Cours complet en Visual Basic.net (Framework 1.1 et 2.0)

par Aspic (autres articles)

Date de publication : 06/10/2007

Dernière mise à jour : 06/10/2007

Cet article constitue un cours complet en vb.net (framework 1.1 et 2.0). Vous trouverez les bases nécessaires pour commencer la programmation en vb.net. Ce cours est avant tout destiné aux débutants mais les amateurs de vb.net sont les bienvenues.

I - Introduction

I-A - Etes vous apte à commencer à programmer ?

Considérant que vous savez normalement déjà allumer un PC et l'éteindre, utiliser une souris et un clavier de préférence, et que vous êtes motivez pour écrire des programmes en Visual Basic.NET, que ce soit pour le plaisir ou le travail, vous êtes donc prêt à débuter la programmation en VB.NET !

En plus d'un PC, il vous faut un exemplaire de Visual studio.NET. Dans le cas où vous douteriez de vos capacités à apprendre la programmation, souvenez-vous d'Albert Einstein (le célèbre physicien qui a inventé la théorie de la relativité). Un de ses instituteurs avait déclaré qu'il apprenait si lentement qu'il était sans doute attardé !

Peut-être en réponse à cela, Albert Einstein a dit un jour : " L'imagination est plus importante que le savoir ". Certaines personnes soutiennent qu'il aurait aussi dit : " Si mon instituteur est si intelligent, où est son prix Nobel ? " (Ce qui reste à vérifier !!)

Avec de l'imagination, un ordinateur et VS.NET (Visual Studio qui est l'outil de développement), vous êtes fin prêt pour le grand saut dans le monde de la programmation#

I-B - A la découverte de Visual Basic

Le Visual Basic.NET est la dernière version de Visual Basic. Pour comprendre les changements que Microsoft a apportés au langage Visual Basic dans cette version, il faut comprendre ce que Microsoft tente d'accomplir avec le concept .NET.

Il existe des centaines de langages de programmation différents pour résoudre différentes taches. Un gros problème est que ces langages de programmation ne sont pas conçus pour travailler ensemble.

Même les langages de programmation d'un même éditeur, comme Microsoft, ont des problèmes à travailler ensemble. Les versions précédentes de Visual C++++ et Visual Basic enregistrent les données, telles que chaînes et nombres, de façon différente. Du fait qu'il est très fastidieux de découvrir tous les moyens particuliers employés par chaque langage de programmation pour enregistrer et manipuler les chaînes et les nombres, la plupart des programmeurs écrivent leurs programmes à l'aide d'un seul langage de programmation, même lorsqu'un second langage convient mieux pour résoudre un problème particulier. Par exemple, le Visual Basic est très utile pour faire des applications de bureautique, le C++ est plus puissant que le Basic et donc utiliser pour les très gros programmes. Quant au Java, il est utilisé dans la conception de sites Internet sophistiqués.

Microsoft a donc développé quelque chose baptisé " .NET Framework ", qui agit comme couche intermédiaire entre le système d'exploitation (Windows) et tous les programmes que vous écrivez. Cela résout deux problèmes majeurs.

Le premier problème résolu par le .NET est la possibilité pour différents langages de coopérer entre eux. Au lieu d'octroyer à chaque langage un accès direct au système d'exploitation de l'ordinateur, l'infrastructure .NET Framework

force les langages qui ont été conçus pour travailler avec .NET (comme VB.NET) à stocker leurs chaînes et nombres exactement de la même manière. De la sorte, vous pouvez écrire un programme utilisant différents langages sans vous préoccuper du fait qu'un langage stocke et manipule les données de façon différentes d'un autre langage de programmation.

Le second problème résolu par le .NET Framework concerne la distribution du programme. Actuellement, la plupart des utilisateurs exécutent des programmes stockés sur leurs disques durs# Bref passons enfin aux choses plus intéressantes !

I-C - Les inconvénients du Framework

Tout ordinateur se plante périodiquement et la mise en place de nouvelles technologies résout rarement les faiblesses des technologies déjà mises en place. Aussi promoteur que soit .NET, ne vous laissez pas abuser par le marketing de Microsoft#

Le .NET Framework, étant un logiciel est sujet à toutes sortes de bugs qui font de l'usage de l'ordinateur une expérience frustrante. D'autant plus que cette technologie n'est disponible que pour les versions récentes de Windows (XP et suivant). Si vous souhaitez écrire des programmes pour Linux ou Windows versions antérieures, vous devez passer par la version 6.0 de Visual Basic.

Bon c'est fini pour l'introduction, passons au codage pur ! Mais avant tout, regardons d'un peu plus près, les structures utilisées dans la technologie .NET.

II - Structure du programme

II-A - Présentation des objets

Déjà il faut savoir ce qu'est un objet en programmation. Prenons un exemple de la vie courante :

Une voiture est un objet. Jusqu'à présent, rien de bien compliqué. Cette voiture a des phares, un volant, des pédales, un capot, une couleur, une marque# Ce sont des propriétés de la voiture. De plus, toutes ses propriétés ont des valeurs. Par exemple, la couleur de la voiture peut être bleue ou rouge (bleu et rouge sont les valeurs de la propriété " couleur "). En programmation c'est un peu pareil : La voiture fait partie d'un grand ensemble qui est " les voitures ". " Les voitures " représentent la " class " en programmation. L'objet est " ma voiture ". Un objet est créé selon un modèle qu'on appelle une Class.

```
'exemple pour créer 'MaVoiture' à partir de la class " LesVoitures "
Dim MaVoiture As New LesVoitures
```

MaVoiture a été instanciée (par le mot clé 'new'). Utilisons quelques propriétés de MaVoiture :

```
'propriétés de MaVoiture
MaVoiture.Couleur = " Bleue " 'la couleur est bleue
MaVoiture.Phares = 2 'elle possède 2 phares
MaVoiture.Phare.Avant.Allume = True 'le phare avant est allumé (True)
MaVoiture.Phare.Arriere.Allume = False 'le phare arrière est éteint (False)
```

Les chaînes de caractères (texte comme 'Bleue') sont entre guillemets. Les nombres ne sont pas entre guillemets.

MaVoiture possède aussi des méthodes. Par exemple, elle roule : c'est une méthode.

```
'Méthode de MaVoiture
MaVoiture.Roule() ' la voiture roule
```

Une méthode est caractérisée par des parenthèses. Mais une méthode peut demander des paramètres. Par exemple, à quelle vitesse doit rouler MaVoiture ? Un paramètre est un renseignement envoyé à la méthode.

```
'Méthode avec paramètre
MaVoiture.Roule(100) 'MaVoiture va rouler à 100 Km/h
```

On peut aussi fournir plusieurs paramètres :

```
'Méthodes avec plusieurs paramètres
MaVoiture.Roule(100, avant) 'Ici elle va rouler à 100 Km/h en avant !
```

Enfin, un objet (ici MaVoiture), peut avoir des événements. Dans notre exemple, MaVoiture peut tomber en panne, démarrer ou s'arrêter. 'TomberEnPanne', 'Demarrer' ou 'Arrêter' sont des événements.

```
'Evenement de MaVoiture
MaVoiture.Roule(100, avant) 'Ici elle va rouler à 100 Km/h en avant !
'MaVoiture_Demarrer va donc se déclencher à cause de l'appel de la méthode 'Roule'
```

Si on récapitule, un objet est donc dérivé d'une Class. Un objet possède des propriétés, des méthodes et des événements (et d'autres choses#)

II-B - Instructions et procédures

Une instruction permet d'effectuer une opération, une déclaration, une définition.

```
Dim A As Integer ' est une instruction (de déclaration)
A=1 ' est aussi une instruction qui effectue une opération.
```

C'est habituellement une 'ligne de code exécutable'. Une instruction est exécutée lorsque le programme marche. Plusieurs instructions peuvent se suivre sur une même ligne, séparées par ':'

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

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

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

Equivaut à Dim B As String = "Bonjour monsieur ": C= "le professeur" Quand un programme tourne, les instructions sont effectuées ligne après ligne.

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

La ligne 1 est exécutée puis la ligne 2 puis la 3, la 4... Bien que l'on puisse avoir des numéros de ligne, ils ne sont plus utilisés actuellement et ils sont non visibles : Pour mettre des commentaires dans un programme, on le fait précéder d'une apostrophe. À la différence d'une instruction, le commentaire ne sera pas exécuté. 'Ceci est un commentaire, ce n'est pas une instruction. Une procédure est un ensemble d'instructions qui effectue une fonction précise. En Visual Basic.NET, il existe deux sortes de procédures : les 'sub' et les 'function' . Les 'sub' commencent par le mot clé 'sub' puis finissent par 'end sub' et ne retournent aucune valeur contrairement aux fonctions qui débutent par 'function' puis finissent par 'end function' et qui retournent une valeur précise.

```
'Exemple de sub
Sub DemarrerProgramme()
     'la procédure ne retournera aucune valeur.
End Sub

'Exemple de fonction
Function CalculerPi()
     'la fonction va retourner la valeur de PI après une série de calculs#
End Function
```

Pour appeler une fonction ou un sub on utilise le mot clé " Call ". Il n'est pas obligatoire.

```
'Exemple d'appel de fonction ou sub
Call DemarrerProgramme
CalculerPi()
'Ces deux appels sont identiques
```

II-C - Les modules

Les modules sont des feuilles (ou formulaires) qui contiennent plusieurs procédures et/ou fonctions. Un module commence toujours par le mot clé 'Module' et finit par 'End Module'

```
'Exemple de module
Module NomDeMonModule
     Sub Test1()
          'Procédure 1
     End Sub

     Function Test_1()
          'Fonction 1
     End Function

     Function Test_2()
          'Fonction 2
     End Function

     Sub Test2()
          'Procédure 2
     End Sub
End Module
```

*Les '**sub**' et '**fonctions**' ne peuvent pas avoir les mêmes noms.*

Récapitulatif : Une instruction est une ligne de code qui va être exécutée par le programme. Une fonction retourne une valeur contrairement à la procédure qui n'en retourne pas. Un module n'est en fait qu'une feuille contenant de nombreuses procédures et fonctions.

III - Le langage Visual Basic

III-A - Les variables

En programmation, une des choses essentielle est l'utilisation des variables. Une variables est un sorte de boite que va contenir une valeur. Par exemple, je mets une chaussure dans une boite. La 'boite' représente la variable et la valeur est la 'chaussure'. L'intérêt principal des variables est leur utilisation ultérieure. En effet, on peut utiliser une variable à n'importe quel moment dans le programme en fonction de la portée de cette variable. (Voir chapitre sur la portée des variables)

III-A-1 - Les variables

En ce qui concerne les noms des variables, on peut utiliser des majuscules, minuscules et chiffres. Les espaces et autres caractères spéciaux ne sont pas acceptés (accents, @, #).

