INSTITUT PRIVE D'ENSEIGNEMENT TECHNIQUE

Programmation Evénementielle

Visual Basic .NET

Cours & Exemples pratiques

LA VOIE INSET

Formateur : Sirraj Jaber

INTRODUCTION À VISUAL BASIC .NET

Avec MS ACCESS Vous avez étudié les techniques de modélisation et de création d'une base de données relationnelle en mode autonome (utilisée par une personne à la fois sur un PC). Cependant, dans la vraie vie les besoins sont beaucoup plus complexes que ça. Dans ce cours nous allons commencer à explorer les applications complexes. Nous allons étudier et appliquer les concepts tels que: la Programmation evenementielle, l'architecture client-serveur et le développement "object-oriented".

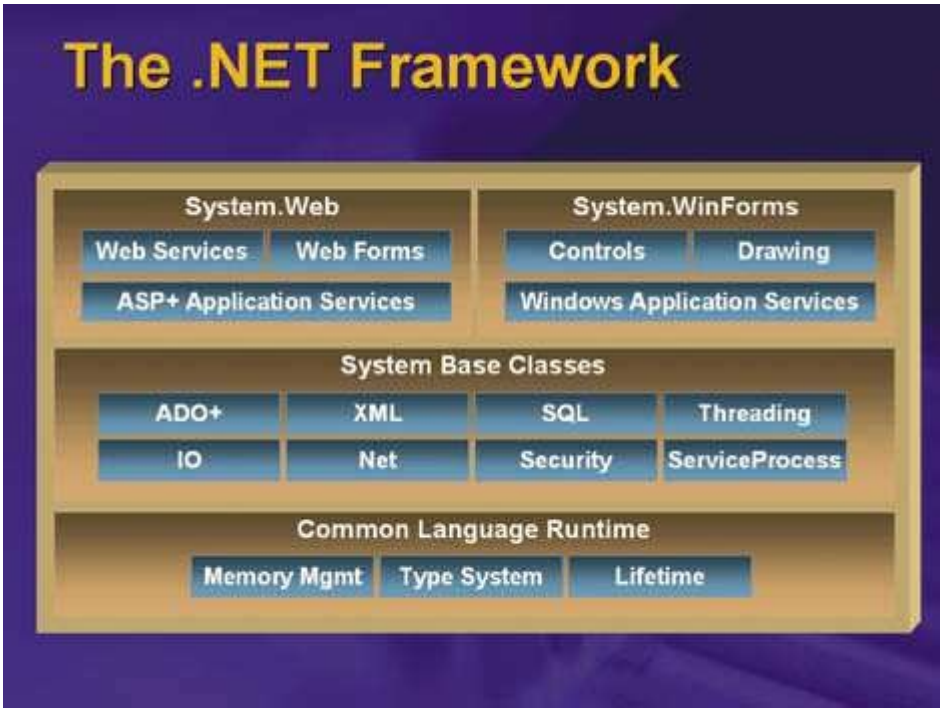
Le langage utilisé pour apprendre à développer les applications est Visual Basic.

La version la plus récente de VB fait partie de la suite Visual Studio .NET introduit l'an dernier. Le fait d'inclure VB dans la suite intégrée va permettre aux programmeurs de développer des applications dans des langages différents, VB et C++, par exemple, en utilisant des outils communs.

Visual Studio .NET

- Un environnement pour développer des applications pour Windows et pour le Web.
- Basé sur le .NET Framework - un ensemble d'outil entre le Windows et l'application.
  - Framework consiste de deux blocs fondamentaux:
    - le Common Language Runtime qui permet d'écrire des applications composées de différents langages: VB, C++, COBOL, etc.

    - les System Base Classes, une librairie qui contient les classes(objets) qui peuvent être utilisées dans diverses applications.



- On peut créer des applications de type Windows pour faire des calculs, accéder à une base de données, etc. On peut aussi créer des applications Web qui sont accessibles par le biais de l'Internet.

Installation de Visual Studio .NET

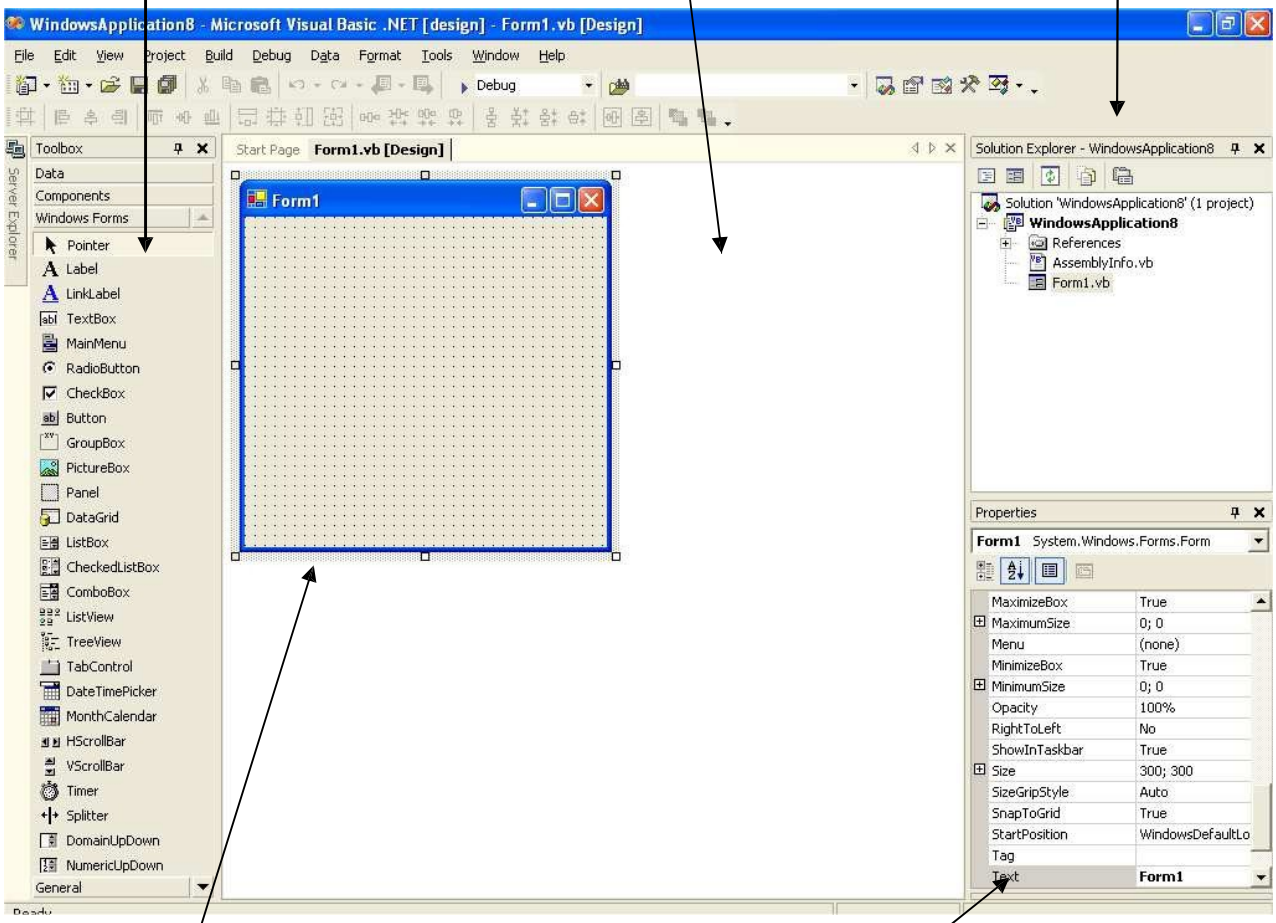
- Il faut la configuration appropriée: au moins Win 2000 (de préférence Win XP), 256 meg de RAM, 2-3 gig d'espace-disque.
- On installe d'abord le Framework (1 cd).
- On installe Visual Studio (2 cd) - on peut choisir quels langages on veut inclure (VB, C++, C#, J#) mais pas des langages commme COBOL à ce moment-ci.
- On installe MSDN (3 cd) pour toutes les références, l'aide, etc.

SECTION 0 : L'environnement VB

Espace de travail VB

Boîte à outils (*Toolbox*)
(non active = barre grise)

Explorateur de solution



Un formulaire (Form) sélectionné
de nom 'Form1'

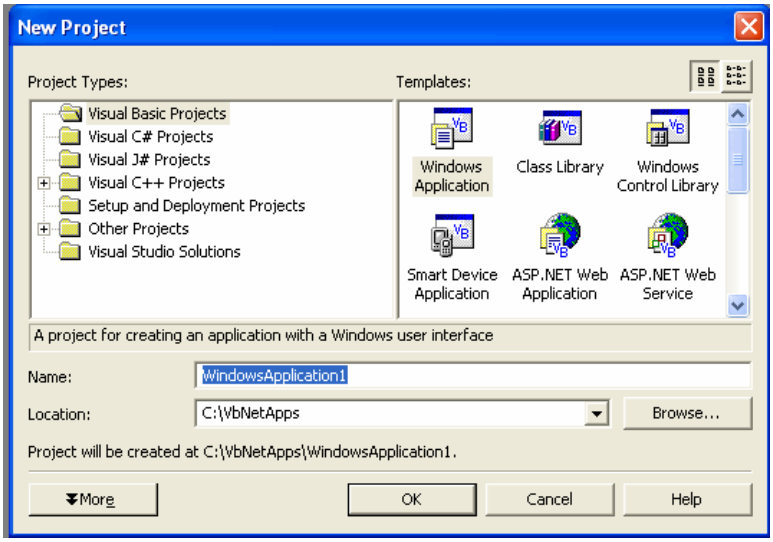
Fenêtre des propriétés (Property window). *Propriétés de l'objet sélectionné (Form1)*

Figure 1: Environnement de développement Visual Basic .NET

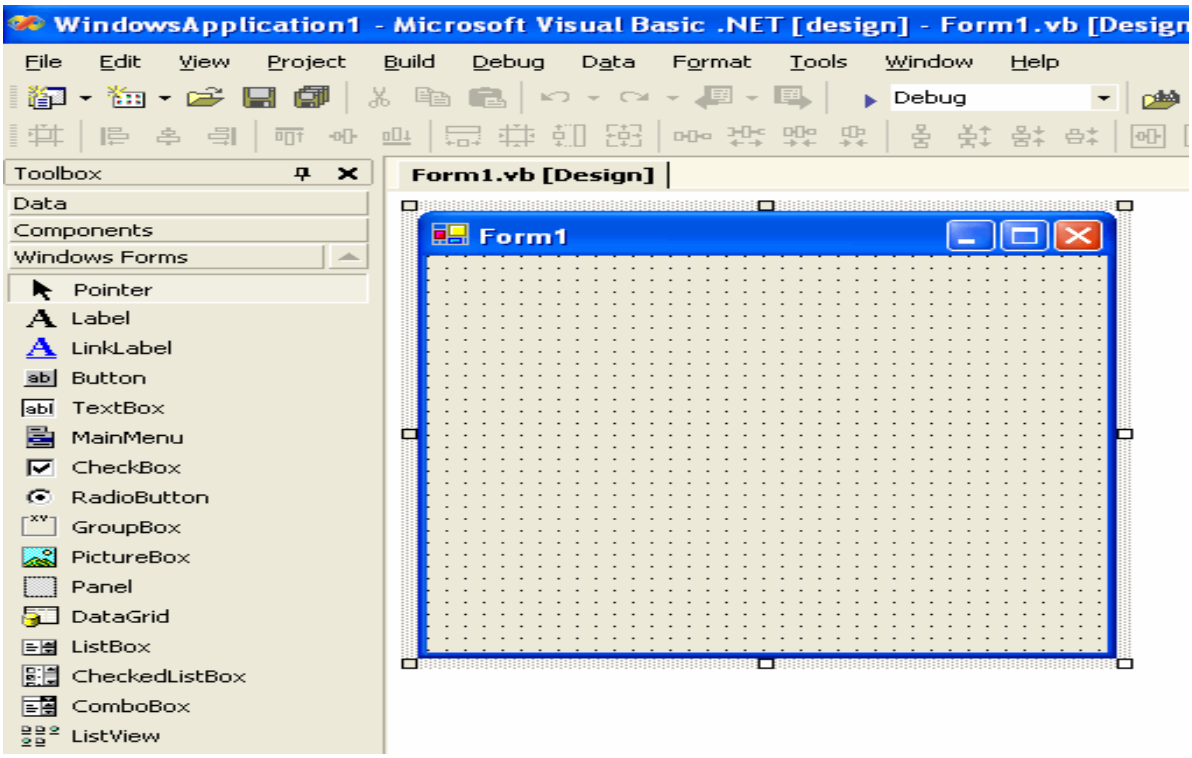
APPLICATION:

Série d'objets (fenêtres, programmes, menus, etc.) qui travaillent sur un même sujet. On appelle l'application une Solution et parfois un Project (Project est le nom qu'on donnait à une application en VB 6). La Solution ScoreBoard servira à manipuler les données pour des équipes de hockey. On pourrait créer un Projet Vidéo pour gérer les opérations d'un magasin de vidéos.

En démarrant VB on doit choisir le langage et on décide de travailler sur un projet existant ou d'en créer un nouveau. Il y a différentes sortes de projets mais, pour l'instant nous allons créer un Windows Application . Le projet sera sauvegardé dans un dossier portant le nom du projet. Vous devriez créer un répertoire VBapps sur votre N: pour tous les exercices VB que vous allez réaliser.



Tout d'abord, remarquez qu'en lançant VB vous avez une première feuille, un Form , qui s'ouvre pour vous. Le form est l'objet le plus visible de VB. On l'utilise pour créer l'interface avec l'utilisateur. Pour créer un form on y place des Controls tels que ceux du Toolbox à la gauche de l'écran. En vous familiarisant avec l'interface VB vous verrez aussi que vous pouvez personnaliser plusieurs des fonctions d'édition de la feuille en allant au menu Tools --> Options .



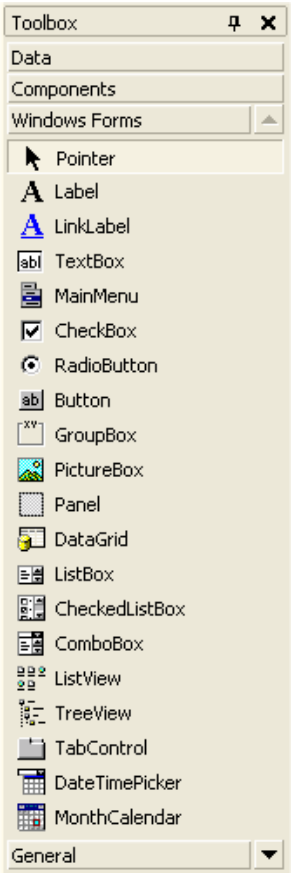


Figure 8 : Toolbox

La partie graphique de votre application va contenir un (ou plusieurs) formulaire(s). Sur un formulaire, on peut placer un ou plusieurs *objets graphiques* ou ce qu'on appellera des **contrôles les** (Bouton à cliquer, Champ libellé (texte statique), Champ texte à saisir au clavier, Menu, etc.).

Ces contrôles sont des objets pré-programmés dont l'utilité principale est de faciliter l'*interaction* avec l'utilisateur. Chacun de ces objets graphiques a une fonctionnalité bien précise. Le tableau suivant résume les contrôles standards de base les plus utilisés:

Contrôle	Nom du contrôle	Utilité
	Label	Afficher un texte statique : un libellé
	Text Box	Afficher et rentrer une valeur au clavier
	Button	Lancer l'exécution une procédure événementielle
	ListBox	Afficher une liste statique de valeur
	ComboBox	Combiner l'utilité des contrôles **TextBox** et **ListBox**
	PictureBox	Afficher une image dans un cadre. Celui-ci peut être redimensionné en fonction de l'image (*Autosize = True*)
	RadioButton	Sélectionner une option. Si utilisé en plusieurs instances (*Option Button*), une seule peut être choisie
	Check Box	Sélectionner une option. Si utilisé en plusieurs instances (*Check Box*), une ou plusieurs peuvent être choisies
	GroupBox	Créer une fenêtre au sein d'un formulaire et créer un groupe de contrôles.

Les contrôles standards dans VB se trouvent dans la Boîte à outils (*ToolBox*), voir figure 8. D'autres contrôles plus élaborés (*Components*) peuvent être ajoutés dans la boîte à outils, en sélectionnant dans la barre du menu : *Project, Add Components*.

Comment placer un contrôle sur un formulaire ?

