans la bibliothèque devront être listés dans une liste avec possibilité d'ouvrir la fiche d'un film pour plus d'informations.
- La fiche d'un film contiendra des informations basiques : nom, acteurs, date de sortie, synopsis, tout ce que votre esprit pourra imaginer.
- La possibilité de créer, modifier et supprimer un film (autrement dit la fiche d'un film).
- Et pourquoi pas une fonction de recherche, dans de grandes collections de films.

Concernant la réalisation, je l'ai déjà dit, je le répète, vous faites comme vous voulez.
Quelques conseils cependant :

- Un objet "Film" ou "FicheFilm" serait de rigueur pour contenir les informations d'une fiche film.
- Une collection pour contenir toutes vos fiches ne serait pas une mauvaise idée ?
- La sérialisation pour stocker vos films (je dis ça, je dis rien).

Alors bien évidemment, comme pour les autres TP ne sautez pas directement à la solution. Essayez de chercher un minimum. Ce TP n'a rien de compliqué en soi, il va juste falloir que vous trouviez la bonne méthode pour agir entre les films et la liste. Bref, une réflexion qui a du bon.

Courage et à l'attaque .

La correction.

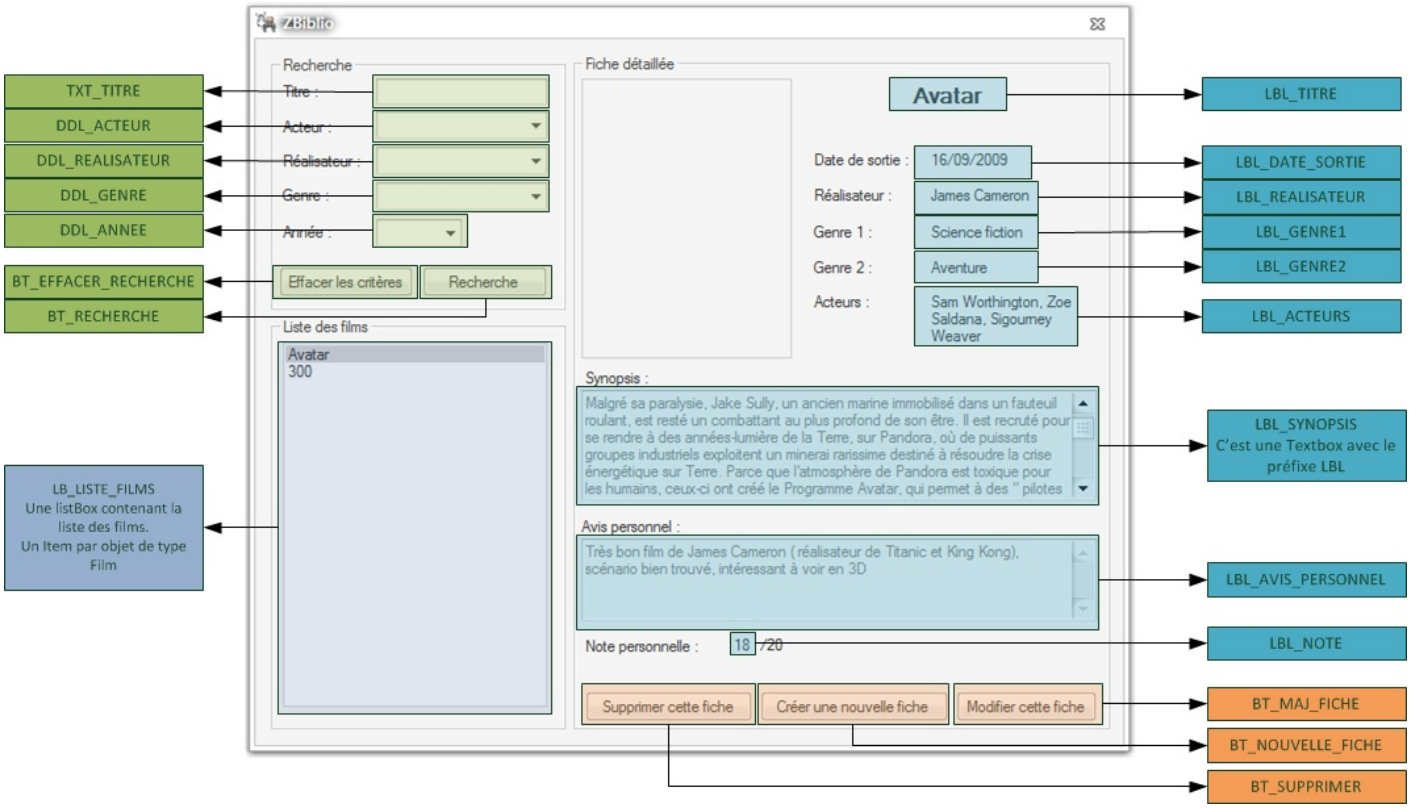
Avant tout je tiens à dire que la correction n'est pas universelle. Elle servira juste à ceux qui n'ont absolument pas réussi à manipuler les objets et autres collection à s'en tirer un minimum.

Mais si vous n'avez rien compris à ce TP et si vous n'avez pas été capable d'en réaliser un ébauche, je ne saurai que vous conseiller de recommencer la lecture de la partie sur la POO.

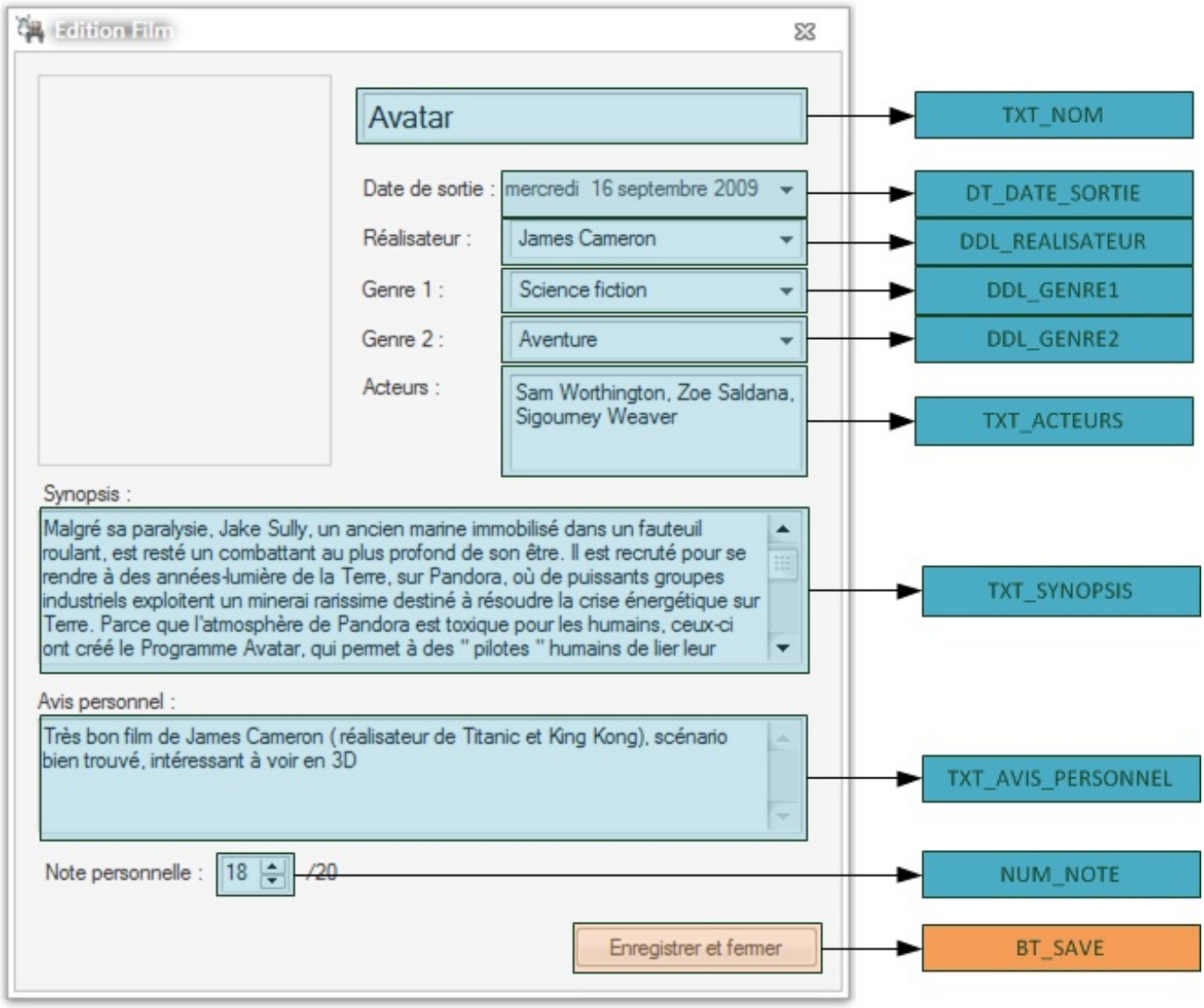
Pour les autres, chaque situation étant différente je vais tâcher de vous présenter un programme aussi concis et universel que possible.