Le nom de la variable ne peut pas commencer par un chiffre !

```
'Exemples de noms de variables
MaVariable 'est acceptée
COOL 'est acceptée
Je_Suis_1_Variable 'est aussi acceptée
2Variable 'est refusée
2_Variable 'est également refusée
The Variable 'est réfusée
Différent 'est aussi refusée
```

Je vous conseille de donner des noms explicites aux variables pour ne pas vous y perdre dans le programme. De plus, mettez la première lettre de votre variable en majuscule. Si votre variable est composée de plusieurs noms, séparer les par des " _ " ou mettez une majuscule à chaque nouveau mot.

```
'Exemple :
Variable_pour_calculer_pi
VariablePourCalculerPi 'ou bien
```

Maintenant pour utiliser une variable, il faut avant tout la déclarée et l'instancier dans certains cas. Pour cela on utilise le mot clé " Dim " pour la déclarée et " new " pour 'instancier. De plus, il vaut lui assigner un type (Voir plus loin dans le chapitre)

```
'Exemples de déclarations de variables
Dim Ma_Variable As String
Dim Nombre As Integer
Dim Compteur As Integer

'Instancier des variables
Dim Msn As New Msn_Protocol 'Msn_Protocol est une class
Dim Test As New Class
```

On peut aussi instancier les variables précédentes comme cela :

```
'Exemple avec les variables précédentes
Dim Ma_Variable As String = " toto "
Dim Nombre As Integer = 45
Dim Compteur As Integer = 1000
```

Une variable peut aussi être instanciée par une fonction :

```
'Exemple de variable instanciée par une fonction
Dim Pi As Decimal = Fonction_pour_calculer_pi()
```

III-A-2 - Les différents types de variables

Il existe une multitude de type de variables. A notre niveau, on en utilisera essentiellement 4 : String, Integer, Boolean et Byte

Regroupons tous les types dans un tableau

Type de variable	Mémoire occupée	Plage de nombres acceptés	Utilisation principale
Boolean	2 octets	True (0) ou False (1)	Pour les conditions
Byte	1 octet	De 0 à 255	Pour les fichiers
Char	2 octets	De 0 à 65 535	Pour les caractères alphanumériques
Date	8 octets	Dates entre le 1 Janvier 0001 et le 31 Décembre 9999	Pour les dates
Decimal	16 octets	+ / - 79 228 162 514 264 337 593 543 950 335 sans séparateur décimal sinon + / - 7.9 228 162# avec 28 chiffres après le séparateur décimal	Pour les nombres décimaux
Double	8 octets	- 1,79769313486231E+308 à - 4.94065645841247E-324 (nombres négatifs) et 4.94065645841247E-324 à 1,79769313486231E+308 (nombres positifs)	Pour les grands nombres à virgules (avec double précision)
Integer	4 octets	De - 2 147 483 648 à 2 147 483 647	Pour les nombres entiers
Long	8 octets	De - 9 223 372 036 854 775 808 à 9 223 372 036 854 775 807	Pour les entiers longs
Short	2 octets	De - 32 768 à 32 767	Pour les entiers courts
Single	4 octets	De - 3.402823E-45 à - 1.401298E+38	Pour les grands nombres à virgules (avec simple précision)
String	Variable	0 à 2 milliards de caractères Unicode	Pour les chaîne de caractères

Utiliser les types qui prennent le moins de mémoire. Par exemple, si vous souhaitez stocker une note dans votre variable, utilisez le type Integer mais n'utilisez pas le type

Long dans la mesure où votre variable sera comprise entre 0 et 20 (ou 40 ou 100 en fonction des examens lol)

Une chaîne de caractères n'est en fait que du texte (" Je suis gentille ") et elle se met TOUJOURS entre guillemets contrairement aux nombres !

*En VB.NET 2005 (dernière version de Microsoft), il existe 4 autres **types de variables** mais cela ne concerne pas notre cours.*

III-A-3 - Convertir des variables

Dans tout langages de programmation, il peut être utile voire même nécessaire de convertir des variables d'un type dans un autre type. Cela permet d'éviter la création de nouvelles variables qui alourdiraient le programme.

Tableau récapitulatif des fonctions permettant de convertir des variables

Fonction de conversion	Convertir vers le type...
CBool	Boolean
CByte	Byte
CChar	Char
CDate	Date
CDbl	Double
CDec	Decimal
CInt	Integer
CLng	Long
CObj	Object
CShort	Short
CSng	Single
CStr	String

```
'Exemples
Dim Pi As Decimal = 3.14 'crée la variable Pi qui vaut 3.14
Dim a As Integer = 15 'crée la variable a qui vaut 15

Dim Pi_Entier As Integer = CInt(Pi) 'retournera 3
Dim a_caractère As String = CStr(a) 'retourner " 15 " 'en chaîne de caractère
```

Dans le langage VB.NET, la virgule n'existe pas. On utilise le point qui remplace la virgule.

Noter que ces fonctions sont construites à partir de C (pour Convert) et de l'abréviation du type.

Pour convertir un Object dans un type (String, Integer...), on utilise la commande 'CType' [CType(a, String) 'a va être convertit en 'String']

III-A-4 - La portée des variables

Une variable peut être utilisée que si elle est visible dans le programme. Pour définir le niveau de visibilité de chaque variable, il existe un mot clé propre à chaque cas.

Tableau pour la portée des variables

Mot clé	Portée de la variable
Public	Partout dans le programme
Private	Partout dans la classe ou le module où elle a été créée
Dim	Seulement dans la procédure ou la fonction créée
Protected	Les membres Protected sont similaires aux Private, mais ils ont une particularité en cas d'héritage. Ce point ne sera pas abordé
Friend	Les membres Friend ne sont accessibles qu'à l'intérieur du projet, et pas par des éléments extérieurs au projet en cours
Shared	Permet de partager une variable. N'est pas indispensable

Ce tableau est aussi valable pour les procédures et les fonctions

```
'Exemple avec des sub et fonctions

Module Test_Portee

     Private Sub Test()

     End Sub

     Protected Sub Azerty(ByVal Argument As String)

     End Sub

     Private Shared Function Calcul(ByVal Nombre As Integer)

     End Function

End Module
```

Cependant, il est déconseillé d'utiliser des variables globales (publics) car ce la est une source importante d'erreurs. Pour affecter une valeur à une variable public, il est préférable de passer par une procédure d'affectation.

```
'Exemple
Module Test
     Public Nombre As Integer

     Public Sub Routine_Qui_Va_Affecter_La_Variable(ByVal Valeur As Integer)
            Nombre = Valeur
     End Sub
End Module

'Quelque part d'autre dans le programme on appel la procédure avec un argument de type Integer
Call Routine_Qui_Va_Affecter_La_Variable(45)
```

```
'Ensuite dans le programme totale, la variable Nombre va valoir 45
MsgBox(Nombre.ToString) 'Affiche dans une boite de dialogue la valeur
```

Bien sûr qu'il est possible directement de faire directement

```
Nombre = 45
```

Mais cela est source d'erreurs... En effet, vous pouvez modifier la variable à tout moment sans même vous en rendre compte ! A utiliser à vos risques et périls

III-B - Les conditions

Les conditions comme les boucles permettent de gérer le déroulement du code. Il existe deux grands types de conditions : celles avec " If - Then " et celles avec " Select Case - End Select "

III-B-1 - Avec "If -Then"

Structure générale :

```
If condition 1 vraie Then
    'exécuter condition 2
Else
    'exécuter condition 3
End If
```

If : Mot clé signifiant " si " et qui ouvre la structure

Then : Mot clé signifiant " puis "

Else : Mot clé signifiant " sinon "

End If : Mot clé permettant de fermer la structure

En français : Si la condition 1 est vraie alors exécuter la condition 2 sinon exécuter la condition 3

```
'Exemple
Dim code As Integer = 1739

If code = 1739 Then
     MsgBox("condition vraie")
Else
     MsgBox("condition fausse")
End If
```

On peut tester une condition fausse et dans ce cas utiliser Not.

```
If Not A=B Then MsgBox("A est différent de B") 'Si A et B sont différent (Not A=B signifie NON
 égaux) afficher "A est différent de B".
```

Il peut y avoir des opérateurs logiques dans la condition:

```
If A=B And C=D Then..    'Si A égal B et si C égal D
```

Des structures If - Then peuvent être imbriquées. C'est-à-dire que l'on peut emboiter plusieurs conditions dans une seule condition :

```
'Exemple
If condition_1 vraie Then
      If condition_2 vraie Then
            MsgBox("Condition_1 Vraie et Condition_2 Vraie")
      Else
            If condition_3 Vraie Then
                  MsgBox("Condition_1 Vraie et Condition_2 Fausse et Condition_3 Vraie")
            End If
      End If
End If
```

Astuce : Pour ne pas vous trompez, utiliser les tabulations et décaler chaque structure (dans l'exemple : Bleue, Violet et Vert) et son code au même niveau. Pour vérifier s'il n'y a pas d'erreur, compter les 'If', il doit y en avoir autant que des 'End If'. Visual Studio vous avertira en cas d'erreur.

Autre syntaxe possible :

```
If Condition1 vraie Then
    ..
ElseIf condition2 Then 'sinon on teste la condition 2
    ..
ElseIf condition3 Then 'la condition 3
    ..
End If
```

III-B-2 - Avec "Select Case - End Select"

Cette structure est très utile pour tester un grand nombre de possibilités en fonction de la valeur d'une expression :

Structure générale :

```
Select Case X
    Case 1 : 'faire Y
    Case 2 : 'faire Z
    Case Else : 'faire A
End Select
```

Select Case : Mot clé permettant d'ouvrir la structure

Case : Représente les différents cas (Case 1 = cas 1, Case 2 = cas 2, Case Else = autre cas...)

End Select : Mot clé permettant de fermer la structure

X : Expression à évaluée (nombre, chaîne de caractères, booléen#)

faire Y : Exécute le cas 1

faire Z : Exécute le cas 2

faire A : Exécute le cas "else"

Prenons un exemple : Un serveur t'envoie un certain code mais tu ne sais pas si c'est le 1,2 ou un autre (comme 1000, 453, 876#) et toi tu veux que s'il t'envoie 1 alors il doit se connecter au client, si il t'envoie 2 alors il doit se déconnecter et s'il t'envoie un autre code alors mettre un message d'erreur# Voilà la structure :

```
Select Case code
    Case 1 : 'se connecter
    Case 2 : 'se déconnecter
    Case Else : 'erreur code non reconnu
End Select
```

La structure précédente est relativement simple mais limitée. On peut aussi utiliser d'autres types d'expressions plus complexes et plus pratiques : Plusieurs expressions peuvent être séparées par des virgules.

Le mot-clé **To** permet de définir les limites d'une plage de valeurs correspondantes pour N.

Le mot-clé **Is** associé à un opérateur de comparaison (=, <> < <=, > ou >=) permet de spécifier une restriction sur les valeurs correspondantes de l'expression. Si le mot-clé Is n'est pas indiqué, il est automatiquement inséré.

Vous pouvez aussi mixer les différents cas précédents.

Enfin, vous pouvez aussi indiquer des plages et des expressions multiples pour des chaînes de caractères. Dans l'exemple suivant, Case correspond aux chaînes qui sont absolument identiques à " ttt ", aux chaînes comprises entre "bbb" et "eee" dans l'ordre alphabétique, ainsi qu'à la valeur de " Nombre " :

```
'Exemple avec les virgules
Select Case N
    Case 8, 9, 10
        'Effectuer le code  si N=8 ou N=9 ou N=10
End Select
'Exemple avec une plage de valeurs
Select Case N
    Case 8 To 20
       'Effectuer le code  si N est dans la plage 8 à 20
End Select
'Exemple avec un opérateur de comparaison
Select Case N
     Case Is >= 15
         'Effectuer le code  si N supérieur ou égal à 15.
End Select
Select Case N
     Case 3 To 6, 7 To 15, 200, 1654, Is > MaxNumber
         'Effectuer le code si N, compris entre 3 et 15, si N = 200 ou N = 1654, est supérieur à
 MaxNumber
End Select
Select Case N
     Case "ttt", "bbb" To "eee", Nombre
          'Effectuer le code si N ="ttt" ou si N est compris entre "bbb" et "eee" (ordre
 alphabétique) ou si N = Nombre
End Select
```

III-C - Les boucles

Les boucles permettent de répéter un bloc d'instruction autant de fois que l'on à indiqué à la boucle. Elles sont très utilisées par exemple pour faire des calculs itératifs (Méthode d'Euler) ou de lister le contenu d'un tableau. Il en existe quatre types majeurs : celles avec " For - Next ", celles avec " Do - Loop ", celles avec " While - End While " et celles avec " For Each - Next "

III-C-1 - Avec "For - Next"

Structure générale :

```
For Variable = Debut To Fin
   'Execute la boucle
Next Variable
```

For : Mot-clé permettant d'ouvrir la boucle.

To : Mot-clé signifiant " jusqu'à "

Next : Mot-clé pour fermer la boucle.

Variable : Variable qui va servir de compteur pour la boucle.

Debut : Début du compteur.

Fin : Fin du compteur.

```
'Exemple de boucle
Dim i As Integer
For i = 3 To 10
   'Exécute la boucle pour i variant de 3 à 10
Next i
```

La variable i va s'incrémenter automatiquement de 1 à la fin de chaque boucle.

On peut aussi définir un pas, le compteur s'incrémente de la valeur du pas à chaque boucle :

```
'Exemple de boucle
Dim i As Integer
For i = 3 To 10 Step 2
    MsgBox(i.ToString) 'affiche 3 puis 5 puis 7 et enfin 9
Next i
```

On peut aussi utiliser un pas négatif mais cela est plus compliqué.

La variable de la boucle peut être déclarée en même temps que la boucle. De plus, la variable après le Next est facultative.

```
'Exemple
For Test As Integer = 0 To 43
    'boucle de 0 à 43
Next
```

Dernière chose, on peut quitter la boucle prématurément :

```
'Exemple de sortie de boucle
For Test As Integer = 0 To 43
    If Test = 3 then Exit For 'Quand Test sera égale à 3 alors on sort#
Next
```

III-C-2 - Avec "Do - Loop"

Structure générale :

```
Do
   'Instructions
Loop Until Condition

Do
   'Instructions
Loop While Condition
```

Do : Mot-clé qui ouvre la boucle.