Sélectionnez dans la boîte à outils le contrôle désiré. Dessinez sur le formulaire le rectangle dans lequel vous voulez placer le dit contrôle. Pour ce faire, cliquez (sans relâcher) sur le bouton gauche de la souris, sur le coin haut gauche du rectangle et déplacez la souris vers le coin bas droit du rectangle puis relâchez le bouton de la souris. Le contrôle apparaît par magie sur le formulaire.

Comment déplacer un contrôle ou le redimensionner ?
Sélectionnez d'abord (en cliquant dessus) le contrôle placé sur le formulaire. Glissez le vers l'endroit désiré ou cliquez et tirez sur l'un des huit petits carrés bleus délimitant l'objet sélectionné (le contrôle **Label** *Euro est sélectionné dans la figure 8).*

Chaque contrôle peut être vu comme un objet défini par un ensemble de propriétés. Quand un contrôle, placé sur un formulaire, et sélectionné, ses propriétés apparaissent dans la fenêtre **Properties**.

De manière générale, une fois qu'un *objet* est sélectionné, ses propriétés apparaissent dans la fenêtre **Properties**. Notez que certains objets (contrôles, formulaires, etc.) ont les mêmes propriétés, par exemple *Name* et *Text*, mais ont **bien évidemment** des valeurs différentes qui leurs sont propres.

Pratique. Placez les contrôles suivants sur le formulaire qui a été créé:

Contrôle	Propriété	Valeur
Label	Text	FB
TextBox	Name	Franc_Belge
Label	Text	Euro :
CommandButton	Text	Convertir
CommandButton	Text	Sortir

Modifiez leurs positions et leurs propriétés pour que le formulaire apparaisse comme suit :

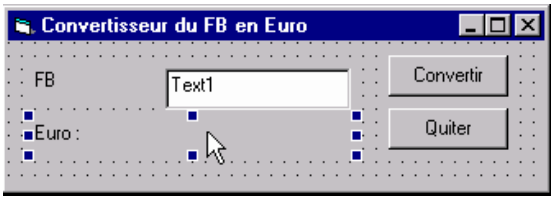


Figure 9 : Exemple de conception d'un formulaire

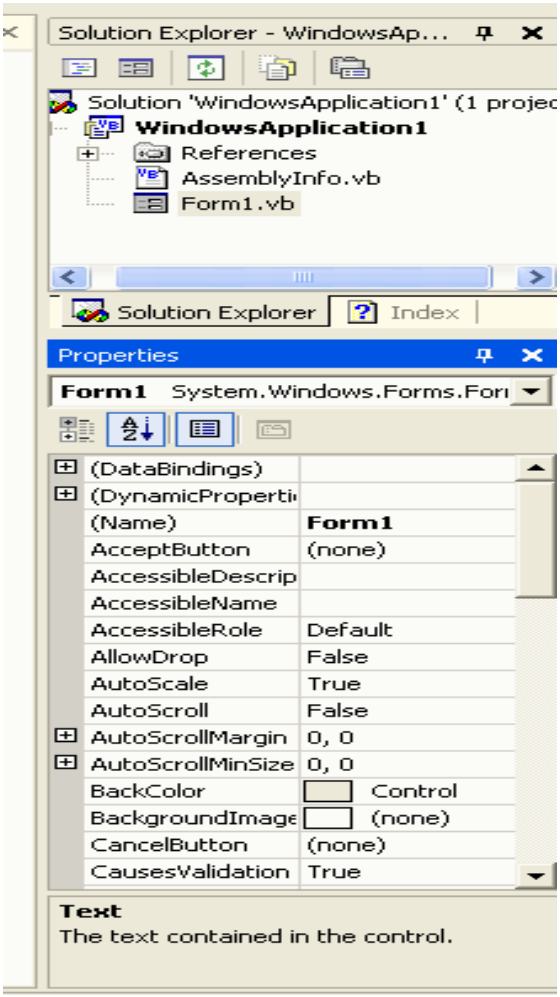
Nous venons de créer un programme (graphique et statique) sans écrire aucune ligne de *code* VB. Celui-ci peut d'être exécuté. Ce programme ne fait qu'afficher une boite de dialogue ayant la forme de la figure 9. Pour lui donner vie (lui faire faire ce qui est désiré), c'est-à-dire convertir du *Franc Belge* en *Euro*, il faudra mettre la main à la pâte et écrire du code VB.

Comment exécuter votre programme ?
Pour exécuter un programme, appuyez sur la touche **F5** *ou sélectionnez dans la barre de menu,* **Run, Start**, *ou cliquez sur le bouton* **Start**.

Comment arrêter l'exécution d'un programme ?

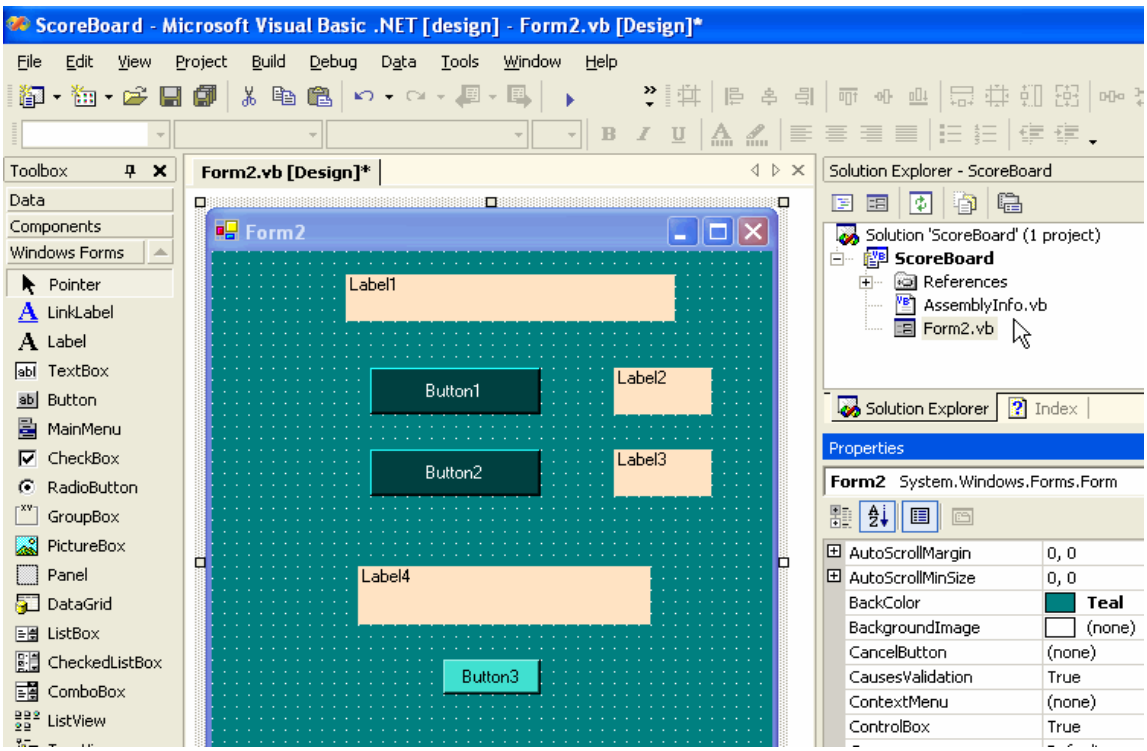
Pour arrêter l'exécution de votre programme, cliquez sur la petite croix située en haut à droite du formulaire ou cliquez sur le bouton End.

Il est important de noter que le côté droit de l'écran contient deux fenêtres utiles: le Solutions Explorer qui nous donne les détails de l'application courante et la fenêtre Properties qui décrit les propriétés de chaque objet.



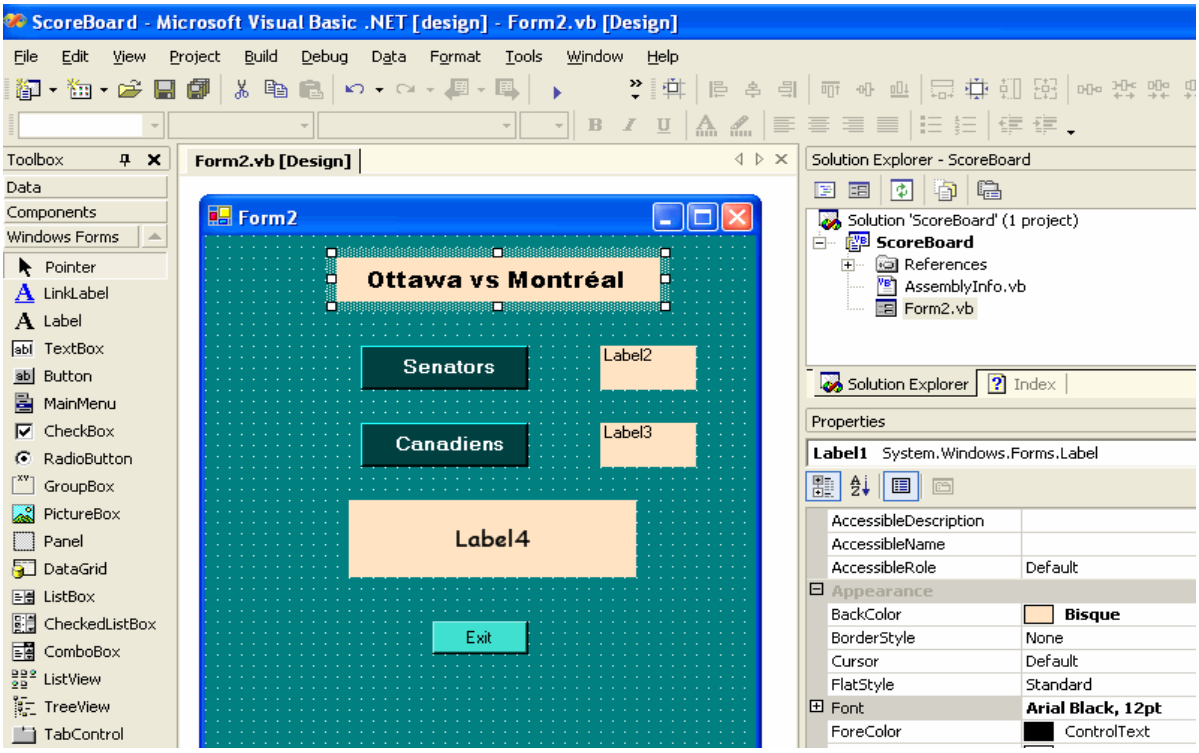
Premier exercice - Le ScoreBoard

Comme premier exercice en VB vous allez créer le form que vous voyez dans l'image suivante:



Vous devez mettre sur la feuille 3 Button et 4 Label . Ces contrôles devraient être alignés à peu près comme l'illustration. Une fois les contrôles placés vous pouvez expérimenter avec la fenêtre Properties . La fenêtre reste ouverte et à chaque fois que vous cliquez sur un contrôle vous voyez les propriétés de ce contrôle s'afficher. Expérimentez avec Text, TextAlign, BackColor et ForeColor . Ne changez pas (Name) pour l'instant. Remarquez aussi que les propriétés ne sont pas les mêmes pour un Button et pour un Label - tous les contrôles, incluant la feuille elle-même, ont des propriétés différentes.

Après quelques manipulations des propriétés, votre form pourrait commencer à avoir l'air de ceci:



Ecrire le code VB

Pour ouvrir l'éditeur de code on fait un double clique sur la feuille ou on clique sur l'onglet Form1.vb.

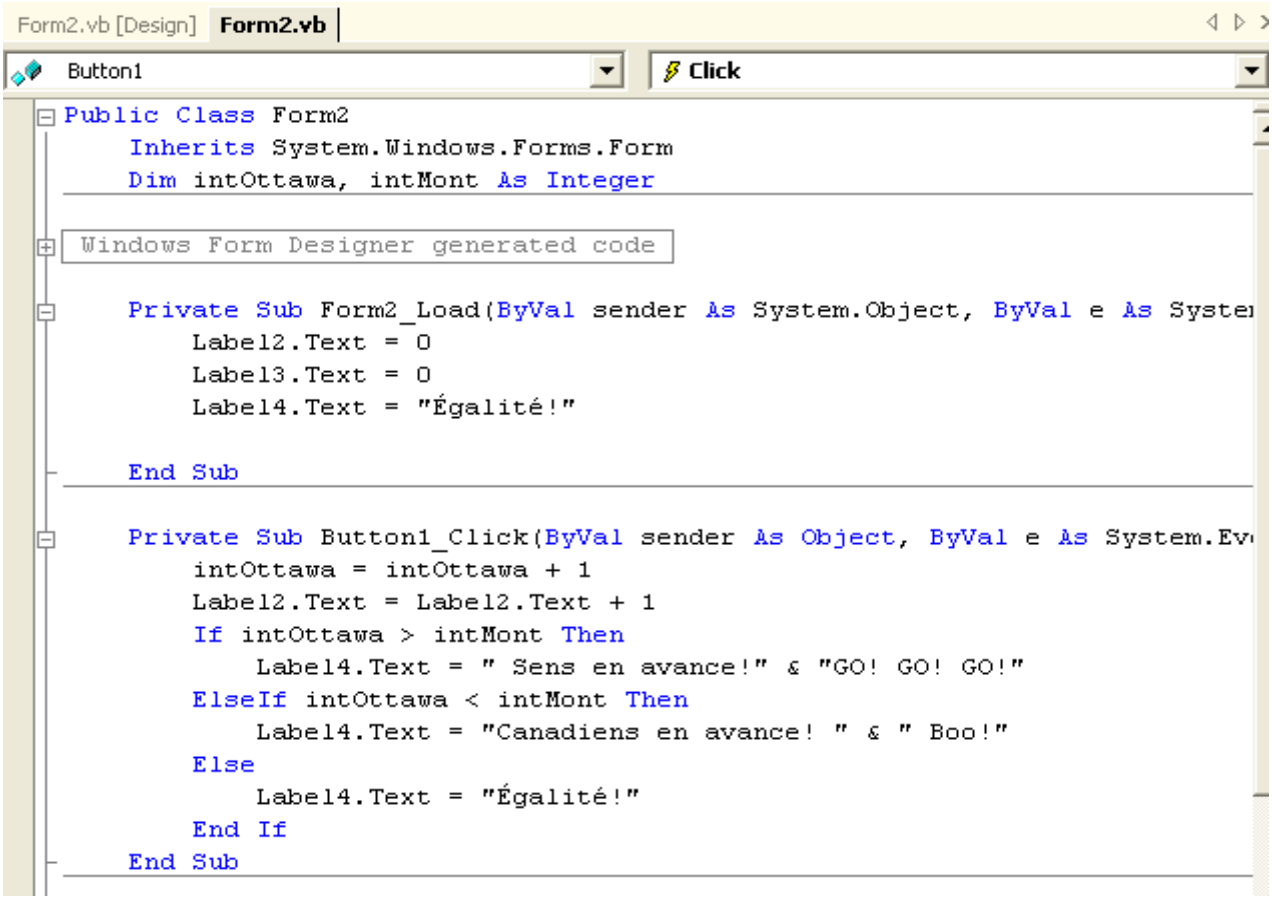
La première chose à savoir est qu'on aura besoin de 2 compteurs dans le programme. Comme vous savez, un compteur est simplement une variable de type numérique. On doit donc déclarer ces variables. On le fait dans la section General Declarations pour que les variables soient accessibles à tous les objets de la feuille. On pourrait déclarer chaque variable à l'intérieur d'une procédure - un Private Sub - mais ces variables seraient alors locales et ne seraient valables que pour la procédure où elles sont nommées. Nous verrons les détails concernant les différents types de variables au prochain cours.

Notre application ne contient encore qu'un seul objet: le form Form1.

Quand on voudra lancer l'application, faire un Debug --> Start ou F5 . La première action qui va se passer est que la form va s'ouvrir. L'action d'ouvrir est un event , dans ce cas, le Open event for Form Form1 . En programmation VB on écrit toujours du code pour des events . Donc, si on veut exécuter certaines tâches lors de l'ouverture de la feuille, comme initialiser des variables locales ou donner des valeurs de départ aux propriétés des contrôles, on programme le Open de la feuille. Dans l'éditeur de code on choisi Form dans le premier ListBox et Open dans le deuxième ListBox, ce qui génère une procédure Private Sub Form1_Load(...) .

Maintenant on code les actions qu'on veut voir lorsqu'on clique sur un bouton de commande. Encore on invoque l'éditeur soit en faisant un double-clique sur le bouton lui-même ou dans l'éditeur, en choisissant le nom du bouton, Command1 dans le premier ListBox et l'événement Click dans le deuxième ListBox. Cela génère une nouvelle procédure, Private Sub Button1_Click(...).