Mon programme est composé de deux fenêtres. La première regroupant les fonctions de recherche, liste et visualisation d'une fiche. La seconde permettant la création et modification d'une fiche.

Je vais déjà vous détailler des deux fenêtres :



Fenêtre principale. Name : ZBiblio



Fenêtre d'ajout/modification. Name : AjoutEditionFilm

Voici mes deux fenêtres. Quelques contrôles spécifiques. Un **DateTimePicker** pour la sélection de la date et un **NumericUpDown** pour la sélection de la note.

Vous avez sans doutes remarqués les **DropDownList** pour les genres et le réalisateur. Elles sont là pour une fonctionnalité ultérieure qui permettra d'insérer les acteurs et genres déjà remplis sur des films pour permettre un choix rapide. Mais en attendant on peut s'en servir comme de simples **TextBox** en utilisant la propriété **Text**.

Je vous demanderai de vous passer de moqueries sur mes interfaces, je n'ai jamais eu l'âme d'un designer.

Bien bien, passons aux feuilles de code.

La classe Film

Code : VB.NET

```
Public Class Film

    Private _Nom As String
    Public Property Nom As String
        Get
            Return _Nom
        End Get
```

```
        Set(ByVal value As String)
            _Nom = value
        End Set
    End Property

    Private _DateSortie As Date
    Public Property DateSortie As Date
        Get
            Return _DateSortie
        End Get
        Set(ByVal value As Date)
            _DateSortie = value
        End Set
    End Property

    Private _Realisateur As String
    Public Property Realisateur As String
        Get
            Return _Realisateur
        End Get
        Set(ByVal value As String)
            _Realisateur = value
        End Set
    End Property

    Private _Genre1 As String
    Public Property Genre1 As String
        Get
            Return _Genre1
        End Get
        Set(ByVal value As String)
            _Genre1 = value
        End Set
    End Property

    Private _Genre2 As String
    Public Property Genre2 As String
        Get
            Return _Genre2
        End Get
        Set(ByVal value As String)
            _Genre2 = value
        End Set
    End Property

    Private _Acteurs As String
    Public Property Acteurs As String
        Get
            Return _Acteurs
        End Get
        Set(ByVal value As String)
            _Acteurs = value
        End Set
    End Property

    Private _Synopsis As String
    Public Property Synopsis As String
        Get
            Return _Synopsis
        End Get
        Set(ByVal value As String)
            _Synopsis = value
        End Set
    End Property

    Private _RemarquePerso As String
```

```
        Public Property RemarquePerso As String
            Get
                Return _RemarquePerso
            End Get
            Set(ByVal value As String)
                _RemarquePerso = value
            End Set
        End Property

        Private _NotePerso As Integer
        Public Property NotePerso As Integer
            Get
                Return _NotePerso
            End Get
            Set(ByVal value As Integer)
                _NotePerso = value
            End Set
        End Property

        Public Sub New()

        End Sub

        Public Sub New(ByVal Nom As String, ByVal DateSortie As Date,
    ByVal Realisateur As String, ByVal Genre1 As String, ByVal Genre2 As
    String, ByVal Acteurs As String, ByVal Synopsis As String, ByVal
    RemarquePerso As String, ByVal NotePerso As Integer)
            _Nom = Nom
            _DateSortie = DateSortie
            _Realisateur = Realisateur
            _Genre1 = Genre1
            _Genre2 = Genre2
            _Acteurs = Acteurs
            _Synopsis = Synopsis
            _RemarquePerso = RemarquePerso
            _NotePerso = NotePerso
        End Sub

        Public Sub Update(ByVal Nom As String, ByVal DateSortie As Date,
    ByVal Realisateur As String, ByVal Genre1 As String, ByVal Genre2 As
    String, ByVal Acteurs As String, ByVal Synopsis As String, ByVal
    RemarquePerso As String, ByVal NotePerso As Integer)
            _Nom = Nom
            _DateSortie = DateSortie
            _Realisateur = Realisateur
            _Genre1 = Genre1
            _Genre2 = Genre2
            _Acteurs = Acteurs
            _Synopsis = Synopsis
            _RemarquePerso = RemarquePerso
            _NotePerso = NotePerso
        End Sub

        'Je surcharge le Tostring
        Public Overrides Function ToString() As String
            Return _Nom
        End Function

    End Class
```

Beaucoup de lignes pour ce qui concerne les propriétés. Mais c'est un passage obligé .

Le constructeur a deux signatures (une signature étant une façon de l'appeler) grâce à une surcharge. Une avec arguments, l'autre sans.

Une méthode **Update** permettant la Mise à jour en une ligne de tous les attributs de la fiche.

Et finalement une surcharge de **ToString** spécifiant qu'il faut retourner le nom du film lorsque j'utiliserai cette fonction.