Loop : Mot-clé qui ferme la boucle.

While : Mot-clé qui signifie " Tant que "

Until : Mot-clé qui signifie " Jusqu'à ce que "

Condition : La condition pour sortir de la boucle.

Il faut préciser après le " Loop " une condition précéder de " While " ou " Until ". Dans le cas contraire, la boucle est sans fin !

```
'Exemple avec Until
Dim i As Integer = 0
Do
   i = i + 1 'incrémente i de 1
   MsgBox(i.ToString) 'affiche la valeur de i
Loop Until i = 10 'Quand i = 10 on sort de la boucle.

'Exemple avec While
Dim Fichier As String
Do
   'Liste les fichiers d'un répertoire
Loop While Fichier <> "NomDeMonFichier " 'Tant que Fichier n'est pas égal à NomDeMonFichier alors
 on boucle sinon on sort.
```

III-C-3 - Avec "While - End While"

Structure générale :

```
While Condition
    'Execute la boucle
End While
```

While : Mot-clé permettant d'ouvrir la boucle.

End While : Mot-clé qui ferme la boucle.

Condition : Condition de la boucle.

```
'Exemple :
Dim Compteur As Integer = 0
While Compteur < 20
   Compteur = Compteur + 1 'incrémente le compteur de 1
   Msgbox(Compteur.ToString) 'affiche le compteur
End While ' Dès que Compteur = 21, on sort de la boucle.
```

III-C-4 - Avec "For Each - Next"

Structure générale :

```
For Each Item In Collection
   'boucle
Next Item
```

For : Mot-clé pour ouvrir la boucle.

Each : Mot-clé signifiant " Chaque ".

In : Mot-clé signifiant " Dans ".

Next : Mot-clé pour fermer la boucle.

Item : Objet à récupérer dans la collection.

Collection : Tableau qui contient des Objets.

```
'Exemple
Dim Var_Item As String
For Each Item As Object In MonTableau
    Var_Item = Var_Item + Item 'Var_Item va contenir tous les éléments du tableau en admettant que
 le tableau soit déjà remplit.
Next Item

'Autre exemple
Dim Chaine As String = "Toto va bien"
Dim Caractere As String
For each Caractere In Chaine
    Msgbox(Caractere.ToString) 'affiche chaque caractère de Chaine
Next
```

"**For Each - Next**" est donc très utile pour tester une chaîne de caractère.

III-D - Les opérateurs

Pour travailler avec les variables on utilise des opérateurs. Il en existe de trois sortes : les opérateurs d'arithmétiques et de concaténation (mise bout à bout de chaîne de caractères), les opérateurs de logiques et les opérateurs de comparaisons.

III-D-1 - Les opérateurs de comparaison

Opérateurs d'arithmétique et de concaténation :

Opérateur	Opération réalisée par l'opérateur
+	Additionne deux nombres
-	Soustrait deux nombres
*	Multiplie deux nombres
/	Divise deux nombres et retourne un nombre à virgule flottante (décimal)
\	Divise deux nombres et retourne un nombre entier
Mod (Modulo)	Divise deux nombres et retourne seulement le reste
^	Elève à la puissance un nombre
&	Additionne (concatène) deux chaînes

```
'Exemples
Dim x, y, Somme As Single
X = 10
Y = 15.4
Somme = X + Y 'Somme retourne 25,4

Dim Revenu, Taxes, Revenu_Reel As Integer
Revenu = 2000
Taxes = 2500
Revenu_Reel = Revenu - Taxes 'retourne -500

Dim a, b, c As Integer
a = 12
b = 3
c = a Mod b 'c = 0 car la division tombre juste donc le reste = 0

Dim Prenom, Nom, NomComplet As String
Prenom = "Florent"
Nom = "Aspic"
NomComplet = Prenom & Nom 'retourne "FlorentAspic"
```

III-D-2 - Les opérateurs de logique

Les opérateurs de logiques manipulent des valeurs 'True' ou 'False'.

Opérateurs	Syntaxe	Emploi
And (et)	Var_1 And Var_2	True si Var_1 et Var_2 sont vraies
Or (ou)	Var_1 Or Var_2	True si une des deux est vraie
Xor (ou exclusif)	Var_1 Xor Var_2	True si une et une seule est vraie
Not (non)	Not Var_1	True si Var est faux et vice versa

```
'Exemples
Dim a, b As Integer
Dim Var, Var_1 As String
a = 13
b = - 15
Var = "Cool"
```

```
Var_1 = "Snif"

If a Or b < 0 Then
   'Exécute la condition si a ou b est négative : ici c'est Vrai
End If

If a And b > 0 Then
   'Exécute la condition si a et b sont positifs : ici c'est Faux
End If

If Not Var = Var_1 Then
    'Exécute la condition si Var est différent de Var_1 : Ici c'est Vrai
End If
```

III-D-3 - Les opérateurs de comparaison

Les opérateurs de comparaison comparent deux nombres ou chaînes pour voir si ces nombres ou chaînes sont égaux, différents, supérieurs ou inférieurs.

Opérateurs	Signification
>	Strictement supérieur à
<	Strictement inférieur à
>=	Supérieur ou égal à
<=	Inférieur ou égal à
<>	Différent de
=	Egal à

```
'Exemples
Dim Age, AgeMin, AgeMax As Integer
Dim EntreeBoiteDeNuit As Boolean

Age = 12
AgeMin = 16
AgeMax = 45

If AgeMin < Age < AgeMax Then
    EntreeBoiteDeNuit = True 'La condition est Vraie
End If

If Age < AgeMin Then
     EntreeBoiteDeNuit = False 'La condition est Fausse
End If

If Age = AgeMin then
    EntreeBoiteDeNuit = True 'La condition est Vraie
End If

If Age >= 46 Then
    EntreeBoiteDeNuit = False 'La condition est Fausse
End If
```

III-E - Les constantes et énumérations

Les constantes sont des variables qui ne peuvent pas être la cible d'une assignation. C'est-à-dire que ces variables ne peuvent pas être modifiées. Pour déclarer une constante on utilise le mot-clé " Const ". Un avantage majeur des constantes est qu'elles améliorent la visibilité du code et son exécution.

```
'Exemples de constantes
Private Const Pi As Decimal = 3.14
```

```
Public Const Plat_Prefere As String = "Pates"

'Ensuite on peut les utiliser
Dim Rayon As Decimal = 5 'Soit un cercle de rayon 5
Dim AireCercle, PerimetreCercle As Decimal

AireCercle = Pi * Rayon ^ 2
PerimetreCercle = 2 * Pi * Rayon

MsgBox(Plat_Prefere) 'Affiche mon plat préféré !!
```

Les énumérations sont des structures qui permettent de créer une liste de constantes. Pour cela on utilise le mot-clé " Enum "

```
'Exemple
Private Enum Erreur
     Logique
     Execution
     Syntaxe
     Grammataire
End Enum

'les constantes ainsi créées sont
Erreur.Logique
Erreur.Execution
Erreur.Syntaxe
Erreur.Grammaire
```

Exemple concret d'utilisation des constantes :

```
'On crée une énumération
Enum Action
     Ajouter
     Supprimer
     Inserer
     Selectionner
End Enum

'Plus loin dans le code
Select Case Action
    Case Action.Ajouter 'dans ce cas on ajoute quelque chose
            'ici le bloc de code
    Case Action.Supprimer 'dans ce cas on supprime un élément
            'ici le bloc de code
    Case Action.Selectionner 'ici on sélectionne un élément
             'ici le bloc de code
    Case Action.Inserer 'ici on insère un élément dans un tableau par ex
             'ici le bloc de code
    Case Else
           MsgBox("Enumération non reconnue")
End Select
```

Les énumérations comme les constantes allègent le programme et le rende plus clair.

III-F - Les tableaux

Un tableau permet de regrouper des données de même type. Pour créer un tableau, on procède comme cela :

```
'Exemple création tableau
Dim Tableau(6) As Integer 'déclare un tableau de 7 entiers
```

Les index d'un tableau commence toujours à 0. Le nombre d'éléments dans le tableau est donc égal à son indice de création (dans l'exemple 6) + 1 = 7

On peut ajouter des valeurs dans le tableau :

```
'Exemple d'ajout de valeurs
Tableau(2) = 3 'ajoute pour l'indice 2 la valeur de 3
Tableau(0)= 98 'ajoute pour l'indice 0 la valeur de 98
Tableau(7) = 3 'provoquera une erreur car l'indice est supérieur à 6
```

Le tableau contient donc :

98
0
3
0
0
0
0

Pour affecter à une variable un élément du tableau, on procède comme cela :

```
'Affectation à une variable
Dim Var As Integer
Var = Tableau(0) 'Var contient donc 98
```

Pour effacer un tableau et récupérer la mémoire allouée par le tableau :

```
'Efface le tableau
Erase Tableau
Tableau = Nothing
'Ces deux commandes sont équivalentes.
```

Pour réinitialiser le tableau (remise à zéro des éléments) :

```
'Réinitialisation du tableau de l'index 0 au dernier index du tableau
Array.Clear(Tableau, 0, Tableau.Length)
```

Pour parcourir un tableau, rien de plus simple. On utilise une boucle :

```
'Lister les éléments d'un tableau
Dim Tableau(5) As Integer
For Each Item As Object In Tableau
     MsgBox(Item.ToString) 'affiche chaque valeur du tableau
Next Item

'une autre méthode
For i As Integer = 0 To Tableau.Length
      MsgBox(tableau(i).ToString)
Next
```

Il existe des classes spéciales pour les tableaux 'ArrayList et SortedList' mais cela ne sera pas abordé dans ce cours.

III-G - Les options de codage

Il en existe seulement trois types dont deux sont très répandus. Ces options permettent un meilleur codage et une meilleure compréhension du code. Le fait de les activer empêche le programmeur de faire des erreurs de logiques implicites (conversions implicites ou liaisons tardives par exemple). Voila comment les utiliser :

```
'Ces lignes de code doivent être écrites tout en haut de la feuille
Option Strict On 'active le mode Strict
Option Strict Off 'désactive le mode Strict
Option Explicit On 'active le mode Explicit
Option Explicit Off 'désactive le mode Explicit
Option Compare Binary 'mode binaire pour la comparaison de chaînes
Option Compare Text 'mode texte pour la comparaison de chaînes

'Si Option Strict Off
Dim a As Decimal = 5.987
Dim b As Integer
b = a 'a vaut 5. Il y a eu une conversion implicite de Decimal vers Integer. VB accepte.

'Si Option Strict On
Dim a As Decimal = 5.987
Dim b As Integer
b = a 'VB plante !!
'Il faut écrire
b = CInt(a)

'Si Option Strict Off
Dim a As Object = "Toto"
Dim b As Integer
b = a.Length 'a vaut 4. Il y a eu une liaison tardive. VB accepte.

'Si Option Strict On
Dim a As Object = "Toto"
Dim b As Integer
b = a.Length 'VB plante !!
'Il faut être strict et écrire :
b = CType(a, String).Length

'Si Option Explicit Off
a = "Toto" 'a est un String
b = 13 'b est un Integer
'VB accepte cela mais cela est très déconseillé !
'Exemple de problèmes causés avec les déclarations implicites de variables :
Dim Var 'Déclare Var implicitement
Varr = 10 'On croit avoir mis 10 dans Var mais le fait on créé une nouvelle variable nommée Varr et
 qui vaut 10 et Var vaut toujours 0 !!

'Si Option Explicit On
a = "Toto" 'a est un String
b = 13 'b est un Integer
'Cela est tout simplement refusé par Visual Basic#

'Option Strict Off permet n'importe quoi. C'est du Basic au mauvais sens du terme.
'Option Strict On oblige à une grande rigueur. C'est du VB.Net
```

Je vous conseille de laisser *Option Explicit* à *On*, ce qui oblige à déclarer **toutes** les variables avant de les utiliser Dans ce cas si vous tapez le nom d'une variable non déclarée, VB.NET génère une erreur. Par défaut, l'*Option Explicit* est à *On* et l'*Option Strict* à *Off*.

IV - L'interface utilisateur

IV-A - Présentation

Ici nous sommes dans la section 'Interface utilisateur'. On va apprendre à créer des éléments pour rendre dynamique les programmes. Sachez que cette partie est très importante car pour rendre agréable l'utilisation de votre programme, il faut avant tout créer une belle interface.

Pour créer un élément, il faut aller dans la boite à outils à gauche dans l'IDE (Integrated Development Environment). Sélectionner le composant à ajouter au formulaire (bouton, label, #) puis positionner le dans le formulaire.

