

Apprenez à programmer en VB .NET

Par Hankerspace

Ce PDF vous est offert par



Découvrez des métiers plein d'envies

http://www.fr.capgemini.com/carrieres/technology_services



www.openclassrooms.com

*Licence Creative Commons 6 2.0
Dernière mise à jour le 14/06/2013*

Sommaire

Sommaire	2
Partager	4
Apprenez à programmer en VB .NET	6
Partie 1 : La théorie et les bases	7
Historique et Visual Basic Express 2010	7
Historique, naissance du Visual Basic	7
D'où vient le Visual Basic ?	7
Le framework .NET	8
Notre outil : Visual Basic 2010 Express	9
L'environnement de développement	9
Installation de Visual Basic 2010 Express	10
Étape par étape	10
Découverte de l'interface	11
L'interface de VB 2010 Express	11
Premiers pas	16
Hello World !	17
Notre premier programme !	17
Objets, fonctions	20
Fonctions, arguments	21
Les variables	21
Les types	22
Les utiliser - la théorie	22
Les utiliser - la pratique	24
Modifications des variables et opérations sur les variables	28
Opérations sur une variable	28
Plus en profondeur	29
Différentes syntaxes	31
Les commentaires	32
Lire une valeur en console	33
Conditions et boucles conditionnelles	35
Les boucles conditionnelles	35
Aperçu des différentes boucles	35
Select	38
While	41
Do While	42
For	43
Mieux comprendre et utiliser les boucles	44
Opérateurs	44
Explication des boucles	45
And, or, not	45
TP : La calculatrice	47
Addition	48
Minicalculatrice	50
Jouer avec les mots, les dates	52
Les chaînes de caractères	53
Les dates, le temps	53
TP sur les heures	55
L'horloge	55
Les tableaux	59
Les dimensions	59
Autres manipulations avec les tableaux	62
Mini-TP : comptage dans un tableau	63
Exercice : tri	65
Les énumérations	67
Les fonctions	68
Créons notre première fonction !	69
Ajout d'arguments et de valeur de retour	70
Petits plus sur les fonctions	72
Petit exercice	73
Les inclassables	76
Les constantes	76
Les structures	76
Boucles supplémentaires	78
Les casts	79
Le type Object	81
Les MsgBox et InputBox	81
La MsgBox	82
InputBox	84
Partie 2 : Le côté visuel de VB	85
Découverte de l'interface graphique	85
Les nouveautés	85
Avantages par rapport à la console	85
Manipulation des premiers objets	85

Les paramètres de notre projet	86
Les propriétés	87
À quoi ça sert ?	88
Les utiliser	89
Les assigner et les récupérer côté VB	91
With	93
Les événements	95
Pourquoi ça encore !	95
Créer nos événements	95
Les mains dans le cambouis !	96
Mini-TP : calcul voyage	96
Les contrôles spécifiques	100
Checkbox, boutons radio	100
La pratique	101
Les combobox	104
MicroTP	105
Les timers	106
Créer son premier timer	107
TP : la banderole lumineuse	108
Les menus	111
Présentation des menus	111
La barre de menus	111
Les différents contrôles des menus	115
La barre de statut	117
Le menu contextuel	119
TP : navigateur web	121
Le cahier des charges	121
Les ébauches	121
Bien exploiter les événements	124
Le design	125
Fenêtres supplémentaires	128
Ajouter des fenêtres	128
Ouverture et fermeture	129
Notions de parent et d'enfant	130
Communication entre fenêtres	132
Les fichiers - partie 1/2	135
Introduction sur les fichiers	136
Le namespace IO	137
Notre premier fichier	138
Nos premières manipulations	139
Programme de base	139
Explications	141
Les fichiers - partie 2/2	146
Plus loin avec nos fichiers	147
La classe File	147
Les répertoires	149
Fonctions de modification	149
Fonctions d'exploration	150
Mini-TP : lister notre arborescence	151
Un fichier bien formaté	153
TP : ZBackup	155
Le cahier des charges	155
Correction	155
L'interface	161
Sauvegarde en fichier .ini	162
Sauvegarde	163
Récapitulatif du fichier ini	164
Pour aller plus loin	166
Partie 3 : La programmation orientée objet	167
Les concepts de la POO	168
Pourquoi changer ?	168
Mesdames, Messieurs, Sa Majesté POO	168
Les accessibilités	169
Les fichiers de classe	170
Notre première classe	172
Notre première classe	172
Des méthodes et des attributs	174
Les propriétés	174
Notre petit Mario	176
Concepts avancés	179
L'héritage	179
Les classes abstraites	181
Les événements	183
La surcharge	184
La surcharge d'opérateurs et les propriétés par défaut	187
Paramètres dans les propriétés	187
Les propriétés par défaut	188
Surcharge d'opérateurs	189
Les collections	190
Les bibliothèques de classes	192

Les créer	192
Les réutiliser dans un projet	193
La sauvegarde d'objets	194
La sérialisation, c'est quoi ?	195
La sérialisation binaire	196
La sérialisation XML	198
La sérialisation multiple	200
TP : ZBiblio, la bibliothèque de films	203
Le cahier des charges	204
La correction	204
Améliorations possibles	213
Partie 4 : Les bases de données	215
Introduction sur les bases de données	215
Qu'est-ce qu'une base de données ?	215
Les bases de données	216
Les SGBD	216
SGBD et SQL	216
VB .NET et SGBD	217
Lexique	217
SQL Server 2008 R2	218
Notre SGBD	218
Installation de SQL Server 2008 R2	219
Étape par étape	219
Découverte de l'interface	225
L'interface de SQL Server 2008 R2	225
Introduction au langage SQL	229
Rechercher des informations	229
La clause WHERE	229
La clause WHERE... IN	230
Clause WHERE... BETWEEN	230
La clause WHERE... LIKE	230
La clause ORDER BY	231
Ajouter des informations	231
La mise à jour d'informations	232
Supprimer des informations	232
Création et remplissage de la BDD	232
Création de notre base de données (BDD)	233
La création de la table	234
L'analyse	234
La création	235
Le remplissage de données	237
La communication VB .NET - BDD	240
ADO.NET	240
Le fonctionnement d'ADO.NET	240
Connexion à la BDD	241
Insertion ou modification	242
Lecture de données	244
Le DataSet à la loupe	246
Qu'est-ce ?	247
La lecture de données	247
Lecture plus poussée	249
L'ajout de données	250
L'utilisation graphique : le DataGridView	251
La découverte du DataGridView	252
Liaison avec le code VB .NET	252
Liaison via l'assistant	253
TP : ZBiblio V2	259
Cahier des charges	259
Correction : partie BDD	259
Correction : partie VB	260
Connexion	260
Récupération	261
Suppression d'une fiche	262
Modification et ajout	262
Conclusion	264
Partie 5 : La communication par le réseau	264
Introduction à la communication	265
La communication, pourquoi ?	265
Communication interne	265
Communication réseau	265
Les sockets	265
TCP : mode connecté	265
UDP : mode déconnecté	266
.NET remoting	266
WCF, Windows Communication Fundation	266
Communication via sockets	267
Client/serveur	268
La connexion	268
L'IP et le port	269
Le serveur	269

Le client	270
Le transfert de données	271
Mini-TP : demande d'heure	273
TCPListener/TCPClient	276
TCPListener	276
TCPClient	276
L'acceptation du serveur	276
La communication par flux	277
Les threads	278
Introduction	279
Notre premier thread	280
Suspend/Resume	281
Join	281
Abort	281
La synchronisation	282
La variable globale	282
Le SyncLock	282
SemaphoreSlim	283
Les Windows Forms et les threads	283
Les delegates	284
TP : ZChat	286
Cahier des charges	286
La correction	286
Le client	286
Le serveur	290
Conclusion	292
Partie 6 : Annexes	293
Gérer les erreurs	293
Découvrons le Try	293
Finally	293
Catch, throw	294
Les ressources	296
Qu'est-ce qu'une ressource ?	297
Ajoutons nos ressources	298
Récupérons-les maintenant	299
Le registre	300
1 - Les fonctions internes de VB	301
2 - En utilisant les API	301
Récapitulatif	302
Diffuser mon application	303
Définition de l'assembly	304
Debug et Release	304
La publication	305



Apprenez à programmer en VB .NET

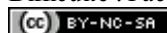
 Le tutoriel que vous êtes en train de lire est en **bêta-test**. Son auteur souhaite que vous lui fassiez part de vos commentaires pour l'aider à l'améliorer avant sa publication officielle. Notez que le contenu n'a pas été validé par l'équipe éditoriale du Site du Zéro.



Par Hankerspace

Mise à jour : 14/06/2013

Difficulté : Facile  Durée d'étude : 1 mois



Vous trouvez le C et le C++ trop compliqués mais aimeriez concevoir des programmes fonctionnels, ergonomiques et facilement accessibles ? Le **Visual Basic** est fait pour vous ! Il s'agit d'un langage extrêmement 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 cours va vous initier aux bases du Visual Basic, ce qui est tout de même normal quand on s'adresse à des Zéros. Aucun prérequis n'est demandé : il n'est pas nécessaire de connaître le moindre langage, tout vous sera expliqué.

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



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 de 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. 😊

Partie 1 : La théorie et les bases

Partie consacrée à l'apprentissage rapide et précis des concepts de base qui vont nous permettre de 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 fondés 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 pour poursuivre la lecture de ce tutoriel sans embûches.

Historique, naissance du Visual Basic

D'où vient le Visual Basic ?

Nous allons donc commencer 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 des articles de Wikipédia sur notre sujet et vous l'ai résumé.

Le BASIC

BASIC est un acronyme pour *Beginner's All-purpose Symbolic Instruction Code*. Le BASIC a été conçu 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 ! ».

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 vous permettra 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 graphiques et visuels des 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 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 est également possible d'exécuter un programme créé en VB sous d'autres plates-formes que Windows grâce à Mono.



M'sieur... c'est quoi 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 F#, le J#, etc.

Le *framework .NET*

La plate-forme .NET fournit un ensemble de fonctionnalités qui facilitent le développement de tous types d'applications :

- Les applications Windows classiques ;
- Les applications web ;
- Les services Windows ;
- Les services web.

En Visual Basic, toutes ces applications sont réalisables grâce au **framework .NET**.

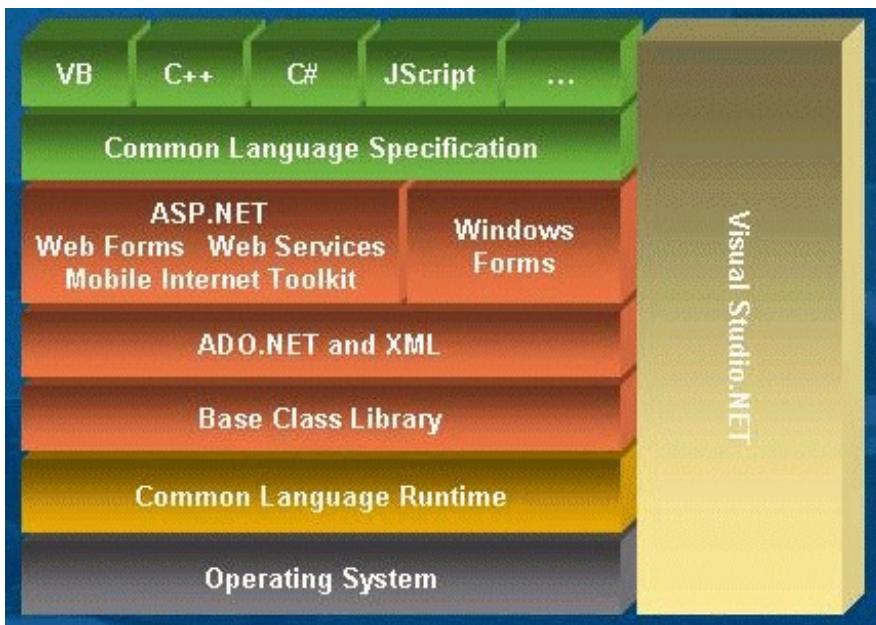


« .NET » se prononce « *dote nette* » (en anglais).

Du développement de l'application jusqu'à son exécution, le *framework .NET* prend en charge l'intégralité de la vie de cette application.

Ce framework doit être hébergé sur le système d'exploitation avec lequel il doit interagir. Pas d'inquiétude, nous allons nous en charger.

Le *framework .NET* (voir figure suivante) a été créé par **Microsoft** : le premier système d'exploitation permettant de le posséder est bien sûr **Windows**, mais d'autres versions sont disponibles, permettant l'adaptation de la plate-forme .NET à des systèmes d'exploitation tel que **Linux** ou **Unix**.



Vue d'ensemble du framework .NET

Cela doit vous sembler bien compliqué, mais retenez bien son utilité première : nous mâcher le travail. Par exemple, si vous souhaitez lire et écrire dans un fichier (ce que nous verrons plus tard), le développement depuis zéro d'un programme capable d'effectuer cette tâche est longue et fastidieuse. Il va falloir envisager toutes les possibilités d'erreurs, trouver un moyen d'interagir avec votre disque dur, etc. Cela s'appelle de la programmation **bas niveau** (proche du matériel informatique en lui-même).

Cependant, des personnes ont déjà codé les éléments permettant d'effectuer ces actions. Tout cela a été intégré au framework .NET et installé sur vos machines. Vous allez donc pouvoir réutiliser leur travail pour vous simplifier la vie et diminuer le risque d'erreurs. Cela s'appelle de la programmation **haut niveau** (éloigné du matériel).

Notre outil : Visual Basic 2010 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 bibliothèques) 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 inexistant 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.

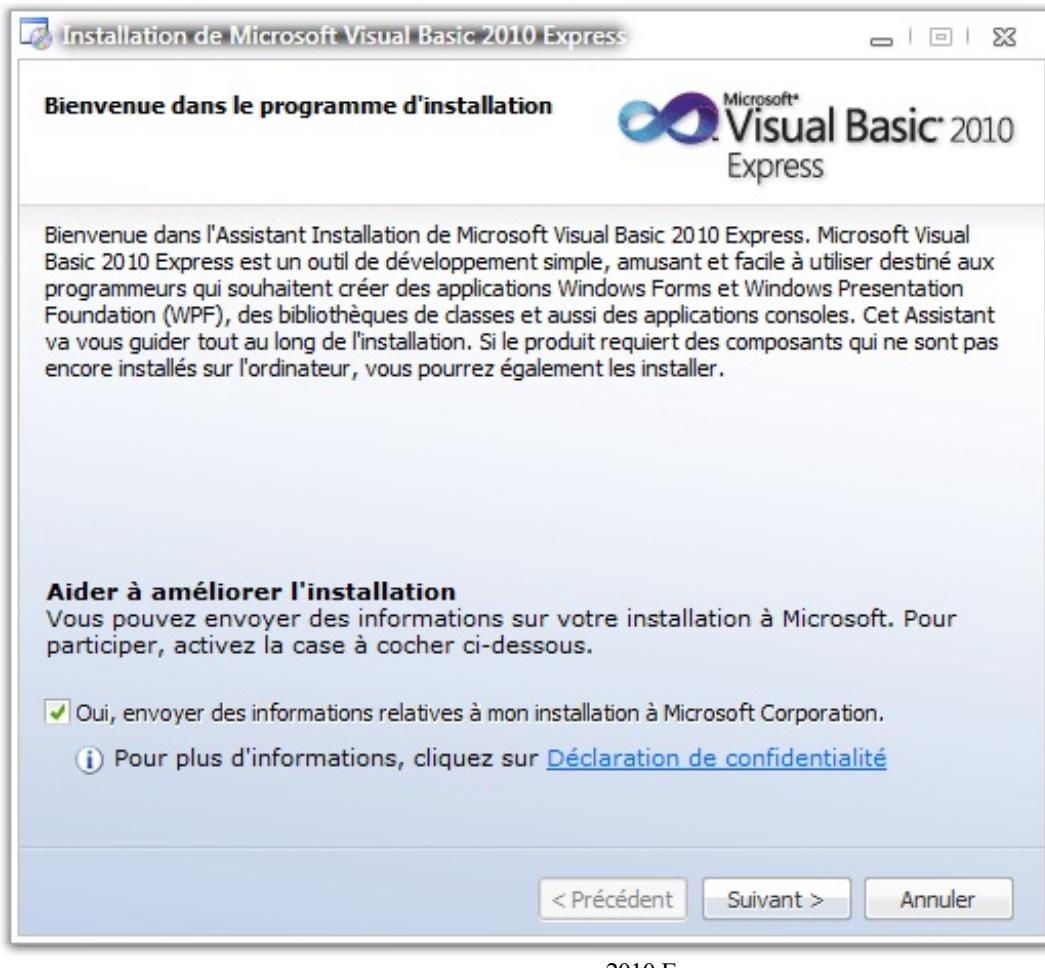
Installation de Visual Basic 2010 Express

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. 

Vous lancez donc le programme et arrivez à la première page (figure suivante).

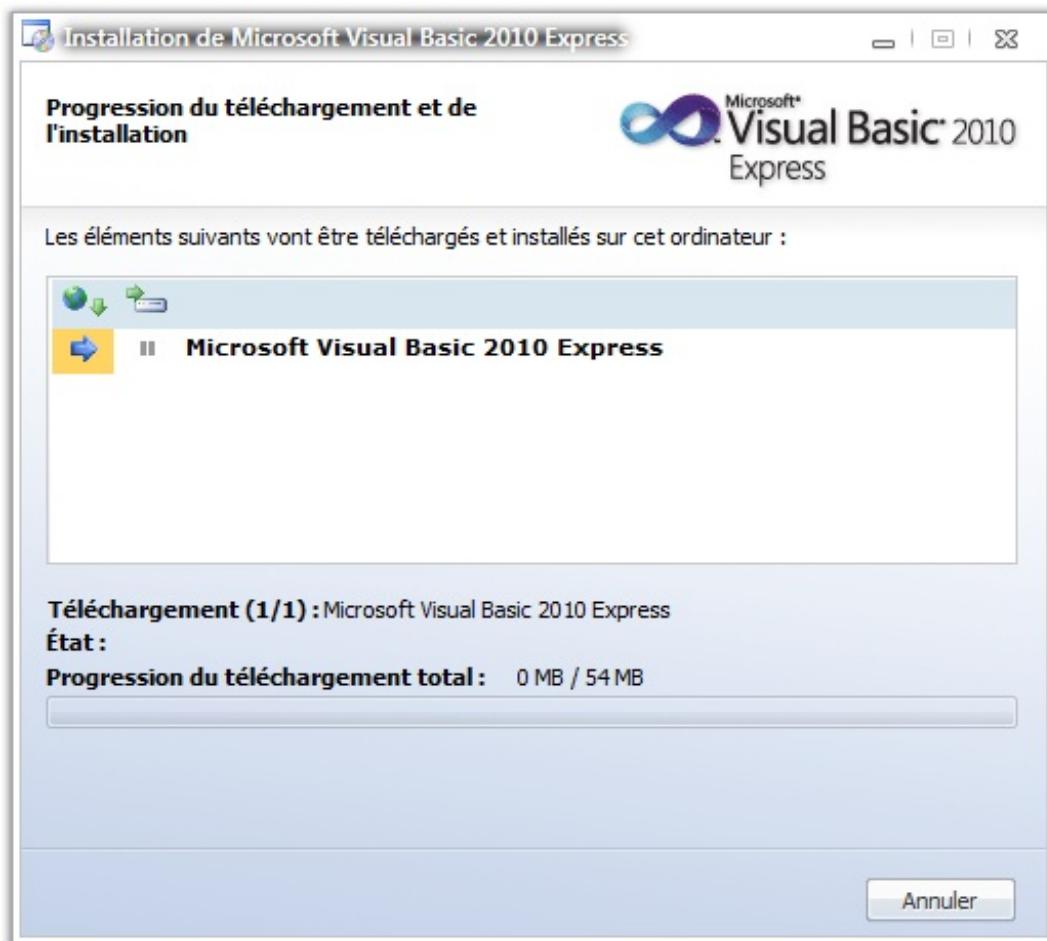


Installation de Visual Basic

2010 Express

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

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



Visual Basic 2010 Express est

en cours de téléchargement

Il ne vous reste plus qu'à attendre la fin du téléchargement, suivi de l'installation. Une fois cela terminé, vous voilà avec Visual Basic 2010 Express installé !

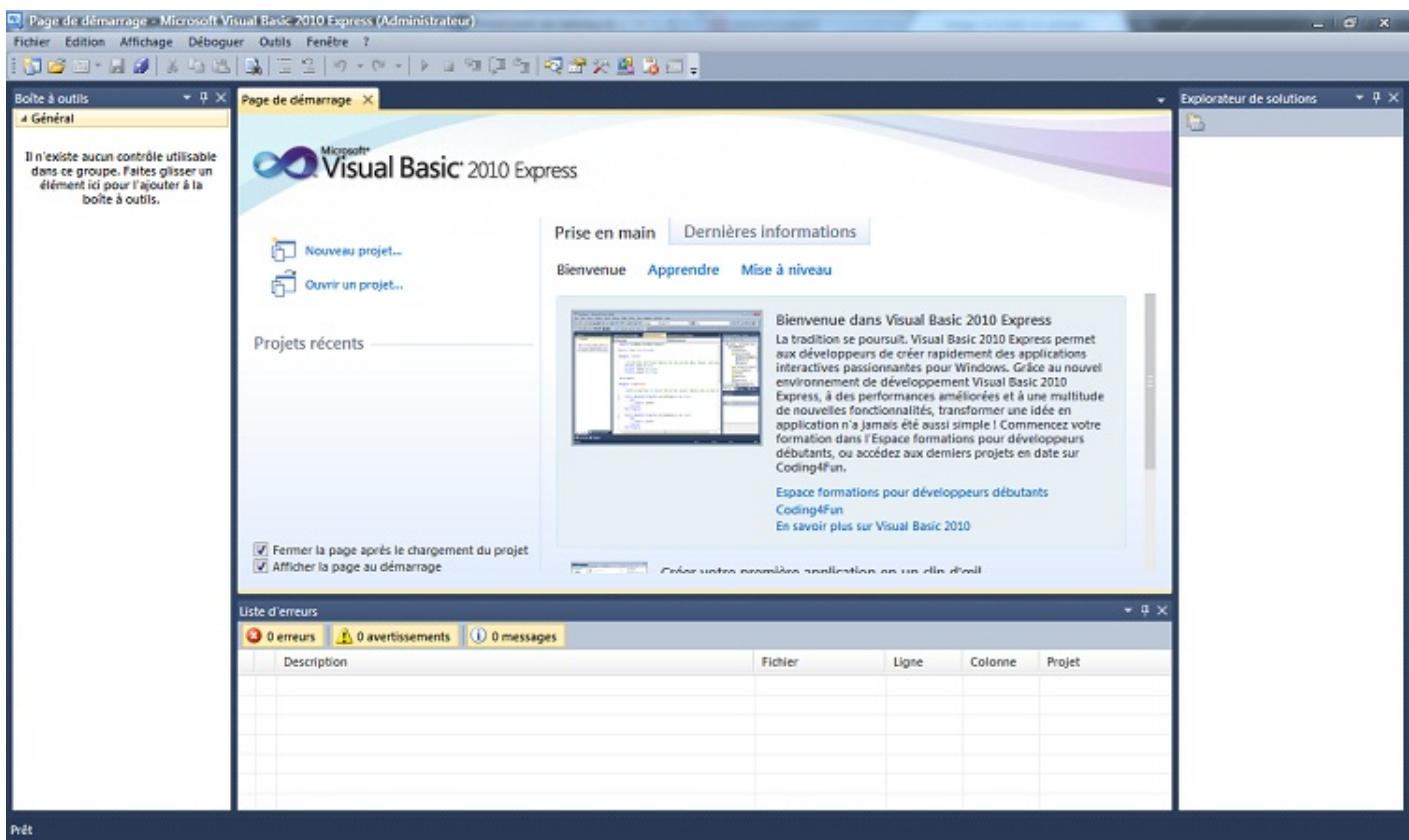
Découverte de l'interface

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 (voir figure suivante). Vous pouvez la parcourir, elle contient des informations utiles aux développeurs (vous) et conservera l'historique de vos projets récents.



Page de démarrage

Barre d'outils

La barre d'outils (voir figure suivante) vous sera indispensable afin de travailler avec une parfaite ergonomie. Je vais récapituler les boutons présents (de gauche à droite), actifs ou non durant vos travaux.



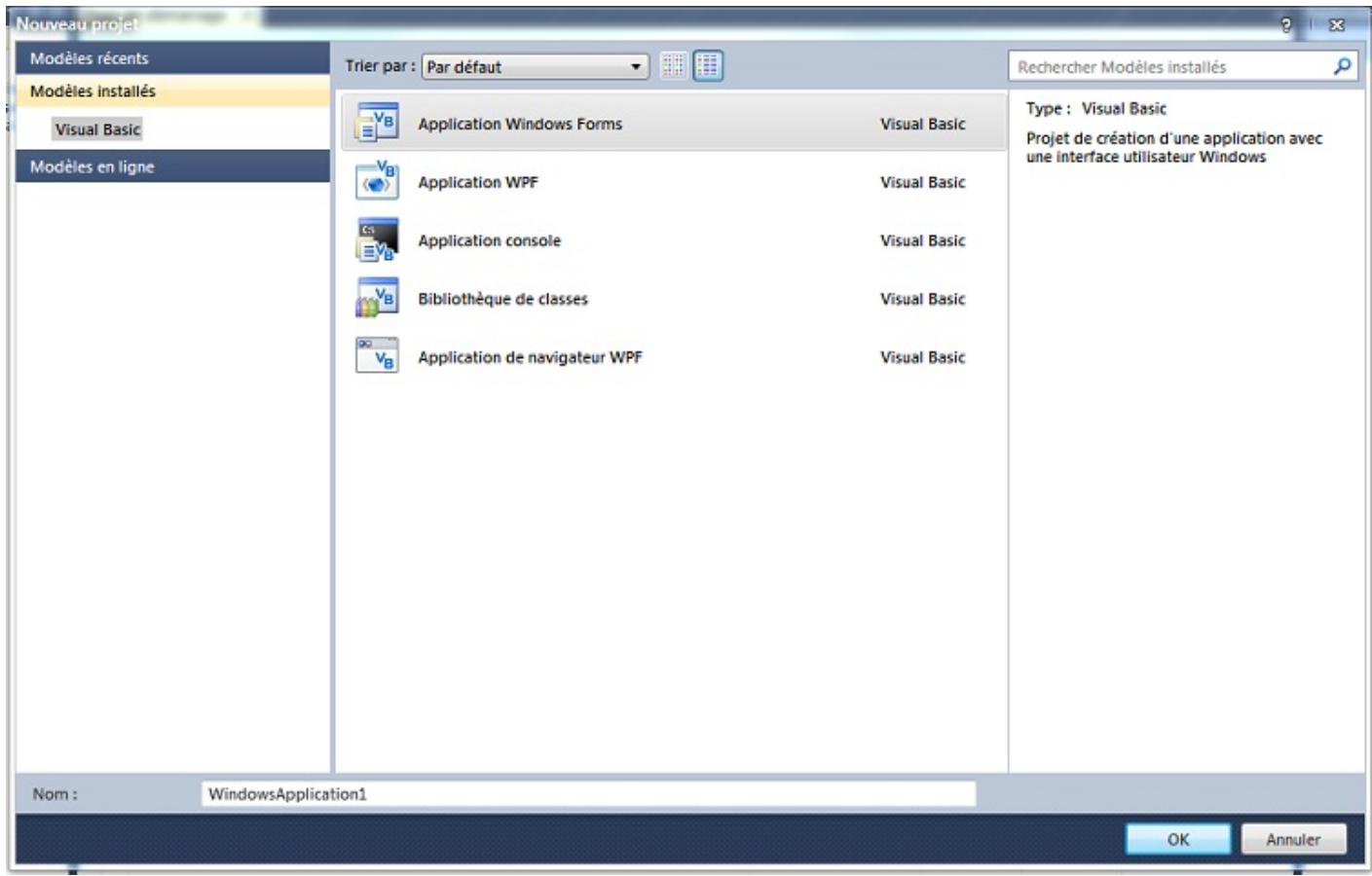
- Nouveau projet : crée un projet.
- Ouvrir un fichier : à utiliser pour ouvrir un projet existant ou une page simple.
- Ajouter un nouvel élément : disponible quand un projet est ouvert ; il permet d'ajouter des feuilles au projet.
- Enregistrer : raccourci CTRL + S.
- Enregistrer tout : raccourci CTRL + 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 des erreurs.

Toutes ces commandes ne seront pas forcément utiles ; au besoin, n'hésitez pas à revenir voir cette liste.

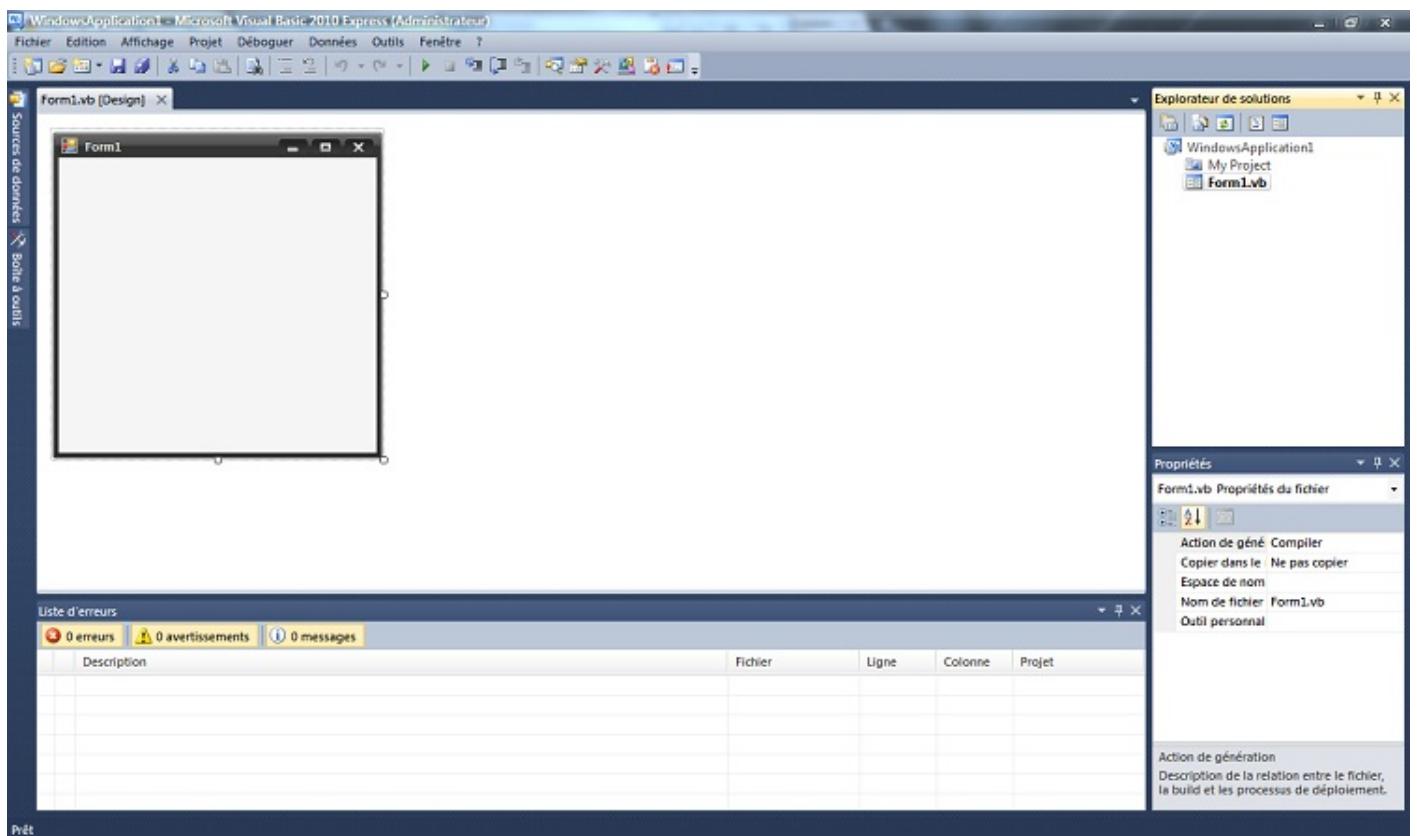
Nouveau projet

Je vous invite, seulement pour appréhender l'interface, à créer un projet Windows Forms (voir figure suivante). 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 (voir figure suivante).



Projet « Application Windows Forms »

Nous allons tout voir en détail.

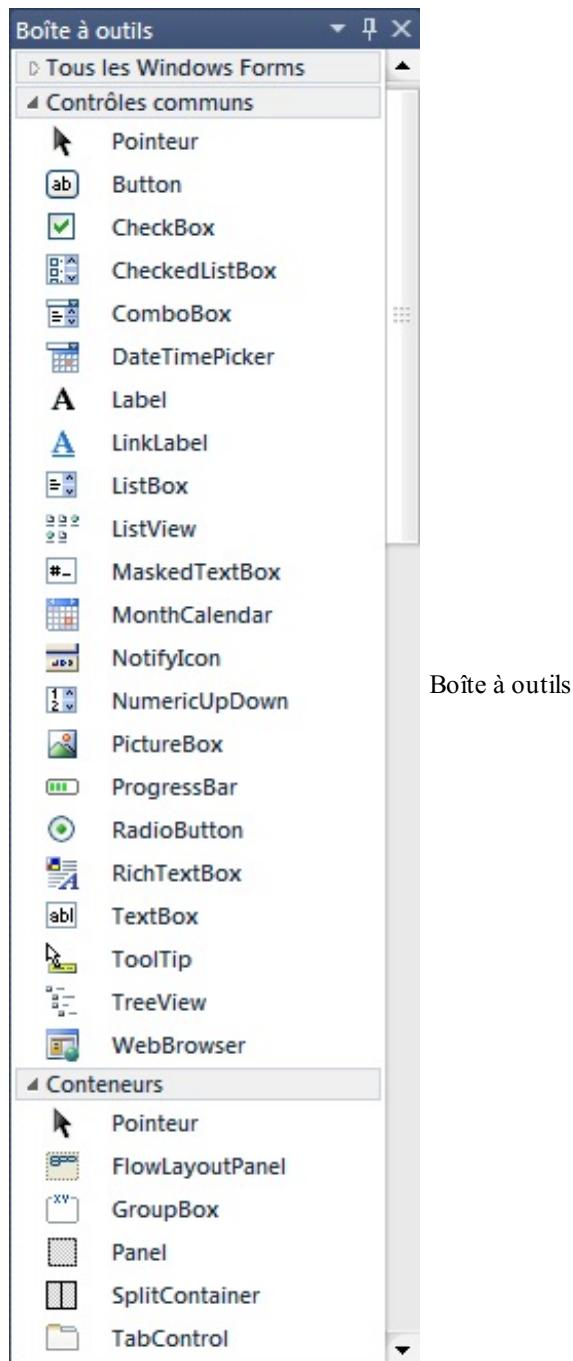
Espace de travail

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



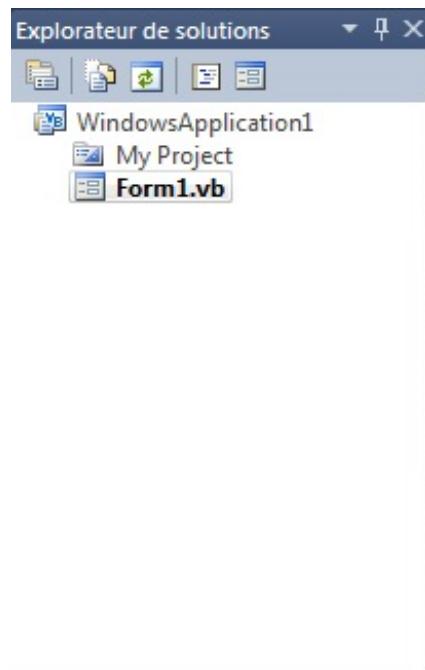
Boîte à outils

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 (voir figure suivante) 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

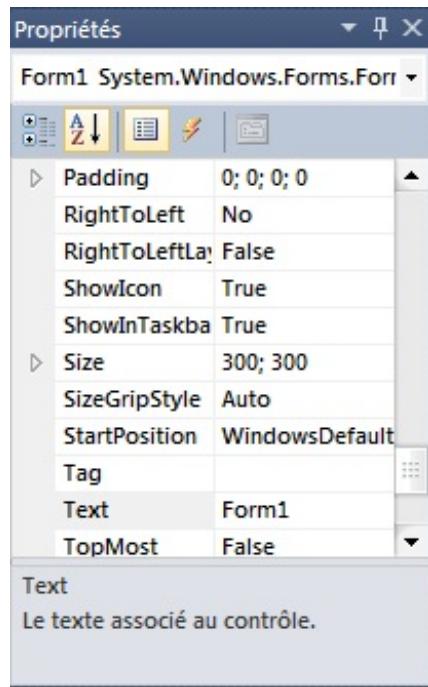
De l'autre côté de notre écran, nous remarquons la fenêtre de solutions (voir figure suivante) : elle récapitule l'arborescence de notre projet. Elle servira également à gérer les bases de données, mais plus tard. 😊



Fenêtre de solutions

Fenêtre des propriétés

Autre partie essentielle : la fenêtre des propriétés (voir figure suivante) 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.



Fenêtre des propriétés

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.

- Le Visual Basic .NET est une amélioration du langage BASIC qui ajoute une partie de gestion de l'interface visuelle associée au framework .NET de Microsoft.
- On télécharge Visual Basic Express edition sur le site de Microsoft, cette version est gratuite et sans limitation de temps.

Premiers pas

Après cette petite découverte de notre environnement de développement, nous allons immédiatement entrer dans le monde fabuleux de la programmation !

Hello World !



Je tiens à m'excuser pour les termes que j'utiliserais 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.

Créez un nouveau projet (CTRL + N) et cliquez sur Application console. De nombreux mots de langue étrangère apparaissent. Pas de panique, je vais vous tout 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 dans votre feuille de code sont indispensables ! Si vous les supprimez, l'application ne se lancera pas. C'est le code minimal que l'IDE (Visual Studio) génère lorsque l'on crée un projet de type console.

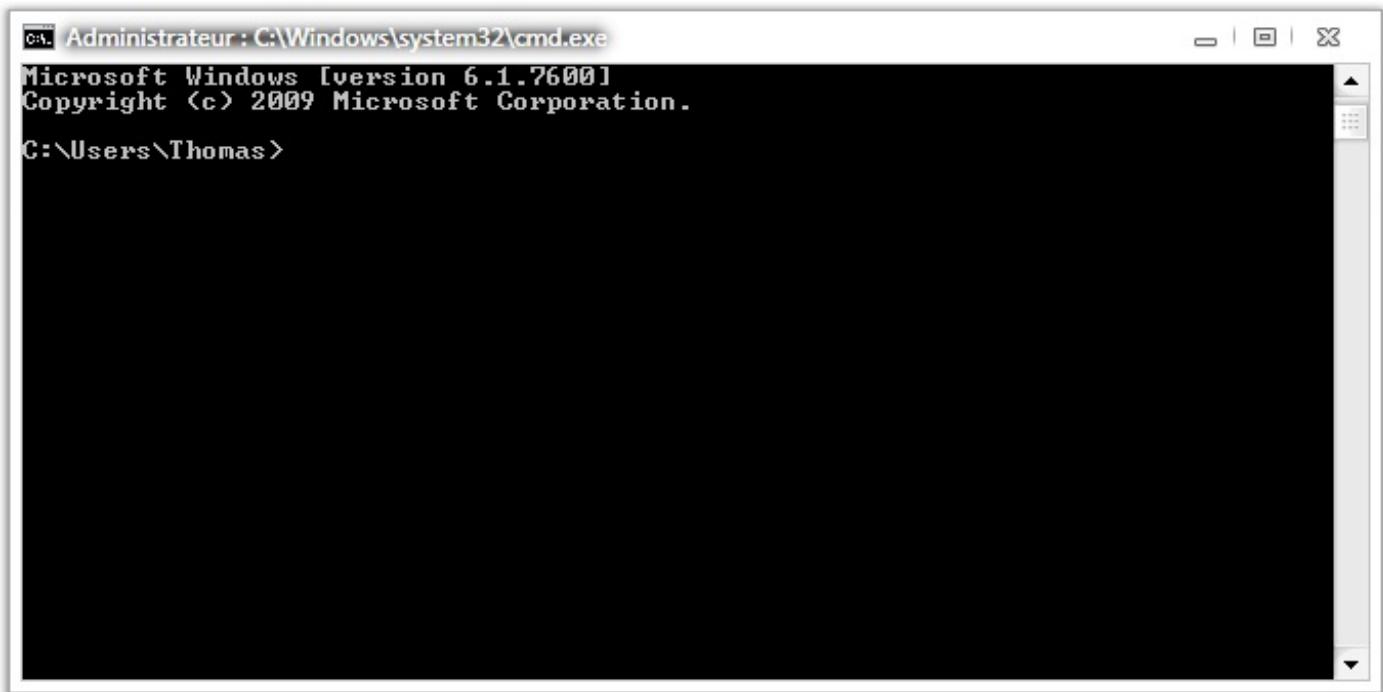
Chaque grosse partie, telle qu'une *fonction*, un *module*, un *sub*, voire une *boucle conditionnelle* (nous allons revenir sur ces termes), aura une balise de début : ici, **Module** Module1 et **Sub** Main (), et une balise de fin : **End Module** et **End Sub**. Module1 est le nom du module, que vous pouvez modifier si l'envie vous en 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 la partie appelée Main () et de sauter dedans. 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 ! Regardez la figure suivante.



Voici à quoi ressemble une console

Voilà 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 ! 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.

Hello World !

Code : VB.NET

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

Donc, pour ceux qui ont quelque peu suivi, où va-t-on placer cette ligne ?



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

Dans le `Main()` ! 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 !

Déroulement du programme

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 Sub**). 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 ?

Code : VB.NET

```
Module Module1
    Sub Main()
        Console.WriteLine("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 :

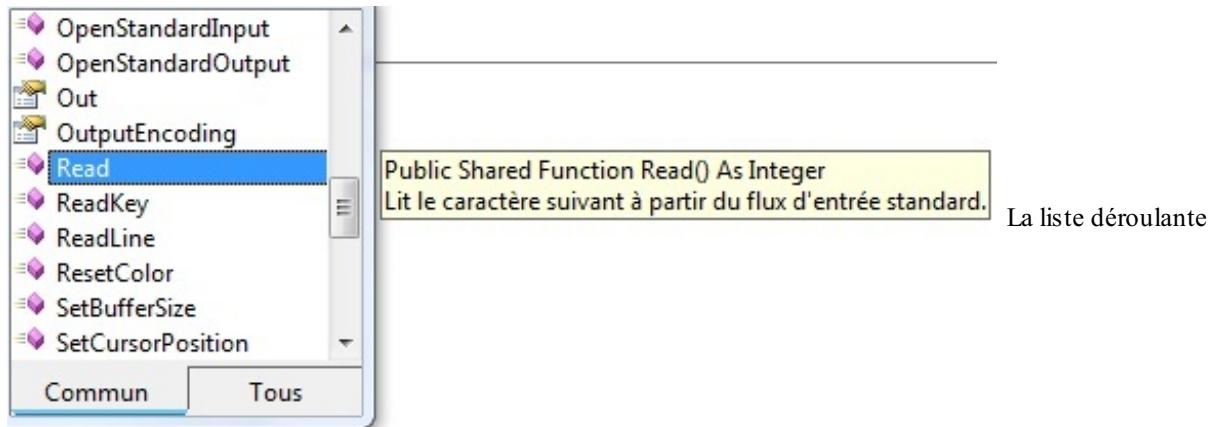
Code : Console

```
Hello World !
```

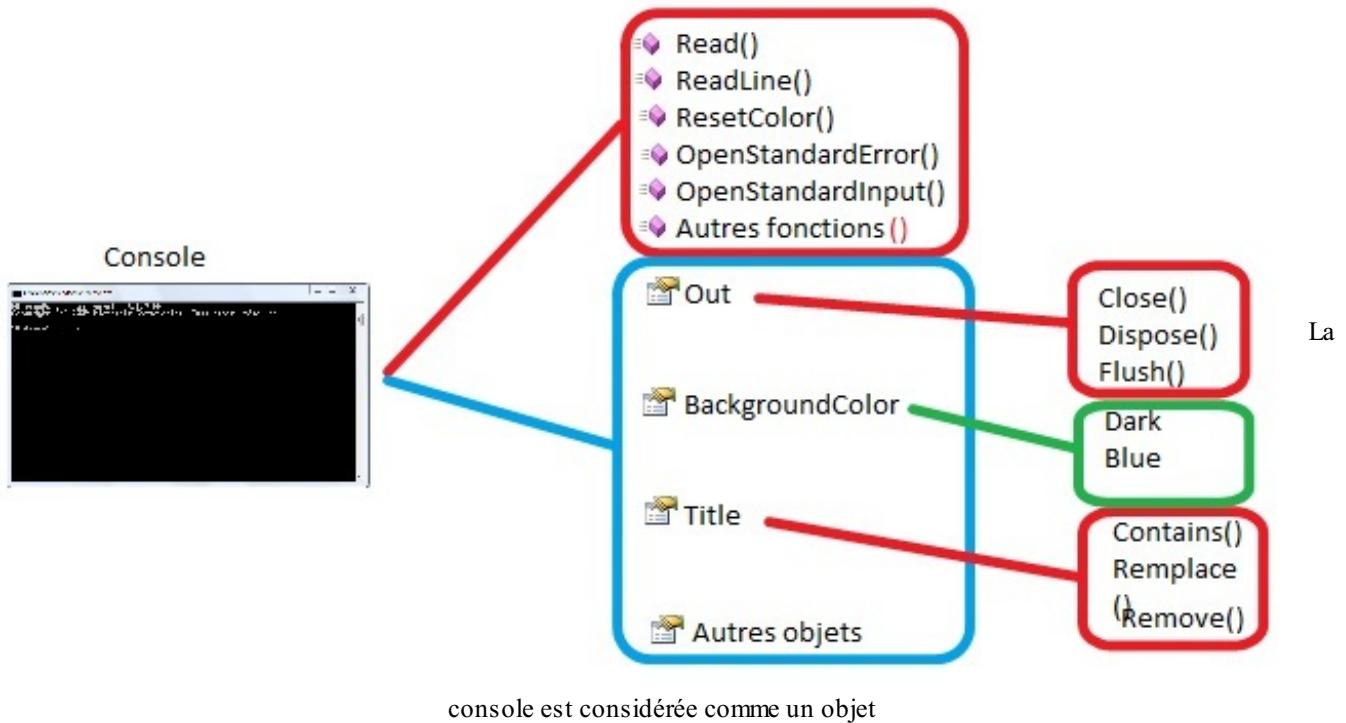
Victoire, notre « Hello World ! » reste affiché ! Si l'on presse la touche Entrée, la console se ferme : nous avons atteint nos objectifs !

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 (voir figure suivante). Dans cette partie, je vais vous expliquer l'utilité de cette liste.



Nous avons donc notre console au début du schéma de la figure suivante. Sous VB, la console est considérée comme un objet. Cet objet possède des fonctions et d'autres objets. 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 (la séquence 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 ces concepts 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 ; 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 alors (`frequency as integer, duration as integer`).



Hou là là, ça devient pas cool, ça !

On va y aller 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 ci-avant. 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 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 !

- Une fonction permet d'effectuer des traitements. Elle peut contenir des arguments (ou paramètres).
- Dans un projet de type console, la fonction `Main()` est celle appelée lors du démarrage du programme.
- Chaque ligne de programme est une instruction.

Les variables

Comme son nom l'indique, une variable... varie. On peut y stocker pratiquement tout ce qu'on veut, comme par exemple des nombres, des phrases, des tableaux, etc. C'est pour cette raison que les variables sont *omniprésentes* dans les programmes. Prenons comme exemple votre navigateur web préféré : 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

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 

Au tableau suivant, vous trouverez un récapitulatif des types que nous allons utiliser.

Tableau des types de variables 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 entiers (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 il est grand (16, 32, 64), 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 les variables ?

Telle est la question à laquelle nous allons répondre.

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

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

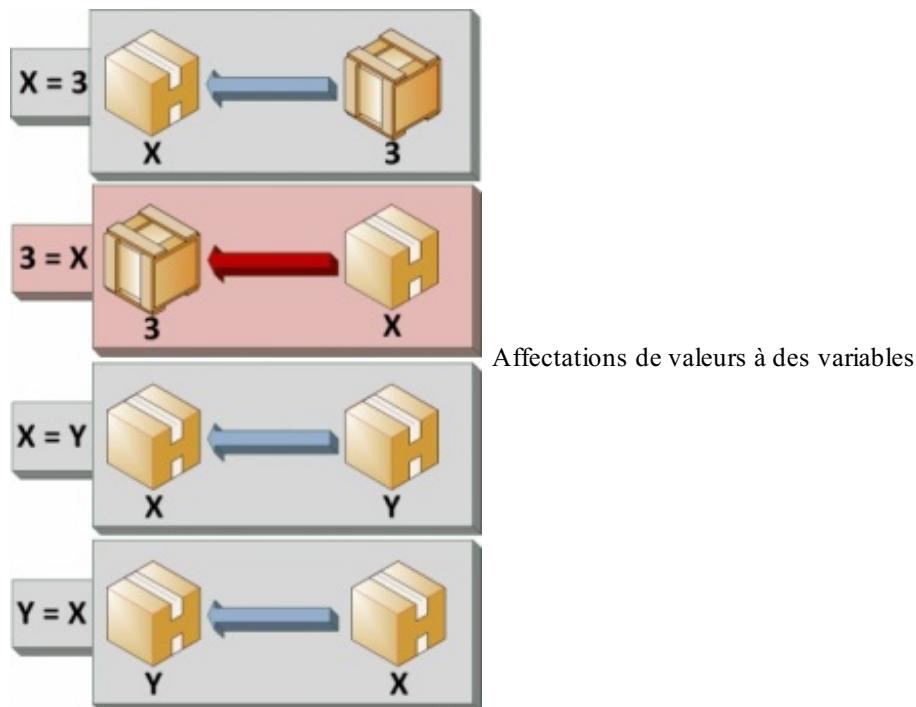
Excusez-moi de vous avoir attaqués par derrière comme je l'ai fait, mais c'était dans le but de vous faire observer que l'attribution des variables est en de nombreux points similaire à notre vieil ami *x* en maths. 

Comme pour attribuer une valeur à une variable, on place un « = » entre deux éléments.

Le sens

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

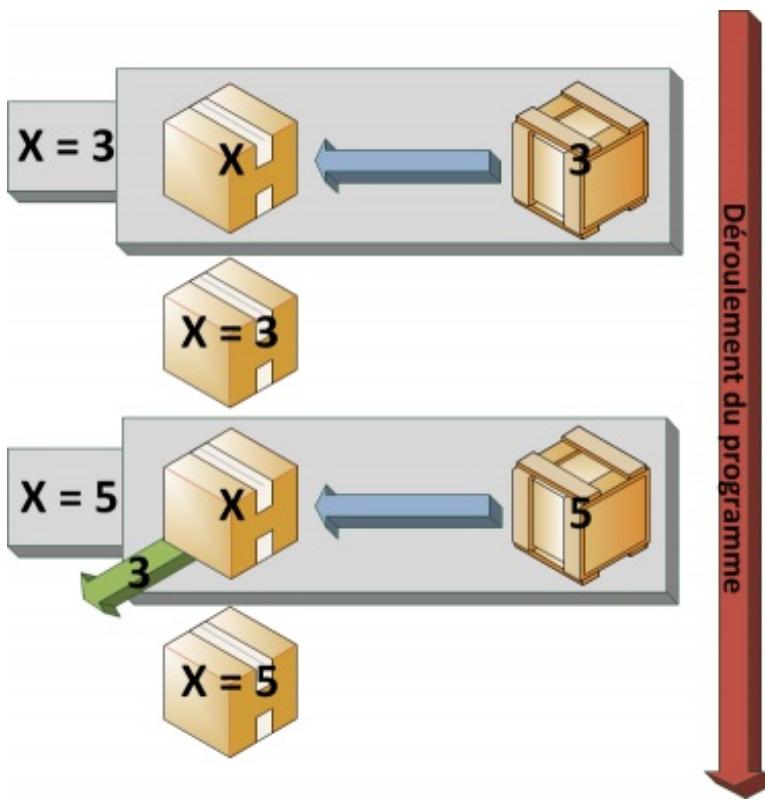
Regardez la figure suivante.



1. On entre le chiffre 3 dans la variable appelée X, pas de problème ;
2. Ensuite, on souhaite mettre X dans 3 ! Aïe, cela ne va pas fonctionner ! Si vous écrivez quelque chose de ce 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 !
3. Ensuite, on met la variable Y dans la variable X ;
4. 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, comme schématisé à la figure suivante.



Attention, une valeur peut en écraser une autre

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 ? 23 !

Les utiliser - la pratique

Cette petite partie de lecture vous a ennuyés ? 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	entier

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.WriteLine("Salut")
        Console.ReadLine()
    End Sub
End Module
```

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

Code : VB.NET

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



Maintenant, où placer ces instructions ?

C'est la question fatidique ! Si 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.WriteLine`. Pensez-vous être capables de faire en sorte d'afficher la valeur de la variable dans la console ?

Code : VB.NET

```
Module Module1
Sub Main()
    Dim MaVariable As Integer
    MaVariable = 5
    Console.WriteLine(MaVariable)
    Console.ReadLine()
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 apprenez !

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*. 🎉

- Une variable peut contenir une valeur.
- Les variables et donc les valeurs qu'elles contiennent peuvent être de différents types.
- Des guillemets : « " » délimitent une variable de type chaîne de caractères.
- Le type `String` contient les chaînes de caractères, `Integer` les entiers, `Double` les nombres à virgule, et `Date` les dates et heures.

Modifications des variables et opérations sur les variables

Nous voici en possession de notre nouvel outil : les variables.

Le monde des variables étant gigantesque, nous allons encore les étudier pendant un chapitre.

Maintenant que nous savons comment les déclarer et les utiliser, nous allons apprendre à les modifier de différentes façons. Viendront ensuite les opérations mathématiques, la concaténation, les commentaires et, pour finir, la lecture de valeurs depuis la console.

Je ne vous en dis pas plus, il faut maintenant s'y mettre.

Opérations sur une variable

Nous allons à présent apprendre à modifier des variables, et effectuer des opérations avec elles.

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.

Reprendons 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.WriteLine(MaVariable)
        Console.ReadLine()
    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
```

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 nous 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.

Les différentes 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 par 5, comment auriez-vous procédé ?

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

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



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`.

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.WriteLine("test")
Console.WriteLine("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 ! 😊

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

        Console.ReadLine()
    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.WriteLine(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 fait ; 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.WriteLine("9 * 8 = " & MaVariable3)
```



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

Bonne question. C'est ce qu'on appelle la **concaténation**, elle permet d'assembler deux éléments en un ; ici, par exemple, j'ai assemblé la chaîne de caractères "9 * 8 = " et le contenu de la variable, ce qui aura pour effet de m'afficher directement 9 * 8 = 72 (je parle d'assembler deux éléments en un, car en faisant cela on assemble 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.WriteLine("9 * 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.WriteLine("9 * 8 = " & MaVariable * MaVariable2)

        Console.ReadLine()
    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.WriteLine("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 et qui porte 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 :

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.

- Une variable peut stocker différentes valeurs.
- Les variables ont un type leur permettant de stocker des informations spécifiques : numérique, chaîne de caractères, etc.
- Le modulo permet de retourner le reste d'une division entière.
- Les commentaires sont des lignes de code non exécutées, uniquement destinées au programmeur, elles sont précédées par une apostrophe : « ' ».
- L'instruction Console.WriteLine permet d'afficher un message à la console.

Conditions et boucles conditionnelles

Une boucle conditionnelle est quelque chose de fort utile et courant en programmation. Cela permet d'effectuer une action si, et seulement si, une condition est vérifiée. Par exemple vous voulez que votre programme dise « bonne nuit » s'il est entre 22 h et 6 h. Eh bien, c'est précisément dans ce cas de figure que les boucles conditionnelles trouvent leur utilité.

Les boucles conditionnelles

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.

Aperçu des différentes boucles

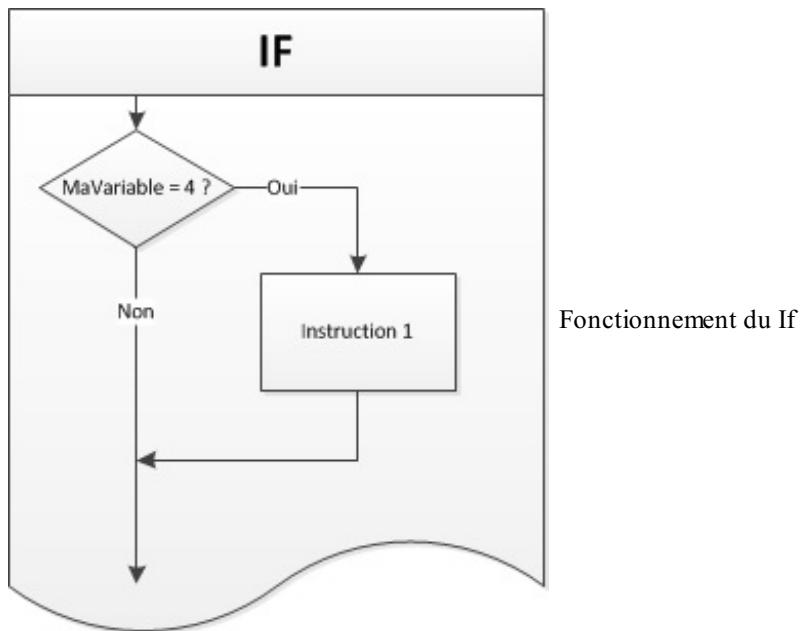
« If », mot anglais traduisible par « si »

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. Regardez la figure suivante.

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.

Si MaVariable est égale à 10, il met MaVariable à 5.
Exactement !

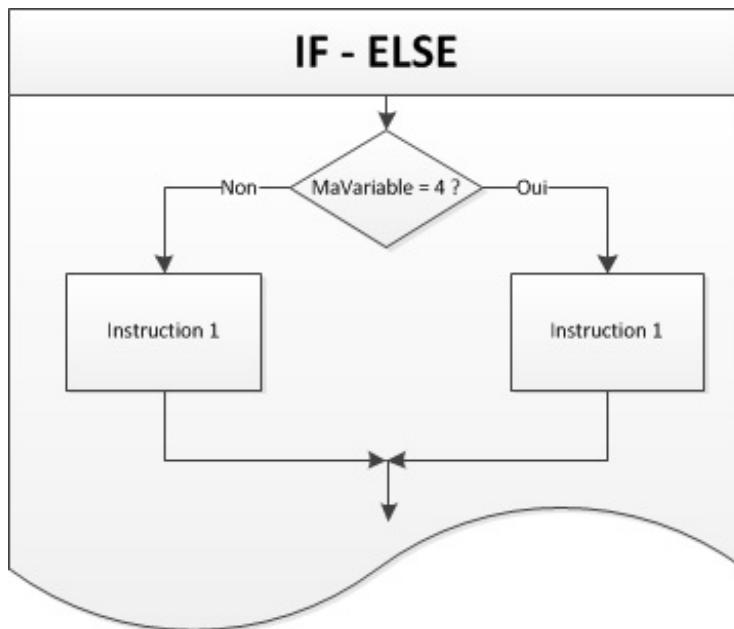


Mais si tu mets MaVariable à 5 dans la boucle, le programme ne va pas en sortir puisque ce n'est plus égal à 10 ?

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 »

« Simon », il faut y penser parfois pour gérer toutes les éventualités. Le **Else** doit être placé dans une boucle **If**, donc entre le **Then** et le **End If**. Regardez la figure suivante.



Fonctionnement de Else

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



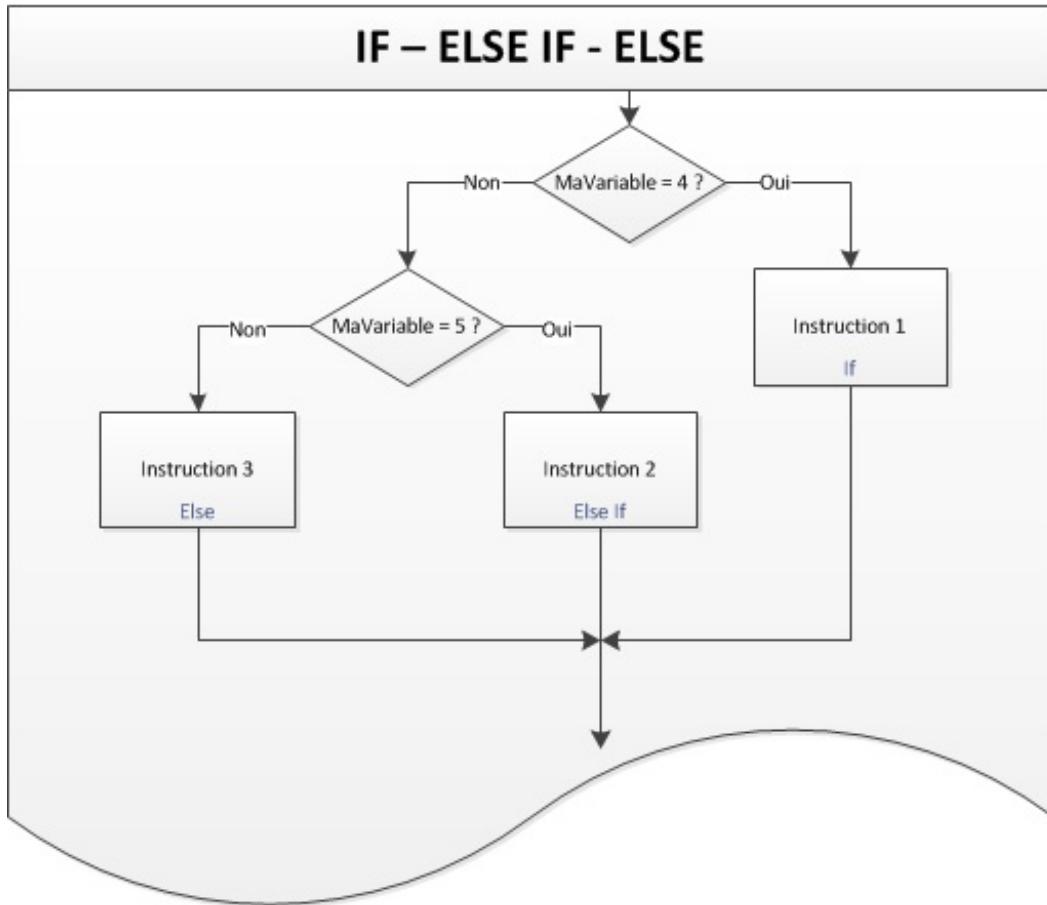
Je vais en profiter pour vous signaler que le symbole « différent » en VB s'écrit « **<>** ». Autrement dit, un signe «



inférieur » et un signe « supérieur » accolés.

ElseIf

La figure suivante schématisise le **ElseIf**.



Fonctionnement de 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. Contrairement à **If** et **ElseIf**, le **Else** ne peut être placé qu'une seule et unique fois dans une condition.

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 = 2
    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 dans lequel vous avez 10 choix différents, 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, et grâce auxquels on simplifie le tout. La syntaxe se construit de la manière 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 :

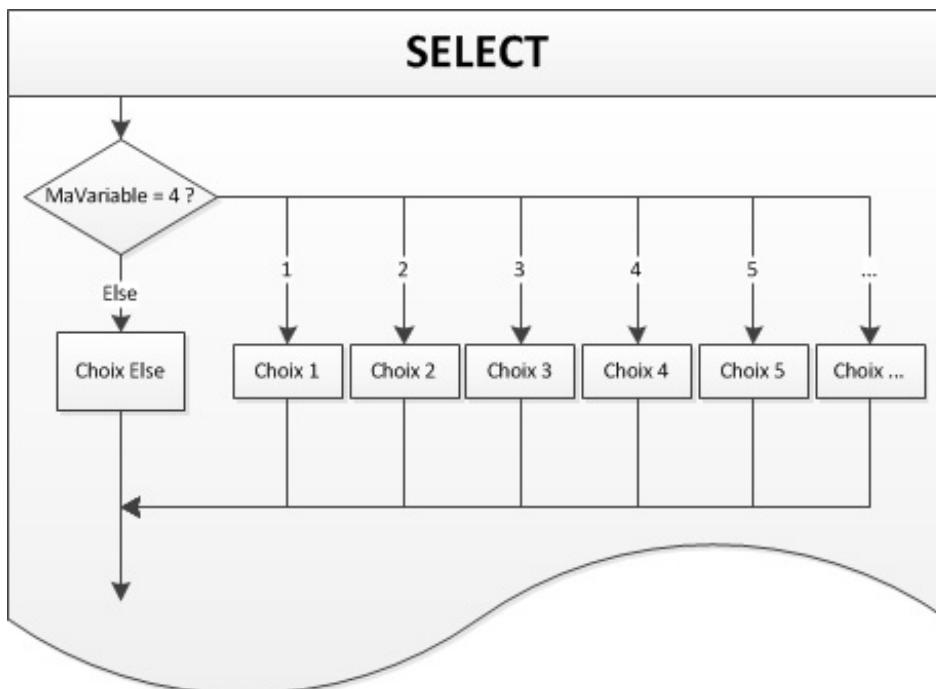
Code : VB.NET

```

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 en figure suivante.