A tout moment on peut tester l'application en faisant F5. Si le résultat n'est pas satisfaisant, on revient au mode Design et on modifie l'interface.



Une fois qu'on a maitriser la technique pour coder le premier bouton, on peut coder le deuxième facilement. Cependant, on doit faire les changements appropriés pour la situation.

Le troisième bouton, Exit, sert à arrêter l'exécution de l'application. La commande nécessaire est End.

SECTION 1 : STRUCTURES DE BASE

Dans cette section nous allons présenter les structures de base de la programmation en VB. Nous allons d'abord présenter la notion de variable, les différents types standards, les opérateurs arithmétiques, les structures usuelles: structures de contrôles (Instructions conditionnelles et répétitives), les structures de données élaborées (vecteurs et matrices) et finalement, les procédures et les fonctions.

Quelques remarques préliminaires :

- Cette section présente les structures théoriques de VB. Il est conseillé de la parcourir une première fois (*sans trop insister*) pour avoir une idée des structures de base de VB. Au fur et à mesure que vous avancerez dans le cours, vous devez y revenir pour approfondir la partie qui vous intéresse.
- Votre application VB sera composée d'un ou plusieurs fichiers (**formulaires et modules**) contenant du code VB.
- Il n'existe pas de séparateurs d'instructions en VB (comme le ' ; ' en *Pascal* ou en *C* et le '. ' en *Cobol*).
- VB ne fait pas de distinction entre les minuscules et les majuscules. Ainsi, *ValeurVariable* et *vALEURvARIABLE* représentent la même variable. En fait, VB est '*très intelligent*', en ce sens qu'il vous réécrira (automatiquement) la variable dans le format de caractères que vous avez utilisé lors de sa déclaration.

1.1 Variable et opérations arithmétiques

1.1.1 Notion de Variable

Les variables sont nécessaires pour **stocker** (conserver) une valeur dynamique et réutilisable. C'est en fait une simple **zone mémoire** qui porte un nom choisi par le programmeur. Le nom de la variable est une **adresse mémoire**. Si l'on veut une programmation cohérente, il faut déclarer chaque variable en précisant le type de celle-ci. La déclaration se fait avec le mot réservé **Dim**.

Syntaxe

```
Dim NomVariable As Type
```

Pour la lisibilité du code, on peut ajouter un commentaire après une apostrophe (**'**)

Exemple

```
                                ' Ceci est un commentaire
Dim Taux As Single              ' Taux de la TVA
Dim Réponse As String           ' Mot proposé par le joueur
```

Par défaut, le compilateur VB considère que toute variable qui apparaît doit avoir été déclarée. Toutefois, si vous ajoutez dans votre code la ligne

```
Option Explicit Off
```

VB sera **permissif** et vous autorisera à utiliser des variables sans les déclarer. Prenez **la très bonne habitude** de toujours déclarer vos variables.

Pour éviter tout problème il est préférable d'initialiser les variables déclarées.

Exemples

```
Compteur = 0        ' = est le symbole d'affectation en VB
Taux = 21
```

Le langage VB utilise **plusieurs types** de données dont les plus utilisés sont le type **String** (chaîne de caractères), le type **Integer** (entier) et le type **Single** (décimal). Les types standards de données en VB sont résumés dans le tableau ci-dessous.

Types standards de données

Opérateur	Plage de valeurs	Déclaration et affectation
`Integer`	Nombres entiers de –32 768 à +32 767	`Dim Nb As Integer` `Nb = 100`
`Single`	Nombres réels avec précision de sept décimales Valeurs négatives : de –3,402823^{E}38 à –1,401298 ^{E}-45 Valeurs positives: de 1,401298^{E}-45 à 3,402823E38	`Dim Mt As Single` `Mt = 45.11`
`String`	Chaîne de caractères pouvant aller jusqu'à 65535 caractères (environ 2 milliards si la longueur est variable)	`Dim as String` `Prénom = "Jean"`
`Long`	Nombres entiers de –2 147 483 648 à +2 147 483 647	`Dim Profit As Long` `Profit = 123 465 789`
`Double`	Nombres réels avec précision de seize décimales Valeurs négatives : De –1,79769313486232^{E}308 à –4,94065641247^{E}-324 Valeurs positives: De 4,94065641247^{E}-324 à 1,79769313486232^{E}308	`Dim DblPrec As Double` `Mt = 1.23456789012`
`Byte`	Nombres entiers de 0 à 255	`Dim BitPattern As Byte` `BitPattern = 128`
`Boolean`	Vrai ou faux (valeur logique)	`Dim Test As Boolean` `Trouvé = True`
`Date`	De 1/1/100 à 31/12/9999	`Dim JourPlus As Date` `JourPlus = "06/06/44"`
`Currency`	Nombres entiers de – 922337203685477,5808 à 922337203685477,5808	`Dim Valeur As Currency`

1.1.2 Opérateurs arithmétiques

VB reconnaît les opérateurs arithmétiques usuels qui sont résumés dans le tableau suivant :

Opérateurs arithmétiques

Opérateur	Description	Exemples
+, -	Addition et soustraction	12 + 34; 87.56 – 387.222
*	Multiplication	45.87 * 4
/	Division décimale	36 / 25 = 1.44
^	Puissance	5 ^ 3 =125
\	Division entière	36 \ 25 = 1
MOD	Modulo (reste de la division entière)	36 MOD 25 = 11

Si, dans une expression arithmétique plusieurs opérateurs sont utilisés, les priorités sont résolues comme indiqué dans le tableau qui suit :

Priorité des opérateurs arithmétiques

Opérateur	**Description**	**Priorité**
( )	Parenthèses	1
^	Puissance	2
-	Négation	3
* , /	Multiplication et division	4
\	Division entière	5
MOD	Modulo	6
+, -	Addition et soustraction	7

1.2 Instructions conditionnelles

Les deux instructions conditionnelles le plus utilisées en VB sont **If** et **Select Case.**

1.2.1 If … Then … Else … End If

Si la condition se trouvant après le mot réservé **If** est *vraie*, les instructions qui suivent le mot réservé **Then** sont exécutées sinon, ce sont celles qui suivent le mot réservé **Else** qui sont exécutées. L'instruction **If** se termine (obligatoirement) avec les mots réservés **End If**.

Forme simple :

Syntaxe

```
If condition(s) Then
       Instruction11
       Instruction12
       …
Else
       Instruction21
       Instruction22
       …
End If
```

Exemple

```
If Moyenne >= 12 Then
       Admis = Admis + 1
       MsgBox(" Candidat admis ")    ‘ affiche une fenêtre avec le message indiqué
Else
       Ajournés = Ajournés + 1
       MsgBox(" Candidat ajourné ")
End If
```

Forme imbriquée

Syntaxe

```
If condition(s) Then
      Instruction11
      If condition Then
            Instruction12
      Else if condition Then
            Instruction13
      Else
            Instruction14
      End If
      …
Else
      Instruction21
      Instruction22
      …
End If
```

Exemple

```
If NombreProposé > NombreATrouver Then
      MsgBox("Votre nombre est trop grand !")
ElseIf NombreProposé < NombreATrouver Then
      MsgBox("Votre nombre est trop petit !")
Else
      MsgBox("Gagné !")
End If
```

Opérateurs de comparaison

Opérateur	Signification	Exemple	Résultat
=	Egal à	15 = 11 + 4	True
>	Supérieur à	17 > 11	True
<	Inférieur à	17 < 11	False
<>	Différent de	23 <> 23.1	True
>=	Supérieur ou égale à	23 >= 23.1	False
<=	Inférieur ou égal à	23 <= 23.1	True

Si plusieurs conditions doivent être testées, celles-ci doivent être combinées avec des opérateurs logiques. VB accepte les opérateurs logiques suivants: *AND, OR, NOT et XOR*. La signification de chacun d'eux est présentée dans le tableau qui suit:

Opérateurs logiques

Opérateur	Signification	Exemple	Résultat
AND	Connexion ET. Il faut que les conditions soient vraies pour que le résultat soit vrai	(1 = 1) AND (2 < 4) (1 > 2) AND (2 = 4)	True False
OR	Connexion OU. Il faut que l'une des deux conditions soit vraie pour que le résultat soit vrai	(1 = 2) OR (3 < 2) (1 > 2) OR (2 > 1)	False True
NOT	Connexion NON. La valeur logique est inversée	EstCeVrai = True NOT EstCeVrai	 False
XOR	Connexion OU exclusif. Une seule des deux conditions doit être vraie pour que le résultat soit vrai	(1 = 1) XOR (2 = 2) (2 > 1) XOR (3 < 1)	False True

1.2.2 IIf (Condition, ValeurSiVrai, ValeurSiFaux)

Cette instruction (IIF) fonctionne comme le **IF** d'EXCEL.

Syntaxe

```
IIf (Condition, ValeurSiVrai, ValeurSiFaux)
```

Exemple

```
Dim Note As Single
Dim Réponse As String
Note = InputBox (" Tapez votre note ")
Réponse = IIf (Note >= 10, " Admis ", " Ajourné ")
MsgBox (Réponse)
```

1.2.3 Select case … Case … Case …Else Case … End Select

L'instruction **Select Case** est une instruction conditionnelle *alternative*, c'est-à-dire qu'une *expression* peut être testée par rapport à plusieurs valeurs possibles.

Syntaxe

```
Select Case expression
       Case Liste_Valeurs_1
              Instruction11
              Instruction12
              …
       Case Liste_Valeurs_2
              Instruction21
              …
              …
       Else Case
              InstructionElse1
              InstructionElse2
              …
End Select
```

Les instructions se trouvant après '*Case Liste_Valeurs_i'* seront exécutées si '*expression = à l'un des éléments de Liste_Valeurs_i', i = 1, 2, 3, ….* Sinon, les instructions se trouvant après '*Else Case'* seront exécutées. *Liste_Valeurs_i* peut être :

- une suite de valeurs : 1, 3, 5, 7, 9
- une fourchette de valeur : 0 To 9
- une plage de valeur : **Is** >= 10 (Is est un mot réservé)

Exemple

```
Select Case CodeASCIICaractère
       Case 65, 69, 73, 79, 85
              MsgBox(" C'est une voyelle ")
       Case 66 To 90
              MsgBox(" C'est une consonne ")
       Case Else
              MsgBox(" Ce n'est pas une lettre ")
End Select
```

Notez que '*Liste_Valeurs_i'* peut être une combinaison de listes de valeurs comme dans le cas des exemples suivants :

```
Case 1 To 4, 7 To 9, 11, 13, Is > NombreMAx
Case "Lundi", "Mercredi", "Dimanche", VariableJour
```

1.3 Tableaux

Un tableau permet de stocker une suite d'éléments de même type. L'accès à un élément précis se fait à l'aide d'un indice (valeur ou variable entière). En VB, pour un vecteur déclaré avec une dimension (N), le premier élément a l'indice 0, le deuxième a l'indice 2, le troisième a l'indice 3, …, le dernier a l'indice N[1].

Syntaxe

```
Dim NomVecteur(N) As TypeVecteur
```

Cette instruction déclare un vecteur *NomVecteur* de taille *N+1*. Pour accéder au i ème élément du vecteur, il faut préciser l'indice entre parenthèses comme suit : *NomVecteur(i-1)*, i doit être compris dans l'intervalle [0, N].

TypeVecteur est un type standard (Boolean, Integer, String, etc.) ou tout autre type (**type d'objet**) définie dans VB ou dans votre application.

Exemple

```
Dim TabTemp(12) As Single
```

Numéro	1	2	3	4	5	…
Température	6	5,5	7	11,5	15	…

L'accès à la case numéro 3 se fait par TabTemp(3) qui vaut **7**.

Syntaxe

```
Dim NomVecteur(1 To N) As TypeVecteur ‘ déclare un vecteur de N éléments
```

Exemple

```
Dim TabMajuscules(65 to 90) As String
```

Numéro	65	66	67	…	89	90
Majuscule	A	B	C	…	Y	Z

VB permet de travailler avec des tableaux de deux, trois, quatre, dimensions ou plus

Exemple d'un tableau à deux dimensions:

```
Dim ExempleMatrice(10, 10) As Single
```

ExempleMatrice est une matrice (de nombres réels) de 11 lignes et 11 colonnes et où *ExempleMatrice(1, 9)* est l'élément se trouvant à l'intersection de la première ligne et de la dixième colonne [2].

Exemple de déclaration d'un tableau à trois dimensions:

```
Dim ExempleMatrice(10, 10, 10) As Single ‘ matrice à trois dimensions
```

1.4 Instructions répétitives

Les instructions répétitives sont utilisées pour boucler sur une suite d'instructions.

1.4.1 For … To … Next

Si le nombre de boucles est connu à l'avance, on utilise l'instruction *For … To … Next*.

Syntaxe

```
For Compteur = Début To Fin [Step Incrément]
      Instructions
      [ ... Exit For]          ‘ pour une interruption préalable de la boucle
      [Instructions]
Next [Compteur]                ‘ le mot Compteur est facultatif
```

Le test (Compteur = Début) est effectué au début de la boucle. La variable numérique *Compteur* est incrémentée à chaque fin de boucle du nombre indiqué par l'incrément. Si l'*Incrément (le pas par lequel Compteur augmente à chaque boucle)* n'est pas spécifié, il est fixé par défaut à 1.

Si la valeur de *Fin* est inférieure à la valeur de *Début*, l'incrément est négatif. La valeur de *Compteur* peut être utilisée (par exemple, pour numéroter le passage dans la boucle) mais **ne doit pas** être modifiée dans le corps de la boucle.

Exemple

```
      Dim i As Integer
      Dim Chaîne As String
      Dim TabInitial(1 To 12) As Single
      For i = 1 To 12
Chaîne = InputBox("Température N° " & Compteur
      TabInitial(i) = Chaîne
      Next i                              ‘le i n’est pas obligatoire
```

1.4.2 Do While … Loop / Do … Loop While …

Test antérieur

Syntaxe

```
Do While Condition
      Instructions
      [... Exit Do]
      [Instructions]
Loop
```

La condition est ici testée au début, c'est-à-dire à l'entrée de la boucle. Avec *While* (tant que), la boucle est répétée tant que la condition est vraie. Si la condition n'est pas vraie au départ, les instructions de la boucle ne sont pas exécutées.

Exemple

```
Do While MotProposé <> MotDePasse

      MotProposé = InputBox("Donnez votre mot de passe")

Loop
```

Cela présuppose que MotProposé soit initialisé par une valeur autre que MotDePasse (par exemple, la valeur par défaut "").

Test postérieur

Syntaxe

```
Do
	Instructions
	[... Exit Do]
	[Instructions]
Loop While Condition
```

La condition est alors testée à la fin de la boucle. Avec While (tant que), la boucle est répétée tant que la condition est vraie. Les instructions de la boucle sont donc exécutées au moins une fois.

Exemple

```
Do
	MotProposé = InputBox("Donnez votre mot de passe")
Loop While MotProposé <> MotDePasse
```

Cet exemple ne présuppose aucune initialisation de MotProposé.

1.4.3 Do Until … Loop / Do … Loop Until …

Test antérieur

Syntaxe

```
Do Until Condition
	Instructions
	[... Exit Do]
	[Instructions]
Loop
```

La condition est ici testée au début, c'est-à-dire à l'entrée de la boucle. Avec *Until* (jusqu'à), la boucle est répétée jusqu'à ce que la condition soit vraie. Si la condition est vraie au départ, les instructions de la boucle ne sont pas exécutées.

Exemple

```
Do Until MotProposé = MotDePasse
	MotProposé = InputBox("Donnez votre mot de passe")
Loop
```

Cela présuppose que MotProposé soit initialisé par une valeur autre que MotDePasse (par exemple, la valeur par défaut `""`).

Test postérieur

Syntaxe

```
Do
	Instructions
	[... Exit Do]
	[Instructions]
Loop Until Condition
```

La condition est alors testée à la fin de la boucle. Les instructions de la boucle sont donc exécutées au moins une fois. Avec *Until* (jusqu'à), la boucle est répétée jusqu'à ce que la condition soit vraie.

Exemple

```
Do
      MotProposé = InputBox("Donnez votre mot de passe")
Loop Until MotProposé = MotDePasse
```

Cet exemple ne présuppose aucune initialisation de MotProposé.

1.4.4 For … Each … Next

C'est **une extension** de la boucle For ... To … Next.