La fenêtre principale : ZBiblio

Code : VB.NET

```
Imports System.Xml.Serialization
Imports System.IO

Public Class ZBiblio

    Private _FenetreAjout As AjoutEditionFilm
    Private _FilmEnVisualisation As Film
    Private _ListeFilms As List(Of Film)
    Public Property ListeFilms As List(Of Film)
        Get
            Return _ListeFilms
        End Get
        Set(ByVal value As List(Of Film))
            _ListeFilms = value
        End Set
    End Property

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

        'Instancie une nouvelle liste
        _ListeFilms = New List(Of Film)

        'Récupère les infos
        Deserialisation()

        'MAJ la liste de films
        UpdateListe()

    End Sub

    Private Sub ListeFilms_FormClosing(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
MyBase.FormClosing
        'Sérialise les films lors de la fermeture
        Serialisation()
    End Sub

    Private Sub LB_LISTE_FILMS_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
LB_LISTE_FILMS.SelectedIndexChanged

        'On vérifie qu'on a sélectionné quelquechose
        If Not Me.LB_LISTE_FILMS.SelectedItem Is Nothing Then
            'Retrouve le film avec ce nom
            For Each FilmALister As Film In _ListeFilms
                If FilmALister.Nom =
LB_LISTE_FILMS.SelectedItem.ToString Then
                    'L'insère dans une variable globale
                    Me._FilmEnVisualisation = FilmALister
                End If
            Next

            'On MAJ les infos de la fiche
            Me.LBL_TITRE.Text = Me._FilmEnVisualisation.Nom
```

```
            Me.LBL_DATE_SORTIE.Text =
Me._FilmEnVisualisation.DateSortie.ToShortDateString 'La date seule
            Me.LBL_GENRE1.Text = Me._FilmEnVisualisation.Genre1
            Me.LBL_GENRE2.Text = Me._FilmEnVisualisation.Genre2
            Me.LBL_REALISATEUR.Text =
Me._FilmEnVisualisation.Realisateur
            Me.LBL_ACTEURS.Text = Me._FilmEnVisualisation.Acteurs
            Me.LBL_SYNOPSIS.Text = Me._FilmEnVisualisation.Synopsis
            Me.LBL_AVIS_PERSONNEL.Text =
Me._FilmEnVisualisation.RemarquePerso
            Me.LBL_NOTE.Text = Me._FilmEnVisualisation.NotePerso

        End If

    End Sub

    Public Sub UpdateListe()
        'On vide la liste et on la reremplit
        Me.LB_LISTE_FILMS.Items.Clear()
        'Parcours les films de la bibliotheque
        For Each FilmALister As Film In _ListeFilms
            'Remplit la liste en se basant sur le nom (vu que j'ai
surchargé tostring)
            'A le même effet que FilmALister.Nom sans la surcharge.
            Me.LB_LISTE_FILMS.Items.Add(FilmALister)
        Next
    End Sub

#Region "Boutons modif fiche"

    Private Sub BT_SUPPRIMER_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles BT_SUPPRIMER.Click

        'Confirmation
        If MsgBox("Etes vous certain de vouloir supprimer ce film ?
", vbYesNo, "Confirmation") Then
            'On le retire de la liste
            Me._ListeFilms.Remove(_FilmEnVisualisation)
        End If

        'MAJ
        UpdateListe()

    End Sub

    Private Sub BT_NOUVELLE_FICHE_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
BT_NOUVELLE_FICHE.Click
        'Si nouveau film, on passe nothing
        _FenetreAjout = New AjoutEditionFilm(Nothing)
        _FenetreAjout.Show()
    End Sub

    Private Sub BT_MAJ_FICHE_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles BT_MAJ_FICHE.Click
        'Si une maj on passe le paramètre du film actuel
        _FenetreAjout = New AjoutEditionFilm(_FilmEnVisualisation)
        _FenetreAjout.Show()
    End Sub

#End Region

#Region "Sauvegarde et récupération"

    Private Sub Deserialisation()
        If File.Exists("BibliothequeFilm.xml") Then
            'On ouvre le fichier et recupere son flux
```

```
            Dim FluxDeFichier As Stream =
File.OpenRead("BibliothequeFilm.xml")
            Dim Deserialiseur As New XmlSerializer(GetType(List(Of
Film)))
            'Désérialisation et convertion de ce qu'on récupère dans
le type "Film"
            _ListeFilms = Deserialiseur.Deserialize(FluxDeFichier)
            'Fermeture du flux
            FluxDeFichier.Close()
        End If
    End Sub

    Private Sub Serialisation()
        'On crée le fichier et récupère son flux
        Dim FluxDeFichier As FileStream =
File.Create("BibliothequeFilm.xml")
        Dim Serialiseur As New XmlSerializer(GetType(List(Of Film)))
        'Serialisation et écriture
        Serialiseur.Serialize(FluxDeFichier, _ListeFilms)
        'Fermeture du fichier
        FluxDeFichier.Close()
    End Sub

#End Region

#Region "Section recherche"

    Private Sub RemplissageChampsRecherche()
        'Fonction utilisée plus tard pour pré-replir les DDL de
genres / réalisteurs ...
    End Sub

    Private Sub BT_RECHERCHE_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles BT_RECHERCHE.Click
        Recherche()
    End Sub

    Private Sub BT_EFFACER_RECHERCHE_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
BT_EFFACER_RECHERCHE.Click
        Me.TXT_TITRE.Text = ""
        Me.DDL_ACTEUR.Text = ""
        Me.DDL_ANNEE.Text = ""
        Me.DDL_GENRE.Text = ""
        Me.DDL_REALISATEUR.Text = ""

        UpdateListe()
    End Sub

    Private Sub Recherche()

        'On vide la liste et on la reremplit
        Me.LB_LISTE_FILMS.Items.Clear()
        'Parcours les films de la bibliotheque
        For Each FilmALister As Film In _ListeFilms

            If Me.TXT_TITRE.Text <> "" Then
                If FilmALister.Nom.Contains(Me.TXT_TITRE.Text) Then
                    Me.LB_LISTE_FILMS.Items.Add(FilmALister)
                End If
            End If

            If Me.DDL_ACTEUR.Text <> "" Then
                If FilmALister.Acteurs.Contains(Me.DDL_ACTEUR.Text)
Then
                    Me.LB_LISTE_FILMS.Items.Add(FilmALister)
                End If
```

```
                End If

                If Me.DDL_ANNEE.Text <> "" Then
                    If CDate(FilmALister.DateSortie).Year =
    Me.DDL_ANNEE.Text Then
                        Me.LB_LISTE_FILMS.Items.Add(FilmALister)
                    End If
                End If

                If Me.DDL_GENRE.Text <> "" Then
                    If FilmALister.Genre1.Contains(Me.DDL_GENRE.Text) Or
    FilmALister.Genre2.Contains(Me.DDL_GENRE.Text) Then
                        Me.LB_LISTE_FILMS.Items.Add(FilmALister)
                    End If
                End If

                If Me.DDL_REALISATEUR.Text <> "" Then
                    If
    FilmALister.Realisateur.Contains(Me.DDL_REALISATEUR.Text) Then
                        Me.LB_LISTE_FILMS.Items.Add(FilmALister)
                    End If
                End If

            Next

        End Sub

    #End Region

    End Class
```

Cette fenêtre contient les fonctions de sérialisation et désérialisation. En ce qui concerne la sérialisation elle s'effectue automatiquement lors de la fermeture de la fenêtre (autrement dit du programme). La désérialisation quand à elle, se lance au démarrage du programme en vérifiant bien évidemment la présence du fichier xml.

Une propriété permettant de modifier la variable **_ListeFilms** est publique de façon à pouvoir l'appeler depuis l'autre fenêtre. Il en va de même pour la méthode **Update**.

Pour le reste, je pense que mes commentaires sont assez explicites  .

Fenêtre d'ajout et de modification.

Code : VB.NET

```
Public Class AjoutEditionFilm

    Private _FilmAModifier As Film

    Sub New(ByVal FilmAModifier As Film)

        ' Cet appel est requis par le concepteur.
        InitializeComponent()

        ' Ajoutez une initialisation quelconque après l'appel InitializeComponer

        'Recupere le film à modifier
        _FilmAModifier = FilmAModifier

    End Sub

    Private Sub AjoutEditionFilm_Load(ByVal sender As System.Object, ByVal e As
```

```
System.EventArgs) Handles MyBase.Load

        If _FilmAModifier Is Nothing Then
            'S'il ne contient rien, on en crée un nouveau
        Else
            'Sinon on recupère les infos et on les entre dans les cases
correspondantes
            Me.TXT_ACTEURS.Text = _FilmAModifier.Acteurs
            Me.TXT_AVIS_PERSONNEL.Text = _FilmAModifier.RemarquePerso
            Me.TXT_NOM.Text = _FilmAModifier.Nom
            Me.TXT_SYNOPSIS.Text = _FilmAModifier.Synopsis
            Me.DDL_GENRE1.Text = _FilmAModifier.Genre1
            Me.DDL_GENRE2.Text = _FilmAModifier.Genre2
            Me.DDL_REALISATEUR.Text = _FilmAModifier.Realisateur
            Me.NUM_NOTE.Value = _FilmAModifier.NotePerso
            Me.DT_DATE_SORTIE.Value = _FilmAModifier.DateSortie
        End If

    End Sub

    Private Sub BT_SAVE_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BT_SAVE.Click

        If _FilmAModifier Is Nothing Then
            'Enregistre notre film
            Dim MonFilm As New Film(Me.TXT_NOM.Text, Me.DT_DATE_SORTIE.Value,
Me.DDL_REALISATEUR.Text, Me.DDL_GENRE1.Text, Me.DDL_GENRE2.Text, Me.TXT_ACTEURS.
Me.TXT_SYNOPSIS.Text, Me.TXT_AVIS_PERSONNEL.Text, Me.NUM_NOTE.Value)
            'On l'ajoute à la liste
            ZBiblio.ListeFilms.Add(MonFilm)
            MsgBox("Fiche correctement crée", vbOKOnly, "Confirmation")
        Else
            'Sinon on le modifie en récupérant son index dans la liste de la fen
parent

ZBiblio.ListeFilms(ZBiblio.ListeFilms.IndexOf(_FilmAModifier)).Update(Me.TXT_NOM
Me.DT_DATE_SORTIE.Value, Me.DDL_REALISATEUR.Text, Me.DDL_GENRE1.Text,
Me.DDL_GENRE2.Text, Me.TXT_ACTEURS.Text, Me.TXT_SYNOPSIS.Text,
Me.TXT_AVIS_PERSONNEL.Text, Me.NUM_NOTE.Value)
        End If

        'MAJ de la liste dans la fenêtre parent
        ZBiblio.UpdateListe()
        'Ferme la fenêtre d'édition
        Me.Close()
    End Sub

    Private Sub IMG_AFFICHE_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles IMG_AFFICHE.Click
        'Ultérieur ; possibilité d'ajouter une fiche.
    End Sub

End Class
```