Les contrôles présentés dans ce chapitre sont des contrôles de Visual Basic .NET Version 2005 (Framework 2.0). Il se peut que certaines propriétés n'existent pas avec la version 2003 de VB.NET. Donc ne vous affolez pas si toutes les propriétés des tableaux n'apparaissent pas pour les différents outils de la boite à outils

IV-B - La Form

Les *Forms* sont des formulaires Windows. En programmation, on les appelle des " Windows Form ". VB.NET génère automatiquement une *Form* lors de la création d'un projet. Pour créer une nouvelle *Form*, cliquer sur '**Ajouter un formulaire**' dans le menu '**Projet**'. Comme chaque contrôle, les *Forms* possèdent des propriétés (cf tableau).

Pour afficher la boite de propriété, cliquer sur le contrôle puis appuyer sur *F4*.

Tableau des propriétés les plus utilisées :

Propriété	Description
BackColor	Définis l'arrière plan du contrôle
BackGroundImage	L'image en arrière plan du contrôle
Cursor	Permet de changer le curseur lors du passage
Font	Le font (police) utilisé dans le contrôle
ForeColor	La couleur du premier plan pour le texte
FormBorderStyle	Comportement de la form (bordure et barre de titre)
Text	Le titre de la form
AllowDrop	Indique si on autorise le glisser / coller
ContextMenuStrip	Associe un menu contextuel au contrôle
Enabled	Indique si le contrôle est activé ou pas
Name	Le nom du contrôle utilisé dans le codage
Size	La taille du contrôle en pixels
WindowsState	L'état de la form au démarrage (minimize...)
AcceptButton	Le bouton d'acceptation du formulaire
CancelButton	Le bouton d'annulation du formulaire
ControlBox	Indique si le menu système est activée ou pas

HelpButton	Indique si le bouton d'aide est activé ou pas
Icon	Définis l'icône du contrôle
MainMenuStrip	Définis le menu principale du formulaire
MaximizeBox	Indique si le bouton d'agrandissement est activé
ShowIcon	Indique si l'icône est visible
ShowInTaskBar	Indique si le formulaire est visible dans la TaskBar
TopMost	Indique si le formulaire est au premier plan

Bien sur il existe d'autres fonctions.

IV-C - La Console

La console est une fenêtre du type DOS. C'est assez archaïque comme interface mais c'est la définition de la console. Dans une console on peut afficher uniquement du texte. Pour créer une application console, aller dans 'Fichier' puis 'Nouveau Projet' suivit 'Application Console'.

```
'Exemple de lignes de code utilisant la console
Dim LigneTapee As String = Console.In.ReadLine() 'permet de lire du texte
Console.ReadLine() 'marche aussi
Console.Out.WriteLine("Ligne à afficher") 'pour afficher du texte
Console.WriteLine 'est aussi accepté

'Petit programme en application console
Console.Out.Write("Tapez une ligne : ")
Dim LigneTapee As String = Console.In.ReadLine()
Console.Out.WriteLine("Je viens de taper : " & LigneTapee)
```

IV-D - Le Bouton

Le bouton est le composant le plus important puisque c'est grâce à lui que l'utilisateur pourra interagir avec le programme. Un bouton permet essentiellement de valider ou d'annuler une fonction.

Tableau des propriétés les plus utiles

Propriété	Description
BackColor	Définis l'arrière plan du contrôle
BackGroundImage	L'image en arrière plan du contrôle
Cursor	Permet de changer le curseur lors du passage de la souris
FlatStyle	Le style du bouton
Font	Le font (police) utilisé dans le contrôle
ForeColor	La couleur du premier plan pour le texte
Text	Le texte du bouton
TextAlign	L'alignement du texte du bouton
AllowDrop	Indique si on autorise le glisser / coller
ContextMenuStrip	Associe un menu contextuel au contrôle
Enabled	Indique si le contrôle est activé ou pas
Visible	Indique si le contrôle est visible ou pas
Name	Le nom du contrôle utilisé dans le codage

Size	La taille du contrôle en pixels
AutoSize	Définis si le contrôle est dimensionné automatiquement

IV-E - Le Label

Un label est un élément permettant d'informer l'utilisateur d'une chose. Par exemple, pour indiquer à l'utilisateur d'entrer son nom dans une TextBox, on utilise un 'Label'. Ils renseignent donc sur l'utilisation du programme. Un label ne peut pas être modifié par l'utilisateur.

Tableau des principales fonctions du label :

Propriété	Description
BackColor	Définis l'arrière plan du contrôle
Image	L'image affichée sur le label
Cursor	Permet de changer le curseur lors du passage de la souris
FlatStyle	Le style du bouton
Font	Le font (police) utilisé dans le contrôle
ForeColor	La couleur du premier plan pour le texte
Text	Le texte du bouton
TextAlign	L'alignement du texte du bouton
AllowDrop	Indique si on autorise le glisser / coller
ContextMenuStrip	Associe un menu contextuel au contrôle
Enabled	Indique si le contrôle est activé ou pas
Visible	Indique si le contrôle est visible ou pas
Name	Le nom du contrôle utilisé dans le codage
Size	La taille du contrôle en pixels
AutoSize	Définis si le contrôle est dimensionné automatiquement

NB : Propriétés presque similaires aux boutons.

IV-F - La Textbox

Les *TextBox* sont des emplacements pour accueillir du texte (d'où le nom de *TextBox*). Elles sont très utiles pour récupérer du texte. Par exemple dans MSN tu entres ton login et ton mot de passe dans des TextBox prévues à cet effet. Une *TextBox* peut donc être modifiée par l'utilisateur.

Tableau des différentes propriétés d'une TextBox :

Propriété	Description
BackColor	Définis l'arrière plan du contrôle
Cursor	Permet de changer le curseur lors du passage de la souris
Font	Le font (police) utilisé dans le contrôle
ForeColor	La couleur du premier plan pour le texte
Text	Le texte initial visible dans la TextBox

TextAlign	L'alignement du texte
ContextMenuStrip	Associe un menu contextuel au contrôle
Enabled	Indique si le contrôle est activé ou pas
MaxLength	Indique le nombre maximal de caractères dans la TextBox
Multiline	Indique si la TextBox est multi ligne ou pas
PasswordChar	Caractère à afficher pour la saisie du mot de passe
ReadOnly	Indique si la TextBox est en lecture seule
Visible	Indique si le contrôle est visible ou pas
Name	Le nom du contrôle utilisé dans le codage
Size	La taille du contrôle en pixels
AllowDrop	Indique si on autorise le glisser / coller

IV-G - Les cases à cocher

Les *CheckBox* sont des cases à cocher. Elles permettent de savoir si l'utilisateur veut ou pas activer la fonction. Il est possible aussi d'utiliser des *CheckBox* avec 3 états d'activation. (cf Tableau)

Tableau des propriétés des CheckBox

Propriété	Description
Appearance	Indique l'apparence du contrôle
BackColor	Définis l'arrière plan du contrôle
BackGroundImage	L'image en arrière plan du contrôle
CheckAlign	Emplacement de la case à cocher
Checked	Indique si la case est cochée
CheckState	Indique l'état de la case à cocher
Cursor	Permet de changer le curseur lors du passage de la souris
FlatStyle	Le style du bouton
Font	Le font (police) utilisé dans le contrôle
ForeColor	La couleur du premier plan pour le texte
Text	Le texte de la CheckBox
TextAlign	L'alignement du texte
AllowDrop	Indique si on autorise le glisser / coller
AutoCheck	Indique si la case se coche automatiquement
ContextMenuStrip	Associe un menu contextuel au contrôle
Enabled	Indique si le contrôle est activé ou pas
ThreeState	Indique si la case à 3 états d'activation
Visible	Indique si le contrôle est visible ou pas
Name	Le nom du contrôle utilisé dans le codage
Size	La taille du contrôle en pixels

Les RadioButton sont identiques aux CheckBox. Cependant il ne peut y avoir qu'un seul RadioButton activé contrairement aux CheckBox où il peut y avoir autant de cases cochées que de CheckBox. Les propriétés sont identiques aux CheckBox. La seule différence est que les propriétés 'CheckState' et 'ThreeState' n'existent pas.

IV-H - La Combobox

Les *ComboBox* sont des menus déroulants. Elles permettent de choisir une option parmi plusieurs. Exemple : Quel est ta couleur préférée ? On utilisera une *ComboBox* pour afficher les réponses possibles (Rouge, Bleu, Vert, Noir...)

Tableau des propriétés des ComboBox

Propriété	Description
BackColor	Définis l'arrière plan du contrôle
Cursor	Permet de changer le curseur lors du passage de la souris
DropDownStyle	Apparence de la zone du menu déroulant
FlatStyle	Le style de la ComboBox
Font	Le font (police) utilisé dans le contrôle
ForeColor	La couleur du premier plan pour le texte
Text	Le texte de la ComboBox
AllowDrop	Indique si on autorise le glisser / coller
ContextMenuStrip	Associe un menu contextuel au contrôle
DrawMode	Indique si le code gère le dessin des éléments de la liste
Enabled	Indique si le contrôle est activé ou pas
Sorted	Indique comment sont triés les éléments de la liste
Visible	Indique si le contrôle est visible ou pas
Name	Le nom du contrôle utilisé dans le codage
Size	La taille du contrôle en pixels
Items	Permet d'ajouter les éléments dans la liste déroulante
Tag	Données définies par l'utilisateur associées à l'objet

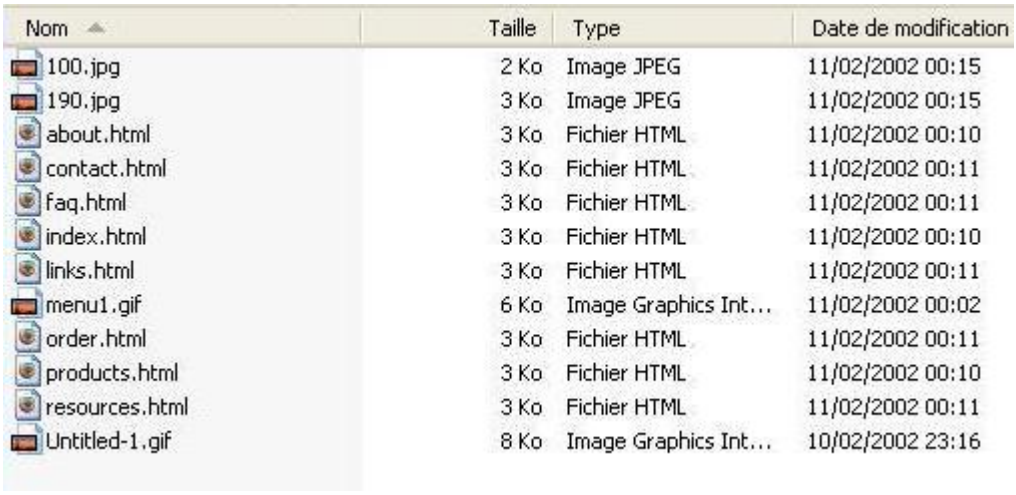
IV-I - Les listes

Il existe trois types de liste : les *ListBox*, les *ListView* et les *TreeView*.

La *ListBox* est le contrôle le plus facile à maîtriser. Il permet d'ajouter en ligne des éléments. Quand on organise des dossiers ou fichiers avec Windows en mode '*liste*' on obtient le résultat dans une *ListBox* (voir photo).

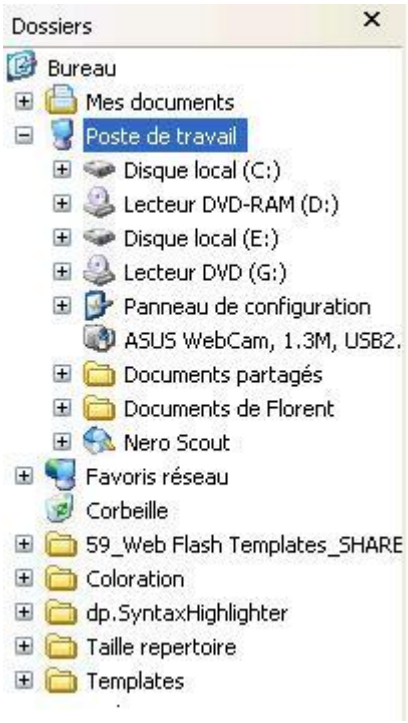
Une ListBox dans Windows XP

La *ListView* est plus complexe car elle permet plus de choses. En effet, on peut créer plusieurs colonnes avec différents modes (détails, icones, grandes icones#). Quand on organise des dossiers ou fichiers avec Windows en mode '*détails*' on obtient le résultat dans une *ListView* (voir photo).