Fonctionnement de Select

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 seul **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

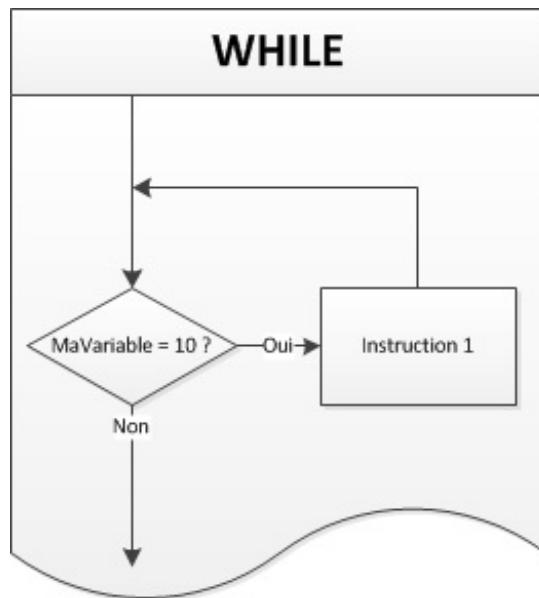
```

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 à la figure suivante pour vous aider à comprendre.



Fonctionnement de While

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.).



Attention aux 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. Si vous êtes confrontés à ce genre de programme (cela peut arriver, ne serait-ce que pour voir ce qu'est une boucle infinie), cliquez sur la croix qui ferme la console pour arrêter le programme.

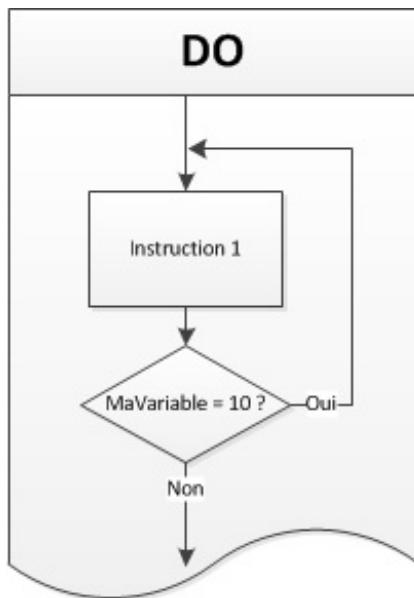


Et si je veux passer au moins une fois dans la boucle même si la condition est fausse, comment dois-je faire ?

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. Regardez la figure suivante.



Fonctionnement de Do

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
    x = x + 1 'Augmente x de 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 parcourue 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 :

Symbol	Fonction
=	Égal
◊	Different
>	Strictement supérieur
<	Strictement inférieur
<=	Inférieur ou égal
>=	Supérieur ou égal

Grâce à 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 précédent 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 supérieure 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. 😊

- **If** signifie « si », on s'en sert pour exécuter une série d'instructions uniquement si une condition est vérifiée.
- **Else** signifie « sinon », on l'utilise avec **If** pour couvrir les cas où la condition ne sera pas vérifiée.
- **While** et **Until** permettent d'effectuer sans arrêt une série d'instructions tant que la condition sera, respectivement, vraie et fausse.
- **For** permet de boucler un certain nombre de fois.

TP : La calculatrice

Nous allons enchaîner avec deux travaux pratiques. 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

Cahier des charges

Donc, c'est parti : je veux (j'ai toujours rêvé de dire ça ! 😊) un programme qui 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 parce que 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 !

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

```



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

Comme 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.

Et voilà l'occasion d'utiliser le **not** vu précédemment. Et de toute façon, si vous ne vous souveniez plus de ce mot clé, 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 ! 😊

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
```



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

Nous venons de voir comment travailler avec des variables de type numérique.

Mais votre clavier vous permet aussi d'écrire des mots, non ? Il faut donc désormais apprendre à utiliser ses capacités. Nous allons donc nous tourner vers les variables de type caractère et chaîne de caractères.

Et puis pour continuer dans les nouveaux types de variables, nous allons aussi en profiter pour étudier les variables temporelles de type date.

D'ici peu de temps vous pourrez modifier des mots aussi rapidement que des nombres.

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 en question (considérée comme un objet) 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 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 ? 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 21:06:33
```

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
```



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* (heures:minutes:secondes).

Heureusement que vous avez lu ce qui est indiqué précédemment, 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.

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.ToString("HH:mm:ss"))
            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.ToString("HH:mm:ss"))
            '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 !

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.

- Nous venons de voir différentes fonctions disponibles sur une chaîne de caractères.
- Une chaîne de caractères a le type `String`.
- Une date et une heure sont contenus dans des variables de type `Date`.

Les tableaux

Un tableau va 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, on saisit les valeurs dans un tableau.

Les dimensions

Tableau à une dimension

Représentation d'un tableau à une dimension

(0)	(1)	(2)	(3)	(4)
-----	-----	-----	-----	-----

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** (entier) 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 inscrites 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 longueur 4 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 celle de la fonction ; le logiciel de développement vous donnera des indications lorsque vous allez écrire la ligne, pour que vous évitez de confondre fonctions et tableaux.

Si vous comprenez aussi vite, passons au point suivant.

Tableaux à deux dimensions

Représentation
d'un tableau à deux
dimensions

(0,0)			(0,4)
(1,0)			
(3,0)			(3,4)

Il s'agit ici d'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.

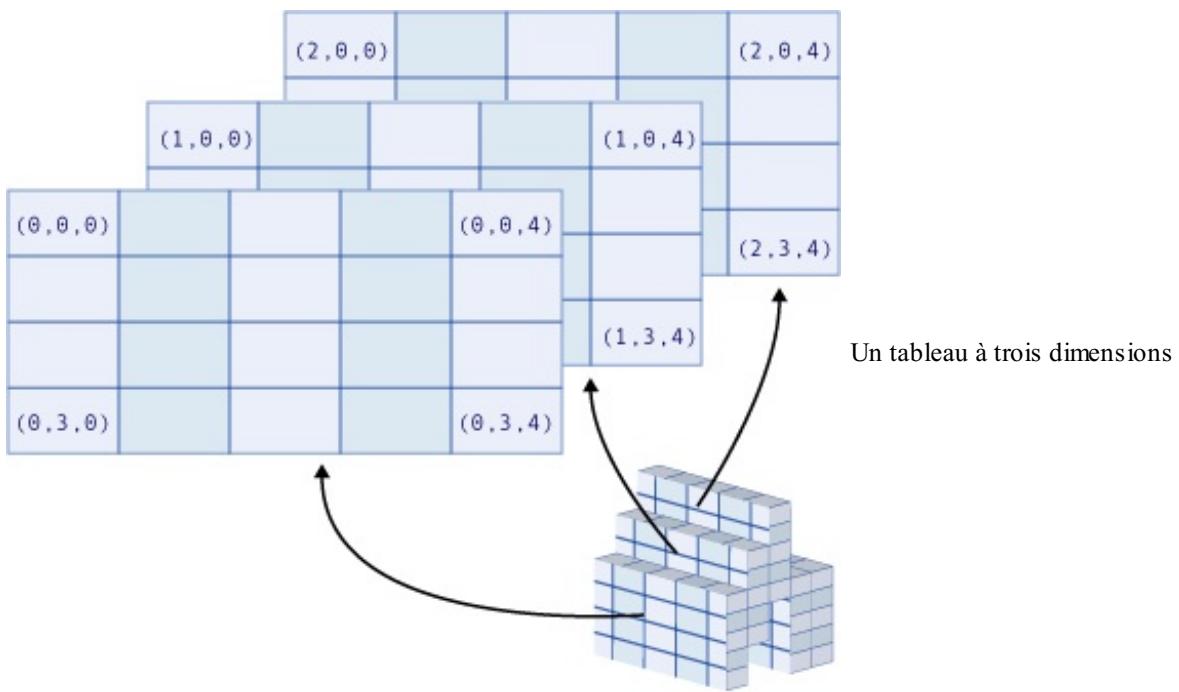


Il est bien important que vous vous représentiez mentalement la façon dont laquelle est construit un tableau à plusieurs dimensions, pour comprendre comment agir.

Un tableau à deux dimensions peut servir comme tableau à double entrée par exemple.

Tableaux à trois dimensions

Regardez la figure suivante.



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 trois dimensions (cela devient compliqué !).

Code : VB.NET

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

Ce code crée un tableau de tableaux 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 !

Autres manipulations avec les tableaux

Ici, nous allons découvrir quelques trucs et astuces supplémentaires concernant les tableaux.

Redimensionner un tableau

La taille d'un tableau peut être redimensionnée au cours d'une application. L'instruction **Redim** permet de modifier la taille du tableau.

Code : VB.NET

```
Redim monTableau(20)
```

Imaginons que ce tableau qui possédait 10 cases auparavant en possède maintenant 21 pour stocker des informations. Le seul problème, c'est que lorsque le programme rencontre cette ligne, le contenu entier du tableau est supprimé de la mémoire lors du redimensionnement. 😞

Mais... il existe une solution ! 😊 Pour pouvoir conserver le contenu d'un tableau lors d'un redimensionnement, il faut spécifier le mot-clé **Preserve** après **Redim**.

Code : VB.NET

```
Redim Preserve monTableau(20)
```



L'instruction **Redim** n'est valable que pour les tableaux à une seule dimension, si vous utilisez cette instruction sur un tableau multidimensionnel, seule la dernière dimension du tableau peut être modifiée.

Retourner un tableau

Voici une fonction qui nous sera utile si l'on souhaite inverser le sens d'un tableau.

Par exemple, j'ai un tableau qui contient les nombres de 1 à 10, je souhaite avoir ce comptage de 10 à 1. Cette méthode peut alors être utilisée pour effectuer cette opération.

Son utilisation :

Code : VB.NET

```
Array.Reverse(monTableau)
```

Pas besoin de beaucoup d'explications je suppose, on passe simplement le tableau à retourner en paramètre.

Vider un tableau

Pour vider rapidement un tableau, une méthode existe. Supposons que vous veniez de faire un traitement sur un tableau, vous

voulez le retrouver ensuite « comme neuf » pour pouvoir le réutiliser. Plutôt que d'en créer un nouveau, pourquoi ne pas nettoyer l'ancien ?

Code : VB.NET

```
Array.Clear(monTableau, 0, 10)
```

Trois paramètres sont nécessaires ici. Le premier est très simplement le tableau à vider, le second spécifie à partir de quel index vider, et le troisième indique le nombre de cases à vider.

Vous l'avez donc compris, si je veux garder les dix premiers éléments de mon tableau intacts et vider les éléments de 10 à 20, j'écrirai :

Code : VB.NET

```
Array.Clear(monTableau, 10, 10)
```

 Attention, le dernier paramètre est le nombre de cases à vider et non pas la dernière case à vider.

 Cette méthode est intelligente, elle va s'adapter en fonction du type de votre tableau. Si votre tableau contient des nombres, la valeur de « nettoyage » sera un 0. S'il contient des booléens, ce sera un **false**. Pour le reste, la valeur sera NULL.

Copier un tableau dans un autre

Dernière petite fonction utile, celle permettant de copier un tableau dans un autre.

Code : VB.NET

```
Array.Copy(monTableau1, monTableau2, 5)
```

Trois paramètres, les deux premiers étant des tableaux. Le premier tableau étant la *source* (celui dans lequel nous allons copier les éléments) et le second est la *destination* (celui dans lequel nous allons coller les éléments). Le troisième paramètre est le nombre d'éléments à copier (depuis l'élément 0). Ainsi, 5 indique que 5 cases seront copiées dans l'autre tableau.

 Si vous souhaitez remplir le second tableau entièrement, utilisez `Array.Copy(monTableau1, monTableau2, monTableau2.Length)`. Cela permet de spécifier que l'on veut copier autant de cases que disponibles dans le deuxième tableau. Nous analyserons ce `.Length` en détail plus tard.

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 !

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`, comme à la figure suivante.

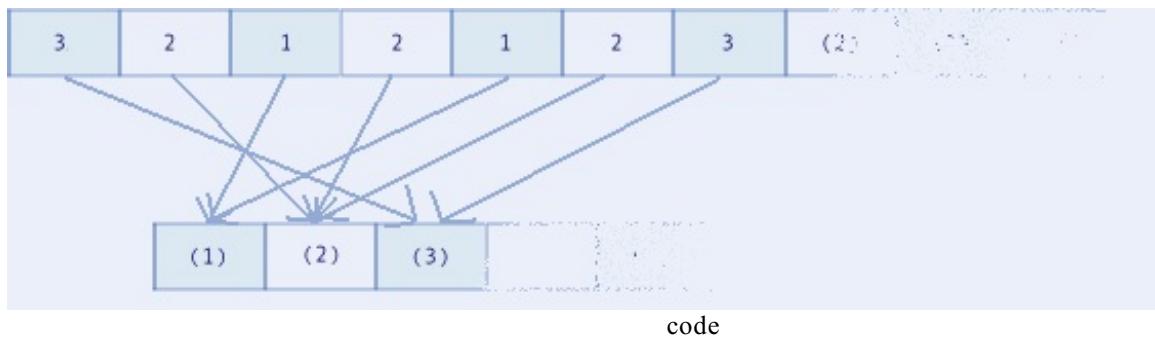


Schéma explicatif du

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(NombreTrouve) + 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, je vous invite à [lire ce cours](#) rédigé par ShareMan.

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 énumérations

Nous allons maintenant nous pencher sur un type un peu plus spécial : les **énumérations**. Une énumération va nous permettre de définir un ensemble de constantes qui sont liées entre elles.



Une énumération n'est pas un tableau !

Par exemple, il pourrait être très facile de représenter les jours de la semaine dans une énumération plutôt que dans un tableau :

Code : VB.NET

```
Enum jours
    Dimanche
    Lundi
    Mardi
    Mercredi
    Jeudi
    Vendredi
    Samedi
End Enum
```

On définit une énumération par le mot-clé **Enum**, une énumération se termine par **End Enum**.



Je fais appel à votre attention : une énumération se déclare *en dehors* du `Main()`.

Dans une énumération, la première valeur est initialisée à 0, la suivante est augmentée de 1 : dans notre exemple Dimanche vaut 0, Lundi vaut 1, Mardi vaut 2...

Il est possible d'entrer des valeurs pour chaque valeur de l'énumération :

Code : VB.NET

```
Enum jours
    Dimanche = 10 'Dimanche vaut 10
    Lundi = 11 'Lundi vaut 11
    Mardi = 26 'Mardi vaut 26
    Mercredi = 11 'Mercredi vaut 11
    Jeudi = 12 'Jeudi vaut 12
    Vendredi = 30 'Vendredi vaut 30
    Samedi = 31 'Samedi vaut 31
End Enum
```

Les valeurs de l'énumération qui n'ont pas reçu de valeur particulière prendront la valeur précédente augmentée de 1.

Une fois définie, une énumération peut ensuite être utilisée comme un type de variable spécifique :

Code : VB.NET

```
Dim joursSemaine As jours
```

Ce qui donne en entier :

Code : VB.NET

```
Enum jours
    Dimanche
    Lundi
    Mardi
    Mercredi
    Jeudi
    Vendredi
    Samedi
End Enum

Sub Main()
    Dim joursSemaine As jours

    joursSemaine = jours.Dimanche
    Console.WriteLine(joursSemaine)
End Sub
```

Ce qui nous donne :

Code : Console

```
Dimanche
```

- Un tableau peut contenir plusieurs valeurs.
- Un tableau a des cases, on peut accéder à chaque case en particulier.
- La première case d'un tableau a toujours la position 0.
- Un tableau peut être à plusieurs dimensions. Les tableaux de base sont à une dimension et sont comparables à des listes.

Les fonctions

Une fonction répète une action bien précise. Nous en connaissons déjà, par exemple BEEP ou IsNumeric(), qui vérifie que la valeur d'une variable est bien un nombre. Vous voyez, des programmeurs ont déjà créé et intégré des fonctions dans les bibliothèques, d'énormes fichiers qui les rassemblent toutes et 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.

Créons notre première fonction !

Nous allons donc créer notre première fonction, la plus basique qui soit : sans argument, sans retour. Mais 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 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 qualiferais 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 quelque chose de génial : les arguments et le retour.

Les arguments

Vous savez déjà ce qu'est un argument. 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, le mot **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 arguments.



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 arguments 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 renournée, `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>
```

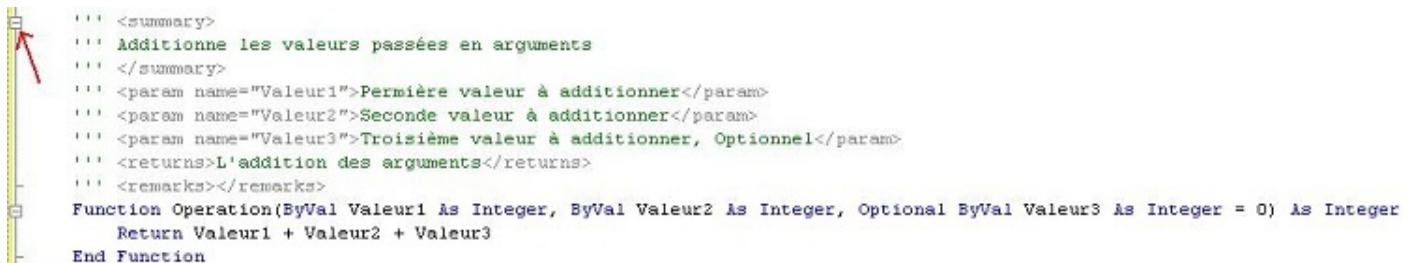
Ces lignes permettent de commenter la fonction : dans *summary*, expliquez brièvement le rôle de la fonction ; dans les paramètres, précisez à quoi ils correspondent ; et dans la valeur de retour, indiquez ce que la fonction retourne.

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,
Optional</param>
''' <returns>L'addition des arguments</returns>
''' <remarks></remarks>
```

Cliquez ensuite sur cette petite flèche pour « replier » cette zone, comme à la figure suivante.



```
''' <summary>
''' Additionne les valeurs passées en arguments
''' </summary>
''' <param name="Valeur1">Première valeur à additionner</param>
''' <param name="Valeur2">Seconde valeur à additionner</param>
''' <param name="Valeur3">Troisième valeur à additionner, Optional</param>
''' <returns>L'addition des arguments</returns>
''' <remarks></remarks>
Function Operation(ByVal Valeur1 As Integer, ByVal Valeur2 As Integer, Optional ByVal Valeur3 As Integer = 0) As Integer
    Return Valeur1 + Valeur2 + Valeur3
End Function
```

Cliquez sur la flèche pour « replier » la 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, comme à la figure suivante.

Operation(10, 15,	Operation (Valeur1 As Integer, Valeur2 As Integer, [Valeur3 As Integer = 0]) As Integer
Valeur3:	Troisième valeur à additionner, Optionnel

Un cadre s'affiche

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 !

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.

- Si l'on commence la déclaration de la fonction avec le mot-clé **Sub**, elle ne retournera pas de valeur, ce sera alors une méthode.
- Une fois l'instruction contenant **return** exécutée, la fonction ne continue pas son exécution.
- Vous utilisez des centaines de fonctions déjà écrites dans le framework de Visual Basic .NET, vous pouvez en créer autant que vous le voulez.
- Les arguments optionnels sont définis avec le mot-clé **Optional**.

Les inclassables

Vous êtes déjà en possession de beaucoup de connaissances pour programmer. Cependant, la programmation contient des milliers et des milliers de concepts et de notions. Vous vous doutez bien que je ne peux pas toutes vous les apprendre, à vous de chercher un peu par vous-mêmes.

Toutefois, dans ce dernier chapitre, je vais tenter de vous apporter quelques nouveaux concepts qui pourront être plus ou moins utiles lors de vos projets. Ces notions sont un peu inclassables, car n'ayant aucun rapport entre elles. Mais elles n'en sont pas moins intéressantes !

Accrochez-vous donc bien pour ces six nouveaux concepts.

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 difficultés.

Les structures

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

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

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

Vous avez l'intention de créer des fiches de livres 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 supplémentaires

Vous vous souvenez encore des boucles conditionnelles ? 🍑

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



Oh, 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 vérifie si chaque valeur est inférieure à 10 ; si c'est le cas, il l'affiche.



On ne peut pas assigner de valeur 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

```
i
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 : en un certain sens, il simplifie le **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 casts

J'ai brièvement parlé des *casts* 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 *casts*, 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 *casts* : 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 que j'ai fait précédé d'une astérisque « * » sont les plus utilisées. Attention, les « * » ne font pas partie de la fonction, c'est uniquement pour mieux les repérer.



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

Un élément supplémentaire dans 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 qu'il est 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.

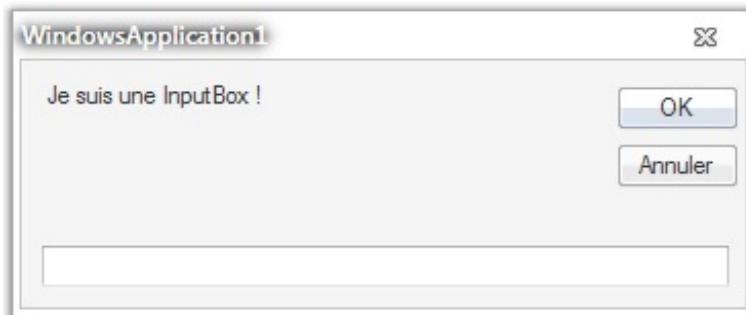
Les MsgBox et InputBox

Deux petites choses qui peuvent également vous aider : les **MsgBox** et les **InputBox** (voir figures suivantes).





Une MsgBox



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 paramétrier, je vais vous les expliquer.

La MsgBox

Les paramètres

Voici 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 bouton 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 dans le tableau suivant divers exemples de style.

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.

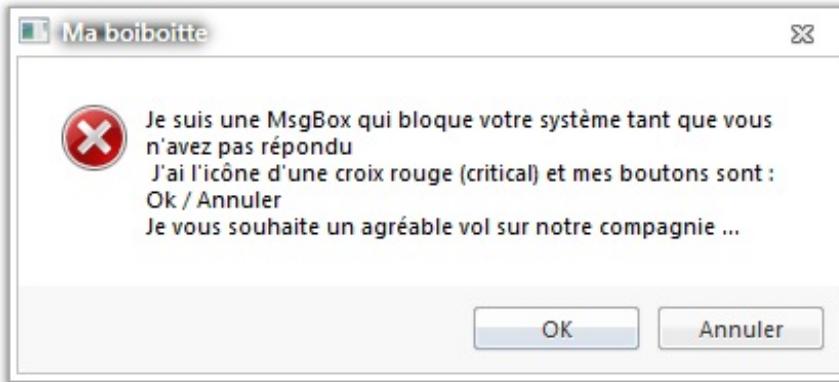
Les numéros indiqués correspondent aux *ID*, que vous pouvez cumuler. En gros, si vous souhaitez que votre boîte bloque le système et que l'on doive y répondre avant de continuer, avec une icône « Message critique » et des boutons « OK – Annuler », il faut que vous tapiez... $4096 + 1 + 16 = 4113$!

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  
boîte")
```

Cela donne le résultat visible à la figure suivante.



Le rendu du code précédent

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 et ne sont 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 (sur l'horizontale, en partant de la gauche).
5. La position de la boîte en Y (sur la verticale, en partant du haut).

L'origine du repère se situe donc 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 !

- Les constantes sont des variables destinées à ne pas changer de valeur.
- **For each** permet d'itérer sur chaque valeur d'une liste, d'un tableau.
- MsgBox et InputBox sont des fenêtres de dialogue pour capter l'attention de l'utilisateur.

Partie 2 : Le côté visuel de VB

Déprimés avec tout ce noir ? Voir la console pendant toute une partie, ça vous 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 a 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

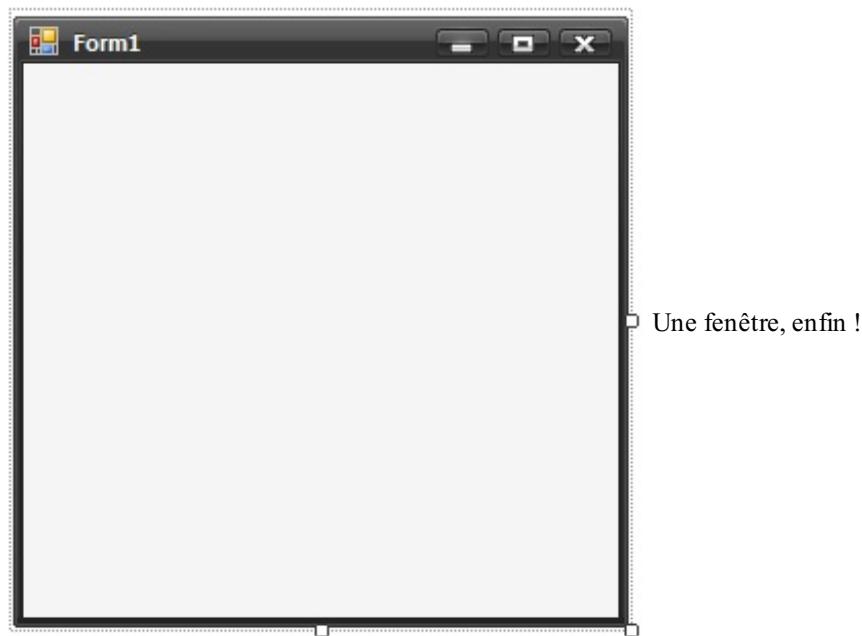
Des fenêtres, je veux des fenêtres ! À partir de maintenant, finis les essais au milieu du noir et du blanc de notre console. Nous allons donc commencer à aborder les nouveaux concepts du graphisme en commençant par placer des **contrôles** et découvrir les **événements**.

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 uniquement eu besoin de deux fonctions 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. Admirez le superbe résultat visible à la figure suivante.



Sur notre gauche nous retrouvons le panneau que je vous ai présenté tout au début de ce tutoriel : la boîte à outils. Cette boîte 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 une feuille de papier sur laquelle vous allez dessiner.

Cette "feuille de papier" est appelée feuille de style ou fenêtre de design. Elle est uniquement dédiée à construire la partie "graphique" de votre futur programme.

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 que la console. Il faudra passer par nos outils pour écrire et interagir avec l'utilisateur. Il faudra donc bien les connaître 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 ; le bouton, à déclencher un évènement.

Manipulation des premiers objets

Retournons à notre feuille :

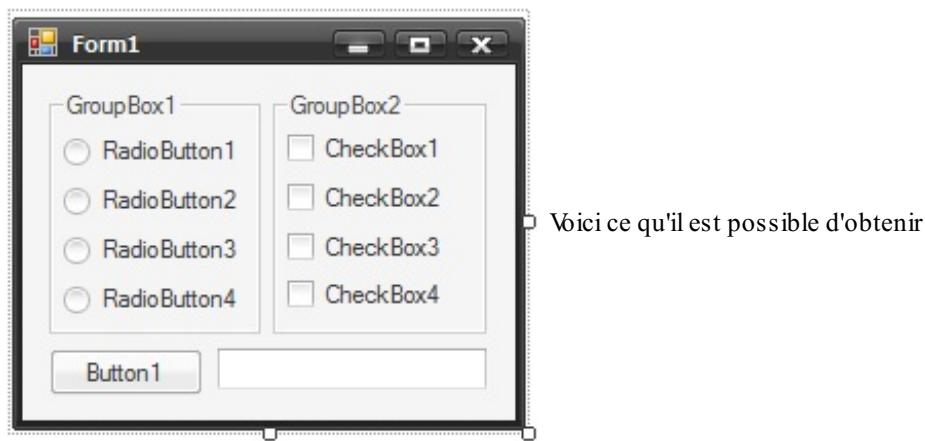
Rien qu'elle, vous pouvez déjà la manipuler : quand vous cliquez dessus, des carrés blancs apparaissent sur les bords (voir figure suivante), ils vont nous permettre d'agrandir ou réduire la fenêtre, comme n'importe quelle autre fenêtre Windows.



d'agrandir ou de réduire la fenêtre

Une fois cette fenêtre à la hauteur de vos espérances, nous allons apprendre à ajouter des **objets** dedans, ces objets sont appelés des **contrôles**. Pour ce faire, je vais vous laisser vous amuser avec les objets : prenez-les en cliquant dessus, puis faites-les glisser jusqu'à la fenêtre sans relâcher le clic.

Laissez libre cours à votre imagination, essayez tous les objets que vous voulez ! Regardez ce que j'obtiens à la figure suivante



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

Stop ! pourquoi vouloir savoir courir avant de savoir 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 ! 😊

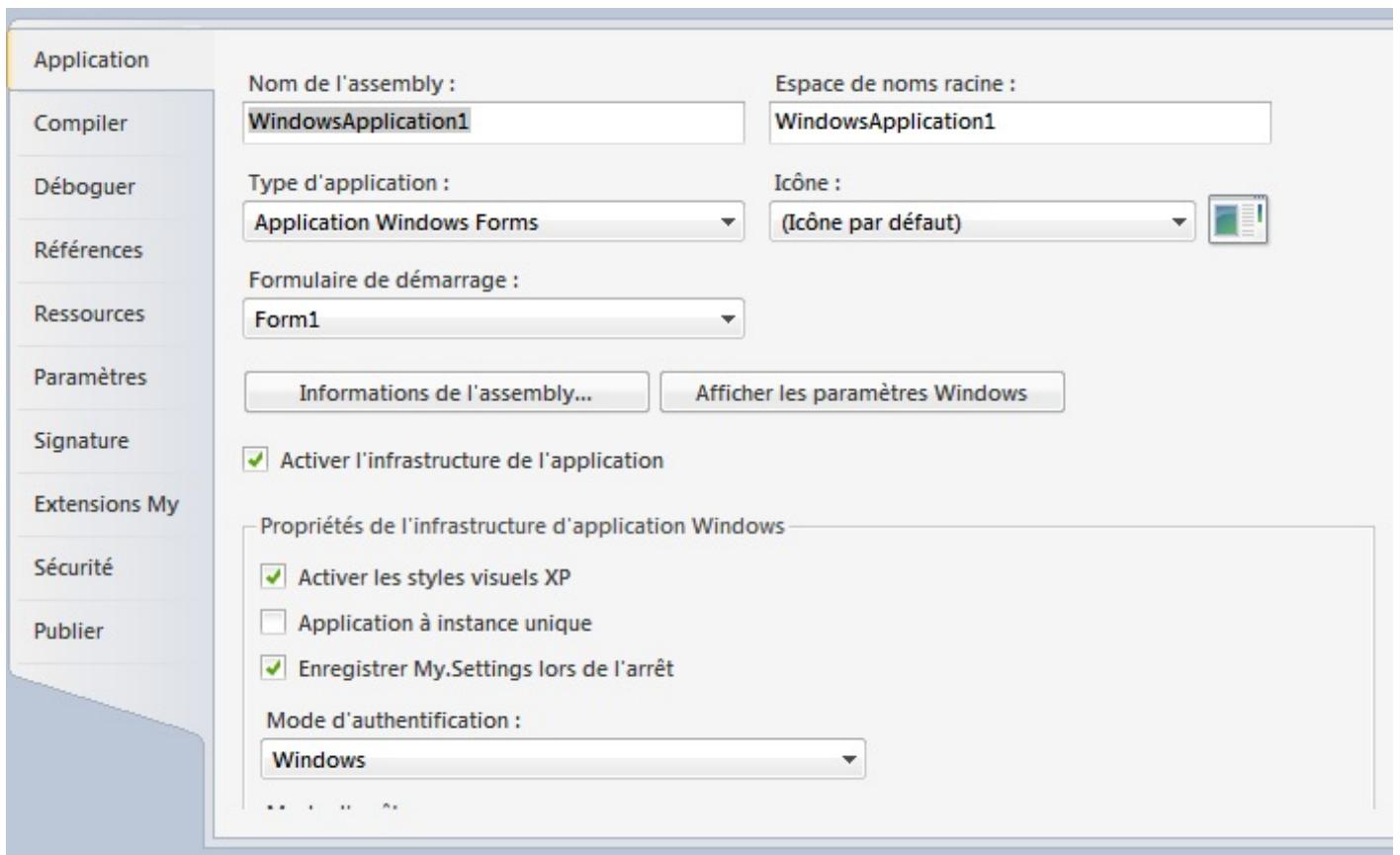
En attendant vous pouvez déjà lancer ce projet, votre fenêtre apparaîtra. Bon, rien ne se passe quand vous appuyez sur les boutons, pourquoi ?

Eh bien, nous n'avons pas encore codé **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]. Une fenêtre semblable à la figure suivante devrait apparaître.



Une fenêtre s'ouvre

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 sont plus techniques, et pas nécessaires actuellement.

- On crée un projet Windows Forms pour pouvoir utiliser l'interface graphique.
- Les contrôles sont disponibles dans la boîte à outils, ils nous permettent de concevoir notre interface.
- La feuille de style (design) est la fenêtre dans laquelle on conçoit l'interface graphique.

Les propriétés

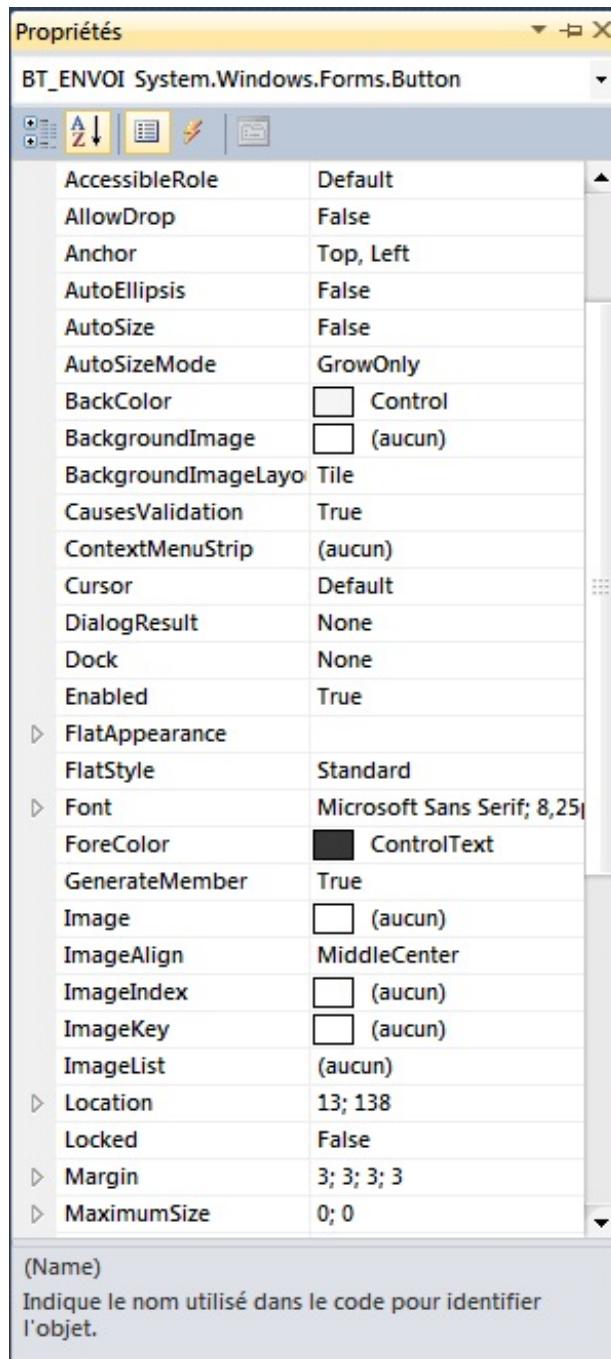
Eh bien, cela fait déjà pas mal de nouveaux concepts à apprêhender, mais ce n'est pas fini ! Vous ne voulez quand même pas en rester là ? Modifions maintenant les **propriétés** de nos contrôles !

Ces propriétés vont nous permettre de modifier à souhait et à la volée les contrôles visuels. Que ce soit la couleur, le texte, l'emplacement, la taille, **le poids**, tous ces paramètres vont pouvoir être modifiés quand vous le souhaitez.

Je ne vous en dit pas plus, allons-y.

À quoi ça sert ?

Je pense que vous avez sûrement déjà vu la fenêtre contenant les propriétés (voir figure suivante).



La fenêtre des propriétés

La chose magnifique est que nous sommes sous Visual Basic Express, un module du grand IDE qu'est Visual Studio. Et cet IDE va vous permettre :

- D'avoir les messages d'erreur écrits en français la plupart du temps ;
- De résoudre les erreurs sans se poser de questions ;
- D'avoir la description de toutes les propriétés des objets ;

- D'avoir la description de toutes les fonctions ;
- D'avoir un système de debug 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, tout sera très intuitif, un vrai plaisir.

Donc revenons à nos moutons : qu'est ce qu'une propriété sur un objet VB ? Attention, ici je parle des objets graphiques 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 faire apparaître ou disparaître un objet dynamiquement, l'activer ou 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 côté VB.



OK, OK, on te croit. Au fait, comment ouvrir le code ? je ne vois que le design...

Oh, excusez-moi, pour ouvrir le code, allons dans notre fenêtre de 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 :

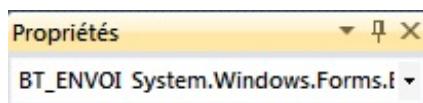
- Le concepteur de vues pour le côté design ;
- Et... afficher le code ! Voilà comment afficher le code.

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 : ouvrez la fenêtre des propriétés. Si vous ne savez plus comment on fait, appuyez sur la touche F4 de votre clavier.

Bon, votre fenêtre est ouverte. La figure suivante représente ce qu'il y a dans la partie supérieure.



La partie supérieure de la fenêtre « Propriétés »

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 qu'il n'est pas possible d'accéder côté VB aux propriétés entre parenthèses.

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

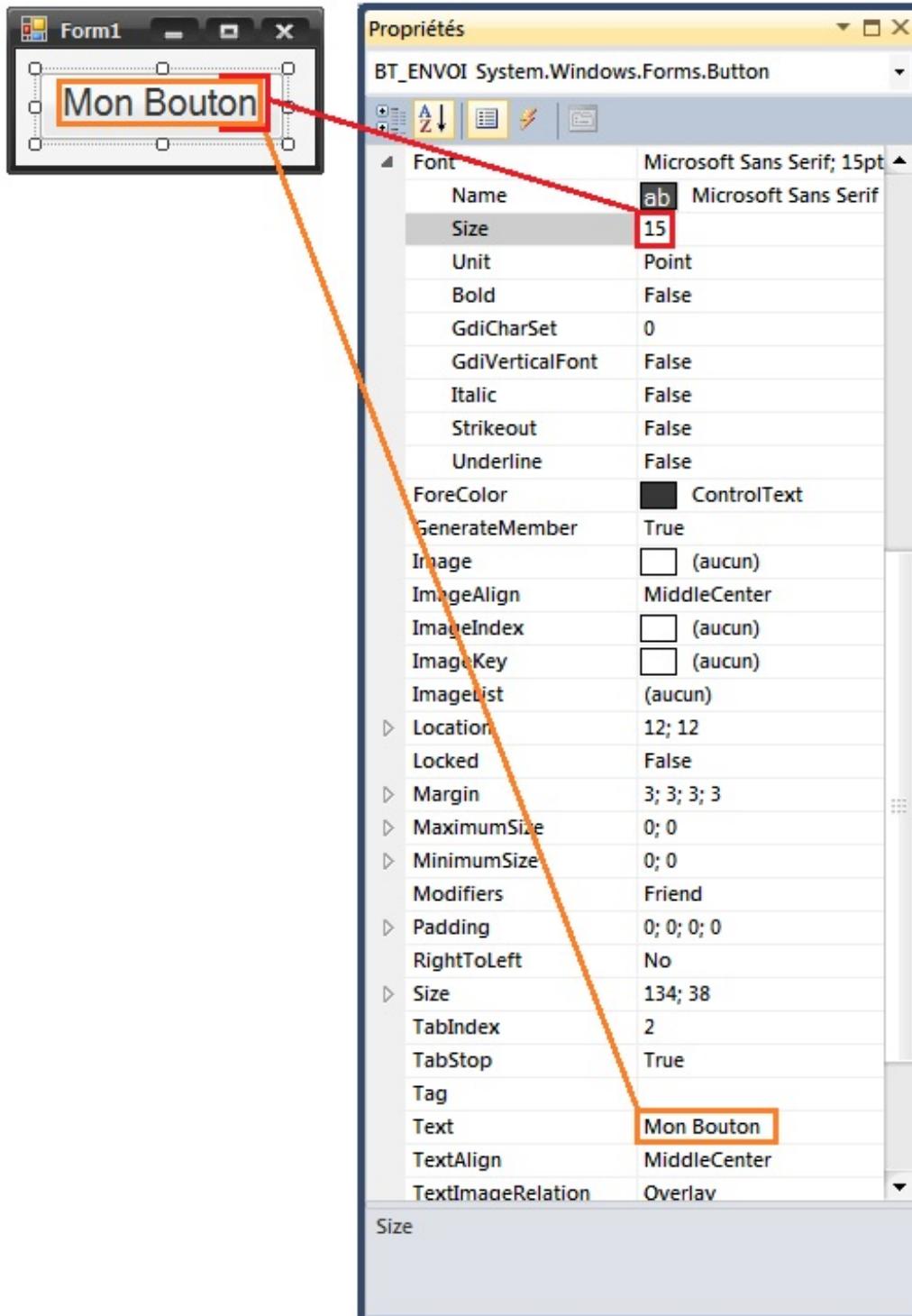
Je vous explique ma manière de procéder : mes contrôles sont toujours en MAJUSCULES.
Ils contiennent un préfixe en fonction de leur type :

- BT pour les boutons ;
- LBL pour les labels ;
- TXT pour les textbox ;
- LNK pour les *labels links* ;
- RB pour les boutons radio ;
- CHK pour les checkbox ;
- Et j'en passe...

Je ne vous oblige absolument pas, mais je vous le 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 dont il s'agit, c'est un button.

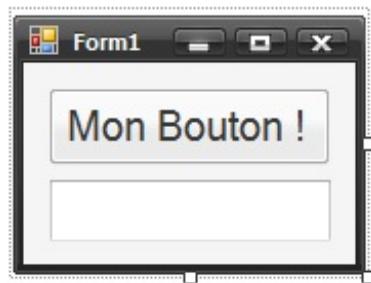
Dans le reste de cette fenêtre (voir figure suivante), vous voyez la liste des propriétés de l'objet. Cliquez sur la case correspondante pour lui assigner une propriété. Dans le cas de mon bouton, vous le voyez à droite, j'ai modifié sa propriété font, qui veut dire en anglais « police », j'ai changé sa size (autrement dit sa taille), et j'ai modifié également sa propriété Text, pour lui changer son nom.



Propriétés du bouton

 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 modifiais le mauvais...

Pour qu'on travaille tous sur la même chose, on va tous créer la fenêtre visible à la figure suivante.



Créez une fenêtre identique à celle-ci

Donc j'ai un bouton appelé « BT_ENVOI », et une TextBox, que vous trouvez également sur le côté pour placer vos objets et qui s'appelle TXT_RECOIT. Et c'est parti pour l'aventure !

Les assigner et les récupérer côté VB

Nous allons donc modifier les propriétés des objets côté VB.

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

Ici, c'est à peu 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 deux manières : l'écrire, mais comme vous ne connaissez 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 créez (pas un *contrôle* surtout !).

Vous atterrissez côté 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

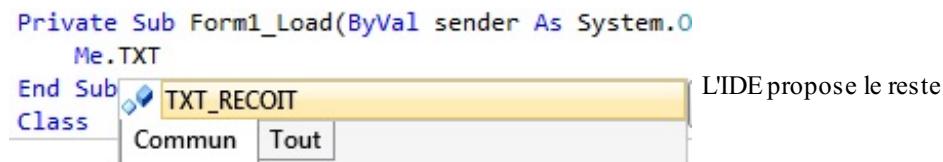
    End Sub

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, utilisons donc le caractère « . » : 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** ». À peine la partie « **TXT** » écrite, notre formidable IDE nous donne déjà le reste, comme le montre la figure suivante.



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 précédemment, il nous affiche déjà le reste.

Nous voilà donc sur la propriété `text`, nous avons deux choix : attribuer sa valeur ou l'utiliser ; pour l'utiliser le signe « = » se place avant, et pour l'assigner, après.

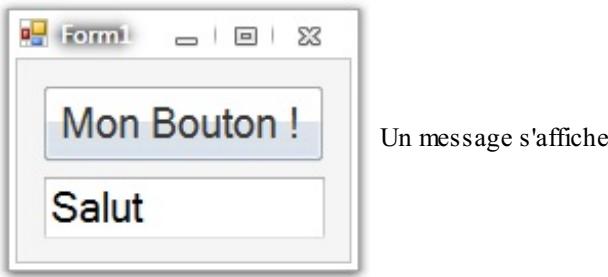
Assigner est utile pour entrer des valeurs, utiliser sa valeur est pratique pour des cas comme les textbox, et récupérer ce que l'utilisateur a entré.

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

Code : VB.NET

```
Me.TXT_RECVIT.Text = "Salut"
```

Voilà, lançons le programme pour essayer : un salut s'est affiché dans la textbox (voir figure suivante) !

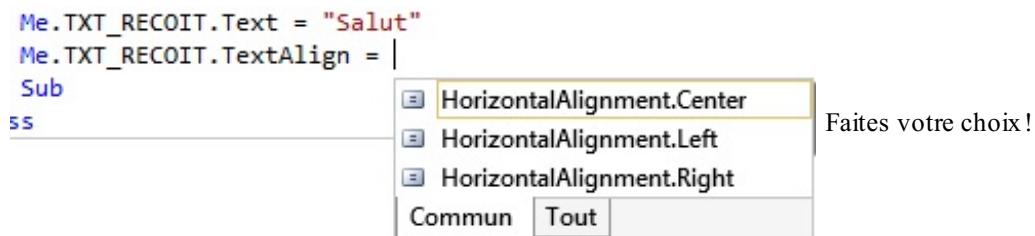


Nous avons réussi notre premier accès à une propriété côté VB.

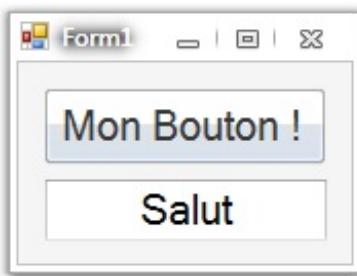
Pourquoi s'arrêter là ? alignons ce texte !

Tapons donc `Me.TXT_RECVIT.TextAlign`, et là l'autocomplétion 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 (voir figure suivante) !



Centré, à gauche ou à droite ? Double-cliquez sur votre choix (j'ai choisi centré dans cet exemple). Au nouveau test de notre application, vous obtenez la figure suivante.



Le texte est centré

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

With

Voici un petit mot qui va changer votre vie : **With** (autrement dit en français : « avec »).

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 d'e-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 expliqué, vous allez écrire en toute logique ceci :

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 informations **Top** et **Left** positionnent le coin supérieur gauche de votre contrôle à la manière des **inputbox**, avec **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 c'est un peu lourd comme notation, n'est-ce pas ?

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 à d'autres objets que le bouton durant le **With**. Un `MonLabel.Text = "Test"` aurait bien sûr été 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.

- Le `Form_Load` est désormais appelé lors du chargement du programme.
- Il faut bien veiller à accéder à la bonne propriété de chaque objet. Apprenez à connaître les objets en expérimentant ou en lisant la documentation.
- Le mot-clé **With** permet de se substituer à l'objet auquel on veut accéder, et ce jusqu'au **End With**.

Les événements

Bon, attaquons maintenant réellement les événements.

Je vous ai déjà expliqué un événement, le `form_load`. Eh bien découvrons les autres afin de réagir à plein d'autres choses : un clic, une touche, une ouverture, une fermeture, que sais-je encore, les possibilités sont nombreuses !

Pourquoi ça encore !

Alors, un événement s'écrit comme un **Sub** ; par exemple, l'événement `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'événement, `load`. 😊



Je peux donc l'appeler autrement ?

Bien sûr, mais je vous conseille de vous habituer à 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 crée 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'événement 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 : un événement attend que le chargement de la fenêtre s'effectue !

Créer nos événements

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 au clic du bouton ?

Allons-y ! Comme pour générer l'événement `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-y les instructions voulues ; fainéant 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 événement le plus courant ; pour un bouton : le clic, pour une textbox : le changement de texte, etc.

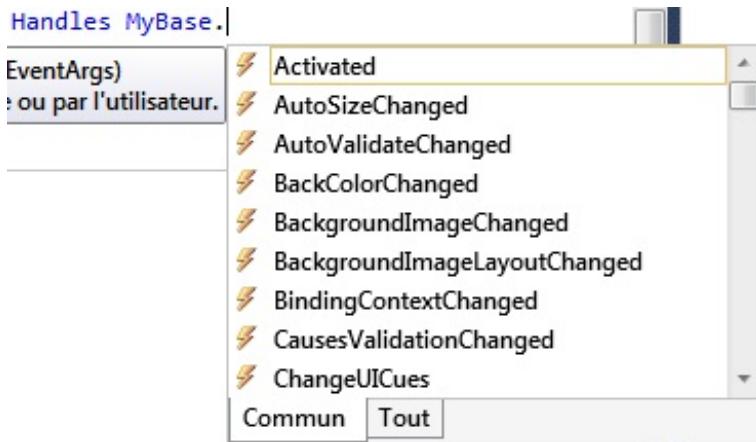
Mais pourquoi se limiter à cela puisqu'il existe des dizaines d'événements différents pour chaque objet ? Allons réagir manuellement à tous ces événements !

Les mains dans le cambouis !

Créons donc nos événements !

Enfin modifions-les, plutôt ! Eh oui, je vous l'ai dit, je suis fainéant, pourquoi les créer puisqu'un assistant le fait pour nous ? Enfin, puisqu'il fait la moitié du travail.

Double-cliquons donc sur la fenêtre, l'événement `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 (« . »). Voici la liste d'événements de l'objet fenêtre qui s'ouvre, comme à la figure suivante.



La liste d'événements de l'objet fenêtre s'ouvre

Eh bien, choisissons l'événement `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'événement du clic sur le bouton dans ce nouvel événement. Essayons, effectivement lors du clic de la souris sur la fenêtre (pas le bouton) le texte s'affiche ! Encore réussi !

Amusez-vous avec ces événements, essayez-en, soyez amis avec. 😊

Mini-TP : calcul voyage

Nous passons à un mini-TP pour utiliser les événements 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/100 km) ;
- Le nombre de kilomètres ;
- Le prix de l'essence (en euros).

L'utilisateur entrera ces informations 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 peu marcher vos méninges, c'est aussi ça la programmation !

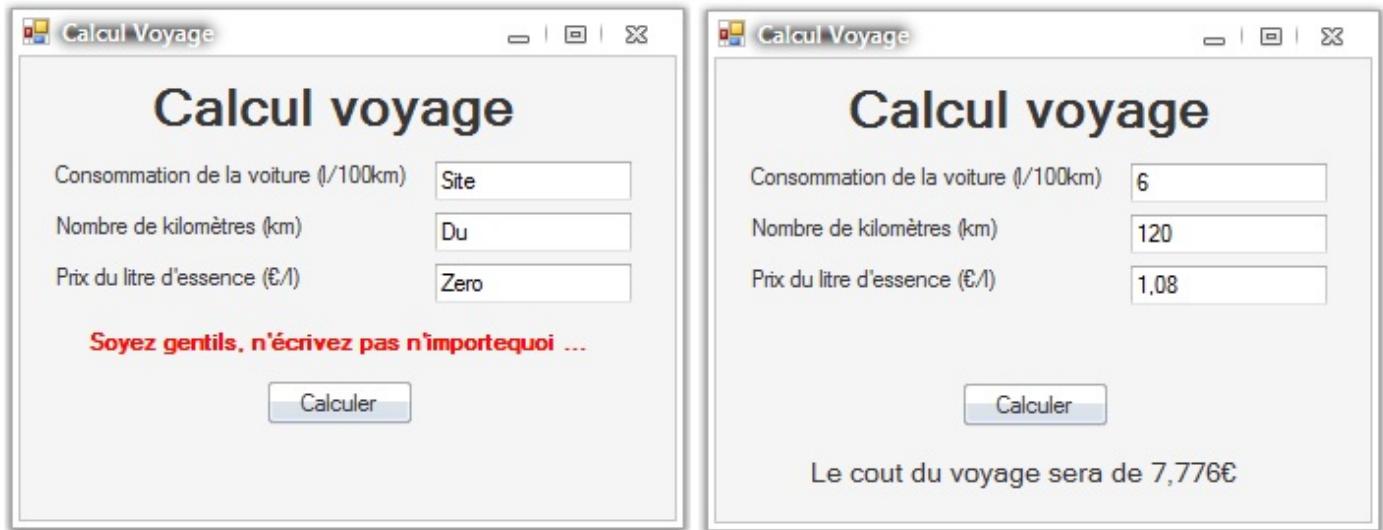
Et je veux que ce programme réagisse aussi aux utilisateurs qui s'amusent à entrer n'importe quoi.

Bonne chance !

Correction

J'espère que vous avez trouvé par vous-mêmes, je vous avais tout expliqué ! Bon, je vous montre !

Avant toute chose, regardez la figure suivante. Elle vous montre mon résultat final. À gauche le programme initial, à droite une fois le calcul effectué.



Voici à quoi ressemble mon programme

Bon, l'explication des objets : j'ai placé trois textbox, une pour chaque valeur à entrer.

Leurs noms sont respectivement TXT_CONSUMMATION, 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'agirai pas dessus pendant le programme, alors autant laisser comme ils sont.

Ensuite je leur ai attribué 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'y ai écrit le texte « Calculer ».

Il reste deux 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 !

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_CONSUMMATION.Text, Me.TXT_NBKM.Text,
            Me.TXT_PRIXESSION.Text) & "?"
        Else
            Me.LBL_ERR.Visible = True
        End If
    End Sub

    ''' <summary>
    ''' Vérifie les trois 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_CONSUMMATION.Text Is Nothing Or Not
        IsNumeric(Me.TXT_CONSUMMATION.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_PRIXESSION.Text Is Nothing Or Not
        IsNumeric(Me.TXT_PRIXESSION.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 nombre de kilomètres
    ''' </summary>
    ''' <param name="Consommation">Consommation</param>
    ''' <param name="NbKm">Distance parcourue</param>
    ''' <param name="PrixEss">Prix du kérosène</param>
    ''' <returns>Le coût 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 événement : l'appui sur le bouton. Événement que j'ai créé grâce à l'assistant, en double-cliquant dessus.

Dans cet événement, j'utilise ma fonction `Verification()` ; si le résultat est vrai, j'utilise ma fonction `calcul()` en lui

passant comme arguments les valeurs des trois textbox, et j'écris le résultat sous la forme « Le coût du voyage sera de XXX ».

Si la fonction `Verification()` renvoie faux, j'affiche le message d'erreur.

Passons donc aux fonctions.

La fonction `Verification()` : cette fonction est spécifique à ce programme, je ne pourrai pas l'utiliser ailleurs, pourquoi ? Tout simplement parce que j'accède à des objets qui sont sur cette feuille uniquement :

Code : VB.NET

```
Dim Bon As Boolean = True
If Me.TXT_CONSUMMATION.Text Is Nothing Or Not
    IsNumeric(Me.TXT_CONSUMMATION.Text) Then
        Bon = False
End If
```

Ce code crée un booléen à `true` au début, il vérifie si le texte entré `is nothing`, donc est nul, ou `not isnumeric()`, donc n'est pas un numérique.

Si l'une de ces deux conditions est vérifiée (autrement dit crée 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 puisqu'elle a une forme universelle. Je m'explique : on lui passe les valeurs nécessaires 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.

Voici 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 prend 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 possibilités pour arriver au même résultat sans faire les mêmes choses.

Vous pourriez faire évoluer ce programme, par exemple :

- Gérer un message d'erreur pour chaque textbox ;
- Personnaliser le message d'erreur (vide ou mauvaise valeur) ;
- Créer un bouton « Effacer » qui remet à 0 toutes les valeurs et cache les messages d'erreur.

Dites-vous que ce programme est déjà très bien, il vous apprend à interagir avec les contrôles, utiliser les fonctions, arguments, retours et réactions à une possible erreur. Vous avancez vite !

- Le mot-clé `Handles` permet d'écouter les événements.
- Les événements peuvent être écoutés sur des objets différents : fenêtre principale, composant graphique, pointeur de souris, etc.
- Chaque événement écouté appelle une fonction.

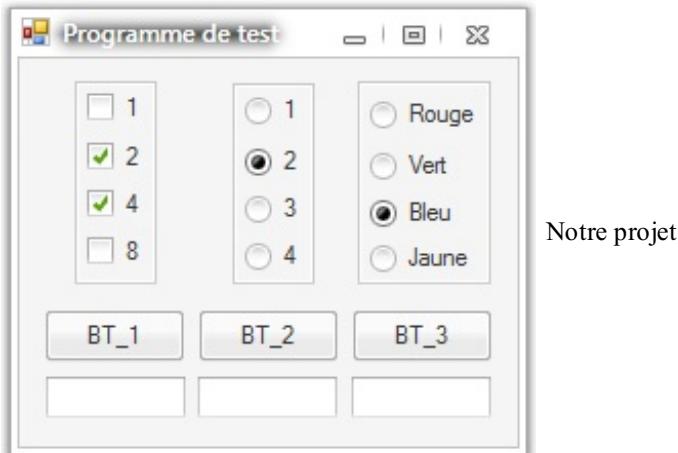
Les contrôles spécifiques

Vous savez désormais comment vous servir des contrôles basiques : les textbox, les labels, les boutons, etc. Mais qu'en est-il pour des contrôles plus spécifiques, mais non moins intéressants ? Vous allez sûrement vouloir faire un peu plus que mettre des boutons et des textbox dans vos programmes.

Je parle des checkbox et des boutons radio entre autres. Comment s'en servir ? C'est ce que nous allons voir.

Checkbox, boutons radio

Créons d'abord un nouveau projet (gardez le TP « voyage en voiture » dans un coin, ça pourrait toujours vous servir 😊), qui va ressembler à la figure suivante.



Notre projet

Je vous donne les noms des composants que j'ai utilisés :

- Les checkbox (à gauche) :

- CHK_1
- CHK_2
- CHK_4
- CHK_8

- Les boutons radio (au centre) :

- RB_1
- RB_2
- RB_3
- RB_4

- Les boutons radio (à droite) :

- RB_ROUGE
- RB_VERT
- RB_BLEU
- RB_JAUNE

- Bouton BT_1
- Bouton BT_2
- Bouton BT_3
- Textbox TXT_CHK
- Textbox TXT_RBNB
- Textbox TXT_RBCOL

Vous n'êtes évidemment pas obligés de travailler avec les mêmes noms que moi, mais je vous le conseille, ce sera plus facile pour vous de vous y retrouver.

Si vous testez ce petit programme, vous pouvez cliquer sur les cases, elles s'allument bien ; seulement, problème du côté des boutons radio : cliquer sur n'importe lequel d'entre eux en décoche un 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.

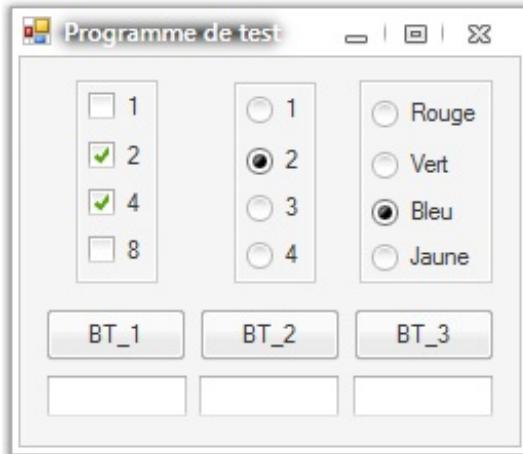
Comment faire ?

Eh bien, retournez sur votre IDE et cherchez le contrôle **groupbox**. Entourez grâce à deux groupbox vos deux colonnes de boutons radio, et allez dans les propriétés des deux 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. Vous devriez obtenir la figure suivante.



Le programme fonctionne

On peut sélectionner un bouton dans chaque colonne. 😊

La pratique

Bon, le côté design fonctionne, on va passer à l'accès aux propriétés.

Allons donc du côté du code VB en double-cliquant sur **BT_1**, ce qui créera notre événement de clic sur le bouton.

Dans cet événement 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 textbox **TXT_CHK** la somme des cases cochées.

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.

Ce n'était pas sorcier !



Je vous livre un secret : la propriété pour voir quel bouton radio est coché est la même !

Alors, à vos claviers ! Écrivez dans la seconde textbox quel bouton a été coché et dans la dernière la couleur sélectionnée !

Je vous laisse quand même réfléchir !

Solution

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_RB1.Text = Me.RB_1.Text
        End If
        If Me.RB_2.Checked Then
            Me.TXT_RB2.Text = Me.RB_2.Text
        End If
        If Me.RB_3.Checked Then
            Me.TXT_RB3.Text = Me.RB_3.Text
        End If
    End Sub

```

```

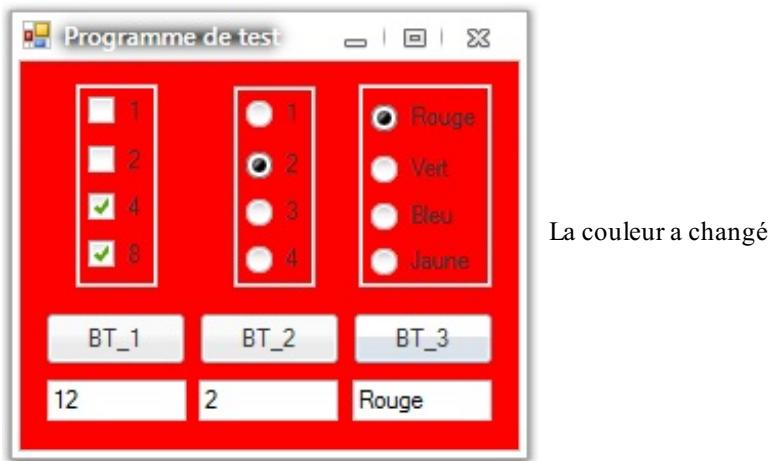
End If
If Me.RB_4.Checked Then
    Me.TXT_RBNB.Text = Me.RB_4.Text
End If
End Sub

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

```

Ce code est lourd puisqu'il vérifie toutes les checkbox une par une... De plus, je n'ai mis aucun commentaire. Mais bon il fonctionne et vous avez réussi à accéder et réagir aux checkbox et boutons radio. Essayez donc de le simplifier à coup de **IIF** !

Petit plus : la couleur (voir figure suivante). Vous auriez dû vous douter que je ne mettais pas des couleurs juste comme ça 😊, et la propriété, vous auriez pu la trouver par vous-mêmes !



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 aux checkbox et aux boutons radio, et les utiliser !

Les combobox

Bon, attaquons les **combobox** (aussi appelées `DropDownList`), que vous retrouvez souvent. Il s'agit de boîtes déroulantes. Nous allons apprendre à les remplir et à réagir avec.

Créez donc la fenêtre visible à la figure suivante : une combobox nommée `CB_CHOIX`, et une textbox appelée `TXT_CHOIX`.

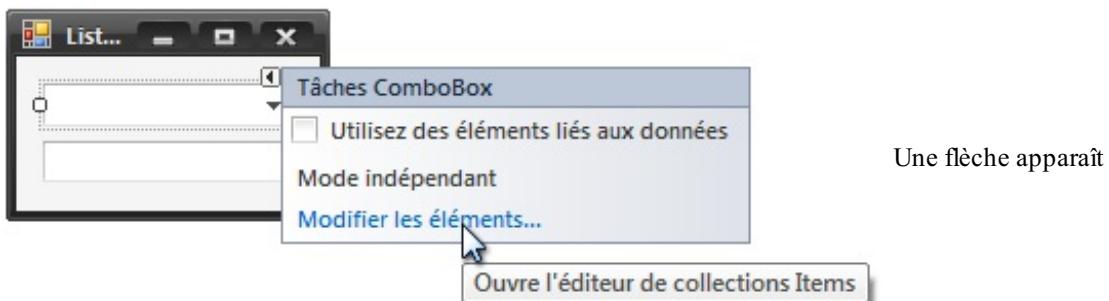


Cette fois, au lieu d'utiliser un bouton pour déclencher l'événement, nous allons utiliser l'événement propre de la combobox. Cet événement se déclenche lors du changement de sélection.

Tout d'abord il faut attribuer des valeurs à la combobox, deux choix s'offrent à nous : la manuelle (en dur dans le code) ou l'automatique (grâce à l'assistant de l'IDE). Je vais d'abord vous expliquer l'automatique, puis la manuelle qui offre beaucoup plus de possibilités.

Méthode assistée

Lors du clic sur la combobox (dans l'IDE), elle apparaît sélectionnée et une petite flèche apparaît en haut à droite de cette sélection, comme à la figure suivante.



Cliquez maintenant sur `Modifier les éléments` pour lui en attribuer.

Méthode manuelle

La seconde méthode nous amène côté VB, double-cliquez sur la fenêtre pour créer l'événement `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 (également 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 à une combobox et à la remplir !

MicroTP

Bon, pour vérifier vos connaissances sur les accès aux propriétés et l'utilisation de nouveaux contrôles, je vais vous demander de réaliser un petit programme contenant une **progressbar** et une **trackbar**.

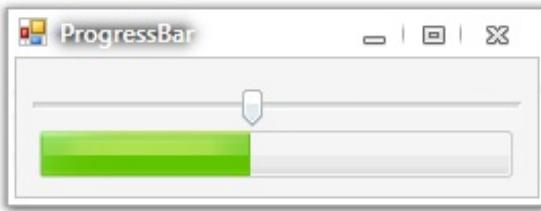
Le déplacement de la trackbar par l'utilisateur se répercutera sur le remplissage de la progressbar : si la trackbar est au milieu, la progressbar aussi.

Ce petit TP vous apprendra à trouver par vous-mêmes 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

La figure suivante vous montre le résultat.



Une barre de progression

Alors, une seule ligne côté VB à ajouter dans l'événement 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 ! La propriété à utiliser était **value**. Vous avez dû avoir des surprises aux premiers tests, 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 deux 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 checkbox sont utiles pour les choix multiples.
- Les boutons radio permettent un choix unique dans une liste.
- Les combobox sont utiles pour choisir une valeur dans une liste déroulante.

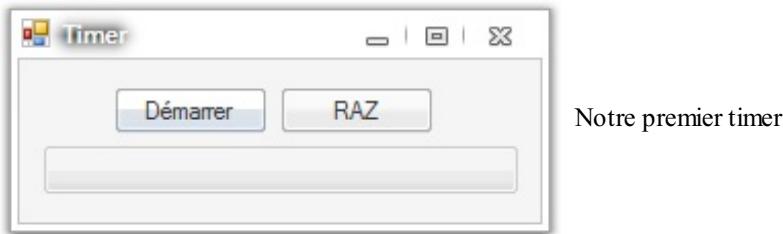
Les timers

Un timer nous sera très utile pour effectuer des actions temporelles et réagir à des événements temporels. Reprenons l'horloge réalisée dans la première partie : avec un timer, on pourrait prendre la date actuelle et ajouter une seconde toutes les secondes. Même effet, mais pas de la même façon.

Le **timer** est un contrôle comme n'importe quel bouton ou textbox, mais au lieu de pouvoir le placer où 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 à la figure suivante.



Donc, nous avons deux 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. À l'appui sur RAZ, elle va retourner à 0.

Le timer contient deux propriétés essentielles : **enabled**, comme pour tous les autres contrôles, détermine s'il est activé ou non, et la propriété **interval** (ce n'est pas une marque de cigarettes, non) détermine l'intervalle entre deux actions du timer(exprimée en ms).