J'ai ajouté manuellement le constructeur de cette fenêtre. Lorsque vous créez le constructeur, automatiquement il ajoute :

Code : VB.NET

```
  ' Cet appel est requis par le concepteur.
        InitializeComponent()

        ' Ajoutez une initialisation quelconque après l'appel
  InitializeComponent().
```

Il ne faut surtout pas effacer, c'est ce qui initialise les éléments graphiques. Ajoutez ce que vous voulez après. Personnellement j'ai demandé un argument au constructeur, cet argument étant le film à modifier. Si c'est **nothing** qui est passé, cela signifie que c'est une nouvelle fiche qu'il faut créer.

Pour le reste, même remarque : les commentaires doivent être assez clairs à mon avis.

Si un problème vient à vous bloquer, n'hésitez pas à laisser un commentaire sur ce chapitre détaillant votre problème.

Améliorations possibles

Concernant les amélioration possibles, je pense qu'il y a vraiment possibilité de faire évoluer ce petit logiciel jusqu'à le rendre agréable à l'utilisation.

- Ajout d'une affiche de film. Cette affiche pourrait être sauvegardée dans un dossier les regroupant toutes.
- Remplir mes DDL, ajouter autant d'éléments que de possibilités. Par exemple, pour les genres. Faites une recherche sur tous les genres de tous les films et créer une fonction permettant de les distinguer afin de garder des genre distincts, pas de double. Ce n'est qu'une idée  .
- Améliorer un peu l'interface, faire pourquoi pas un système d'onglets, une image de fond, etc.

Cette liste n'est évidemment pas exhaustive, elle ne dépend que de votre imagination. Je prévois cependant d'améliorer ce TP lors de la partie concernant les bases de données et une troisième version à la fin du tutoriel complet.

Partie 4 : Annexes

Les annexes peuvent être consultées n'importe quand, elles peuvent servir pour sécuriser votre programme ou l'améliorer en général.

Gérer les erreurs

Pourquoi ?

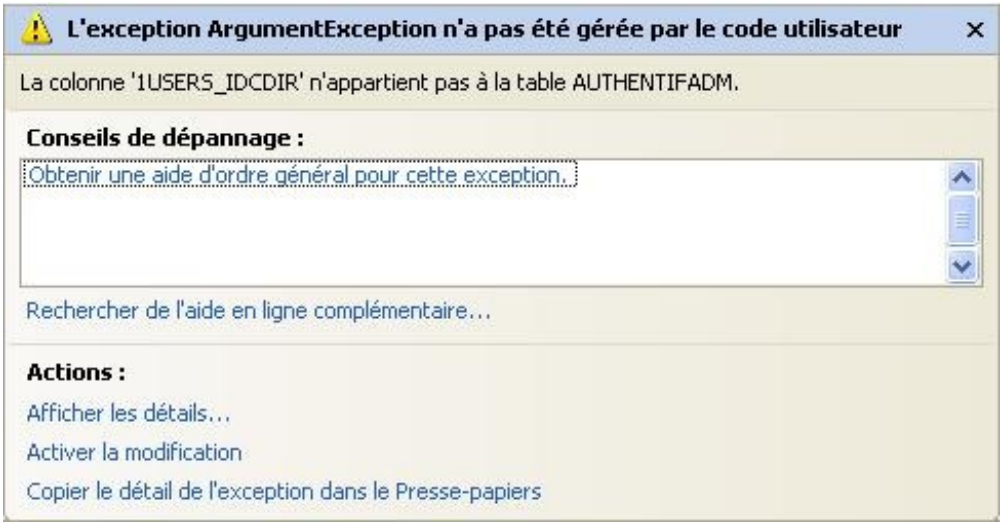
Toi zéro, ou alors toi, développeur expérimenté, tu as déjà lancé un programme qui a provoqué une erreur !

Oui je le sais ! C'est obligé !

Bon, trêves de plaisanteries, les erreur, c'est parfois bien : ça aide à progresser.

Votre superbe IDE : Visual Basic Express 2010, ou Visual Studio faute de ; permet facilement de retrouver et de gérer les erreur. Il indique la ligne ayant provoqué l'erreur, l'erreur expliquée (en Français) et des fois comment la résoudre.

Mais pour ce qui est des autres erreurs ? Les erreurs qui ne sont pas liées à notre programme.



Ras la bol de faire 50 if pour vérifier si la base de données à laquelle on veut accéder est bien là, voir si la table est bien présent, voir si la requête a bien fonctionné etc ...

Nous allons donc utiliser un autre point de vue pour gérer ces passages : la gestion d'erreur.

Vous allez découvrir le ... **Try** !

Découvrons le Try

Le **try** c'est quoi ? Ça se mange ?

Bon pour vous expliquer simplement : le mot **try** est le mot anglais pour *essayer*.

Le programme va donc essayer les lignes de code que vous définirez, si une erreur se présente dedans, il va automatiquement dans une partie de votre programme, et l'erreur ne vous saute pas aux yeux comme si vous veniez de tuer quelqu'un

Syntaxe :

Code : VB.NET

```
Try
    'Code à exécuter
End Try
```

Donc, ce code exécutera ce qu'il y a entre le **try** et son **end try**, si une erreur se produit il sort du **try**.

Pour le moment pas compliqué donc.

Gros malin, ça m'avance pas ! Une erreur et je me retrouve à la fin de mon programme direct !

D'où l'intérêt d'utiliser ce try dans chaque fonctions ! A chaque début de fonction vous mettez votre try et à la fin votre end try !

Si la fonction échoue, elle sera ignorée c'est tout !

Et alors ? Une fois de retour ou la fonction a été appelée si je n'ai pas de valeurs ça va également planter !

Mmmmmh c'est pas faux tout ça, passons alors au finally

Finally

Dans le **try** nous avons d'autres instruction pour nous aider : Tout d'abord le **finally**.

Je vous ai dit que si une erreur se produisait dans le **try**, il sautait tout. Oui mais dans une fonction ça va faire quoi ? Si il saute tout même le retour de la fonction ?

Code : VB.NET

```
Function Erreur() as integer
    Try
        'Code pas très sûr
    Finally
       Return 0
    End Try
End Function
```

Et voilà la solution : si une erreur se produit, paf il saute dans le finally et il retourne une valeur quoi qu'il arrive (même si aucune erreur n'est à déclarer) !

Donc si vous avez suivis depuis le début, vous mettez un return d'office dans le **finally** et un return dans le déroulement normal de la fonction. Si aucune erreur n'est a déplorer, le return de votre fonction aura la priorité et rendra l'autre inopérant.

Si c'est une demande de connections à un site web, que le site en question ne trouve pas la base de données, on aura une erreur mais l'utilisateur sera bloqué ... Que faire ?