Une ListView dans Windows XP

La *TreeView* est assez dur à manipuler dans la mesure où elle représente une arborescence avec plusieurs niveaux de n#uds. Dans une *TreeView*, il y a un n#ud racine, suivit de n#uds puis de sous n#uds, ce qui rend ce contrôle difficile d'accès aux débutants (comme je l'étais avant, je confirme que j'ai galéré avec ce contrôle !). Dans Windows, cliquer sur '*Poste de Travail*' puis '*Dossiers*' et vous obtenez une *TreeView* (voir photo).



Une TreeView dans Windows XP

Tableau des propriétés d'une ListBox

Propriété	Description
BackColor	Définis l'arrière plan du contrôle
Cursor	Permet de changer le curseur lors du passage de la souris
Font	Le font (police) utilisé dans le contrôle
ForeColor	La couleur du premier plan pour le texte
AllowDrop	Indique si on autorise le glisser / coller
DrawMode	Indique si le code gère le dessin des éléments de la liste
ContextMenuStrip	Associe un menu contextuel au contrôle
MultiColumn	Indique comment doivent être affichées les données
Sorted	Indique comment sont triés les éléments de la liste
Visible	Indique si le contrôle est visible ou pas
Name	Le nom du contrôle utilisé dans le codage
Size	La taille du contrôle en pixels
Items	Permet d'ajouter les éléments dans la ListBox
Tag	Données définies par l'utilisateur associées à l'objet
Enabled	Indique si le contrôle est activé ou pas

Tableau des propriétés d'une ListView

Propriété	Description

BackColor	Définis l'arrière plan du contrôle
BackGroundImage	L'image en arrière plan du contrôle
BorderStyle	Indique le style de la bordure de la ListView
Cursor	Permet de changer le curseur lors du passage de la souris
CheckBoxes	Indique si des CheckBox sont visibles à coté des éléments
Font	Le font (police) utilisé dans le contrôle
ForeColor	La couleur du premier plan pour le texte
FullRowSelect	Indique si toute la ligne est sélectionnée
GridLines	Affiche un quadrillage autour des éléments en mode détails
View	Définis la vue dans laquelle les éléments seront affichés
AllowDrop	Indique si on autorise le glisser / coller
Columns	Gère les colonnes de la ListView
ContextMenuStrip	Associe un menu contextuel au contrôle
Items	Gère les éléments de la ListView
Sorting	Indique comment sont triés les éléments de la liste
Visible	Indique si le contrôle est visible ou pas
Name	Le nom du contrôle utilisé dans le codage
Size	La taille du contrôle en pixels
Tag	Données définies par l'utilisateur associées à l'objet
Enabled	Indique si le contrôle est activé ou pas

Tableau des propriétés d'une TreeView

Propriété	Description
BackColor	Définis l'arrière plan du contrôle
BorderStyle	Indique le style de la bordure de la TreeView
Cursor	Permet de changer le curseur lors du passage de la souris
CheckBoxes	Indique si des CheckBox sont visibles à coté des éléments
Font	Le font (police) utilisé dans le contrôle
ForeColor	La couleur du premier plan pour le texte
AllowDrop	Indique si on autorise le glisser / coller
ContextMenuStrip	Associe un menu contextuel au contrôle
DrawMode	Indique si le code gère le dessin des éléments de la liste
FullRowSelect	Indique si la surbrillance s'étend sur la largeur du TreeView
Nodes	Gère l'ensemble des noeuds du contrôle TreeView
Visible	Indique si le contrôle est visible ou pas
Name	Le nom du contrôle utilisé dans le codage
Size	La taille du contrôle en pixels
Tag	Données définies par l'utilisateur associées à l'objet

Enabled	Indique si le contrôle est activé ou pas

IV-J - La picturebox

Les *PictureBox* permettent d'afficher des images et de les redimensionner.

Tableau des propriétés d'une TreeView

Propriété	Description
BackColor	Définis l'arrière plan du contrôle
BackGroundImage	L'image en arrière plan du contrôle
Cursor	Permet de changer le curseur lors du passage de la souris
Image	L'image a affichée dans la PictureBox
ErrorImage	Icone à afficher lorsque le chargement d'une image échoue
InitialImage	Image à afficher pendant le chargement d'une autre image
ContextMenuStrip	Associe un menu contextuel au contrôle
Enabled	Indique si le contrôle est activé ou pas
SizeMode	Manière dont l'image va être redimensionnée
Visible	Indique si le contrôle est visible ou pas
Name	Le nom du contrôle utilisé dans le codage
Size	La taille du contrôle en pixels

IV-K - La progressbar

Les *ProgressBar* permettent de suivre l'évolution d'une application. Windows utilise de nombreuses *ProgressBar* (lors de la recherche d'un fichier par exemple). Pour faire fonctionner une *ProgressBar*, il faut spécifier une valeur pour la propriété *Minimal* et une autre valeur pour la propriété *Maximal*. Si vous souhaitez représenter un pourcentage d'avancement d'une opération, mettez 0 pour *Minimum* et 100 pour *Maximum*.

Tableau des propriétés d'une TreeView

Propriété	Description
BackColor	Définis l'arrière plan du contrôle
Cursor	Permet de changer le curseur lors du passage de la souris
ForeColor	La couleur du premier plan pour le texte
Maximum	Limite supérieure de la plage que la ProgressBar utilise
Minimum	Limite inférieure de la plage que la ProgressBar utilise
Style	Définis le style de la ProgressBar
Value	Valeur actuelle pour la ProgressBar
ContextMenuStrip	Associe un menu contextuel au contrôle
Visible	Indique si le contrôle est visible ou pas
Name	Le nom du contrôle utilisé dans le codage
Size	La taille du contrôle en pixels

Tag	Données définies par l'utilisateur associées à l'objet
Enabled	Indique si le contrôle est activé ou pas

IV-L - Le tabcontrol

Le TabControl est un outil très important et surtout bien pratique. En effet, il permet de mettre beaucoup de choses dans peu d'espace puisqu'il gère cela en onglets. Dans un TabControl, il y a des TabPages. Vous pouvez définir autant de TabPages que vous le souhaitez.

Tableau des propriétés d'un TabControl :

Propriété	Description
Cursor	Permet de changer le curseur lors du passage de la souris
Font	Le font (police) utilisé dans le contrôle
ImageList	Objet qui permet de contenir des images pour les onglets
AllowDrop	Indique si on autorise le glisser / coller
DrawMode	Indique si l'utilisateur ou le système peint les légendes
HotTrack	Indique si l'aspect des onglets change lors du survol de la souris
ContextMenuStrip	Associe un menu contextuel au contrôle
MultiLine	Indique si plusieurs lignes d'onglets sont autorisées
SizeMode	Indique la façon dont les onglets sont redimentionnés
TabPages	La liste des TabPages contenus dans le TabControl
Visible	Indique si le contrôle est visible ou pas
Name	Le nom du contrôle utilisé dans le codage
Size	La taille du contrôle en pixels
Tag	Données définies par l'utilisateur associées à l'objet
Enabled	Indique si le contrôle est activé ou pas

IV-M - Les boites de dialogue

Les boites de dialogue sont des fenêtres utilisées par Windows pour exécuter une fonction précise. Il existe en gros cinq boites de dialogue avec chacune un rôle spécifique :

a L'*OpenFileDialog* permet de demander à l'utilisateur d'ouvrir un fichier. Cela est très utile pour effectuer un transfert de fichier par exemple.

b La *FolderBrowserDialog* est en fait une boite de dialogue qui permet de chercher un dossier spécifique ou un disque dur. Par exemple, pour lister le contenu d'un dossier ou d'un lecteur, on demande à l'utilisateur de choisir ce dossier ou lecteur et on utilise ce contrôle.

c La *ColorDialog* est moins utilisée. Elle permet tout simplement d'affecter une couleur à un élément à partir d'une boite de couleurs prédéfinies. Utile pour changer le font d'écran d'un programme par exemple.

d La *SaveFileDialog*, comme son nom l'indique, permet tout simplement de sauvegarder un fichier.

e Enfin la dernière est la *FontDialog* qui affecte une certaine police à un texte. Très pratique pour les logiciels de messagerie instantanée.

Le plus intéressant bien sur est de récupérer dans le code la sélection de l'utilisateur. Cela est très simple :

```
'Exemples avec les boites de dialogues
Dim Selection As String
Dim Couleur As Color
Dim MonFont As Font

FolderBrowserDialog.ShowDialog()
Selection = FolderBrowserDialog.SelectedPath 'récupère le chemin du dossier

OpenFileDialog.ShowDialog()
Selection = OpenFileDialog.FileName 'récupère le nom et le chemin du fichier

ColorDialog.ShowDialog()
Couleur = ColorDialog.Color 'récupère la couleur sélectionnée

FontDialog.ShowDialog()
MonFont = FontDialog.Font 'récupère le font (police, taille et style)
```

V - Le débogage

Le débogage est une étape très importante et fastidieuse. Elle permet de trouver et de corriger les bugs du programme. De nombreux outils sont à notre disposition pour nous aider mais le débogage reste encore une tâche difficile.

V-A - Les différents type de bugs

En programmation, il existe **trois** types d'erreurs possibles :

a Les erreurs de syntaxe
b Les erreurs de logique
c Les erreurs d'exécution

Une **erreur de syntaxe** est un bug qui nait de la mauvaise orthographe d'une commande. Si vous tapez *INTEGEER* au lieu de *INTEGER*, VB.NET n'a aucune idée de ce que *INTEGEER* signifie et il n'essaye même pas d'exécuter le reste du programme. Les erreurs de syntaxes sont affichées dans la liste d'erreurs. Lorsque VB.NT rencontre une erreur de syntaxe, il met gentiment dans la liste d'erreur le mot mal orthographié pour vous montrer exactement quel est le problème. Il suffit donc de corriger l'orthographe de la commande et d'exécuter à nouveau le programme. Une suffit d'une seule erreur de syntaxe pour empêcher le programme de s'exécuter. Lorsque votre programme s'exécute finalement pour la première fois, vous savez que le programme ne possède aucunes erreurs de syntaxe.

Une **erreur d'exécution** est plus subtile qu'une erreur de syntaxe car elle ne se produit que lorsque votre programme reçoit des données qu'il n sait pas vraiment gérer. Un programme peut être truffé d'erreurs d'exécution, vous ne le saurez pas tant que vous n'avez pas exécuté le programme. Pour déclencher une erreur d'exécution dans votre propre vie, allez au MacDo et, lorsque l'employé vous demande ce qu'il peut faire pour vous, commander une crêpe. Du fait que l'employé s'attend de vous que vous commandiez quelque chose qui fait partie du menu des MacDo, il ne saura pas vous répondre et il est probable qu'il en durera une erreur d'exécution. (C'est une image mais c'est compréhensible).

```
'Exemple d'erreur d'exécution
Dim MonTableau As New ArrayList 'déclare un tableau
MonTableau = Nothing 'détruit le tableau
MonTableau.Items.Add(" Cool ") 'provoquera une erreur d'exécution car MonTableau n'existe plus
 puisqu'il vaut " nothing "
```

Sachez que même les grands programmes (MSN et autres) comportent des bugs qui sont des erreurs d'exécution. Il est impossible de créer un gros programme sans aucunes erreurs.

Le type de bugs le plus délicat est dû à une **erreur de logique**. Une erreur de logique se produit lorsque le programme ne fonctionne pas correctement parce que vous lui avez fourni de mauvaises commandes. Par exemple, si vous avez des enfants adolescents, demander leur de ranger leur chambre ou de faire la vaisselle. Ils s'acquitteront de la tache mais pas forcément de la façon prévue. Au lieu de bien nettoyer les assiettes, il va peut être seulement les mettre sous l'eau froide, ou alors mettre tout son boxons sous le lit# Dans ces deux situations, l'adolescent a suivi vos instructions. Vos instructions n'étaient tout simplement pas assez précises ! Un ordinateur n'est pas très différent d'un adolescent !? (douteux). S'il peut trouver une échappatoire à vos instructions, il ne s'en privera pas ! Le problème est que comme vous pensez que vous avez donné à l'ordinateur la bonne marche à suivre, vous n'avez aucune idée de pourquoi le programme ne fonctionne pas. Il faut maintenant trouver l'endroit où les instructions ne sont pas assez précises et claires. Dans le cas d'un grand programme, vous avez éventuellement à le relire entièrement ligne par ligne (quel plaisir).

V-B - Comment détruire les bugs ?