Mettons donc pour ce TP 20 ms d'intervalle.

À chaque fois que ce temps sera écoulé, l'événement du timer, nommé **tick**, se déclenchera. Pour créer cet événement sur l'assistant, double-cliquez sur le timer, en bas. Faites de même pour les événements des deux boutons.

Nous avons donc créé trois événements dans notre code : le timer et les deux boutons.

Je pense que vous êtes capables de faire cet exercice seuls, avec tout ce que vous savez, mais je vais quand même vous le corriger.

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

```

```

    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_TICK.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 cela, vous voyez que je me sers de l'événement du timer, donc déclenché toutes les 20 ms 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 avancera plus vite.

TP : la banderole lumineuse

Petit TP : la banderole lumineuse ! Je sais, le nom n'est pas très imaginatif...

Le but de ce TP va être d'allumer différents boutons radio (une dizaine) au rythme du timer, les faire défiler en gros. J'ai pris des boutons radio et pas des checkbox, parce que les boutons radio n'ont pas besoin d'être décochés, ils le sont automatiquement lorsqu'un autre est coché.

Donc un bouton « Démarrer la banderole » 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.*

À vos claviers !

Solution

Mon programme ressemble à la figure suivante.



Le rendu final

Il y a bien les 10 boutons radio.

Maintenant le code :

Code : VB.NET

```

Public Class Form1
    Private Sub 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 boutons radio 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 reprend 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 principal :

Vous voyez que dans l'événement `tick` du timer j'ai créé un tableau, mais ce n'est pas un tableau de `string` ou de `integer`, non c'est un tableau de... boutons radio (`Dim RB(9) As RadioButton`) !

Eh, oh ! je savais pas moi !

C'est pour ça 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 boutons radio, je le remplis avec mes boutons !

Et donc si vous avez compris, la boucle en dessous est un petit algorithme qui parcourt 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'intervalle d'un timer ne doit jamais être égal à 0 !

Pour finir ce chapitre, je tiens 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 lourd, 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 boutons radio. 😊

- Les timers nous permettent de déclencher des événements en fonction du temps.
- L'événement appelé à chaque fois que le timer est écoulé est Tick.
- Le temps entre chaque Tick est défini avec la propriété Interval.

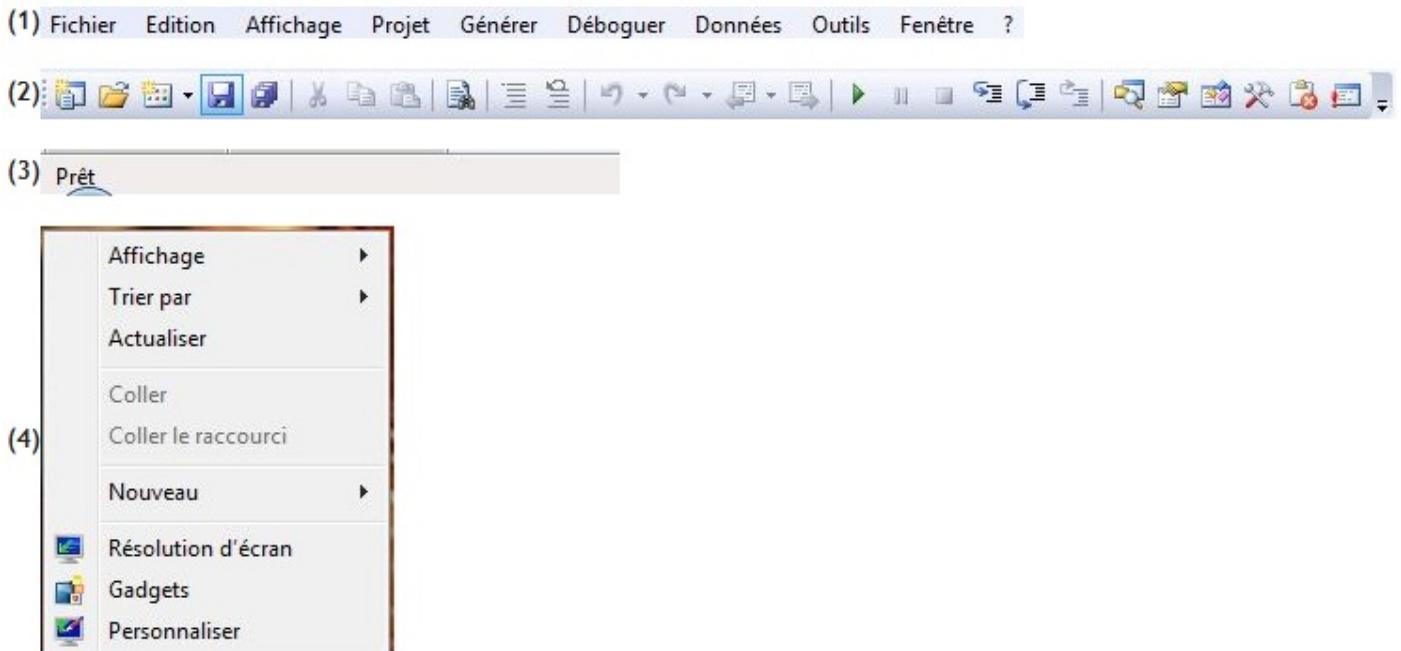
Les menus

Dans ce chapitre, nous allons parler d'un élément important : les menus. Vous savez, les menus, la barre en haut de votre navigateur favori par exemple, avec Fichier, Édition, etc. Et celle juste en dessous, avec les images (la barre d'outils) !

Encore une fois, l'IDE nous mâche le travail ; vous allez voir !

Présentation des menus

Vous devez voir dans votre boîte à outils un sous-menu Menus et barres d'outils, semblable à la figure suivante. Comme vous pouvez le constater, ces objets nous permettront 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).



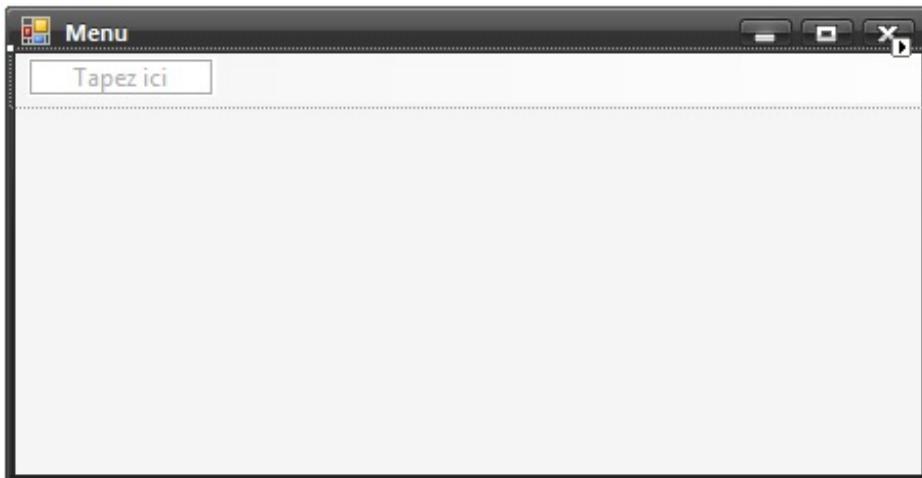
Il est possible de créer toutes sortes de menus

Passons tout de suite au menu le plus intéressant : la barre de menus (1) !

La barre de menus

Création graphique

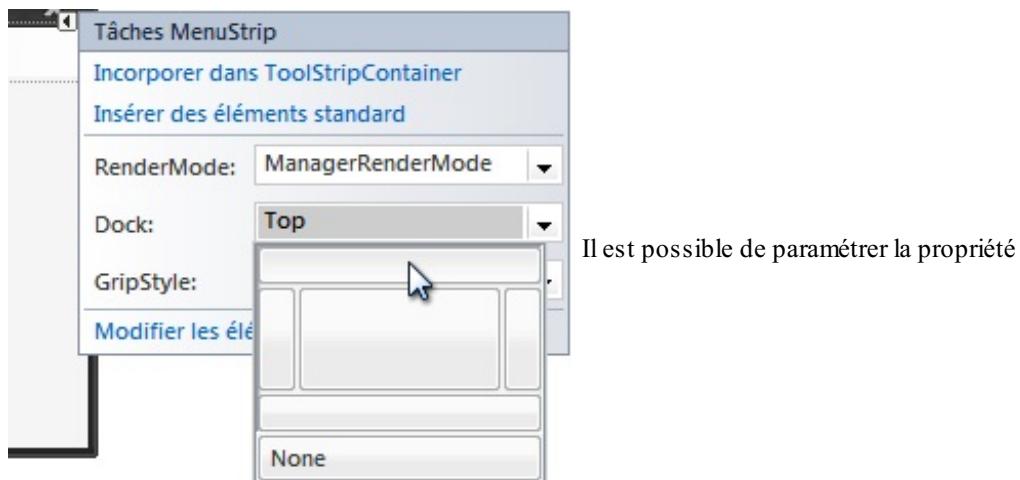
Comme je vous l'ai dit, L'IDE va grandement nous mâcher le travail : un assistant est fourni avec pour créer ces menus. Prenez l'objet **MenuStrip** et insérez-le sur votre feuille (feuille vide de préférence), comme à la figure suivante.



Insérez l'objet MenuStrip sur votre feuille

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, comme à la figure suivante.



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 c'est 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 deux 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.

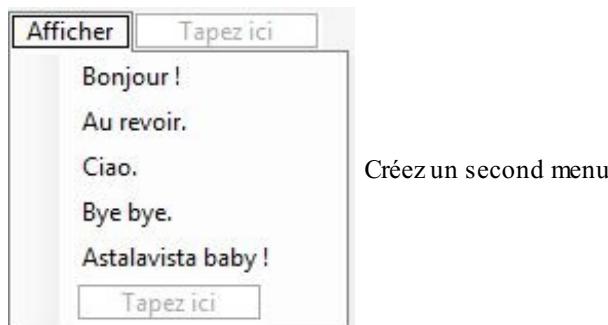


Il y a encore des « Tapez » qui apparaissent, je fais quoi ?

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 (voir figure suivante), faites de même.



Puis, pour finir un petit label au centre de la feuille : LBL_TEXTE.

Événements

Maintenant, attaquons la gestion des événements !

Ces événements 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 un 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 par vous-mêmes, 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
```

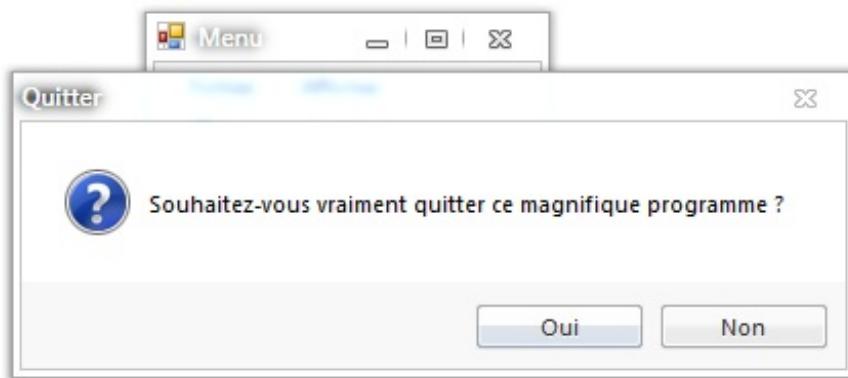


Pourquoi 36 en deuxième argument ?

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, comme le montre la figure suivante.



Le programme demande confirmation avant de se fermer

Les différents contrôles des menus

Je viens de vous montrer un menu classique avec du texte comme contrôle, mais vous en voulez sûrement plus. Eh bien, c'est parti : nous allons créer des combobox (listes déroulantes) et des textbox.



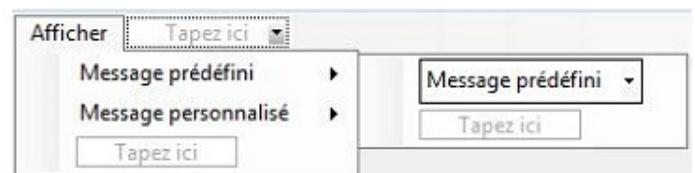
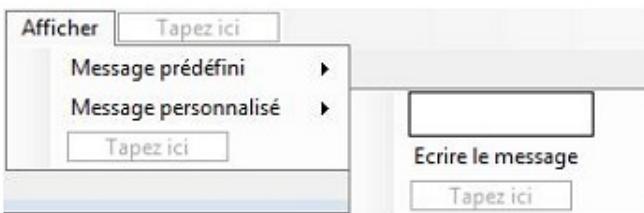
Dans le menu ??

Eh bien oui ! Vous ne devez pas en voir souvent, mais ça peut être utile !

Donc, pour avoir accès à ces contrôles supplémentaires, il faut cliquer sur la petite flèche disponible à côté du « Tapez ici » (voir figure suivante).



Vous voyez que s'offrent à vous les contrôles tant désirés ! Eh bien, personnalisons un peu notre menu pour arriver à la figure suivante.



Ce que nous devons obtenir

Sachant que dans la combobox `Message_prédefini` j'ai remis les messages d'avant (vous devez vous servir de la propriété `collection` de cette combobox, du côté design, pour en assigner les choix ou alors passer par le code VB, au choix).

Schématiquement :

- Fichier
 - Reset
 - Quitter
- Affichage
 - Message prédefini
 - Combobox
 - Bonjour !
 - Au revoir.
 - Ciao.
 - Bye bye.
 - Astalavista baby !
 - Message personnalisé
 - Textbox
 - Écrire

Ce qui est assez gênant avec cet assistant, c'est que les noms qui sont entrés automatiquement sont assez coton à repérer ; avec une textbox, une combobox, ça passe, mais au-delà, aïe ! 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 événements correspondants : sur le clic du bouton Écrire et lors du changement de la combobox.

Si vous avez utilisé l'assistant pour créer l'événement de la combobox, lorsqu'elle est dans un menu, l'événement 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 événements 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'événement **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
    MeLBL_TEXTE.Text = Me.CB_MENU.SelectedItem
End Sub
```

Bon, pour notre bouton Écrire, ce n'est pas sorcier : on récupère la valeur de la textbox 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
    MeLBL_TEXTE.Text = ""
End Sub

Private Sub QuitterToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
QuitterToolStripMenuItem.Click
    'Fermeture avec confirmation
    If MsgBox("Souhaitez-vous vraiment quitter ce magnifique
programme ?", 36, "Quitter") = MsgBoxResult.Yes Then
        End
    End If
End Sub

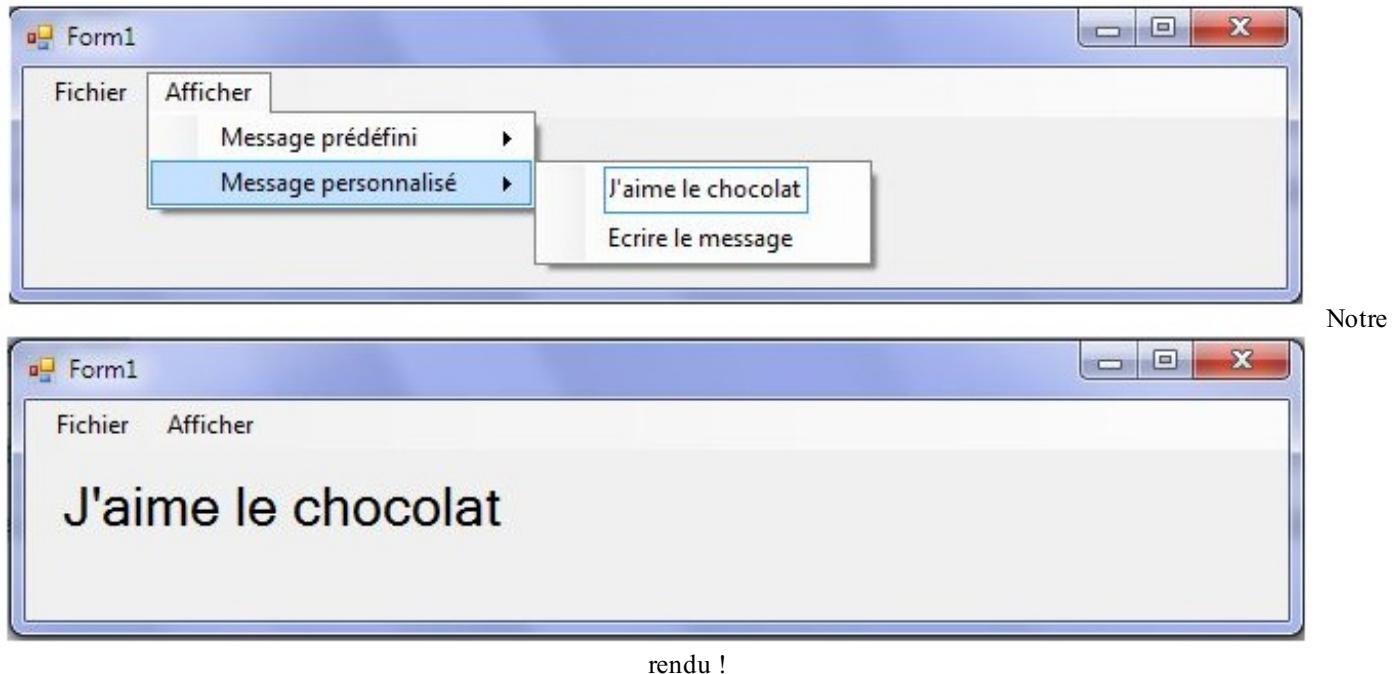
Private Sub CB_MENU_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
CB_MENU.SelectedIndexChanged
    'Écrit le texte de la combobox lors du changement d'index
    MeLBL_TEXTE.Text = Me.CB_MENU.SelectedItem
End Sub
```

```

Private Sub EcrireToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
EcrireToolStripMenuItem.Click
    'Écrit le texte de la textbox lors de l'appui sur « Écrire »
    >>
    Me.LBL_TEXTE.Text = Me.TXT_MENU.Text
End Sub
End Class

```

Et voici le rendu final à la figure suivante.



rendu !

Comme vous voyez, le VB est assez facile à 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. Il s'agit de la barre qui vous indique... le statut de l'application. À quoi va nous servir cette barre ? Eh bien à afficher le statut de votre application par exemple, ou alors tout simplement à mettre des boutons dessus ! 🍫 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 !

Créons déjà ladite 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 deux contrôles : un label et une progressbar. La progressbar est accessible, comme pour la combobox de la partie précédente, avec la petite flèche. Renommez-les : 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. Lors 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.

Représons donc un petit timer pour simuler le temps d'attente (TIM_STATUT) et utilisons le même procédé que lors du 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 confirmation
    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
    'Écrit le texte de la combobox lors 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
    'Écrit le texte de la textbox lors de l'appui sur « Écrire
»
    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

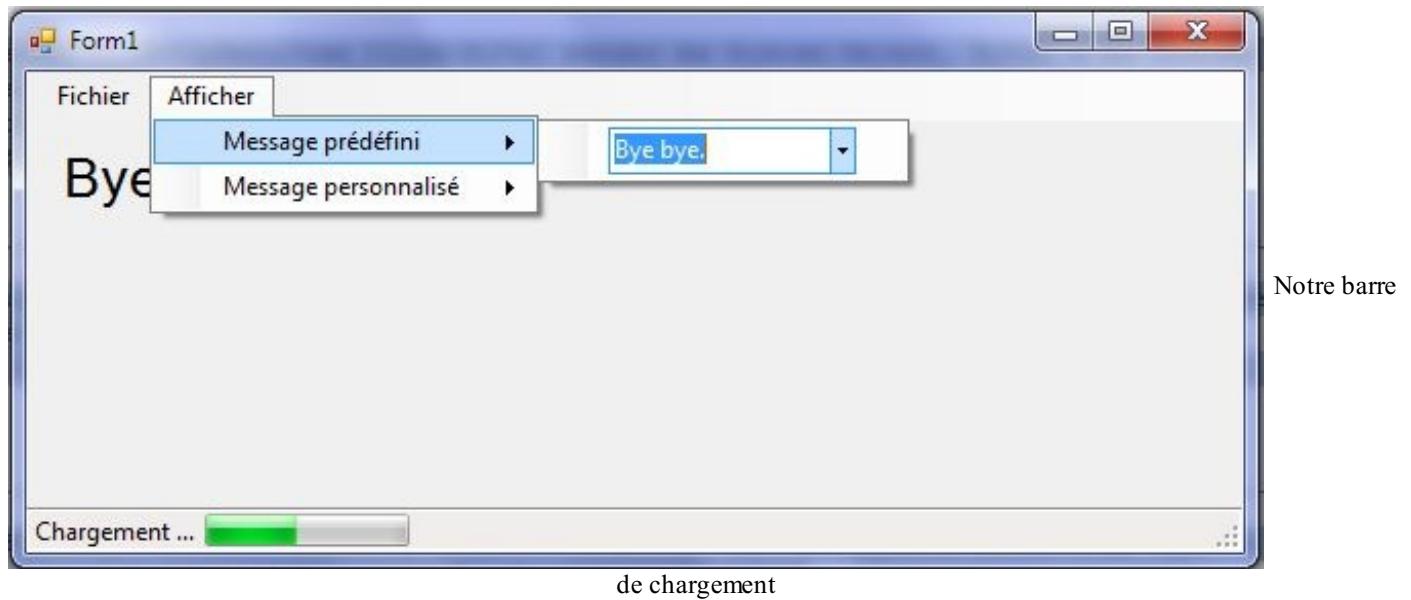
    Private Sub TIM_STATUT_Tick(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles TIM_STATUT.Tick
    'Si la progressbar est arrivée au bout, on désactive le
timer
    If Me.PGB_STATUT.Value = 100 Then
        Me.TIM_STATUT.Enabled = False
        LBL_STATUT.Text = "Prêt"
    Else
        '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 le code du chapitre sur les timers. La « pause » n'est pas effectuée donc le texte s'affiche pendant la progression. C'est un exemple, vous pourrez utiliser les barres de chargement en situation réelle plus tard. L'idéal aurait été de placer un sémaphore (un flag), le tout avec une boucle **While**. Le rendu se trouve à la figure

suivante.



Alors ce code n'est pas dur à 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.

Le menu contextuel

Alors, le menu contextuel est, comme je vous l'ai expliqué, le menu visible lors du clic droit. 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 boîte à outils : Menus et barres d'outils, vous prenez le **ContextMenuStrip** et vous l'intégrez à la feuille. Une fois cela fait, créez un élément contenant le texte : « Déplacer le label ici ». Ensuite, comme à l'accoutumée, on crée son événement correspondant. Dans cet événement, nous allons récupérer la position du curseur et changer la propriété **location** du label :

Code : VB.NET

```
MeLBL_TEXTE.Location = Control.mousePosition
```

Control.mousePosition est la propriété de 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()
        MeLBL_TEXTE.Text = ""
    End Sub

    Private Sub QuitterToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles QuitterToolStripMenuItem.Click
        'Fermeture avec confirmation
        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
    'Écrit le texte de la combobox lors du changement d'index
    PauseFactice()
    MeLBL_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
    'Écrit le texte de la textbox lors de l'appui sur « Écrire »
    PauseFactice()
    MeLBL_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.
    MeLBL_TEXTE.Location = Control.mousePosition
End Sub
End Class

```

Lors d'un clic et de la sélection, le label bouge. Ce programme était là pour vous montrer les utilisations des différents menus, il est bien évidemment inutile (donc indispensable) !

- Il existe des menus contextuels, des barres de menus, des barres d'outils...
- Le menu contextuel apparaît lors du clic droit de la souris.
- Les menus sont réservés aux applications volumineuses et nécessitant une grande interaction avec l'utilisateur.

TP : navigateur web

Vous êtes prêts pour créer un vrai programme, utilisable et fonctionnel. Je vous propose donc de réaliser votre propre navigateur web ! Je pense aussi que ce petit exercice vous donnera envie de l'améliorer, le faire évoluer, bref, le personnaliser à votre guise.

Le cahier des charges

Vous allez voir, avec ce que j'ai expliqué jusqu'à maintenant, vous allez pouvoir faire quelque chose de sympa, que nous améliorerons plus tard !

Mais avant toute chose, voici un contrôle qui va nous être *indispensable* pendant ce TP : **WebBrowser**. Pour ceux qui sont nuls en anglais, ça veut dire « navigateur web ». Ce contrôle va nous permettre de créer notre navigateur : vous lui entrez une adresse et il affiche ce qu'il y a dans la page. Il utilise le même moteur web qu'Internet Explorer. Le menu contextuel est déjà géré par ce contrôle ainsi que le téléchargement de fichiers. Vous l'avez compris, nous allons créer l'interface.

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 à utiliser et de chercher comment l'utiliser en suivant la syntaxe donnée.

Il va falloir chercher un peu, c'est sûr, mais vous devrez le faire pour vos propres programmes, alors autant commencer tout de suite.

Pour ce qui est de l'interface, nous allons commencer doucement, je ne vais pas vous demander l'impossible : une barre d'adresse avec son bouton « Envoyer », « Précédent », « Suivant », « Arrêter », « Rafraîchir », le statut de la page (terminé, en chargement, etc.), le menu « Fichier », et « Quitter ».

Un dernier conseil : la méthode URL du WebBrowser sera sûrement utile.  Bonne chance !

Les ébauches

J'espère que vous avez passé au moins quelques minutes à chercher. 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 deux 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 bouton « 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
    MeLBL_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
        'S'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 à 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 à 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
        'Désactive le bouton « 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
        'Rafraîchit 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

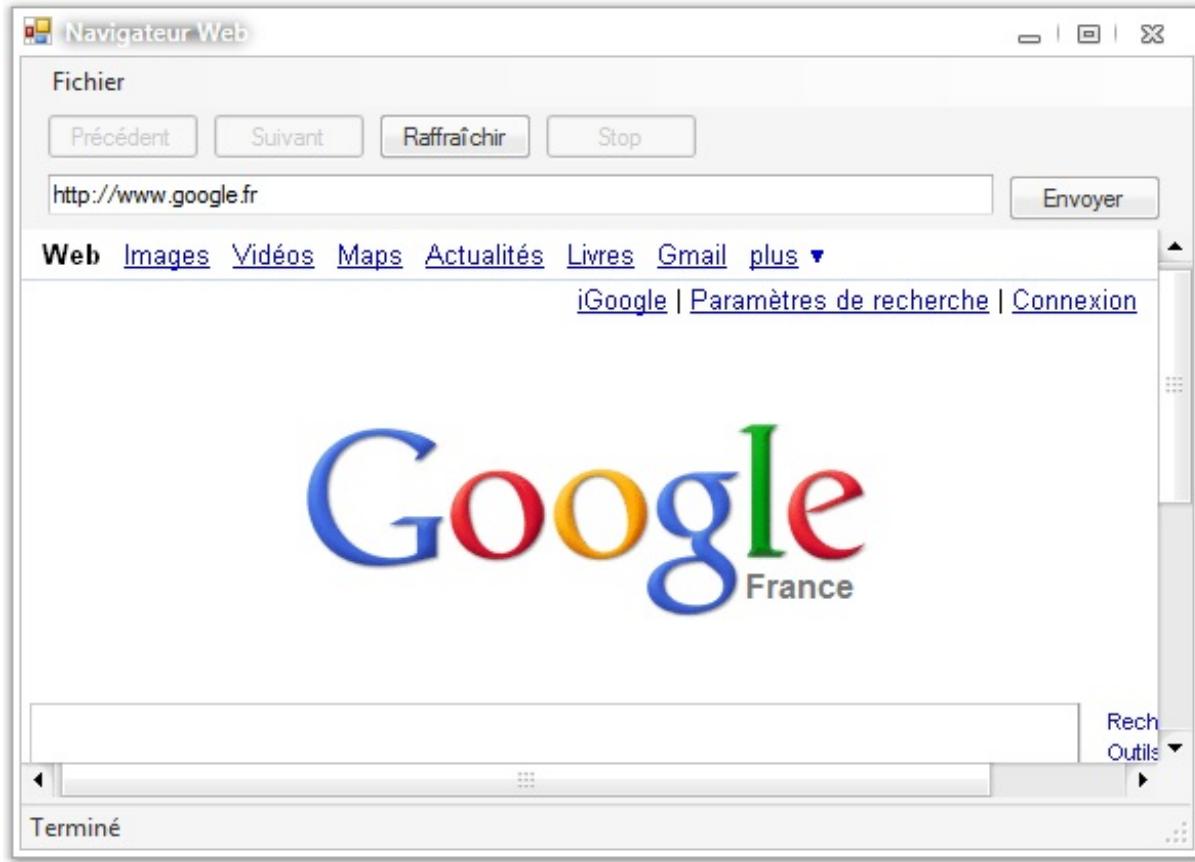
```



Ça y est, il y a plein de trucs que je ne comprends pas, c'est quoitès #region par exemple !?!

Pas de panique, votre code marche parfaitement sans, ça sert seulement à créer une zone rétractable où l'on peut mettre les fonctions dont on est sûr du résultat pour plus de lisibilité.

Revenons à nos moutons, la partie design. La figure suivante représente ce que j'ai obtenu.



Mon navigateur

web

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 patience, les plus pressés d'entre vous auront craqué et seront passés directement à cette partie je pense. 😊

Je vais vous les lister avec mes noms d'objets (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 a une page suivante (si vous avez fait précédent auparavant) ;
- **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 à l'adresse de la page passée en argument (ici le texte de TXT_ADRESSE) ;
- **Me.WB_NAVIGATEUR.GoBack ()** : le navigateur va à 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 ()** : rafraîchit la page.

Comme vous l'avez remarqué dans le code, j'ai deux événements 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 deux événements 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 événements de notre contrôle (eh oui, les fonctions c'est bien beau, mais les événements, c'est magnifique 😊).

Bien exploiter les événements

Bon, je ne sais pas si vous avez prêté attention à tous les événements 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 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, qui 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 événements pour activer ou désactiver les boutons « Précédent » et « 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, seulement pour la gestion des événements du navigateur :

Code : VB.NET

```
#Region "Evènements du WBrosert"

'À 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

'À 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_StatusTextChanged(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
    'on désactive le bouton « 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 ai-je fait pour réaliser cette prouesse ?

Eh bien si vous avez tapé « `e` » dans l'événement du changement de progression, votre IDE vous fournit les fonctions, objets et propriétés pouvant être utilisés. Vous voyez deux lignes qui s'écartent du lot :

- `CurrentProgress`;
- `MaximumProgress`.

En mettant le curseur dessus, votre IDE vous 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` à télécharger pour avoir la page et le `CurrentProgress`, le nombre de `bytes` actuellement téléchargés.

Ensuite, il ne faut pas sortir de Saint-Cyr pour savoir ce qu'il faut faire : on attribue le nombre de bytes à télécharger en tant que maximum pour la progressbar, et on met comme valeur le nombre de bytes actuellement téléchargés.

Et on obtient notre premier événement dans lequel on exploite les arguments transmis par `e`.

Si, en remplaçant `StatusTextChanged` par `ProgressChanged` dans ce code :

Code : VB.NET

```

Sub WB_NAVIGATEUR_StatusTextChanged(ByVal sender As Object, ByVal e
As EventArgs) Handles WB_NAVIGATEUR.StatusTextChanged

```

... une erreur inconnue au bataillon est apparue, pas de panique ! Certains événements 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 événements des 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 réagirez avec des objets, c'est ce `e` qui gérera les retours d'événements.

Le design

Je vous l'accorde, tel quel, notre programme ne donne pas vraiment envie. Nous allons donc l'améliorer un peu visuellement. J'ai 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](#).

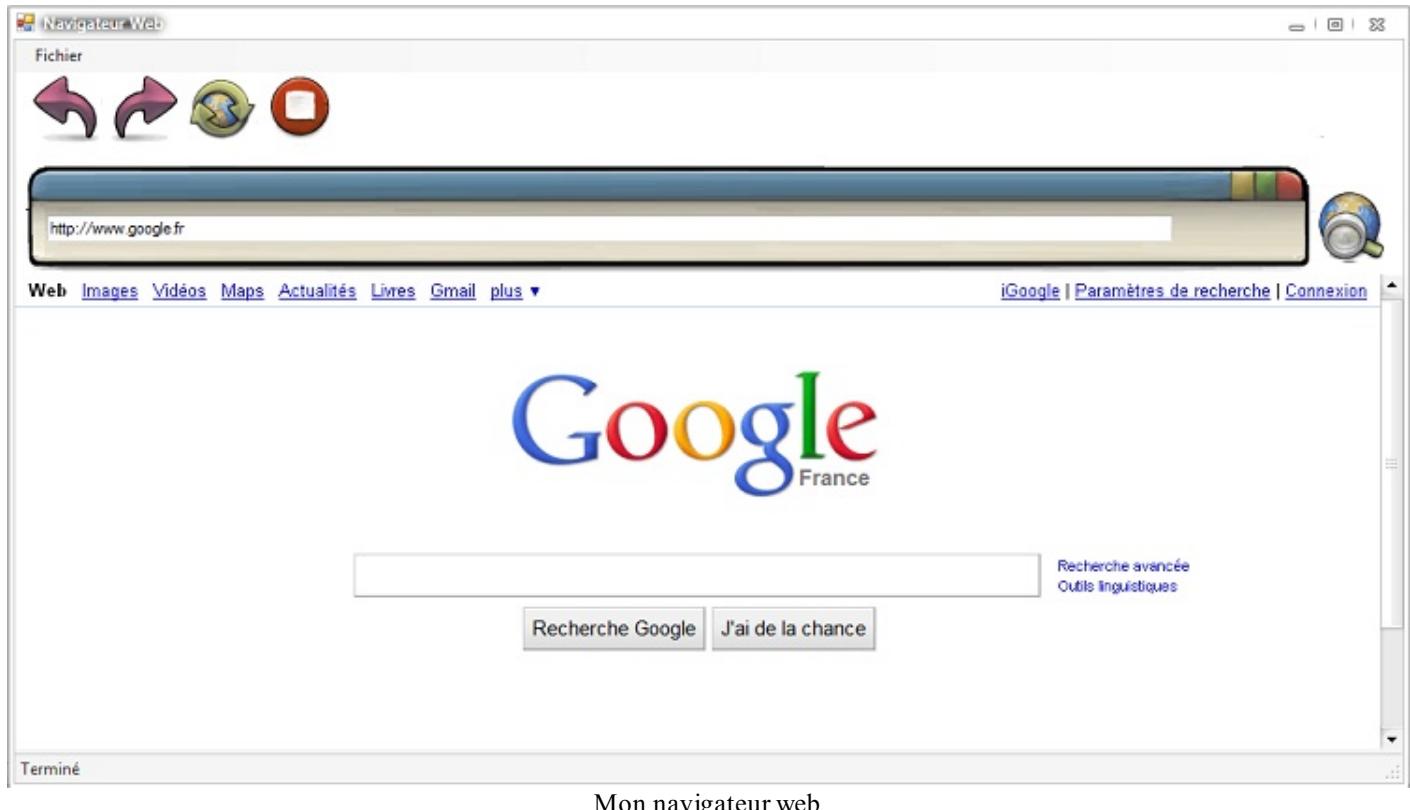


Le premier sera pour le bouton d'envoi, le second pour le rafraîchissement de page, ensuite le « Stop », « Suivant », puis « Précédent ». Vous êtes bien évidemment libres d'en choisir d'autres.

Nous allons donc intégrer une image à nos boutons, pour ça il faut agir sur la propriété... **Image** ! Lorsque vous allez vouloir choisir une image, une fenêtre vous propose deux choix : utiliser une ressource locale ou un fichier de ressources, la différence : le fichier de ressources rassemble toutes les ressources : 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 ressources. Cliquez donc sur le petit **Importer** en bas et choisissez vos images. Attribuez les bonnes images aux 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.

Après, à vous de toucher un peu les propriétés de la page, des éléments comme vous le sentez pour les adapter à vos goûts. La figure suivante vous présente mon résultat final.



Pas de moqueries, je ne suis pas graphiste. Je vais quand même vous donner mon image d'arrière-plan (si certains osent la prendre 😊).



Mon image d'arrière-plan

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-la de côté, un prochain chapitre consacré à son amélioration viendra quand j'aurai apporté de nouveaux concepts et de nouvelles connaissances.

- On utilise la navigation web avec le composant WebBrowser.
- Ce composant peut vous permettre de télécharger le texte ou le code source d'une page.
- Pour gérer le WebBrowser, il faut utiliser ses événements.

Fenêtres supplémentaires

Vous devez commencer à en avoir assez de travailler avec une seule et unique fenêtre. La plupart des programmes que vous utilisez en ont plusieurs. Ne serait-ce que pour afficher une fenêtre d'options, une fenêtre de visualisation et bien d'autres cas de figure.

Dans ce chapitre nous allons donc nous pencher sur l'ajout de fenêtres à notre projet, interagir avec elles, et leur faire échanger des données. Cela ouvrira la porte à des programmes plus complexes.

Plutôt que de regrouper des dizaines de fonctionnalités dans la même fenêtre, imaginez les répartir entre plusieurs : chaque feuille de code sera épurée et aura sa propre fonction. Voilà un des intérêts des fenêtres multiples.

Je ne vous en dit pas plus, à l'attaque !

Ajouter des fenêtres

Nous allons commencer par ajouter des fenêtres supplémentaires, puis nous apprendrons à les faire communiquer entre elles ! Créons 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 maintenant très importants.

Renommons donc cette fenêtre principale. Appelons-la `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`.

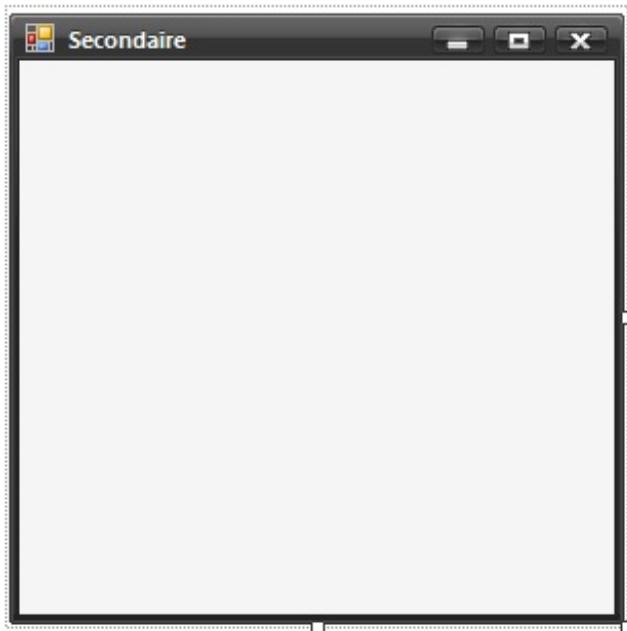
Puis renommons la feuille contenant cette fenêtre que nous voyons dans la fenêtre de solutions. Faites-un clic droit sur `Form1.vb`, puis renommez-la : inscrivez à la place `Main.vb` (voir figure suivante).



Voilà, vous venez de renommer entièrement votre fenêtre. Il faudra faire de même avec les supplémentaires.

Ajoutons une seconde fenêtre maintenant. Faites un clic droit sur le nom de votre projet (pour moi : `FenetresSupplementaires`), déplacez votre souris sur le menu `Ajouter` qui nous donnera accès à `Nouvel élément` et cliquez dessus. Une fenêtre proposant différents types d'éléments à ajouter à votre projet apparaît. 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 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 modifications de renommage que nous avons effectuées sur la fenêtre `Main` ont été automatiquement effectuées sur cette deuxième fenêtre (voir figure suivante).



Les modifications ont été effectuées

Vous voilà avec votre première deuxième fenêtre (dur à suivre 😊). Allons nous amuser avec elle !

Ouverture et 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, un bouton **Fermer** par exemple.



On vient à peine de la créer et 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 (voir figure suivante).



La fenêtre secondaire

Créons donc un bouton **Ouvrir** identifié **BT_OUVRIR** sur la fenêtre principale (**Main**). Vous pouvez y accéder soit grâce au système d'onglets si vous ne l'avez pas fermée, soit grâce à la fenêtre de solutions en double-cliquant sur son nom.



Un bouton pour ouvrir une fenêtre

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'événement **BT_FERMER_CLICK** en double-cliquant sur le bouton.



L'événement **Load** (ici **Secondaire_Load**) d'une fenêtre supplémentaire sera appelé à chaque fois que la demande d'ouverture de cette fenêtre sera formulée.

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 est 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 suivi, 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 événement correspondant.

Et insérons dans cet événement 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.` ?

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 teste, 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 votre nouvelle fenêtre, puis à la fermer.

Notions de parent et d'enfant



Pourquoi veux-tu nous parler de famille ?

Non, non. Le concept de parent et d'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 (abrégée en POO).

La POO est un style de programmation de plus en plus répandu. Certains langages ne sont pas orientés objet, d'autres le sont totalement. Notre langage, VB .NET utilise énormément la notion d'orienté objet.



Le langage VB6, quant 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 détails concernant la POO, je vous renvoie sur le tutoriel sur le C++ du **Site Du Zero** qui explique très bien le concept d'objet (réservé aux plus curieux d'entre vous).

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'appliquent sur les objets.

L'héritage

Cette notion a été introduite avec la notion d'**héritage**.



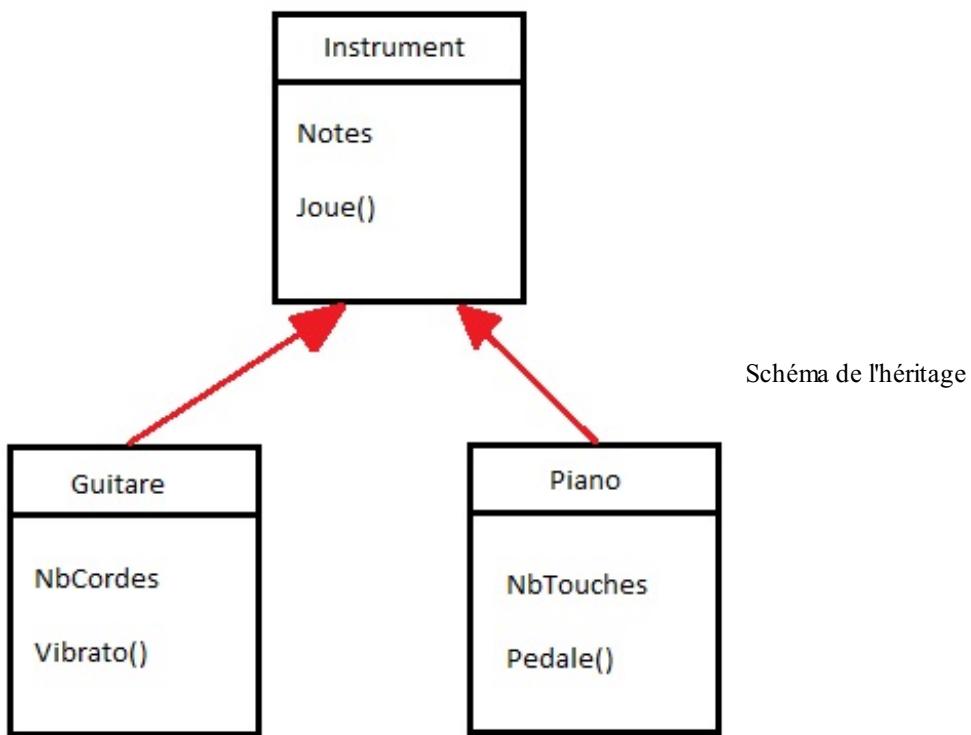
L'héritage existait bien avant l'informatique...

Mais moi je vous parle de l'héritage en informatique. C'est un concept qui s'applique aux objets.

Imaginez que vous ayez 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`.

Regardez le schéma à la figure suivante.



Donc, chaque rectangle est un objet. Vous voyez l'objet `Instrument`, l'objet `Guitare` et l'objet `Piano`. L'objet `Guitare` et l'objet `Piano` héritent tous 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

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 considérée comme la fenêtre *parent*, et donc toutes les fenêtres supplémentaires créées seront ses *enfants*.

Si vous avez suivi ce que j'ai expliqué, si je ferme la fenêtre `Main`, les autres fenêtres se fermeront également.

Donc il va falloir faire bien attention à ça dans nos programmes. Ne pas fermer la fenêtre principale !



Et si je ne veux pas la voir ?

Eh bien il vous suffit d'effectuer un `Visible = false` sur cette dernière (j'ai dit que ce n'est pas bien, mais ici vous êtes obligés). Mais attention avec ça, ce n'est pas le tout de cacher la fenêtre et de ne jamais pouvoir la réafficher.

Bon, avec toutes ces nouvelles notions, nous allons pouvoir attaquer la communication entre fenêtres.



Vous pouvez réouvrir les yeux, le cauchemar est terminé.

Communication entre fenêtres

Bon, après cette lourde partie théorique, attelons-nous à faire communiquer nos fenêtres entre elles.



Pourquoi diable aurais-je envie de faire ça ?

Eh bien, vous avez vu 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 principale. Puis inversement lors de la fermeture de la fenêtre secondaire.

Si vous avez bien appréhendé toutes les notions d'objets, vous devriez être capables de le faire cela par vous-mêmes.

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 événements 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()
        SecondaireLBL_SECOND.Text = "J'ai réussi !"
    End Sub

End Class
```

Fenêtre Secondaire :

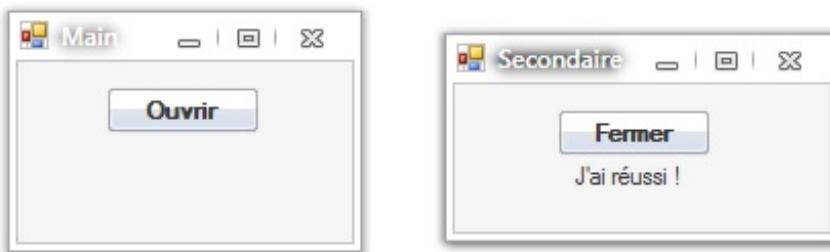
Code : VB.NET

```
Public Class Secondaire

    Private Sub BT_FERMER_Click(ByVal sender As System.Object, ByVal
        e As System.EventArgs) Handles BT_FERMER.Click
        MainLBL_MAIN.Text = "De ce côté également"
        Me.Close()
    End Sub

End Class
```

Et le résultat à la figure suivante.



Le résultat



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 déclarons nos variables avec un **Dim** à l'intérieur d'une fonction ou alors directement 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 créées comme je vous l'ai appris (**Dim X as Integer**), ces variables sont accessibles uniquement à partir de la fenêtre les ayant créées. Pour pouvoir y accéder depuis 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
        MeLBL_SECOND.Text = MonString
    End Sub

    Private Sub BT_FERMER_Click(ByVal sender As System.Object, ByVal
        e As System.EventArgs) Handles BT_FERMER.Click
        MainLBL_MAIN.Text = "De ce côté également"
        Me.Close()
    End Sub

End Class
```

Et voici 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.

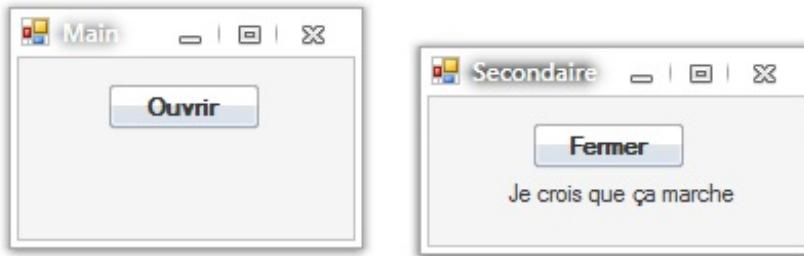
Code : VB.NET

```
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
```

Le résultat se trouve à la figure suivante.



Le résultat

Et voilà, votre première variable *globale publique*, et vous y avez déjà accédé à partir d'un autre objet !

- On crée une nouvelle fenêtre au projet en effectuant un clic droit, puis Ajouter.
- Une nouvelle fenêtre signifie un nouveau fichier.
- On peut faire interagir les fenêtres entre elles pour leur envoyer ou recevoir des variables.

Les fichiers - partie 1/2

Dans cette partie, nous allons commencer à travailler sur les fichiers. 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). 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

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.

Amis *Windowsiens*, vous pouvez apercevoir dans votre poste de travail des lecteurs et des disques durs. Dans le(s) disque(s) dur(s), vous pouvez naviguer suivant un système d'**arborescence** : le disque dur contient des dossiers, ces dossiers contiennent des fichiers (récuratif : 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 quant 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. En revanche, j'ai été méchant, j'ai ajouté 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 en binaire correspond à un chiffre hexadécimal).



Mais pourquoi nous dis-tu 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. 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.

Exemple : la première suite de bits du tableau plus haut (01100011) correspond au caractère « c ».



On retient que 1 caractère = 1 octet = 8 bits.



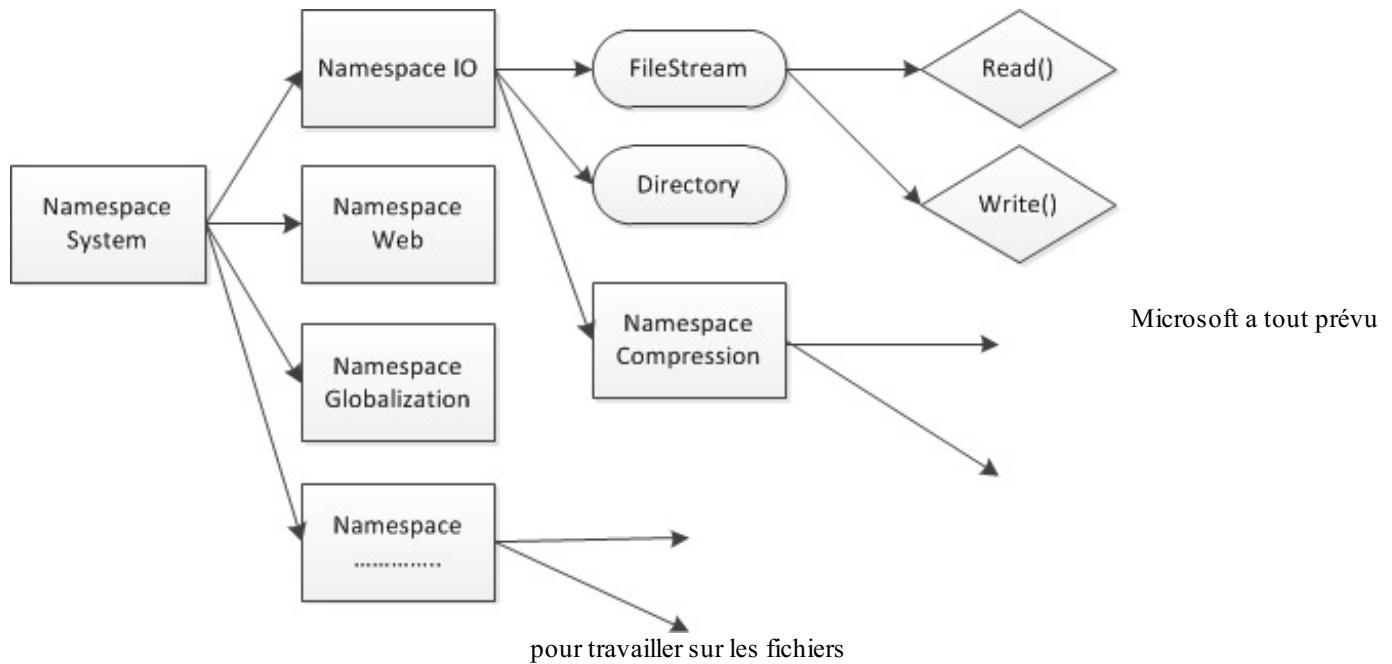
Bref, je ne vous demande pas d'apprendre la table ASCII par cœur, notre IDE se chargera d'effectuer les codages tout seul. Tout ça pour vous sensibiliser un peu quant à la taille de vos fichiers. Windows a l'habitude de noter les tailles en **ko** (kilooctets) pour les petits fichiers, jusqu'aux **Mo** (mégoctets), voire aux **Go** (gigaoctets) : 1 ko = 1024 octets, 1 Mo = 1 048 576 octets et 1 Go = 1 073 741 824 octets.

Donc 1024 caractères équivaudront à 1 ko.

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 bibliothèque contenant des centaines de classes, fonctions, objets) a développé tous les outils nécessaires pour travailler facilement sur les fichiers, comme le montre la figure suivante.



Ce namespace (un namespace est une sorte de dossier contenant des classes et 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 ça ? **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, il 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, eh bien ici c'est pareil avec les namespaces et les objets et 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 le manipuler ; pour cela nous allons découvrir un nouveau mot récurrent en programmation : **New**.

Notre premier fichier

Créez un nouveau projet et rendez-vous dans l'événement 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 allions créer un objet `FileStream`, pourquoi ne pas l'entrer dans une variable de type `FileStream` ?

J'ai donc créé 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 ! Vous reconnaisserez tous le `Dim` `MonFichier` `As` `IO.FileStream`, une simple déclaration de variable de type `FileStream`.



Pourquoi `IO.` devant ?

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 l'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 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 parfois ce constructeur nécessite des **arguments**.

Dans notre cas, le constructeur de `FileStream` accepte plein de « combinaisons » d'arguments possibles (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 ne 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 créé 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 créé, 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. À moins que vous ne gériez toutes ces éventualités. Mais nous sommes des Zéros, allons au plus simple.

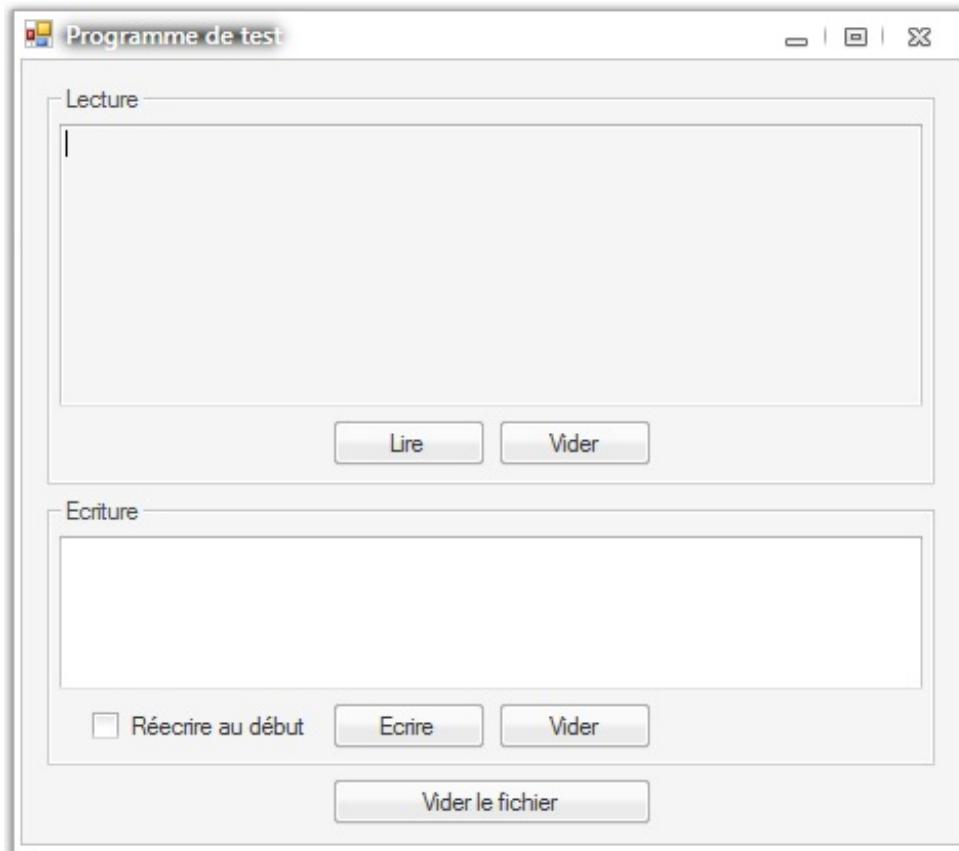
Résumé

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.

Nos premières manipulations

Programme de base

Bon continuons, créons une petite interface basique permettant de lire et écrire dans le fichier. J'aimerais donc que vous créiez quelque chose qui ressemble à la figure suivante.



Notre programme ressemblera 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).

Voilà 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 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 textbox
        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
        'Écrit 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 complexes. 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'événement **FormClosing** (traduisible par « fenêtre en cours de fermeture », à ne pas confondre avec **FormClosed** : « fenêtre fermée »). Lorsque cet événement 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 textbox
        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
        'Écrit 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 vérifications sont souvent inutiles, mais il ne coûte rien de les faire et elles peuvent parfois éviter des erreurs, pensez aussi à gérer les cas d'erreurs).
- 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 **Byte** et les **Char** ne sont pas homogènes, il faut passer par une fonction qui va récupérer la valeur binaire de notre caractère (*transformation ASCII => 8 Bits* grâce à la fonction `Asc()`) de façon à pouvoir l'inscrire dans le **Byte**.
- Vient 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 **Byte** 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.

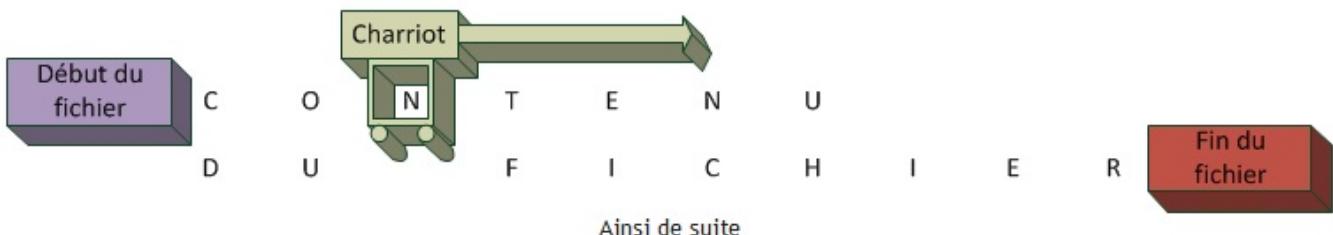
Alors je viens de parler de curseur dans notre fichier, mais qu'est-ce que cela ?

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 chariot qui avance dans notre fichier. Lorsqu'on lui demande de lire ou d'écrire, ce petit chariot 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.

Regardez le schéma présenté à la figure suivante.



Ici le chariot 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

Explications concernant le chariot

Bref, tout ça pour dire que ce petit chariot 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'événement 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
```

```

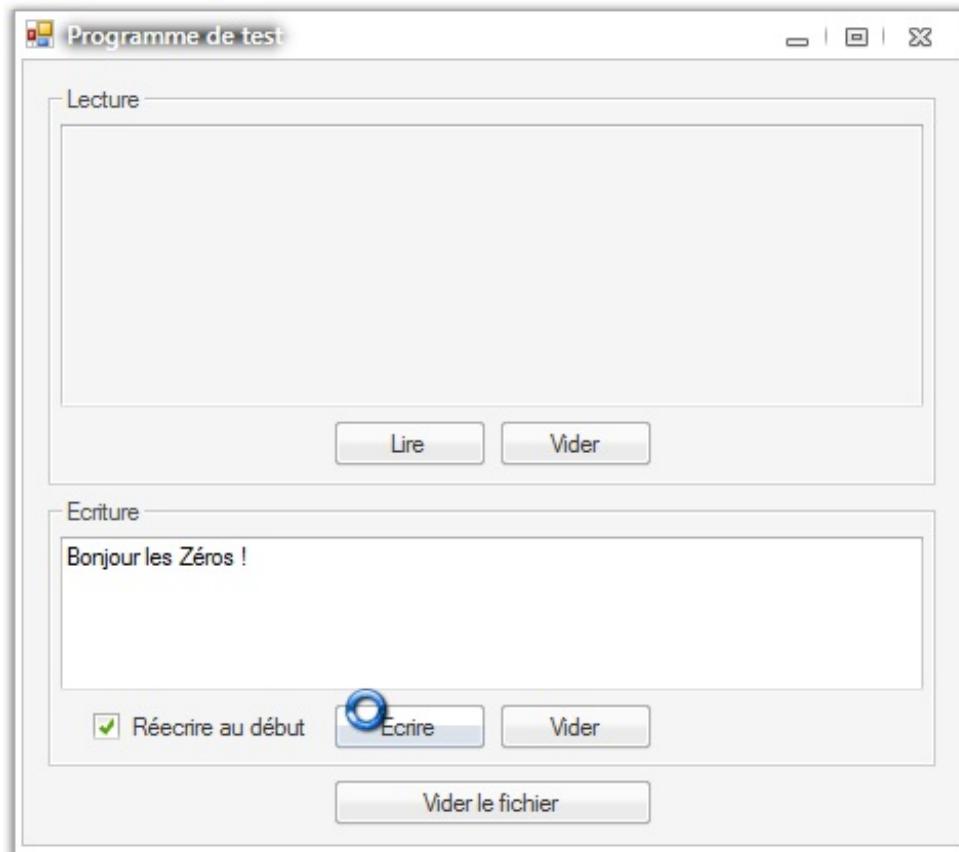
MonFichier.Read(Contenu, 0, 1024)
'L'affiche
Me.TXT_LECTURE.Text = ""
For Each Lettre As Byte In Contenu
    Me.TXT_LECTURE.Text += Chr(Lettre)
Next
End If

End Sub

```

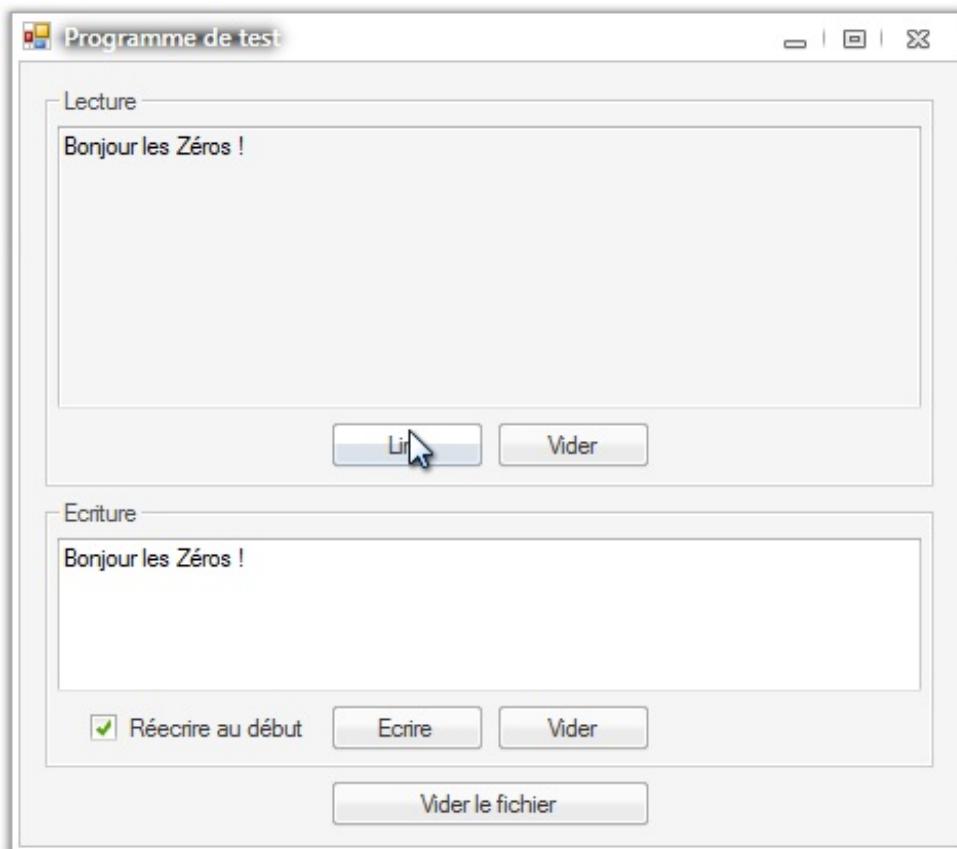
- Première ligne, le même principe que pour l'écriture, on effectue un petite vérification pour savoir si l'on peut effectuer notre lecture.
- On crée un tableau de **Byte** (comme l'écriture : 1025 cases).
- On place le *curseur* à la position 0 (début de mon fichier).
- On lit sur 1024 bytes (si le curseur rencontre la fin du fichier, la lecture s'arrête), et on place ce qui a été lu dans le tableau de **Byte** déclaré juste avant.
- Puis un traditionnel **For Each** afin de parcourir toutes les cases de ce tableau de **Byte**.
- 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 tout dans la textbox(grâce à **+=** on ajoute les caractères à la suite).

Eh bien voilà, cela nous donne la figure suivante en résultat de tests.



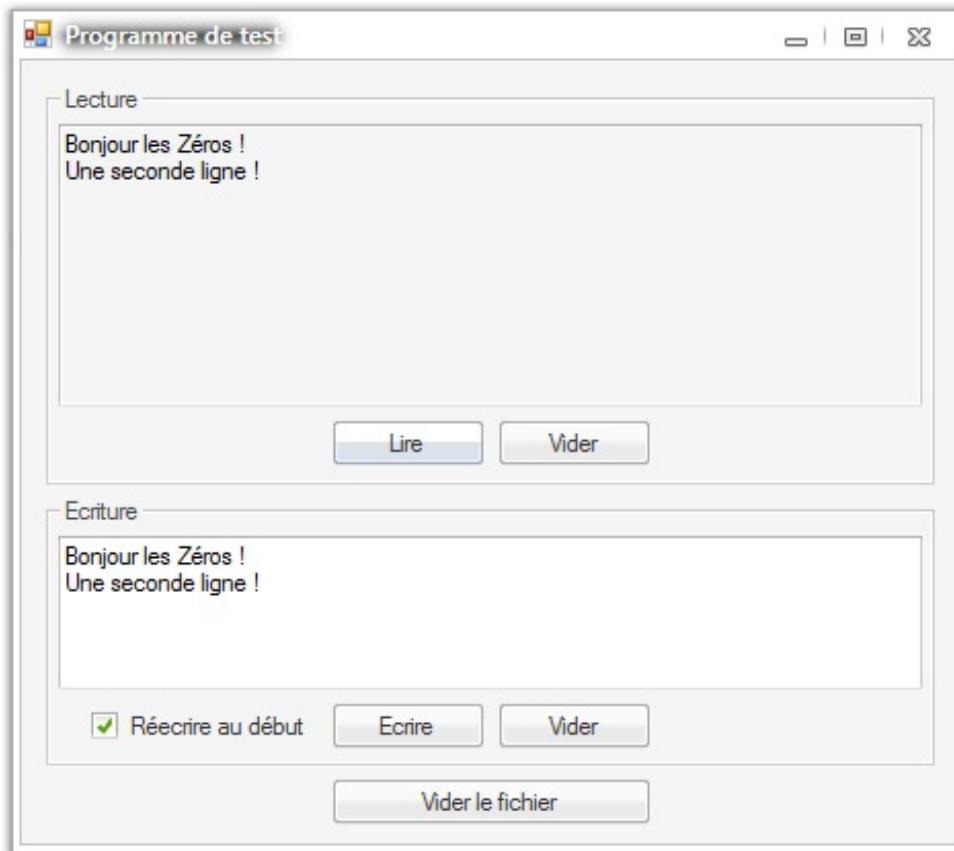
Une demande d'écriture dans notre

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

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 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.