On va les renvoyer.

Catch, throw

Notre salut : le catch !

Se place comme le **finally** entre le **try** et son **end try**, mais **avant** le **finally**

Le catch va nous permettre de récupérer l'erreur dans une variable de type **exception**.

Code : VB.NET

```
Catch ex As Exception
```

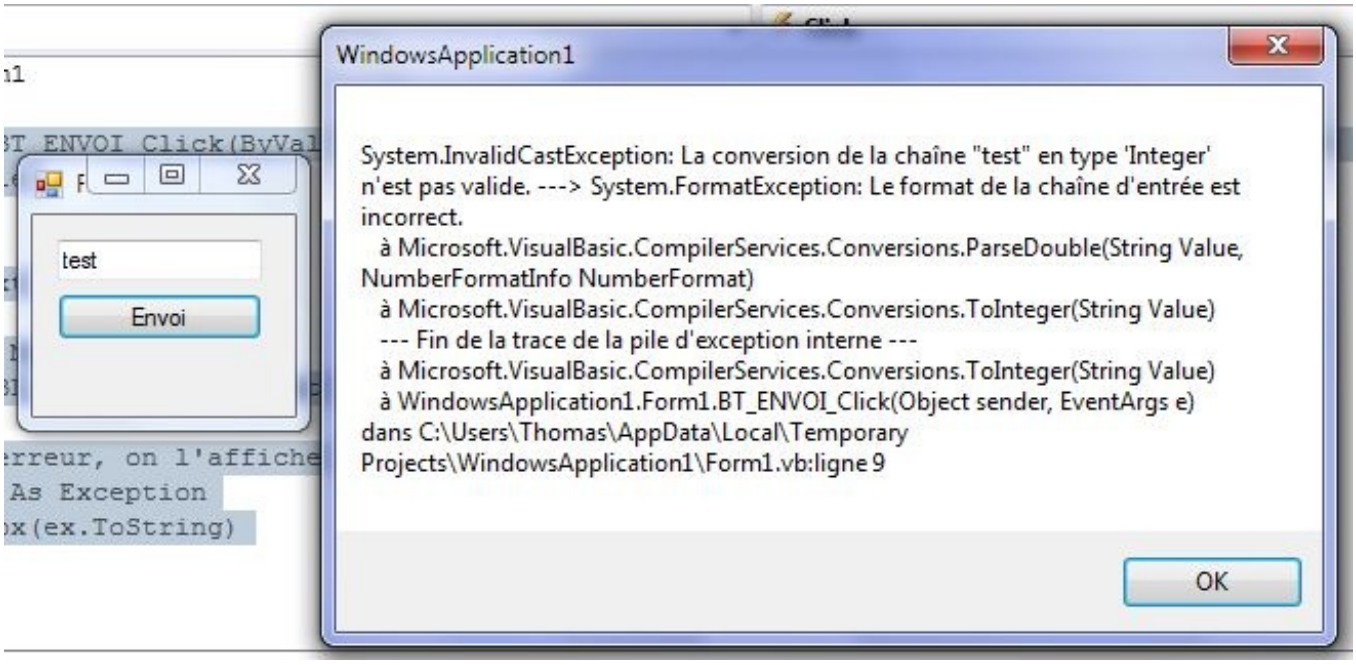
Que j'appelle ici "ex".

Ensuite je peux récupérer cette variable et m'en servir pour afficher l'erreur par exemple :

Code : VB.NET

```
Private Sub BT_ENVOI_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BT_ENVOI.Click
        'On essaie
        Try

        Dim MonTxt As Integer

        MonTxt = Me.TXT_IN.Text
            Me.LBL_OUT.Text = MonTxt

            'Si erreur, on l'affiche
        Catch ex As Exception
MsgBox(ex.ToString)
        End Try

    End Sub
```

Je l'affiche donc dans une MsgBox.
Voici le résultat :



Vous allez me dire que l'utilisateur lambda n'en a rien à faire de notre message d'erreur que lui il veut que ca marche !

Mais l'affichage n'est pas forcément nécessaire : on peut récupérer cette variable, l'écrire dans un fichier log, les possibilités sont multiples ou alors on la renvoie.

 Pardon ?

Oui on la renvoie à l'étage du dessus : si c'est dans une fonction que l'erreur se produit ;

Code : VB.NET

```
Private Sub BT_ENVOI_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BT_ENVOI.Click
        Try
            Bouton()
        Catch ex As Exception
            MsgBox(ex.ToString)
        End Try
    End Sub

    Private Sub Bouton()
        'On essaie
        Try

            Dim MonTxt As Integer

            MonTxt = Me.TXT_IN.Text
            Me.LBL_OUT.Text = MonTxt

            'Si erreur, on la renvoie à la fonction qui l'a appelée
        Catch ex As Exception
Throw ex
        End Try
    End Sub
```

Donc, ici, j'envoie à l'instance du dessus l'erreur, une fois de retour dans cette fonction, le programme voit qu'une erreur s'est produite en amont, elle rentre elle-même dans son **catch**.

Inutile me diriez-vous ? Pas forcément, pourquoi ne pas utiliser ce **try**, **catch** avec son **throw** dans tout vos accès aux bases de données et ne pas utiliser un **try catch** avec une gestion spécifique d'erreur dans la fonction qui les appelles toutes ?

Une seule gestion d'erreur pour vérifier des dizaines de requêtes, ce n'est pas magnifique  !

Les ressources

Vous l'avez compris, le VB est essentiellement basé sur le design de l'interface utilisateur.

C'est bien beau ce que l'on fait pour le moment mais on a toujours pas vu comment ajouter une image, un son, une vidéo ... Bref c'est ce qu'on appelle une ressource, on étudie ça tout de suite.

Qu'est-ce qu'une ressource

Une ressource en VB va contenir des données "externes". Cela peut être une image que l'on veut en arrière plan de fenêtre, un son qu'il faudra jouer pendant un jeu, ou même un chaine de caractère que l'on veut facilement modifiable.

J'appelle mon ami Wikipedia : Ressources

Pour vous résumer le tout :

Les ressources sont des données **statiques** (au même titre que les constantes) qui sont intégrées à l'exe ou aux DLL lors de la compilation. Donc si vous insérez toutes vos images, videos, etc ... En tant que ressources, l'utilisateur ne verra pas un dossier à rallonge avec toutes les images utilisées pendant votre programme, elles seront intégrées dans l'exe, dans les DLL pour un projet plus conséquent.

Mais attention, le système des ressources n'est pas infaillible. Si vous intégrez des informations en tant que ressources, elles pourront toujours être récupérées. Il existe des "décompilateurs" de ressources permettant de faire ressortir les ressources utilisées dans un exe.