Tout d'abord il semble logique de créer un bug. Une fois le bug crée, il faut le trouver, mais il faut trouver ce qui provoque le bug. Une fois le bug localisé, il suffit de le corriger mais attention en corrigeant un bug vous pouvez en avoir créés d'autres ! Donc je vous conseille vivement de réécrire la partie du code qui contient le bug. En d'autres termes, le débogage est loin d'etre une tache facile.

Trouver l'endroit où se cache le bug est la partie la plus difficile. Le moyen le plus simple est d'exécuter le programme ligne par ligne mais cela reste assez fastidieux. Imaginez que votre programme comporte 10 000 lignes de code, je vous vois mal examiner les lignes une par une ! Il faut donc remédier à d'autres méthodes. Il convient donc de regarder la partie du programme où vous pensez que le bug est susceptible d'agir. Par exemple, si votre fichier ne s'enregistre pas correctement, l'erreur se situe dans la partie du code qui code pour l'enregistrement du fichier.

Pour combattre les bugs, on peut mettre en place un piège à erreur. Un piège à erreur indique à l'ordinateur : *Voici un ensemble d'instructions génériques à suivre en cas de rencontre d'un problème que tu ne sais gérer*. ces instructions génériques et simple peuvent être simple pour afficher un message à l'écran. C'est toujours mieux que faire planter l'ordinateur !

```
'Exemple de piège à erreur dans VB.NET
Try 'Ouvre le piège
   Dim a As Integer = 30
   Dim b As Integer = 0

   Dim Resultat As Decimal = a / b

   Call CalculerPi()
Catch Ex As Exception 'Intercepte l'erreur en cas de pb
   MsgBox(Ex.ToString) 'Affiche l'erreur : Ici Division par 0 impossible
End Try 'Ferme le piège
```

Dans cet exemple, le programme va planter car la division par 0 n'est pas possible en mathématiques. Lorsque le programme va exécuter la ligne du calcul du résultat de la division, le code va passer immédiatement dans le ***Catch*** et afficher le message d'erreur. Par conséquent, la fonction ***CalculerPi*** ne sera pas exécutée. Vous comprenez donc que pour tester une partie de code, il suffit de copier ce code entre le ***Try*** et le ***Catch***. Il ne vous reste plus qu'à gérer l'erreur en l'affichant.

Il existe d'autres méthodes pour intercepter les erreurs. Le ***pas à pas*** consiste à exécuter le programme ligne par ligne en s'interrompant après chaque ligne. Vous pouvez alors voir ce que le programme a réalisé et étudier le code. Si le programme fonctionne bien de façon dont vous le voulez, c'est que la ligne est correcte. Dans le cas contraire, il y a un bug que vous venez de découvrir. Pour traverser un programme pas à pas, appuyer sur F11 pour un ***pas à pas détaillé*** : La commande ***pas à pas détaillé*** traverse la totalité du programme une ligne à la fois, y compris les lignes dans les procédures du programme. L'autre option est d'utiliser un ***pas à pas principal*** en appuyant sur ***F10*** : Le programme traverse la totalité du programme mais à chaque fois que Visual Basic rencontre une procédure, il exécute toutes les instructions dans cette procédure sans vous obliger à visionner ligne par ligne la procédure (c'est donc plus rapide).

Les commandes ***pas à pas détaillé*** et ***pas à pas principal*** commencent l'exécution au début du programme et la poursuivent jusqu'à la fin. Pour des gros programmes, cette méthode devient fastidieuse et compliquée. Pour sauter certaines parties de votre programme dont vous savez déjà qu'elles fonctionnent, vous pouvez définir des ***points d'arrêt***. Un point d'arrêt indique à Visual Basic d'exécuter le programme jusqu'à atteindre le point d'arrêt. Pour mettre

en place un point d'arrêt rien de plus simple, cliquer à gauche de la ligne où vous souhaitez insérer une point d'arrêt. Visual Basic affiche un point rouge dans la marge de gauche pour vous montrer à quelle ligne correspond le point d'arrêt. Pour supprimer un point d'arrêt, cliquer de nouveau sur la ligne dans la marge. Si vous souhaitez supprimer tous les points d'arrêt d'un seul coup, aller dans le menu déboguer puis cliquer sur effacer tous les points d'arrêt.

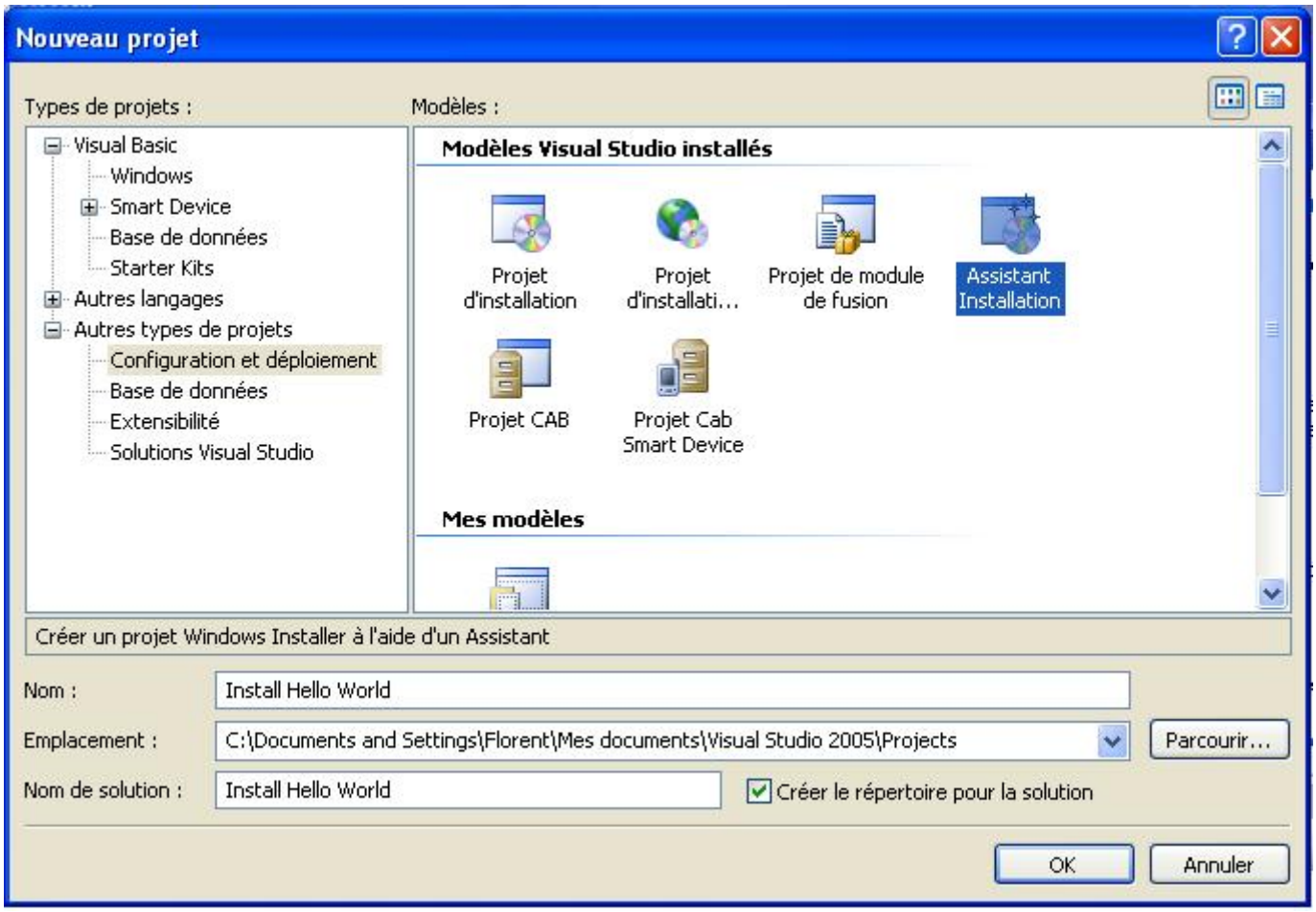
La dernière méthode pour intercepter les bugs est l'espionnage de variables. Cette fonction est très utile. Elle permet de vérifier la valeur d'une variable à tout moment, comme ça vous pouvez contrôler la variable. Pour ajouter un espion, vous devez d'abord déboguer la programme. Ensuite sélectionner la variable et faites un clique droit puis sélectionner Ajouter un espion. L'espion apparaît dans la fenêtre des espions. Elle affichera la valeur de la variable à chaque instant.

Pour récapituler, pour combattre un bug, il faut le localiser, puis le détruire. Pour détruire les bugs, vous disposez du ***pas à pas*** (principal ou détaillé), des ***points d'arrêt*** (pour intercepter le code à un endroit précis), des espions (pour espionner des variables) et des pièges à erreur (pour gérer des erreurs).

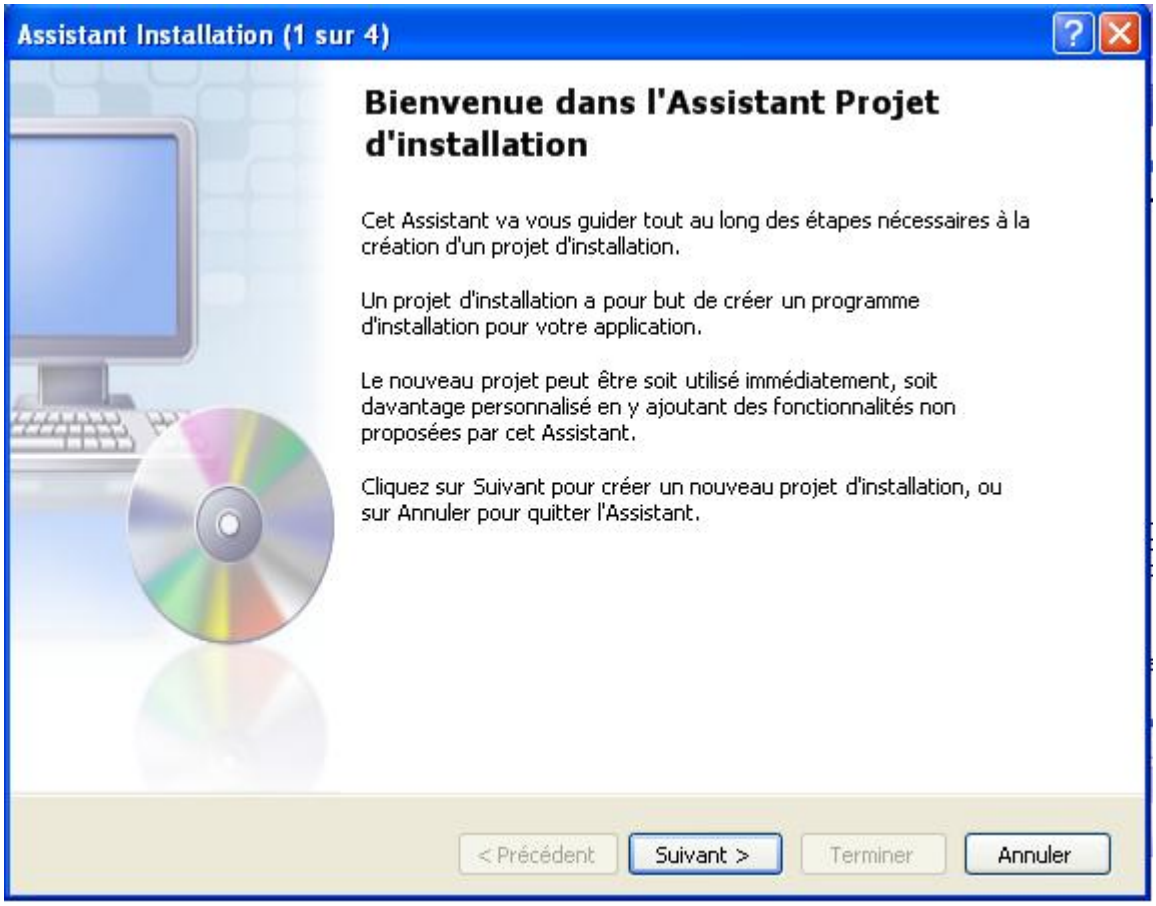
VI - La diffusion de l'application

VI-A - Comment créer une installation (Setup) ?