Syntaxe

```
For Each Elément In Ensemble
      Instructions
      [ ... Exit For]
      [Instructions]
Next [Elément]
```

Ensemble est le plus souvent un tableau.

Exemple

```
Dim TabHasard(100) As Integer
Dim Cellule As Integer
Dim Réponse As String
Randomize                          'initialise le générateur de nombres au hasard
For Each Cellule In TabHasard
      Cellule = Rnd * 100 + 1      'génère un nombre au hasard entre 1 et 100
Next
For Each Cellule In TabHasard
      Réponse = Réponse & Cellule & " " 'Concaténation de chaînes de caractères
Next
MsgBox (Réponse)
```

1.4.5 Conclusion

Selon le problème à traiter, vous aurez **le choix** entre ces différentes structures de contrôle. Il s'agira de choisir **la plus élégante** ou du moins, celle qui ne provoquera pas de dysfonctionnement de votre programme.

Trouvez les erreurs dans les exemples suivants :

Exemple 1:

```
Dim VotreRéponse As String
Réponse = "LaRéponse"
Do
      VotreRéponse = InputBox("Donnez votre réponse")
Loop While VotreMot = Réponse
```

Exemple 2

```
Dim Cote As Single
Do Until Cote >= 0 And Cote <= 20
      Cote = InputBox("Taper une note entre 0 et 20")
Loop
```

1.5 Procédures et Fonctions

Comme dans le cas du langage *Pascal*, VB .NET permet l'utilisation des procédures et des fonctions avec ou sans paramètres. Rappelez vous que la grande différence entre la procédure et la fonction est que cette dernière retourne une valeur lorsqu'elle est appelée.

Lors de l'appel de la procédure, un paramètre peut être transmis soit par valeur, soit par **référence** (variable).

1.5.1 Procédure (Transmission par valeur : ByVal)

Pour transmettre un paramètre par valeur, celui-ci doit être **obligatoirement** précédé par le mot réservé **ByVal.** Sinon, il est considéré de passer par référence.

Syntaxe

```
Private Sub NomProcédure( ByVal argument As Type, … )
      Instruction1
      Instruction2
      …
End Sub
```

Exemple

```
Private Sub Affectation( ByVal valeur1,valeur2 As integer)
      Dim Chaîne As String
      Chaîne = "La somme de " & valeur1 & " et " & valeur2 & " = "
      valeur1 = valeur1 + valeur2
      Chaîne = Chaîne & valeur1
      MsgBox (Chaîne)
End Sub
```

L'appel de la procédure se fait soit en inscrivant **call** suivi du nom de la procédure, et des paramètres à lui transmettre, soit en écrivant uniquement le nom de la procédure, suivi des paramètres à lui transmettre.

```
Dim X As integer
Dim Y As integer
Call Affectation (X, Y)    ' avec les parenthèses
MsgBox (" Et X = " & X & " n'a pas changé ")
```

1.5.2 Procédure (Transmission par référence : ByRef)

Si **ByVal** n'est pas précisé ou si le paramètre est précédé par le mot réservé **ByRef,** la variable est transmise par référence (c'est-à-dire transmise en tant que **variable).** Ainsi, toute modification de la variable locale correspondante dans la procédure se répercute sur la variable utilisée lors de l'appel. VB suppose que la transmission se fait par référence si le mot réservé **ByVal** est omis.

Exemple

```
Private Sub Transvase ( valeur1 As Integer, valeur2 As Integer )
      Dim variable As Integer
      variable = valeur1
      valeur1 = valeur2
      valeur2 = variable
End Sub
```

L'appel suivant transvase le contenu de X dans Y et inversement.

Exemple

```
Dim X As Integer, Y As Integer
X = 100
Y = 200
MsgBox (" X = " & X & " et Y = " & Y)
Transvase(X, Y)
MsgBox (" Alors que maintenant X = " & X & " et Y = " & Y)
```

1.5.3 Fonction

Lors de la déclaration d'une fonction, la valeur qui doit être retournée par celle-ci doit être affectée *au nom de la fonction*. La déclaration de la fonction se termine par les mots réservés *"End function"*.

Syntaxe

```
Private function NomFonction( Argument As Type, … ) As Type
      Instruction1
      Instruction2
            …
      NomFonction = RésultatDeLaFonction
End function
```

Exemple

```
Private function Somme( valeur1 As Integer, valeur2 As Integer ) As integer
      Somme = Valeur1 + valeur2
End function
```

L'appel suivant retourne la somme de X et Y et affecte le résultat à la variable Z.

Exemple

```
Dim X As Integer, Y As Integer, Z As Integer
X = 10
Y = 20
Z = somme(X, Y)
```

1.5.4 Transmission d'un tableau comme argument d'une procédure ou d'une fonction

Pour transmettre un tableau comme argument d'une fonction ou d'une procédure, il suffit de déclarer (à l'intérieur des parenthèses) une variable (le nom local du tableau) *sans type, ni dimension*. Lors de l'appel de la fonction ou de la procédure, VB donne à cette variable le type et la taille du tableau envoyé. On peut aussi utiliser comme type de la variable locale, le type *Variant*. Comme tout variable, un tableau peut être envoyé par valeur ou par référence.

Ci-après vous trouvez un exemple de déclaration d'une procédure qui reçoit un vecteur (passation par référence: par défaut).

Exemple

```
Private Sub Init(vec)                    ' ou Private Sub init(vec As Variant)
      Dim i As Integer
      For i = 1 To 10
            vec(i) = 0
      Next
End Sub
```

L'appel de la procédure avec un vecteur comme argument se fait comme pour toute variable.

Exemple

```
Dim vecteur(10) As Integer
Call Init(vecteur)
```

1.5.5 Portée des variables, procédures et fonctions

Une application VB peut être composée d'un ou de plusieurs formulaires et d'un ou de plusieurs modules. Dans chaque module ou formulaire, des variables, des procédures et/ou des fonctions peuvent être déclarées. Dans chaque procédure et fonction, des variables locales peuvent être déclarées.

Une fonction ou une procédure peut être déclarée soit *Privée* (*Private*), soit *Publique* (*Public*). Le sens de *Privé ou Public* se comprend par rapport au formulaire ou au module dans lesquelles elles sont déclarées.

Se pose alors le problème de la portée des variables, des fonctions et des procédures.

Si une variable est déclarée au début de la procédure (fonction) qui la manipule (**Dim** ou **Private**), elle n'est alors valide que pour cette procédure (fonction). L'existence et la valeur de la variable disparaissent avec l'instruction **End Sub** (End Function). Toute référence à cette variable en dehors de cette procédure (fonction) provoquera une erreur de compilation. Si une variable est déclarée dans la section des déclarations d'un module (formulaire), elle est valide dans toutes les procédures (fonctions) du module (formulaire).

Une variable peut aussi être déclarée **Public** ou **Global** et sera alors valide pour toute l'application.

Exemple :

```
Global MotInitial As String ‘ premier mot à traiter
```

Le tableau qui suit résume la portée des variables, des procédures et des fonctions en fonction du type de déclaration (*Dim, Private ou Public*) et de *l'endroit* où la déclaration a eu lieu.

Portée des variables, procédures et fonctions

Type	**Déclaré dans**	**Mot clé**	**Portée**
Variable	Procédure événementielle	Dim	Procédure événementielle
Variable	Procédure / fonction générale du formulaire	Dim	Procédure / fonction générale
Variable	Procédure / fonction générale de module	Dim	Procédure / fonction générale
Variable	Partie générale d'un formulaire	Dim/private	Formulaire
Variable	Partie générale d'un module	Dim/private	Module
Variable	Procédure événementielle	Private/Public	Interdit
Variable	Procédure générale d'un formulaire	Private/Public	Interdit
Variable	Procédure générale de module	Private/Public	Interdit
Variable	Partie générale d'un formulaire	Public	Formulaire
Variable	Partie générale d'un module	Public	Projet
Procédure / fonction	Partie générale d'un formulaire	Private	Formulaire
Procédure / fonction	Partie générale d'un module	Private	Module
Procédure / fonction	Partie générale d'un formulaire	Public	Formulaire
Procédure / fonction	Partie générale d'un module	Public	Projet

1.5.6 Quelques fonctions globales

Les deux tableaux suivant résument quelques fonctions mathématiques et quelques fonctions pour la manipulation des chaînes de caractères.

Fonctions mathématiques

Fonction	Utilité	Exemple
Abs(Nb)	Donne la valeur absolue du nombre	Abs (- 89) = 89
Atn(Angle)	Donne l'arc tangente de l'angle	Atn(0) = 0
Cos(Angle)	Donne le cosinus de l'angle	Cos(0) = 1
Exp(Nb)	Donne l'exponentielle du nombre	Exp(1) = 2.71828
Fix(Nb)	Tronque les décimales du nombre	Fix(-4.6) = -4 Fix(4.6) = 4
Int(Nb)	Donne la partie entière du nombre Int et Fix ne diffèrent que pour les valeurs supérieures à 0	Int(-4.6) = -5 Int(4.6) = 4
Log(Nb)	Donne le logarithme naturel (base e)	Log(1) = 0
Sgn(Nb)	Donne le signe du nombre : 1, 0 ou –1	Sgn(- 89) -1
Sin(Angle)	Donne le sinus du nombre	Sin(0) = 0
Sqr(Nb)	Donne la racine carrée du nombre	Sqr(4) = 2
Tan(Angle)	Donne la tangente de l'angle	Tan(0) = 0
Round(Nb)	Arrondi à la valeur supérieure si (Nb - Int(Nb)) > 5 inférieure si (Nb - Int(Nb)) <= 5	Round(4.5) = 4 Round(4.51) = 5

Fonctions de chaîne de caractères

Fonction	Utilité	Exemple
Asc(Car)	Donne le code ASCII d'un caractère	Asc("A") = 65
Chr(N)	Donne le caractère correspondant au code ASCII	Chr(65) = "A"
Len(Chaîne)	Donne la longueur d'une chaîne	Len ("Orange") =6
Lcase(Chaîne)	Transforme la chaîne en minuscules	Lcase("ABC")= "abc"
UCase(Chaîne)	Transforme la chaîne en majuscules	Ucase("abc")= "ABC"
LTrim(Chaîne)	Supprime les espaces de tête	LTrim(" Hello")="Hello"
RTrim(Chaîne)	Supprime les espaces de fin	LTrim("Hello ")="Hello"
Trim(Chaîne)	Supprime les espaces de tête et de fin	Trim(" Hello ")="Hello"
Left(Chaîne, N)	Renvoie les N caractères de gauche	Left("Auto",2)= "Au "
Right(Chaîne, N)	Renvoie les N caractères de droite	Right("Auto",2)= "to "
Mid(Chaîne, Pos, N)	Renvoie N caractères à partir de la position Pos	Mid("Locom", 3, 2) = "co"
InStr(Chaîne, Car)	Renvoie la position de la première occurrence du caractère dans la chaîne ou la valeur 0 si la chaîne ne contient pas le caractère	InStr ("Locom","o") = 2 InStr ("Locom","a") = 0
Str(N)	Convertit N en chaîne de caractères	Str(123) = "123"
String(N, Car)	Génère N fois le caractère spécifié	String(5, "A") = "AAAAA"
Space(N)	Génère des espaces	Space(4) = " "
Val(Chaîne)	Convertit en nombre les chiffres d'une chaîne (la conversion s'arrête au premier caractère qui n'est	Val("123") = 123 Val("123abcd") = 123

1.5.7 Interruption de séquences

Pour interrompre l'exécution d'une séquence d'instructions (dans une fonction, procédure ou boucle For), on utilise l'instruction *Exit*. Le tableau suivant résume son utilisation.

Les possibilités d'interruption de séquences

Instruction	**Porté**	**Description**
Exit function	Limitée à la fonction	Interruption de la fonction, sans exécution des instructions restantes
Exit Sub	Limitée à la procédure	Interruption de la procédure, sans exécution des instructions restantes
Exit For	Limitée à la boucle For	Interruption de la boucle, sans exécution des instructions restantes
Exit Do	Limitée à la boucle Do	Interruption de la boucle, sans exécution des instructions restantes

Exemples de petites routines

E 1.1 Exemples : Petites routines

On prendra des exemples de routines très simples ne contenant que du code :

- Avec les strings
- Avec les nombres

Avec les strings

Vous avez une chaîne de caractères, comment afficher, le premier caractère puis les 2 premiers, puis 3… ?

Dans un formulaire (une fenêtre), il y a un TextBox1(zone de texte) (avec sa propriété Multiline=True)

```
Dim C As String = "DUBONET"
Dim Tx As String
Dim i As Integer
For i = 1 To Len(C)
        Tx += Microsoft.VisualBasic.Left(C, i) + ControlChars.CrLf
Next i
TextBox1.Text = Tx
```

Mettre ce code dans Form_Load puis lancer le programme.
Affiche:

```
D
DU
DUB
DUBO
DUBON
DUBONE
DUBONET
```

On remarque que Tx est une string permettant de stocker temporairement la string à afficher, a chaque boucle on ajoute la nouvelle string (Tx += est équivalent à Tx=Tx+..) et un caractère de retour à la ligne.

Left fait partie de l'espace de nom Microsoft.VisualBasic.

Avec les nombres

Somme de N entiers.
Calculer par exemple pour Nombre=20 la Somme=1+2+3+4...+18+19+20

```
Dim Somme As Integer      'Variable somme
Dim Nombre As Integer=20
Dim i As Integer      'Variable de boucle

For i=0 To Nombre
        Somme += Nombre
Next i
```

On rappelle que Somme += Nombre est équivalent à Somme =Somme+ Nombre

Afficher les tables de multiplication.
On fait 2 boucles :

Celle avec i (qui décide de la table: table des 1, des 2…) On affiche 'table des' puis valeur de i

Celle avec j (allant de 1 à 10 pour chaque table)
Pour chaque ligne, on affiche la valeur de i puis ' X ' puis la valeur de j puis ' = ' puis la valeur de i fois j.

ControlChars.Crlf permet un saut à la ligne

A chaque fois que l'on a quelque chose à afficher, on l'ajoute à la variable String T
A la fin on affecte T à la propriété text d'un TextBox pour rendre visible les tables.

```
Dim i As Integer
Dim j As Integer
Dim T As String

For i = 1 To 10
        T += ControlChars.CrLf
        T += "Table des " & i & ControlChars.CrLf
        For j = 1 To 10
                T += i.ToString & " X " & j.ToString & "=" & i * j & ControlChars.CrLf
        Next j
Next i
TextBox1.Text = T
```

Affiche :
Table des 1
1 X 1 =1
1 X 2 =2
...

E 1.2 Exemples : Petits programmes de maths

On prendra des exemples de routines mathématiques simples :

- Calcul de l'hypoténuse d'un triangle rectangle
- Calcul de factorielle (avec ou sans récursivité)
- Un nombre est-il premier?

Calcul de l'hypoténuse d'un triangle rectangle

On crée pour cela une fonction, on envoie 2 paramètres de type Single, les 2 cotés du triangle, la fonction retourne l'hypoténuse.

```
Function Hypotenuse (ByVal Side1 As Single, ByVal Side2 As Single) As Single
        Return Sqrt((Side1 ^ 2) + (Side2 ^ 2))
End Function
```

On rappelle que le carré de l'Hypoténuse est égal à la somme des carrées des 2 autres cotés.

Factorielle

On rappelle que N! (factorielle N)= 1*2*3*...*(N-2)*(N-1)*N Exemple :
Factorielle 3 =1*2*3

```
Dim R As Long
R=Factorielle(3)        'retournera 6
```

Cette fonction n'est pas fournie par VB, créons une fonction Factorielle :

```
Function Factorielle (ByVal N as Long) As Long
        Dim i As Long
        Resultat=1
        For i= 1 to N
                Resultat=i* Resultat
        Next i
        Return Resultat
end Function
```

Cela crée une fonction recevant le paramètre N et retournant un long.

Une boucle effectue bien 1*2*3...*N-1*N.

Factorielle avec Récursivité : :

Une autre manière de calculer une factorielle est d'utiliser la récursivité : :
Une procédure est récursive si elle peut s'appeler elle même.

VB gère la récursivité.

Comment faire pour les factorielles ?
On sait que Factorielle N= N * Factorielle(N-1)
N!= N*(N-1)! en sachant que 1!=1

Créons la fonction :

```
Function Factorielle (ByVal N as Long) As Long
        If N=1 then
                Return 1
         Else
                Return N* Factorielle(N-1)

         End IF
End Function
```

Dans la fonction Factorielle on appelle la fonction Factorielle, c'est bien récursif.