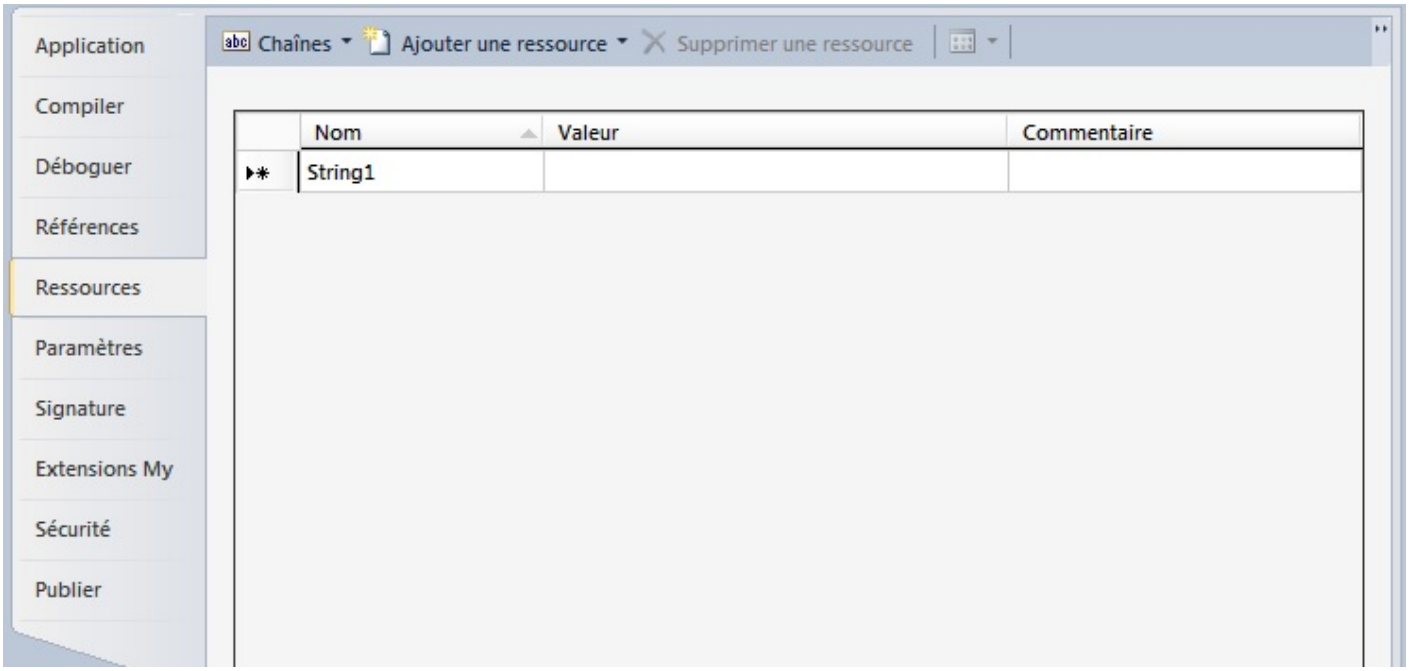
L'utilisation de ressources est habituellement une tâche fastidieuse : compilation et intégration de ses ressources puis la récupération ... Bref les programmeurs hésitent des fois à les utiliser.

Dans notre cas ça va être un véritable jeu d'enfant d'intégrer et d'utiliser des ressources, les assistants de visual studio se chargent de tout.

Découvrons tout de suite comment cela se présente :

Les ressources dans VB 2010

Vous avez surement déjà vu l'onglet "ressources" lorsque vous vous situez dans la fenêtre de configuration de votre projet :

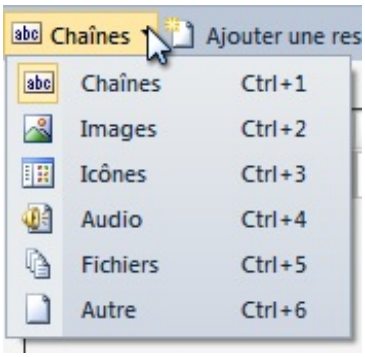


C'est là que l'on va se rendre pour ajouter nos ressources, les éditer etc ...

Rendez vous donc sur l'icône "My Project" dans l'explorateur de solutions puis onglet "Ressources".

Vous tombez nez à nez avec une grande zone blanche et vide, c'est ici que viendront s'ajouter nos ressources. Vous êtes actuellement sur le tableau des strings. Ce sont les ressources de type chaînes de caractères, vous pourrez stocker les chaînes de connection à la BDD lorsque nous y serons ou simplement des titres, des noms, etc ...

Utilisez la petite flèche à côté de "Chaînes" pour naviguer entre les différents écrans de ressources (images, videos, ...).



Vos ressources sont bien organisées et classées.

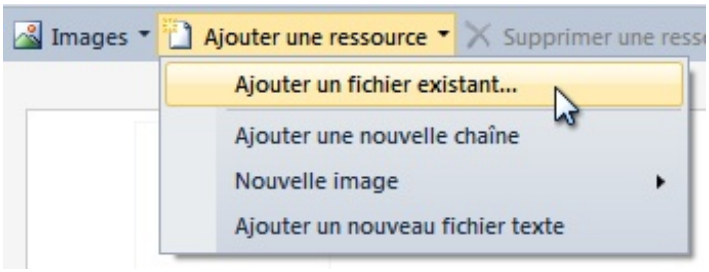
Ajoutons nos ressources

Nous allons avoir deux grandes manières d'ajouter nos ressources.

Prenons les images comme exemple.

Vous allez pouvoir soit ajouter un fichier contenant déjà une image.
Vous vous souvenez sûrement du TP sur la navigateur web, à la fin de ce TP une partie "design" nous apprenait a utiliser les images en tant que ressources "externes", cette fois nous allons utiliser les images en "interne".

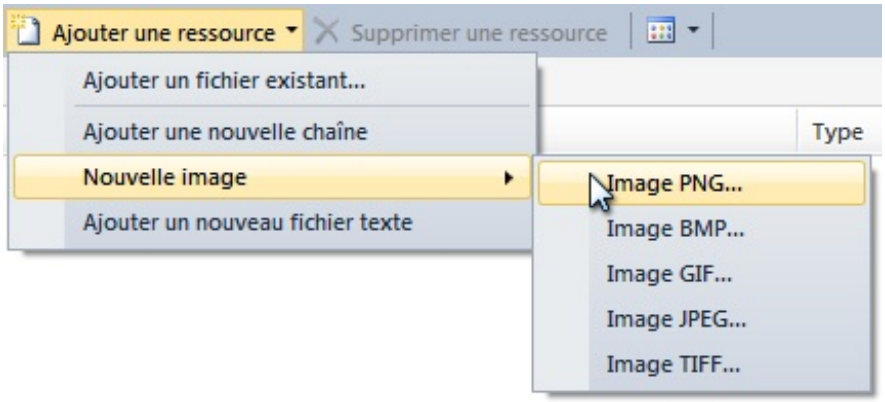
Donc placez-vous dans la fenêtre ressources dédiée aux images, cliquez sur ajouter une ressource, fichier existant.



Selectionnez ensuite l'image souhaitée.

Vous vous appercevez qu'elle s'ajoute directement et qu'un aperçu est disponible.

La seconde manière d'ajouter un fichier ressource est de le créer directement :



Rendez vous sur Nouvelle image > Le type que vous souhaitez, donnez lui un nom, pour moi ce sera "fond" et votre éditeur d'images préféré s'ouvrira (pour moi ce sera paint).

Créez un motif basique : 

Puis sauvegardez, vous voici avec une ressource rapidement crée.

Vous pouvez faire de même avec un fichier texte (très pratique lorsque vous voulez un fichier de configuration caché).
Attention tout de fois, avec cette méthode, il faut que l'utilisateur lance le programme avec les droits en écriture (le plus souvent administrateur) pour avoir accès à cette fonctionnalité.

Pour ce qui est des chaînes de caractères, inscrivez simplement le nom de la Clé (comme dans un fichier ini), la valeur que vous voulez lui assigner et pourquoi pas un commentaire.

Bien, vous savez maintenant ajouter vos ressources, tâchons de les récupérer.

Récupérons les maintenant

Bon, j'ai crée un nouveau programme de test, vous pouvez faire de même.

J'ai ajouté deux ressources : l'étoile et la chaine de caractères de nom **APP_NOM**.

Essayons de les récupérer.

Rendez-vous dans le form_load de votre application.
Pour accéder aux ressources nous n'allons pas utiliser **Me** en préfixe d'instruction mais **My**.

Je vous explique rapidement l'utilité de My car il sera exploré dans un chapitre futur donc je passe brièvement dessus.

My va permettre d'accéder directement aux fonctionnalités de votre ordinateur. C'est avec My que nous accèderons a l'audio de votre PC, a ses périphériques, aux informations sur l'utilisateur actuel de l'ordinateur, etc ... Finalement c'est aussi là que nous trouverons les ressources que nous avons ajoutés précédemment.