- On utilise le namespace `IO` pour effectuer des opérations d'entrée-sortie sur les fichiers.
- Il faut bien penser au curseur qui avance en même temps que la lecture ou l'écriture dans un fichier.
- On peut utiliser des formats absous ou relatifs pour les chemins des fichiers. Les chemins absous ne sont pas fiables pour un programme destiné à être mis sur d'autres postes.

Les fichiers - partie 2/2

Prêts à acquérir encore plus de notions sur les fichiers ?
 Nous allons voir plus de fonctions sur les fichiers, puis nous aborderons les répertoires !
 Vous aurez bientôt les clés en main pour interagir avec tout votre système de fichiers.
 C'est parti !

Plus loin avec nos fichiers

La technique que je vous ai montrée utilise le principe du stream. Autrement dit, du 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 préimplémentée dans le *framework*. On va créer le même programme de lecture/écriture que précédemment avec cette classe. La même interface donc, mais 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.CHR_DEBUT.Checked Then
      'Depuis le début
      File.WriteAllText(PATHFICHIER, Me.TXT_ECRITURE.Text)
    Else
      'À 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 dont nous avions besoin.



Mais tu es stupide ! Pourquoi nous as-tu ennuyés avec tes 500 lignes au chapitre précédent ?

Eh bien, je vous aurais montré qu'on aurait pu le faire ainsi, auriez-vous 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 la déclaration d'une variable constante 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 utilisée lorsque la checkbox est cochée.

Il nous reste finalement la lecture :`Me.TXT_LECTURE.Text = File.ReadAllText(PATHFICHIER)`. Une fonction qui cette fois-ci lit depuis le début et entre le tout dans un `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 le déplacement du fichier.

Code : VB.NET

```
File.Move(Source as string, Destination as string)
```

Vous pouvez bien évidemment utiliser des chemins relatifs, absous, 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 un Boolean : **True** si présence du fichier, **False** dans le cas contraire.

Les répertoires

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 chemin 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 trois dossiers de créés, suivant l'arborescence suivante : le dossier `Dossier1` sera créé directement dans le répertoire, le dossier `SousDossier1` sera créé 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 chemin 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 chemin que vous avez indiqué 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.

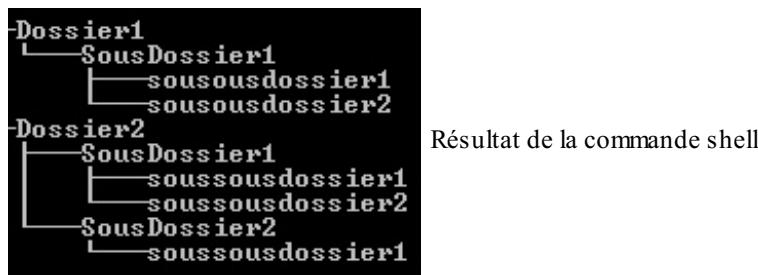
Un rapide bout de code permet de lister les fichiers présents en utilisant cette fonction :

Code : VB.NET

```
For Each Fichier As String In Directory.GetFiles("c:/")
    MsgBox(Fichier)
Next
```

Mini-TP : lister notre arborescence

Tout d'abord, explorons notre arborescence avec une commande toute faite dans notre invite de commande Windows. La commande shell (commande spécifique à Windows) s'appelle **Tree**. Elle donne un résultat similaire à la figure suivante.



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 à se retrouver avec le même schéma, le tout grâce à un algorithme. Je vous ai déjà parlé 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 nœud principal (on peut cliquer dessus pour le « déplier »), et un fichier sera un nœud simple, pas de possibilité de le « déplier ».

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'algorithme, j'aimerais 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 et appeler comme fonctions.

Un algorithme version texte tout simple

Parcourir le répertoire, pour chaque dossier ajouter un nœud principal, pour chaque fichier ajouter un nœud 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.

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 nœud
        Me.TV_ARBORESCENCE.TopNode =
        Me.TV_ARBORESCENCE.Nodes.Add(RepertoireALister, RepertoireALister)

        'Arborescence du premier nœud
        For Each Repertoire As String In
        Directory.GetDirectories(RepertoireALister)
            Me.TV_ARBORESCENCE.TopNode.Nodes.Add(Repertoire,
            Path.GetFileName(Repertoire))
            'Récuratif
            ListeArborescenceDossier(Repertoire,
```

```

Me.TV_ARBORESCENCE.TopNode)
Next
    'Fichiers du premier nœud
    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 nœud
    For Each Repertoire As String In
        Directory.GetDirectories(RepertoireActuel)
        Node.Nodes.Add(Repertoire, Path.GetFileName(Repertoire))
        'Récursif
        ListeArborescenceDossier(Repertoire, Node)
    Next
    'Fichiers de ce nœud
    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* ».

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 nœud 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, j'ajoute chacun 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 fonctions 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.

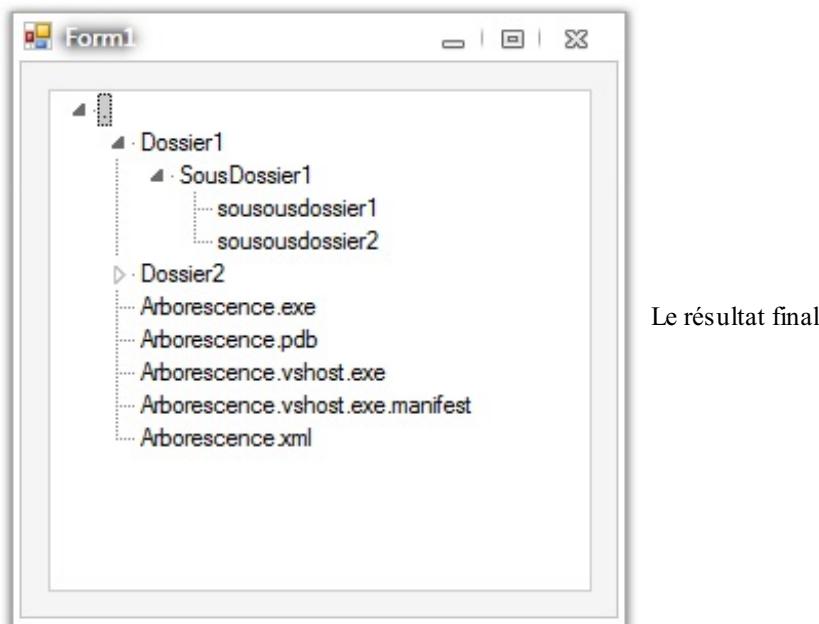
Quand il n'y a plus de dossiers on passe aux fichiers, sur le même principe sauf que ces nœuds n'auront pas de nœuds enfants, donc ne seront pas dépliables.

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.

Et voici le résultat à la figure suivante.



Je tiens juste à vous conseiller d'essayer 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 grand pas vers des notions de programmation plus complexes que nous allons aborder dans la partie 3 de ce tutoriel.

Autre conseil pour vous éclaircir 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 certains d'entre vous ont déjà vu les fichiers de configuration standard de Windows : les fichiers .ini. Ils ont été utilisés par Windows pour définir les paramètres de configuration. Ce sont 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'images, mais ils peuvent très bien contenir du texte et être ouverts avec le bloc-notes.

Les fichiers .ini contiennent donc du texte, mais formaté d'une certaine manière ; nous allons étudier ce formatage 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 trois 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 : « [] » et «] » 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. À 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 ce 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.

- File et Directory sont des objets destinés à contenir respectivement des fichiers et des répertoires.
- Vous pouvez récupérer les fichiers d'un répertoire par la commande Directory.GetFiles().

TP : ZBackup

Eh bien, chers amis Zéros, on va attaquer un TP de taille ! Il nous fera développer un programme 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

C'est parti pour le cahier des charges ! En premier lieu je vais vous décrire ce que notre programme sera susceptible de faire.

Tout d'abord ce programme est un programme d'auto-backup, autrement dit de 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 laisse libre cours 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 y insérer, etc.

Pour le choix des dossiers, je ne vous en avais pas parlé avant, mais un petit module très pratique existe : le **FolderBrowserDialog**.

Dans la boîte à outils, section `boîtes de dialogue`. Ce module ouvre une boîte 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.

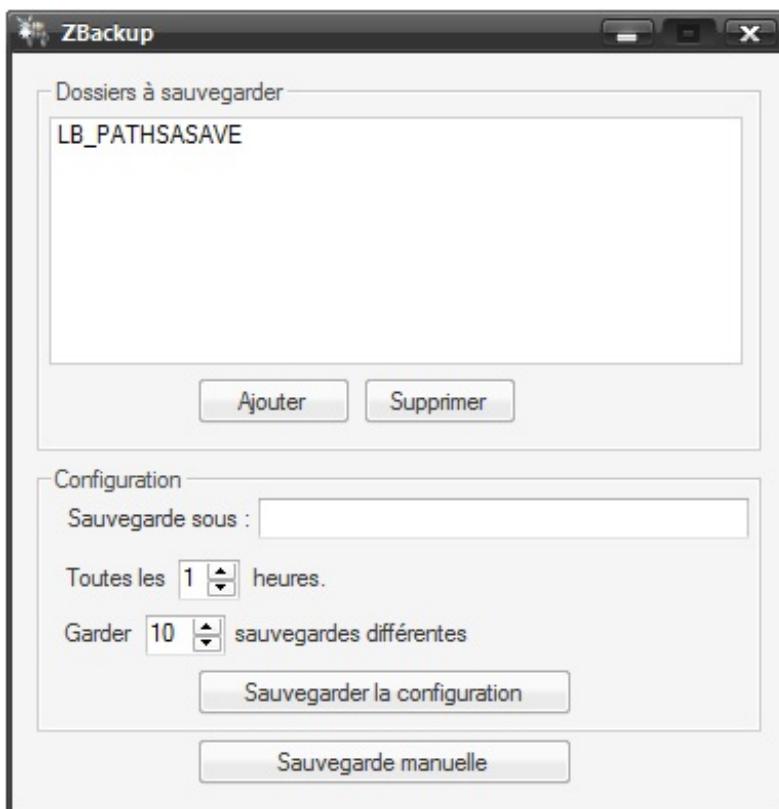
Pour le reste, je vous laisser agrémenter au choix et selon vos goûts.

Je dois dire que vous avez toutes les compétences et les méthodes de réflexion (savoir trouver les propriétés qui vous seront utiles) requises. Essayez de ne pas vous décourager trop rapidement et de ne pas aller trop vite à la correction.

Eh bien, amis Zéros, au travail !

Correction

Commençons par mon interface (voir figure suivante) et par les fonctionnalités que mon programme contient.



Mon programme ressemble à ça

Comme vous le voyez, j'ai une **listbox** qui me permet de lister les répertoires dont je veux la sauvegarde. Vient ensuite un bouton d'ajout et de suppression des répertoires, l'ajout se fait par **FolderBrowserDialog**, la suppression par lignes sélectionnées dans la listbox. Ensuite, 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**. Après se trouvent 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.

Je vais vous montrer le code tout de suite, on développera ensuite section par section.

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 sélectionné
        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 sélectionné
        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()
        'Vérification 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

```

```

        End If
    End Sub

    Function CreeStructureFichierIni(ByVal SavePath As String, ByVal
TempssSave 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
        'Remplit la structure du fichier ini
        FichierIni(0) = ";Fichier de configuration du Zbackup"
        FichierIni(1) = "[configuration]"
        FichierIni(2) = CleSavePath & "=" & SavePath
        FichierIni(3) = CleTempSave & "=" & TempssSave
        FichierIni(4) = CleNbSaves & "=" & Nbsaves
        FichierIni(5) = ""
        FichierIni(6) = "[paths]"
        'Remplit 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 première 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

        'Vérification de la présence du .ini
        If File.Exists(FichierIni) Then
            'Récupération
            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 incrémentera avant
            Do
                i += 1
                Paths(i - 1) = RecupereCleFichierIni(ClePaths & i)
            Loop While Paths(i - 1) <> ""
            'Donc nombre de paths valides : i-1

            'Vérification
            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 chaîne 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("/", "-") &
        " à " & Date.Now.ToShortTimeString.Replace(":", "-")
        If Not Directory.Exists(DossierSave) Then 's'il existe
        deux sauvegardes 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 répertoire est un répertoire de sauvegarde
    If Path.GetFileName(Repertoire).Contains("Sauvegarde")
Then
    'Incrémantation 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 répertoire est un répertoire de sauvegarde
            If
                Path.GetFileName(Repertoire).Contains("Sauvegarde") Then
                    'Si le répertoire est plus ancien 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-répertoires
        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 volumineux !

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.
- À chaque tick de timer (timer configuré sur le temps souhaité entre deux sauvegardes), on effectue la sauvegarde ;
- Avec le bouton de sauvegarde manuelle, la même action est réalisée ;
- La sauvegarde consiste à créer un dossier sous la forme Sauvegarde du DD-MM-AAAA à HH-MM ;
- Puis copie l'intégralité des dossiers en respectant leur arborescence.

L'interface

Voilà donc, commençons par l'analyse la plus simple : celle de l'interface.

Première information : l'ouverture de la FolderBrowserDialog lors du clic sur la textbox.
Eh bien, rien de sorcier, l'événement Click 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 sélectionné
    dans le dialogue
    If Me.BD_DOSSIER.ShowDialog() Then
        Me.TXT_SAVEPATH.Text = Me.BD_DOSSIER.SelectedPath
    End If
End Sub
```

Lors du clic, on se sert du contrôle BD_DOSSIER qui est mon FolderBrowserDialog. Vu que c'est une boîte de dialogue, c'est la fonction ShowDialog qui est appelée. Puis on récupère la sélection avec la propriété SelectedPath.

Le bouton d'ajout a le même principe, mais il agit sur la listbox, sur le même principe que le treeview que nous avons étudié 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 à 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 sélectionnée.

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

    'Remplit 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]"
    'Remplit 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 créé avec comme taille le nombre de lignes initiales plus le nombre de chemins à insérer. Pour les premières lignes, j'écris manuellement dedans les premières clés. Ce qui nous donne dans le fichier `ini` une fois créé :

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 deux sauvegardes et le nombre de sauvegardes maximal.

La section paths contient les différents chemins, tous avec un numéro différent. Les techniques peuvent différer, il aurait été possible de mettre tous les chemins sur la même ligne, séparés par des « ; ».

Bref, la création du fichier n'est pas sorcier, le tableau de variables FichierIni() est renvoyé et est écrit dans le fichier.

Maintenant que vous avez vu 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 première 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 parcourt toutes les lignes du fichier ini, chaque ligne est découpée grâce à la fonction Split().

La fonction Split() s'applique sur une chaîne de caractères, elle permet de « découper » cette chaîne à chaque occurrence du caractère ou de la chaîne passée en argument. Voici un exemple : pour une chaîne de caractères sous la forme Cle1=Valeur1, un Split("=".0) donnera un tableau de String sous la forme :

Tableau(0) = Cle1
Tableau(1) = Valeur1

Donc un test bête et méchant sur le tableau(0) qui est retourné avec la clé souhaitée nous indique la ligne contenant cette clé. 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 maintenant le principe de la sauvegarde.

La méthode Sauvegarde () effectue diverses vérifications 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-réertoires
    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 ne sont 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'instancie deux objets avec comme paramètres les chemins des dossiers voulus.

Une fois dans la méthode de copie, un dossier est tout d'abord créé, les fichiers contenus y sont copiés également, puis cette action est répétée pour tous ses sous-réertoires. De la même manière que le treeview du chapitre précédent.

Il y a finalement la méthode de nettoyage des sauvegardes (si on ne demande qu'un certain nombre de sauvegardes). Elle parcourt les répertoires créés, 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 est décortiqué.

Allons un peu plus loin.

Récapitulatif du 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
```

```

'Remplit 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 à 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 2 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 première 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 fichiers `ini` de tous nos prochains TP ! 😊

Pour aller plus loin

Bon, je ne vais pas continuer l'évolution, car le code me suffit amplement ainsi.

Cette petite idée de TP m'est venue lors d'un après-midi de programmation. J'ai l'habitude de sauvegarder régulièrement mon travail, mais après une fausse manip tout mon projet s'est retrouvé irrécupérable, impossible de faire machine arrière. Ce petit programme effectuant des sauvegardes périodiques du travail aurait pu m'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 entendu 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`.

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 et gratuit.

Voici sa fiche Clubic : [7zip](#).

Cet utilitaire dispose d'une interface graphique, mais aussi d'une utilisation en lignes de commande.

Les commandes (arguments) possibles avec l'exécutable `7z.exe` sont visibles à la figure suivante.

```

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 1 <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
  -ul[-][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

```

Les

commandes possibles avec l'exécutable 7z.exe

Cette manipulation est réservée aux plus expérimentés d'entre vous, il va falloir combiner commandes et chemins de fichiers dans une fonction VB nommée **Shell()** permettant l'exécution de programmes shell.

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 possibles un écran supplémentaire listant les sauvegardes effectuées et la possibilité de restaurer l'une d'entre elle.

Également : 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 principale à **Visible = false**.

Puis créer une icône contextuelle, un contrôle tout fait existe, cherchez dans votre boîte à outils.

Puis récupérer l'événement correspondant au clic ou au double-clic sur cette icône 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'améliorations possibles ! En y passant un peu de temps votre programme peut devenir une véritable sauvegarde périodique de vos données vitales, tout 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, mais 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'objet, et nous allons apprendre à concevoir nos propres objets, ça vous dirait de construire votre propre voiture ?

Bref, je rigole, mais attaquons tout de suite.

Pourquoi changer ?

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, alors pourquoi ne pas continuer ?

La méthodologie de programmation que nous avons vue jusqu'à présent est très bonne, nous avons même vu quels genres de programmes nous étions capables de réaliser en suivant ce concept. Mais d'un autre côté... il y a plein de choses qui sont impossibles 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 prendrait 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 remarqué que pendant tous nos TPs chaque « action » à effectuer était souvent décomposée en un sous-ensemble de fonctions et procédures (**Sub**). C'est cela la programmation procédurale.

Comme vous le voyez sur le schéma présent à la figure suivante, ces fonction s'imbriquent les unes aux autres. Pour le moment, aucun problème. Mais lorsque nous attaquerons de gros projets, cette structure va devenir un véritable capharnaüm.

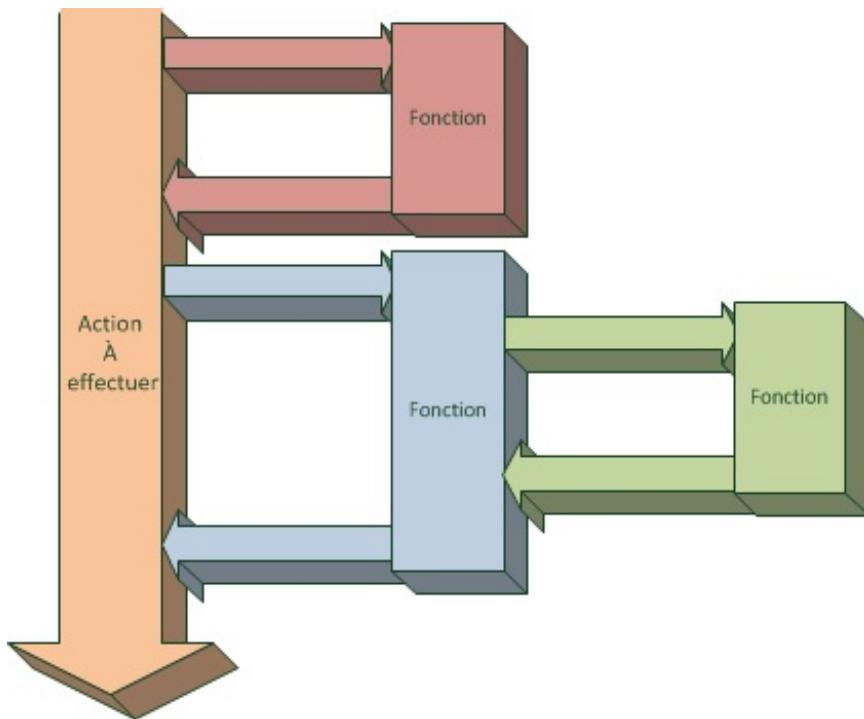


Schéma de la programmation procédurale

C'est pourquoi 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 ?

Nos nouveaux amis : les objets

Dans la vie de tous les jours, vous voyez ce qu'est 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 programmons des fonctions qui seront liées à l'objet Voiture ; après peu importe que l'on décide d'en faire 1 ou 100, il n'y aura pas à recommencer. Imaginez un gâteau. Nous allons coder le « moule » du gâteau. Une fois créé, il sera capable de nous fabriquer des dizaines de gâteaux.

Vous avez utilisé pas mal d'objets jusqu'à maintenant, comme la classe File par exemple. 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 **constructeur** (ce qui se produit lorsque l'on crée notre objet, en l'occurrence notre voiture) et il y a la possibilité de mettre un **destructeur** (je pense que vous avez compris son utilité).

Ces méthodes seront présentes dans le fichier de la classe. 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, dans la classe, et ce qui va se passer à l'extérieur.

Dans une classe, il y a une partie considérée comme privée, qui ne sera accessible que par elle-même. Cette partie contient des fonctions ou des variables, pour faire avancer la voiture. Il va falloir un énorme travail en interne pour qu'au final on ait juste à « appuyer sur une touche » pour que la voiture bouge.

Vient ensuite la partie publique, 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 la forme MaVoiture.Avancer (où MaVoiture est la voiture créée 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 boîte de vitesses, etc., bref tout ce qui la fait fonctionner en « interne », 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'autres mots « préfixes » que nous pourrons employer avant nos déclarations. Il y a Shared, Protected... Mais pour le moment intéressons-nous uniquement à Private et Public.

Les fichiers de classe

Je comprends tout à fait que cette notion d'objet soit très dure à appréhender, je ne vais donc pas 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 seul.

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-la en publique.

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 pourra utiliser. Des membres et des attributs en langage technique. Pour les créer ce sera comme ce que nous avons vu 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

Le constructeur est la méthode qui va être appelée à l'instanciation de l'objet ; au moment où nous ferons = New MaClasse, ce seront des arguments que l'on pourra spécifier de cette manière :

Code : VB.NET

```
= New MaClasse(ArgumentConstructeur1, ArgumentConstructeur2)
```

Comme une fonction qui demande des arguments, le constructeur réagira de la même manière. Il récupèrera les informations passées en arguments et en fera ce qu'il veut, les attribuer à des membres privés par exemple. Je vous expliquerai bientôt comment le coder et l'utiliser.

Le destructeur

Le destructeur est particulier, il est surtout utilisé pour libérer les ressources mémoire 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 connexion si elle a été établie pour cette classe.

Bref, voici ce qui explique son utilité : les variables qui sont créées pour la classe sont automatiquement libérées de la mémoire quand elles « meurent ».

L'objet sera détruit si la valeur Nothing lui est affectée ou s'il arrive à la fin de sa portée (fin d'une fonction dans laquelle il a été

créé) par exemple, comme pour les variables normales.

- La programmation orientée objet est basée sur le concept de classes.
- Les accessibilités permettent au programmeur de ne pas accéder aux membres d'une classe qu'il ne doit pas modifier.

Notre première classe

Nous venons de voir la théorie concernant la programmation orientée objet, mettons tout cela en pratique !

Nous allons voir dans ce chapitre comment créer notre première classe en Visual Basic .NET, puis comment lui ajouter des attributs et des méthodes, d'autant plus de paramètres qui détermineront la complexité de votre classe.

Pour finir et vous récompenser de votre assiduité, nous allons mettre en pratique le concept de classes pour vous montrer l'utilité concrète de ces dernières.

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 ? 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 Classe, comme le montre la figure suivante.

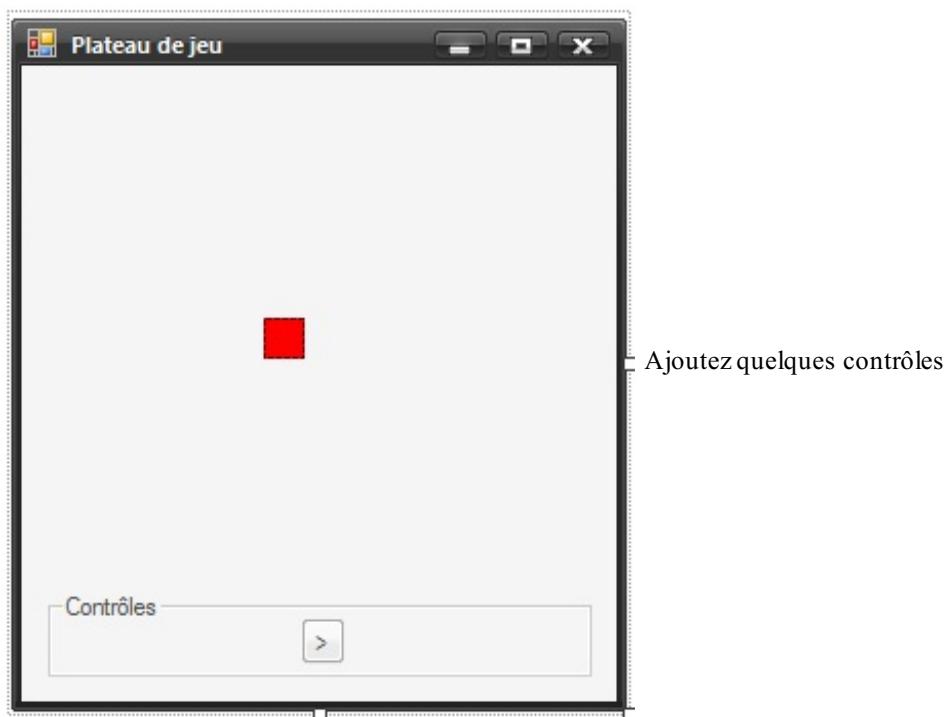


Nous allons créer une classe

Nommez tout de suite ce fichier, « Mario » par exemple. Nous voilà donc avec notre classe totalement vide, juste la déclaration de début et de fin.

L'interface

Retournons dans notre fenêtre et ajoutons-y quelques contrôles pour pouvoir commencer (voir figure suivante).



Donc j'ai juste ajouté un petit **panel** dans lequel j'ai spécifié une taille de 20 x 20, ainsi qu'une 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'événement 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

Pour le moment je vous ne demande pas de créer une classe vous-mêmes, 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 50 x 50, une case sera égale à 50 x 50 (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, trêve de commentaires, allons-y.

Notre classe

Dans notre fichier de 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. Il ne 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 avoir mis un underscore (« _ ») devant tes variables ?

C'est une vieille habitude, lorsque je déclare des membres ou des attributs privés, je les précède d'un underscore, mais vous n'êtes pas obligés de faire comme moi.

Ces variables sont déclarées en `Global`, elles seront donc accessibles 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-mêmes 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 sont les méthodes et les attributs. Les méthodes sont des fonctions de type privé, leur nécessité est interne, rien n'est visible depuis l'extérieur. Les attributs sont publics, visibles depuis l'extérieur, accessibles.

Codons quelques fonctions qui vont nous permettre de déplacer notre Mario. À 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 « case » 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-la 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 d'un underscore 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, et pour le **_PasY**, c'est la hauteur. Si vous avez suivi, cette méthode publique va déplacer Mario d'une case sur la droite. Mais ce n'est pas une fonction, on n'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 sont 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 fourni 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 par vous-mêmes) mais on aurait pu effectuer beaucoup d'autre choses en même temps. Assigner plusieurs variables, incrémenter un compteur, 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.

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 récupère la nouvelle position
    Me.PAN_MARIO.Location = MonMario.Position
End Sub

End Class

```

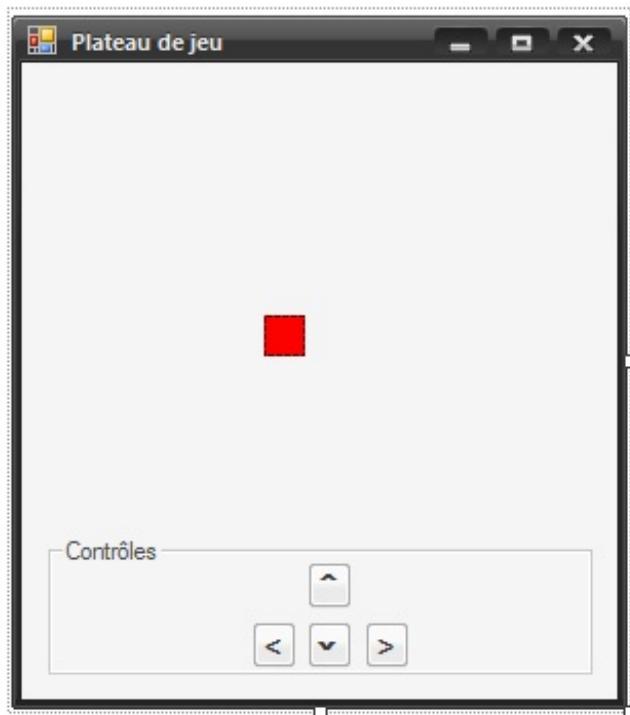
Je déclare un Mario en variable globale, donc il sera accessible et utilisable pendant toute la durée de vie de la fenêtre. Je l'instancie au load de la fenêtre en passant la position du panel et ses dimensions en paramètres au constructeur. Puis lors du clic sur le bouton, on fait avancer Mario et on récupère sa nouvelle position pour l'affecter au panel.

Vous pouvez essayer, ça fonctionne !

Notre petit Mario

Cette section est là pour vous donner le reste du code de notre déplacement de Mario, on s'en servira sûrement plus tard.

La figure suivante vous montre à quoi ressemble ma fenêtre finale (on mettra une image de Mario plus tard 😊)



Des flèches pour déplacer Mario dans toutes les directions

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

```

Et le 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 récupère 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 récupère 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 récupère 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 récupère la nouvelle position
        Me.PAN_MARIO.Location = MonMario.Position
    End Sub

#End Region

End Class

```

Voilà. Amusez-vous bien. 😊

- Une classe contient des variables au même titre que notre programme principal.
- Pour chaque classe instanciée, la variable va être différente.
- Donc chaque classe est unique et se comportera différemment en fonction des ses attributs et méthodes.

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 en attendant que tout ça devienne limpide, je vais vous apporter quelques nouvelles notions afin de vous permettre d'employer au mieux ces nouvelles connaissances.

Dans ce chapitre nous allons aborder quelques notions plus poussées et utiles. Au menu : l'héritage, les classes abstraites, les événements, la surcharge d'opérateurs, les propriétés par défaut, les collections, les bibliothèques de classes et comment les utiliser. Soyons fous !

Bon appétit ! 

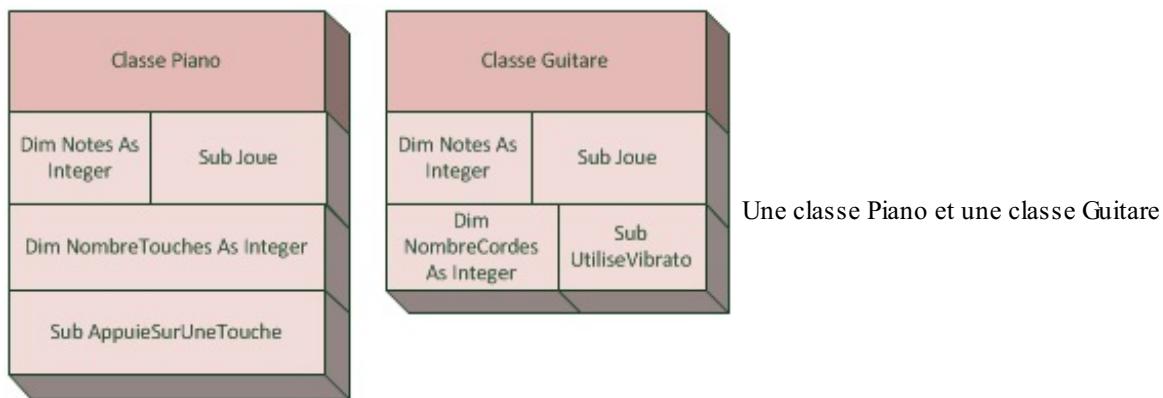
L'héritage

Premier concept important : l'héritage. J'ai déjà tenté de vous exposer cette notion dans la partie sur les fenêtres, mais ce n'était pas très judicieux, vous ne connaîtiez 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 ». Wikipédia nous donne la définition suivante : « 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.

Regardez la figure suivante.

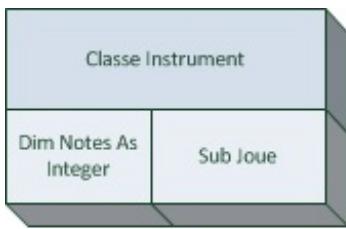


Admettons que je veuille créer deux classes comme celles-là. Une classe **Guitare** et une classe **Piano**. Vous remarquez qu'elles présentent des similitudes, elles possèdent toutes les deux un attribut **Notes** qui va contenir le panel de notes que cet instrument peut jouer et un **Sub Joue** qui va lui permettre de produire un son.

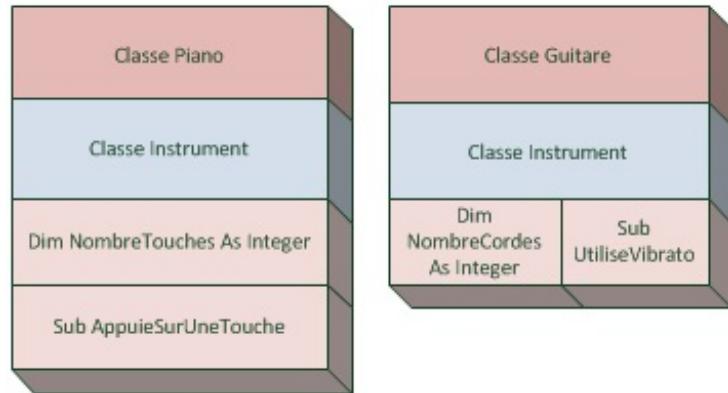
En plus de ces deux éléments communs, elles présentent des particularités spécifiques à leur type : la guitare aura en plus le nombre de cordes qu'elle possède et une fonction permettant d'utiliser le vibrato. Le piano quand à lui, contiendra une variable comptant le nombre de touches qu'il possède et 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 les programmeurs ont introduit le concept de l'héritage.

Regardez, si nous créons une troisième classe nommée **Instrument** qui contient des choses communes à tous les instruments, comme l'attribut **Notes** par exemple, il serait simple d'inclure cette classe dans les autres, de façon à bénéficier de ses caractéristiques, comme le montre le schéma de la figure suivante.



La classe Instrument est incluse dans les autres classes



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 autre classe et ainsi de suite...



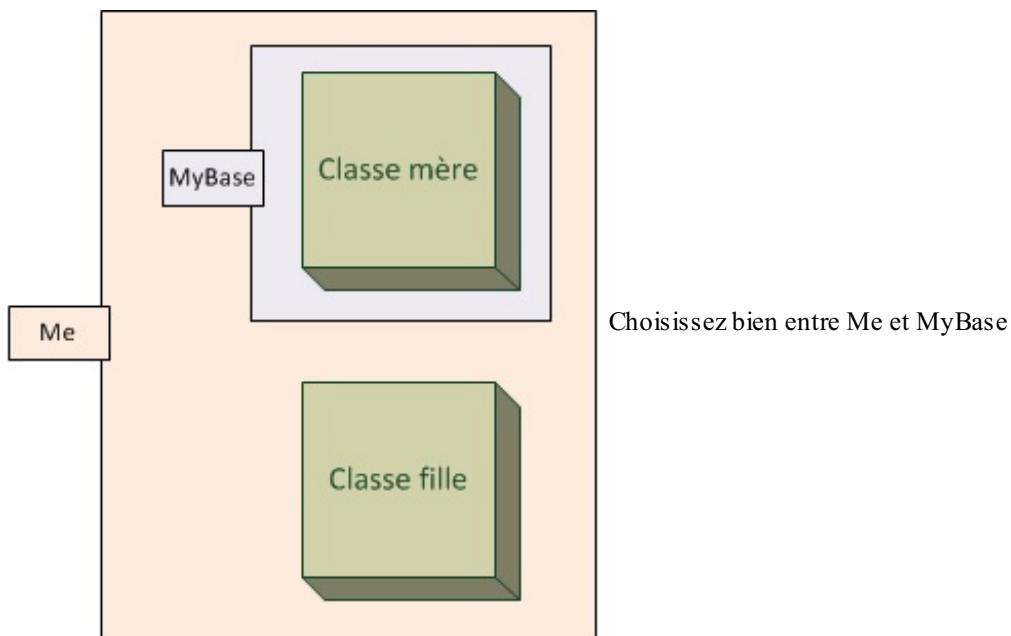
Alors, qu'est-ce que cette modification implique concrètement pour nos classes ?

Eh bien d'un point de vue extérieur à la classe, une fois instanciée par exemple, 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.

En revanche, d'un point de vue interne à la classe, ça se complique. Nous allons devoir apprendre à jongler entre les membres appartenant à la classe fille ou à la classe mère au travers de préfixes (du même type que **Me**).

Ce mot est **MyBase**.

Dans le schéma visible à la figure suivante, 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 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 information 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 possibles 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 et 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 créé 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

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 ?

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 en dériver. Comme notre exemple sur les instruments. Une guitare, un piano, bref des instruments concrets, nous allons les instancier et les utiliser. Cependant la classe `Instrument`, la classe mère de toutes les autres, nous aurions pu la définir en classe abstraite.

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 qu'ici le mot-clé **MustInherit** spécifié dans la déclaration de la classe (qui signifie en français « doit hériter »), mais qu'il est mieux de traduire par « doit être hérité »), spécifie que cette classe sera abstraite, elle ne pourra pas être utilisée telle quelle.

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`. La syntaxe `MaGuitare.Notes()` est tout à fait correcte (où `MaGuitare` est mon objet créé).

Il existe toutefois une seconde utilisation possible pour les classes abstraites ; en tant que bibliothèques de fonctions.

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 d'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 événements

Vous vous souvenez des événements dans nos contrôles ? Eh bien ici c'est le même principe.

Un événement est une fonction appelée lorsque quelque chose se produit. Ce « quelque chose » peut être l'appui d'un clic sur un bouton, l'insertion de texte dans une textbox, le chargement d'une fenêtre, etc. Vous vous souvenez du mot-clé **Handles** ? C'est ce qui indique l'écoute d'un événement sur un objet. On avait l'habitude de l'utiliser pour des contrôles visuels, nous allons désormais apprendre à écouter un événement sur notre classe.

Pour commencer, il faut spécifier ce qui va déclencher l'événement 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 événements de cet objet. Vous avez déjà utilisé un timer en tant que contrôle (les contrôles étant d'office avec **WithEvents**), cette fois il ne sera visible que côté code. 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'événement `Tick` du timer, j'incrémente un compteur ; une fois arrivé à 10 je déclenche l'événement.

Ça donne le code suivant 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'événement 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 classe 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 créé")
End Sub
```

Puis je l'instancie dans le `form`.

Pour finir j'ai écouté l'événement 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 événements. Côté classe on déclare l'événement avec :

Code : VB.NET

```
Public Event DixSecondes(ByVal Message As String)
```

On voit qu'il attend un argument de type `String`.

Donc lorsqu'on va l'appeler :

Code : VB.NET

```
RaiseEvent DixSecondes("Dix secondes que l'objet est créé")
```

... je lui passe mon argument. Et finalement côté fenêtre, le `Handles` de mon événement s'effectue ainsi :

Code : VB.NET

```
Sub AttendsLesDixSecondes(ByVal Message As String) Handles
    MaClasse1.DixSecondes
    MsgBox(Message)
End Sub
```

La surcharge

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 pour passer ces arguments, comme à la figure suivante.

```
Dim Fluxfichier As New IO.FileStream(
```

▲ 2 sur 15 ▼ New(path As String, mode As System.IO.FileMode)
 Initialise une nouvelle instance de la classe System.IO.FileStream avec le chemin et le mode de création spécifiés.
path: Chemin d'accès relatif ou absolu pour le fichier que l'objet FileStream en cours doit encapsuler.

L'assistant de Visual Basic vous propose plusieurs possibilités

Vous voyez que l'infobulle spécifie « 2 sur 15 ». J'ai choisi la seconde possibilité, sur quinze possibles, de donner les arguments. C'est cela la surcharge, pour la même fonction, avoir plusieurs « résultats » possibles en fonction du passage des arguments.

Créons tout de suite un constructeur surchargé pour vous montrer 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
```

J'ai déclaré dans ma classe `Guitare` trois constructeurs différents, j'ai donc surchargé le constructeur.

Ce qui me donne le droit à l'infobulle visible à la figure suivante lorsque je veux l'instancier.

```
Dim MaGuitare As New Guitare(
```

▲ 1 sur 3 ▼ New()

Une infobulle apparaît lorsque j'instancie la classe `Guitare`

Deux fonctions avec des arguments de types différents 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 ajouter 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 ajouté **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 Paramètres dans les propriétés

Avant de comprendre les propriétés par défaut il faut juste que je vous montre comment utiliser un paramètre dans une 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'aurez plus besoin d'écrire MaClasse.Variable(0), mais seulement MaClasse().

À utiliser avec précaution si vous ne voulez pas vite être embrouillés. 😊

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())
    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 étant 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 **Integer** : `10 + 10 = 20` ;
- Entre deux **String** : « Sal » + « ut » = « Salut » ;
- Entre deux **Date** : `CDate("20/10/2010") + CDate("20/10/2010") = 20/10/201020/10/2010`.

Bref, rien à voir.

Apprenons à surcharger un opérateur pour notre classe pour que celle-ci réagisse 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

Tout d'abord, et comme d'habitude, qu'est-ce qu'une collection ? À quoi ça va nous servir ?

Eh bien je vais d'abord vous exposer 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 accolées 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 créée 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.

La figure suivante est un schéma d'exemple.

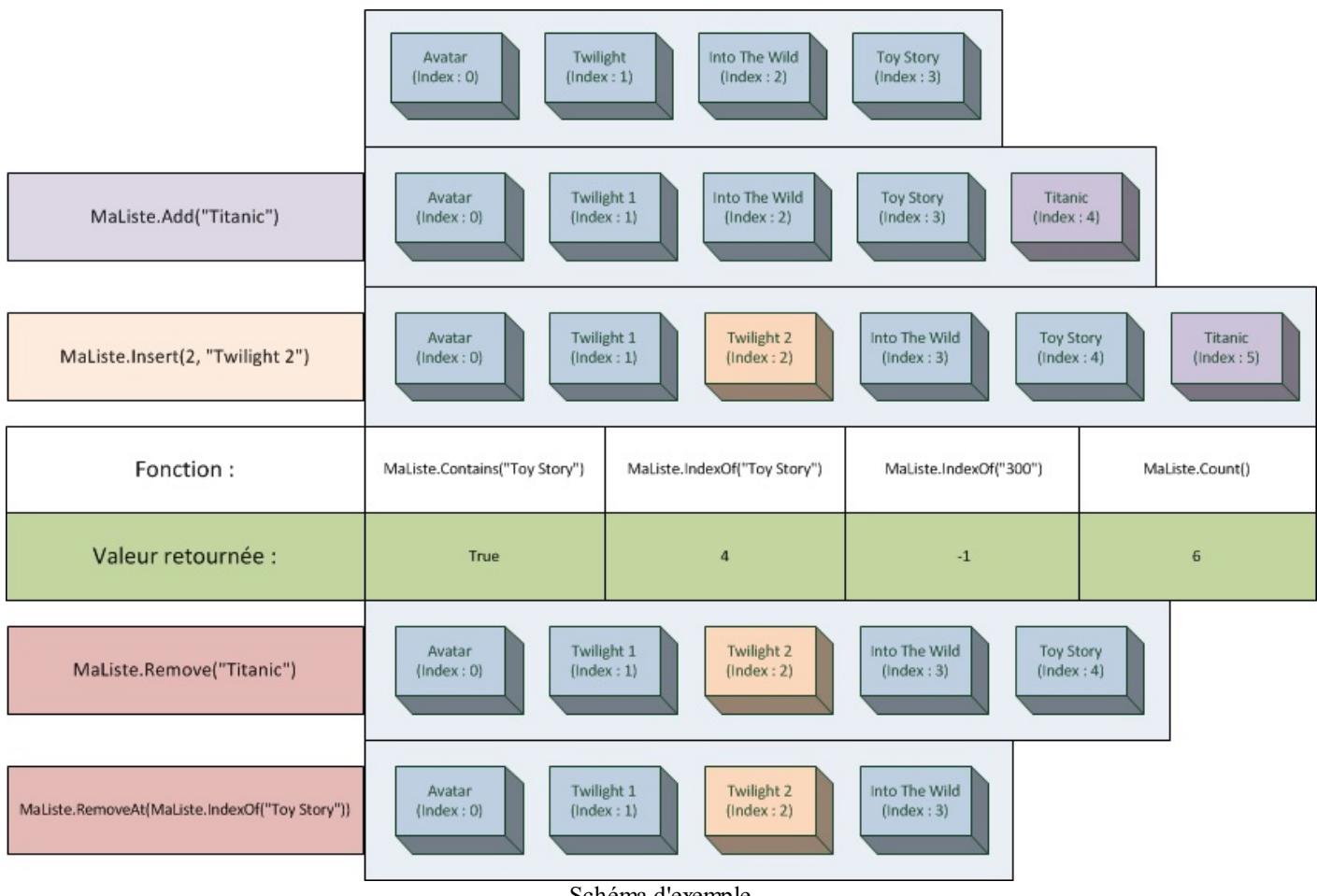


Schéma d'exemple

- 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. S'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**, quant à elle, renvoie le nombre d'éléments dans la liste. À la même manière que **Length** 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 programmation :

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 créeerez de gros objets, 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 développées seront compilées et facilement réutilisables.

Un hic : vous ne pouvez plus modifier les classes une fois le fichier DLL compilé.

Les créer

Donc pour commencer votre bibliothèque, nouveau projet > Bibliothèque de classes (voir figure suivante).



Je vais l'appeler « MesClasses ». À l'intérieur je 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 de la même manière que System.IO par exemple.

Bien sûr si vous avez plusieurs classes à créer, soit vous les insérez dans le même fichier, soit vous créez un fichier par classe.



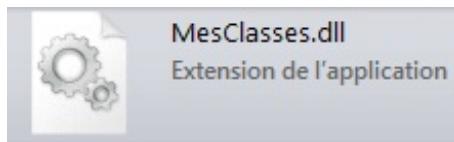
Ce type de projet n'est pas exécutable, il ne contient ni de Main, ni de Load, il doit être obligatoirement utilisé par un autre projet.

Une fois votre classe créée et vérifiée, il va falloir générer le projet. Pour ce faire, un clic droit sur le projet dans l'explorateur de solutions et cliquez sur Générer. La DLL est maintenant compilée.

Pour la retrouver, direction

VosDocuments\Visual Studio 2010\Projects\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 plutôt que dans Debug.

Une fois dans ce répertoire donc, le fichier DLL s'est compilé. Gardez-le bien au chaud, dans un répertoire contenant toutes vos bibliothèques, pourquoi pas.

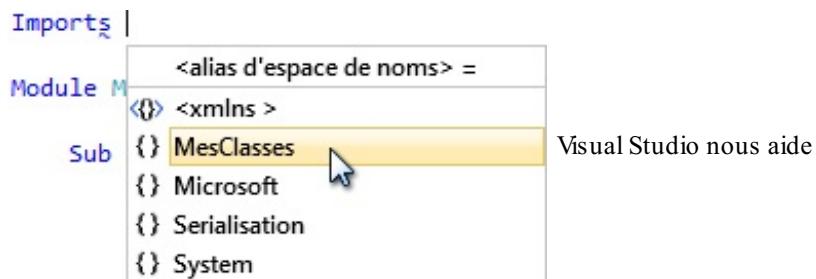


Les réutiliser dans un projet

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 pré-incorporé 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. Et cliquez sur OK.

Maintenant il va falloir importer le namespace. Vous voyez à la figure suivante que Visual Studio nous aide.



Code : VB.NET

```
Imports MesClasses
```

Et voilà votre bibliothèque est totalement utilisable, comme le montre la figure suivante.

Imports MesClasses

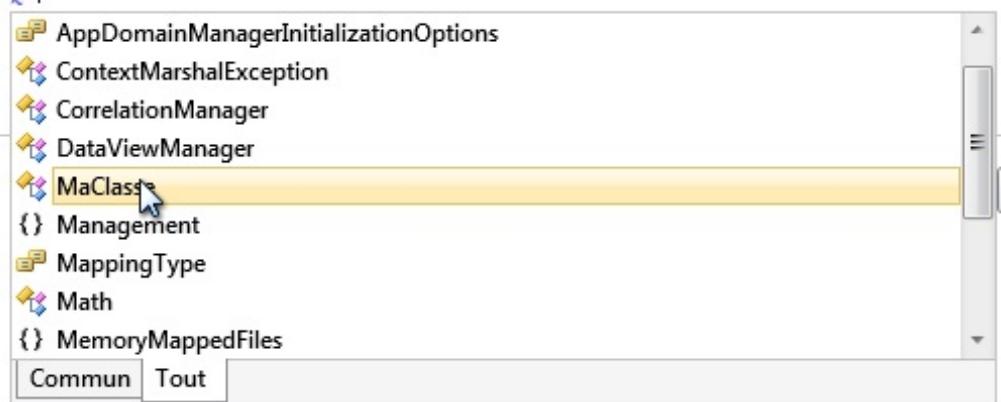
Module Module1

Sub Main()

Dim UneClasse As New Ma

End Sub

End Module



La bibliothèque est totalement utilisable

- L'héritage est utile si on souhaite créer plusieurs classes ayant un lien hiérarchique entre elles (véhicule → quatre roues → voiture → décapotable...).
- La surcharge d'opérateur nous permet de modifier le comportement des opérateurs pour agir avec notre classe. Ainsi Classe1 + Classe2 ne fait pas forcément référence à une addition.
- Une bibliothèque de classes peut être utile si votre programme est complexe ou si vous voulez diffuser votre structure de classes.

La sauvegarde d'objets

Attaquons maintenant la sauvegarde d'objets. Vous avez vu comment créer vos objets, vous avez aussi vu 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 et Messieurs les Zéros, je vous présente la sérialisation.

La sérialisation, c'est quoi ?

Vous vous souvenez du cycle de vie d'une variable normale, dès que la boucle, fonction, classe dans laquelle elle a été créée est terminée, la variable est détruite et libérée de la mémoire. Les données qu'elle contient sont donc perdues.

Vous avez dû 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 contenait (des attributs). Pour sauvegarder notre variable pour une utilisation ultérieure (un autre lancement du programme), nous avons vu comment stocker cette variable dans un fichier.

Les `Integer`, `String`, `Date`, 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 créé 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 Description As String

    Sub New()
        End Sub

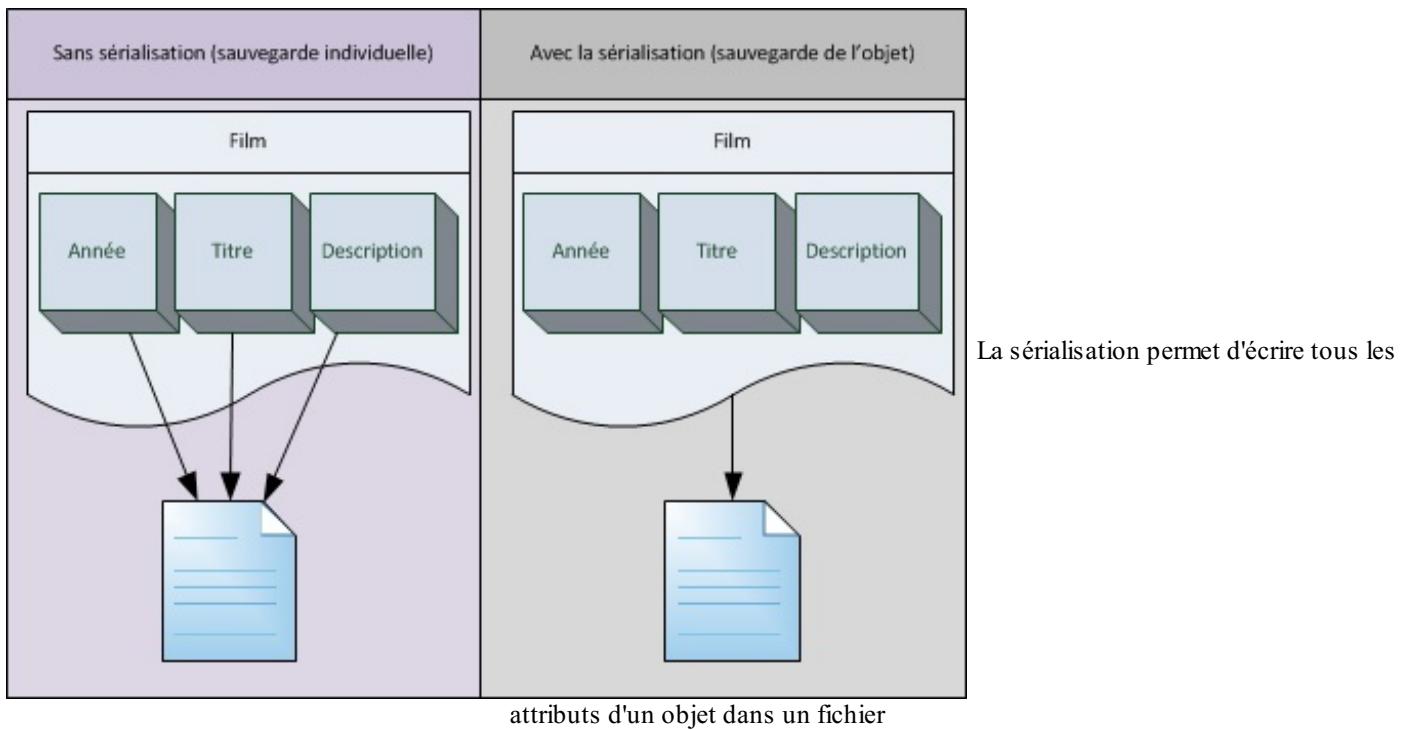
    Sub New(ByVal TitreFilm As String, ByVal AnneeFilm As Integer,
        ByVal DescriptionFilm As String)
        Titre = TitreFilm
        Annee = AnneeFilm
        Description = DescriptionFilm
        End Sub

    End Class
```

Ma classe s'appelle `Film`, elle va contenir des informations pour créer un objet `Film` auquel on spécifiera le nom, l'année de sortie et la description.

Cette classe est très basique, seulement trois attributs. Mais si je veux la sauvegarder, il va déjà falloir écrire trois 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 (voir figure suivante). Ce fichier sera automatiquement formaté. Ce formatage va dépendre de la méthode de sérialisation utilisée.



Il existe donc deux méthodes.

La méthode binaire 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 Année As Integer
    Public Description As String

    Sub New()
    End Sub

    Sub New(ByVal TitreFilm As String, ByVal AnnéeFilm As Integer,
    ByVal DescriptionFilm As String)
        Titre = TitreFilm
        Année = AnnéeFilm
        Description = 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
'Sérialisation 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, il 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 récupère 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 renvois de valeurs via des fonctions, le type de retour n'est pas forcément spécifié, une variable de type **Object** (spécifique à Visual Basic) sera alors retournée, mais pour un programmeur un code avec des « oublis » de ce type peut très vite devenir incompréhensible et ouvre la porte à beaucoup de possibilités d'erreurs.

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, quant 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, quant à elle, respecte le protocole SOAP (*Simple Object Access Protocol*), le fichier XML que vous allez générer va pouvoir être transporté et utilisé plus facilement sur d'autres plateformes et langages de programmation qui respecteront eux aussi le protocole SOAP.

Le format XML (je vous renvoie à sa page [Wikipédia](#) pour plus d'informations) formate le fichier avec une structure bien spécifique composée de sections et sous-sections.

Je reprends l'exemple de Wikipédia 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.

Vient une ligne de commentaire, elle n'est pas censée nous intéresser, car notre sérialisation ne générera pas de commentaire. En revanche, le reste est intéressant. La sérialisation va convertir la classe en un nœud 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'œil 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 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 Serializeur As New XmlSerializer(GetType(Film))
'Sérialisation et écriture
Serializeur.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>
    <Description>Avatar, film de James Cameron sorti en décembre
2009.</Description>
</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 récupère son flux
Dim FluxDeFichier As Stream = File.OpenRead("Film.xml")
Dim Deserialiseur As New XmlSerializer(GetType(Film))
'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

```

Juste un petit mot sur la taille du fichier résultant avec les deux méthodes, binaire et XML.

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 imaginez que nous voulions enregistrer toute une bibliothèque de films (une idée de TP ? 😊).

Déjà, avant que vous vous enfoncez, je vous dis qu'avec notre méthode ça ne va pas être possible.



Et pourquoi ça ?

Eh bien vous avez vu que le nœud principal de notre document XML est <Film>, et j'ai dit qu'il ne pouvait y 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 nos amis les tableaux. Un tableau de **String**, vous vous en souvenez, et dans un TP je vous ai tendu un piège en faisant un tableau de **RadioButton**.

Ça va être le même principe ici, un tableau de **Film**.

Oui, il fallait juste y penser. Ce qui nous donne, en création d'un tableau de **Film** 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()))
'Sérialisation 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 deux 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>
    <Description>Avatar, film de James Cameron sorti en décembre
2009.</Description>
  </Film>
  <Film>
    <Titre>Twilight 3</Titre>
    <Annee>2010</Annee>
    <Description>Troisième volet de la quadrilogie
Twilight</Description>
  </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 récupère 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 de la longueur 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 la taille requise. J'ai fait cela 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"))

  End Sub
End Module
```

```

    '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)))
    'Sérialisation 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 tableaux (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 sauvegardée 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, ce 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 publics, 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 utiliseront sûrement des documents XML, c'est une base de données comme une autre ...

Bref, n'allons pas trop vite, il nous reste à finir cette méchante partie d'orienté objet avant d'attaquer ces autres concepts. 😊

- La sérialisation permet de sauvegarder des objets.
- On peut ensuite les sauvegarder sous forme de fichiers ou alors de flux (nous verrons cela plus tard).
- Nous pourrons ainsi envoyer des classes entières par le réseau à un programme distant.

TP : ZBiblio, la bibliothèque de films

Pour mettre ce qu'on a vu 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-mêmes.

Donc je vais vous donner un cahier des charges, et quelques informations. Tout cela a 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 solution 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 dans 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 faisant précéder le titre de votre sujet par [VB .NET], ça aide à les distinguer.

Bien, bien, excusez-moi, je m'égare, le cahier des charges donc.

Ce TP a 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 cours concernant le design et les méthodes de programmation. Je vous donne juste quelques conseils :

- 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 de 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. 

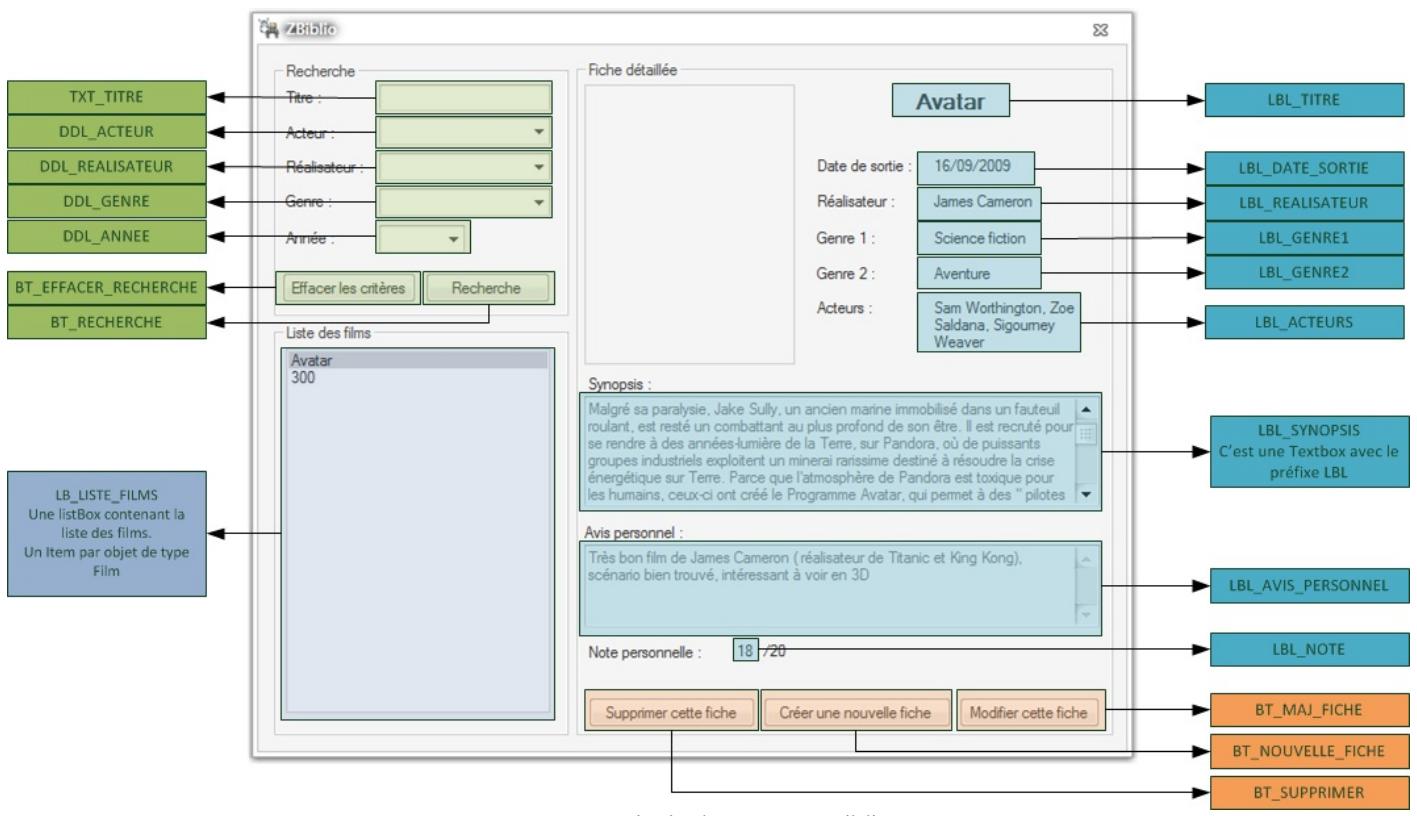
La correction

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 collections à s'en tirer un minimum. Mais si vous n'avez rien compris à ce TP et si vous n'avez pas été capables d'en réaliser une ébauche, je ne saurais que vous conseiller de recommencer la lecture de la partie sur la POO.

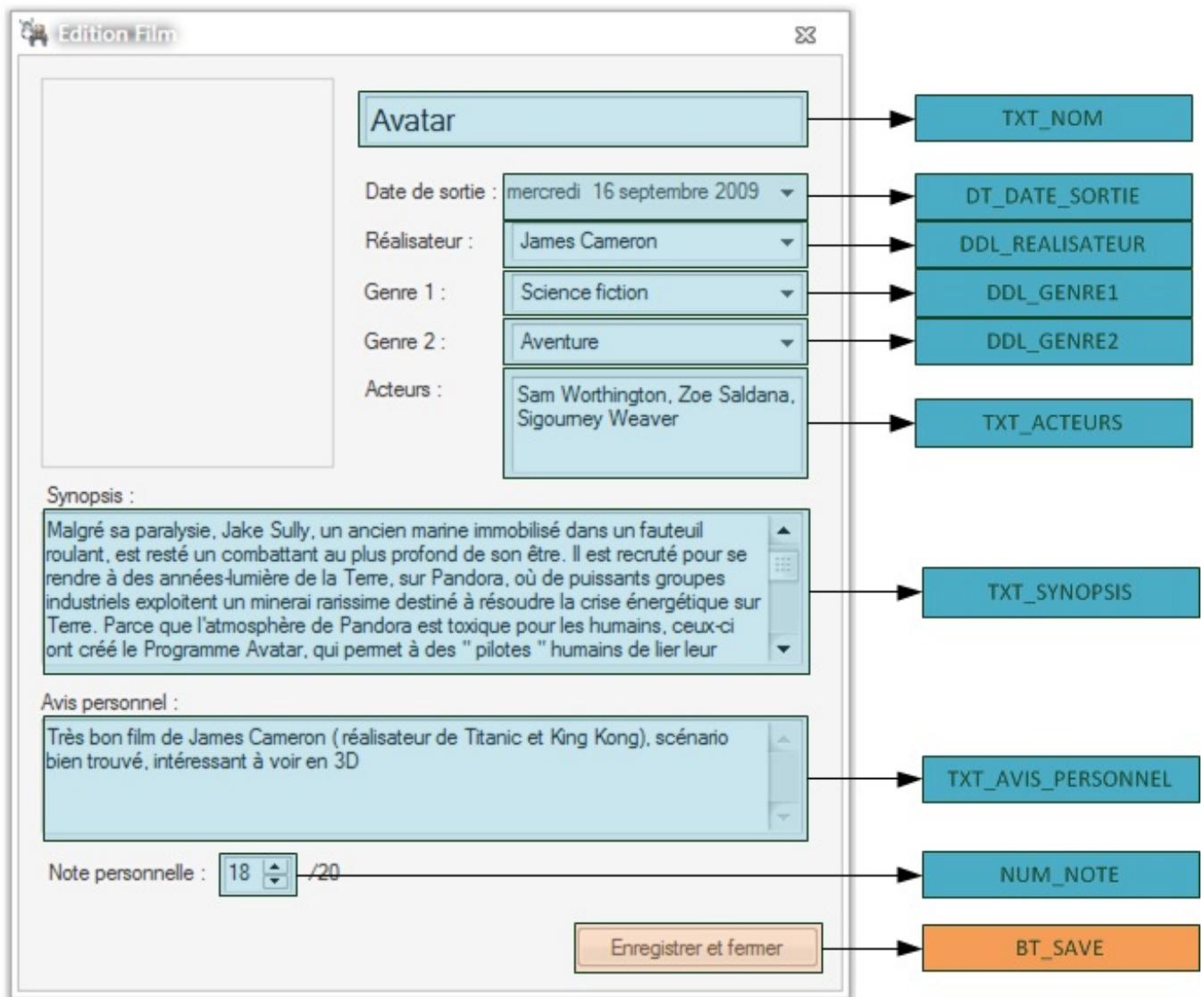
Pour les autres, chaque situation étant différente, je vais tâcher de vous présenter un programme aussi concis et universel que possible.