Pour N=4, la fonction Factionne elle est appelée 4 fois : Factorielle (4) pui is Factorielle(3) puis Factorielle(2) puis Factorielle **(1)**

Factorielle (1) retourne 1
Factorielle (2)retourne 2 '2*factorielle(1)
Factorielle (3)retourne 6 '3*factorielle(2)
Factorielle (4) retourne 24 '4*ffactorielle(3)

Vb gère cela avec une pile des s appels. il met dans une pile les uns aux ddessus des autres les appels, quand il remonte, il dépile de haut en bas (Dernier rentré, premier sortie)


Attention : La pile a une e taille maximum, si N est trop grand, , on déclenche une erreur de type StackOverflow w.

Un nombre est-il premier ?

Un nombre premier est seulement divisible par 1 et lui-même.

Pour voir si N est entier on regardera successivement si ce nombre est ddivisible par 2 puis 3 puis 4... Jusqu'a N-1

Un nombre est divisible par un autre si la division donne un entier.

Comment voir si un nombre esest entier ? Pour ma part, j'utilise la méth hode suivante, A est entier si A = Int(A).

```
Dim IsPremier As Boolean
Dim N As Double=59
Dim I As Double
I=2: IsPremier=True
Do
        If N/I= Int(N/I) then
                IsPremier=False
        else
                i += 1
        end if
Loop While IsPremier=True And I<N
```

Pour 59 IsPremier sera égal à True.

On peut améliorer la routine en remarquant :
Si un nombre n'est pas premier il admet 2 diviseurs dont un est inférieur à racine N.

On peut donc :
- Vérifier que le nombre n'est pas pair puis
- Vérifier s'il est divisible par les nombres allant de 3...jusqu'à racine de N en ne tenant compte que des nombres impairs.

E 1.3 Exemples : Programme de Tri et de Recherche

On a parfois besoin de trier par ordre alphabétique un tableau de string.
Il existe maintenant des méthodes de tri 'automatique' entièrement gérées par VB grâce à la méthode 'sort'.

Il existe aussi des routines de tri entièrement écrite en VB, elles deviennent inutile mais c'est didactique de voir comment elle fonctionnent.

Parfois il faut chercher dans un tableau un élément; la aussi on peut écrire les routines ou utiliser les méthodes VB

Tri avec la méthode SORT

Pour un tableau unidimensionnel.

```
Dim Animals(2) As String
Animals(0) = "lion"
Animals(1) = "girafe"
Animals(2) = "loup"
Array.Sort(Animals)
```

Et le tableau est trié!!!

On rappelle que l'on ne peut pas trier un tableau multidimensionnel, mais il y a des ruses. (Voir rubrique : tableau)

Les Collections peuvent être triées automatiquement aussi.

Enfin si la propriété Sorted d'une ListBox est à True, la liste est triée automatiquement quand on la charge.

Routine de Tri

Pour trier un tableau de chaînes de caractères, il faut comparer 2 chaînes contiguës, si la première est supérieure (c'est à dire après l'autre sur le plan alphabétique) on inverse les 2 chaînes, sinon on n'inverse pas. Puis on recommence sur 2 autres chaînes en balayant le tableau jusqu'à ce qu'il soit trié.

Tout l'art des routines de tri est de faire le moins de comparaisons possible pour trier le plus vite possible.

Voyons une des routines les plus rapides, le Bubble Sort (ou tri à bulle); on le nomme ainsi car l'élément plus grand remonte progressivement au fur et à mesure jusqu'à la fin du tableau comme une bulle.

Une boucle externe allant de 1 à la fin du tableau balaye le tableau N fois, une seconde boucle interne balaye aussi le tableau et compare 2 élément contigus et les inverse si nécessaire. La boucle interne fait remonter 1 élément vers la fin du tableau, la boucle externe le fait N fois pour remonter tous les éléments.

Swap n'existe plus)
Cette routine tri bien le tableau mais n'est pas optimisée : il n'est pas nécessaire que la

boucle interne tourne de 0 à N-1 à chaque fois car après une boucle ,le dernier élément est à sa place.

Pour i=0 la boucle interne tourne jusqu'à N-1, pour i=1 jusqu'à N-2...

Cela donne :

```
Dim i, j , N As Integer        'Variable de boucle i, j ; N= nombre d'éléments-1
Dim Temp As String
N=4 'tableau de 5 éléments.
Dim T(N) As String 'élément de 0 à 4
For i=0 To N-1
        For j=0 To N-i-1
                If T(j)>T(j+1) then
                        Temp=T(j): T(j)=T(j+1):T(j+1)=Temp
                End if
        Next j
Next i
```

Il existe d'autres méthodes encore plus rapides (Méthode de Shell et Shell-Metzner).

Recherche dans une liste

On a une liste de string, on veut chercher ou (en quelle position) se trouve une string.

Pour une liste non triée, on n'a pas d'autres choix que de comparer la string cherchée à chaque élément du tableau, on utilisera donc une boucle :

```
        N=4    'tableau de 5 éléments.
        Dim T(N) As String  'élément de 0 à 4
        T(0) = "vert"
        T(1) = "bleu"
        T(2) = "rouge"
        T(3) = "jaune"
        T(4) = "blanc"
        Dim i As Integer       'Variable de boucle
        Dim AChercher As String = "rouge" 'String à chercher
        For i=0 To N
                If T(i)=AChercher then
                        Exit For
                End if
        Next i
```

'i contient 2

Pour une liste ~~triée~~ (suite ordonnée), on peut utiliser la méthode de recherche dichotomique : On compare l'élément recherché à l'élément du milieu du tableau, cela permet de savoir dans quelle moitié se situe l'élément recherché.

De nouveau on compare à l'élément recherché à l'élément du milieu de la bonne moitié...jusqu'à trouver. Pour cela on utilise les variables Inf et Sup qui sont les bornes inférieure et supérieure de la zone de recherche et la variable Milieu.
On compare l'élément recherché à l'élément du tableau d'indice milieu, si ils sont égaux on a trouvé, on sort, si ils sont différent on modifie Inf et Sup pour pointer la bonne plage puis on donne à Milieu la valeur du milieu de la nouvelle plage et on recommence.

```
        Dim N As Integer
        Dim T(N) As String            'élément de 0 à 4
        Dim Inf, Sup, Milieu As Integer '
        Dim Reponse As Integer      'contient le numero de l'élément
                                    'Ou -1 si élément non trouvé
```

```
Dim i As Integer                'Variable de boucle
Dim AChercher As String= "c"    'String à chercher

N=4   'tableau de 5 éléments.
T(0)="a"
T(1)="b"
T(2)="c"
T(3)="d"
T(4)="e"
Inf=0: Sup=N
Do
        if inf>Sup then Reponse=-1: Exit Do
        Milieu= INT((Inf+Sup)/2)
        If Achercher=T(Milieu) then Reponse=Milieu:Exit Do
        If Achercher<T(Milieu) then Sup=Milieu-1
        If Achercher>T(Milieu) then Inf=Milieu+1
Loop

'Reponse =2
```

La recherche dichotomique est rapide car il y a moins de comparaisons.

Mais comme d'habitude VB.Net possède des propriétés permettant de rechercher dans un tableau trié ou non et cela sans avoir à écrire de routine.

Binarysearch recherche un élément dans un tableau trié unidimensionnel. (Algorithme de comparaison binaire performant sur tableau trié, probablement une recherche dichotomique)

Exemple :

```
I=Array.BinarySearch(Mois, "Février")
```

IndexOf
Recherche un objet spécifié dans un tableau unidimensionnel (trié ou non), retourne l'index de la première occurrence.

```
Dim myIndex As Integer = Array.IndexOf(myArray, myString)
```

Retourne -1 si l'élément n'est pas trouvé.
LastIndexOf fait une recherche à partir de la fin.

E 1.4 Exemples : Petits calculs financiers

Coût d'augmentation de la vie

Si un objet de 100€ augmente de 3% par an, combien coûtera-t-il dans 10 ans.

```
Dim Prix As Decimal
Dim Taux As Decimal
Dim Periode As Integer=10
Dim i As Integer For i=
1 to Periode
Prix=Prix+(Prix*3/100)
Next i
```

On peut remplacer les 3 dernières lignes par:

```
Prix=Prix*(1+Taux/100)^Periode
```

Noter que l'on utilise des variables de type décimales, c'est une bonne habitude pour faire des calculs financiers (pas d'erreurs d'arrondis).

Remboursement d’un prêt

Quel est le remboursement mensuel d'un prêt d'une somme S durant une durée D (en année) à un taux annuel T ?
R=S x T / 1-(1+T)^-D (ici avec T en % mensuel et D en mois

```
Dim R, S , D, T As Decimal
S=5000        '5000€
D=15          'Sur 15 ans
T=4           '4% par an
T=T/12/100 'Taux au mois
D=D*12        'Durée en mois
R=S*T/(1-(T+1)^(-D))      'Formule connue par tout bon comptable!!
```

Si on voulait afficher le résultat dans un label (on verra cela plus loin)

```
Label1.text= R.ToString("C")
```

Ici le résultat est transformé en chaîne de caractères (grâce à ToString) au format monétaire ("C"), on obtient '36,98€' que l'on met dans le label pour l'afficher.

Ultérieurement on verra un exemple plus complet utilisant les fonctions financières de VB.

Démarche (ou méthode) de Développement Evénementiel

Dossier de spécifications détaillées

Objectif De la Démarche

Avoir une démarche d'analyse pour le développement d'un composant logiciel à réaliser dans un environnement de développement événementiel (langage de 4 éme génération).

Obtenir une description détaillée de ce composant logiciel avant le codage.

Démarche :

Réaliser le dialogue écran,

Construire les menus si nécessaire,

Construire les écrans (fenêtres) avec les objets du langage,

Définir l'accès aux données,

Etablir le tableau des objets/propriétés,

Etablir le tableau des procédures événements,

Spécifier les procédures événements (pseudo-code),

Les différentes étapes de la démarche sont formalisées dans un dossier de spécifications détaillées, à chaque étape un document est produit.

Un canevas de dossier existe en format Word, il est à copier à partir du répertoire repstag/evedos

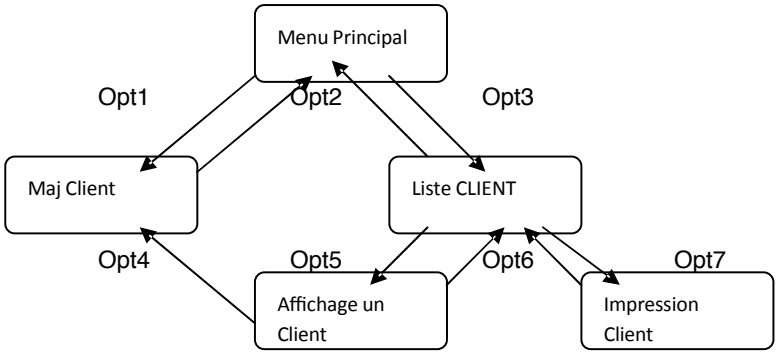
Suivre avec le canevas du dossier de spécifications détaillées les étapes de la démarche développées dans ce support.

Réaliser le Dialogue Ecran

Avant d'implémenter les différentes fenêtres de l'application, il est nécessaire d'avoir une vue d'ensemble de l'enchaînement de celle-ci.

On utilise pour formaliser l'enchaînement un schéma sous forme de graphe que l'on appelle : Graphe du Dialogue Ecran.

Exemple de Graphe de Dialogue Ecran



On passe d'une fenêtre à une autre par une commande ou un contrôle.

Exemple de commande ou contrôle

Les touches F1 à F12 ou F24
Un bouton de commande
Une Option de menu

Lien avec la méthode FAULLE :

Avec la méthode FAULLE on réalise une analyse en utilisant comme point de départ le graphe du dialogue écran. De ce dialogue on en déduit les différentes actions à partir de conditions basées sur un écran plus une option.

En analyse événementielle la nature des conditions est plus large puisque sur chaque objet on déclenche différents événements. La liste de ces conditions/actions est utilisée dans la suite de la démarche avec le tableau des procédures événements.

Il n'est pas nécessaire de réaliser l'organigramme du moniteur de dialogue, les langages de 4éme génération ont leur propre moniteur.

Construire Les Menus

Certaines applications ont comme principe d'enchaînement l'utilisation d'option dans des menus.

Cette étape consiste à hiérarchiser ces options et à indiquer sous chaque option quelle partie de l'application est déclenchée.

Le graphe du dialogue écran fait apparaître les différents chemins ou parcours possibles pour une application.

Dans la suite de la démarche, on analyse chaque partie de l'application (une option de menu) indépendamment l'une de l'autre.

Construire Les Ecrans (fenêtre)

A ce niveau de la démarche informatique les écrans ont été conçus dans le dossier d'analyse. Si le niveau de détail n'est pas suffisant, compléter ces maquettes avec tous les objets et contrôles à utiliser pour le bon fonctionnement de l'application. Les objets ou contrôles doivent être présents dans le langage événementiel utilisé en développement.

Dessiner la version définitive de l'écran, en utilisant un support papier ou mieux encore en utilisant l'outil de développement.

Voir le guide d'ergonomie Windows par exemple pour la conception générale des écrans ; couleurs, taille des boutons, type d'option, format des messages….

On cherchera toujours à utiliser une ergonomie connue des utilisateurs.

Définir le style des boites dialogue :

Informations,
Avertissements,
Erreurs.

Définir L'accès Aux Données

Les langages de programmation événementielle proposent différentes méthodes d'accès aux données. Ces méthodes ont leurs propres caractéristiques et s'utilisent en fonction du type d'application à développer.

Lorsque l'application est un simple affichage de données, les langages fournissent généralement des assistants ou méthodes d'accès simplifiées ou pré-programmées.

Dans le cas d'application complexe on utilise plutôt les méthodes qui permettent une programmation sur mesure afin d'optimiser les accès.

Dans tous les cas de figure, une réflexion est à mener à chaque fois pour déterminer quelle est la meilleure méthode d'accès aux données dans une application voire dans une feuille ou une autre.

La documentation des langages donne une synthèse des accès aux données en fonction des besoins et des types d'application.

Dans le dossier de spécifications détaillées on précise par feuille ou fenêtre les données utilisées (base de données, table(s)) avec la méthode d'accès utilisée.

Dans le cas d'utilisation d'objets spécialisés pour l'accès aux données, on les décrit dans le tableau des objets/propriétés.

Etablir Le Tableau Des Objets/Propriétés

Les langages de la programmation visuelle permettent d'implémenter des valeurs sur les propriétés des objets utilisés dans les fenêtres. Ces valeurs personnalisent l'objet en fonction des besoins ergonomiques.

Etablir le tableau des objets-propriétés consiste à reprendre chaque objet sur chaque fenêtre est de définir :

Un nom à l'objet : suivre la codification définie au niveau application.

Donner le type de l'objet :

Exemples :

Zone-texte, Bouton de Commande, Bouton-option, Boite-à-cocher, liste-déroulante, liste-simple, ascenseur,…

Donner les valeurs de propriété initiale :

Couleur, visible/non visible, taille, …

Etablir Le Tableau Des Procédures/Evénement

Un des problème de la programmation événementielle est le suivant :

Ou placer le code programme qui effectuera les traitements ?

Les langages de programmation événementielle ont comme caractéristique de proposer pour chaque objet graphique utilisé dans une fenêtre des procédures associées aux événements possibles sur cet objet.

L'objectif de ce tableau est de déterminer quels sont les événements qui sont déclenchés sur les objets.

Dans le tableau on précise le nom de l'objet et le ou les événement(s) associé(s).

Exemples :

Sur un objet Bouton de commande : utilisation de la procédure CLICK.
Sur un objet Zone-texte : utilisation de la procédure CHANGE

C'est dans ces procédures que les traitements sont définis.

Exemple :

Pour un objet fenêtre, utilisation de la procédure LOAD pour initialiser une valeur à des variables.

Spécifier Les Procédures

Généralités

Chaque procédure événement est un traitement associé à un objet. Dans cette étape il s'agit d'écrire le traitement sous forme de pseudo-code. Chaque procédure-événement sera un composant logiciel à part entière, avec les règles de la programmation structurée.

Dans une procédure-événement on utilise les structures classiques de la programmation structurée : répétitive (tant que , jusqu'à), alternative (si alors sinon), déclaration utilisation des variables

Comme pour un développement traditionnel, certains traitements peuvent être communs à plusieurs procédure-événement, dans ce cas on peut mettre ces traitements en procédure ou fonction dans une partie du code de l'application accessible par l'ensemble des procédures événements (exemple en VB : utilisation de l'objet module d'une application, en PB utilisation des fonctions utilisateurs)

Spécificités programmation événementielle :

Les objets :

Une caractéristique de la programmation événementielle basée sur l'utilisation d'objet et de faire référence à ces objets dans la programmation soit :

faire référence à une propriété d'un objet, en pseudo-code cela donne :

Objet.propriété = …..