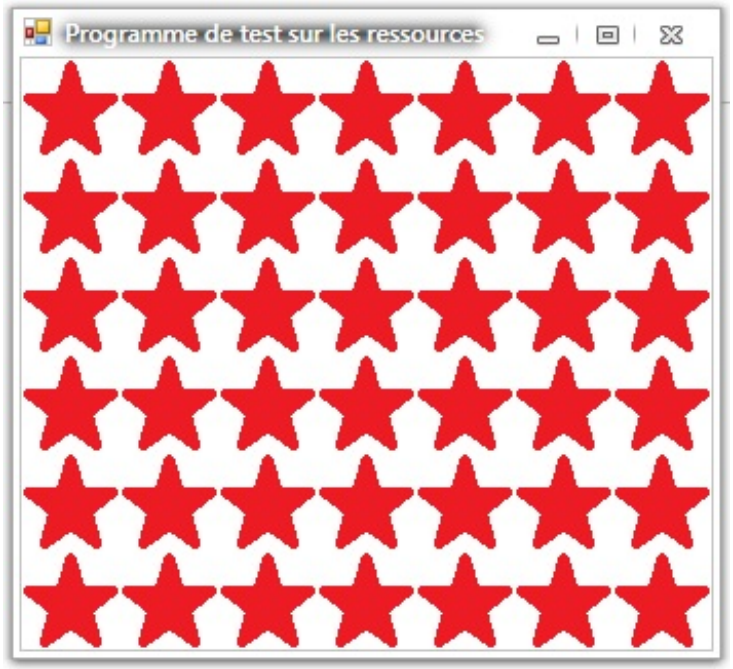
Pour y accéder c'est plus qu'enfantin, il vous suffit d'inscrire `My.Resources.` pour que l'assistant vous affiche les différents noms de vos ressources. Elles sont directement accessibles comme des propriétés.

Donc dans mon cas, je veux donner comme nom à ma fenêtre la valeur de la ressource APP_NOM et en image de fond l'image Fond, il me reste à écrire :

Code : VB.NET

```
Me.Text = My.Resources.APP_NOM
        Me.BackgroundImage = My.Resources.Fond
```

Et je me retrouve avec une fenêtre un peu folklorique :



Mais notre utilisation des ressources est parfaitement fonctionnelle.

Pour l'utilisation des sons et des videos nous aborderons leur utilisation ultérieurement, mais vous savez quand même les ajouter à votre projet.

Le registre

Bon, les ressources incorporées dans l'executable c'est bien beau, mais pour certains programmes il serait plus utile de placer des valeurs (comme de la configuration) dans le registre.

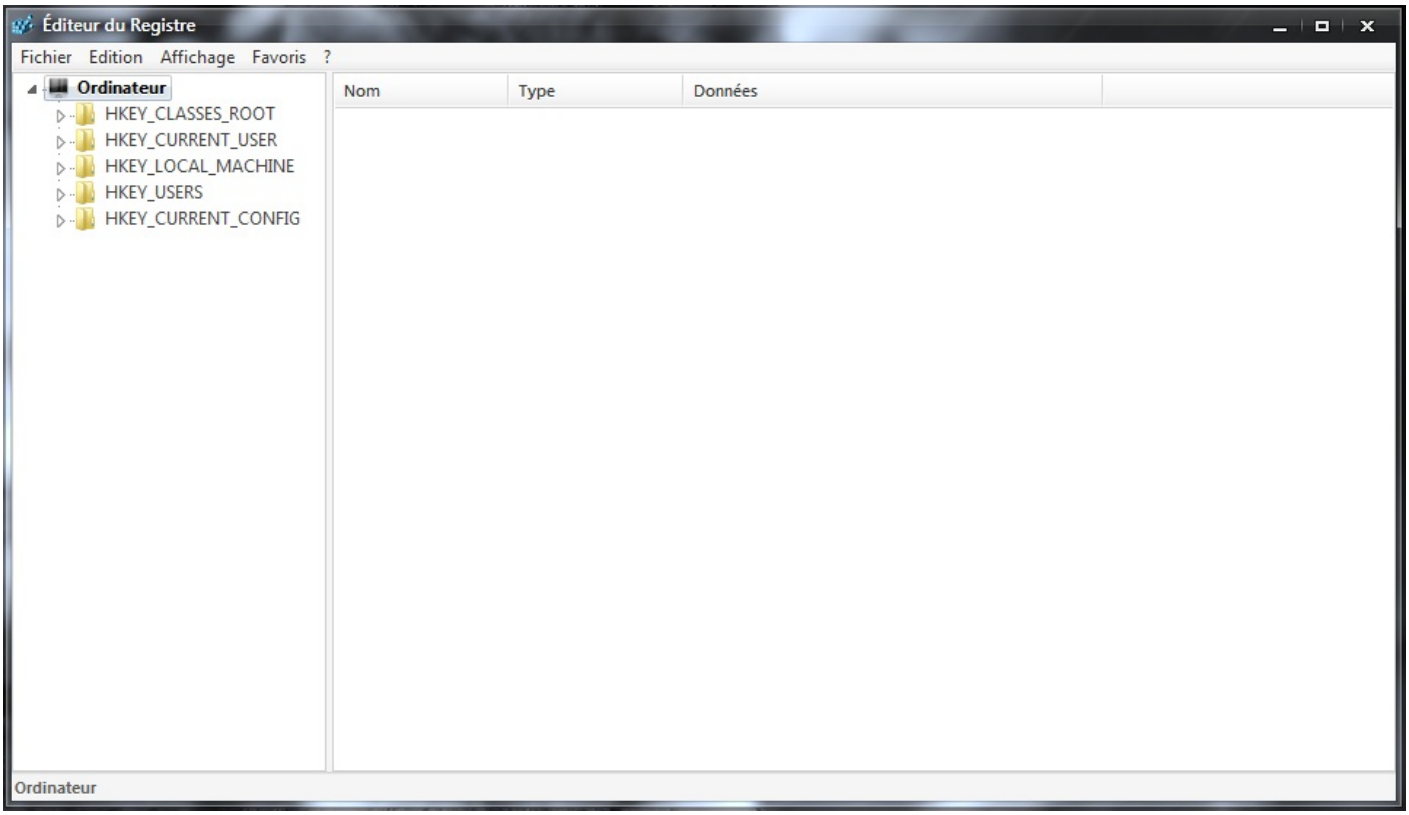
C'est quoi ça, le registre ?

Le registre, ou plutôt base de registre est en fait une base de données utilisée par Windows pour stocker des quantités monstres d'informations sur la configuration. C'est dans le registre que tous vos paramètres Windows sont stockés, il faut donc faire très attention lorsqu'on le manipule.

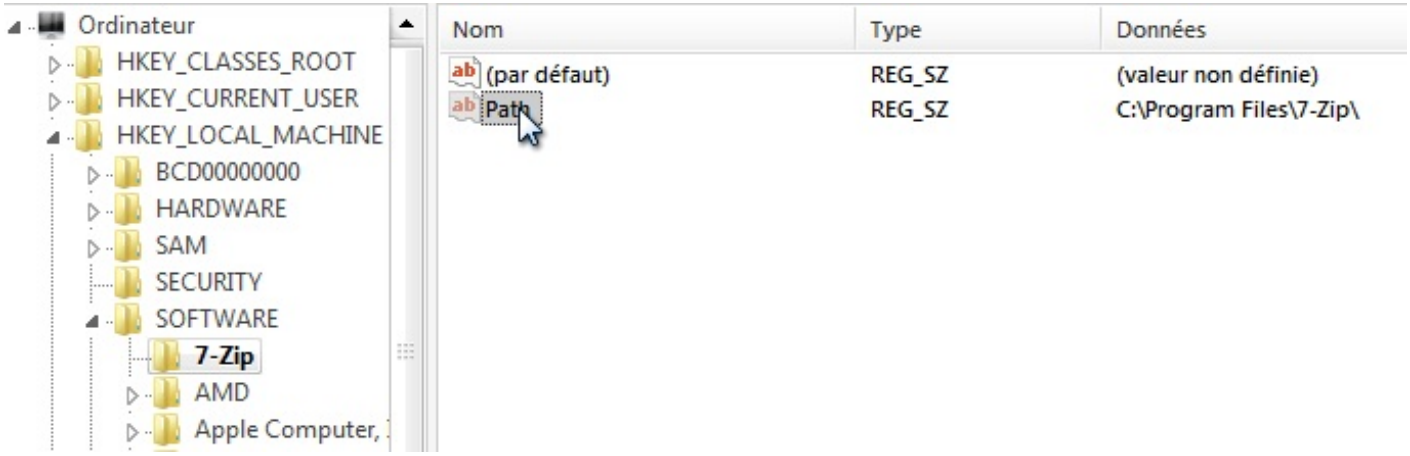
Nous allons nous en servir pour stocker nos informations de configuration.