Dans cet exemple, le projet dont l'Install sera généré s'appelle ***Hello World***. Une fois votre projet ouvert et terminé de préférence, cliquer sur ***Fichier*** puis ***ajouter*** suivit de ***nouveau projet***. Dans le volet de gauche, dérouler le menu ***Autre types de projets*** puis choisissez Configuration et déploiement. Sélectionner ensuite dans le volet de droite, ***Assistant Installation***

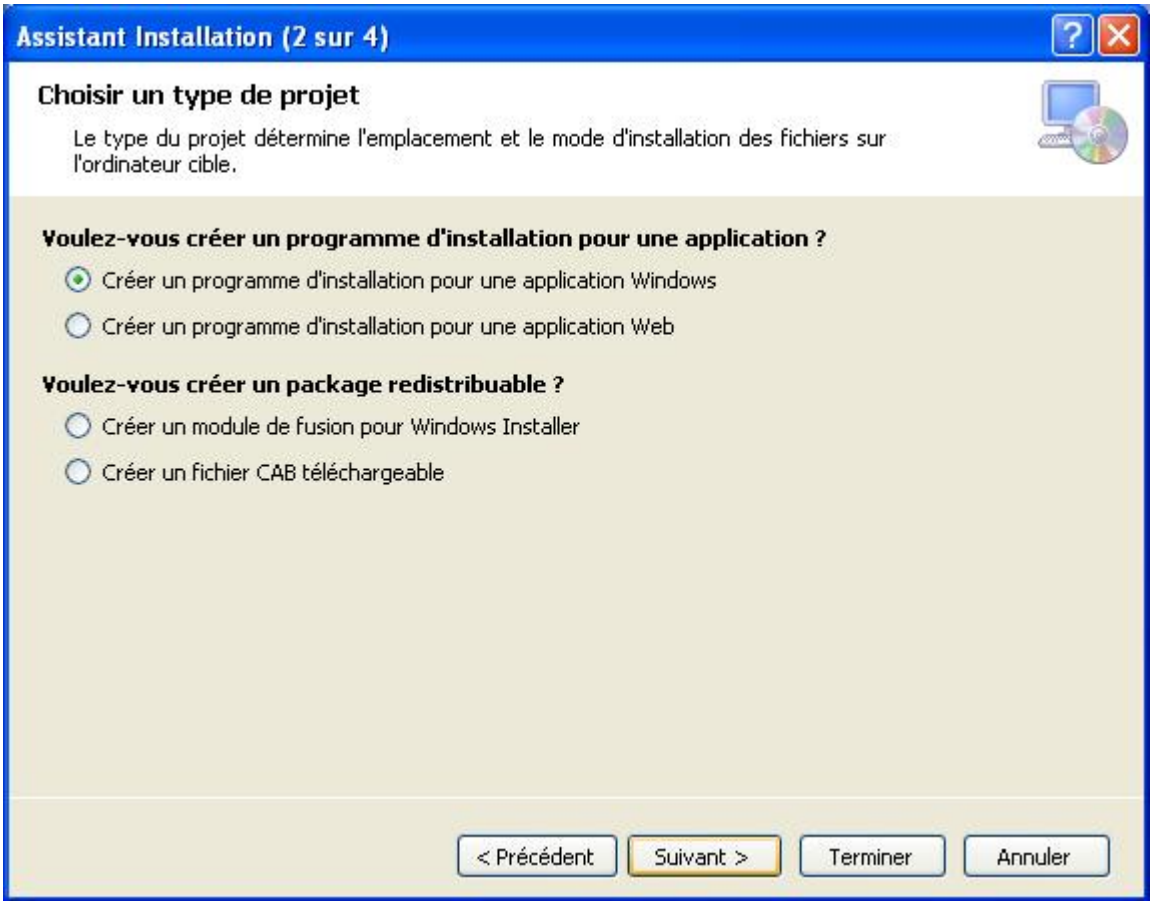


L'assistant d'installation s'ouvre. Il va vous guidez à travers les étapes à suivre pour créer votre installation. La page d'accueil s'affiche. Cliquer sur ***Suivant***

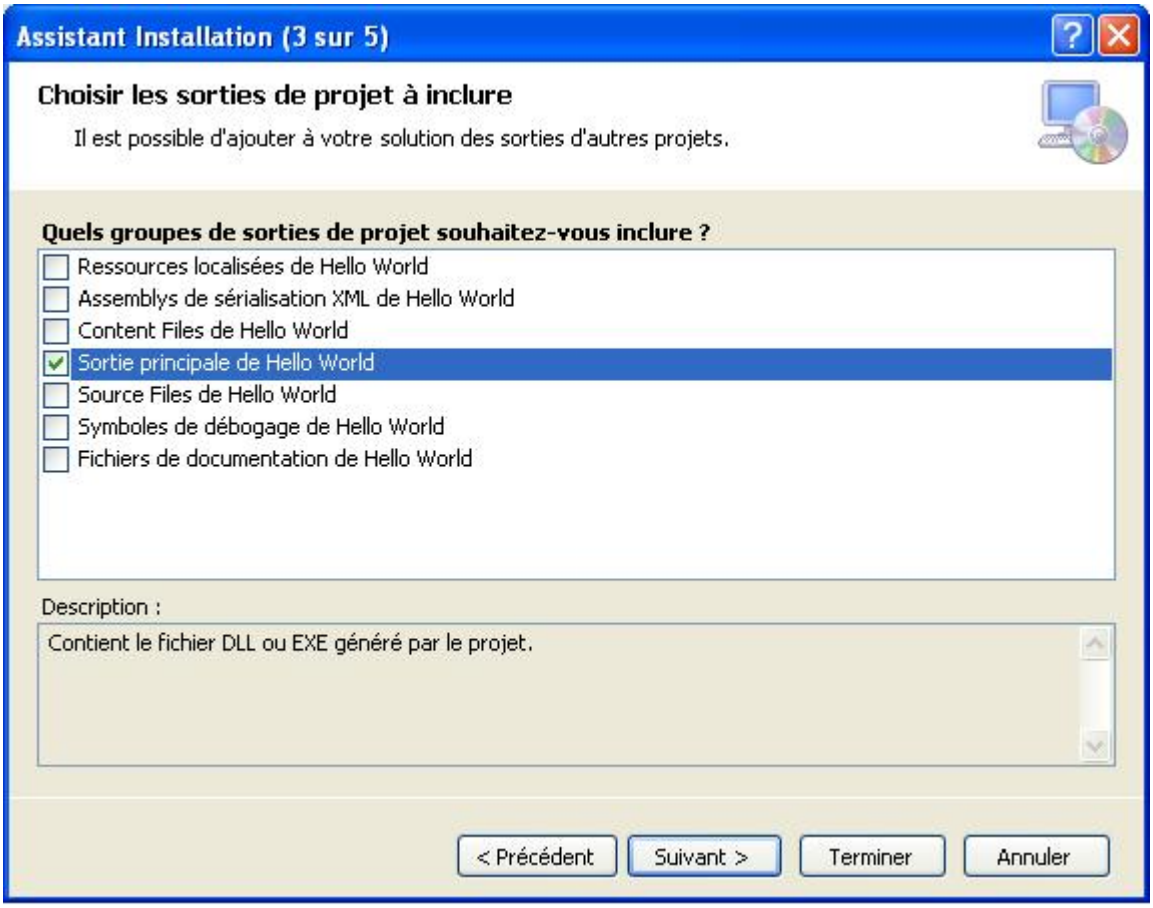


Choisissez l'option ***Créer un programme d'installation pour une application Windows*** (cochée par défaut).

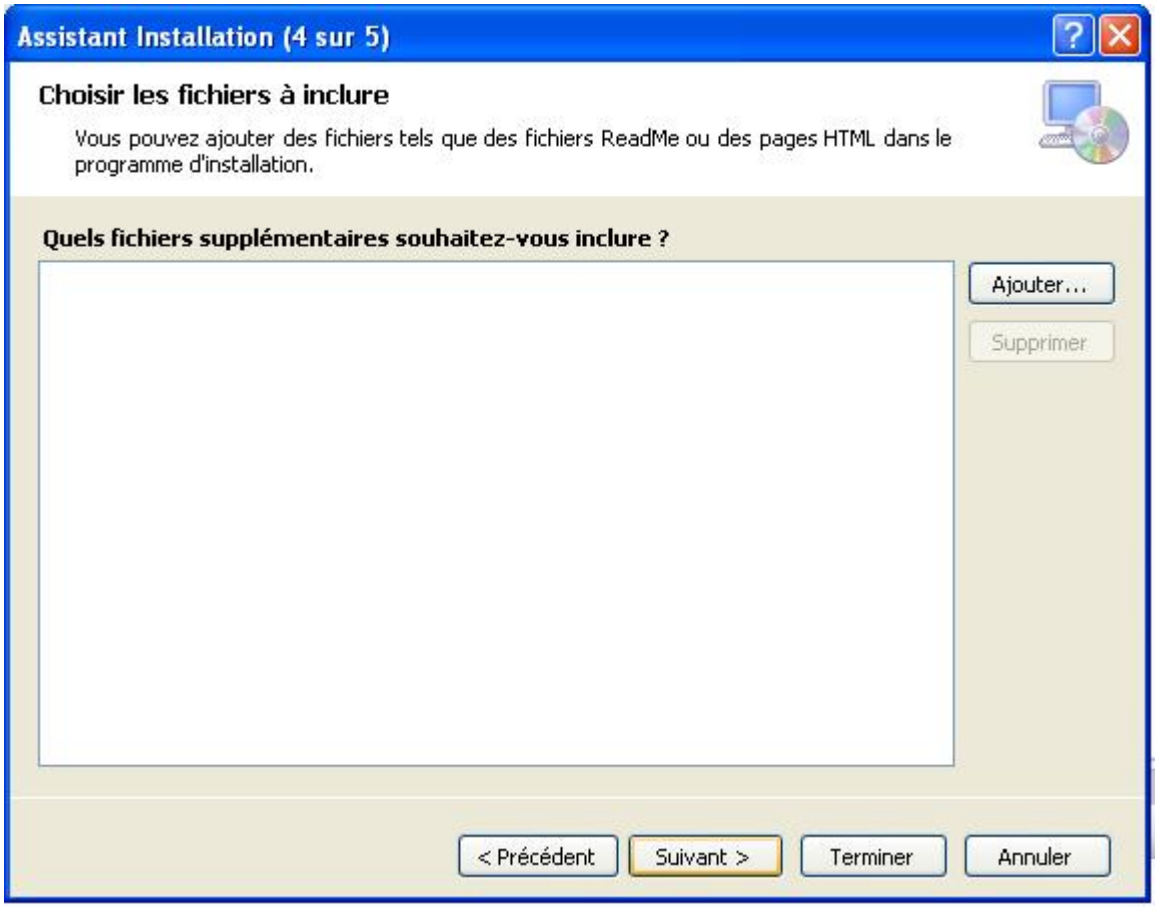
S'il s'agit d'une application Web, choisissez l'autre option. Ne cocher rien dans ***Voulez vous créer un package redistribuable****.*



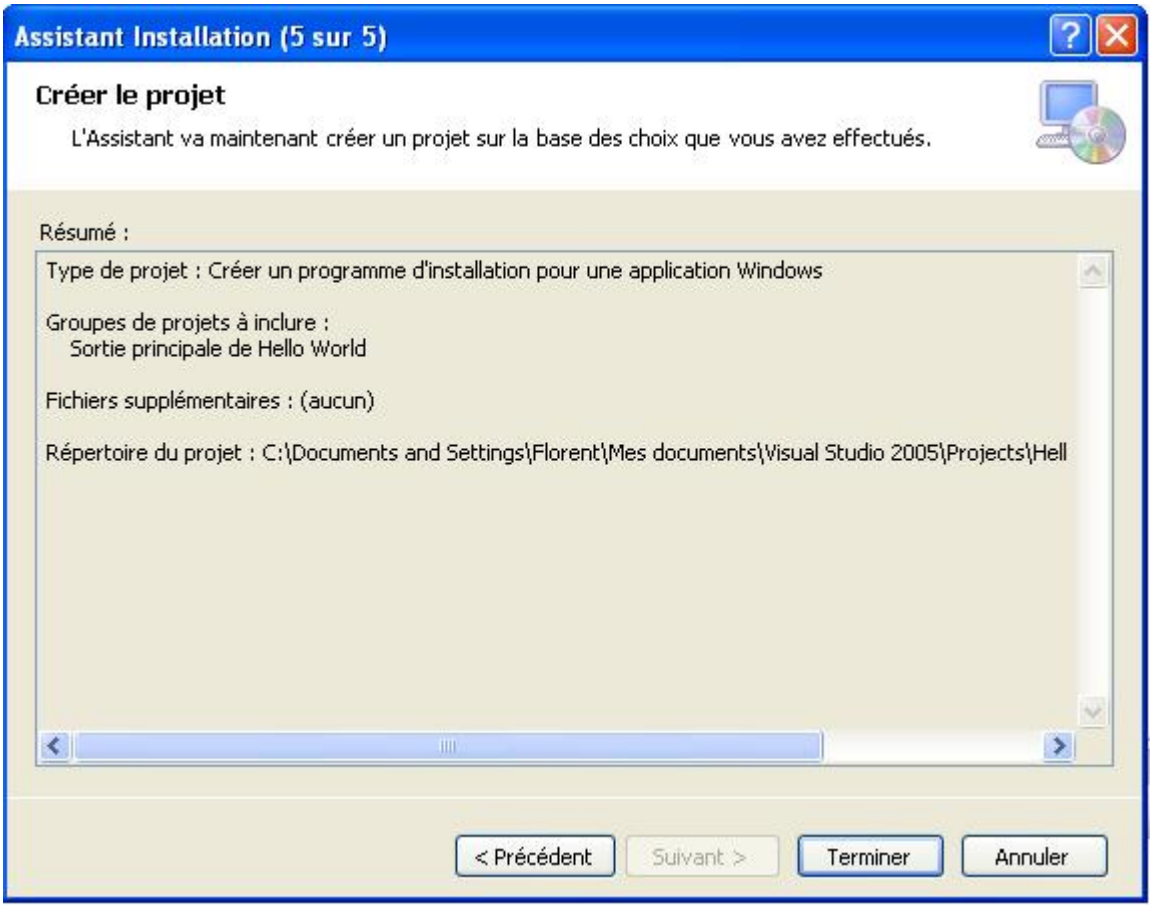
Dans la fenêtre suivante, sélectionner la sortie principale du projet. Visual Studio générera un fichier Exe pour votre application (dans certains cas cela peut être un fichier dll s'il s'agit d'un projet bibliothèque de classe). Cliquer sur ***Suivant***.