Pour modifier une propriété par défaut ou définie avant traitement
Pour vérifier une valeur de propriété

faire référence à une méthode (ou procédure) de cet objet, en pseudo-code

…. Objet.méthode ….

Une méthode peut être de deux types soit :

1)-Une fonction avec la possibilité de tester le retour (code)

Syntaxe :
Variable = Objet.méthode(Param)

Exemple :
Booléen : Etat
Etat = Table.Ouvrir("LECTURE")

2)-Une procédure

Syntaxe :
Objet.méthode(param1,param2,……,paramX)

Exemple :
Fenêtre2.Afficher(P1,P2)

Les Variables

Portée des variables, principes :

Il y a deux types de variables : les Globales et les locales

Une **variable globale** est définie au niveau de l'application et est connue ou visible pour l'ensemble des objets de l'application. Dans certains langages on définit ces variables au niveau d'un module commun (VB) ou pour d'autre dans le contexte de développement (PB).

Une **variable locale** est définie au niveau d'une procédure est n'est connue ou visible uniquement dans cette procédure.

Portée des objets :

On accède à un objet, propriétés et méthodes, de trois manières différentes :

Directement à l'objet lui-même
(En programmation objet on dit Self ou This pour nommer l'objet)
Syntaxe :

Self.propriété ou propreté = …..
Self.méthode() ou méthode()

Contexte de l'objet

Accès aux objets d'un "conteneur"

Syntaxe :

Objet.Propriété = …..
Objet.Méthode()

Contexte du Conteneur

Accès aux objets externes d'un "conteneur" ou objet parent

Syntaxe :

ObjetParent.Objet.Propriété = …..
ObjetParent.Objet.Méthode()
Ou
Parent.Objet.Propriété = …..
Parent.Objet.Méthode()

Parent est mot réservé pour indiquer un objet parent

Contexte hors objet ou objet parent.

Les différents objets:

Faire la liste exhaustive de tous les objets possibles en programmation événementielle est impossible, chaque langage en a une multitude. On peut citer quelques objets principaux :

Objets globaux

Application ou projet
Imprimante
Bibliothèque
Module

Objet d'interface ou de présentation

Contrôles
- Champs
  - Libelle ou texte
  - Liste
  - ComboBox
  - Table
- Case
  - Option
  - A cocher
- Bouton
  - Texte
  - Graphique
- Image
  - ZoneCliquage
  - Ascenseur

Fenêtre
- SDI
- MDI

Boite de dialogue standard

Menu
OptionMenu

Objet d'accès aux données

Table
Curseur
Requête
Transaction
Base de données

Les méthodes :

Elles sont nombreuses et une typologie est difficile.

En pseudo-code on prend le mot français le plus proche de l'action à effectuer par la méthodes

Exemple :

Pour une fenêtre

Ouvrir(), Fermer(), Cacher(), Montrer()

Il existe des noms identiques de méthodes pour des objets différents.

Exemple :

Objet Liste ou objet ComboBox

AjouterLigne(), SupprimerLigne

Objet Table, BasedeDonnées, Transaction

Ouvrir(), Fermer()

C'est le nom de l'objet.méthode qui permet de les différencier.

2.2 Programmation par événements

A la différence de la programmation séquentielle, où les instructions s'exécutent de manière séquentielle, VB est un langage qui permet de réaliser de la programmation par événements, c'est-à-dire programmer des procédures qui s'exécutent quand un événement est déclenché. La plupart du temps, l'événement est déclenché par l'utilisateur du programme.

Quand on travail dans un environnement multifenêtrés (Windows) chaque fois, qu'*on clique sur la souris*, qu'on *ouvre ou ferme une fenêtre*, qu'on *appuie sur une touche du clavier, on déclenche un événement* auquel le programme utilisé réagit. La programmation par événements consiste à programmer ce que le programme doit faire quand un événement particulier survient.

A chaque objet VB (*contrôle, formulaire, etc.*) peut être associé *une ou plusieurs procédures événementielles* écrites avec le langage de programmation VB.

Procédures événementielles *(Private Sub NomObjet_NomEvénement… End Sub)*

Une procédure *événementielle* n'est rien d'autre qu'une procédure classique mais qui s'exécute quand un *EVENEMENT particulier* se produit [3].

La déclaration de l'événement *NomObjet_NomEvénement()* se fait comme suit (voir syntaxe), où *NomObjet* est le nom de l'objet auquel est rattaché l'événement *NomEvénement*. Comme dans une procédure classique, *aucun, un ou plusieurs paramètres et leurs types respectifs* peuvent être déclarés entre parenthèses.

Pour attacher une procédure événementielle à un objet, il suffit de « double cliquer » sur celui-ci. VB inscrit alors la déclaration de la procédure avec des paramètres par défaut (ne pas mofifier ces paramètres).

Syntaxe

```
Private Sub NomObjet_NomEvénement( Argument As Type, … )
      Instruction1
      Instruction2
      …
End Sub
```

Un ensemble d'événements peut être rattaché à chaque type d'objet. Ci-dessous quelques exemples d'événements :

Evénement	Se produit quand
Click	*On clique sur le bouton gauche de la souris*
DblClick	*On double clique sur le bouton gauche de la souris*
Load	*L'objet* **NomObjet** *est chargé*
Change	*La valeur stockée par l'objet* **Nomobjet** *change*
MouseDown	*On clique sur la souris sans relâcher le bouton*
MouseUp	*On a relâché le bouton de la souris*
MouseMove	*On a bougé la souris*
KeyDown	*On a appuyé sur une touche du clavier sans la relâcher*
MouseUp	*On a relâché une touche du clavier*
MousePress	*On a appuyé sur une touche du clavier et on l'a relâché*

A chaque formulaire sera associé un fichier logique portant le nom '*Nom_Formulaire'*, voir figure 10. Celui-ci contiendra le **code VB** des différentes procédures relatives aux événements associés au formulaire en question ainsi qu'aux différents objets qui lui sont rattachés.

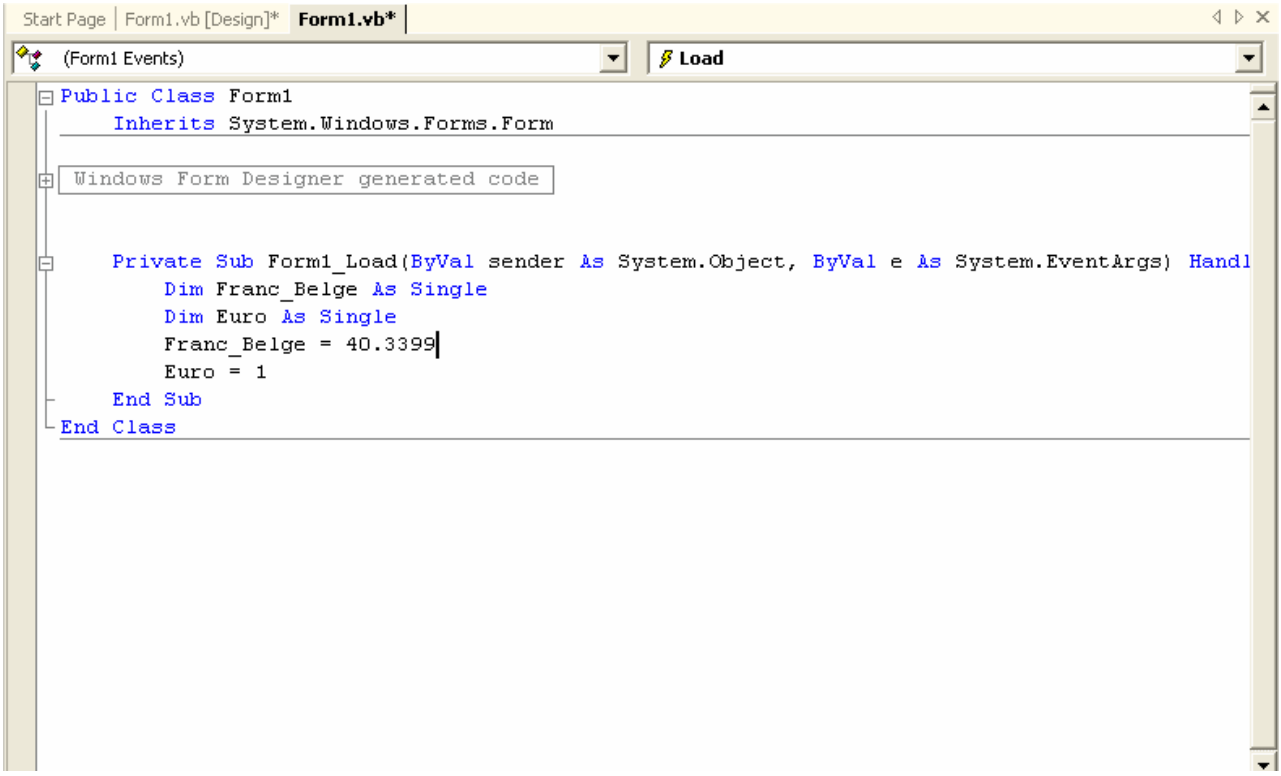


Figure 10 : Fenêtre du code VB relative au formulaire *Convertisseur*

Comment attacher une procédure événementielle « Load » à un formulaire ?
*Pour attacher une procédure événementielle à un formulaire, double cliquez sur celui-ci (et non pas sur un des contrôles qui le composent). VB ouvre alors une fenêtre textuelle et place le curseur dans le cadre d'***une procédure événementielle particulière : Form_Load().**

Form_Load()

La procédure de nom *Form_Load()* s'exécute lors du chargement du formulaire correspondant, c'est-à-dire **avant que** le formulaire n'apparaisse à l'écran.

Pratique. Placez les deux lignes de codes comme indiqué à la figure 10 (entre les deux lignes *Private Sub Form_Load()* et *End Sub)*. Ainsi, avant que le formulaire n'apparaisse à l'utilisateur, *Franc_Belge.Text* et *Euro.Text* seront initialisés à *40.3399* et à *"Euro : 1"* (voir figure 9).

Exécutez votre programme pour noter l'effet de l'initialisation. Y-a-t il moyen d'initialiser sans écrire du code ?

Pratique.

a. On désire que, lorsque l'utilisateur clique sur le bouton *Convertir* (figure 9), une procédure s'exécute et convertisse le montant dans la zone Franc_Belge et donne le montant équivalent en Euro (dans la zone label).

b. On désire que, lorsque l'utilisateur clique sur le bouton *Quitter* (figure 9), une procédure s'exécute et ferme la fenêtre. L'instruction *End* ferme une fenêtre.

Exécutez votre programme, introduisez un montant en Franc Belge et appuyez sur *Convertir*.

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
        Franc_Belge.Text = 40.3399
        Euro.Text = "Euro = 1"

    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
        Dim valEuro As Double
        valEuro = Franc_Belge.Text / 40.3399
        Euro.Text = "Euro = " & valEuro

    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e
        End
    End Sub
End Class
```

Figure 11 : Fenêtre du code VB relative au formulaire *Convertisseur* (suite)

Comment Sauver votre travail ?
Sélectionnez dans la barre du menu : File, Save Project as. VB vous demandera de donner un nom à votre projet et, à chaque formulaire et module, le composant.

L'intérêt de donner un nom à chaque *formulaire et module* réside dans le fait qu'un formulaire ou un module peut être **réutilisé** dans des projets différents.

Comment ajouter un nouveau formulaire dans un projet ?
Cliquez avec le bouton droit de la souris sur le mon du Projet se trouvant dans la fenêtre Projet, sélectionnez dans le menu proposé : Add, Windows form.

Comment ajouter un formulaire existant dans un projet ?
Cliquez avec le bouton droit de la souris sur le mon du Projet se trouvant dans la fenêtre Projet, sélectionnez dans le menu proposé : Add, Add Existing Items. Sélectionnez le nom du fichier correspondant au formulaire recherché, puis appuyez sur Open.

Comment retrouvez les différentes fenêtres (ToolBox, Project, Properties) ?

Si ces fenêtres sont fermées vous pouvez toujours les ouvrir en sélectionnant dans le barre du menu, View (Toolbox, Project Explorer, Properties Windows).

En résumé

- Les objets manipulés sont appelés des **contrôles** (bouton de commande, boîte de dialogue, zone de texte, zone d'image, etc.)
- L'interface utilisateur créée est multifenêtrée. Une fenêtre est appelée un **formulaire** (Form). Un **formulaire** est lui-même un contrôle.
- Chaque contrôle peut réagir à des **événements** qui lancent des procédures (dédiées) codées en VB.
    - Des **modules généraux** de code peuvent porter sur tout le programme. Ces modules sont réutilisables.

3.1 Concept d'objet

Le concept d'objet

Comme vous l'avez et vous allez encore le constater, le terme **OBJET** est souvent cité dans ce texte, et ceci est loin d'être le fruit du hasard. En effet, VB .NET est un **langage orienté objet**, c'est-à-dire que toute **CHOSE** que vous aurez à manipuler et à utiliser n'est rien d'autre qu'un *OBJET indépendant*. Un objet est défini par un **nom** et un certain nombre de **propriétés**. Il est aussi défini par un ensemble de méthodes (procédures ou fonctions).

Notez bien que les propriétés et les méthodes qui définissent l'objet ne peuvent être invoquées qu'en spécifiant le nom de celui-ci.

Le concept de propriété d'un objet

Une propriété d'un objet est un attribut ou une caractéristique de celui-ci. Chaque propriété porte un nom (attribut ou variable) et a une valeurqui lui est associée. La figure 7 montre une partie des propriétés de l'objet portant le nom '*Form1*'. Comme propriété d'un objet, on peut citer: nom, forme graphique, dimension, couleur, structure de données associée, etc.

Pour accéder à la propriété d'un objet avec du code VB, il faut **obligatoirement** préciser le nom de l'objet suivi d'un point suivi du nom de la propriété en question. On peut ainsi distinguer et utiliser les même priorités appartenant à des objets différents. Pour accéder et modifier les propriétés d'un objet, on procède comme dans l'exemple suivant :

Exemple

```
Form1.Text = " Convertisseur FB en Euro "
Form1.BackColor = &H800000
Label1. Text = " Entrez votre texte "
Button. Text = "  Franc Belge"
```

Pour les objets existants lors du développement, leurs propriétés peuvent **aussi** être modifiées à l'aide de la fenêtre de propriétés : **Properties,** voir figure 7.

Le concept de méthode d'un objet

Une méthode d'un objet est une fonction ou une procédure rattachée à l'objet en question. Pour être appelée, *comme dans le cas de la propriété*, elle doit être précédée par le nom de l'objet correspondant suivi d'un point.

Syntaxe

```
NomObjet.NomProcedure( paramètre1, paramètre2, …)
Variable = NomObjet.NomFonction (paramètre1, paramètre2, …)
```

Exemple