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 la modification d'une fiche.

Je vais déjà vous détailler les deux fenêtres aux figures suivantes.



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 doute remarqué 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
End Class
```

```
    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
```

```

        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'utiliserais 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 quelque chose
        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
            MeLBL_TITRE.Text = Me._FilmEnVisualisation.Nom
            MeLBL_DATE_SORTIE.Text =
            Me._FilmEnVisualisation.DateSortie.ToShortDateString 'La date seule
            MeLBL_GENRE1.Text = Me._FilmEnVisualisation.Genre1
            MeLBL_GENRE2.Text = Me._FilmEnVisualisation.Genre2
            MeLBL_REALISATEUR.Text =
            Me._FilmEnVisualisation.Realisateur
            MeLBL_ACTEURS.Text = Me._FilmEnVisualisation.Acteurs
            MeLBL_SYNOPSIS.Text = Me._FilmEnVisualisation.Synopsis
            MeLBL_AVIS_PERSONNEL.Text =
            Me._FilmEnVisualisation.RemarquePerso
            MeLBL_NOTE.Text = Me._FilmEnVisualisation.NotePerso
        End If
    End Sub

```

```

    End If

End Sub

Public Sub UpdateListe()
    'On vide la liste et on la reremplit
    Me.LB_LISTE_FILMS.Items.Clear()
    'Parcourt les films de la bibliothèque
    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 récupère son flux
        Dim FluxDeFichier As Stream =
File.OpenRead("BibliothequeFilm.xml")
        Dim Deserialiseur As New XmlSerializer(GetType(List(Of
Film)))
        'Déserialisation et conversion 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)))
    'Sérialisation 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 préremplir les DDL de
    genres, réalisateurs...
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()
    'Parcourt les films de la bibliothèque
    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, quant 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 être appelée 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 InitializeComponent()

    'Récupère 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 récupère 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 créée", vbOKOnly, "Confirmation")
    Else
        'Sinon on le modifie en récupérant son index dans la liste de la fer
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 l'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, en détaillant votre problème.

Améliorations possibles

Concernant les améliorations 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 créez une fonction permettant de les distinguer afin de garder des genres 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 : Les bases de données

Introduction sur les bases de données

Après une longue introduction au VB .NET suivie de plusieurs concepts plus ou moins avancés, vous voilà désormais capables de créer vos programmes avec de multiples possibilités. Mais un programme n'est-il pas au fond un corps sans âme ? Ne faut-il pas lui offrir le moyen de stocker les connaissances pour pouvoir lui offrir encore plus de possibilités ?

Vous savez déjà stocker des informations dans des fichiers. Mais nous allons maintenant apprendre à stocker des centaines, des milliers d'informations dans des bases de données.

Amis Zéros, préparez-vous, vous décollez pour le monde du stockage des informations nécessaires au bon fonctionnement de vos programmes.

Qu'est-ce qu'une base de données ?

Comme je l'ai écrit dans l'introduction, une base de données est un énorme endroit où sont stockées des données (non, sans blague ?). 



Pourquoi stocker des données ?

Prenons un exemple simple. Si vous réalisez un logiciel, un jeu vidéo ou encore un site web plus tard, vous aurez (à un moment bien avancé) un problème : comment sauvegarder toutes les données de l'application ?

Vous connaissez déjà la sérialisation et le stockage dans des fichiers. C'est déjà ça. Mais avez-vous observé la complexité de votre fichier généré lors du TP ZBackup ?

C'est là qu'intervient la fameuse **base de données** (abrégée BDD pour les intimes, et vu que c'est une amie à moi je vais utiliser cette abréviation pour le reste du tuto). Une BDD stocke donc les informations à l'instar d'un fichier. Une BDD est en effet un fichier, mais enregistré dans un format spécial et terriblement complexe.

On va essayer d'y aller mollo et de se servir d'images compréhensibles par tous pour vous faire comprendre la structure d'une BDD.

Imaginez une **armoire**. Comme vous le savez, une armoire est composée de plusieurs **tiroirs**, et dans ces tiroirs vous pouvez placer certaines choses.

À l'instar de notre armoire (appelée **base**), notre BDD est divisée en différentes **tables** (les tiroirs) et c'est à l'intérieur de ces tables que vous stockez les données.

Les créateurs du concept de BDD ont été très malins, ils ont organisé ces tables comme des tableaux (un tableau Excel par exemple). On va pouvoir ajouter autant de colonnes que besoin en fonction des informations à enregistrer.

Voici un tableau représentant une table.

Table Musiques

Numero	Artiste	Titre	Album
1	Don't Cry (Original)	Guns N' Roses	Greatest Hits
2	Metallica	Nothing Else Matters	Metallica
3	Saez	Jeune et Con	Jours étranges
4	Noir Désir	Un jour en France	666.667 Club

Cette table représente un des tiroirs de l'armoire. Les **champs** (colonnes) de cette table sont Numero, Artiste, Titre et Album. La table contient 4 **entrées** (lignes).

L'importance de votre BDD va différer en fonction de l'utilisation de votre programme. Un logiciel de chat en ligne par exemple pourra se contenter de champs stockant les pseudos, les droits accessibles par les membres et pourquoi pas la date de la dernière connexion. Un forum, quant à lui, contiendra sûrement plusieurs tables (une pour les informations des membres, une pour le stockage des messages du forum, etc.) et chaque table aura elle-même des champs spécifiques à la disposition du webmaster.

Les entrées, elles, seront les données pures, une table de stockage de membres avec 10 000 entrées signifie que le site auquel appartient la BDD a 10 000 membres (imaginez la taille de celle du SdZ avec un peu moins de 300 000 membres).



J'ai compris, mais quel est le rapport avec le langage de programmation ?

Le langage de programmation va **demander** des informations à la base de données, par exemple :

- « Donne-moi les informations de la chanson numéro 1 dans la table Musiques. »
- « Donne-moi les chansons ayant pour artiste Metallica dans la table Musiques. »
- Etc.

Pour demander des informations à la base de données, il faut utiliser le **langage SQL**... nous reviendrons dessus.



En français, SQL signifie « langage de requêtes structurées ».

Grâce à une BDD, tout le système de fichiers sera beaucoup mieux géré et plus facilement que si vous gériez vous-mêmes le vôtre (comme lorsqu'on a fait les apprentis sorciers avec notre sérialisation lors du TP bibliothèque de films).

Le **système de gestion de base de données** (SGBD), est un logiciel qui joue le rôle d'interface entre l'utilisateur et la base de données. Un SGBD permet de décrire, manipuler et interroger les données d'une BDD manuellement. Il est utilisé dans un premier temps pour structurer la BDD : création de tables et de champs.

Les bases de données

Les SGBD

Les **SGBD** (systèmes de gestion de base de données) sont des programmes qui s'occupent du stockage des données.

Il en existe plusieurs, dont voici les plus connus :

- **Microsoft SQL Server** : le SGBD de Microsoft, nous l'utiliserons dans ce cours.
- **MySQL** : c'est un des SGBD les plus populaires.
- **Oracle** : c'est le SGBD le plus utilisé en entreprise, le plus complet aussi... mais payant !
- **PostgreSQL** : ressemble un peu à MySQL, avec plus de fonctionnalités, mais un peu moins connu.



De notre côté, nous allons prendre SQL Server, mais vous pourrez changer de SGBD plus tard si vous le souhaitez. Sachez que la plus grande partie de ce que vous allez apprendre fonctionnera de la même manière avec un autre SGBD.

SGBD et SQL

J'ai écrit plus haut que le langage SQL permet de dialoguer avec la base de données. Le langage SQL est un standard, c'est-à-dire que, quel que soit le SGBD que vous utilisez, vous utiliserez le langage SQL. Le seul problème, c'est qu'il existe plusieurs langages SQL (en fonction du SGBD), dans ce cours nous utiliserons du langage SQL « neutre », donc indépendant de toute base de données.

Le langage SQL est un *langage*, donc il s'apprend (mais comparé au VB .NET c'est extrêmement simpliste).

Voici un avant-goût du langage SQL :

Code : SQL

```
SELECT id, nom, prenom FROM Client ORDER BY CodeClient
```

Nous allons dans un premier temps apprendre à manier ce fameux langage avant de passer à la suite, c'est-à-dire envoyer des ordres au SGBD.

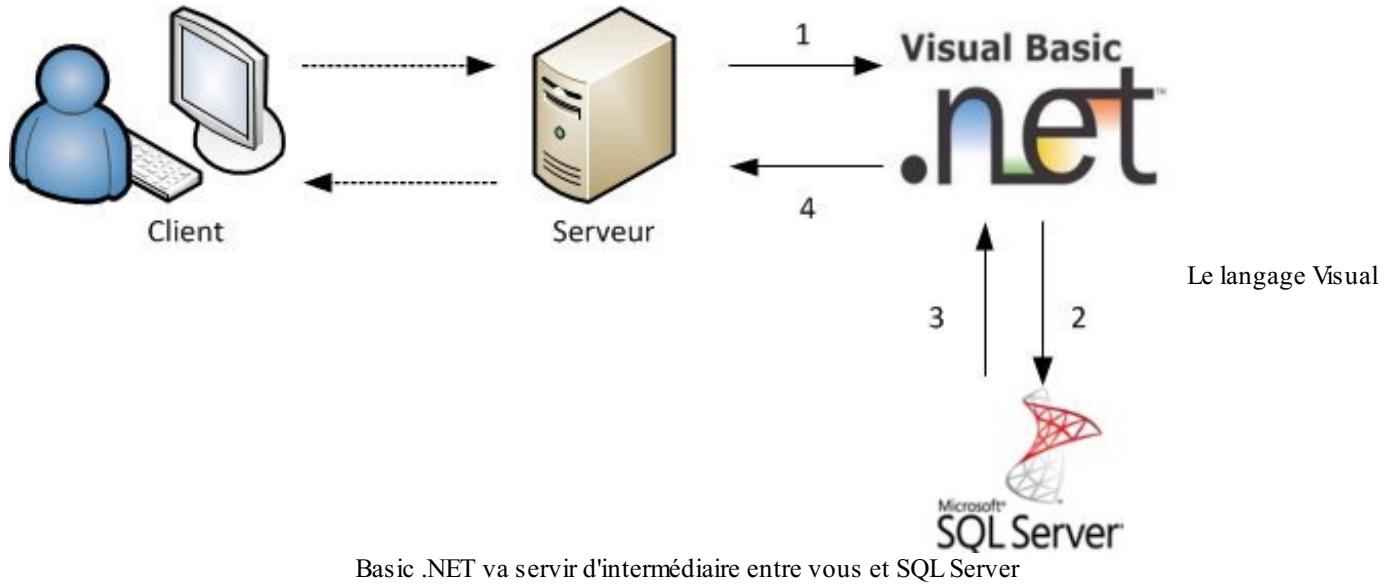
VB .NET et SGBD

Nous utiliserons SQL Server, le seul problème qui persiste actuellement est que l'on ne peut pas parler à SQL Server directement, c'est à ce moment là qu'intervient le VB .NET (par l'intermédiaire d'ADO.NET). Le langage Visual Basic .NET va servir d'intermédiaire entre vous et SQL Server.



C'est-à-dire ?

Par exemple, vous allez demander à VB .NET de demander à SQL Server de faire ceci, de faire cela... , comme représenté à la figure suivante.



Voici ce qu'il peut se passer lorsque le serveur a reçu une demande d'un client qui veut poster un message sur un chat :

1. Le serveur utilise VB .NET.
2. Le code VB .NET demande à SQL Server d'enregistrer un message dans la base de données.
3. SQL Server va « répondre » à VB .NET en lui disant « OK, c'est bon, je stocke le message ».
4. VB .NET va alors renvoyer au serveur que tout s'est bien déroulé (donc SQL Server a bien fait son travail). 😊

Lexique

Cette sous-partie va présenter un peu de vocabulaire spécifique aux bases de données, ces termes sont fréquemment utilisés, vous pourrez vous en servir à tout moment.

Base de données relationnelle

Une base de données relationnelle est un type de base de données qui utilise des tables pour le stockage d'informations. Elles utilisent des valeurs issues de deux tables, pour associer les données d'une table aux données d'une autre table. En règle générale, dans une BDD relationnelle, les informations ne sont stockées qu'une seule fois.

Table

Une table est un composant d'une base de données qui stocke les informations dans des enregistrements (lignes) et des champs (colonnes). Les informations sont généralement regroupées en catégories au niveau d'une table. Par exemple, il y aura la table des Clients, des Produits ou des Commandes... dans le contexte d'un magasin.

Enregistrement

L'enregistrement est l'ensemble des informations relatives à un élément d'une table. Les enregistrements sont les équivalents des lignes d'une table. Par exemple, une table Clients contient les caractéristiques d'un client en particulier.

Champ

Un enregistrement est composé de plusieurs champs. Chaque champ d'un enregistrement contient une seule information sur l'enregistrement. Par exemple, un enregistrement Client peut contenir les champs CodeClient, Nom, Prénom...

Clé primaire

Une clé primaire est utilisée pour identifier, de manière unique, chaque ligne d'une table. La clé primaire est un champ ou une combinaison de champs dont *la valeur est unique dans la table*. Par exemple, le champ CodeClient est la clé primaire de la table Clients, il ne peut pas y avoir deux clients ayant le même code.

Clé étrangère

Une clé étrangère représente un ou plusieurs champs d'une table, qui font référence aux champs de la clé primaire d'une autre table. Les clés étrangères indiquent la manière dont les tables sont liées.

Relation

Une relation est une association établie entre des champs communs dans deux tables. Une relation peut être de un à un, de un à plusieurs, ou de plusieurs à plusieurs. Grâce aux relations, les résultats de requêtes peuvent contenir des données issues de plusieurs tables. Une relation de un à plusieurs entre la table Clients et la table Commandes permet à une requête de renvoyer le numéro des commandes correspondant à un client.

Voilà un petit lexique que vous pourrez regarder de nouveau à n'importe quelle étape de votre apprentissage, je ne vous demande pas de tout comprendre tout de suite !

SQL Server 2008 R2

Notre SGBD

Pour dialoguer avec une base de données, il nous faut un outil adapté.

Nous allons utiliser le langage VB .NET et le langage SQL. Nous pourrons créer des bases de données et lier cette base de données avec Visual Basic Express. SQL Server est un SGBD de Microsoft.

Grâce à ce SGBD, on peut stocker des données sur une base et gérer ces données en les modifiant et en les mettant à jour. Je vous ai expliqué plus haut que le langage de requête était le SQL, mais chaque SGBD a son propre langage SQL : ainsi, SQL Server utilise le langage T-SQL (Transact-SQL). Je ne vais pas entrer dans les détails, car nous utiliserons un langage SQL indépendant de tout SGBD.

SQL Server 2008 R2

Microsoft a déjà sorti plusieurs versions de SQL Server, nous allons utiliser dans la suite de ce tutoriel **SQL Server 2008 R2**. J'ai décidé de prendre ce SGBD, car il s'adapte très bien avec les suites Express de Visual Studio.



J'ai déjà une version antérieure de SQL installée sur mon ordinateur. Cela pose-t-il problème ?

Non, ça ne pose pas de problème, il n'empêche qu'il faudra alors vous débrouiller pour faire les mêmes manipulations que moi.

La première version de SQL Server est sortie en 1989 sur les plateformes UNIX et OS/2, mais, depuis, Microsoft a préféré mettre SQL Server uniquement sous un système d'exploitation **Windows**. En 1994, Microsoft a sorti la version 6.0 et 6.5 sur la plateforme

Windows NT. Ensuite Microsoft a continué de commercialiser le moteur de base de données sous le nom de Microsoft SQL Server et a publié la version 2008 de Microsoft SQL Server, et enfin la version 2008 R2.

C'est pourquoi je vous conseille d'utiliser la même version que moi, car les anciennes ne sont plus très bien adaptées pour nos manipulations. De ce fait, certains passages évoqués dans le cours sont peut-être différents voire inexistant dans les versions précédentes. De plus ce SGBD est plutôt léger et vous permettra de suivre le tutoriel dans les meilleures conditions possibles.

Installation de SQL Server 2008 R2

Étape par étape

Avant toute chose, il faut [télécharger SQL Server](#). Vous devriez arriver sur une page semblable à la figure suivante. Cliquez sur Télécharger.

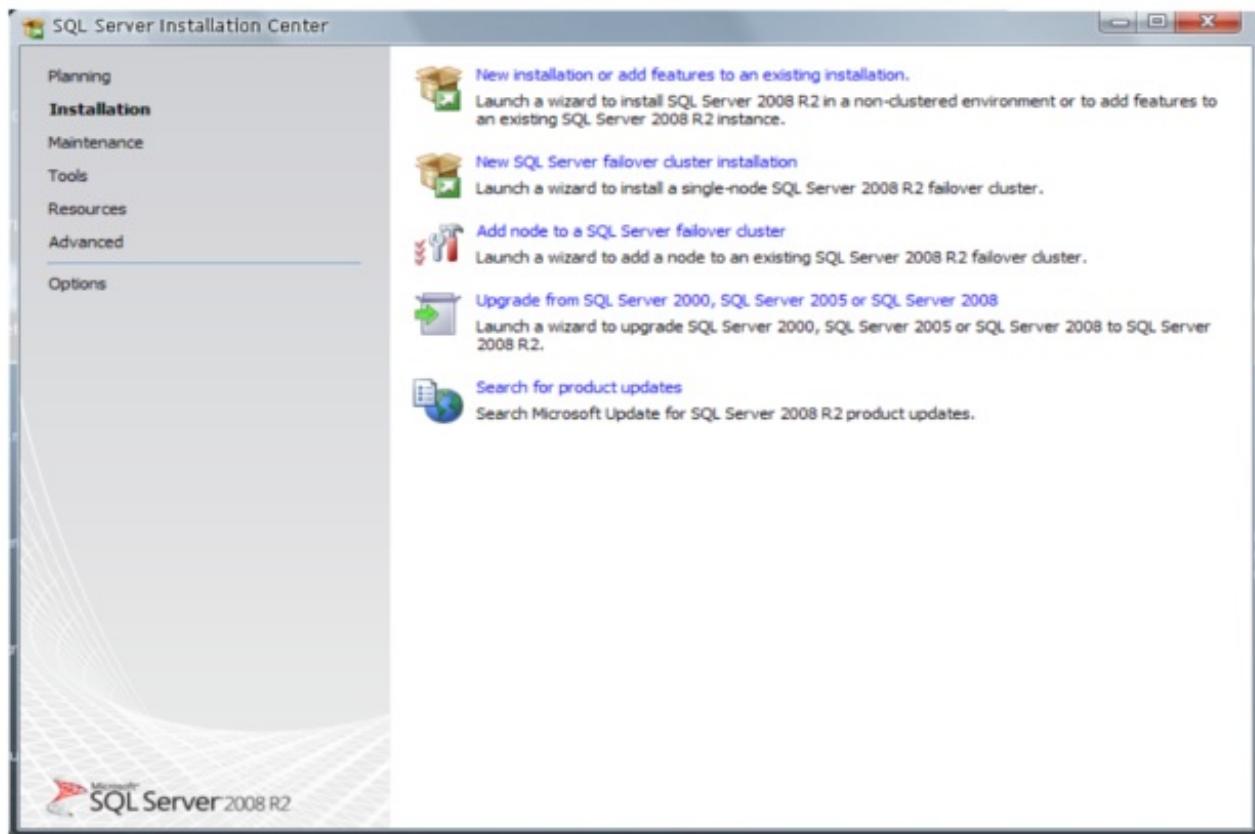


La page de téléchargement de SQL Server

Passons directement à l'installation ! Elle est facile à réaliser, il suffit juste de suivre les consignes de l'assistant pendant toutes les étapes de l'installation.

Accueil de l'installation

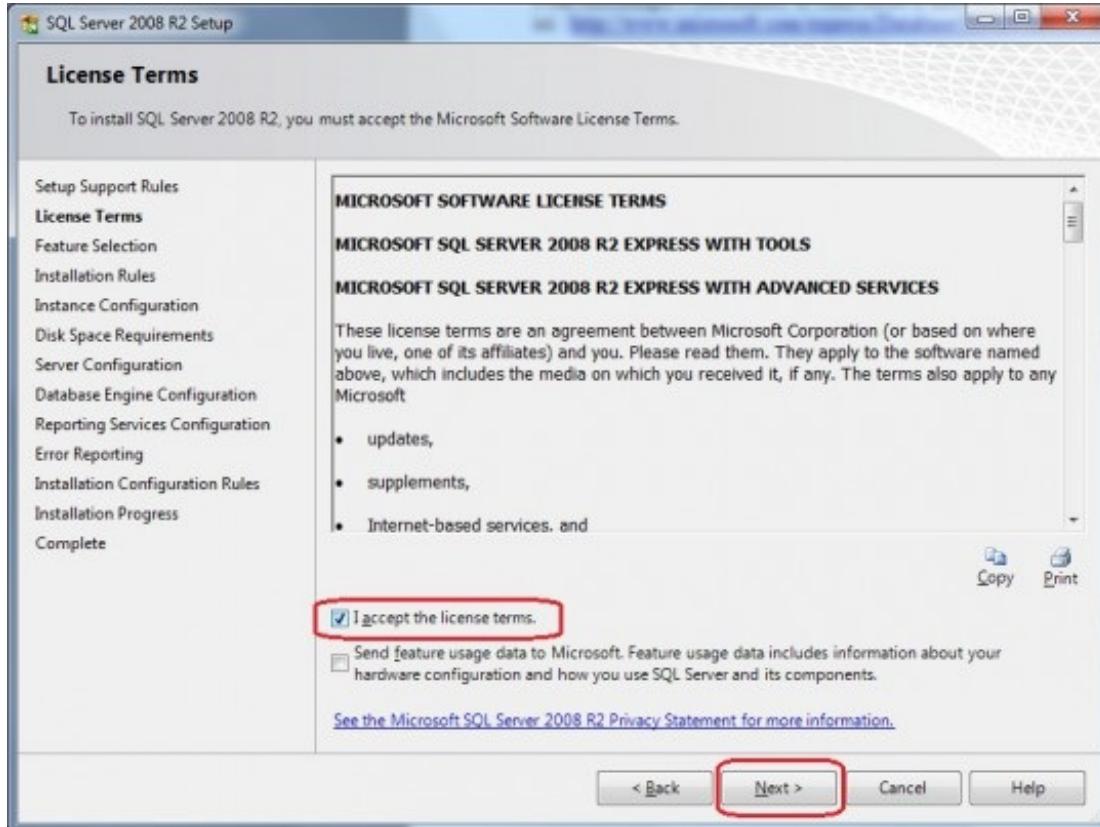
Le centre d'installation de Microsoft SQL Server aide à lancer l'installation. Sur la fenêtre visible à la figure suivante, le centre d'installation vous propose plusieurs options, pour l'instant une seule nous intéresse : cliquez sur **New Installation or add features to an existing installation**.



Plusieurs options sont proposées

Contrat de licence

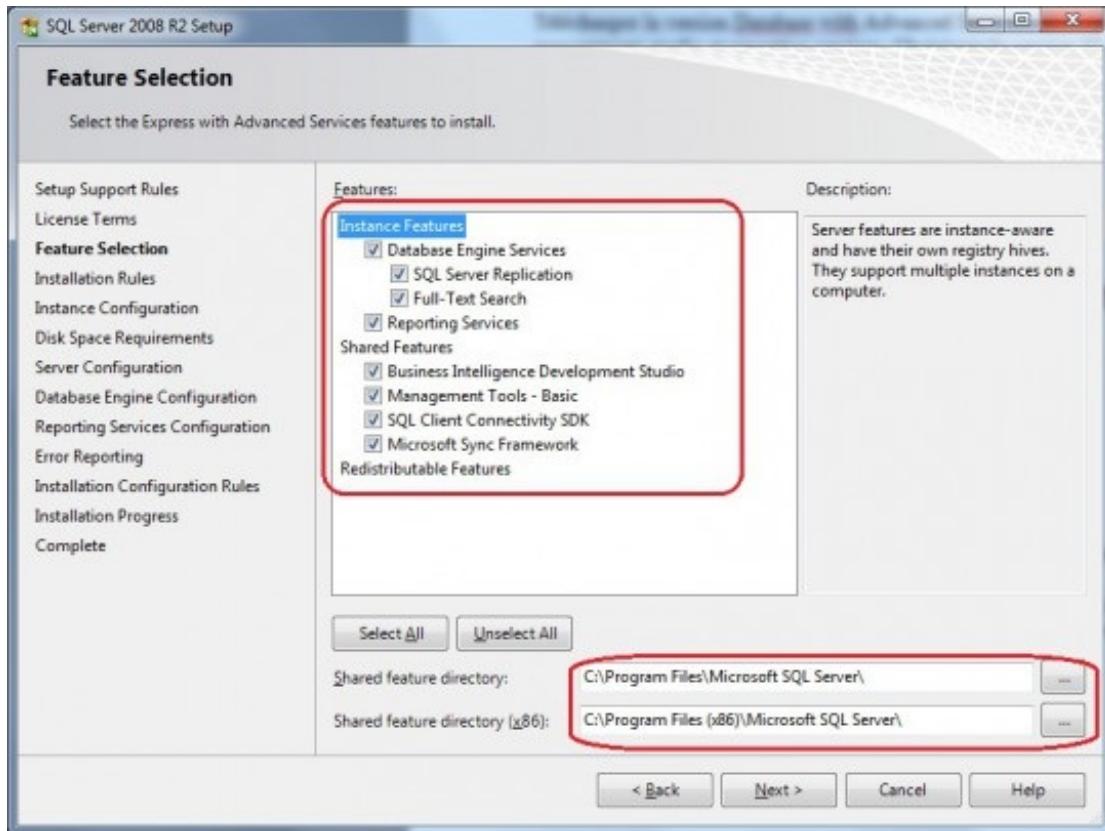
Lisez puis acceptez les termes du contrat de licence (voir figure suivante). Cela fait, appuyez sur Suivant.



Le contrat de licence

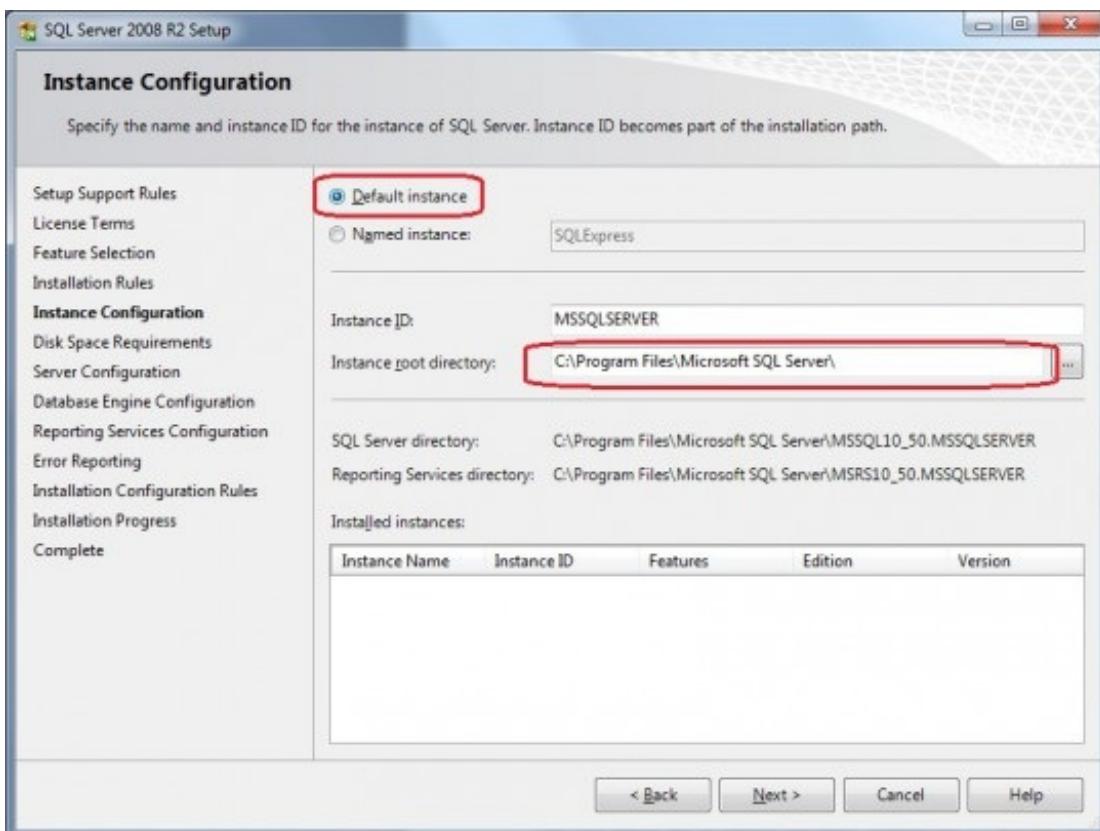
Chemin d'installation

Laissez les *features* par défaut et choisissez le dossier dans lequel vous souhaitez que le logiciel s'installe si besoin (voir figure suivante). Cliquez ensuite sur Suivant.



Sélection des features

Comme à la figure suivante, cliquez sur Default instance, changez le répertoire d'installation si besoin, puis cliquez sur Next.

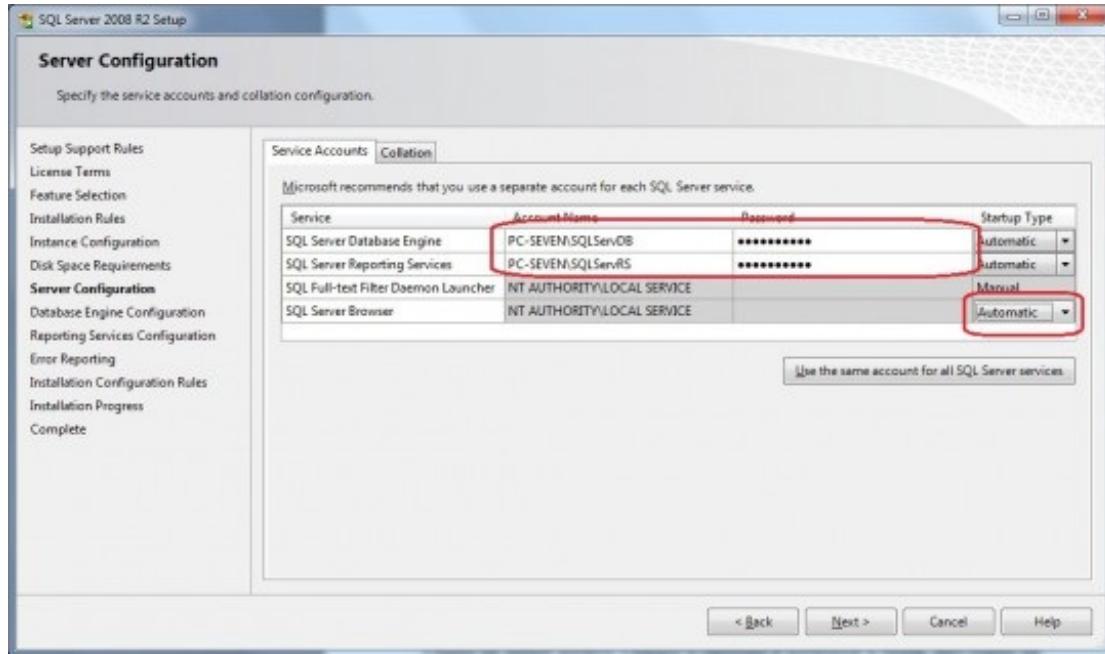


Cliquez sur Default

instance

Configuration des comptes de services

Les noms de comptes déjà renseignés par votre PC sont censés être corrects, laissez les tels quels. Puis passez Server browser en Automatique (voir figure suivante) et finalement, cliquez sur l'onglet Collation.

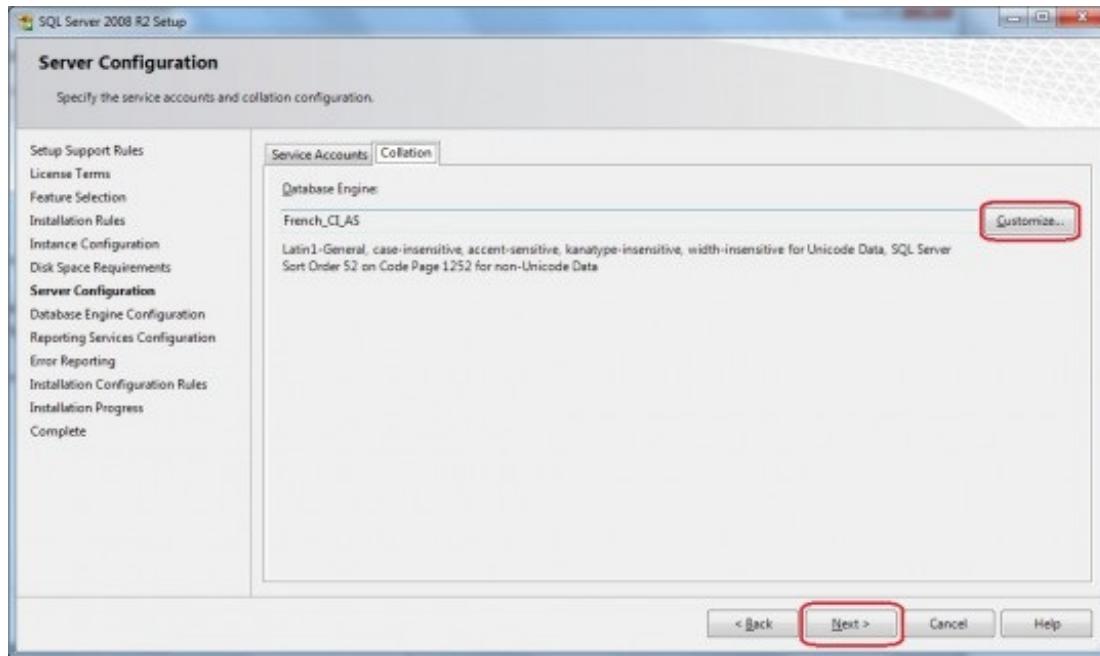


Les noms de comptes par

défaut sont censés être corrects

Onglet Collation

Choisissez le code page que vous souhaitez utiliser pour votre base de données, ici nous garderons le code page par défaut (voir figure suivante).

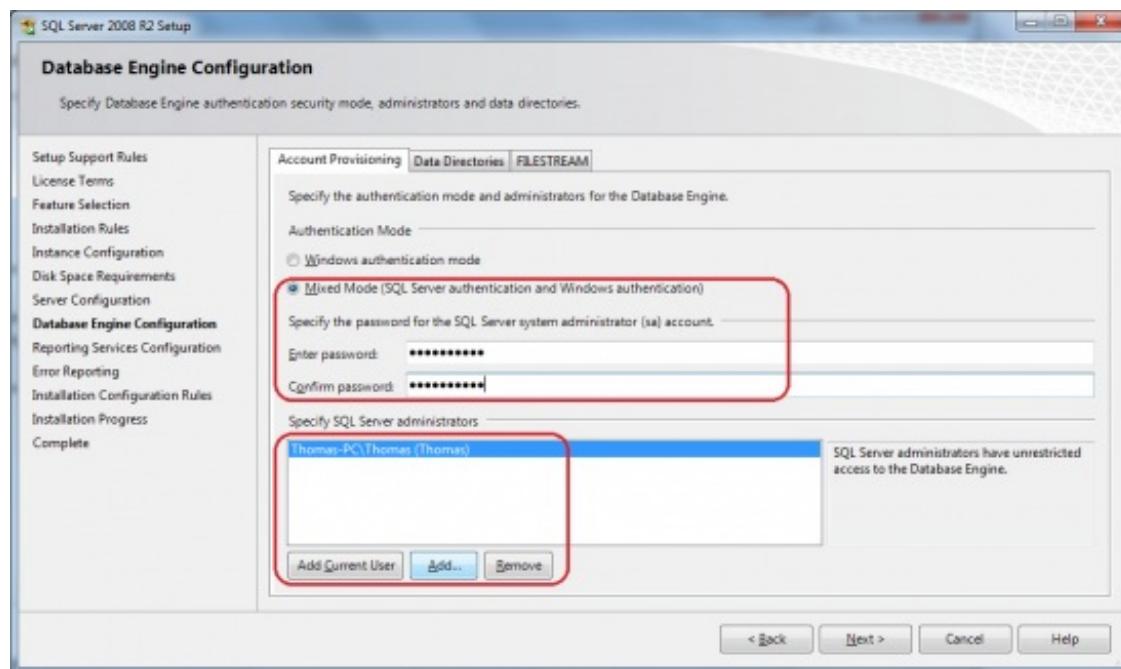


Nous garderons le code

page par défaut

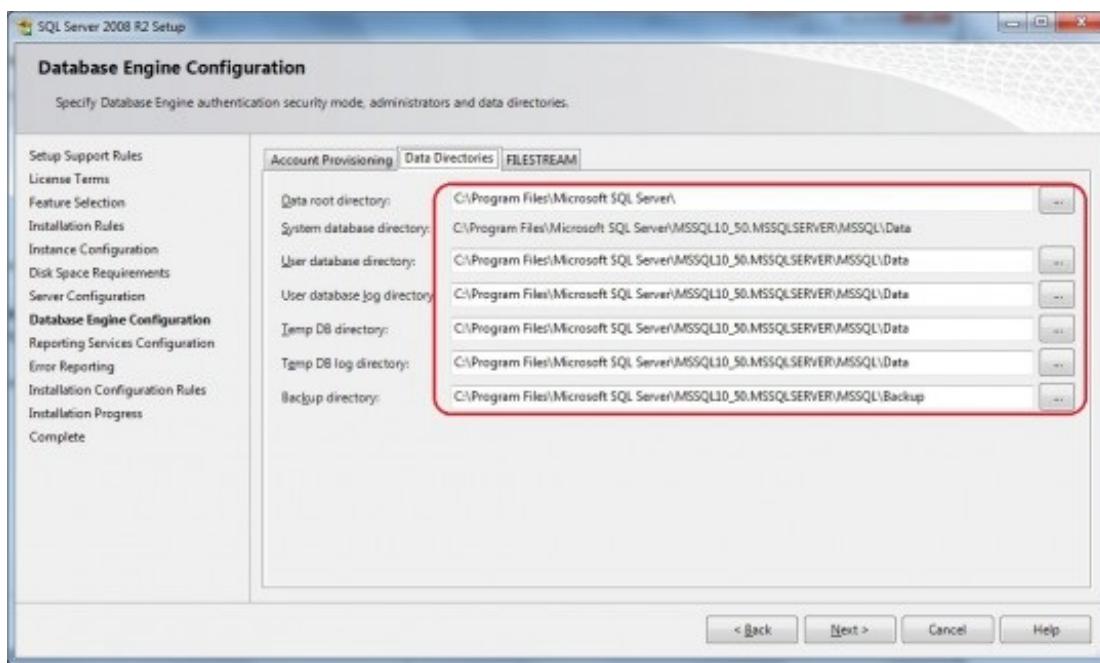
Database Engine Configuration

À l'étape Database Engine Configuration (voir figure suivante), choisissez Mixed mode, cela vous permettra de créer le compte SA (*System Administrator*), choisissez un mot de passe pour SA. Ajoutez les utilisateurs à qui vous souhaitez donner les droits d'administration de la base, puis cliquez sur l'onglet Data Directories.



Créez le compte SA

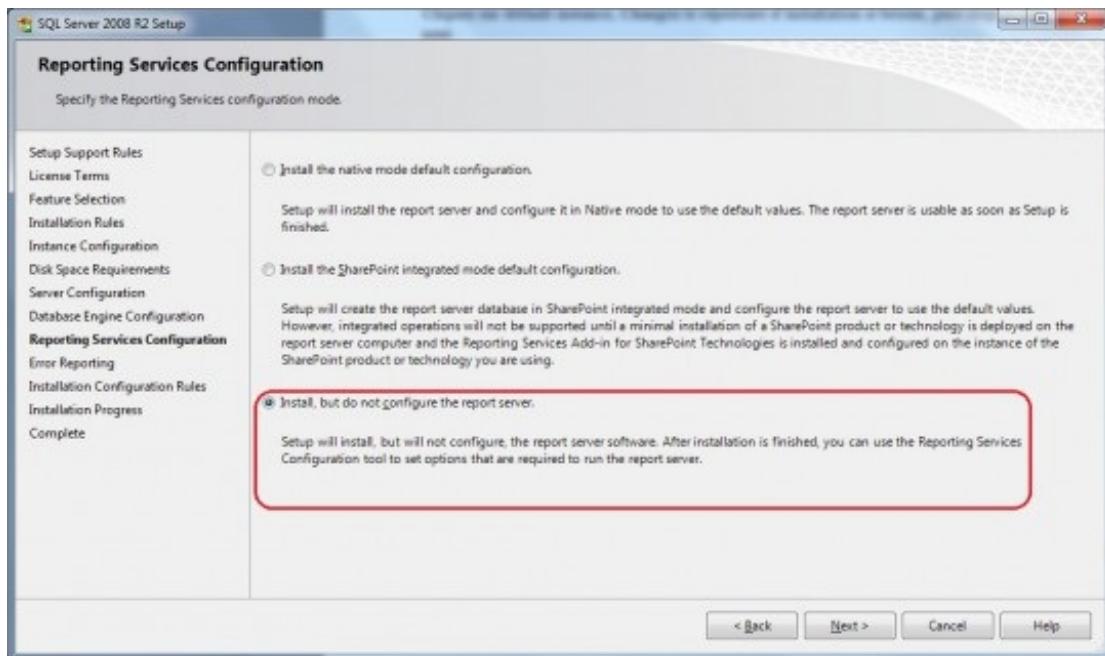
Modifiez les répertoires où seront stockés les fichiers de données si besoin (voir figure suivante), puis cliquez sur Next.



Modifiez les répertoires

Reporting Services Configuration

Comme à la figure suivante, cliquez sur Install, but not configure the report server.

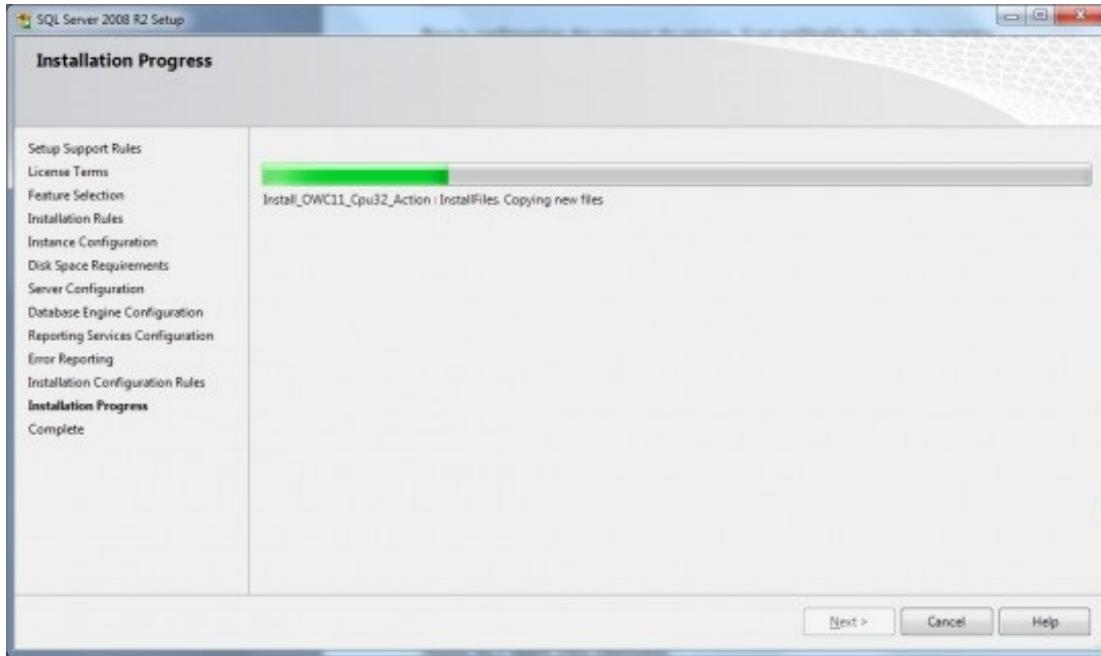


Error reporting

Normalement, il n'y a aucune erreur, cliquez sur Next.

Téléchargement et installation

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

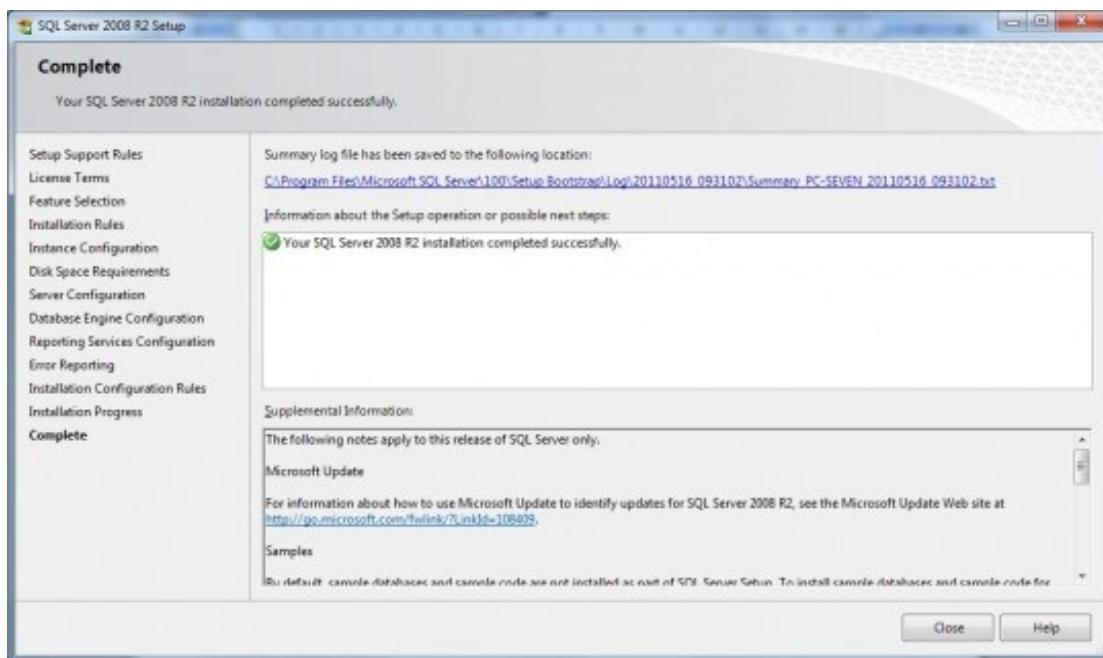


Barre de progression de

l'installation

Fini !

Une nouvelle page apparaît (voir figure suivante), SQL Server 2008 R2 est installé !



SQL Server 2008 R2 est installé !

Découverte de l'interface L'interface de SQL Server 2008 R2

Vous avez maintenant installé le SGBD SQL Server 2008 R2. Il est temps de découvrir un peu l'interface de travail. Allez dans Démarrer > Tous les programmes > Microsoft SQL Server 2008 R2 > SQL Server Management Studio. Un petit temps de chargement s'écoule jusqu'à ce que la fenêtre visible à la figure suivante apparaisse : cette fenêtre va vous demander de vous connecter au serveur.

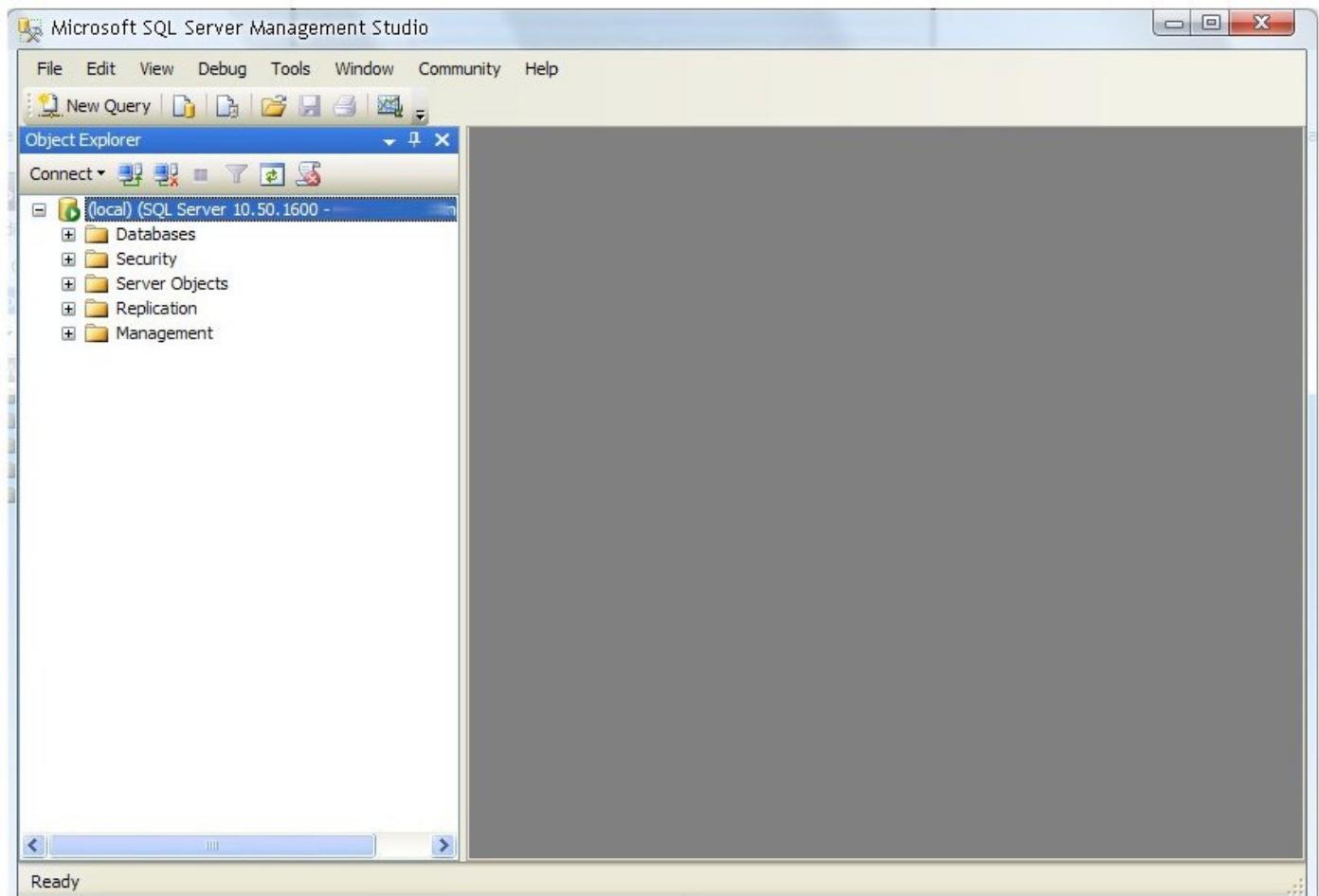


Connexion au serveur

Remplissez de la même manière : « (local) » pour le server name et « sa » comme login. Puis entrez le mot de passe que vous avez choisi lors de l'installation.

Si la connexion est correcte, vous accédez à l'interface de SQL Server Management Studio.

On constate rapidement que l'interface principale de SQL Manager est très simple. Elle est formée d'un menu contextuel, d'une barre d'outils et d'une fenêtre Explorer à gauche, comme à la figure suivante.



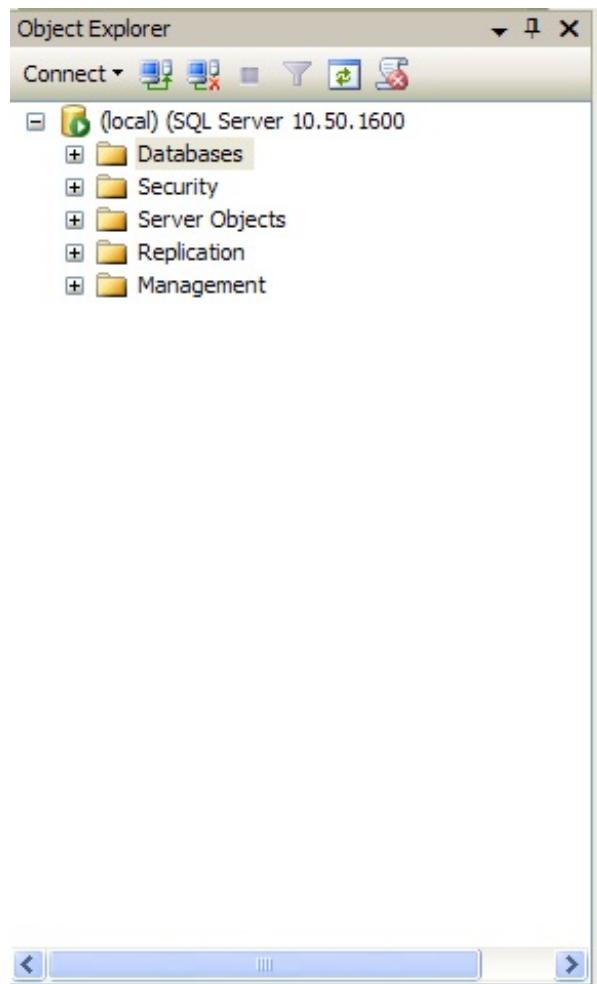
La page d'accueil

La barre d'outils (voir figure suivante) de SQL Server Management Studio permet d'accéder à certaines fonctionnalités, je ne les détaillerai pas ici.



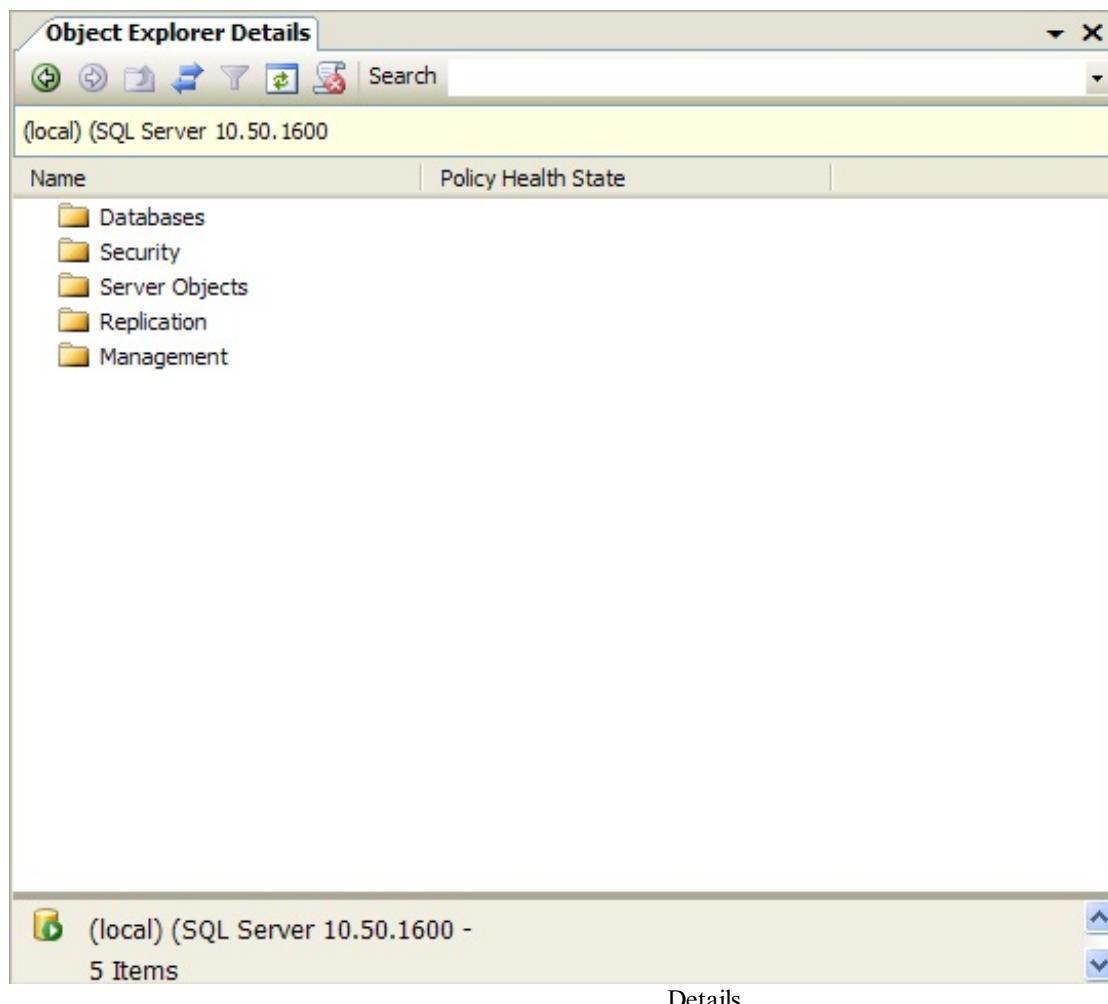
La barre du menu permet aussi d'accéder à plusieurs options.

Vous voyez à la figure suivante le menu Object Explorer : il affiche les objets du serveur de base de données en une hiérarchie conçue pour faciliter la navigation. On peut cliquer sur le symbole + à gauche du dossier pour pouvoir afficher son contenu.



Le menu Object Explorer

La fenêtre visible à la figure suivante n'est pas encore affichée, mais on peut la faire apparaître grâce à l'aide de la touche F7 ou par le menu View > Object Explorer Details.



La fenêtre Object Explorer

Cette fenêtre nous affiche les détails des objets sélectionnés sur la fenêtre Explorer.

- Une base de données est indispensable pour stocker des informations qui seront destinées à être partagées entre plusieurs utilisateurs.
- Nous allons utiliser SQL Server 2008 pour créer notre base de données.
- La base de données contient des tables, elles-mêmes contiennent des champs.

Introduction au langage SQL

Dans l'introduction aux bases de données, je vous ai parlé d'un certain **langage SQL**. Nous allons faire une première approche de ce fameux langage afin de nous familiariser avec ce dernier. Le langage SQL (*Structured Query Language*) permet de dialoguer avec la base de données. Je vous ai aussi signalé qu'il existait différentes versions du langage SQL en fonction de la base de données utilisée. Mais SQL dispose également d'une syntaxe élémentaire, normalisée et indépendante de toute base de données.

Rechercher des informations



Avant toute chose, sachez que ce chapitre est loin d'être exhaustif et qu'il n'enseignera que rapidement les fondamentaux du langage SQL. Si vous voulez aller plus loin, je vous conseille l'excellent cours consacré à MySQL de Taguan.

Grâce au langage SQL, vous pourrez rechercher certains enregistrements afin de les extraire, dans l'ordre dans lequel vous souhaitez les extraire. Par exemple, vous pouvez créer une instruction SQL qui extrait les informations de plusieurs tables simultanément, ou alors un enregistrement spécifique.

Pour ce faire, nous allons utiliser l'instruction **SELECT** : cette instruction est utilisée pour renvoyer des champs spécifiques d'une ou de plusieurs tables de la base de données.

Par exemple, cette instruction...

Code : SQL

```
SELECT Artiste, Titre FROM Musiques
```

... renverra la liste des Artistes et des Titres de tous les enregistrements de la table Musiques.

Vous pouvez aussi utiliser le symbole « * » à la place de la liste des champs pour lesquels vous souhaitez la valeur :

Code : SQL

```
SELECT * FROM Musiques
```

Ainsi, cette requête vous renverra un tableau contenant toutes les informations sur toutes les musiques présentes dans la table.

Nous pouvons aussi *limiter* le nombre d'enregistrements sélectionnés. Nous allons pour cela utiliser un ou plusieurs champs qui vont permettre de filtrer la recherche. Nous allons maintenant voir certaines clauses disponibles qui permettront ce filtrage.

La clause WHERE

La clause **WHERE** va permettre de spécifier les conditions : seule une partie des enregistrements seront concernés. Prenons un exemple : nous voulons retrouver les informations concernant le titre *Nothing Else Matters*.

Code : SQL

```
SELECT * FROM Musiques WHERE Titre = 'Nothing Else Matters'
```



Vous avez sûrement remarqué les guillemets : lorsque l'on utilise la syntaxe de la clause **WHERE**, les guillemets servent de délimiteurs de chaînes de caractères.

On peut traduire la commande **WHERE** par « où » en français, donc l'instruction **WHERE** `Titre = 'Nothing Else Matters'` peut se traduire par « où le champ Titre est égal à *Nothing Else Matters* ».

Il existe en SQL des conditions associées à la clause **WHERE** tout comme en VB .NET :

Condition	Valeur
=	Égal
!=	Different
<	Inférieur
>	Supérieur
<=	Inférieur ou égal
>=	Supérieur ou égal
AND ou &&	Et
OR ou	Ou

Par exemple :

Code : SQL

```
SELECT * FROM Musiques WHERE Titre != 'Nothing Else Matters'
```

Ce code affichera toutes les informations dont le champ est différent de Nothing Else Matters. Nous allons maintenant découvrir d'autres conditions de la clause **WHERE**...

La clause WHERE... IN

Cette clause permet de renvoyer tous les enregistrements qui répondent à une liste de critères. Par exemple, nous pouvons rechercher les artistes nés en France :

Code : SQL

```
SELECT * FROM Artistes WHERE Pays IN ('France')
```

Clause WHERE... BETWEEN

Nous pouvons sélectionner quelques enregistrements se trouvant entre deux critères de recherche spécifiés. La requête suivante permet de récupérer la liste des albums de l'année 2011 :

Code : SQL

```
SELECT * FROM Albums WHERE DateSortieAlbum BETWEEN '01/01/11' AND '31/01/11'
```



Vous remarquerez l'utilisation de **AND** pour dire « et » : « Entre... et... ».

La clause WHERE... LIKE

Cette clause permet de renvoyer tous les enregistrements pour lesquels il existe une condition particulière dans un champ donné. Ci-dessous la commande qui permet de rechercher tous les artistes dont le nom commence par un « s » :

Code : SQL

```
SELECT * FROM Artistes WHERE Nom LIKE 's%'
```



Le symbole % est utilisé pour remplacer une séquence de caractères quelconque.

Et enfin, une dernière clause un peu différente de celles vues jusqu'à maintenant...

La clause ORDER BY

La clause **ORDER BY** permet de renvoyer les enregistrements dans un ordre donné. Il en existe deux :

Option	Traduction
ASC	Ordre croissant
DESC	Ordre décroissant

Il peut y avoir plusieurs champs spécifiés comme ordre de tri. Ils sont analysés de la gauche vers la droite.

Code : SQL

```
SELECT * FROM Artistes ORDER BY Nom DESC, Prenom ASC
```

Ce code va retourner les artistes triés par ordre décroissant sur le nom, et en cas d'égalité, par ordre croissant sur le prénom.

Ajouter des informations

Grâce au SQL, nous pouvons aussi ajouter des informations dans une table avec la commande **INSERT INTO**. Pour ce faire, il faut indiquer la table dans laquelle on souhaite intégrer une ligne ainsi que la liste des champs pour lesquels on spécifie une valeur, et enfin la liste des valeurs correspondantes. Voyons cela avec un petit exemple :

Code : SQL

```
INSERT INTO Musiques (Numero, Artiste, Titre, Album) VALUES (5,
'Blood Brothers', 'Iron Maiden', 'Brave New World')
```

On utilise **INSERT INTO** en spécifiant à côté dans quelle table nous allons ajouter des informations (ici la table **Musiques**) ; ensuite, entre parenthèses, on déclare la liste des champs dans lesquels nous allons entrer des valeurs. Le mot-clé **VALUES** va permettre d'entrer les données entre parenthèses, on entre les informations *dans le même ordre* que celles d'avant **VALUES**.

Il y a un moyen de raccourcir le code précédent :

Code : SQL

```
INSERT INTO Musiques VALUES (5, 'Blood Brothers', 'Iron Maiden',
'Brave New World')
```

En effet, lors du dernier code j'avais spécifié les champs dans lesquels je voulais entrer des valeurs, mais j'avais spécifié *tous* les champs de la table **Musiques** ! Donc au lieu de mettre tous les champs qui composent cette table, j'ai le droit de déclarer uniquement le nom de la table dans laquelle je souhaite ajouter des informations.



Si vous n'indiquez pas de valeur pour un certain champ dans une table, celui-ci doit prendre la valeur **NULL** par défaut : l'instruction **INSERT** exige que tous les champs soient remplis. Donc vous devez mettre **NULL** si vous n'avez aucune

 information à placer dans un champ.

Comme ceci :

Code : SQL

```
INSERT INTO Musiques VALUES (5, 'Blood Brothers', NULL, NULL)
```

La mise à jour d'informations

On peut modifier certains champs d'enregistrements existants grâce au mot-clé **UPDATE** : cette instruction permet de mettre à jour plusieurs champs de plusieurs enregistrements d'une table, à partir des expressions qui lui sont fournies.

Pour ce faire, vous devez indiquer le nom de la table à mettre à jour ainsi que les nouvelles valeurs à affecter aux champs.

Le mot-clé **SET** va permettre cette affectation.

Si l'on veut que les modifications ne se fassent que sur un ensemble limité d'enregistrements, on doit utiliser la clause **WHERE**. Si aucune clause n'est indiquée, la modification se fera sur tous les enregistrements de la table !

Par exemple, pour modifier le titre d'une chanson , on utilise le code suivant :

Code : SQL

```
UPDATE Musique SET Titre = 'nouveau titre' WHERE Numero = 4
```

La modification du titre de la chanson ne portera ici que sur l'enregistrement qui porte le numéro 4 ! Tandis que si l'on souhaite que la mise à jour soit globale, inutile de mettre de clause.

Supprimer des informations

Il se peut que l'on soit amené à supprimer un ou plusieurs enregistrements d'une table, il existe pour cela l'instruction **DELETE FROM**. On doit alors fournir *au minimum* le nom de la table sur laquelle va s'effectuer la suppression.