Ici, vous pouvez ajouter des fichiers supplémentaires à votre application. Par exemple, si votre programme contient une icône, vous devez ajouter le fichier Ico manuellement via cette interface. Pour cela, cliquer sur ***Ajouter*** puis chercher votre fichier sur votre disque dur. Idem si vous souhaitez ajouter un fichier *Read Me*. Une fois tous vos fichiers ajoutés, cliquer sur ***Suivant***.



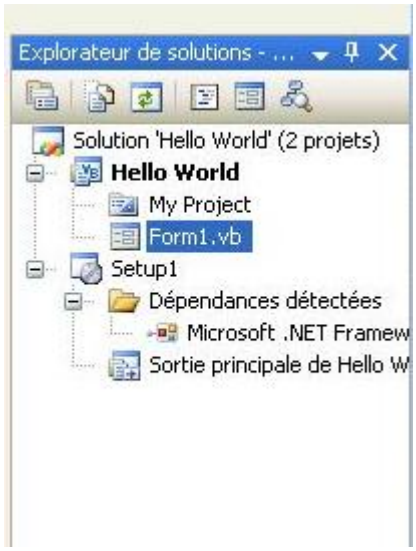
Un récapitulatif de votre installation s'affiche. Vérifier vos choix. En cas d'erreur vous pouvez cliquer sur ***Précédent*** sinon cliquer sur ***Terminer***.



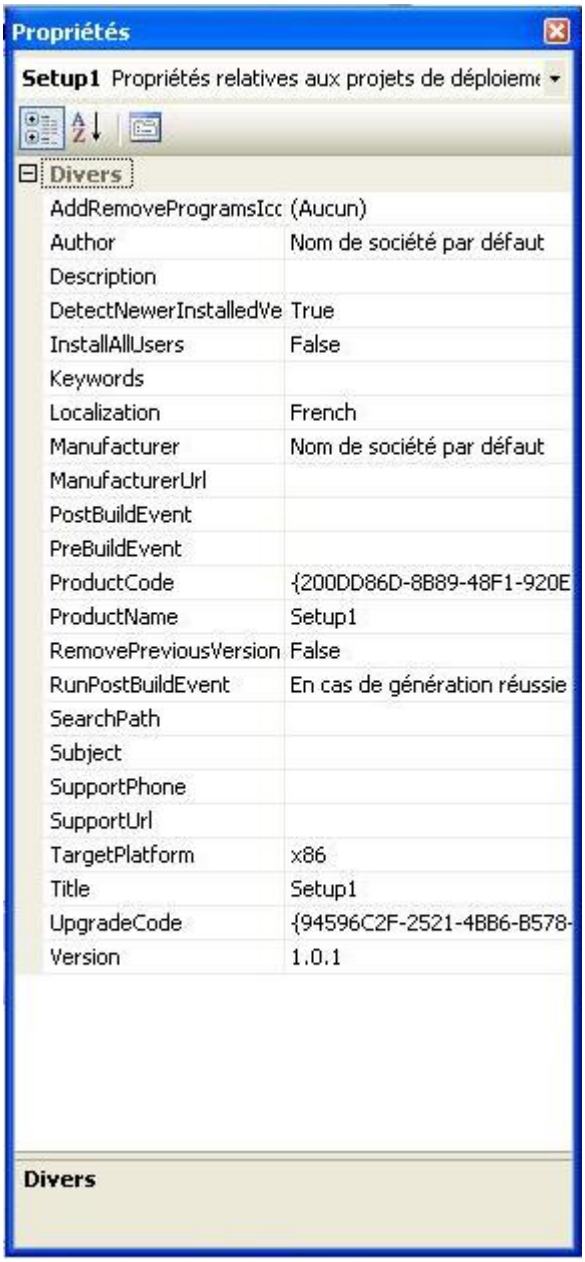
Voila la création de l'installation est terminée ! Mais il reste comme même à configurer quelques petites choses !

VI-B - Faire connaître son application

Vous apercevez un nouveau projet dans votre solution. Ceci est le projet de l'installation de l'application.



Cliquer sur ***Setup1*** (le nom de votre installation) puis appuyer sur ***F4*** (fenêtre propriétés).



La ligne ***AddRemoveProgramsIcons*** permet d'affecter une icône qui s'affichera lors de la désinstallation de votre programme.

Dans ***Author***, mettez le nom de votre société ou votre propre nom.

Dans ***Manufacturer***, mettez votre nom.

ProductName permet de donner un nom à votre programme (exemple : Hello World).

Mettez la valeur True dans ***RemovePreviousVersion*** (si vous faites des mises à jour de votre programme, l'ancien programme sera effacé lors de l'installation de la mise à jour).

ProductName permet de donner un nom à votre installation genre *Installation de Hello World*. Enfin dans Version, mettez le même numéro de version que dans votre programme. Exemple : Si la version de votre programme est 1.2.3 alors mettez 1.2.3 dans la case version. Cliquer sur ***Oui*** en réponse au message d'avertissement.

Le ***ProductCode*** à changé, cela est normal. En effet, pour détecter les versions d'un même programme, le programme d'installation regarde le ***ProductCode*** de l'ancienne installation. Si vous avez spécifié *True* dans ***RemovePreviousVersion*** alors le programme d'installation effacera automatiquement l'ancienne version (pratique non ?)

Faites un clique droit sur ***Setup1*** et choisissez ***Affichage*** et ***Interface Utilisateur***.

Ici, ce sont les fenêtres de votre installation qui sont référencées. Ne touchez rien dans *Installation d'administration*. Cliquer sur ***Bienvenue*** et faites ***F4***. Vous pouvez ici modifier le texte de bienvenue ou alors le copyright et même ajouter une bannière. Configurer le à votre guise ! Vous pouvez répéter l'opération pour les fenêtres *Dossier d'installation*, *Confirmer l'installation*, *Progression* et *Terminé*.

Une option intéressante est l'ajout de nouvelles fenêtres. Pour cela, choisissez la catégorie où vous souhaitez ajouter une boite de dialogue (*Début*, *Progression* ou *Fin* toujours dans *Installer* !!), faites un clique droit et sélectionner ***Ajouter une boite de dialogue***. Faites vos choix. Je n'expliquerais pas les différentes boites de dialogue puisqu'elles sont expliquées dans l'explorateur de Visual Studio (cliquer sur une boite de dialogue et la description apparaît en bas). De plus, les noms sont assez explicites.

Enfin, si vous souhaitez modifier l'ordre d'apparition des fenêtres lors de l'installation, clique droit sur la boite de dialogue et sélectionner ***Monter*** ou ***Descendre*** selon votre convenance.

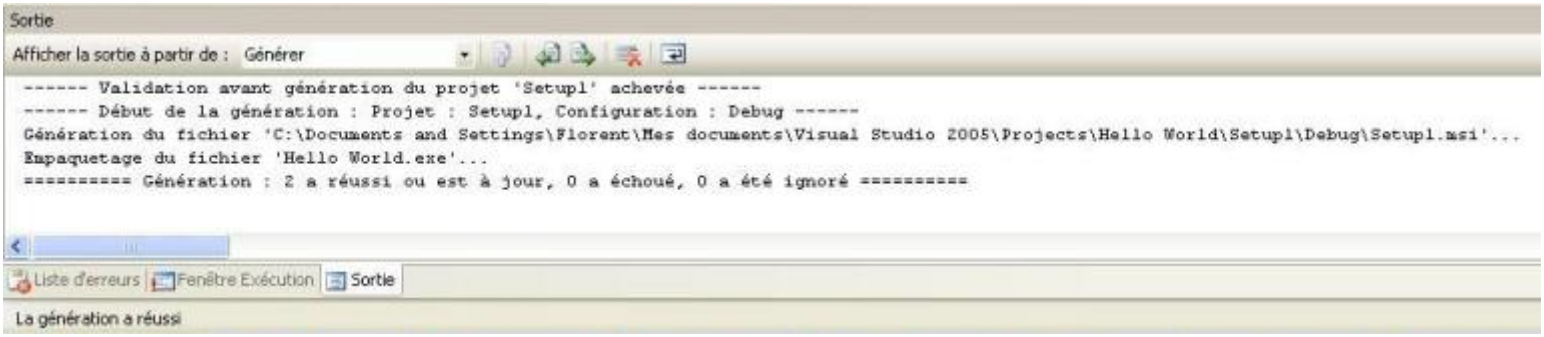
VI-C - Créer un raccourcie pour votre application

Faites un clique droit sur ***Setup1*** et choisissez ***Affichage*** puis ***Système de Fichier***. Cliquer sur ***Dossier d'application***. Dans le volet de droite, faites un clique droit sur la sortie principale du projet et sélectionner ***Créer un raccourci***. Renommer votre raccourci puis glisser le dans *Menu Programmes de l'utilisateur* et dans *Bureau de l'utilisateur* volet de gauche). Vous l'avez déjà compris, le raccourci s'affichera sur le bureau de l'utilisateur (Bureau de l'utilisateur) et dans le menu démarrer (Menu Programmes de l'utilisateur)

Enfin, pour affecter une icône à votre raccourci, sélectionner le puis clique droit et aller dans ***Propriétés***. Cliquer sur ***Aucun*** dans *Icon* et sélectionner ***Parcourir*** dans le menu déroulant. Aller chercher votre icône ou ajouter la si cela n'est déjà fait. Ne toucher pas aux autres propriétés.

*Si vous avez oublié d'ajouter un fichier à votre installation, pas de panique ! Clique droit sur **Setup1** (et oui encore là !!) et faites **Ajouter** puis **Fichier**.*

Voilà c'est vraiment fini, c'était si dur que ca ? Maintenant n'oubliez pas de générer votre projet. Rien de plus simple, clique droit sur ***Setup1*** (décidément y'en a marre...) et choisissez ***Générer***. La génération s'affiche dans la fenêtre de Sortie. La barre de statut vous informe que la génération à réussi.



Le setup est créé dans le dossier de votre solution et contient un fichier Msi et Exe. Celui que vous distribuerez à vos amis est le fichier **Msi**.

*Il existe deux modes de génération Debug et Release. Si votre application est en Beta-Test, choisissez le mode Debug sinon si votre application est fini et comporte aucun bugs, sélectionner Release. Pour changer le mode de génération, clique droit sur **Setup1** puis **Propriétés**. En haut à gauche, dans configuration, choisissez **Release** ou **Debug**. Faites une nouvelle génération puis prendre en compte les modifications.*

VII - Remmerciements et liens intéressants

Je vous conseille d'aller jeter un coup d'oeil sur l'excellent **cours de Plasserre** sur la Programmation vb.net Orientée Objet.

Vous avez un soucis technique ? Rendez vous sur le **forum de Developpez**, club d'entraide des développeurs.