```
    Form1.Showdialog          ' fait apparaître la fenêtre Form1
```

La figure 13 montre l'éditeur des classes (objets) VB. La colonne de droite présente les méthodes et les propriétés de la classe sélectionnée. La fenêtre du bas donne une explication succincte de la classe, propriété ou méthode sélectionnée. Pour ouvrir l'éditeur des classes VB, il suffit de cliquer sur le bouton *Object Browser*.

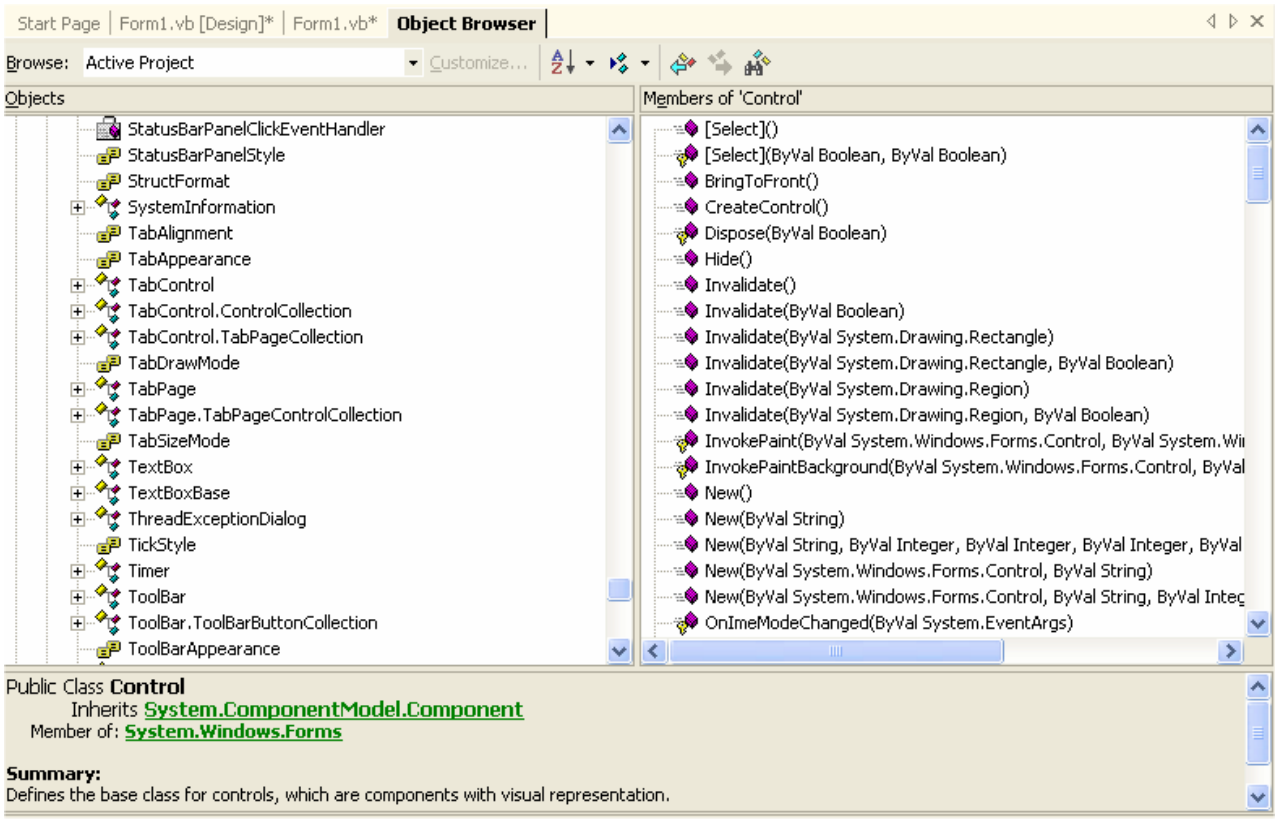


Figure 11 : Editeur des objets VB

3.2 Contrôles standards

3.2.1 La propriété "Name"

Dans tous les contrôles, la propriété *Name* permet de référencer le contrôle correspondant dans du code VB. A l'intérieur *d'un même formulaire*, la propriété *Name* doit être unique. Comme indiqué ci-dessus, *Name* permettra aussi d'accéder aux différentes propriétés et d'appeler les différentes méthodes de l'objet.

Lorsqu'un contrôle est placé sur un formulaire, VB lui affecte un nom, généré automatiquement, composé du nom du contrôle, suivi d'un chiffre correspondant au nombre de contrôles de même type déjà intégrés dans le formulaire.

3.2.2 Label

Le contrôle *Label* permet d'afficher un texte statique. La propriété (de type String) chargée de stocker ce texte (une chaîne de caractères) est la propriété *Text*. Celui-ci sera affiché lors de l'affichage du formulaire dans lequel il est placé. L'instruction qui suit modifie le texte correspondant au contrôle Label de nom *Label1*.

Exemple

Label1.Text = "l'équivalent en Euro = 15.689,89"

3.2.3 TextBox

Le contrôle *Textbox* permet *d'afficher* et *de saisir* un texte au clavier. La propriété (de type String) chargée de stocker ce texte (une chaîne de caractères) est la propriété *Text*. Celui-ci sera affiché lors de l'affichage du formulaire dans lequel il est placé et modifiable par l'utilisateur. L'instruction qui suit modifie le texte correspondant au contrôle *TextBox* de nom *Text1*.

Exemple

Text1.Text = *"Entez votre texte ici"*

La figure 9 montre des exemples des contrôles *TextBox* et *Label*.

3.2.4 RadioButton

Le contrôle *RadioButton* combine deux fonctionnalités. Il permet *de sélectionner* une option présentée par *un texte statique* (un contrôle *Label*). Le contrôle *RadioButton*, utilisé sur un formulaire en au moins deux instances, permet de faire *une seule sélection* parmi les différents choix proposés (parmi les différents *RadioButton* affichés). La propriété du contrôle qui stock l'état de celui-ci est la propriété *Enabled*, de type *Boolean*. La valeur *True* veut dire que l'option est choisie. Notez bien que VB se charge de mettre à jour la propriété *Enabled* une fois qu'une sélection est faite (mettre *True* à la propriété *Enabled* du bouton sélectionné et *False* aux autres).

Exemple

OptionButton1.Enabled = True
OptionButton2. Enabled = False

Pratique. Dans l'exemple de la figure 14, l'utilisateur peut soit convertir du FB vers l'euro ou inversement. Les contrôles ont été déclarés comme suit :

Contrôle	**Name**	**Caption**
TextBox	Montant	
Label	Label1	Equivalent en Euro = 1
Label	Label2	Entrez le montant à convertir
Label	Label3	Montant à convertir en
OptionButton	OptionEuro	Euro
OptionButton	OptionFB	FB
CommandButton	Quitter	Quitter

Figure 14 : Exemple d'utilisation du contrôle *RadioButton*

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Hand

        entree.Text = 40.3399
        sortie.Text = "equivalent en euro = 1"
    End Sub



    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Ha
        MsgBox("je termine et je sors")
        End
    End Sub


    Private Sub RadioButton1_CheckedChanged(ByVal sender As System.Object, ByVal e As Syster
        Dim Val_euro As Single
        Val_euro = entree.Text / 40.3399
        sortie.Text = "L'equivalent en euro est " & Val_euro


    End Sub

    Private Sub RadioButton2_CheckedChanged(ByVal sender As System.Object, ByVal e As Syster
        Dim Val_franc As Single
        Val_franc = entree.Text * 40.3399
        sortie.Text = "L'equivalent en FB est " & Val_franc
    End Sub
End Class
```

Figure 15 : Code VB de l'exemple de la figure 14 – 1 ère possibilité

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    Dim Montanten As Char
```

```
Private Sub Montant_Change()
        Dim valeuro, valeFB, value As Double
        value = Double.Parse(entree.Text)
        If entree.Text <> " " Then
            Select Case Montanten
                Case "E"
                    valeuro = value / 40.3399
                    sortie.Text = "Equivalent en Euro: " & valeuro

                Case "F"
                    valeFB = value * 40.3399
                    sortie.Text = "Equivalent en FB: " & valeFB
            End Select
        End If
End Sub
```

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
        MsgBox("Quitter?")
        End
End Sub


Private Sub euro_CheckedChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles euro.CheckedChanged
        Montanten = "E"
        Montant_Change()
End Sub


Private Sub FB_CheckedChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles FB.CheckedChanged
        Montanten = "F"
        Montant_Change()
End Sub

End Class
```

Figure 16 : Code VB de l'exemple de la figure 14 – 2 ème possibilité

Dans l'exemple ci-dessus, la conversion se fait de manière *dynamique* chaque fois que le montant change (*Montant_Change*). La procédure événementielle *Montant_Change* est aussi appelée comme simple procédure dans les deux autres procédures événementielles : *OptionEuro_Click* et *OptionFB_Click*

Nom_objet_Change()
La procédure événementielle de nom *Nom_objet_Change()* s'exécute quand l'objet portant le nom *Nom_objet* change.

3.2.5 CheckButton

Ce contrôle ressemble de très près au contrôle *RadioButton*. Il combine aussi deux fonctionnalités. Il permet *de sélectionner* une option présentée par *un texte statique* (un contrôle *Label*). La différence majeure réside dans le fait que l'utilisateur peut faire de multiples sélections parmi les différents contrôles de même type. La propriété du contrôle qui stocke l'état de celui-ci est la propriété *Checked* qui prend l'une des deux valeur suivantes : false = choix non sélectionné, true = choix sélectionné.

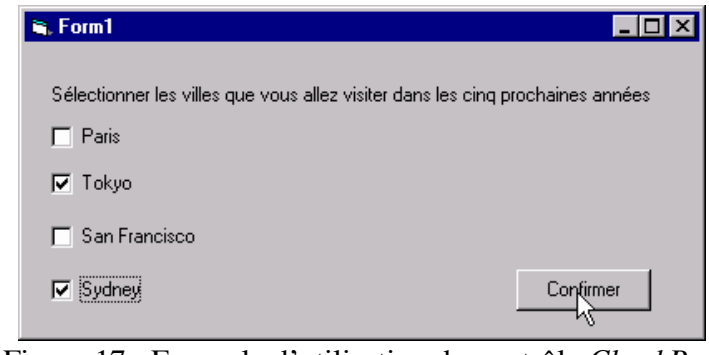


Figure 17 : Exemple d'utilisation du contrôle *CheckButton*

Dans l'exemple de la figure 17, l'utilisateur peut choisir jusqu'à 4 villes.

Notez que c'est à l'utilisateur de gérer la relation entre les différents choix en programmant la *procédure événementielle Click()*.

Exemple

```
Private Sub Check1_CheckedChanged()     ' on clique sur le 1er choix
      If Check1.checked = true Then     ' si le 1er choix vient d'être sélectionné
            Check3.enabled = false      ' le 3ème devient indisponible
      Else
            Check3.Checked = true       'sinon le 3 ème devient disponible
      End If
End Sub
```

3.2.7 GroupBox

Un *GroupBox* est une fenêtre. C'est un contrôle qui peut être placé sur un formulaire pour créer un groupe de contrôles. Tout contrôle placé sur le *GroupBox* (lui-même placé sur un formulaire) appartiendra à ce groupe.

On a vu dans le cas du contrôle *RadioButton* qu'une seule option peut être choisie. Cependant, si on veut présenter à l'utilisateur deux groupes de choix dans lesquels il peut sélectionner deux choix non exclusifs, un dans le premier groupe et un dans le second, ceci n'est pas possible. Pour résoudre ce problème, il suffit de placer l'un des groupes d'options dans un *GroupBox*, l'autre appartiendra au formulaire.

Notez que les *GroupBox* peuvent être utilisés pour créer un groupe de contrôles de différents types, *TextBox, RadioButton, Label, etc*.

La propriété *Text* permet de donner un titre au *GroupBox*.

Pratique.

Dans l'exemple de la figure 18, deux *GroupBox* sont utilisés : *Destination* et *Moyen de transport*. Chaque *GroupBox* intègre un groupe d'options. Ainsi, l'utilisateur peut choisir une seule ville et un seul moyen de transport.

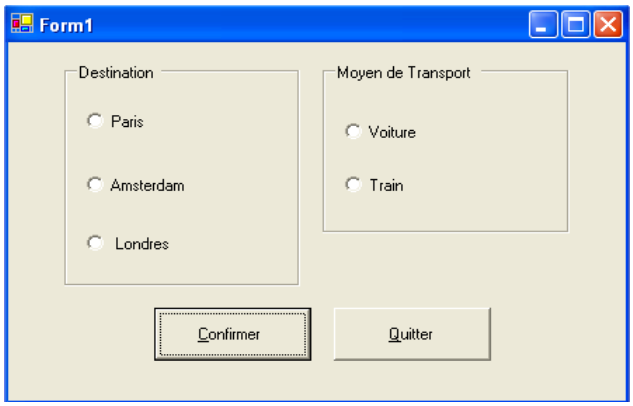


Figure 18 : Exemple d'utilisation du contrôle *GroupBox*

Le code correspondant à ce programme est présenté dans la figure 19. Les contrôles ont été déclarés comme suit :

Contrôle	Name	Text
Label	*Label1*	*Choissisez les villes que vous allez visiter*
CheckBox	*C_Paris*	*Paris*
CheckBox	*C_Tokyo*	*Tokyo*
CheckBox	*C_SanFrancisco*	*San Francisco*
CheckBox	*C_Sydney*	*Sydney*
CommandButton	*Confirmer*	*Confirmer*
CommandButton	*Quitter*	*Quitter*

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    Dim villeChoisie As String
    Dim MoyenChoisi As String