Si il y a *uniquement* le nom de la table de renseignée, alors tous les enregistrements de cette table seront supprimés.

Pour limiter la suppression, on réutilise la clause **WHERE**.

La commande suivante efface tous les enregistrements de la table Musiques :

Code : SQL

```
DELETE FROM Musiques
```

Tandis que celle-ci ne supprime qu'un seul enregistrement :

Code : SQL

```
DELETE FROM Musiques WHERE Numero = 2
```

- **SELECT** permet d'effectuer des recherches et de récupérer des données.
- **UPDATE** met à jour des données.
- **INSERT** ajoute des données.
- **DELETE** supprime des données.

Création et remplissage de la BDD

Bon, maintenant que vous avez les outils nécessaires pour concevoir et créer votre base de données, on va commencer par la remplir ! Pour communiquer entre le programme et la base de données, nous allons mettre en place un fil rouge dans la continuité des premiers chapitres. Nous allons donc créer une base de données référençant une bibliothèque musicale et un programme permettant de lire ces informations, d'en ajouter, de les modifier...

Création de notre base de données (BDD)

Je suppose à partir de cet instant que vous avez Microsoft SQL Server Management Studio lancé et que vous êtes connectés à votre serveur local (authentification réussie). Si ce n'est pas le cas, les forums sont là pour que vous puissiez poser vos questions et résoudre vos problèmes.

Normalement la fenêtre Object Explorer doit être sur la partie gauche de l'écran. C'est à partir de là que nous allons commencer.



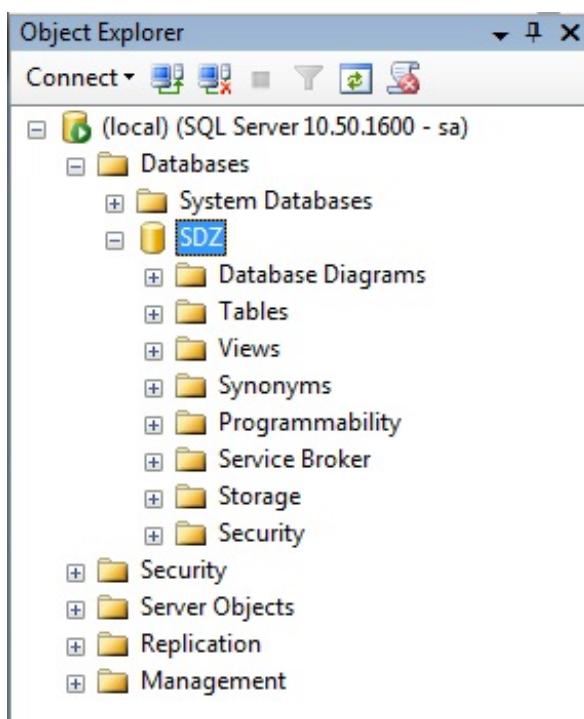
F8 permet de réouvrir cette fenêtre si vous la fermez par mégarde.

Effectuez un clic droit sur le dossier Databases, puis sélectionnez New Database. Une nouvelle fenêtre est apparue, elle va nous permettre de créer notre BDD. Comme à la figure suivante, entrez simplement le nom souhaité pour votre base de données dans le champ Database name. Pour ma part, cette base s'appellera « SDZ ».

Logical Name	File Type	Filegroup	Initial Size (MB)	Autogrowth
SDZ	ROWS	PRIMARY	3	By 1 MB, unrestricted growth
SDZ_Log	LOG	Not Applicable	1	By 10 percent, unrestricted growth

Donnez un nom à votre base de données

Vous devriez maintenant la voir en agrandissant le dossier Databases, comme à la figure suivante.



La base de données est bien là !



Votre base de données a été créée avec une icône de cylindre. C'est l'icône générique pour les bases de données.

Voilà, votre première base de données est créée ! Allons tout de suite la remplir avec de jolies données !

La création de la table

L'analyse

Nous allons donc créer une table pour y insérer des données ! En utilisant le gestionnaire de base de données, nous allons pouvoir utiliser l'interface qui nous est fournie pour créer manuellement nos tables (un peu comme Microsoft Access).

Commençons tout de suite avec la création de notre première table !

Agrandissez votre BDD dans l'Object Explorer et effectuez un clic droit sur Tables, puis New table.

L'interface de SQL Server se remplit, vous vous retrouvez avec deux nouvelles fenêtres. Celle du centre, la plus importante, contient les champs à créer dans la table. En dessous de celle-ci s'afficheront les propriétés des différentes colonnes (nous allons essentiellement utiliser cette partie). La seconde fenêtre, à droite, contient les propriétés de la table actuelle.



La fenêtre de propriétés doit vous sembler familière, elle a la même structure que celle de l'IDE de Visual Basic (lors de créations graphiques).

Commençons par spécifier le nom de notre table. C'est une bibliothèque musicale que nous voulons. Commençons donc par la table Musiques.

Pour nommer votre table, remplacez le Table_1 de la fenêtre de droite par Musiques.

Nous sommes prêts pour créer nos champs. Vous vous souvenez du tableau schématisant une base de données, et que je vous ai montré au début du chapitre ?

Table « Musiques »

Numéro	Artiste	Titre	Album
1	Guns N' Roses	Don't Cry	Greatest Hits

2	Metallica	Nothing Else Matters	Metallica
3	Saez	Jeune et Con	Jours étranges
4	Noir Désir	Un jour en France	666.667 Club

Nous allons nous en inspirer.

Je suggère donc de créer notre table avec cinq champs :

- L'ID. Correspond au numéro de la musique. Cependant, dans les bases de données, il est préférable d'avoir un identifiant de ligne unique (que l'on va incrémenter automatiquement de 1 à chaque nouvelle ligne et que nous ne pourrons pas modifier). C'est pour cela que ce champ s'appelle **ID**.
- L'artiste. Champ que nous allons donc nommer **Artiste**.
- Le titre, que nous allons appeler **Titre**.
- L'album, avec le champ **Album**.
- Et finalement le classement de la musique : **Classement**.

Il faut désormais réfléchir à ce que nous allons insérer dans chacun de ces champs. Ainsi, les champs **Artiste**, **Titre** et **Album** devront accepter les chaînes de caractères.

ID et **Classement**, quant à eux, doivent accepter des nombres.

Nous nous fixons comme règle qu'une musique doit forcément avoir un titre et un artiste, mais l'album et le classement ne sont pas nécessaires.



Donc **Album** et **Classement** pourront être **NULL**.

De plus nous avons dit que **ID** doit être en incrément automatique.

Avec toutes ces informations nous pouvons remplir notre table avec ces champs.

La création

Pour créer un champ, entrez son nom dans **Column name** (dans la fenêtre au centre), son type de données et si on lui autorise les **NULL**.

ID

On commence par l'**ID**. Le champ se nomme **ID**, son type est **int** et on ne lui autorise pas les **NULL**. De plus, pour activer l'incrément automatique, rendez-vous dans la fenêtre du bas.

Recherchez la ligne **Identity specification**, puis modifiez-la en **Yes**, comme à la figure suivante.

Column Properties	
(Name)	ID
Allow Nulls	No
Data Type	int
Default Value or Binding	
Table Designer	
Collation	<database default>
Computed Column Specification	
Condensed Data Type	int
Description	
Deterministic	Yes
DTS-published	No
Full-text Specification	
Has Non-SQL Server Subscriber	No
Identity Specification	
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1
Indexable	Yes
Is Columnset	No
Is Sparse	No
Merge-published	No
Not For Replication	No
Replicated	No
RowGuid	No
Size	4

Modification d'une ligne



Cela indique que ce champ servira d'identité de la table (ID).

L'incrémentation automatique s'est mise d'elle-même, vous n'avez plus rien à faire, passons au champ suivant.



Le type `text` est préférable lors du stockage d'un grand nombre de caractères.

Titre

Idem, avec comme nom `Titre`.

Artiste

Dans une ligne qui est apparue, entrez le nom du champ : `Artiste`.

Concernant son type, il n'existe pas de type `String`. Le type `varchar(50)` spécifie que c'est une chaîne de caractères, longue de 50 caractères maximum. Nous pouvons modifier cette valeur, spécifiez « 255 » à la place de « 50 ». Vous voici donc avec un champ acceptant 255 caractères maximum.

On a dit que l'artiste ne pouvait pas être nul (ha ha ! la blague ! 😊), donc il ne faut pas cocher la case `Allows Nulls`.

Album

Créez le champ `Album` de la même manière en prenant soin de cocher la case pour autoriser les `NULL`, comme convenu.

Classement

Un champ `Classement` avec cette fois des données numériques (donc un `int`).

Voilà tous nos champs de créés. On peut sauvegarder notre table. 😊 Notre table est terminée, vous pouvez la voir dans le

dossier Tables de notre BDD.

Le remplissage de données

Nous avons trois moyens d'ajouter des données dans notre base de données.

- Le langage SQL (vu précédemment) ;
- Le code Visual Basic ;
- L'interface de SQL Server.

Évidemment, chacun a ses préférences en fonction de ses connaissances et de son niveau.

Via des requêtes SQL

L'utilisation de requêtes SQL peut nous permettre de créer notre base de données et de la remplir.

Pour utiliser ces requêtes nous allons devoir les soumettre à notre base de données qui va les exécuter.

Je vais vous donner les requêtes qui peuvent être utiles, et vous dire comment les utiliser.

- Insertion :
Si je souhaite ajouter une musique par exemple : `Insert into Musiques (Titre, Artiste) values ('Hotel California', 'Eagles')`.
- Modification :
Pour ajouter un classement à une musique : `UPDATE Musiques SET Classement=7 WHERE Titre='Hotel California' AND Artiste='Eagles'`.
- Affichage :
Pour afficher la table sous forme de tableau : `SELECT * FROM Musiques`.

Ces exemples de commandes SQL sont là à titre informatif, consultez un tuto sur le SQL pour plus de détails.

Pour exécuter une requête, effectuez un clic droit sur la base SDZ, puis New query. Entrez votre requête SQL dans le panel central, puis cliquez sur Execute dans la barre de menu. Vous devriez obtenir quelque chose de similaire à la figure suivante.

The screenshot shows the SSMS interface. In the Object Explorer, under the database 'SDZ', the 'Tables' node is expanded, showing 'dbo.Musiques' and its columns (Titre, Artiste). A query window titled 'SQLQuery2.sql - (local).SDZ (sa (53))*' contains the SQL command: 'SELECT Titre, Artiste FROM Musiques'. The results grid displays five rows of data:

	Titre	Artiste
1	Dont't Cry	Guns N' Roses
2	Nothing Else Matters	Metallica
3	Jeune Et Con	Saez
4	Un Jour En France	Noir Désir
5	Hotel California	Eagles

La requête a été exécutée



Vous vous serviez souvent des exécutions de requêtes via le gestionnaire de base de données pour tester vos requêtes avant de les intégrer au code VB !

Via le code VB .NET

Il est possible de remplir une base à partir d'un programme VB .NET, cette partie est là pour ça, donc notre préambule ne se fera pas ici de cette manière.

Cependant, à la fin cette partie du tutoriel vous devriez être capables de concevoir un programme qui effectuera les opérations de création et de remplissage.

Via l'interface SQL Server

Voilà la méthode que nous allons utiliser pour le moment.

Commençons par ouvrir notre table Musiques en effectuant un clic droit dessus, puis Edit Top 200 rows.

Vous voici avec un tableau que vous allez pouvoir remplir avec vos données. Je vais donc le remplir avec les données d'exemple.



Notez que vous ne pouvez pas spécifier l'ID, car il est en auto-incrémentation. Vous devez obligatoirement entrer un artiste et un titre pour valider la ligne.

Voici à la figure suivante ma table remplie avec quelques valeurs ; j'ai entré une note et je n'ai pas rempli tous les albums pour avoir différents cas de figure.

THOMAS-PC.SDZ - dbo.Musiques					
	ID	Titre	Artiste	Album	Classement
	2	Don't Cry	Guns N' Roses	... <i>NULL</i>	<i>NULL</i>
	3	Nothing Else Mat...	Metallica	... Metallica ...	9
	4	Jeune Et Con	Saez	... Jours Etranges ...	<i>NULL</i>
	5	Un Jour En France	Noir Désir	... <i>NULL</i>	<i>NULL</i>
▶*	<i>NULL</i>	<i>NULL</i>	<i>NULL</i>	<i>NULL</i>	<i>NULL</i>

dans la table

J'ai ajouté des valeurs

Vous pouvez évidemment ajouter autant de valeurs que vous voulez !

- La création d'une base de données avec une structure complexe requiert une étude préalable.
- Le remplissage de données s'effectue soit manuellement, soit via programmation.

La communication VB .NET - BDD

Maintenant que nous avons toutes les clés en main pour pouvoir créer et remplir notre base de données, je pense que vous avez envie de vous amuser un peu avec ? Ça tombe bien, c'est dans ce chapitre que nous allons commencer à interfaçer notre code VB.NET et notre base de données fraîchement créée.

Pour cette tâche, nous allons étudier et utiliser un concept spécialement conçu pour ça : ADO.NET.

ADO.NET

ADO.NET (*ActiveX Database Objects.NET*) est une couche d'accès aux bases de données, c'est un peu le SQL Server Manager de Visual Basic. ADO.NET fournit des modules pour accéder à des BDD de différents types (Access, SQL Server, Oracle, etc.).

Pour le connecter à SQL Server, il faut **SQLServer Managed Provider**. Il faut donc importer le namespace `System.Data.SqlClient` pour pouvoir l'utiliser.

Le fonctionnement d'ADO.NET

Créons un projet console pour apprendre le fonctionnement d'ADO.NET.

On effectue donc un **Imports** :

Code : VB.NET

```
Imports System.Data.SqlClient
```

On va différencier trois types d'accès à la BDD :

- *Les accès en lecture unitaire.*

Ces accès seront effectués grâce à un objet de type `SqlCommand`. Cet objet va exécuter une requête sur la BDD (un **SELECT** par exemple). Puis nous allons nous servir de `SqlDataReader`, un objet qui va lire la réponse de la BDD à notre précédente requête. Cette combinaison d'objets va nous permettre de lire une petite quantité de données. Cette solution est souvent utilisée pour lire une entrée ou même une seule donnée.

- *Les accès en lecture de masse.*

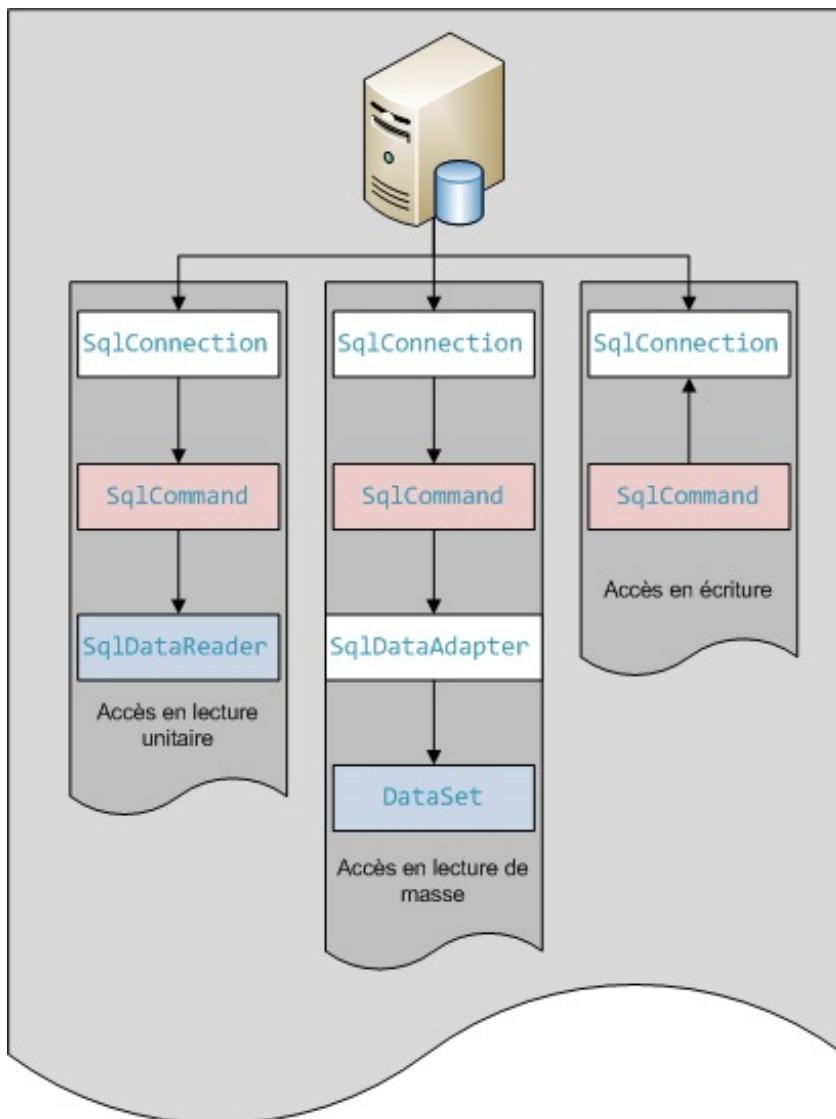
Ils seront effectués de la même manière que précédemment, seulement la lecture changera. Nous allons utiliser cette fois-ci un `SqlDataAdapter` permettant d'adapter les données lues pour un objet de type `DataSet` (comparable à un tableau). Cette méthode sera utile pour récupérer de multiples informations (toute une table par exemple).

- *Les accès en écriture.*

On va uniquement utiliser un objet de type `SqlCommand` pour exécuter une requête d'insertion de données ou de modification par exemple.

Dans tous les cas, on doit se connecter à la BDD en utilisant un objet de type `SqlConnection`.

Voici à la figure suivante un schéma pour résumer les trois cas de figure qui se présentent à nous.



Trois cas de figure se présentent à nous

Connexion à la BDD

Pour ce faire, nous devons générer une chaîne de caractères contenant les informations nécessaires pour se connecter à notre base de données, puis l'utiliser dans un objet de type **SqlConnection**.

Voici la chaîne de connexion pour notre base de données : `"Data Source=localhost;Initial Catalog=SDZ;User Id=sa;Password=*****;"`.

À modifier si votre base ne s'appelle pas pareil (ici « SDZ ») ; il faut également entrer votre mot de passe.



Vous pouvez trouver les chaînes de connexion vers d'autres types de BDD sur le site <http://www.connectionstrings.com/>.

Création de l'objet de connexion : son constructeur prend en paramètre cette chaîne de connexion, sinon il faut l'assigner manuellement avec un `.ConnectionString` :

Code : VB.NET

```
Dim Connexion As New SqlConnection("Data Source=localhost;Initial Catalog=SDZ;User Id=sa;Password=*****;")
```

Cet objet est donc de type **SqlConnection**.

Il faut ouvrir la connexion avec la méthode `Open()`.

Code : VB.NET

```
Connexion.Open()
```

Il est préférable de mettre l'ouverture de connexion dans un **Try... Catch** (voir l'annexe de la gestion des erreurs). Ainsi, si le serveur n'est pas accessible pour une raison X ou Y, le programme ne plante pas et on peut gérer l'erreur :

Code : VB.NET

```
Try  
    Connexion.Open()  
Catch ex As Exception  
    Console.WriteLine(ex.Message)  
End Try
```

Attention, il faut toujours fermer sa connexion à la fin du programme avec un **Close ()** :

Code : VB.NET

```
Connexion.Close()
```

... à mettre dans un **finally** par exemple.

Voilà, vous êtes connectés à votre base de données.

Insertion ou modification

Comme vous l'avez vu, dans tous les cas il faut effectuer une requête grâce à l'objet `SqlCommand`.

Nous allons donc voir comment générer une requête et l'exécuter. Pour commencer, je souhaiterais ajouter à ma BDD une ligne contenant un nouveau titre.

Je veux ajouter « Hotel California » des Eagles.

La requête SQL est `Insert into Musiques(Titre, Artiste) values ('Hotel California', 'Eagles')`.

Je crée cette requête et mon objet `SqlCommand` :

Code : VB.NET

```
Dim Requete As String = "Insert into Musiques(Titre, Artiste) values  
('Hotel California', 'Eagles')"  
Dim Commande As New SqlCommand(Requete, Connexion)
```

Puis l'exécution de la requête s'effectue simplement en faisant appel à la méthode `ExecuteNonQuery ()`.

Cette méthode retourne le nombre de lignes affectées par la modification. Dans le cas d'une insertion, ce sera souvent 1. Mais certaines modifications de masses (comme des `update`) pourront en modifier un grand nombre.

On execute donc la requête (avec un **Try... Catch**).

Code : VB.NET

```
Try  
    Commande.ExecuteNonQuery()
```

```

Catch ex As Exception
    Console.WriteLine(ex.Message)
End Try

```

Et voilà une nouvelle ligne d'insérée en BDD.

Supposons que je veuille attribuer « Album inconnu » à chaque entrée qui n'a pas son album de renseigné, le programme serait :

Code : VB.NET

```

Imports System.Data.SqlClient

Module Module1

Sub Main()

    Dim Connexion As New SqlConnection("Data
Source=localhost;Initial Catalog=SDZ;User Id=sa;Password=*****;" )

    Try
        Connexion.Open()

        Dim Requete As String = "UPDATE Musiques SET
Album='Album inconnu' WHERE Album is null"
        Dim Commande As New SqlCommand(Requete, Connexion)
        Try
            Console.WriteLine("Il y a eu " &
Commande.ExecuteNonQuery() & " lignes mises à jour")
            Catch ex As Exception
                Console.WriteLine(ex.Message)
            End Try

            Commande.Dispose()
            Connexion.Close()

        Catch ex As Exception
            Console.WriteLine(ex.Message)
        End Try

    End Sub

End Module

```

Avec comme résultat console :

Code : Console

```
Il y a eu 3 lignes mises à jour
```



Il faut penser à libérer la mémoire de votre commande avec : `Commande.Dispose()`. Et surtout fermer la connexion avec `Connexion.Close()`.

Voici le résultat en BDD à la figure suivante.

ID	Titre	Artiste	Album	Classement
1	2	Don't Cry	Guns N' Roses	Album inconnu
2	3	Nothing Else Matters	Metallica	9
3	4	Jeune Et Con	Saez	Jours Etranges
4	5	Un Jour En France	Noir Désir	Album inconnu
5	6	Hotel California	Eagles	Album inconnu

Le résultat en BDD

Lecture de données

Maintenant que vous savez exécuter une requête d'ajout ou de modification, on va s'attaquer aux requêtes de lecture. Je souhaiterais connaître l'artiste qui a chanté « Jeune Et Con ».

La requête SQL est `SELECT Artiste from Musiques Where Titre = 'Jeune Et Con'`.

Je crée cette requête et mon objet `SqlCommand` :

Code : VB.NET

```
Dim Requete As String = "SELECT Artiste from Musiques Where Titre = 
'Jeune Et Con'"
Dim Commande As New SqlCommand(Requete, Connexion)
```



Il faut toujours passer l'objet de type `SqlConnection` lors de la création d'un objet `SqlCommand`.

Vous voici avec votre commande, prête à être exécutée sur votre base.

Cependant, la manière va différer entre le `SqlDataReader` et le `DataSet`. Voyons cela.

Lecture avec `SqlDataReader`

L'objet de type `SqlDataReader` va récupérer les données d'une commande `SELECT` et les emmagasiner. Il va falloir ensuite les lire une par une (donc cet objet est utile lorsqu'on a peu de valeurs ou même une seule). Pour lui dire de lire la valeur, nous allons utiliser `Read()`.



`Read()` renvoie un booléen spécifiant s'il y a ou non une valeur à lire.

Puis il faut y accéder en spécifiant le champ que l'on veut lire.

La figure suivante est un schéma pour résumer.

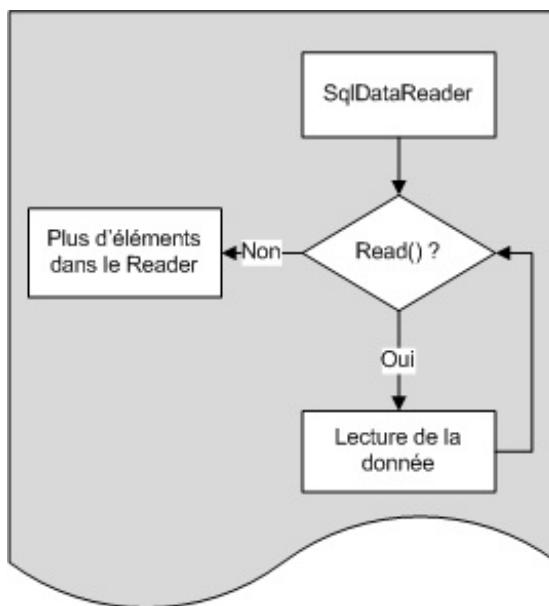


Schéma de fonctionnement du SqlDataReader

Utilisons ces informations pour continuer notre code et connaître notre artiste mystère.

On commence par exécuter la commande sur le reader :

Code : VB.NET

```
Dim MonReader As SqlDataReader = Commande.ExecuteReader()
```

Puis on teste s'il y a une valeur et on l'affiche :

Code : VB.NET

```
If MonReader.Read() Then
    Console.WriteLine(MonReader("Artiste").ToString)
Else
    Console.WriteLine("Aucun artiste trouvé")
End If
```

Et la console nous affiche fièrement :

Code : Console

```
Saez
```

La figure suivante est un schéma qui représente ce qu'il s'est passé.

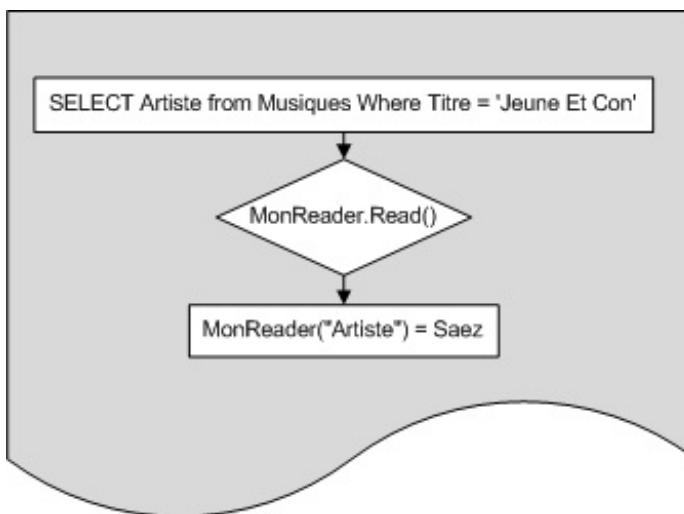


Schéma explicatif de notre exécution



Il y aurait eu de multiples possibilités, un **While** `MonReader.read()` aurait été utilisable.

Lecture avec DataSet

Nous allons désormais apprendre à stocker les données récupérées dans un `DataSet`.

Pour résumer, le `DataSet` est un objet qui va stocker une image de la base de données, que l'on va pouvoir traiter ultérieurement.

Le chapitre suivant vous apprendra tout ce qu'il y a à savoir sur les `DataSet`.

Pour le moment, voyons comment récupérer les données.

Il va falloir passer par un adaptateur, cet objet va remplir le `DataSet` avec la commande que l'on a exécutée.

Je crée donc une requête pour récupérer *toute* la table `Musiques`. Je l'applique à un `SqlCommand`, je l'exécute avec mon `SqlAdapter` et je crée mon `DataSet`.

Code : VB.NET

```

Dim Requete As String = "SELECT * from Musiques"
Dim Commande As New SqlCommand(Requete, Connexion)
Dim Adaptateur As New SqlDataAdapter(Commande)
Dim MonDataSet As New DataSet
  
```

Maintenant la ligne magique qui va remplir notre `DataSet` avec le résultat de la commande :

Code : VB.NET

```
Adaptateur.Fill(MonDataSet, "Musiques")
```

Ici, « `Musiques` » est le nom de la table de mon `DataSet` dans laquelle je vais stocker les données résultantes de la requête.

Vous voici avec un `DataSet` rempli par votre table `Musiques`. Rendez-vous au prochain chapitre pour apprendre comment utiliser notre `DataSet` et traiter ces données !

- ADO.NET permet l'accès rapide aux données, son utilisation peut cependant devenir laborieuse sur de grandes tables.
- On se connecte à la BDD en définissant une `SqlConnection` et une chaîne de connexion.
- On exécute des requêtes SQL avec `ExecuteNonQuery`.

Le DataSet à la loupe

L'ADO.NET c'est bien beau mais ça nous limite assez vite. En effet, pour de très grandes requêtes, il va falloir itérer sur l'intégralité des résultats avant d'avoir la valeur recherchée.

Imaginons maintenant que nous n'ayons plus à manipuler manuellement les données récupérées mais que des objets aient été conçus spécialement pour contenir des « Bases de données ». Je l'écris entre guillemets car on n'a aucun intérêt à stocker l'ensemble de notre base dans un objet. Par contre on va pouvoir récupérer les résultats et les manipuler plus aisément que précédemment. Cet objet existe et s'appelle un dataset, allons le découvrir.

Qu'est-ce ?

Un DataSet, pour résumer, est une représentation d'une base de données sous forme d'objet. Il contient des tables, elles-mêmes contiennent des colonnes et des lignes. On pourrait le schématiser par la figure suivante.

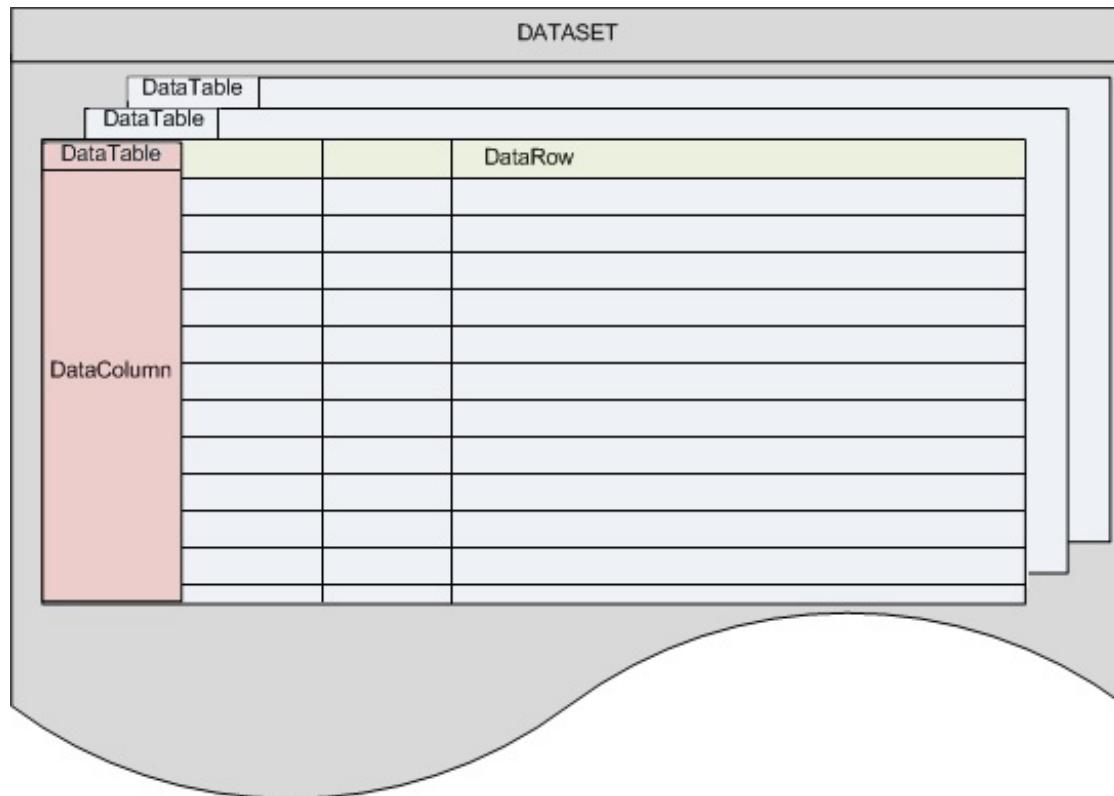


Schéma d'un DataSet

- Les différentes tables sont des objets de type DataTable.
- Les colonnes sont des objets de type DataColumn.
- Et les lignes, des DataRow.



Les items sont les « cases » du DataSet.

Nous avons donc des **tables** de type DataTable qui sont des éléments du DataSet. On y accède soit par leur index (un peu comme un tableau), soit avec le nom de la table (préférable pour être certain de la table à laquelle on accède).

Puis viennent les **lignes** et les **colonnes**, respectivement des DataRow et des DataColumn, qui sont des éléments d'une DataTable. Si on y accède en utilisant un index, on récupère la ligne ou la colonne souhaitée, sinon c'est une collection contenant la liste des lignes ou la liste des colonnes qui est renvoyée.

Et finalement, on a les **items**, dernier maillon de la chaîne, chacun d'entre eux contient un élément unique. Ce sont donc des DataItems, ce sont des éléments de DataRow. Ce sont eux qui contiennent notre donnée.

Nous allons mettre tout ça au clair par des exemples.

La lecture de données

Pour apprendre à lire nos données nous allons réutiliser la commande SQL du chapitre précédent permettant de récupérer l'ensemble de la table Musiques.

Code : VB.NET

```

Imports System.Data.SqlClient

Module Module1

Sub Main()

    Dim Connexion As New SqlConnection("Data
Source=localhost;Initial Catalog=SDZ;User Id=sa;Password=v3vk4pgd;")

    Try
        Connexion.Open()

        Dim Requete As String = "SELECT * from Musiques"
        Dim Commande As New SqlCommand(Requete, Connexion)
        Dim Adaptateur As New SqlDataAdapter(Commande)
        Dim MonDataSet As New DataSet
        Try
            Adaptateur.Fill(MonDataSet, "Musiques")

            'Analyse du DataSet

            Catch ex As Exception
                Console.WriteLine(ex.Message)
            End Try

            Catch ex As Exception
                Console.WriteLine(ex.Message)
            End Try
        End Sub

    End Module

```

On considère donc que les données sont récupérées et prêtes à être analysées où il y a écrit '*'Analyse du DataSet*'.

Je vous avait dit que lors du remplissage des données le second argument spécifiait le nom de la `DataTable` que l'on a créée.

Ainsi, pour accéder au premier artiste de notre table, on utilise :

Code : VB.NET

```
MonDataSet.Tables("Musiques").Rows(0).Item("Artiste").ToString
```

J'accède donc à la colonne avec le nom « Artiste » de la ligne n°0 de la table Musiques. Dur à suivre ! Mais une fois que vous aurez compris, tout ira mieux !

Des simplifications existent :

Code : VB.NET

```
MonDataSet.Tables("Musiques")(0)("Artiste")
```

Ce code aura le même effet que la précédente instruction. On peut s'affranchir des mots-clés `Rows` et `Item`.



`"Artiste"` peut également être remplacé par un index numérique.

Utilisons **For Each**

Vous vous souvenez de **For Each** ? Permettant de parcourir toutes les cases d'un tableau par exemple ? Eh bien utilisons cette boucle pour parcourir toutes nos lignes !

Code : VB.NET

```
'Analyse du DataSet
For Each Ligne As DataRow In MonDataSet.Tables("Musiques").Rows()
    Console.WriteLine(Ligne("Artiste").ToString & " - " &
    Ligne("Titre").ToString)
Next
```

Et voici un listing des musiques et de leur artiste.

Comme je vous le disais, `MonDataSet.Tables("Musiques").Rows()` renvoie une collection. Et une collection, au même titre qu'un tableau, a des éléments, on peut donc y accéder avec **For Each**.

Lecture plus poussée

Vous savez désormais effectuer une lecture de chaque ligne, chaque colonne, chaque case de votre table. Nous allons attaquer les *views*.

« *View* » signifie « vue » en français. On va donc générer à partir de notre `DataSet` des vues de la table. Ces vues sont pratiques, car on va pouvoir les modifier pour, par exemple, trier une certaine colonne par ordre alphabétique, etc.

Cet objet peut être utile lorsque vous avez de nombreux affichages différents à faire de votre `Table`. Plutôt que de faire des dizaines de requêtes SQL à la BDD, on en fait une seule en rapatriant la table souhaitée dans un `DataSet`, puis on effectue les différents affichages souhaités avec des `DataView`.

Pour créer un `DataView` on peut passer en paramètre du constructeur la table voulue à laquelle lier la vue. Je vais donc lui passer la table `Musiques` :

Code : VB.NET

```
Dim MonView As New DataView(MonDataSet.Tables("Musiques"))
```

Puis on effectue des modifications sur la *view*.

Tri

Le tri est la propriété `Sort` du `DataView`. On va l'utiliser avec la même syntaxe que le tri SQL, en spécifiant le champ suivant lequel trier et l'ordre (ASC pour ordre croissant et DESC pour décroissant).

Ainsi, si je veux trier suivant le champ `Artiste` et afficher le résultat :

Code : VB.NET

```
MonView.Sort = "Artiste ASC" 'On trie les artistes par ordre
                           croissant
For Each Ligne As DataRowView In MonView
    Console.WriteLine(Ligne("Titre") & " - " & Ligne("Artiste"))
Next
```



Cette fois, les lignes ne sont plus des `DataRow`, mais des `DataRowView`, car elles proviennent d'un `DataView`.

Filtre

Le filtre s'utilise avec la propriété `RowFilter`. On spécifie le champ à filtrer et la valeur voulue.

Si je veux afficher uniquement les entrées où l'album est inconnu :

Code : VB.NET

```
MonView.RowFilter = "Album = 'Album inconnu'"
```



C'est encore une syntaxe du type SQL, n'oubliez pas les guillemets autour des chaînes de caractères.

La recherche

Pour effectuer une recherche sur un champ, il faut déjà spécifier un tri du `DataView` avec la commande `Sort`. Cela indiquera au `DataView` dans quel champ effectuer la recherche.

Puis `FindRow` permettra de rechercher la valeur souhaitée. Cela retournera une collection de `RowView` (car il peut y avoir plusieurs fois la même valeur). On peut donc réutiliser un `For Each` pour afficher les résultats.

Code : VB.NET

```
MonView.Sort = "Artiste ASC"  
For Each Ligne As DataRowView In MonView.FindRows("Saez")  
    Console.WriteLine(Ligne("Titre") & " - " & Ligne("Artiste"))  
Next
```



La fonction `Find`, quant à elle, renverra l'index de la première ligne qui correspond à la recherche. Il faudra ensuite y accéder avec `MonView(x)`, où `x` est l'index.

L'ajout de données

Un `DataSet` est fait pour être lié à la BDD, cependant il vous est tout à fait possible de créer le vôtre de toutes pièces pour un programme sans liaison à la BDD. Pour substituer à un tableau par exemple.

On a déjà vu chacun des éléments composant le `DataSet`, on va donc en créer un avec les mêmes données que notre BDD pour nous entraîner.



On va donc utiliser la méthode `Add()` qui permet d'ajouter des éléments à chacun des composants du `DataSet` (`DataTables`, `DataColumns`, `DataRows`...).

Donc, commençons par créer une table à notre `DataSet` :

Code : VB.NET

```
Dim MonDataSet As New DataSet  
MonDataSet.Tables.Add("Musiques")
```

Maintenant, attaquons-nous à la création de colonnes.

Comme pour tout, vous avez le choix entre la création séparée d'un objet de type `DataColumn`, puis l'ajout en vous servant de cet objet, ou entrer directement les paramètres voulus dans le constructeur :

Code : VB.NET

```
'Création d'un objet de type DataColumn
Dim MaColonne As New DataColumn
MaColonne.ColumnName = "Titre"
MaColonne.DataType = GetType(String)
MonDataSet.Tables("Musiques").Columns.Add(MaColonne)
'Utilisation du constructeur
MonDataSet.Tables("Musiques").Columns.Add("Artiste",
GetType(String))
MonDataSet.Tables("Musiques").Columns.Add("Album", GetType(String))
MonDataSet.Tables("Musiques").Columns.Add("Classement",
GetType(Integer))
```

Et finalement, après la construction de notre structure, on peut ajouter des données.

Comme vous l'avez sûrement deviné, cela s'effectue avec la méthode `MonDataSet.Tables("Musiques").Rows.Add`. On lui passe en paramètre soit une `DataRow` construite au préalable, soit plus simplement une collection d'objets à ajouter.

Exemple, si je veux ajouter un titre à mon `DataSet` :

Code : VB.NET

```
MonDataSet.Tables("Musiques").Rows.Add("Nothing Else Matters",
"Metallica", "Metallica", 9)
```



Il faut bien respecter l'ordre dans lequel vous avez ajouté vos colonnes ! Ici mon ordre est Titre - Artiste - Album - Classement, je dois donc ajouter des valeurs en suivant cet ordre.

Donc si je veux ne pas mettre d'album, je dois quand même spécifier l'argument en le mettant vide :

Code : VB.NET

```
MonDataSet.Tables("Musiques").Rows.Add("Hotel California", "Eagles",
"", 8)
'Le mot-clé « nothing » (équivalent de NULL) fonctionne aussi
MonDataSet.Tables("Musiques").Rows.Add("Hotel California", "Eagles",
Nothing, 8)
```

Vous voilà donc avec un `DataSet` créé de toutes pièces et sur lequel vous allez pouvoir effectuer les traitements vus précédemment (`DataGridView`, puis tri, etc.).

- Le `DataSet` est un tableau destiné à contenir une base de données.
- Attention à son utilisation, la quantité de mémoire qu'il requiert peut vite devenir excessive.
- C'est l'objet idéal si on sait que les données ne vont pas changer au cours de l'utilisation du logiciel.

L'utilisation graphique : le DataGrid

Vous savez maintenant bien manipuler votre table et récupérer ses données sans trop de problèmes. Le DataSet nous a bien servi et vous savez désormais parfaitement l'utiliser et le manipuler (du moins je l'espère). Cet objet a la particularité de nous faciliter la tâche d'affichage des données qu'il contient. On va toutefois apprendre à connaître un nouvel objet : le DataGrid.

La découverte du DataGrid

Attaquons tout de suite avec le DataGrid (encore un DataQuelquechose... Ils nous envahissent !).

Le DataGrid est un élément graphique qui va nous permettre d'afficher des données récupérées sur une base de données. Son utilisation est très simple et le rendu final est agréable, comme le montre la figure suivante.

	ID	Titre	Artiste	Album	Classement
▶	1	Don't Cry	Guns N' Roses	Album inconnu	
	2	Nothing Else Mat...	Metallica	Metallica	9
	3	Jeune Et Con	Saez	Jours Etranges	
	4	Un Jour En France	Noir Désir	Album inconnu	
	5	Hotel California	Eagles	Album inconnu	
*					

Voici à quoi

ressemble l'affichage d'un DataGrid

Comme vous le voyez, le DataGrid offre de multiples fonctionnalités et outils. Comme pour le reste de ce tutoriel, les connaissances que je vais vous apporter ne sont pas exhaustives, à vous de fouiller dans les propriétés et les **Handles** du DataGrid afin de savoir l'exploiter au mieux.

Ce qu'il faut savoir, c'est que notre DataGrid va avoir la même structure qu'une DataTable ou un DataView, c'est-à-dire qu'il possède des attributs de type Column, Row et Item. Si vous avez compris le fonctionnement de ces objets, vous allez pouvoir interagir de la même manière avec votre DataGrid.

Cependant, pour une plus grande flexibilité, je vais vous apprendre dans ce chapitre à lier votre DataGrid à un objet contenant des données. Cette liaison permettra à notre DataGrid à copier chacune des données pour l'afficher. Ainsi, une fois la liaison effectuée avec notre objet de type DataTable par exemple, on pourra restreindre les modifications à effectuer sur ladite DataTable, puis effectuer une mise à jour de l'objet graphique.

Cette méthode de programmation est spécifique à celle que vous avez pu utiliser précédemment (quand on assignait directement des valeurs à nos contrôles graphiques). Mais elle est presque nécessaire lors du travail sur les bases de données.

Bref, commençons par apprendre à lier des données avec notre DataGrid. Pour cela, il existe deux grandes voies :

- En se servant du code VB .NET pour lier une DataTable ou un DataView au DataGrid ;
- En utilisant l'assistant de liaison d'un DataGrid à une BDD.

Liaison avec le code VB .NET

Nous allons voir les deux en commençant par notre moyen actuel, c'est-à-dire en se servant du code .NET.

Créez donc un nouveau projet, graphique cette fois-ci. Ajoutez la composant DataGrid à votre feuille graphique. Il se trouve dans la rubrique Données. Puis utilisons le code VB que nous avons déjà utilisé pour forger notre DataSet et plaçons-le dans le FormLoad.

Code : VB.NET

```
Dim Connexion As New SqlConnection("Data Source=localhost;Initial
```

```
Catalog=SDZ;User Id=sa;Password=*****;"
```

```
Try
    Connexion.Open()

    Dim Requete As String = "SELECT * from Musiques"
    Dim Commande As New SqlCommand(Requete, Connexion)
    Dim Adaptateur As New SqlDataAdapter(Commande)
    Dim MonDataSet As New DataSet
    Try
        Adaptateur.Fill(MonDataSet, "Musiques")

        'Liaison avec le DataGridView

        Catch ex As Exception
            Console.WriteLine(ex.Message)
    End Try

    Catch ex As Exception
        Console.WriteLine(ex.Message)
    End Try
```

Nous revoilà donc avec notre DataSet contenant les données de la BDD. Pour le lier avec notre DataGridView, nous allons devoir utiliser la propriété **DataSource** de ce dernier.

On peut assigner à **DataSource** une **DataTable** ou un **DataView**, en fonction de ce que l'on souhaite.

Code : VB.NET

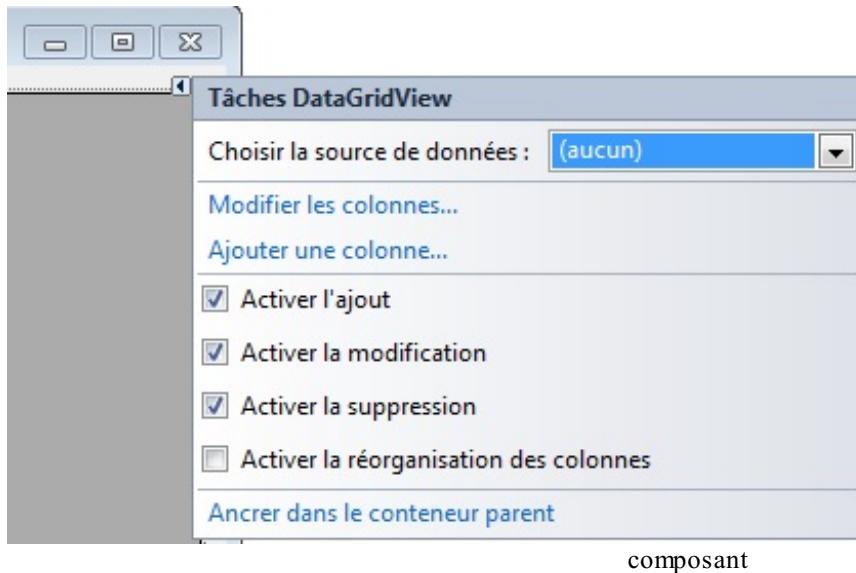
```
DG_DataGrid.DataSource = MonDataSet.Tables("Musiques")
```

J'ai donc lié le DataGridView à ma DataTable, vous pouvez tester le programme et constater qu'il a parfaitement rempli le composant.

Vous pouvez voir qu'en fait ce DataGridView est une « vue » au même titre qu'un **DataView**, vous pouvez effectuer des tris en cliquant sur les colonnes. Il est également possible de modifier les données (les changements ne seront pas effectifs sur la BDD cependant) ou d'en ajouter.

Liaison via l'assistant

La seconde méthode est d'utiliser l'assistant. Créez un nouveau projet et ajoutez un **DataGridView**. Cliquez sur la petite flèche en haut à droite du composant, comme à la figure suivante.

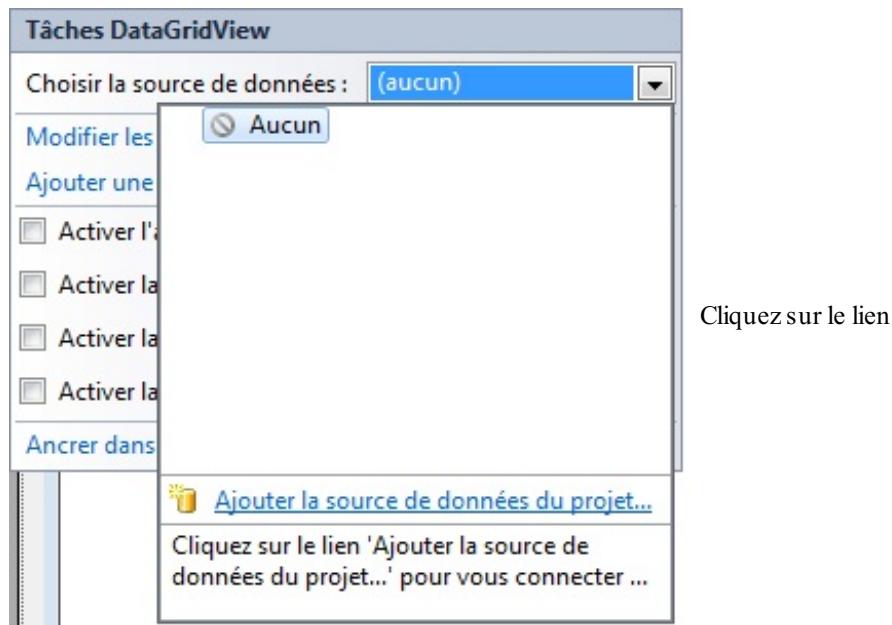


Cliquez sur la petite flèche en haut à droite du

composant

Comme vous le voyez, vous pouvez spécifier à partir de cette fenêtre si le DataGridView aura la possibilité d'ajouter des données, de les modifier, etc. *Décochez tout.*

Puis cliquez sur Choisir la source de données et Ajouter la source de données du projet, comme à la figure suivante.



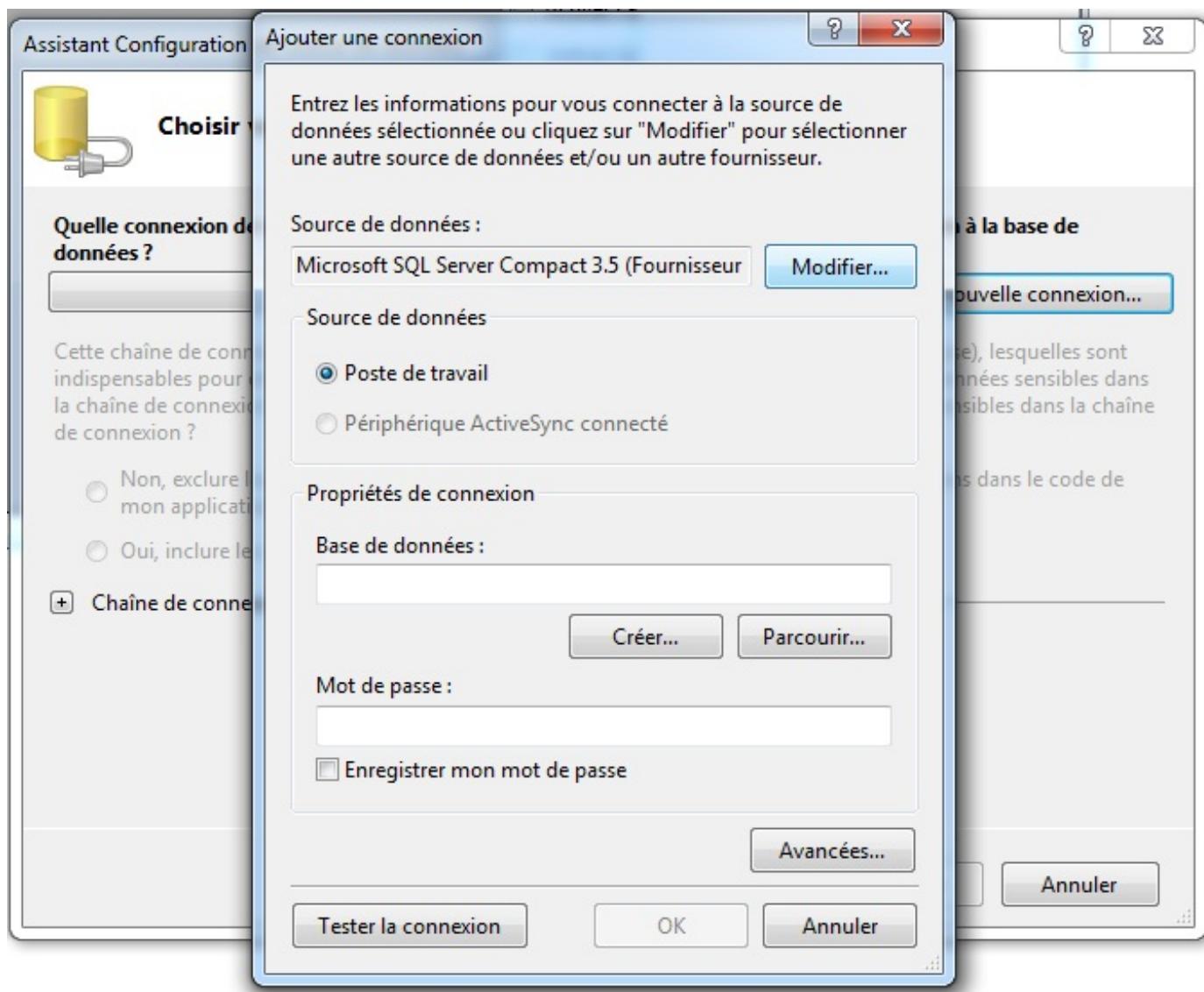
Vous voici devant l'assistant d'ajout d'une base de données au projet. Cela va permettre de lier une base au projet afin de faciliter la liaison d'éléments avec la BDD sans passer par du code VB .NET.

Suivons donc cet assistant. Dans la première fenêtre, spécifiez Base de données, puis dans la seconde, DataSet.

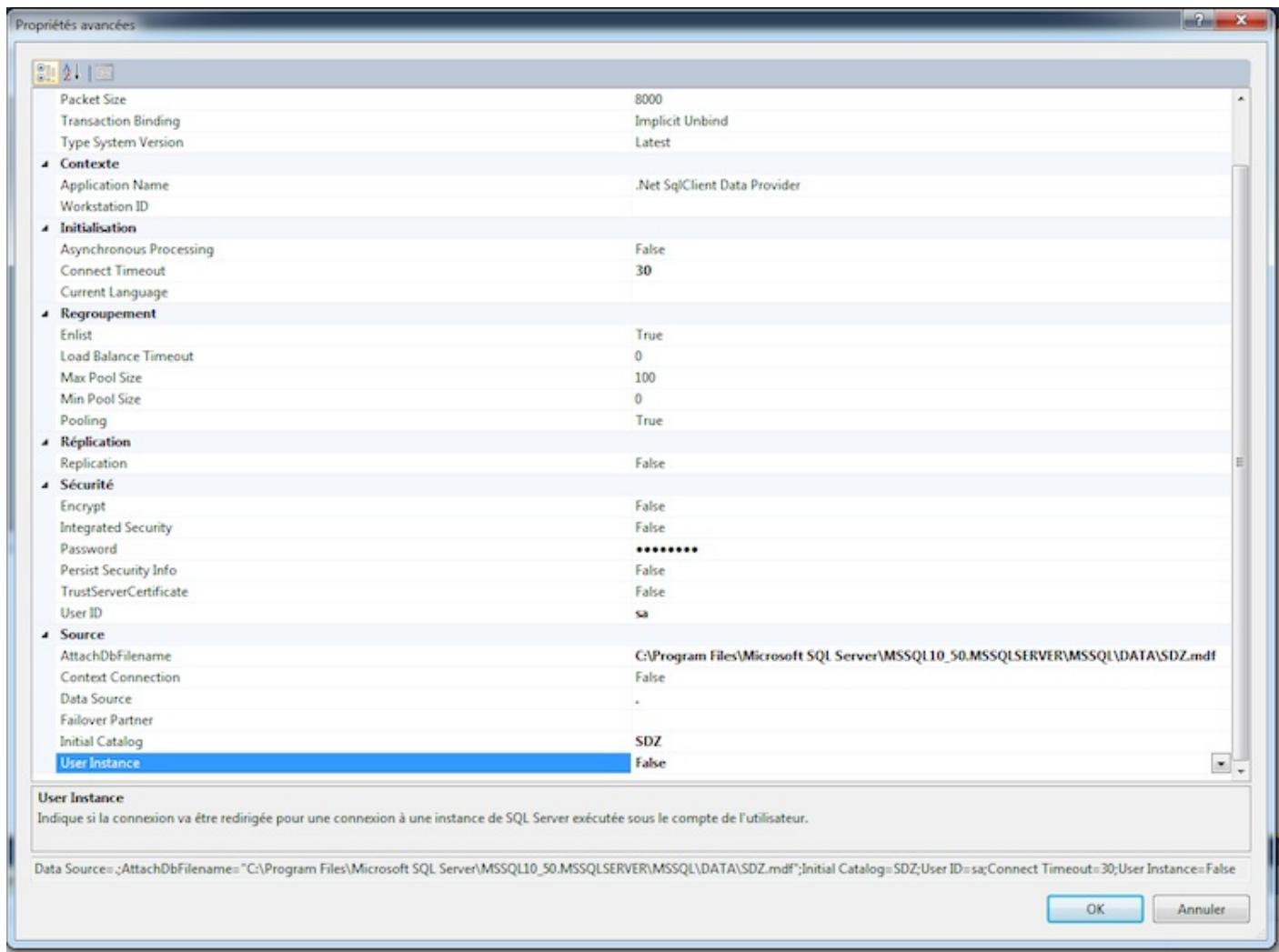


Cela va ajouter un objet de type DataSet à votre projet (mais visible du côté feuille de design graphique).

Maintenant, créons notre connexion vers la BDD. Cliquez sur Nouvelle connexion puis changez la source de données pour utiliser SQL Server (voir figure suivante).



Sélectionnez Avancées et entrez les mêmes valeurs qu'à la figure suivante.



Entrez ces valeurs

Il faut modifier :

- User ID ;
- Password ;
- User Instance ;
- Attach DB Filename et y entrer l'adresse du fichier de BDD :
C:\Program Files\Microsoft SQL Server\MSSQL10_50.MSSQLSERVER\MSSQL\DATA\SDZ.mdf ;
- Initial Catalog ;
- DataSource.

Validez, utilisez la connexion SQL Server et testez la connexion.



Les informations de connexion peuvent différer d'un PC à l'autre, je ne peux pas faire de cas par cas, donc postezi sur le forum si vous rencontrez des difficultés à cette étape.

Puis cliquez sur OK. Vous revoilà à l'ancienne fenêtre, sélectionnez le bouton radio Oui je veux inclure..., cela ajoutera le mot de passe à la chaîne de connexion.

Elle devrait alors ressembler à ça :

Code : VB.NET

```
Data Source=.;AttachDbFilename="C:\Program Files\Microsoft SQL Server\MSSQL10_50.MSSQLSERVER\MSSQL\DATA\SDZ.mdf";Initial Catalog=SDZ;User ID=sa;Password=*****;Connect Timeout=30;User
```

Instance=False

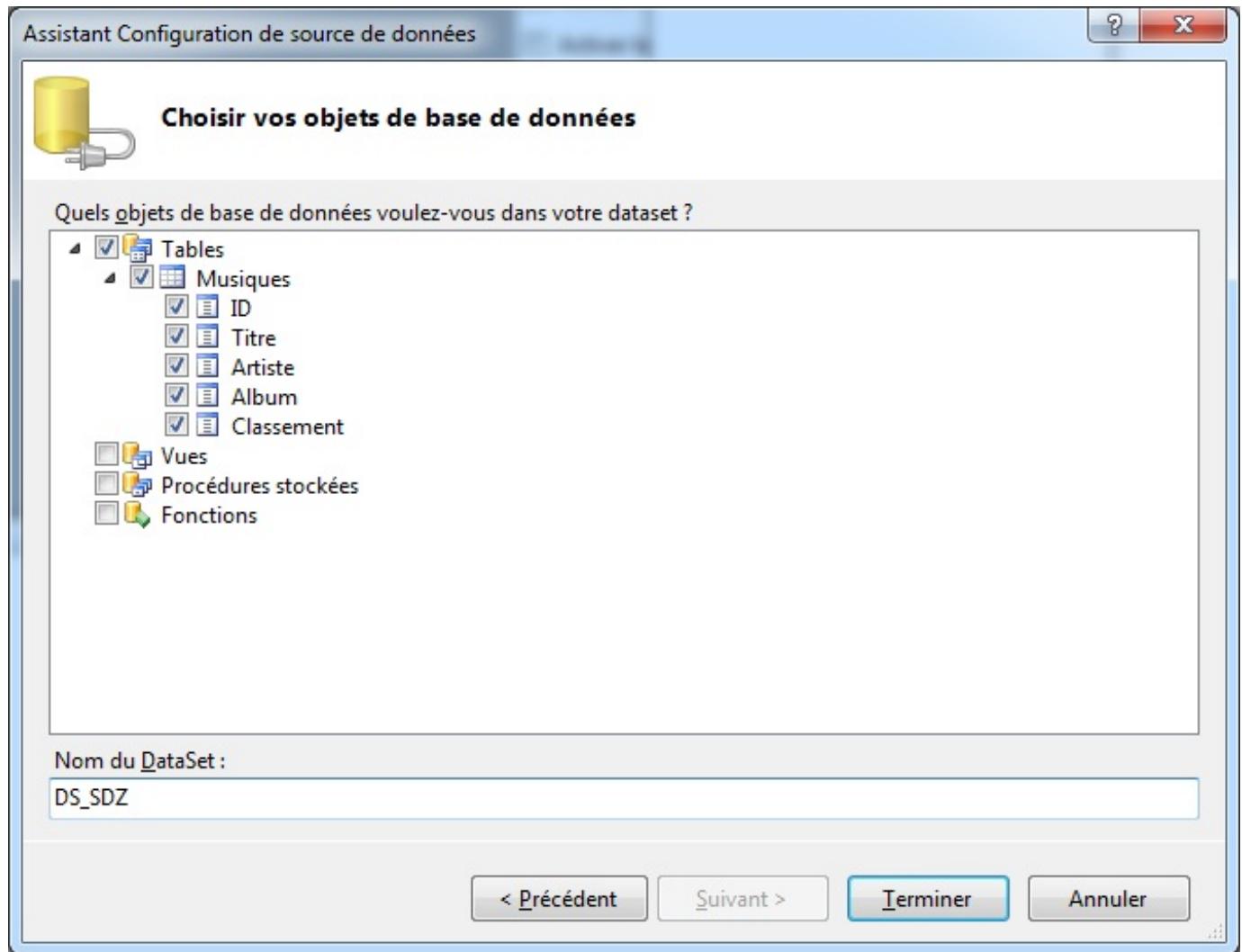
Puis Suivant.

Si le programme vous demande de copier le fichier, répondez Non.

Finalement, ne stockez pas la chaîne dans le fichier de configuration. Lors de plus gros projets, cette méthode est utile, notamment pour le changement fréquent d'identifiants.

Vous voilà désormais dans la création du DataSet.

Spécifiez que vous souhaitez toute la table SDZ et nommez-la (voir figure suivante).



Cochez toutes les cases de la table

En cliquant sur Terminer, la fenêtre visible à la figure suivante s'affiche.

The screenshot shows the Visual Studio IDE interface. At the top, there's a toolbar with various icons. Below the toolbar, a **DataGrid** window is open, displaying a table with columns: ID, Titre, Artiste, Album, and Classement. The grid itself is currently empty, showing a light gray background. To the right of the DataGrid window, the word "DataGrid" is written in bold black text. Below the DataGrid window, there's a toolbar with three items: DS_SDZ, MusiquesBindingSource, and MusiquesTableAdapter. Each item has a small icon to its left.

Vous avez sans doute remarqué les trois objets qui ont été créés suite à cette opération :

- Un **DataSet** ;
- Un **BindingSource** qui lie notre source à notre **DataSet**, car on ne peut pas lier directement un **DataSet** donc cet objet effectue la liaison ;
- Un **TableAdaptater**, il effectue la requête pour récupérer les données dans le **DataSet**. Son fonctionnement est plutôt particulier et compliqué ; pour plus de souplesse je vous recommande d'utiliser du code VB plutôt que ces objets en cas de projets nécessitant des requêtes spécifiques.

Vous voilà en possession du même **DataSet** que précédemment. Si vous lancez le programme, le résultat sera le même.

The screenshot shows the Visual Studio IDE interface again. A **DataGrid** window is open, displaying a table with columns: ID, Titre, Artiste, Album, and Classement. The data grid contains the following rows:

	ID	Titre	Artiste	Album	Classement
▶	1	Don't Cry	Guns N' Roses	Album inconnu	
	2	Nothing Else Mat...	Metallica	Metallica	9
	3	Jeune Et Con	Saez	Jours Etranges	
	4	Un Jour En France	Noir Désir	Album inconnu	
	5	Hotel California	Eagles	Album inconnu	
*					

To the right of the DataGrid window, the text "Le résultat est le" is written in black. Below the DataGrid window, the text "même que précédemment" is written in black.

- Le **DataGridView** permet d'afficher graphiquement des données en provenance d'une table.
- On peut ensuite le programmer pour appliquer les modifications que l'utilisateur effectue.
- L'assistant de VB .NET vous aidera dans l'étape de configuration pour le lier à la BDD.

TP : ZBiblio V2

Mettons en pratique notre travail sur les BDD en faisant un petit TP sur le sujet. Vous vous souvenez de notre TP ZBiblio ? Nous allons tenter d'améliorer notre petit logiciel afin d'intégrer la notion de base de données dans ce dernier.

Je vais avoir besoin de vous en pleine forme pour mener à bien ce petit travail alors allez vous chercher un café et attaquons !

Cahier des charges

Vous vous souvenez de notre TP ZBiblio ? Nous allons tenter d'améliorer notre petit logiciel afin d'intégrer la notion de base de données dans ce dernier.

Dans notre première version de la bibliothèque de films, nous utilisions la sérialisation pour stocker les informations sous forme de fichier. Cette méthode a des avantages (notamment la mise en place très simple sur un PC en local de la méthode de type sérialisation) et des inconvénients.

Une base de données n'est pas vraiment adaptée pour un logiciel individuel fonctionnant uniquement en local sur une seule et unique machine. Cependant, si vous souhaitez partager votre base de données de films afin de créer une bibliothèque collaborative, ce TP va vous enseigner les fondements pour effectuer ces modifications.