Tout d'abord, pour accéder à votre éditeur de registre Windows, inscrivez "**regedit**" dans menu démarrer -> éxécuter.

Voici à quoi ressemble mon éditeur de registre :



Vous voyez qu'on se retrouve avec une arborescence semblable à des dossiers. Cependant ici les dossiers sont matérialisés par des "Clés" et les fichiers des "Valeurs".



La valeur "Path" présente dans la clé 7-Zip, les données qu'elle contient est le path de l'emplacement de ce programme dans mon ordinateur.

Donc pour notre programme nous rassemblerons toutes les valeurs de configurations dans la même clé pour organiser le tout.

Pour ce qui est des clés, la méthode sera la même qu'avec notre fichier .ini : un nom et sa valeur correspondante.

Pour enregistrer et récupérer des configurations dans le registre nous allons étudier deux méthodes :

1 - Les fonctions internes de VB

Des fonctions ont été pré implémentées dans VB.NET pour faire cela facilement, leur avantage : la rapidité et la facilité. Inconvénient : la clé dans laquelle les valeurs seront enregistré n'est pas sélectionnable, elles se situeront dans : HKEY_CURRENT_USER\Software\VB and VBA Program Settings\NomDuProgramme.
Vous pourrez ensuite choisir dans cette clé de créer des sous-clés mais vous ne pouvez pas changer de clé "principale".

Commencons par l'écriture :

Code : VB.NET

```
SaveSetting("Ressources", "Configuration", "Config1", "10")
```

Le premier argument permet de spécifier le nom du programme, la clé qui sera crée dans **VB and VBA Program Settings**. Le second paramètre permet de spécifier la section, ici j'utilise "Configuration", puis viens le nom de la valeur puis son contenu.

Ce qui nous créera la clé
HKEY_CURRENT_USER\Software\VB and VBA Program Settings\Ressources\Configuration

Avec à l'intérieur la valeur
Config1 = 10

Pour la récupérer :

Code : VB.NET

```
Valeur = GetSetting("Ressources", "Configuration", "Config1")
```

Les arguments correspondent à la fonction précédente.
Avec possibilité de spécifier un 4eme argument en valeur par défaut.

La suppression de clé :

Code : VB.NET

```
DeleteSetting("Ressources")
```

Vous pouvez spécifier en paramètre optionnels le nom de la section à supprimer et la valeur si vous ne voulez supprimer qu'une seule valeur de configuration.

2 - En utilisant les API

Une API (Application Programming Interface) est un rassemblement de fonctions ou méthodes permettant de réaliser un certain type de travail.
Ici, cette API va nous permettre de travailler sur le registre.

Passons à la seconde méthode avec plus de possibilités. Nous allons pouvoir, entre autres, spécifier dans quelle section écrire. Au sommet de l'arborescence nous avons le choix entre 2 sections : HKEY_LOCAL_MACHINE et HKEY_CURRENT_USER. Local machine contient tout ce qui est relatif à votre ordinateur, tandis que current user contient uniquement des données utilisables par l'utilisateur actuel.

Après tout dépend l'utilité qu'aura votre programme.

Nous allons travailler dans le namespace Microsoft.Win32, vous pouvez effectuer un `Imports Microsoft.Win32` pour éviter des écritures superflues.

Commençons par créer un objet Clé dans notre programme grâce à :

Code : VB.NET

```
Dim Cle As Microsoft.Win32.RegistryKey
```

Il faut la placer ensuite dans la section dans laquelle nous voulons travailler, pour moi ce sera local machine.

Code : VB.NET

```
Cle = Microsoft.Win32.Registry.LocalMachine
```

Registry contenant la liste des clés disponibles à la racine (current_user ...)

Maintenant on peut :

- Créer une sous-clé
- Ouvrir une sous-clé
- Ecrire ou lire une valeur
- Effacer une valeur

Je vous conseille de créer au minimum une sous clé relative à votre programme pour hiérarchiser le tout.

Ce qui me donne en code pour me placer et créer HKEY_LOCAL_MACHINE\App_Ressources et y créer une Valeur1 qui est égale à 1 :

Code : VB.NET

```
Dim Cle As Microsoft.Win32.RegistryKey = Nothing
        Cle = Microsoft.Win32.Registry.LocalMachine
        Cle.CreateSubKey("App_Ressources").SetValue("Valeur1", "1")
```

On résume : création d'une variable clé que j'initialise à nothing (pour que le code soit un peu plus clair).
Ensuite j'attribue la clé HKEY_LOCAL_MACHINE (qui est une clé principale) a ma variable.
Si vous avez donc suivi ma variable représente le "dossier" HKEY_LOCAL_MACHINE.
De ce point, je crée une sous clé (un répertoire) et j'y insère une valeur (un fichier).

Je trouve beaucoup plus simple de se représenter la base de registre sous cette arborescence de dossiers.

En fait vous naviguez simplement au millieu de dossiers.

Récapitulatif

Je récapitule les fonctions (a utiliser sur un objet de type **RegistryKey**) :

Pour créer une clé (un répertoire) :

Code : VB.NET

```
CreateSubKey("App_Ressources")
```

L'argument représente le nom de la clé à créer

Pour se déplacer dans une clé (un répertoire):

Code : VB.NET

```
OpenSubKey("App_Ressources")
```

Où l'argument est le nom de la clé où se déplacer.

Pour créer une valeur (un fichier) :

Code : VB.NET

```
SetValue("Valeur1", "2")
```

Où le premier argument est le nom de la valeur et le second ... Sa valeur =)

Récupérer une valeur (un fichier) :

Code : VB.NET

```
GetValue("Valeur1")
```

Où l'argument est le nom de la valeur a retrouver, renvoie Nothing si la value n'existe pas.

Pour conclure ce chapitre, je tien à dire que même s'il est relativement court (les ressources ne sont vraiment pas difficiles à utiliser, inutile de s'y attarder), il n'est pas inintéressant.

Les ressources vont êtres très utiles pour stocker les images nécessaire au design de vos application, les chaines de caractères ou valeurs pour être facilement modifiables (même avec 50 fenêtre, les configurations sont réunies au même endroit) et autres petits sons de bienvenue.

Il est hors de question de stocker un film, une vidéo utile dans votre programme ou tout autre fichier réellement volumineux en ressource, le .exe et ses DLL augmenteront inutilement de taille, ce sera réellement désagréable d'utilisation pour l'utilisateur.

Sachez donc bien gérer vos ressources, y mettre les informations jugées necessaires, les ressources sont en effet très utiles mais elles peuvent vite devenir un gros inconvéniant.

La révision des premiers chapitres a été effectuée, je continue maintenant la rédaction.

Les parties prévisionnelles sont les suivantes :

- bases de données ;
- notions avancées ;
- d'autres annexes.

Elles contiendront entre autres la publication de son programme en format `.exe` et installateur, l'utilisation des ressources, l'utilisation d'expressions web pour les interfaces graphiques... Bref, pas mal de travail en perspective.

Bon courage, amis Zéros !