Résumons. Le logiciel de bibliothèque ne va plus devoir écrire et lire dans un fichier, mais dans une base de données. Cette BDD peut être locale ou distante. Avoir une BDD locale perd un peu de son intérêt, mais nous allons faire de cette manière pour ce TP. Libre à vous après de modifier l'adresse de votre BDD par une adresse de serveur contenant cette dernière et de diffuser votre logiciel (à votre famille ou vos amis afin de partager la même bibliothèque avec d'autres personnes).



Attention toutefois, diffuser un logiciel contenant une liaison à votre BDD peut être dangereux, des personnes mal intentionnées peuvent utiliser le logiciel pour effectuer des modifications ou des suppressions non souhaitées.

Dans ce TP, nous n'allons pas implémenter la notion de droits, ainsi, chaque utilisateur du logiciel pourra :

- Consulter une fiche de film ;
- Créer une fiche de film ;
- Modifier ou supprimer une fiche de film.

La structure du logiciel restera la même, les enregistrements seront justes différents.

Qui dit base de données dit conception de cette dernière. À vous de créer la table avec les champs requis pour stocker toutes les informations de notre fiche de films.

Deux particularités à prendre en compte lors de l'insertion de données en BDD :

- Pour écrire une date dans un champ `DATE` de SQL, vous devez formater cette dernière en utilisant :
`ToString("yyyy-MM-dd")`. Cette fonction formatera la date pour qu'elle soit insérée sans problèmes en BDD.
- Dans des chaînes de caractères, il peut y avoir des apostrophes : « ' », cependant, ce caractère est utilisé dans des requêtes SQL pour délimiter le début et la fin de chaînes de caractères. Donc si une apostrophe est au milieu de votre mot, il sera coupé, et provoquera une erreur de surcroît. Pour éviter cette erreur, utilisez une fonction pour « doubler » les apostrophes de vos chaînes de caractères : « ' » => « " ».

Je recommande à tous ceux qui ne se sentent pas sûrs d'eux pour le premier TP de réutiliser le projet fonctionnel que j'ai fait en [le téléchargeant](#).

Je ne vous en dit pas plus, à vos claviers les Zéros ! Et bonne chance.

Correction : partie BDD

Passons à la correction.

Tout d'abord, réfléchissons à la structure de données de notre table.

Un champ de type `ID` sera utilisé pour référencer chaque film et pouvoir modifier ou supprimer rapidement un film. Ainsi, pour le supprimer, je n'aurai qu'à effectuer un `DELETE... WHERE ID = X`.

Ce champ `ID` sera une clé primaire et auto-incrémentale.

Ces deux mots barbares sont juste là pour dire que :

- Ce champ sera unique ;
- Il sera incrémenté automatiquement de 1 à chaque enregistrement.

Ainsi, nous n'avons pas besoin de nous en soucier, il sera créé tout seul et à la bonne valeur quand nous ajouterons un film.

Les autres champs ne devraient pas poser de problème, ce sont tous des chaînes de caractères. J'ai utilisé le type nvarchar avec des valeurs maximum cohérentes.

L'avis personnel et le synopsis sont des types `Text`. Ce type ne limitant théoriquement pas le nombre de caractères possibles dans le champ.

La date de sortie est un type `Date` et la note personnelle un `int`.

Je vous donne ma requête de création de table pour ceux qui auraient déjà été bloqués à cette partie :

Code : SQL

```
USE [SDZ]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[ZBiblio](
    [ID] [int] IDENTITY(1,1) NOT NULL,
    [NOM] [nvarchar](50) NOT NULL,
    [DATE_SORTIE] [date] NULL,
    [REALISATEUR] [nvarchar](50) NULL,
    [GENRE1] [nvarchar](50) NULL,
    [GENRE2] [nvarchar](50) NULL,
    [ACTEURS] [nvarchar](200) NULL,
    [SYNOPSIS] [text] NULL,
    [AVIS_PERSONNEL] [text] NULL,
    [NOTE_PERSONNELLE] [int] NULL,
    [AFFICHE] [nvarchar](255) NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

GO
```

Correction : partie VB

Concernant la partie VB, il n'y a pas énormément de changements, toute la structure est la même.

Il faut cependant intégrer toutes nos requêtes vers la base de données où on accédait précédemment à la classe Film.

Connexion

Commençons avec la chaîne de connexion :

Code : VB.NET

```
Public Connexion As New SqlConnection("Data Source=localhost;Initial Catalog=SDZ;User Id=sa;Password=*****;")
```

Je l'ai mise en champ `public` de la class `ZBiblio` pour pouvoir y accéder depuis la fenêtre de création et modification.

Dans le chargement du programme, intégrons la connexion à la base de données :

Code : VB.NET

```
'Tentative de connexion à la BDD
Try
    Connexion.Open()
```

```

Catch ex As Exception
    MessageBox.Show("Erreur lors de la connexion à la BDD
... Vérifiez votre connexion ou contactez votre administrateur..",
"ERREUR", MessageBoxButtons.OK, MessageBoxIcon.Error)
    Me.Close()
End Try

```

Récupération

Ma méthode UpdateListe aura pour but de récupérer les données en BDD et de mettre à jour la liste.
Ainsi :

Code : VB.NET

```

Public Sub UpdateListe()
    RecuperationListeFilms() 'On récupère les informations en
    BDD
    'On vide la liste et on la reremplit
    Me.LB_LISTE_FILMS.Items.Clear()
    'Parcourt les films de la bibliothèque
    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

```

Avec la méthode RecuperationListeFilms :

Code : VB.NET

```

Private Sub RecuperationListeFilms()
    _ListeFilms.Clear() 'On nettoie la liste

    'Recupère la liste en BDD
    Dim Requete As String = "SELECT * from ZBiblio" ' Récupère
    tous les films de la table

    Try
        Dim Commande As New SqlCommand(Requete, Connexion)
        Dim MonReader As SqlDataReader =
        Commande.ExecuteReader()
        While MonReader.Read() 'Lit chaque film en BDD et crée
        un objet Film avec les informations
            _ListeFilms.Add(New Film(MonReader("ID").ToString,
            MonReader("NOM").ToString, CDate(MonReader("DATE_SORTIE").ToString),
            MonReader("REALISATEUR").ToString, MonReader("GENRE1").ToString,
            MonReader("GENRE2").ToString, MonReader("ACTEURS").ToString,
            MonReader("SYNOPSIS").ToString,
            MonReader("AVIS_PERSONNEL").ToString,
            MonReader("NOTE_PERSONNELLE").ToString))
        End While
        Commande.Dispose()
        MonReader.Close()
    Catch ex As Exception
        MessageBox.Show("Erreur lors de la récupération des
        données EN BDD ... Contactez votre administrateur.", "ERREUR",
        MessageBoxButtons.OK, MessageBoxIcon.Error)
        Me.Close()
    End Try

End Sub

```



Je rappelle qu'il est très important lorsque l'on travaille avec des bases de données d'intégrer des blocs **Try... Catch** à chaque accès. Je vous renvoie vers l'annexe correspondante si vous ne savez pas les utiliser. Une déconnexion, une surcharge des requêtes, etc. sont vite arrivées, il faut pouvoir indiquer à l'utilisateur qu'il y a eu un problème sans crasher l'application.

Arrivés à ce stade, nous avons nos données récupérées et affichées dans la liste.

Passons maintenant aux phases d'ajout, de modification et du suppression

Suppression d'une fiche

La phase de suppression s'effectue toujours lors du clic sur Supprimer la fiche du film. Sauf qu'il faut désormais intégrer la notion de requête à la base de données, puis terminer par une mise à jour de la liste.

Ce qui donne :

Code : VB.NET

```

Private Sub BT_SUPPRIMER_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles BT_SUPPRIMER.Click
    If Not _FilmEnVisualisation Is Nothing Then 'Si un film est
    sélectionné
        'Confirmation
        If MsgBox("Etes vous certain de vouloir supprimer ce
        film ?", vbYesNo, "Confirmation") Then
            'On le supprime en BDD
            Dim Requete As String = "DELETE FROM ZBiblio WHERE
            ID=" & _FilmEnVisualisation.ID
            Dim Commande As New SqlCommand(Requete, Connexion)
            Try
                Commande.ExecuteNonQuery()
            Catch ex As Exception
                MessageBox.Show("Erreur lors de la suppression
                des données EN BDD ... Contactez votre administrateur.", "ERREUR",
                MessageBoxButtons.OK, MessageBoxIcon.Error)
            End Try
            Commande.Dispose()
        End If

        'MAJ
        UpdateListe()
    Else
        MsgBox("Selectionnez d'abord un film", vbOK, "Erreur")
    End If
End Sub

```

L'ID nous est utile ici pour accéder sans ambiguïté au film que nous souhaitons supprimer (je rappelle que l'ID est unique).

Modification et ajout

Ces deux opérations seront effectuées dans la fenêtre Fiche film.

En effet, souvenez-vous de la correction de l'ancien TP, si une fiche de film était passée en paramètre lors de l'ouverture de cette fenêtre, cela signifiait que l'utilisateur souhaitait modifier la fiche, sinon cela indiquait qu'il fallait en créer une nouvelle.

Nous allons donc simplement intégrer nos requêtes SQL à ces emplacements.

Puisque nous avons créé la variable Connexion en **public**, nous pouvons y accéder depuis cette fenêtre, et donc ne pas la réécrire :



C'est ici qu'il faut bien veiller à formater la date et la chaîne de caractères pour qu'elles soient acceptées par la base de données.

Code : VB.NET

```

Private Sub BT_SAVE_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BT_SAVE.Click
    Dim DATE_SORTIE As String =
    Me.DT_DATE_SORTIE.ToString("yyyy-MM-dd") 'Formate la date pour la
    BDD
    If _FilmAModifier Is Nothing Then
        'Enregistre notre film en BDD
        Dim Requete As String = "Insert into ZBiblio(NOM,
DATE_SORTIE, REALISATEUR, GENRE1, GENRE2, ACTEURS, SYNOPSIS,
AVIS_PERSONNEL, NOTE_PERSONNELLE) values ('" &
FormatString(Me.TXT_NOM.Text) & "','" & DATE_SORTIE & "','" &
FormatString(Me.DDL_REALISATEUR.Text) & "','" &
FormatString(Me.DDL_GENRE1.Text) & "','" &
FormatString(Me.DDL_GENRE2.Text) & "','" &
FormatString(Me.TXT_ACTEURS.Text) & "','" &
FormatString(Me.TXT_SYNOPSIS.Text) & "','" &
FormatString(Me.TXT_AVIS_PERSONNEL.Text) & "','" &
Me.NUM_NOTE_PERSO.Value & ")"
        Dim Commande As New SqlCommand(Requete,
ZBiblio.Connexion)
        Try
            Commande.ExecuteNonQuery()
            MsgBox("Fiche correctement créée", vbOKOnly,
"Confirmation")
        Catch ex As Exception
            MessageBox.Show(ex.ToString & "Erreur lors de
l'ajout d'un film EN BDD ... Contactez votre administrateur.",
"ERREUR", MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try

    Else
        'Sinon on le modifie en récupérant son index dans la
        liste de la fenêtre parent
        Dim Requete As String = "UPDATE ZBiblio SET NOM='"
        FormatString(Me.TXT_NOM.Text) & "', DATE_SORTIE='"
        FormatString(Me.DDL_REALISATEUR.Text) & "', REALISATEUR='"
        FormatString(Me.DDL_GENRE1.Text) & "', GENRE1='"
        FormatString(Me.DDL_GENRE2.Text) & "', ACTEURS='"
        FormatString(Me.TXT_ACTEURS.Text) & "', SYNOPSIS='"
        FormatString(Me.TXT_SYNOPSIS.Text) & "', AVIS_PERSONNEL='"
        FormatString(Me.TXT_AVIS_PERSONNEL.Text) & "', NOTE_PERSONNELLE='"
        Me.NUM_NOTE_PERSO.Value & " WHERE ID=" & FilmAModifier.ID
        Dim Commande As New SqlCommand(Requete,
ZBiblio.Connexion)
        Try
            Commande.ExecuteNonQuery()
            MsgBox("Fiche correctement modifiée", vbOKOnly,
"Confirmation")
        Catch ex As Exception
            MessageBox.Show("Erreur lors de la modification d'un
film EN BDD ... Contactez votre administrateur.", "ERREUR",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End If

    'MAJ de la liste dans la fenêtre parent
    ZBiblio.UpdateListe()
    'Ferme la fenêtre d'édition
    Me.Close()
End Sub

```

Vous remarquez que je me sers de la fonction `FormatString`. Cette fonction est relativement simple :

Code : VB.NET

```
Private Function FormatString(ByVal val As String) As String
    Return val.Replace("""", """")
End Function
```

Tout est là. Après, si vous avez suivi le tuto jusqu'ici, je pense que vous êtes aptes à modifier et adapter le code en fonction de vos besoins.

Conclusion

Pour conclure, je ne vous le rappellerai jamais assez, la programmation c'est avant tout de la recherche, de l'expérimentation et des erreurs...

À vous d'adapter ce code suivant vos besoins, il est même possible que votre bibliothèque fonctionne totalement différemment de celle présentée lors de ce TP, chaque développeur structure son programme à sa manière.

Il n'y a pas de bonne ou de mauvaises réponses, surtout pas en VB ! 😊

Améliorations

Quelques pistes de recherche pour vous, amis Zeros :

- Implémenter toute la structure pour les affiches, le stockage en BDD, l'ajout, etc. Vous pouvez utiliser la structure de champ `Image` en base de données ou alors stocker son chemin (URL vers le .jpg par exemple...).
 - Améliorer un peu le design. 😊
 - Créer un *scrapper*, un module qui irait chercher les informations de chaque film sur Allociné pour remplir la fiche.
 - Etc.
-
- Voilà la mise à jour du logiciel ZBiblio pour utiliser la base de données.
 - Gardez ce TP pour avoir les exemples de création SQL d'une BDD.
 - Il vous sera également utile pour la récupération et l'écriture simple à une BDD.

Partie 5 : La communication par le réseau

Vous avez toutes les notions requises pour mener à bien des projets simples en VB .NET.

Certains d'entre vous, pourtant, souhaitent créer des applications communiquant entre elles, basées sur un système client/serveur. Cette partie vous apportera toutes les bases pour ces types de programmes.

Introduction à la communication

L'informatique actuelle n'est que communication. Il y a de la communication entre les composants de notre ordinateur (carte-mère => processeur...) jusqu'à l'immense réseau de communication qu'est internet. La communication peut être interne à un PC ou via le réseau.

La communication, pourquoi ?

Communication interne

Ce que je qualifie de « communication interne » désigne une communication sur une seule et même machine (un seul PC). Exemple, vous avez développé au cours de ce tutoriel la bibliothèque de films ZBiblio et le navigateur web. Pourquoi ne pas ajouter un bouton qui permet, lorsque vous êtes sur la fiche d'un film sur Allociné, d'ajouter automatiquement ce film à votre bibliothèque ?



Oui, je sais, suite à notre dernière modification de liaison à une base de données, cette opération va être relativement simple à réaliser et sans avoir besoin de communication directe entre le navigateur et la bibliothèque (le navigateur ajoute en base de données et la bibliothèque lit en base de données).

Cependant, avec notre ancienne version de ZBiblio, il aurait fallu implémenter une communication afin de transmettre des données du navigateur vers la bibliothèque. Et cela sur votre seul PC.

Il s'agit donc de communication interne.

Vous admettrez que ce n'est pas réellement intéressant, surtout à notre époque.
La chose réellement intéressante est alors la communication réseau !

Communication réseau

Vous avez de la communication par votre réseau dès que vous allez sur internet, votre navigateur (le client) va communiquer avec un serveur avec lequel vous allez échanger des données (une requête pour voir les photos de vos amis sur Facebook va être transmise au serveur et ce dernier vous répondra en vous envoyant lesdites photos).

Ainsi est la communication réseau : un ou des clients effectuent des transferts de données avec un serveur.



Quand je parle de communication client/serveur, n'imaginez pas forcément un PC destiné à être client et un PC destiné à être serveur. Un serveur n'est pas forcément un PC spécifique avec un logiciel spécifique. Vous pouvez parfaitement faire tourner le logiciel client et le logiciel serveur sur le même PC. Vous pouvez même incorporer le client et le serveur dans le même logiciel pour permettre une communication *peer to peer* (client vers client).

Les sockets

Les sockets ne sont pas une abréviation de chaussettes. C'est le nom barbare d'un objet destiné à permettre une communication inter-processus.

La communication peut s'effectuer en utilisant le protocole UDP ou TCP. Je vais vous les exposer tout de suite.

TCP : mode connecté

Le *Transmission Control Protocol* (TCP, en français « protocole de contrôle de transmissions ») est un protocole de communication utilisant les IP. Le mode « connecté » signifie que l'établissement de la connexion est requise avant de pouvoir transmettre des données. Ce protocole utilise un port pour communiquer.



Les ports sont comme les portes de votre ordinateur vers le monde extérieur, les logiciels souhaitant communiquer avec l'extérieur doivent utiliser ces portes. Cependant, deux portes ne peuvent être utilisées par deux processus différents. Il faut donc veiller à utiliser un port libre pour effectuer votre communication. Par exemple, le protocole HTTP vous permettant de naviguer sur internet utilise le port 80, HTTPS utilise lui le port 443. La liste des ports déjà utilisés par des

logiciels ou protocoles connus sont référencés sur ce site : [liste des ports](#).

L'avantage est la fiabilité des données. Une fois la connexion établie, ce protocole offre une certitude dans l'acheminement des données.

UDP : mode déconnecté

Le *User Datagram Protocol* (UDP, en français « protocole de datagramme utilisateur ») est un protocole basé sur le mode déconnecté. Cela signifie que l'on peut s'affranchir de toute la procédure de connexion requise par le protocole TCP. Le grand avantage de ce protocole est la rapidité pour transmettre de petites quantités de données. Il est fortement employé dans les jeux vidéos où les données à transmettre entre le client et le serveur sont peu volumineuses mais doivent être acheminées très rapidement.



Et les sockets dans tout ça ?

J'y viens, les sockets vont nous permettre de grandement simplifier les opérations de communication. On s'en servira notamment pour le protocole TCP. Comme je viens de l'expliquer, en TCP une connexion est requise. Les sockets vont nous permettre d'effectuer rapidement les procédures de demande de connexion, d'acceptation de la connexion, etc.

Les utiliser est un très bon compromis entre temps de développement et fiabilité de la connexion.

.NET remoting

Je vais passer très rapidement sur cette méthode de communication. Elle était très utilisée lors des premières années d'utilisation de la technologie .NET et est désormais dépassée.

Le concept innovateur de .NET remoting à l'époque venait du fait que, une fois la connexion d'une application client et serveur effectuée, les deux programmes pouvaient utiliser des objets « partagés » entre les deux applications. C'est un peu comme si une sérialisation de la classe était effectuée à tout moment et était synchronisée avec l'autre application (pour faire simple).

Contrairement aux sockets, cette méthode de communication imposait aux programmes d'utiliser la technologie .NET s'ils souhaitaient communiquer. Si vous utilisez les sockets, vous pouvez avoir une application cliente utilisant VB .NET et une application serveur codée en C par exemple (ou inversement). La seule contrainte est de faire correspondre le type de socket, le port et le protocole sur les deux programmes. Mais nous verrons cela en temps voulu.

Cette API (interface de programmation) a désormais été supplantée par WCF, que je vais vous présenter tout de suite. Je ne ferai pas de chapitre sur .Net remoting, mais libre à vous d'effectuer vos propres recherches si vous souhaitez l'utiliser. 😊

WCF, Windows Communication Fundation

Depuis le framework .NET v3.0 existe WCF, qui est censé remplacer .NET remoting. La grande force du WCF est la flexibilité de la configuration. En effet, vous pouvez développer votre méthode de communication basée sur un protocole HTTP, puis changer pour utiliser un autre port via TCP, tout cela s'effectue de manière transparente pour le développeur. Cela peut être très utile si vous vous rendez compte que sur un réseau public le port utilisé par votre application est bloqué.



Je parle de manière « transparente » pour le développeur, car un fichier de configuration de type XML va contenir ces informations. Il suffira à l'utilisateur de le modifier pour changer ces informations.



Sinon, WCF, ça fonctionne comment ?

WCF va utiliser des messages SOAP (ancien acronyme de *Simple Object Access Protocol*). Ces messages SOAP sont en fait des messages basés sur la structure XML (comme pour notre sérialisation). Ces messages vont être échangés de manière totalement transparente pour nous, utilisant les protocoles et autres niveaux de sécurité requis.

Contrairement aux sockets qui sont considérés comme de la programmation « bas niveau », c'est-à-dire très près du matériel, WCF va être de « haut niveau ».

Bref, tout ça semble plutôt beau, mais nous n'allons pas aborder cette méthode de communication dans ce cours. Tout d'abord parce que sa mise en œuvre n'est pas possible avec la version Express de Visual Basic. Et ensuite parce que je trouve que c'est très très complexe à mettre en œuvre et que vous n'en aurez pas l'utilité dans l'immédiat.

Pour résumer, nous allons donc uniquement nous concentrer sur les sockets. La communication est une très vaste partie, et les sockets vous apporteront le minimum vital pour faire communiquer des applications en toute simplicité.

Je rappelle que seuls les sockets seront étudiés dans ce cours, si vous avez des besoins spécifiques, tournez-vous vers le WCF.

Mais attention, notre version de Visual Studio ne suffit pas pour ce type de développement.

- Les sockets vont nous permettre d'effectuer une communication réseau.
- Les autres technologies Microsoft pour la communication sont .NET remoting et WCF.

Communication via sockets

Commençons dès maintenant le développement des sockets. Nous allons voir différents types de sockets ; dans cette partie, ce seront des sockets de liaison directe entre client et serveur. Deux programmes destinés à communiquer, donc.

Client/serveur

Nous allons maintenant entrer dans le vif du sujet ! Attaquons-nous au développement de nos applications capables de communiquer.

Vous l'aurez sûrement compris, deux êtres humains sont nécessaires pour une communication humaine. Deux programmes seront donc nécessaires pour une communication informatique. 

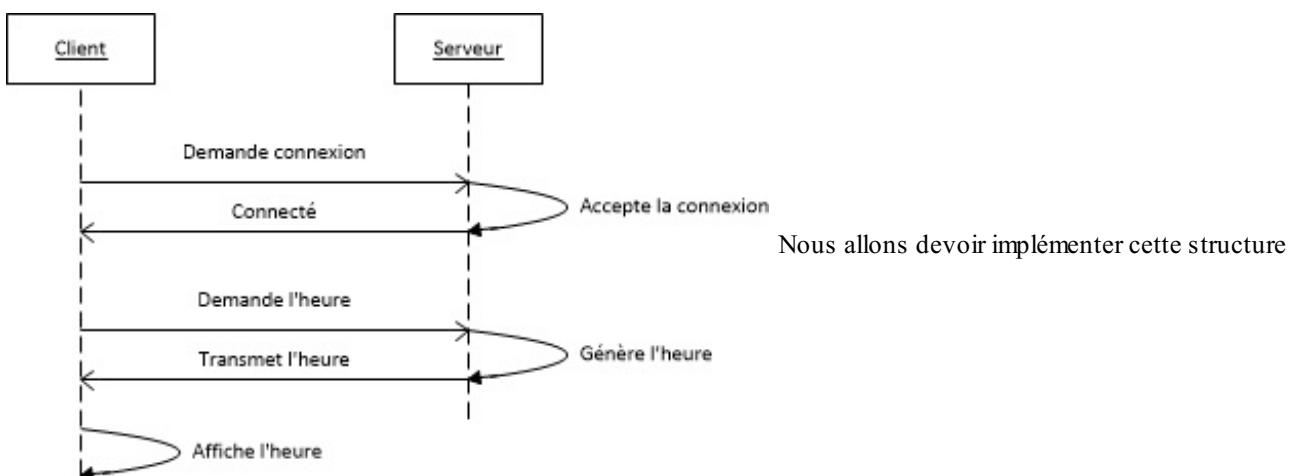
Nous allons appliquer ici le concept de client/serveur. Pour apprendre les sockets, nous allons écrire un client demandant l'heure à un serveur. Ce dernier lui donnera.



Exactement comme si vous demandez l'heure à une personne et que cette dernière vous la donne. Souvenez-vous de cette analogie.

Nous avons donc deux programmes à créer, deux projets. Je nommerai le client `SocketClient` et le serveur `SocketServeur`.

Dans ce chapitre, nous devons donc travailler conjointement sur les deux projets pour implémenter la structure visible à la figure suivante et expliquée dans la liste à puces juste après.



- 1. Le serveur démarre et se met en attente de connexion.
- 2. Le client démarre et demande une connexion au serveur.
- 3. Le serveur voit la connexion client et l'accepte.
- 4. Le client se met en attente de réception des données.
- 5. Le serveur envoie l'heure actuelle.
- 6. Le client reçoit l'heure et l'affiche.

Voilà le concept de base de nos deux programmes. Commençons dès maintenant la phase d'écriture du code.

La connexion

Commençons par le commencement : la connexion.

En sockets, pour se connecter, il faut que le serveur soit en « attente de connexion », et les clients viennent effectuer leurs demandes pendant cette période.

Il existe deux façons pour cette étape : synchrone ou asynchrone.

Synchrone signifie que l'étape d'attente de connexion (par le serveur) et l'étape de demande de connexion (par le client) sont toutes les deux **bloquantes**. Bloquante indique que la fonction bloquera le programme tant que la connexion ne sera pas effectuée. Nous allons travailler en synchrone pour cette partie, cela va nous permettre de bien comprendre à quelle étape du traitement notre programme se situe.

Asynchrone signifie que ces deux opérations ne sont pas bloquantes et seront passées s'il y a connexion ou non. Cela peut être bien si vous souhaitez effectuer de multiples connexions de clients : lorsqu'un client est connecté, il est envoyé dans un processus séparé et sera traité indépendamment. Si nous sommes en synchrone, un seul client peut être traité à la fois (sauf si nous créons un nouveau thread).

Voyons tout de suite la connexion synchrone.

L'IP et le port

Pour toute connexion, il faut connaître l'IP et le port que nous allons utiliser. Le port est comme une porte. Il faut que nos deux programmes soient sur le même pour pouvoir communiquer. L'IP est comme un numéro de téléphone, le client doit entrer celle du serveur pour s'y connecter.



Puisque nous effectuons les tests sur un seul PC (client et serveur sur le même PC), le client aura comme IP à laquelle se connecter 127.0.0.1. Cette IP est locale à votre PC. Autrement dit, c'est comme si vous vous téléphoniez vous-mêmes.

Nous devons donc commencer à définir notre IP et notre port. Personnellement j'utilise le port 8080, si une erreur s'affiche, c'est peut-être que ce port est déjà utilisé, essayez d'en changer.



Je rappelle que le client et le serveur doivent utiliser le même port !

Code : VB.NET

```
Dim port As String = "8080"  
Dim ip As String = "127.0.0.1"
```

Le serveur

On commence par importer le namespace contenant les sockets :

Code : VB.NET

```
Imports System.Net.Sockets  
Imports System.Net
```

Puis on crée notre socket serveur :

Code : VB.NET

```
Dim MonSocketServeur As New Socket(AddressFamily.InterNetwork,  
SocketType.Stream, ProtocolType.Tcp)
```

Il y a trois arguments :

- La famille d'adresse. Nous utiliserons InterNetwork dans ce cours, libre à vous d'examiner la documentation si vous souhaitez en utiliser une plus spécifique.
- Le type de transfert de données. Ici Stream signifie « flux », cela assure un transfert fiable dans les deux sens. Il existe d'autres types comme Raw ou DGram.
- Le protocole, ici TCP, mais il peut être UDP comme je l'ai expliqué dans le dernier chapitre.

Nous avons instancié notre socket serveur. Maintenant, nous devons le configurer pour lui dire quel port utiliser.

Commençons par créer un objet IPEndPoint.

Code : VB.NET

```
Dim MonEP As IPPEndPoint = New IPPEndPoint(IPAddress.Parse(ip), port)
```

Vous remarquez que j'ai effectué un `IPAddress.Parse(ip)`, car le constructeur d'`IPPEndPoint` demande un objet de type `IPAddress` et non pas `String`.

Pour appliquer la configuration d'IP au socket :

Code : VB.NET

```
MonSocketServeur.Bind(MonEP) 'Lie le socket à cette IP
```

Vous l'avez sans doute remarqué, on a lié le serveur à l'adresse IP locale du PC, autrement dit, il va écouter uniquement les connexions provenant de ce PC. Cela est suffisant pour nos tests qui s'effectuent sur le même PC et pour une communication interne. Cependant, si vous voulez ouvrir l'écoute à l'extérieur, vous pouvez le faire en remplaçant l'`IP` par `IPAddress.Any`.

Attention cependant, cela va ouvrir l'écoute à toutes les connexions extérieures, cela peut poser des problèmes de sécurité, surtout si vous n'avez pas mis en œuvre des procédures de sécurité lors de la connexion.

Ensuite, on se met en attente de connexion :

Code : VB.NET

```
MonSocketServeur.Listen(1) 'Se met en mode écoute
```

Le paramètre que j'ai spécifié ici à 1 est la taille de la file d'attente. Ici je sais que j'aurai seulement un client à la fois qui viendra demander l'heure. Si vous souhaitez accepter plus de clients dans la file d'attente, augmentez ce nombre.

Finalement, pour que le serveur soit pleinement apte à recevoir les connexions entrantes et les accepter, il faut se mettre en mode « acceptation ».

Code : VB.NET

```
Dim SocketEnvoi As Socket = MonSocketServeur.Accept() 'Bloquant tant que pas de connexion
```

Nous sommes en communication synchrone, donc la fonction `Accept()` sera bloquante, le programme n'ira pas plus loin tant qu'un socket client ne s'y sera pas connecté.

Vous remarquez que cette fonction nous retourne un socket. Ce socket va être très important, il va nous servir à communiquer. Je vais essayer d'expliquer cela en termes simples.

Le socket serveur est juste là pour accepter les connexions, c'est comme un réceptionniste dans un restaurant : il est à l'accueil et voit des gens entrer. Le socket client, lui, est un client qui entre dans ce restaurant. Le réceptionniste va donc lui attribuer une table et, pour ne pas le laisser seul, va lui attribuer un serveur pour le servir. C'est ce nouveau serveur qui est généré qui va servir de passerelle entre le réceptionniste et le client. Pour communiquer avec le client, le réceptionniste va communiquer avec le serveur et ce dernier communiquera alors avec le client.

Ce socket créé est donc la passerelle par laquelle les messages en provenance ou à destination du client seront envoyés.

Le client

Il faut aussi créer notre socket dans notre programme client :

Code : VB.NET

```
Dim MonSocketClient As New Socket(AddressFamily.InterNetwork,  
    SocketType.Stream, ProtocolType.Tcp)
```



Les paramètres du socket doivent correspondre avec ceux spécifiés pour le socket serveur.

Code : VB.NET

```
Dim MonEP As IPEndPoint = New IPEndPoint(IPAddress.Parse(ip), port)  
MonSocketClient.Connect(MonEP)
```

Comme pour le serveur, on doit spécifier l'`IPEndPoint`, suivi de la méthode `Connect` prenant cet objet en argument.



Je vous conseille d'entourer le `Connect` de blocs `Try... Catch`, si pour une raison X ou Y votre serveur n'est pas accessible, cela ne doit pas faire totalement planter votre programme.

Voilà, le client et le serveur doivent à cette étape avoir établi une connexion. N'oubliez pas de fermer la connexion client avec la méthode `Close()` afin de libérer la connexion lorsque tout est terminé.



N'oubliez pas que les IP que nous avons utilisées ici sont uniquement pour ce cas d'utilisation (les deux programmes sur le même PC). Si vous souhaitez un programme qui communique avec un autre sur deux PC différents, vous devrez remplacer les IP 127.0.0.1 par celles qu'auront les PC qui vont communiquer. Attention toutefois, des IP dans un réseau local peuvent être dynamiques (changer au redémarrage du PC) et si vous souhaitez utiliser ce programme au travers de l'internet, cela amène encore plus de difficultés (redirection de port, etc.) que je ne développerai pas ici. Je vous conseille donc de rester dans le cadre du réseau local et si cela ne fonctionne pas, vérifiez vos pare-feux.

Maintenant, le transfert de données !

Le transfert de données

Une fois notre socket connecté, on peut passer à l'étape qui doit vous intéresser : le transfert de données.

Lorsque nous communiquons par sockets, les données transférées seront de type `Byte`, plus précisément un tableau de `Byte`.

Petit rappel sur les types. Le type `Byte` peut contenir des entiers de 0 à 255 (voir figure suivante). On peut donc l'utiliser pour stocker des caractères au format ASCII (les caractères sont codés sur 8 bits, ce qui correspond à 256 valeurs).

0	24	↑	48	0	72	H	96	`	120	x	144	É	168	¿	192	L	216	½	240	≡
1	25	↓	49	1	73	I	97	a	121	y	145	æ	169	Γ	193	└	217	½	241	±
2	26	→	50	2	74	J	98	b	122	z	146	Œ	170	¬	194	─	218	▀	242	≥
3	27	←	51	3	75	K	99	c	123	{	147	ô	171	½	195	─	219	█	243	≤
4	28	↳	52	4	76	L	100	d	124		148	ö	172	¼	196	—	220	█	244	∫
5	29	⊕	53	5	77	M	101	e	125)	149	ð	173	ı	197	+	221	█	245	ʃ
6	30	▲	54	6	78	N	102	f	126	~	150	û	174	«	198	F	222	█	246	÷
7	31	▼	55	7	79	O	103	g	127	△	151	Ù	175	»	199	█	223	█	247	≈
8	32	■	56	8	80	P	104	h	128	ç	152	ÿ	176	▀	200	█	224	α	248	°
9	33	!	57	9	81	Q	105	i	129	ü	153	Ø	177	▀	201	█	225	ß	249	·
10	34	"	58	:	82	R	106	j	130	é	154	Ü	178	▀	202	█	226	Γ	250	·
11	35	#	59	;	83	S	107	k	131	â	155	¢	179	▀	203	█	227	Π	251	√
12	36	\$	60	<	84	T	108	l	132	ä	156	£	180	▀	204	█	228	Σ	252	¤
13	37	%	61	=	85	U	109	m	133	à	157	¥	181	▀	205	=	229	σ	253	²
14	38	♪	62	>	86	U	110	n	134	å	158	฿	182	▀	206	█	230	μ	254	■
15	39	‘	63	?	87	W	111	o	135	ç	159	ƒ	183	▀	207	█	231	γ	255	a
16	40	(64	@	88	X	112	p	136	ê	160	á	184	▀	208	█	232	¤		
17	41)	65	A	89	Y	113	q	137	ë	161	í	185	▀	209	█	233	θ		
18	42	*	66	B	90	Z	114	r	138	è	162	ó	186	▀	210	█	234	Ω		
19	43	!	67	C	91	[115	s	139	ï	163	ú	187	▀	211	█	235	δ		
20	44	,	68	D	92	\	116	t	140	í	164	ñ	188	▀	212	█	236	ø		
21	45	-	69	E	93]	117	u	141	ì	165	Ñ	189	▀	213	F	237	∅		
22	46	.	70	F	94	^	118	v	142	Ä	166	¤	190	▀	214	█	238	€		
23	47	↓	71	G	95	_	119	w	143	Å	167	¤	191	▀	215	█	239	匚		

Tableau des caractères ASCII

Comme vous le voyez, sur 256 valeurs, on peut en stocker des choses.

Ne vous inquiétez pas, vous n'aurez pas à connaître ce tableau par cœur 😊, des fonctions se chargeront de convertir tout ça pour vous. 😊

Bref, après ce petit quart d'heure théorique, revenons au concret.

Nous n'allons pas directement manipuler ces caractères, nous allons utiliser des fonctions pour convertir un caractère en byte, et par extension une chaîne de caractères en tableau de bytes.

Pour convertir un **String** en tableau de **Byte** :

Code : VB.NET

```
Dim MesBytes As Byte() =
System.Text.Encoding.ASCII.GetBytes(MonString)
```

Sachant que ASCII peut être remplacé au choix par UTF7, UTF8, Unicode, UTF32, etc.

Je vous conseille de rester en ASCII pour le moment, il gère les accents et pas mal de caractères spéciaux. Si vous souhaitez en savoir plus sur ces codages, je vous renvoie vers les pages Wikipédia correspondantes.

Maintenant, la réception. De la même manière, vous allez recevoir un tableau de **Byte**, il faut le retransformer en **String**.

Code : VB.NET

```
Dim MonString As String =
System.Text.Encoding.ASCII.GetString(MesBytes)
```

Le concept de cette fonction est la même que la précédente.



Si vous effectuez la transformation `String → Byte` avec le codage ASCII, il faut effectuer la transformation inverse `Byte → String` avec le même codage.

Bon. Ce petit entracte sur le codage de nos données est passé, attaquons le transfert pur et dur.

Pour la communication synchrone, deux fonctions vont suffire pour l'envoi et la réception de données.

La fonction `Send` permet d'envoyer des données (utilisable de la même manière pour le client ou pour le serveur). On lui passe simplement les bytes à envoyer. La fonction nous renvoie le nombre de bytes qui ont été envoyés.

Code : VB.NET

```
Dim BytesEnvoyes As Integer = MonSocketServeur.Send(MesBytes)
```

Et la méthode `Receive` qui prend en paramètre un tableau de `Byte` à remplir cette fois, il faut donc déclarer ce dernier avant la fonction pour le traiter après. Elle renvoie le nombre de bytes lus.

Code : VB.NET

```
Dim MesBytes(255) As Byte
Dim BytesRecus As Integer = MonSocketClient.Receive(MesBytes)
```



Mais il faut que ma réception et mon envoi soient simultanés ?

Pas forcément, il existe un tampon, je m'explique. La méthode de réception, elle, est bloquante, tant qu'aucun byte ne sera présent dans le tampon de réception des données, le programme sera bloqué dessus. La méthode d'envoi, quant à elle, envoie ses données quoi qu'il arrive. Toutefois, si aucun socket ne vient lire, les bytes envoyés resteront dans un tampon. Le tampon est un stockage temporaire de bytes. Il ne peut pas en contenir une infinité, donc il faut bien veiller à ce que ce tampon soit vidé par la fonction de lecture.

Donc pour résumer, l'envoi et la réception peuvent s'effectuer avec plusieurs secondes d'écart et fonctionner quand même. Attention toutefois à bien veiller à lire les données envoyées quand même à un moment.

Pour terminer notre procédure, bien penser à fermer les sockets quand les connexions sont terminées :

Code : VB.NET

```
MonSocket.Close()
```



La méthode `Close()` ferme et libère les ressources mémoire du socket. Donc si vous souhaitez réouvrir la connexion, il faut réinstancier un nouveau socket.

Mini-TP : demande d'heure

Bon, vous avez toutes les cartes en main pour terminer ce petit fil rouge : le client demande l'heure, le serveur lui donne.

Voici la correction :

Serveur

Code : VB.NET

```
Imports System.Net.Sockets
Imports System.Net

Module Module1
    Dim port As String = "8080"
    Dim ip As String = "127.0.0.1"
```

```

Sub Main()
    Dim MonSocketServeur As New
    Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp)
    Dim MonEP As IPPEndPoint = New
    IPPEndPoint(IPAddress.Parse(ip), port)

        MonSocketServeur.Bind(MonEP) 'Lie le socket à cette IP
        MonSocketServeur.Listen(1) 'Se met en mode écoute

        Console.WriteLine("Socket serveur initialisé.")

    While True 'Boucle à l'infini
        Console.WriteLine("En attente d'un client.")
        'Se met en attente de connexion et appelle
        TraitementConnexion() lors d'une connexion.
        Dim SocketEnvoi As Socket = MonSocketServeur.Accept()
        'Bloquant tant que pas de connexion
        TraitementConnexion(SocketEnvoi)
    End While

End Sub

Sub TraitementConnexion(ByVal SocketEnvoi As Socket)
    Console.WriteLine("Socket client connecté, envoi de
l'heure.")
    Try
        Dim Heure As Byte() =
System.Text.Encoding.ASCII.GetBytes(Now.ToString("yyyy-MM-dd HH:mm:ss")) 'Convertit
l'heure en bytes

        Dim Envoi As Integer = SocketEnvoi.Send(Heure) 'Envoie
l'heure au client
        Console.WriteLine(Envoi & " bytes envoyés au client")
    Catch ex As Exception
        Console.WriteLine("Erreur lors de l'envoi du message au
socket. " & ex.ToString())
    End Try
End Sub

End Module

```

Client

Code : VB.NET

```

Imports System.Net.Sockets
Imports System.Net

Module Module1

    Dim port As String = "8080"
    Dim ip As String = "127.0.0.1"

    Sub Main()

        Dim MonSocketClient As New
        Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp)
        Dim MonEP As IPPEndPoint = New
        IPPEndPoint(IPAddress.Parse(ip), port)
        Console.WriteLine("Socket client initialisé.")

    Try
        Console.WriteLine("Connexion au serveur ...")

```

```

        MonSocketClient.Connect(MonEP)
        TraitementConnexion(MonSocketClient)
Catch ex As Exception
        Console.WriteLine("Erreur lors de la tentative de
connexion : " & ex.ToString())
End Try

        Console.ReadLine()

End Sub

Sub TraitementConnexion(ByVal SocketReception As Socket)
        Console.Write("Connecté, réception de l'heure : ")
        Dim Heure(255) As Byte 'Création du tableau de réception
Try
        SocketReception.Receive(Heure) 'Réception

        Console.WriteLine(System.Text.Encoding.ASCII.GetString(Heure))
'Affichage
Catch ex As Exception
        Console.WriteLine("Erreur lors de la réception des
données : " & ex.ToString())
End Try
End Sub

End Module

```

Voilà le code complet et fonctionnel d'une simple communication client/serveur en utilisant les sockets.

Bien sûr, ce code est très sommaire, il est à agrémenter en fonction des besoins. Je pense notamment à la sécurité : les sockets sont très faibles en termes de sécurité, ne tentez pas de faire transiter par ces dernières des données sensibles, à plus forte raison si vous voulez les transmettre via l'internet ou un réseau public.

Toutefois, la rapidité de mise en œuvre, la simplicité d'utilisation et la robustesse des sockets en font de très bons outils pour la mise en place d'une communication de ce type.

Après, à vous d'analyser votre cahier des charges pour savoir si oui ou non elles peuvent être mises en œuvre.

Voilà, vous savez désormais faire communiquer deux programmes entre eux. Vous l'avez compris, cette partie est bien utile, car les fonctions sont très simples. Cependant, ce type de socket est, comme vous le voyez, plutôt « lourd ». Il y a beaucoup de configurations préalables et qui ne nous intéressent pas forcément. Un autre type de sockets va entrer en jeu pour nous simplifier la vie. Allons dès maintenant apprendre à l'utiliser.

- On se sert de l'IP de l'ordinateur que l'on souhaite contacter comme un numéro de téléphone.
- Bien vérifier que le port utilisé par le client et le serveur est le même. Si la connexion échoue, ce port est peut-être déjà utilisé.
- Le socket serveur se met en écoute avec la méthode Listen.
- On envoie et reçoit les données avec Send et Receive. Ces fonctions sont bloquantes.
- On communique avec des bytes, il faut donc effectuer les conversions à l'envoi et lors de la réception.

TCPListener/TCPClient

Nous venons de voir les sockets de base. Nous pouvions spécifier le protocole à utiliser, le type de transfert de données, etc. Cependant, pour nous simplifier la vie, d'autres objets ont été créés pour nous permettre de simplifier ces opérations, notamment les opérations de lecture/écriture.

Vous allez découvrir dans ce chapitre deux objets dédiés à la communication réseau : le **TCPListener** et le **TCPClient**. L'un servant à écouter, l'autre à se connecter.

TCPListener

Je vais vous présenter l'objet **TCPListener**. Il est inclus dans le namespace `System.Net.Socket` également.

Son constructeur est on ne peut plus simple :

Code : VB.NET

```
Dim MonTcpListener As TcpListener = New  
    TcpListener(IPAddress.Parse("127.0.0.1"), 8080)
```

Il prend deux arguments : l'adresse IP à laquelle se « binder » (ici la sienne, car il va écouter la connexion arrivant sur la machine où il est exécuté) et le port.

Eh oui, pas de prise de tête avec une configuration X ou Y, juste l'IP et le port.

Code : VB.NET

```
MonTcpListener.Start()
```

Et cette méthode lance l'écoute. Voilà, notre serveur est démarré, il est en attente de connexions.



Comme pour les sockets précédents, si vous voulez que le serveur puisse écouter *toutes* les IP et donc accepter *toutes* les connexions, spécifiez `IPAddress.Any` à la place de l'IP à laquelle se « binder ». Attention encore une fois avec son utilisation.

TCPClient

À cet instant, notre listener est en écoute. Puisque le listener est un socket simplifié, vous pouvez parfaitement vous y connecter en utilisant le socket client vu au chapitre précédent. Veillez cependant à bien faire correspondre les options (ici c'est un **TCPListener**, votre socket doit donc être configuré pour utiliser le protocole TCP).

Bon, sinon, autant rester dans la logique des choses et utiliser un TCP client. Le **TCPClient** est un objet dérivant également d'un socket, mais simplifié pour pouvoir correspondre avec le **TCPListener**. Voyons tout de suite sa mise en œuvre.

Pour l'instancier et le connecter :

Code : VB.NET

```
Dim MonTCPClient As TcpClient = New TcpClient("127.0.0.1", 8080)
```

Petite particularité qui diffère du **TCPListener**, ici l'IP en argument est de type `String` (encore une simplification).



Ici, le constructeur effectue également la connexion du socket. C'est donc à cet instant que vous devez effectuer vos vérifications, placer un `Try... Catch`, etc. Je vous conseille donc de séparer cela en deux lignes, une ligne où vous allez créer la variable `MonTCPClient`, et une autre ligne, dans un bloc `Try... Catch`, qui effectue l'instanciation (avec `New`).

L'acceptation du serveur

Repassons du côté de notre **TCPListener** pour voir comment effectuer l'acceptation d'un client. Ici, la communication est purement synchrone, cette acceptation sera donc bloquante en attente d'une connexion.

Code : VB.NET

```
Dim SocketClient As Socket = MonTCPListener.AcceptSocket()
```

Voilà l'acceptation, identique à celle du chapitre précédent, et cela retourne le socket client qui vient de se connecter (de type socket, attention).

C'est encore une fois cette passerelle que notre serveur va utiliser pour communiquer.

La communication par flux

Ces deux classes TCPListener et TCPCClient ne peuvent pas communiquer de la même manière que les sockets classiques (Send/Receive), nous allons devoir instaurer une communication par flux.

Le flux va nous simplifier les opérations de lecture et d'écriture entre sockets. Plus besoin de s'embêter à transmettre des tableaux de **Byte**, le flux est optimisé pour utiliser des **String**, et donc du texte.

Pour utiliser les flux, importez System.IO :

Code : VB.NET

```
Imports System.IO
```

Analysons son fonctionnement :

Code : VB.NET

```
Dim MonFlux As NetworkStream = MonSocket.GetStream()
Dim MonReader As StreamReader = New StreamReader(MonFlux)
Dim MonWriter As StreamWriter = New StreamWriter(MonFlux)
```

Trois lignes pour mettre en place notre flux

La première crée un flux réseau à partir du socket. L'opération est identique côté client ou serveur. La mise en place d'un flux côté serveur se fera sur un socket venant d'être accepté.

La seconde met en place un flux de lecture à partir du flux réseau.

La troisième met en place le flux d'écriture.

Nous avons donc deux « boîtes » : une dans laquelle on va regarder ce qu'il y a (*le reader*) pour voir si le client a voulu nous parler ; et une dans laquelle on va écrire ce que l'on veut transmettre au client. Inversement pour le serveur.

Puis, comme pour afficher sur la console, notre envoi et notre réception s'effectueront avec les fonctions ReadLine et WriteLine. On ne peut plus simple !

La lecture**Code : VB.NET**

```
Dim MonMessage As String = MonReader.ReadLine()
```

Fonction retournant simplement une ligne lue sur le stream. Si vous souhaitez lire tout le stream, utilisez plutôt la fonction Read qui prend cette fois un buffer comme argument, mais je vous le déconseille pour le moment.

L'écriture**Code : VB.NET**

```
MonWriter.WriteLine(Message)
MonWriter.Flush()
```

Deux étapes pour l'écriture : la première permet d'écrire dans le flux de sortie (*writer*). On peut donc écrire un **String** sans problème. Une fois cette ligne exécutée, le message sera encore sur le PC local, pour le transmettre à travers le flux (l'envoyer dans la boîte de lecture de l'autre socket), il faut appeler la méthode `Flush`. Vous pouvez donc écrire plusieurs lignes et effectuer un `Flush` seulement une fois toutes les données écrites dans le flux.

La fermeture des flux

Pensez bien à fermer tous vos flux lorsque votre phase de communication est totalement terminée. Dans le cas d'un chat par exemple, cette fermeture ne doit s'effectuer qu'à la déconnexion ou à la fermeture du programme. Dans le cas d'un transfert de fichier, la fermeture doit s'effectuer une fois le transfert terminé.

La méthode `Close()` est la même pour les trois flux (réseau, écriture et lecture).

Pensez cependant à fermer les flux de lecture et écriture avant le flux réseau. Fermez le socket en dernier si vous n'en avez plus besoin.

- `TCPCClient` permet de créer un socket simplifié destiné à être client.
- `TCPListener` permet de créer un socket simplifié destiné à être serveur.
- On les utilise pour ensuite créer un flux de communication entre eux. Cela est très utile lors de transferts réguliers.

Les threads

Nous avons les clés en mains pour effectuer une complète communication entre deux programmes via le réseau. Pas mal en effet, toutefois, admettez qu'il n'y a pas souvent des programmes dédiés uniquement à la communication (à part un serveur, à la limite). Il va donc falloir adapter nos programmes afin de les rendre capables de faire plusieurs choses à la fois : leur travail habituel *et* la connexion réseau. Et ce n'est pas aussi simple qu'il n'y paraît !

Heureusement, les threads vont nous aider à résoudre ce petit problème.

Introduction

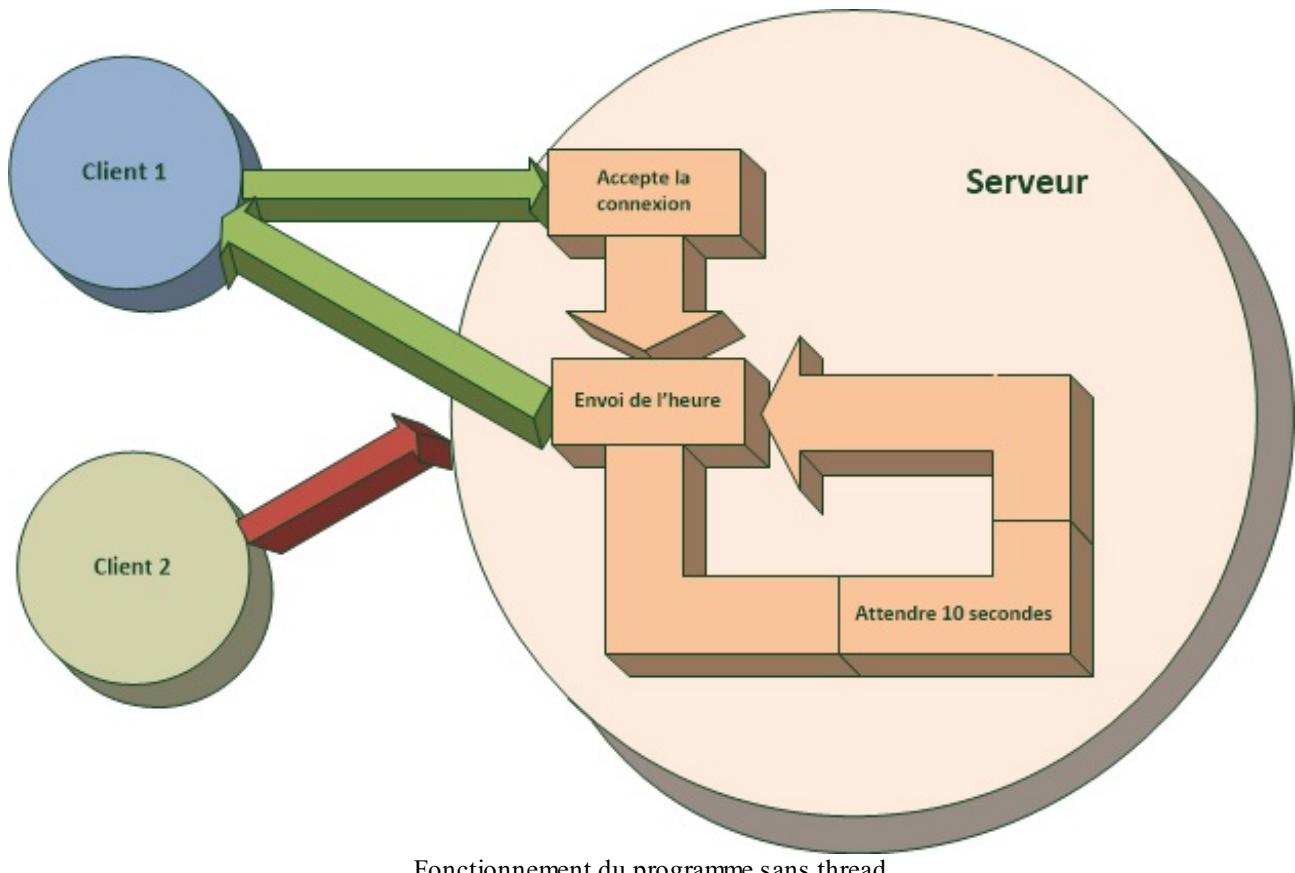
Commençons par le commencement, qu'est-ce qu'un thread et à quoi va-t-il bien pouvoir vous servir ?

Un thread est un processus. Par exemple, tous les programmes que nous avons faits jusqu'à maintenant ne contiennent qu'un seul processus, qu'un seul thread donc. Chaque ligne de code était exécutée après la précédente et ainsi de suite. Avec différents threads, les fonctions peuvent être appelées simultanément. Une fonction de calcul A et une fonction de calcul B peuvent être lancées simultanément. Cela n'a pas vraiment d'intérêt dans ce cas, car récupérer le résultat du thread va être très complexe. On va résumer cela en disant que lancer un thread revient à lancer un programme séparé. Par exemple, un thread ne pourra pas directement accéder aux composants visuels (interface graphique) du programme qui l'a créé, il va falloir effectuer des étapes particulières pour y accéder (nous y reviendrons plus tard).



C'est pas très concret, j'y comprends rien moi...

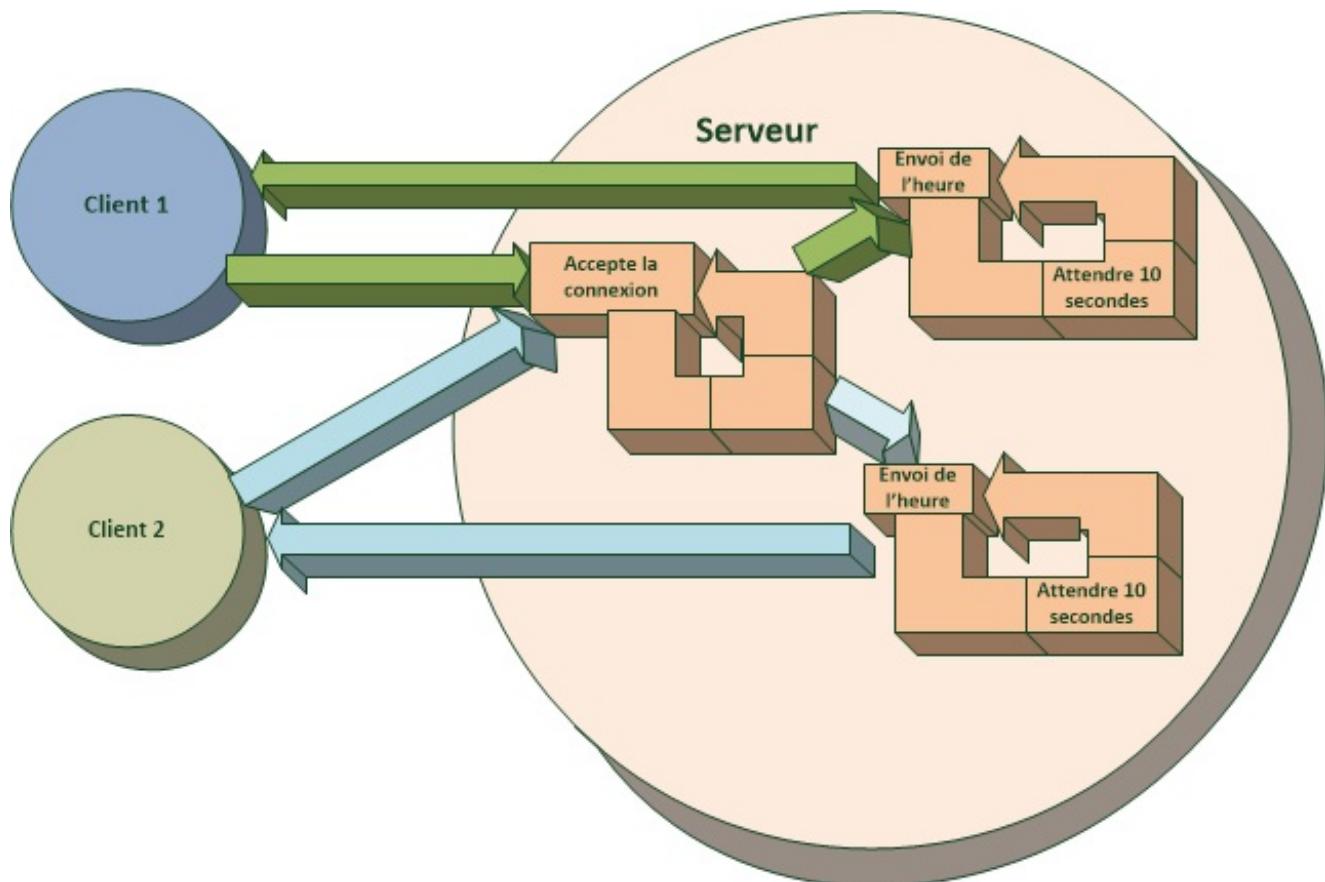
Bon, je vais vous donner un exemple où les threads vont être extrêmement utiles. Imaginons que notre programme demandant l'heure transmette l'heure au client non pas une fois, mais toutes les dix secondes (pas vraiment utile, mais c'est pour l'explication). Nous aurions un fonctionnement ressemblant à celui présenté à la figure suivante.



Dans ce schéma, vous voyez qu'une fois la connexion établie entre le client 1 et le serveur les deux effectuent le travail d'envoi/réception de l'heure et sont bloqués dans une boucle. Le client 2 qui veut se connecter, lui, ne pourra pas, car le serveur s'occupe désormais de l'envoi de l'heure, la phase d'acceptation client est passée, il n'y retournera plus tant que le client 1 sera connecté.

Imaginez maintenant que l'acceptation d'un client et la phase d'envoi de l'heure soient deux processus séparés. Alors, si le serveur crée un processus (thread) par client, chaque client pourra être « servi » de son côté. C'est ce que montre la figure

suivante.



Ici par contre, le serveur a envoyé, une fois l'acceptation effectuée, chaque client vers un processus (thread) dédié à l'envoi de l'heure vers ledit client.

Ici, le serveur peut donc accepter un très grand nombre de clients et gérer chaque connexion de son côté.

En résumé...

Pour résumer tout ça, on utilisera les threads pour une tâche bloquante dans notre programme. Si on la lance dans un thread séparé, elle n'embêtera pas le traitement de notre programme. Ils peuvent être utilisés pour des tâches d'impression, de longs travaux sur les fichiers, des recherches, etc.

Vous avez sûrement parfois remarqué dans vos interfaces graphiques un « blocage », et si vous tentez d'interagir avec, vous obtenez une erreur du type « L'application ne répond pas... ». Cela arrive quand un long traitement est effectué sur le même thread que celui qui gère l'interface graphique.

Voilà donc pour résumer l'intérêt des threads. Allons tout de suite voir comment les mettre en œuvre.

Notre premier thread

Nous en avons fini avec la minute théorique. Désormais, nous allons nous concentrer sur la mise en œuvre des threads.

Pour créer un thread, il faut une fonction à partir de laquelle créer le processus. Dans l'exemple ci-dessus, la fonction qui sera appellée en tant que thread sera celle qui envoie l'heure toutes les dix secondes.

La mise en œuvre des threads est très rapide :

Code : VB.NET

```
Dim MonThread As New Thread(AddressOf FonctionThread)
MonThread.Start()
```

Et l'import :

Code : VB.NET

```
Imports System.Threading
```

Comme d'habitude, déclaration et instantiation de l'objet simultanément. Ici le paramètre du constructeur est **AddressOf** FonctionThread. On utilise le mot-clé **AddressOf** qui nous permet de récupérer l'adresse de la méthode que l'on souhaite appeler. Pour résumer, cela veut simplement dire que ce sera la méthode FonctionThread qui sera exécutée au démarrage du thread.

Le démarrage du thread ne s'effectue pas lors de la déclaration, il faut le démarrer en utilisant la méthode Start () .



Attention, ne surtout pas mettre les parenthèses dans la méthode passée avec **AddressOf**, c'est une source d'erreurs fréquente.

Il n'y a pas beaucoup de fonctions à connaître lorsque l'on manipule les threads. Je vais vous donner les vitales :

Suspend/Resume

Vous pouvez facilement mettre en pause et reprendre un thread.

Code : VB.NET

```
MonThread.Suspend()
```

Ce code mettra en pause le thread MonThread. Son exécution ne continuera pas tant qu'un **Resume** ne sera pas appelé.

Code : VB.NET

```
MonThread.Resume()
```

Ce code reprendra un thread en pause.

Join

Cette méthode vous sera utile. Elle attend la fin du thread désigné pour continuer l'exécution du programme. C'est donc une méthode bloquante.

Si dans le programme principal je lance un thread MonThread et que j'ai besoin de ses résultats de calcul pour continuer mon exécution, je peux écrire :

Code : VB.NET

```
MonThread.Join()
```

Cette ligne ne sera donc pas passée tant que le thread MonThread ne sera pas terminé, autrement dit lorsque la fonction FonctionThread ne sera pas terminée (car c'est l'adresse de la fonction que j'ai passée au thread à la déclaration).

Abort

Je déconseille l'utilisation de cette méthode en fonctionnement normal. Cette méthode arrête brusquement un thread en cours d'exécution. Il est préférable de l'utiliser dans des cas d'urgence (fermeture du programme, erreur, etc.).

Code : VB.NET

```
MonThread.Abort()
```

Ce code arrête l'exécution du thread.



Hey, comment je fais pour appeler des fonctions avec paramètres en utilisant ton **AddressOf** ?

En théorie vous ne pouvez pas. D'ailleurs, vous ne pouvez pas avoir un **Return** dans votre thread. Le seul moyen pour passer des paramètres à notre thread est de passer par une **Class**.

Il fallait y penser : vous instanciez une classe X avec trois membres privés. Le constructeur de cette fonction effectue leur attribution. Il vous reste à déclarer votre thread avec **AddressOf X.MonFonction** et cette fonction interne à la classe pourra alors utiliser les membres privés de cette dernière.

Et pour récupérer des valeurs résultant du thread, il faut les stocker dans un membre public de cette classe ou dans une variable globale. On en parle juste après.

La synchronisation

Un concept important pendant le développement de threads est la synchronisation.

Je m'explique : vous avez x threads de lancés et vous souhaitez partager des informations entre eux. Imaginez que vous ayez un thread qui s'occupe de lire une valeur en base de données toutes les y secondes et un autre qui doit effectuer un traitement sur cette variable.

Le problème ici est que, même si vous lancez le thread qui effectue la lecture avant celui qui effectue la modification, vous ne pouvez pas être certains de l'ordre d'exécution. La modification peut s'effectuer alors avant la lecture et là, **badaboum** ! Il faut mettre en place une synchronisation.

La variable globale

Première méthode très rapide à mettre en œuvre : la synchronisation par variable globale.

C'est tout simplement une variable à portée globale au module (donc en dehors d'une fonction) qui va être accessible même depuis nos threads. Les threads la modifieront pour voir si ce dernier peut effectuer son travail.

Avec le thread 1 qui veut lire en BDD et écrire dans X et le thread 2 qui veut traiter X. On les synchronise par la variable booléenne **LectureTerminee** :

- Thread 2 : je veux traiter la variable X, est-ce que **LectureTerminee** est à **True** ? Non. Je patiente.
- Thread 1 : j'ai terminé ma lecture en base de données, j'écris dans X et je mets **LectureTerminee** à **True**.
- Thread 1 : je veux relire en base de données et réécrire la variable, est-ce que **LectureTerminee** est à **False** ? Non. Je patiente.
- Thread 2 : je veux traiter la variable X, est-ce que **LectureTerminee** est à **True** ? Oui : je traite les données et je replace **LectureTerminee** à **False**.
- Thread 1 : je veux relire en base de données et réécrire la variable, est-ce que **LectureTerminee** est à **False** ? Oui, je commence ma lecture en BDD.
- Et ainsi de suite...

Quand les threads patientent, il faut bien sûr les faire entrer dans une boucle de type **While** ou **Until** en vérifiant la variable de temps en temps.

Attention, veillez bien à ne pas faire tourner le **While** ou le **Until** à l'infini, il faut lui faire faire des pauses avec :

Code : VB.NET

```
Thread.Sleep(1000)
```

... où le paramètre est le temps d'attente en millisecondes (ici 1000 : 1 seconde).

Le SyncLock

Le **SyncLock** est lui plus utile pour vérifier qu'une variable ne sera pas modifiée par deux threads au même moment (on ne saurait alors plus où on en est), ou alors qu'elle est modifiée pendant le traitement d'un autre thread.

L'avantage du **SyncLock** est que les autres blocs tentant d'accéder à la même variable qu'un thread qui l'a déjà bloquée seront mis en attente. Exactement comme si on effectuait un **While** et des **Sleep** en attendant que cette variable change.

La mise en œuvre est particulière, il faut tout d'abord déclarer une variable qu'on va utiliser comme variable de contrôle, puis on peut verrouiller cette variable :

Code : VB.NET

```
Dim VariableLock As Object = New Object()

Sub FonctionThread()

    SyncLock VariableLock
        'Traitement d'une variable commune aux threads
    End SyncLock

End Sub
```

Oui, c'est un bloc avec **End**. Et c'est justifié : pendant qu'on est à l'intérieur du **SyncLock**, on empêche les autres threads qui ont aussi effectué leur **SyncLock** d'entrer dans leur bloc. Une fois qu'un thread a atteint le **End**, un autre entre dedans et ainsi de suite...

SemaphoreSlim

L'objet `SemaphoreSlim` est encore un autre moyen de synchronisation. Il permet d'autoriser X threads de continuer leur exécution.

Le `SemaphoreSlim` est en fait une barrière de parking. Le parking a 10 places, si les 10 places sont occupées et qu'une voiture souhaite entrer, la barrière restera fermée. Si une voiture quitte le parking, une place se libère et la barrière s'ouvre. Une fois la voiture entrée, il n'y a de nouveau plus de place disponible et la barrière se referme. Et ainsi de suite.

La mise en œuvre nécessite encore une fois une variable globale, mais cette fois de type `SemaphoreSlim` :

Code : VB.NET

```
Dim MonSemaphore As SemaphoreSlim = New SemaphoreSlim(10)
'Initialisation d'un SemaphoreSlim avec 10 places

Sub MonThread

    MonSemaphore.Wait() 'Attend qu'une place soit disponible
    'Une place est libre, l'exécution continue
    'Mon super traitement
    MonSemaphore.Release() 'Libère une place dans SemaphoreSlim
    pour les autres

End Sub
```

Mise en œuvre encore une fois très simple. Notez que le `Wait()` est une fonction bloquante. Il effectue à la fois l'attente de la libération du `SemaphoreSlim` et la prise d'une place lorsqu'une est disponible. Ne pas oublier le `Release` à la fin.



Si vous souhaitez pousser un peu plus dans les `Semaphore`, regardez la documentation pour connaître les différents arguments de ces fonctions et jetez un coup d'œil du côté de l'objet `Semaphore`.

Les Windows Forms et les threads

Si vous avez fait vos tests de votre côté pendant ce tutoriel, vous avez sûrement eu des problèmes si vous avez voulu les interfaçer avec les Windows Forms (l'interface graphique).

En effet, il y a un problème. Votre thread principal, le seul et l'unique, est celui qui s'occupe de toute la création et la gestion de l'interface graphique. C'est un peu comme un manager. Si un autre thread va vouloir accéder à ses objets graphiques (ses subordonnés), il va y avoir une erreur, un thread ne peut pas accéder aux ressources d'un autre thread, le manager ne veut pas qu'un autre manager vienne l'embêter dans son management.

Il va falloir effectuer une opération ninja pour aller modifier les ressources d'un autre thread, j'ai nommé cette opération l'opération `Delegate` (prononcez « diliguaite »).

Les delegates

Que sont donc ces delegates ? On peut traduire ça par « délégué » et je trouve que ça correspond bien à son rôle. Le manager du thread secondaire, plutôt que d'aller donner directement des ordres aux subordonnés du manager principal, va lui donner des instructions pour gérer ses subordonnés. Il va déléguer ce travail au thread principal. Alors, les éléments graphiques seront bien modifiés par le thread principal et tout ira bien.

Commençons donc à analyser la mise en œuvre d'un **Delegate**(qui est plutôt folklorique) :

Code : VB.NET

```
Delegate Sub dTest()

Private Sub Test()
    'Ma superbe fonction
End Sub
```

Nous avons donc deux parties dans un delegate :

- 1. La déclaration du delegate ;
- 2. La fonction à appeler.

La déclaration du delegate s'effectue comme une variable globale, une seule ligne commencée par le mot-clé **Delegate** et le nom du delegate.

Vient ensuite la fonction ou la méthode qui sera appelée, c'est à l'intérieur de celle-ci que seront modifiés les composants graphiques (un .text, un .enable, etc.).

Si vous avez des paramètres à passer à votre fonction, ce n'est pas vraiment plus compliqué :

Code : VB.NET

```
Delegate Sub dTest(ByVal Texte1 As String, ByVal Texte2 As String)

Private Sub Test(ByVal Texte1 As String, ByVal Texte2 As String)
    'Ma superbe fonction avec arguments
End Sub
```

Il faut simplement recopier le prototype de votre fonction dans le delegate. Les noms des paramètres dans le delegate n'ont pas vraiment d'importance, mais pour éviter les erreurs je vous conseille de bien copier/coller ces derniers dans le delegate. Quoi qu'il en soit, les arguments doivent avoir le même ordre : si votre fonction appelle dans cet ordre : **Boolean, String, Boolean**, le delegate doit avoir aussi cet ordre : **Boolean, String, Boolean**.

Passons maintenant à l'étape qui doit vous intéresser : appeler le delegate.

Appel du delegate

On va voir un nouveau mot (j'ai l'impression de retourner en primaire 😊) : le **Invoke**.

Invoke va « invoquer » un delegate. On résume : on va invoquer un délégué pour faire notre travail à notre place (elle est pas belle la vie ?).

Code : VB.NET

```
Me.Invoke(New dTest(AddressOf Test), Texte1, Texte2)
```

Alors, si votre delegate est sur la même classe, c'est un **Me** qui précède, sinon c'est le nom de la classe étrangère. Ensuite, vient la fonction **Invoke**. Elle a **x** paramètres où **x** est le nombre d'arguments du delegate à invoquer + 1 (+1 car le premier est le delegate lui-même). Le premier argument est donc le delegate que l'on instancie (avec **New**) et l'on passe un paramètre à ce

delegate, l'adresse de la fonction à appeler. Vous vous souvenez du mot-clé **AddressOf** ? Non ? Eh bien le revoilà ! Il suit ensuite les différents arguments de la fonction à invoquer, dans l'ordre aussi. 😊

Compliqué ? On va résumer comme ça :

- 1. J'écris la fonction qui va modifier les éléments graphiques.
- 2. Je crée le delegate avec les même arguments que ma fonction, je lui donne le nom de ma fonction précédé de « d » pour me souvenir que c'est le delegate de cette dernière.
- 3. J'invoque mon delegate en utilisant le mot-clé `Invoke` et en spécifiant le delegate à appeler, et dans ce delegate, la fonction à laquelle il fait référence.

Bon, j'admetts que cette histoire de delegate est plutôt indigeste. Si vous ne comprenez pas bien leur utilisation, utilisez simplement l'exemple que j'ai donné plus haut en modifiant avec votre fonction et vos arguments. L'intellisense de Visual Studio est là pour vous aider aussi.

Prenez le temps de lire et relire cette partie sur les threads, ils vont être très importants lorsque nous ferons des programmes plus complexes en réseau.

- Les threads permettent de désynchroniser l'exécution d'une fonction, ainsi si une exécution est bloquante, le programme principal ne sera pas bloqué.
- On crée le thread en spécifiant l'adresse de la fonction qu'il exécutera.
- La méthode `Start` permet de démarrer son exécution.

TP : ZChat

Ceci est le dernier TP que vous aurez à faire, il va reprendre toutes les notions que vous avez acquises au cours de votre apprentissage. N'hésitez pas à prendre le temps de bien le faire et de ne pas vous précipiter à la correction. Si vous avez des doutes sur comment faire telle ou telle chose, ne vous en faites pas, revenez sur vos pas et relisez le chapitre correspondant.

Profitez de ce TP pour essayer les deux manières de communiquer que nous venons d'étudier, chacune a son avantage.

Cahier des charges

Oui !!! Enfin le moment où l'on met en œuvre tout ce que l'on a appris ! Pour cela, un classique : le chat !

Je parle là du « tchat » et non pas de l'animal... Alors pour ceux qui ne savent pas, un chat (tchat) est un programme permettant de communiquer avec d'autres personnes : Skype, MSN, mIRC, le chat de Facebook, etc.

On va donc en créer un en VB .Net en utilisant toutes les connaissances que nous venons d'accumuler pendant ce chapitre. Il va donc être en réseau (de quoi discuter avec les collègues au bureau).

Objectifs

Notre TP de chat sera composé de deux parties : le client et le serveur.

Le client contiendra l'interface graphique, c'est le programme que nous allons diffuser à nos collègues. Il faudra pouvoir grâce à lui :

- Se connecter au serveur de chat ;
- Spécifier son pseudo ;
- Écrire des messages ;
- Lire les messages des autres utilisateurs du chat.

Le serveur, quant à lui, sera lancé uniquement sur un seul poste. Il va s'occuper d'accepter les connexions des clients, de récupérer les messages qu'ils envoient et de les diffuser aux autres clients.

Astuces

Je vous le dit tout de suite (je ne suis pas un sadique non plus 😊), le serveur et le client auront *besoin de threads*. Le client aura quant à lui besoin plus spécifiquement de *delegates* (niark niark !).

Pour l'interface graphique du client, ne vous compliquez pas la vie.

Et la communication devra être utilisée soit avec les sockets purs et durs, soit avec les TCPListener/TCPClient.

Voilà, vous avez toutes les informations ! À vous les Zéros !

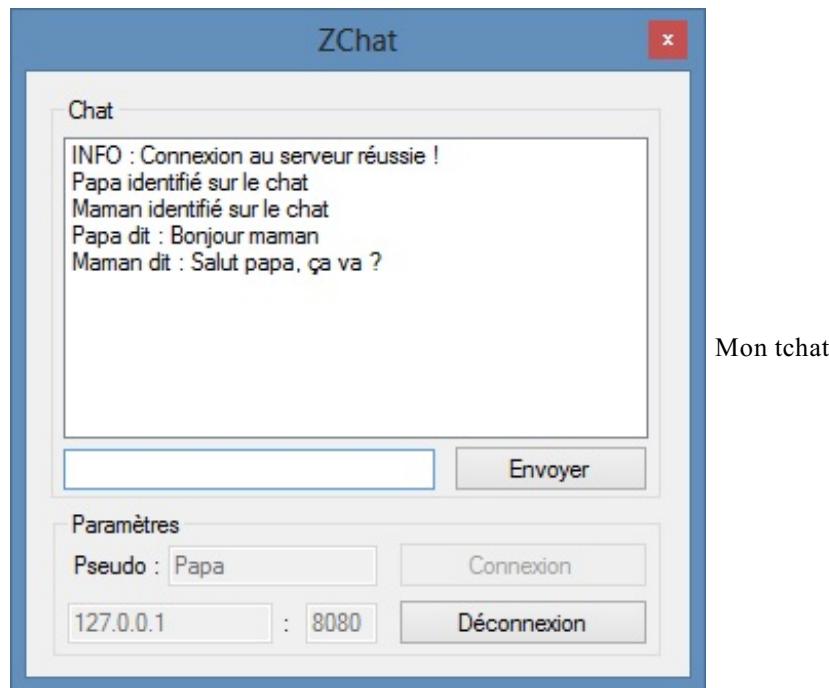
La correction

J'espère que ça s'est bien passé amis Zéros !

Le client

L'interface

Nous allons commencer par la correction du client. La figure suivante vous montre l'interface que j'ai réalisée.



Les messages sont affichés dans une listbox où j'ai choisi de rajouter un item par message. Vous pouvez parfaitement utiliser une textbox vérouillée et en multilignes.

Le reste est très basique : boutons et textbox.

Code

Code : VB.NET

```

Imports System.Net.Sockets
Imports System.Net
Imports System.Threading

Public Class ZChat

    Dim MonSocketClient As Socket
    Dim MonThread As Thread

    'Delegate pour écrire un message
    Delegate Sub dEcrit(ByVal Texte As String)
    Private Sub Ecrit(ByVal Texte As String)
        Me.LST_MESSAGES.Items.Add(Texte)
    End Sub

    'Delegate pour modifier les contrôles suite à une déconnexion
    Delegate Sub dDeconnexion()
    Private Sub Deconnexion()
        Me.TXT_IP.Enabled = True
        Me.TXT_PORT.Enabled = True
        Me.TXT_MESSAGE.Enabled = False
        Me.TXT_PSEUDO.Enabled = True
        Me.BT_CONNEXION.Enabled = True
        Me.BT_DECONNEXION.Enabled = False
        Me.BT_ENVOI.Enabled = False
    End Sub

    Private Sub enFermeture() Handles Me.FormClosing
        If Not MonSocketClient Is Nothing Then 'Si le socket a été
        créé
            MonSocketClient.Close() 'On le ferme
        End If
        If Not MonThread Is Nothing Then 'Si le thread a été créé
            If MonThread.IsAlive Then 'S'il tourne

```

```

        MonThread.Abort() 'On le stoppe
    End If
End If
End Sub

Private Sub BT_DECONNEXION_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles BT_DECONNEXION.Click
    Deconnexion() 'On effectue la modification des contrôles
    MonSocketClient.Close() 'On ferme le socket
    EcritureMessage("Deconnecté sur serveur.", 1)
End Sub

Private Sub BT_CONNEXION_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles BT_CONNEXION.Click
    'Vérification des entrées
    If Me.TXT_PSEUDO.Text = "" Then
        EcritureMessage("Erreur, vous devez spécifier un pseudo
avant de vous connecter", 2)
        Return 'Ne continue pas
    End If
    If Me.TXT_IP.Text = "" Or Me.TXT_PORT.Text = "" Then
        EcritureMessage("Erreur, vous devez spécifier un port et
une adresse ID à laquelle vous connecter.", 2)
        Return 'Ne continue pas
    End If

    MonSocketClient = New Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp) 'Initialise le socket

    Try
        Dim MonEP As IPPEndPoint = New
IPPEndPoint(IPAddress.Parse(Me.TXT_IP.Text), Me.TXT_PORT.Text) 'Entre
les informations de connexion
        MonSocketClient.Connect(MonEP) 'Tente de se connecter
        TraitementConnexion()
    Catch ex As Exception
        EcritureMessage("Erreur lors de la tentative de
connexion au serveur. Vérifiez l'ip et le port du serveur." &
ex.ToString, 2)
    End Try

    End Sub

Private Sub BT_ENVOI_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles BT_ENVOI.Click
    Dim Mess As Byte() =
System.Text.Encoding.UTF8.GetBytes(Me.TXT_MESSAGE.Text)
    Dim Envoi As Integer = MonSocketClient.Send(Mess) 'Envoi du
pseudo au serveur
    Me.TXT_MESSAGE.Text = "" 'Efface la ligne
End Sub

Sub TraitementConnexion()
    EcritureMessage("Connexion au serveur réussie !", 1)
    'Change les statuts des contrôles
    Me.TXT_IP.Enabled = False
    Me.TXT_PORT.Enabled = False
    Me.TXT_MESSAGE.Enabled = True
    Me.TXT_PSEUDO.Enabled = False
    Me.BT_CONNEXION.Enabled = False
    Me.BT_DECONNEXION.Enabled = True
    Me.BT_ENVOI.Enabled = True

    'Envoi du pseudo au serveur
    Dim Mess As Byte() =
System.Text.Encoding.UTF8.GetBytes(Me.TXT_PSEUDO.Text)
    Dim Envoi As Integer = MonSocketClient.Send(Mess) 'Envoi du
pseudo au serveur

    MonThread = New Thread(AddressOf ThreadLecture)

```

```

        MonThread.Start()
    End Sub

    Sub ThreadLecture()
        While (MonSocketClient.Connected) 'Tant qu'on est connecté
au serveur
            Dim Bytes(255) As Byte
            Dim Recu As Integer
            Try
                Recu = MonSocketClient.Receive(Bytes)
            Catch ex As Exception 'Erreur si fermeture du socket
pendant la réception
                EcritureMessage("Connexion perdue, arrêt de la
réception des données ...", 1)
                If Not Me.IsDisposed Then 'Si ce n'est pas le client
qui est en cours de fermeture
                    Me.Invoke(New dDeconnexion(AddressOf
Deconnexion))
                End If
            End Try
            Dim Message As String
            Message = System.Text.Encoding.UTF8.GetString(Bytes)
            Message = Message.Substring(0, Recu)
            EcritureMessage(Message)
        End While
    End Sub

    ''' <summary>
    ''' Écrit un message dans la fenêtre de chat
    ''' </summary>
    ''' <param name="Message"></param>
    ''' <param name="Type">0 : Message normal | 1 : Information | 2
: Erreur </param>
    ''' <remarks></remarks>
    Sub EcritureMessage(ByVal Message As String, Optional ByVal Type
As Integer = 0)
        Dim Texte As String = ""
        Select Case Type
            Case 1
                Texte &= "INFO : "
            Case 2
                Texte &= "ERREUR : "
            Case Else
        End Select
        Texte &= Message
        Try
            Me.Invoke(New dEcrit(AddressOf Ecrit), Texte)
        Catch ex As Exception
            Exit Sub
        End Try
    End Sub
End Class

```

Pas de complications dans ce code, j'explique :

- On envoie le pseudo lors de la connexion ;
- On affiche sans traitement tout ce que le serveur nous envoie ;
- Un thread de lecture effectue cette réception, puis l'affichage ;
- Si la réception vient à échouer, on nous considère comme déconnecté (serveur off, connexion réseau coupée, etc.).

Ici j'ai utilisé un socket plutôt qu'un TCPClient. Donc, j'ai utilisé les méthodes Receive et Send.

Il faut pas mal de tests **Try... Catch** en réseau, une erreur peut arriver à n'importe quelle étape : connexion, réception, etc.

Et bien veiller à fermer le thread et le socket lors de la fermeture du programme (sinon le thread continuera son exécution à l'arrière-plan et le programme ne se fermera pas complètement).

Je vous laisse faire les correspondances pour deviner quel nom j'ai attribué à quel composant graphique, mais je pense avoir été assez clair. 😊

Le serveur

Code : VB.NET

```
Imports System.Net.Sockets
Imports System.Net
Imports System.Threading

Module ServeurChat

    Dim port As String = "8080"
    Dim ListeClients As List(Of Client) 'Liste destinée à contenir
    les clients connectés

    Sub Main()

        'Crée le socket et l'IP EP
        Dim MonSocketServeur As New
        Socket(AddressFamily.InterNetwork, SocketType.Stream,
        ProtocolType.Tcp)
        Dim MonEP As IPEndPoint = New IPEndPoint(IPAddress.Any,
        port)

        ListeClients = New List(Of Client) 'Initialise la liste

        MonSocketServeur.Bind(MonEP) 'Lie le socket à cette IP
        MonSocketServeur.Listen(1) 'Se met en écoute

        Console.WriteLine("Socket serveur initialisé sur le port " &
        port)

        While True 'Boucle à l'infini
            Console.WriteLine("En attente d'un client.")
            'Se met en attente de connexion et appelle
            TraitementConnexion() lors d'une connexion.
            Dim SocketEnvoi As Socket = MonSocketServeur.Accept()
            'Bloquant tant que pas de connexion
            TraitementConnexion(SocketEnvoi) 'Traite la connexion
            du client
        End While

    End Sub

    Sub TraitementConnexion(ByVal SocketEnvoi As Socket)

        Console.WriteLine("Socket client connecté, création d'un
        thread.")

        Dim NouveauClient As New Client(SocketEnvoi) 'Crée une
        instance de « client »
        ListeClients.Add(NouveauClient) 'Ajoute le client à la
        liste

        'Crée un thread pour traiter ce client et le démarre
        Dim ThreadClient As New Thread(AddressOf
        NouveauClient.TraitementClient)
        ThreadClient.Start()
    End Sub

    Sub Broadcast(ByVal Message As String)

        'Écrit le message dans la console et l'envoie à tous les
```

```

clients connectés
    Console.WriteLine("BROADCAST : " & Message)
    For Each Cli In ListeClients
        Cli.EnvoiMessage(Message)
    Next

End Sub

Private Class Client
    Private _SocketClient As Socket 'Le socket du client
    Private _Pseudo As String 'Le pseudo du client

    'Constructeur
    Sub New(ByVal Sock As Socket)
        _SocketClient = Sock
    End Sub

    Sub TraitementClient()

        Console.WriteLine("Thread client lancé. ")

        'Le client vient de se connecter
        Dim Bytes(255) As Byte 'Tableau de 255 : on ne reçoit
        que 255 caractères au maximum
            'Réception du premier message contenant le pseudo
            Dim Recu As Integer
            Try
                Recu = _SocketClient.Receive(Bytes) 'Reçoit les
                premières données : le pseudo
                Catch ex As Exception
                    Console.WriteLine("Erreur pendant la réception du
                    pseudo d'un client ... Fermeture du client")
                Return
            End Try

            _Pseudo = System.Text.Encoding.UTF8.GetString(Bytes)
            _Pseudo = _Pseudo.Substring(0, Recu) 'Retire les
            caractères inutiles

            Broadcast(_Pseudo & " identifié sur le chat") 'Diffuse
            le message à tout le monde
            While (_SocketClient.Connected)
                Try
                    Dim Message As String
                    Recu = _SocketClient.Receive(Bytes)
                    'Message reçu
                    Message =
                    System.Text.Encoding.UTF8.GetString(Bytes)
                    Message = Message.Substring(0, Recu) 'Retire les
                    caractères inutiles
                    Broadcast(_Pseudo & " dit : " & Message)
                    'Diffuse le message à tout le monde
                    Catch ex As Exception 'Le client est déconnecté
                        ListeClients.Remove(Me) 'Le supprime de la
                        liste des clients connectés
                        _SocketClient.Close() 'Ferme son socket
                        Broadcast(_Pseudo & " déconnecté.") 'Diffuse le
                        message à tout le monde
                    Return 'Fin de la fonction
                End Try
            End While

        End Sub

        Sub EnvoiMessage(ByVal Message As String)
            Dim Mess As Byte() =
            System.Text.Encoding.UTF8.GetBytes(Message)
            Dim Envoi As Integer = _SocketClient.Send(Mess)
            Console.WriteLine(Envoi & " bytes envoyés au client " &
            Pseudo)
        End Sub
    End Class

```

```
      End Sub  
End Class
```

End Module

Je pense avoir tout commenté, mais je vais résumer le fonctionnement.

Quand un client se connecte, une classe nommée `Client` est instanciée et ajoutée à une liste (un tableau). Cette classe est utilisée pour stocker le socket du client. Un thread est ensuite lancé, il s'occupe d'écouter les messages reçus par le client. Le premier message est obligatoirement le pseudo du client.

Dès qu'un message doit être diffusé, il appelle la fonction `Broadcast`. Cette fonction parcourt tous les clients connectés (ceux de la liste) et appelle leur fonction `EnvoiMessage` qui envoie un message à un client. Ainsi, chaque client reçoit le message que l'on souhaite diffuser.

Vous l'avez remarqué, j'ai encore utilisé ici la classe de base : `Socket`. À vous d'adapter ce projet pour utiliser des `TCPClient/TCPListener`.

Conclusion

Nous voilà au bout de notre dernier TP ensemble, les amis.

J'espère qu'il a su éclairer vos lanternes sur la communication réseau.

Le chat est un exemple basique de la communication, mais après libre à vous d'adapter ce programme à tous vos besoins : messages plus longs, mise en forme etc.

N'hésitez pas à mettre en œuvre les `TCPClient`/`TCPListener` en les utilisant dans ce programme, je voulais bien vous montrer l'utilisation des sockets qui offrent plus de possibilités qu'eux.

Améliorations possibles

N'hésitez pas à améliorer votre chat ! Le côté serveur est une bonne base, maintenant personnalisez le client !

- Essayer de trouver un moyen pour faire défiler les messages de bas en haut, comme un vrai chat.
 - Mettez de la couleur aux messages d'erreur ou d'information.
 - Intégrer des smileys, de la mise en forme, que sais-je encore !
 - Implémentez une fonctionnalité d'envoi de fichiers aux clients (ça c'est du challenge 😊).

Voilà très chers Zéros, le réseau c'est terminé ! Vous êtes assez grands désormais !

- Ce TP est une bonne base pour effectuer de la communication plus poussée, comme de la diffusion de fichiers ou de musiques.
 - Bien penser à la sécurité lorsque l'on pratique du développement réseau.

La partie sur le réseau, c'est terminé ! Lisez les annexes pour quelques notions complémentaires, mais je pense vous avoir transmis tout mon savoir !

Partie 6 : Annexes

Les annexes peuvent être consultées n'importe quand, elles peuvent servir pour sécuriser votre programme ou l'améliorer en général.

Gérer les erreurs

Votre superbe IDE Visual Basic Express 2010, ou Visual Studio faute de mieux, permet facilement de retrouver et de gérer les erreurs. Il indique la ligne ayant provoqué l'erreur, l'explication de l'erreur (en français) et parfois comment la résoudre. Mais pour ce qui est des autres erreurs ? Les erreurs qui ne sont pas liées à notre programme ? Ras le 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 présente, 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 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.

Voici sa 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 ne 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 fonction ! À 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 où la fonction a été appelée, si je n'ai pas de valeur, ça va également planter !

Hum ! c'est pas faux tout ça, passons alors au **Finally**.

Finally

Dans le **Try** nous avons d'autres instructions 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 ? S'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 suivi 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 à déplorer, le **Return** de votre fonction aura la priorité et rendra l'autre inopérant.

Si c'est une demande de connection 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

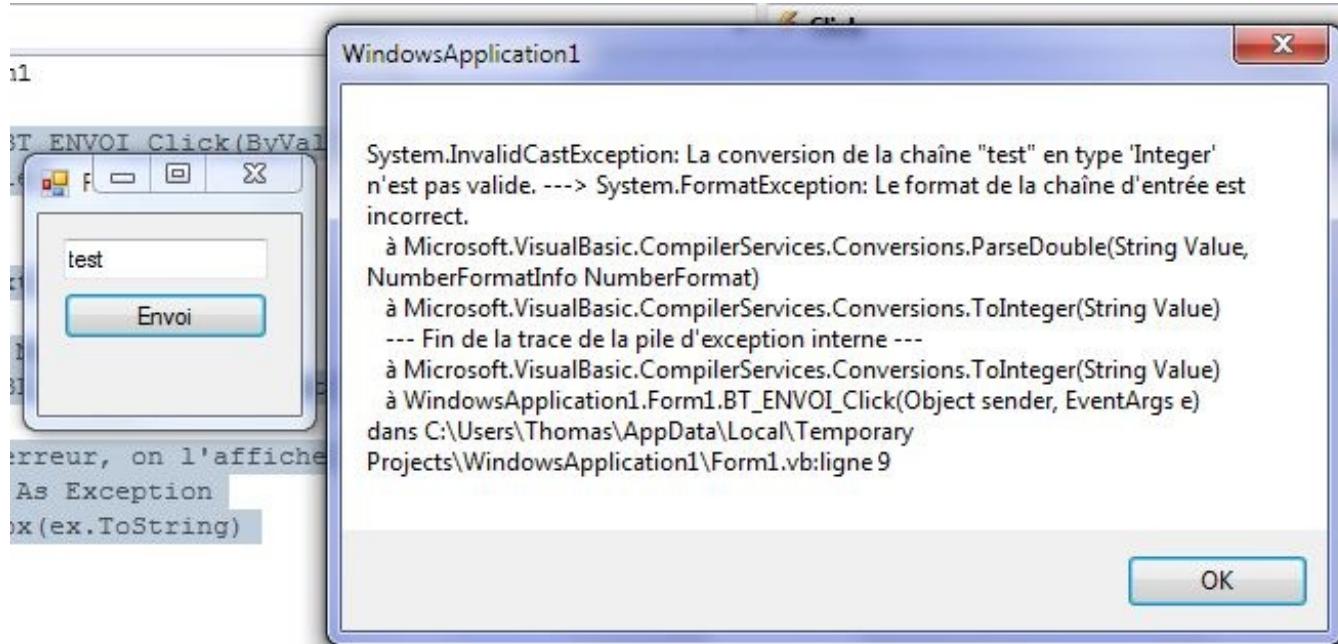
        Dim MonTxt As Integer

        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**. Le résultat se trouve à la figure suivante.



résultat du code précédent

Vous allez me dire que l'utilisateur lambda n'en a rien à faire de notre message d'erreur, que lui, il veut que ça 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 direz-vous ? Pas forcément, pourquoi ne pas utiliser ce **Try... Catch** avec son **Throw** dans tous vos accès aux bases de données, et un **Try... Catch** avec une gestion spécifique d'erreur dans la fonction qui les appelle toutes ?

Une seule gestion d'erreur pour vérifier des dizaines de requêtes, ce n'est pas magnifique ? 😊

- On commence par le bloc **Try**.
- Si une erreur se produit dans le bloc **Try**, le bloc **Catch** est alors exécuté.
- Quoi qu'il arrive, le bloc **Finally** sera exécuté.

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 n'a toujours pas vu comment ajouter une image, un son, une vidéo... Bref, 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 une chaîne de caractères que l'on veut facilement modifiable. Ce sont des données *statiques* (au même titre que les constantes) qui sont intégrées à l'exécutable ou aux DLL lors de la compilation. Donc si vous insérez toutes vos images, vidéos, etc., en tant que ressources, l'utilisateur ne verra pas un dossier à rallonge avec toutes les images utilisées dans votre programme, elles seront intégrées dans l'exécutable, 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.

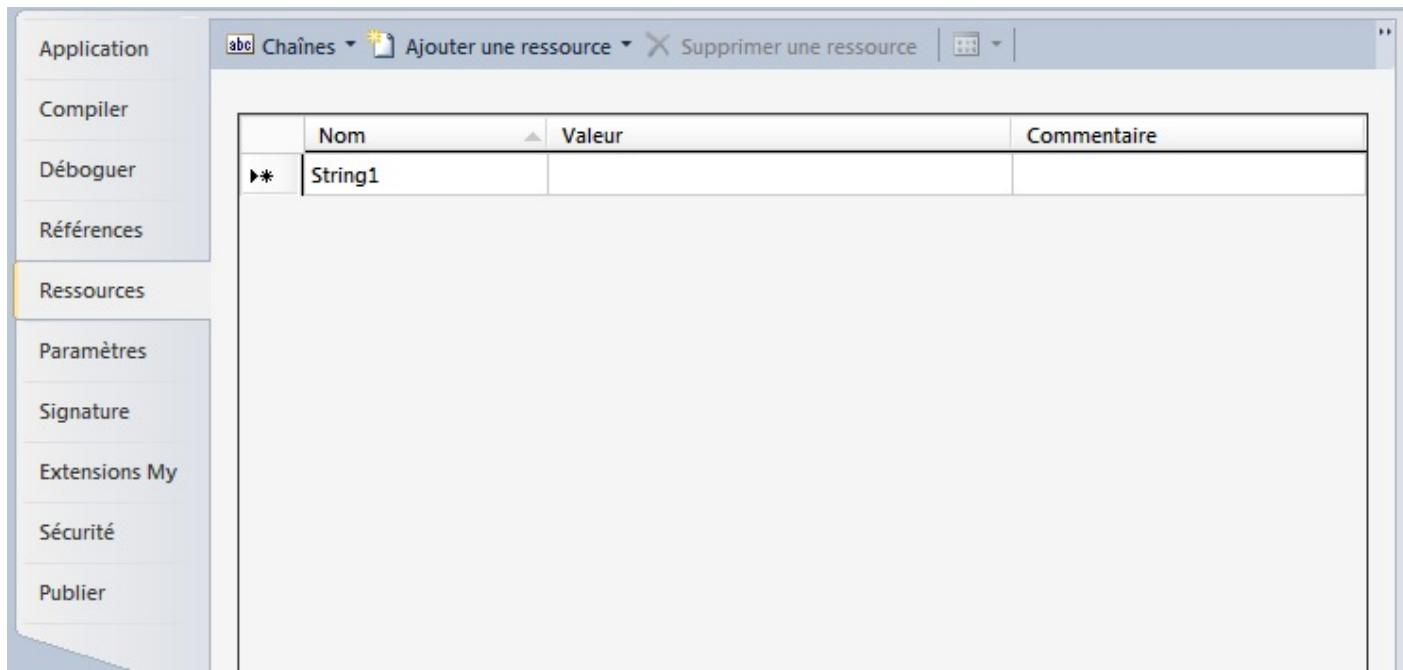
L'utilisation de ressources est habituellement une tâche fastidieuse : compilation et intégration de ses ressources, puis la récupération... Bref, les programmeurs hésitent parfois à les utiliser.

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 sûrement déjà vu l'onglet Ressources lorsque vous vous situez dans la fenêtre de configuration de votre projet (voir figure suivante).



Onglets « Ressources »

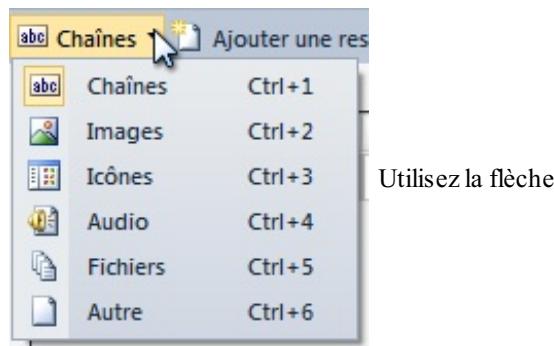
C'est là que l'on va se rendre pour ajouter nos ressources, les éditer, etc.

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 *String*. 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, vidéos...), comme à

la figure suivante.



Vos ressources sont bien organisées et classées.

Ajoutons nos ressources

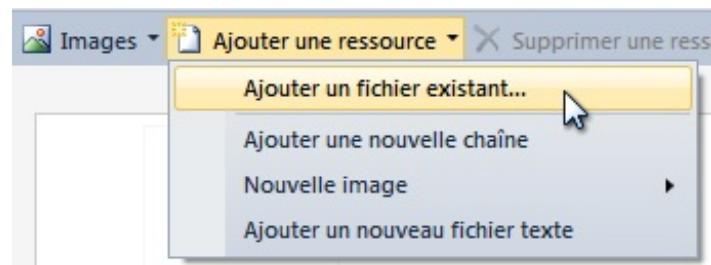
Nous allons avoir deux grandes manières d'ajouter nos ressources.

Prenons les images comme exemple.

La première manière consiste à ajouter un fichier contenant déjà une image.

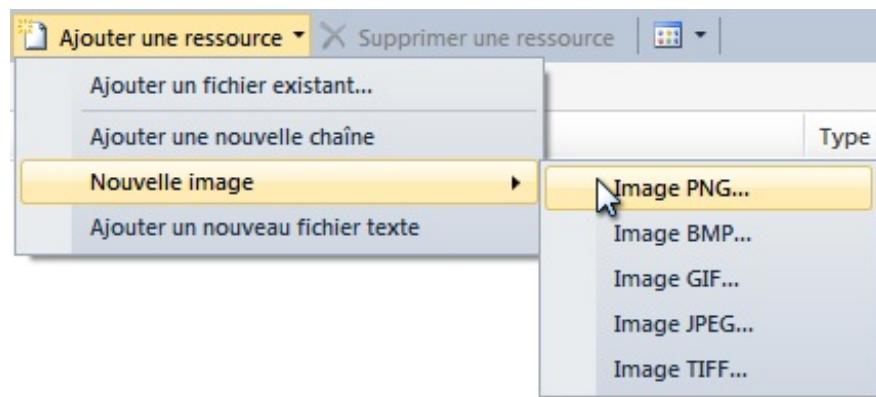
Vous vous souvenez sûrement du TP sur le navigateur web, à la fin de ce TP une partie « design » nous apprenait à utiliser les images en tant que ressources « externes », cette fois nous allons utiliser les images en « interne ».

Donc placez-vous dans la fenêtre Ressources dédiée aux images, cliquez sur Ajouter une ressource, puis Ajouter un fichier existant... (voir figure suivante).



Sélectionnez ensuite l'image souhaitée. Vous voyez qu'elle s'ajoute directement et qu'un aperçu est disponible.

La seconde manière d'ajouter un fichier ressource est de le créer directement, comme à la figure suivante.



Rendez-vous sur Nouvelle image, sélectionnez 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-le ; vous voici avec une ressource rapidement créée.

Vous pouvez faire de même avec un fichier texte (très pratique lorsque vous voulez un fichier de configuration caché). Attention toutefois avec cette méthode, il faut que l'utilisateur lance le programme avec les droits en écriture (le plus souvent

administrateur) pour avoir accéder 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 créé un nouveau programme de test, vous pouvez faire de même.

J'ai ajouté deux ressources : une étoile et la chaîne 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 n'ai pas parlé du mot `My` mais c'est une des choses que vous allez pouvoir découvrir par vous-même. `My` va permettre d'accéder directement aux fonctionnalités de votre ordinateur. C'est avec `My` que nous accéderons à l'audio de votre PC, à 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 ajoutées précédemment.

Pour toutes ces choses spécifiques, vous pouvez vous laisser guider par l'assistant (inscrivez `My` . et regardez la liste de possibilités). Vous allez pouvoir développer des programmes interagissant avec votre machine. Pour le moment, concentrons-nous sur l'accès aux ressources.

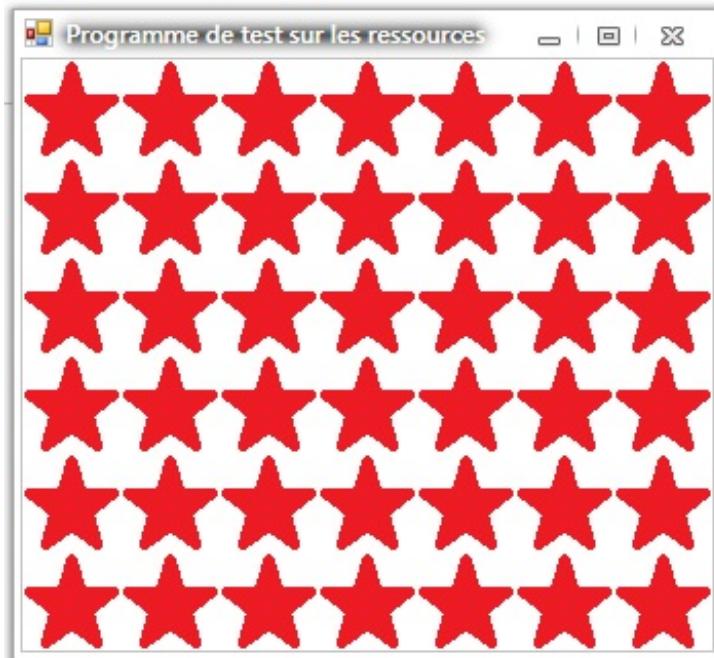
Pour y accéder, c'est plus qu'enfantin, il vous suffit d'inscrire `My.Resources` . pour que l'assistant vous affiche les différents noms de vos ressources. Elles sont directement accessibles comme des propriétés.

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, comme le montre la figure suivante. Mais notre utilisation des ressources est parfaitement fonctionnelle.



Pour l'utilisation des sons et des vidéos, nous aborderons leur utilisation ultérieurement, mais vous savez quand même les

ajouter à votre projet.

Le registre

Bon, les ressources incorporées dans l'exécutable, c'est bien beau, mais pour certains programmes il serait plus utile de placer des valeurs (comme de la configuration) dans le registre.

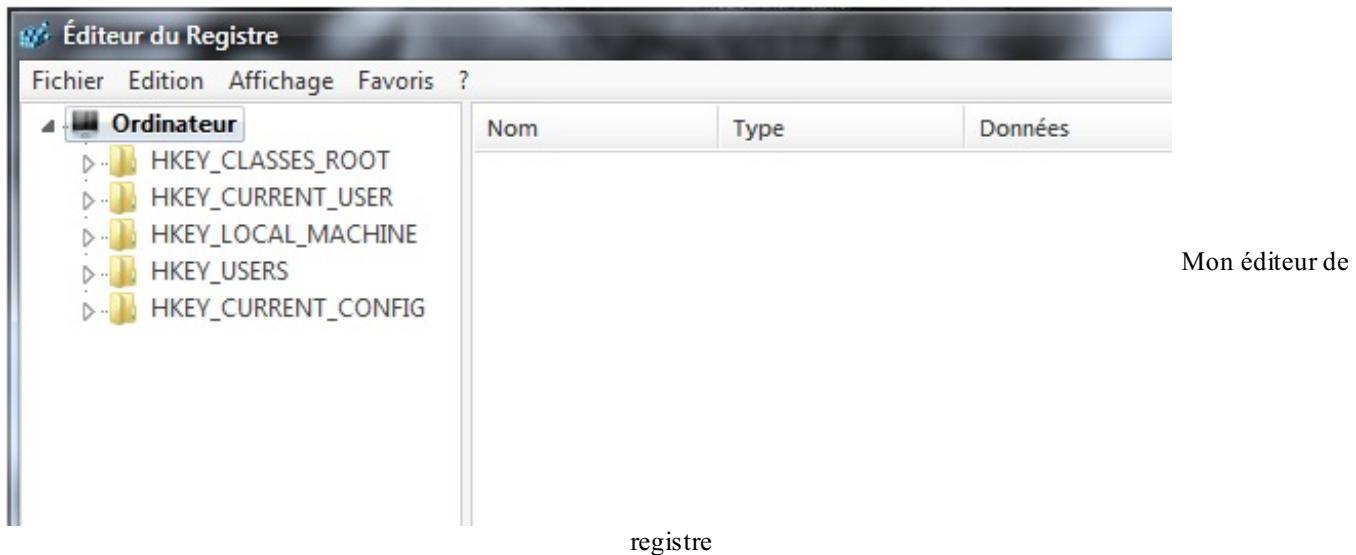


C'est quoi ça, le registre ?

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.

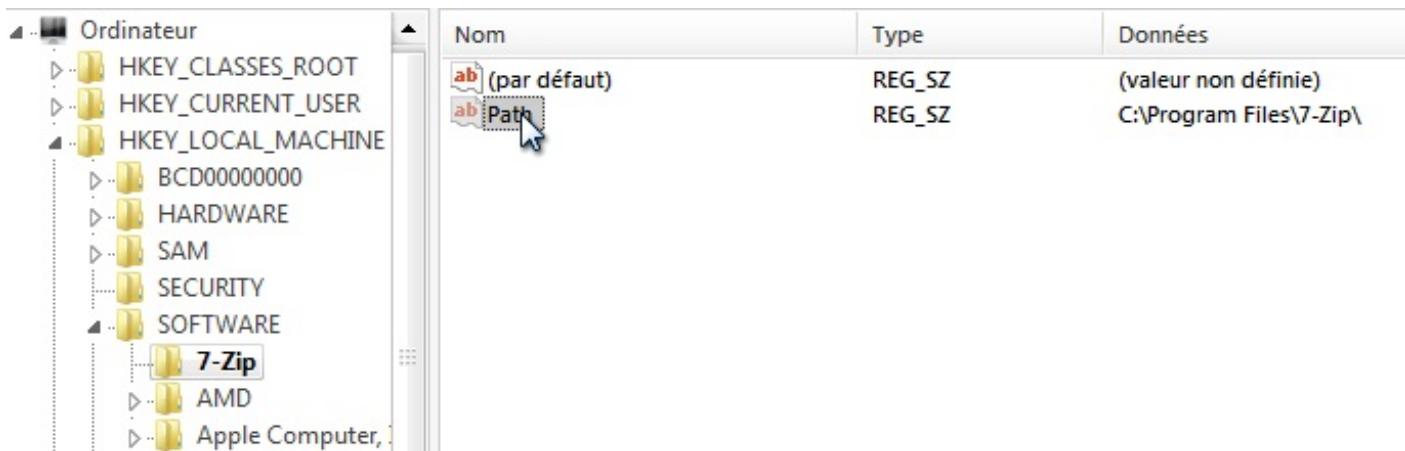
Nous allons nous en servir pour stocker nos informations de configuration. Tout d'abord, pour accéder à votre éditeur de registre Windows, écrivez `regedit` dans le menu Démarrer > Exécuter.

La figure suivante vous montre à quoi ressemble mon éditeur de registre.



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 par des Valeurs, comme le montre la figure suivante.



La valeur « Path » est présente dans la clé 7-Zip, ce qu'elle contient est le chemin 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 enregistrées 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 ».

Commençons par l'écriture :

Code : VB.NET

```
SaveSetting("Ressources", "Configuration", "Config1", "10")
```

Le premier argument permet de spécifier le nom du programme, c'est-à-dire la clé qui sera créée dans VB and VBA Program Settings. Le second paramètre permet de spécifier la section, ici j'utilise Configuration, puis vient le nom de la valeur, puis son contenu.

Ce qui nous créera la clé suivante :

```
HKEY_CURRENT_USER\Software\VB and VBA Program Settings\Ressources\Configuration
```

Avec à l'intérieur la valeur Config1 = 10. Et pour la récupérer :

Code : VB.NET

```
Valeur = GetSetting("Ressources", "Configuration", "Config1")
```

Les arguments correspondent à la fonction précédente. Avec possibilité de spécifier un quatrième 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 deux 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 de 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 LocalMachine.

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 ;
- Écrire 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, puis 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 Cle 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) à 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 milieu de dossiers.

Récapitulatif

Je récapitule les fonctions (à 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 à retrouver, renvoie **Nothing** si la valeur n'existe pas.

Pour conclure ce chapitre, je tiens 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 intéressant.

Les ressources vont être très utiles pour stocker les images nécessaires au design de vos applications, pour modifier facilement les chaînes de caractères ou valeurs (même avec 50 fenêtres, les configurations sont réunies au même endroit) et autres petits bons 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 nécessaires, les ressources sont en effet très utiles, mais peuvent vite devenir un gros inconvénient.

- Les ressources nous permettent de stocker des informations que l'on souhaite réutiliser.
- On les utilise pour stocker les icônes et les images de notre application.
- Le registre nous permettra de sauvegarder les configurations de l'utilisateur. Attention cependant aux logiciels nettoyant le registre.

Diffuser mon application

Vous savez comment créer votre programme, que diriez-vous désormais de le diffuser pour que d'autres personnes puissent l'utiliser ? C'est ici que ça se passe. 😊

Définition de l'assembly

« Assembly », traduisible pas assemblage/montage en français est la base du concept de diffusion d'une application VB .NET.

Cela va nous simplifier énormément la vie en ce qui concerne la diffusion de notre application. Plus particulièrement lorsque c'est une application que l'on souhaite mettre à jour. Ce fichier va contenir les informations relatives aux fichiers que nous allons générer et diffuser.

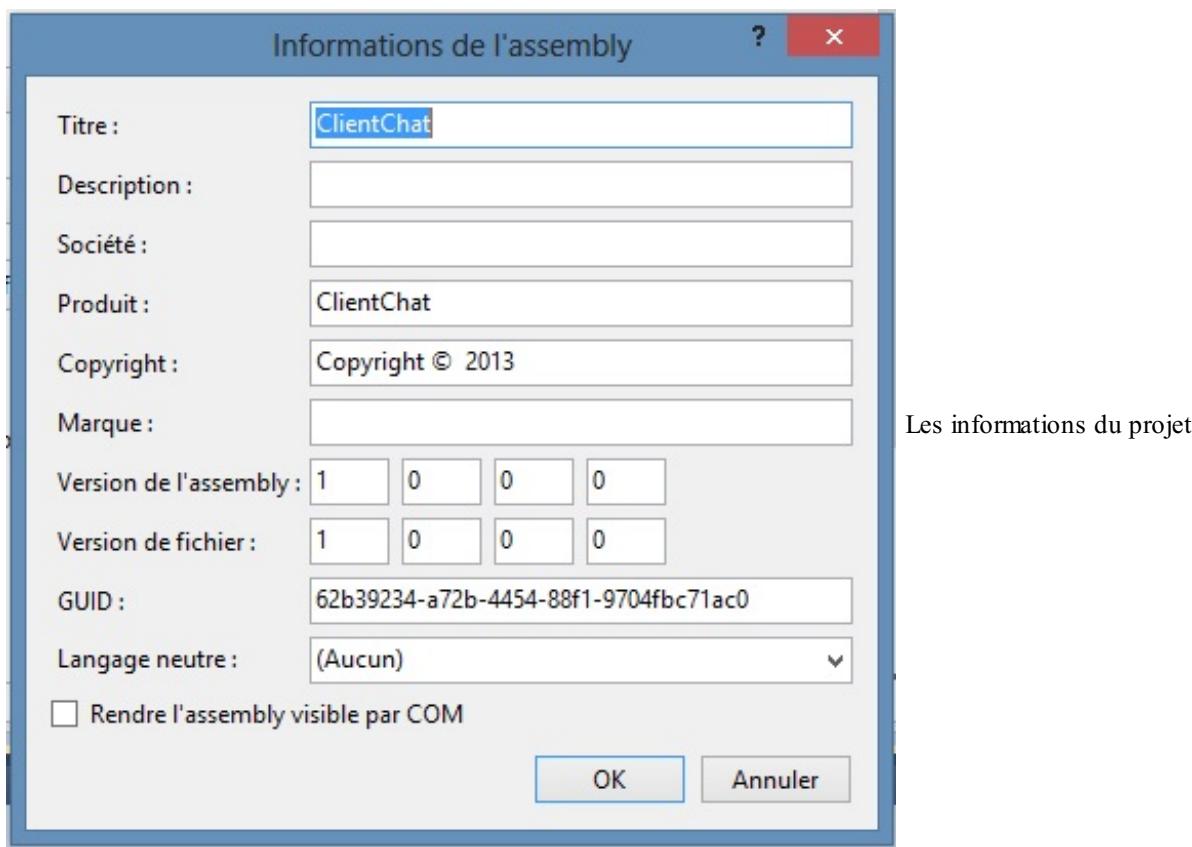


Bon, assez parlé, où est-ce qu'on le trouve ?

Tout d'abord, rendez-vous dans la fenêtre contenant les fichiers de votre projet : *l'explorateur de solutions*. Double-cliquez alors sur *My Project*, cela aura pour effet d'ouvrir les paramètres de votre projet (j'espère que vous les avez déjà vus au moins une fois 😊).

Puis dans l'onglet Application (celui déjà ouvert normalement), cliquez sur le bouton Informations de l'assembly.

Voilà, vous avez ouvert les informations inhérentes au programme que vous venez de développer (voir la figure suivante). Spécifiez son titre, son copyright et même son numéro de version (à incrémenter lorsque vous le mettez à jour).



Votre fichier de sortie est maintenant certifié avec vos informations.

Debug et Release

Vous vous souvenez sûrement, pour le moment nous avons uniquement exécuté le programme en mode Debug. Ce mode compile le programme avec des informations supplémentaires de façon à rendre possible son débogage durant son développement.

Nous en avons terminé avec cette étape, nous souhaitons le diffuser. Pour cela, il faut passer en mode Release.

Si vous utilisez la version Express de Visual Basic dont j'ai présenté l'installation au début de ce cours, *no panic !* Tout est automatique. En effet, lorsque vous démarrez le programme avec la petite flèche verte de la barre d'outils, l'IDE sait qu'il doit

compiler pour du débogage et est donc en mode Debug. Lorsque vous effectuez un clic droit sur le projet et sélectionnez Régénérer, le programme va compiler en mode Release. Si vous utilisez une version complète de Visual Studio, cette étape est plus délicate. Il faut aller dans le gestionnaire de configurations et modifier la liste déroulante pour passer en Release.

Vous savez ce qu'il vous reste à faire : une régénération de votre programme !

Dès la fin de cette étape vous pouvez récupérer un exécutable et les fichiers nécessaires dans C:\Users\xxxx\Visual Studio 2010\Projects\MonProjet\Release.

Apprenons maintenant comment créer un installeur qui va nous mâcher tout le travail.

La publication

Maintenant, place à la publication.

Il y a deux grandes manières de diffuser votre application :

- Vous avez un serveur IIS avec Frontpage de disponible : préférez une diffusion via le web.
- Vous avez uniquement un serveur FTP ou souhaitez le diffuser par support amovible (CD, clé USB...) : préférez une diffusion via un installeur basique.

Le grand avantage de la diffusion via serveur IIS sera la vérification des mises à jour et la possibilité de spécifier l'application comme utilisable uniquement en ligne.

Utilisable uniquement en ligne signifie que l'application sera téléchargée avant chaque utilisation et sera supprimée une fois l'application fermée. Très utile si vous la mettez constamment à jour.

Vous avez aussi la possibilité de vérifier les mises à jour régulièrement si l'application est en ligne.

Bref, je résume en disant que si vous souhaitez une application qui va durer dans le temps, pour diffuser à grande échelle et tout le temps à jour, préférez la diffusion par IIS.

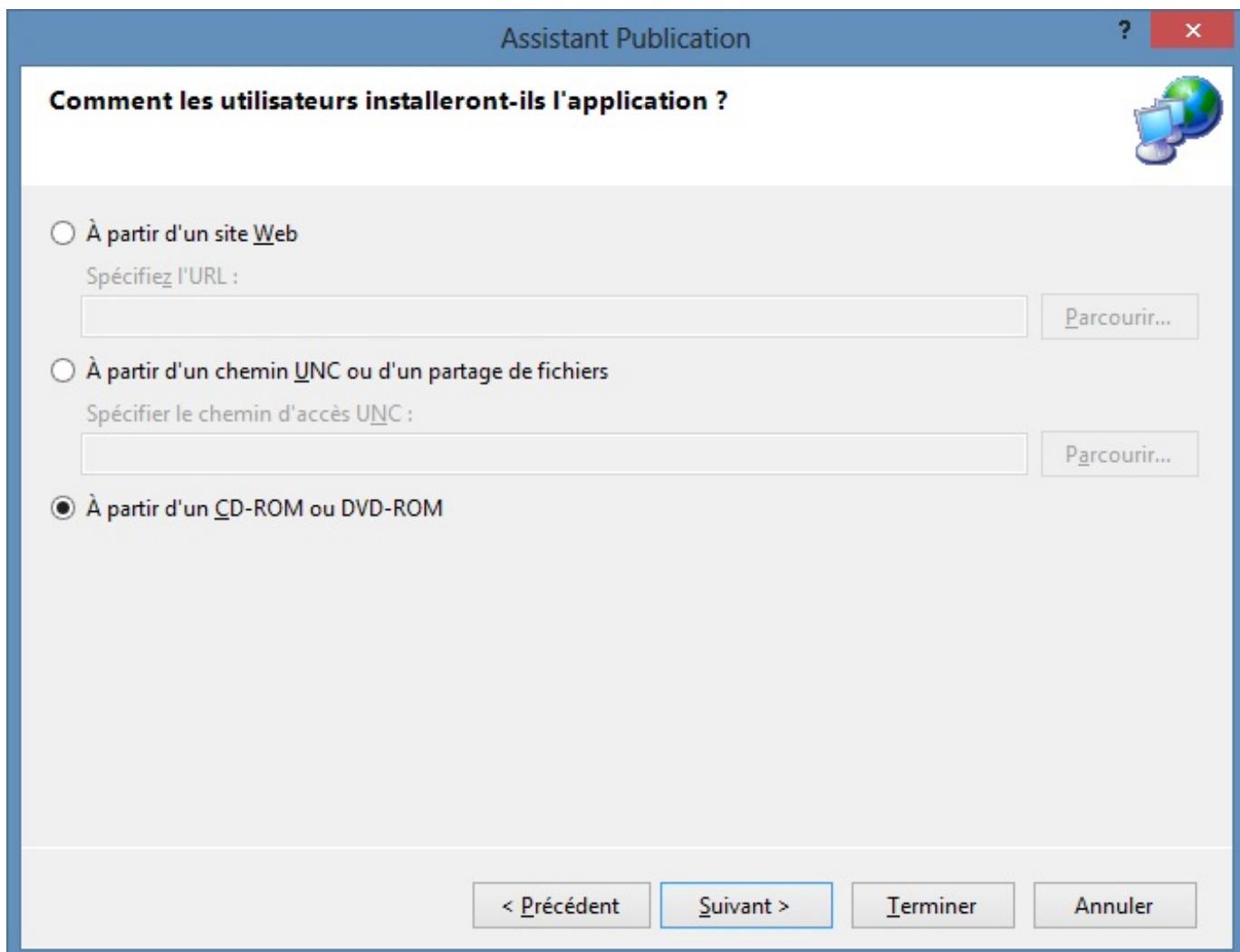
Dans tous les cas, suivez le guide pour la publication.

Démarrer l'assistant de publication

Rendez-vous encore une fois dans les paramètres du projet (double-clic sur My Project). Cette fois, cliquez sur l'onglet Publier.

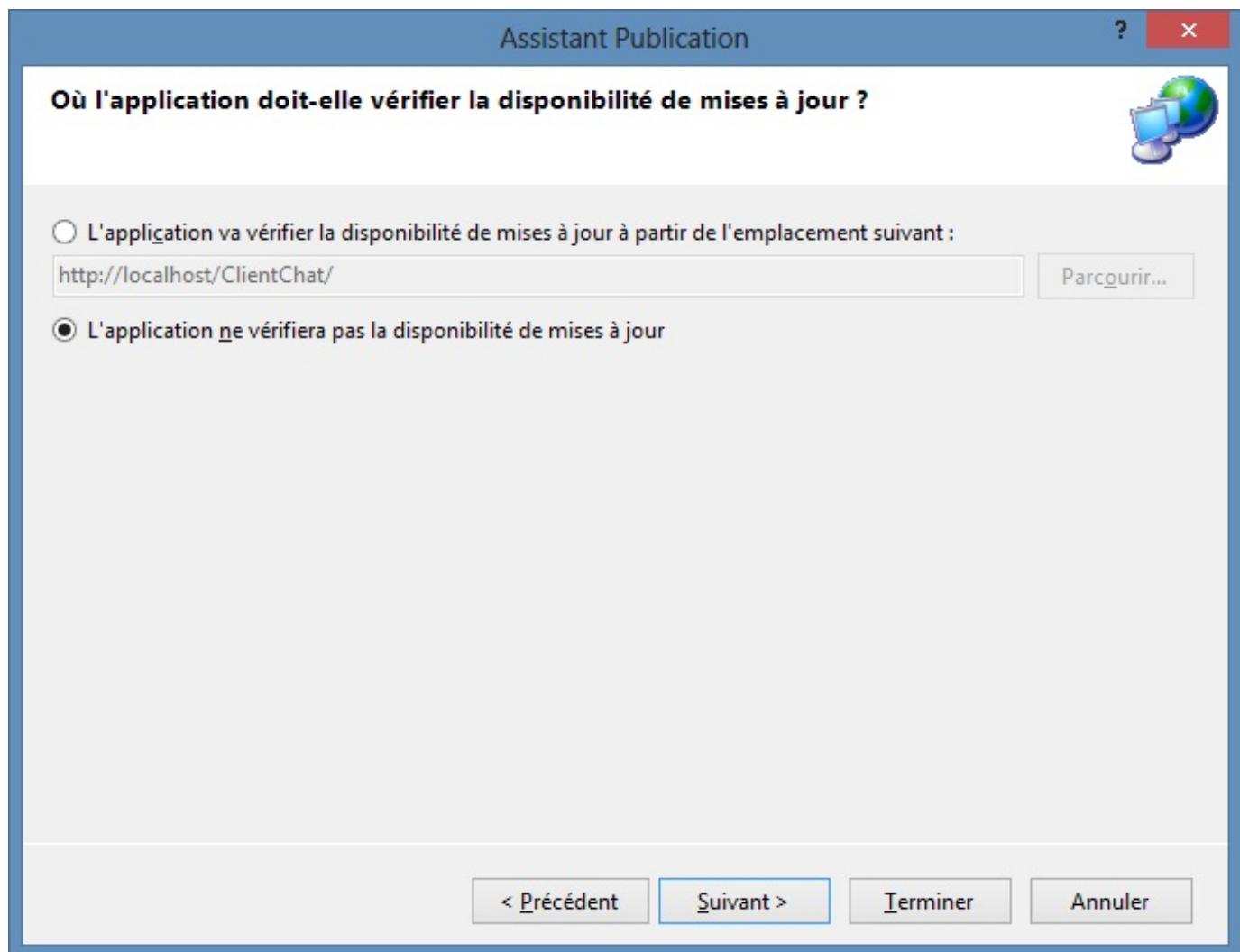
Lancez l'assistant de publication en cliquant sur le bouton du même nom. La première fenêtre est le dossier vers lequel sera publiée l'application ; ici elle sera dans le même dossier que le projet et dans un sous-dossier nommé publish. À vous de modifier le chemin pour l'enregistrer où vous le souhaitez.

Seconde étape : le moyen de diffusion (voir figure suivante). La première option est pour ceux diffusant sur IIS, pour les utilisateurs basiques, c'est la dernière option : À partir d'un CD-ROM ou DVD-ROM qu'il faut choisir.



partir de quoi voulez-vous publier votre application ?

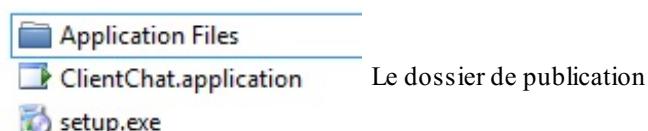
Finalement, comme le montre la figure suivante, il faut spécifier si oui ou non vous souhaitez maintenir à jour votre application (sachant qu'il faut un serveur où stocker les fichiers). Encore une fois, l'option de mise à jour automatique est réservée aux utilisateurs de IIS.



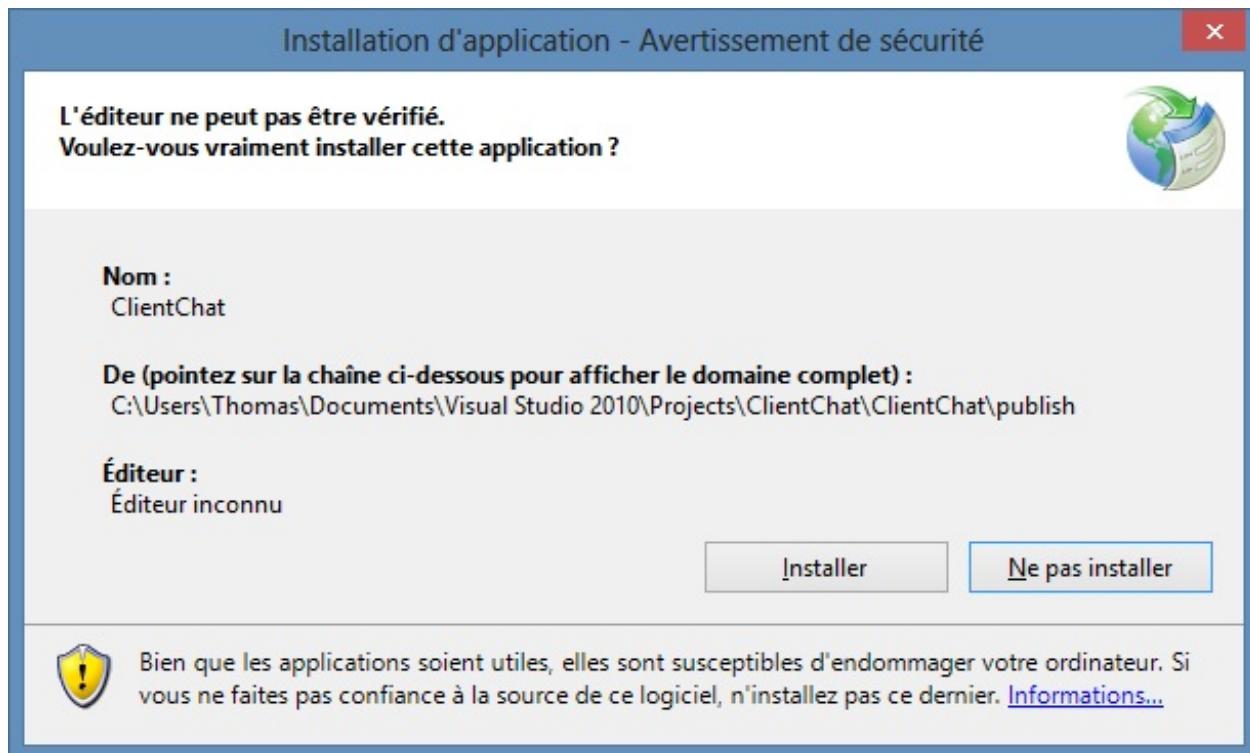
Voulez-vous mettre à jour votre application ?

Et voilà, tout est renseigné, vous pouvez terminer l'assistant. Voyons tout de suite le résultat.

Repérez le dossier de publication (voir figure suivante).

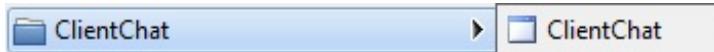


Si je lance `setup.exe` (comme toute installation qui se respecte), il me demande une confirmation avant d'installer l'application, comme à la figure suivante.



sûrs de vouloir installer l'application ?

Si vous acceptez, l'application se lance et est désormais accessible depuis le menu Démarrer, comme à la figure suivante.



L'application est accessible dans le menu Démarrer

Libre à vous de mettre une petite icône. 😊



Mais on n'aurait pas pu mettre le .exe du dossier Release dans un .zip et le diffuser de la même manière ?

Alors deux raisons pour lesquelles on ne fait pas ça :

1. Notre méthode permet de signer numériquement l'installateur et certifie à l'ordinateur qui va l'exécuter qu'il provient bien de chez vous. De plus, les mises à jour sont faciles.
2. Notre méthode va télécharger tous les packages nécessaires à l'exécution de votre programme. Je vous rappelle que l'on a utilisé le framework .NET pour développer en VB.NET. Ce framework doit donc lui aussi être inclus sur les machines qui exécuteront l'application. Cependant, tout le monde ne l'a pas ou n'a pas la bonne version (ici la version 4.0). Ce programme se charge de télécharger et d'installer automatiquement le framework s'il manque sur l'ordinateur cible.

Voilà, notre programme est publié ! Vous savez désormais tout ce qu'il y a à savoir pour développer de A à Z un programme fonctionnel et pour finalement le diffuser à d'autres personnes !



J'ai été vague sur la diffusion en utilisant un serveur IIS, je vous renvoie vers d'autres cours présents sur Internet, mais sachez que cela s'adresse à des personnes ayant des besoins très spécifiques. La compression des fichiers d'installation (setup.exe, etc.) et l'upload sur un FTP sera un très bon moyen de diffuser votre application.

- La diffusion permet de partager son programme, et à d'autres utilisateurs de facilement l'installer.
- Il est préférable d'avoir un serveur IIS pour profiter de tous les moyens de diffusion.

Voilà, c'est la fin de ce cours !

Quelle émotion de vous voir voler de vos propres ailes ! Regardez-vous, au début, ne sachant pas utiliser une variable. Et

désormais capables de communiquer avec une base de données et un autre programme. J'espère avoir été un bon professeur, je vous souhaite bonne chance dans le monde de la programmation. 😊

Bien sûr, si vous aimez le VB .NET, cherchez d'autres cours plus poussés sur le Net, pour des notions toujours plus avancées ! Les possibilités de ce langage sont immenses !

Si vous avez des problèmes de compréhension, de fonctionnement, n'hésitez pas à poster vos messages sur [le forum VB. NET](#), des Zéros compétents sont là pour vous aider (faites une petite recherche avant pour vérifier que cette question n'a pas déjà une réponse).

Voilà tout, adieu mes Zéros.