 Windows Form Designer generated code 

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Cli
        If R_Paris.Checked = True Then
            villeChoisie = "Paris "
        End If
        If R_Amsterdam.Checked = True Then
            villeChoisie = "Amsterdam "
        End If
        If R_Londres.Checked = True Then
            villeChoisie = "Londres "
        End If
        If R_Voiture.Checked = True Then
            MoyenChoisi = "Voiture "
        End If
        If R_Train.Checked = True Then
            MoyenChoisi = "Train "
        End If
        MsgBox("La destination est " & villeChoisie & "avec comme moyen de transport le " & MoyenChoisi)
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Cli
        End
    End Sub
End Class
```

Figure 19 : Code VB relatif à l'exemple de la figure 18

3.2.8 Exercices

a. Programmez l'application qui permet de réaliser l'application présentée à figure 14. Exemple d'utilisation du contrôle *RadioButton*.

b. Sachant qu'1 US$ coûte 38.5168 FB et qu'1 FF coûte 6.1498 FB, modifiez le code de l'exemple pour qu'on puisse aussi avoir la conversion en US$ et en FF. Votre application doit avoir l'allure de la figure 20.

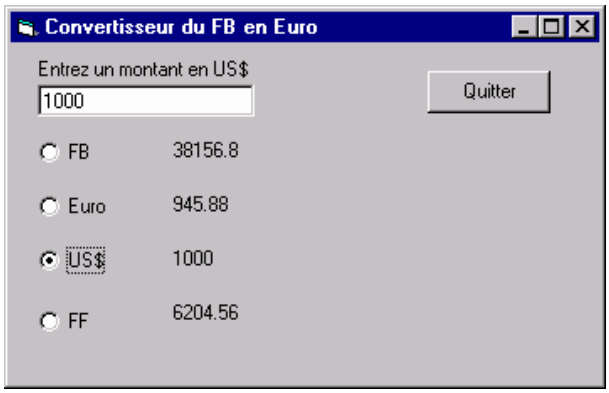


Figure 20 : Exemple de formulaire de conversion de devises

c. Programmez l'application qui permet de réaliser la figure 17 (Exemple d'utilisation du contrôle *CheckButton*). Le bouton *Confirmer* doit ouvrir (avec la procédure *Show*) un autre formulaire contenant quatre contrôles *Label* qui affichent *uniquement* les choix faits par l'utilisateur.

d. Programmez l'application qui permet de réaliser la figure 18 (Exemple d'utilisation du contrôle *GroupBox*). Utilisez deux vecteurs de contrôles pour les différents *RadioButton*. Le bouton *Confirmer* doit ouvrir (avec la procédure *Show*) un autre formulaire résumant les choix faits par l'utilisateur.

e. Vous êtes à présent capable de répondre aux questions suivantes

1. Expliquez le concept d'objet
2. Expliquez le concept de propriété d'un objet
3. Expliquez le concept de méthode d'un objet
4. Comment peut-on accéder à une propriété d'un objet ?
5. Comment fait-on appel à une méthode d'un objet ?
6. Donnez des exemples d'objets en VB.
7. Donnez des exemples de propriétés en VB.
8. Donnez des exemples de méthodes en VB.
9. Comment trouver la liste des classes d'objets VB ?
10. Quel est l'intérêt de la propriété *Name*?
11. Pourquoi à l'intérieur d'un formulaire la propriété *Name* d'un objet doit être unique ?
12. Définir le rôle du contrôle *Label* et donnez des exemples de propriétés
13. Définir le rôle du contrôle *TextBox* et donnez des exemples de propriétés
14. Définir le rôle du contrôle *RadioButton* et donnez des exemples de propriétés
15. Définir le rôle du contrôle *CheckButton* et donnez des exemples de propriétés
16. Quelle est la différence entre les contrôles *CheckButton* et *RadioButton*?
17. Définir le rôle du contrôle *GroupBox* et donnez des exemples de propriétés

3.2.9 ListBox

Un *ListBox* est un contrôle qui permet de proposer une liste de valeurs parmi lesquelles l'utilisateur ne peut en choisir qu'une seule. La dite liste est stockée dans la propriété *Items*. La figure 21 montre un exemple d'utilisation du contrôle *ListBot*.

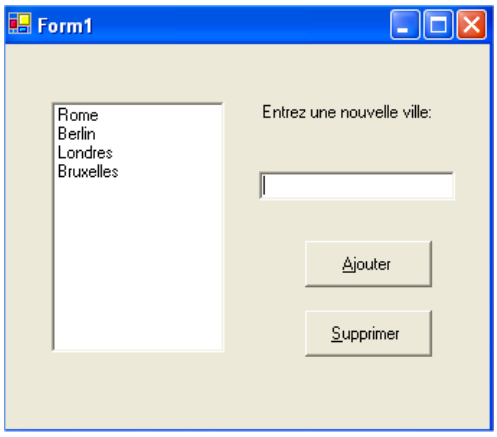


Figure 21 : Exemple utilisant le contrôle *ListBox*

3.2.10 ComboBox

Le contrôle *ComboBox* combine les fonctionnalités des contrôles *TextBox* et *ListBox*. La propriété *Text* stocke l'élément à chercher, à sélectionner ou à ajouter et la propriété *Items* stocke la liste des valeurs possibles, comme dans le cas du contrôle *ListBox*. La figure 22 montre un exemple d'utilisation du contrôle *ComboBox*.

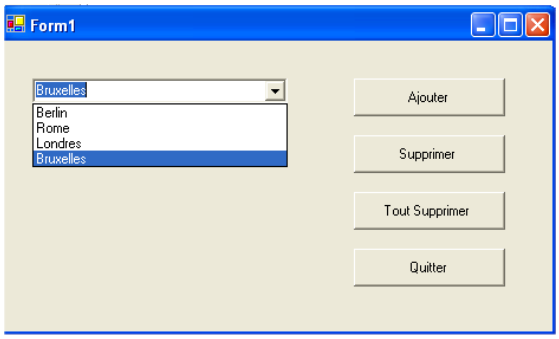


Figure 22 : Exemple utilisant le contrôle *ComboBox*

3.2.11 La propriété Items

La propriété *Items* se trouve dans plusieurs contrôles (ListBox, ComboBox, …). Elle peut être remplie lors de la conception dans la fenêtre *Properties* comme monter dans la figure 23 [4]. Cette liste peut aussi être mise à jour (ajout, suppression) de manière dynamique durant l'exécution du programme.

[4] Pour ajouter une ligne (un élément) dans la liste, il faut cliquer sur les 3 petits points

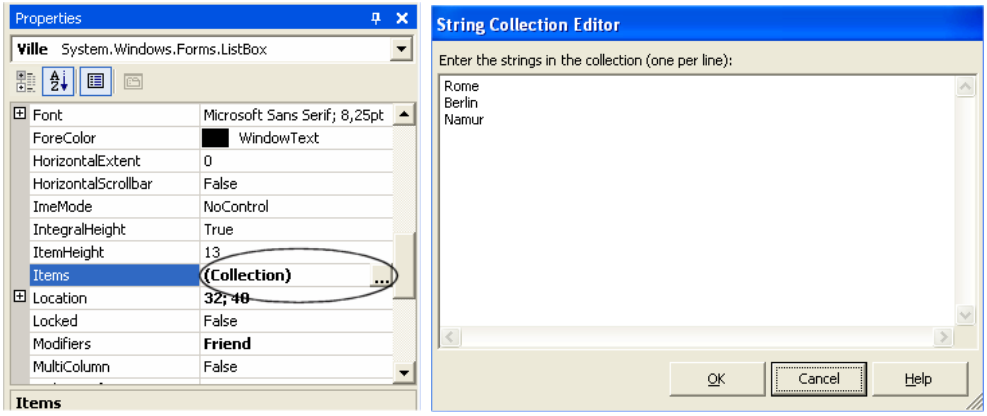


Figure 23: La propriété *Items* d'un contrôle *ComboBox* ou d'un contrôle *ListBox*.

Quelques propriétés et méthodes pour gérer les **ListBox** ou **ComboBox**

Syntaxe

`Nom_List_Box.Items.Add(valeur_i)`	: ajoute la valeur valeur_i dans la propriété *List*
`Nom_List_Box.Items.Remove(valeur_i)`	: enlève l'élément *valeur_i* de *List*
`Nom_List_Box.Items.RemoveAt(i)`	: enlève l'élément d'indice *i* de *List*
`Nom_List_Box.Items.Clear`	: enlève tous les éléments de *List*
`Nom_List_Box.Items.Count`	: donne le nombre d'éléments dans *List*
`Nom_List_Box.items(i)`	: retourne l'item d'indice *i* de la *L ist*
	ListIndex 0 = 1er élément, 1 = 2er élément, …
`Nom_List_Box.SelectedItem`	: retourne l'élément sélectionné
`Nom_List_Box.Sorted`	: = *True*, maintient la liste triée par ordre alphabétique croissant

Exemples

```
ListBox1.Items.Add ("Paris")          'Ajoute Paris à la liste de ListBox List1.
Items.Add(text1.text)                 'Ajoute Text1.Text à la liste de ListBox
Text1.Text = List1.SelectedItem       ' affecte l'élément sélectionné à Text1.Text
List1.Items.Remove(List1.SelectedItem) ' enlève l'élément sélectionné de list
```

La procédure événementielle suivante supprime la ville introduite par l'utilisateur.

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

        Dim Ville As String
        Dim i As Short
        Ville = InputBox("Enter la ville à supprimer : ")
        For i = 0 To (villes.Items.Count - 1)
            If villes.Items(i) = Ville Then
                villes.Items.RemoveAt(i)
                Exit For
            End If
        Next

End Sub
```

Pratique. L'exemple de la figure 24 illustre un *ComboList* pour stocker une liste de villes. L'utilisateur peut manipuler dynamiquement la dite liste. Le code correspondant à ce programme est présenté dans la figure 25.

La fonction *Existe(Ville)* vérifie si *Ville* se trouve dans la liste des villes. Une ville ne peut être ajoutée que si elle n'existe pas. Une ville ne peut être supprimée qu'après confirmation de l'utilisateur, c'est-à-dire, celui-ci a appuyé sur OK. La fonction MsgBox, utilisée avec l'option *OkCancel*, affiche les boutons *Ok* et *Cancel*. Elle retourne 1 si on appuie sur *OK* et 2 si on appuie sur *Cancel*.

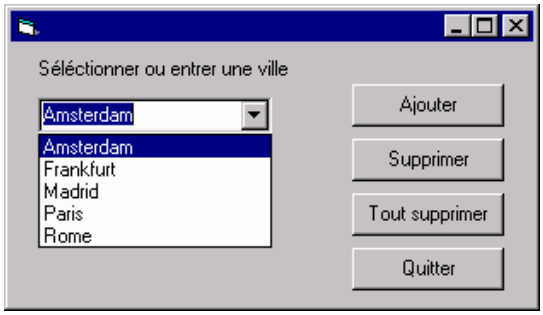


Figure 24: Exemple manipulant la propriété *List* du contrôle *ComboList*

```
Public Class Form1
    Inherits System.Windows.Forms.Form

Windows Form Designer generated code
    Private Function existe(ByVal ville As String) As Boolean
        Dim var As Boolean
        Dim i As Short
        var = False
        For i = 0 To (villes.Items.Count - 1)
            If villes.Items(i) = ville Then
                var = True
            End If
        Next
        existe = var
    End Function
    Private Sub Ajouter_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Ajouter.Click
        If Not existe(villes.Text) Then
            villes.Items.Add(villes.Text)
            MsgBox("ajout de " & villes.Text & " réussi")
        Else
            MsgBox(villes.Text & " est déjà dans la liste")
        End If
    End Sub

    Private Sub Quitter_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Quitter.Click
        End
    End Sub

    Private Sub Supprimer_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Supprimer.Click
        If MsgBox("Etes-vous bien sur de vouloir supprimer " & villes.Text & " ?", MessageBoxButtons.OKCancel) = 1 Then
            villes.Items.Remove(villes.SelectedItem)
        End If
    End Sub

    Private Sub SupprimerTout_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SupprimerTout.Click
        If MsgBox("Etes-vous bien sur de vouloir tout supprimer ?", MessageBoxButtons.OKCancel) = 1 Then
            villes.Items.Clear()
        End If
    End Sub
End Class
```

Figure 25 : Code VB relatif à l'exemple de la figure 24

3.2.12 Exercices

a. Programmez le programme qui permet de réaliser la figure 24: Exemple manipulant la propriété *List* du contrôle *ComboList*

b. Dans le même esprit que l'exercice précédent, créez un programme qui permet de convertir un montant en devise vers une autre divise. Votre application doit avoir l'allure de la figure 26. Utilisez trois contrôles *ListBox* : *DeviseB, DeviseC et TauxC*. *DeviseB* et *DeviseC* serviront, respectivement, à sélectionner la devise de base et la devise de conversion. *TauxC* servira à stocker les taux de change des différentes devises par rapport à une devise de référence (l'Euro). Notez bien que les trois contrôles ne doivent pas être triés, ainsi la devise d'indice i du contrôle *DevideC* correspond au taux i du contrôle *TauxC*.

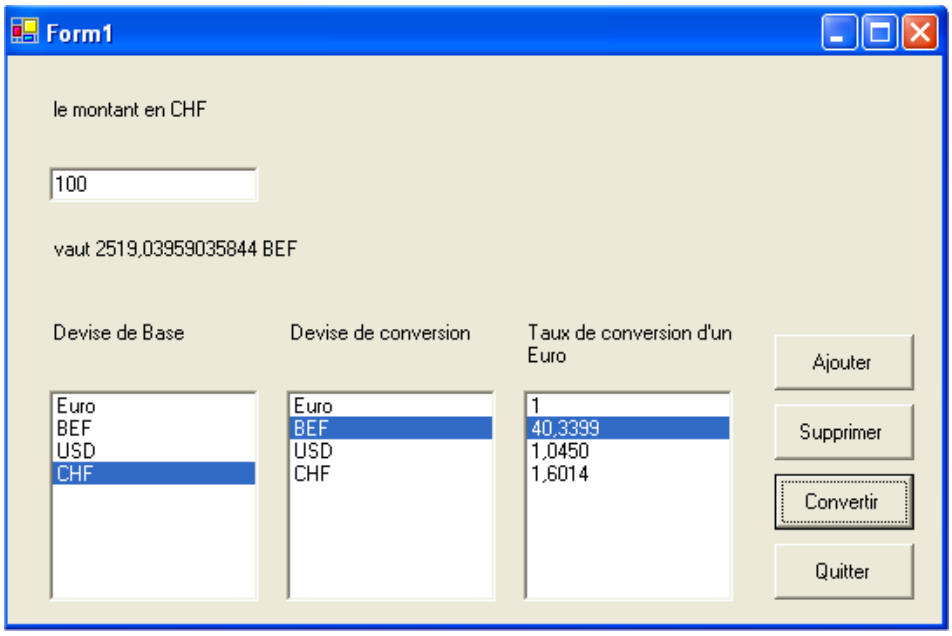


Figure 26 : Application à programmer

L'exemple ci-dessous montre comment convertir un montant (*Montant.Text*) d'une devise de base (*DeviseB*) vers une devise de conversion (*DeviseC*) en utilisant le taux de conversion d'une devise de référence (ici l'Euro). Par exemple, si on désire convertir un montant en CHF (*Devise de base*) en BEF (*Devise de conversion*), il faut d'abord trouver le montant équivalent en Euro puis multiplier ce montant par le taux du FB d'un Euro.

100 CHF = ((100 / 1.6014) * 40.3399) BEF = 2519.04 BEF.

Exemple

```
Private Sub Convertir_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Convertir.Cl:
    Dim MontantConverti, mont As Double
    mont = Double.Parse(montant.Text)
    MontantConverti = (mont / TauxC.Items(DeviseB.SelectedIndex)) * TauxC.Items(DeviseC.SelectedIndex)
    Label1.Text = "le montant en " & DeviseB.SelectedItem
    Label2.Text = "vaut " & MontantConverti & " " & DeviseC.SelectedItem
End Sub
```

Figure 27 : Code associé au bouton « convertir »

c. Modifiez votre programme pour que les différentes listes soient dynamiques, permettant à l'utilisateur d'ajouter ou de supprimer une devise.

3.2.13 Solution

```
Public Class Form1

Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
        DeviseB.SelectedItem = DeviseB.Items(0)
        DeviseC.SelectedItem = DeviseC.Items(1)
        TauxC.SelectedItem = TauxC.Items(1)
        Label1.Text = "le montant en " & DeviseB.SelectedItem
        Label2.Text = "vaut 40.3399 " & DeviseC.SelectedItem
        montant.Text = 1
End Sub

Private Sub Convertir_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Convertir.Click
        Dim MontantConverti, mont As Double
        mont = Double.Parse(montant.Text)
        MontantConverti = (mont / TauxC.Items(DeviseB.SelectedIndex)) *
            TauxC.Items(DeviseC.SelectedIndex)
        Label1.Text = "le montant en " & DeviseB.SelectedItem
        Label2.Text = "vaut " & MontantConverti & " " & DeviseC.SelectedItem
End Sub

Private Function existe(ByVal Devise As String) As Boolean
        Dim var As Boolean
        Dim i As Short
        var = False
        For i = 0 To (DeviseB.Items.Count - 1)
            If DeviseB.Items(i) = Devise Then
                var = True
                Exit For
            End If
        Next
        existe = var
End Function

Private Sub Ajouter_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Ajouter.Click
        Dim NouvelleDevise As String
        Dim NouveauTaux As Short
        NouvelleDevise = InputBox("Entrez la nouvelle devise")
        If NouvelleDevise <> "" And Not existe(NouvelleDevise) Then
            NouveauTaux = InputBox("Entrez le nouveai taux")
            If NouveauTaux > 0 Then
                TauxC.Items.Add(NouveauTaux)
                DeviseC.Items.Add(NouvelleDevise)
                DeviseB.Items.Add(NouvelleDevise)
            End If
        Else
            MsgBox("Entrez une autre devise, celle-ci est déjà répertoriée")
        End If
End Sub

Private Sub Supprimer_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Supprimer.Click
        Dim Indice As Integer
        If MsgBox("Etes-vous bien sur de vouloir supprimer " & DeviseB.SelectedItem & "
             ?", MessageBoxButtons.OKCancel) = 1 Then
             Indice = DeviseB.SelectedIndex
             TauxC.Items.RemoveAt(Indice)
             DeviseB.Items.Remove(DeviseB.SelectedItem)
             DeviseC.Items.RemoveAt(Indice)
        End If
    End Sub
Private Sub Quitter_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Quitter.Click
        End
End Sub
End Class
```

Exemple de petits programmes

E 3.1 Exemples : Conversion Francs/Euros

Comment créer un programme de conversion Francs=>Euros et Euros=> Francs ?

Euros
2
Francs:
13.12

Il y a une zone de saisie Euros, une zone Francs, si je tape dans la zone Euros '2' il s'affiche '13.12' dans la zone Francs, cela fonctionne aussi dans le sens Francs=>Euros.

Comment faire cela?
Un formulaire affichera les zones de saisie, un module standard contiendra les procédures de conversion.

On crée un formulaire contenant :

- 2 TextBox BoiteF et BoiteE, leurs propriétés Text=""
- 2 labels dont la propriété Text sera ="Euros" et "Francs", on les positionnera comme ci-dessus.

Dans le formulaire, je dimensionne un flag (ou drapeau) : flagAffiche, il sera donc visible dans la totalité du formulaire. Je l'initialise à True.

```
Public Class Form1
Inherits System.Windows.Forms.Form
Dim flagAffiche As Boolean = True
```

Comme la conversion doit se déclencher automatiquement lorsque le texte de BoiteF ou BoiteE change, j'utilise les évènements 'TextChanged' de ces TextBox.

Pour la conversion Euros=>Francs, dans la procédure TextChanged de BoiteE, je récupère le texte tapé (BoiteE.Text), j'appelle la fonction ConversionEF en lui envoyant comme paramètre ce texte.

La fonction me retourne un double que je transforme en string et que j'affiche dans l'autre TextBox(BoiteF).

```
Private Sub BoiteE_TextChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BoiteE.TextChanged
If flagAffiche = True Then
flagAffiche = False
BoiteF.Text = (ConversionEF(BoiteE.Text)).ToString
flagAffiche = True
End If
End Sub
```

Idem pour l'autre TextBox :

```
Private Sub BoiteF_TextChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles BoiteF.TextChanged
If flagAffiche = True Then
flagAffiche = False
BoiteE.Text = (ConversionFE(BoiteF.Text)).ToString
flagAffiche = True
End If
End Sub
End Class
```

A quoi sert le flag : flagAffiche?
A éviter une boucle sans fin, sans flag, BoiteF_TextChanged modifie BoiteE_Text qui déclenche BoiteE_TextChanged qui modifie BoiteF_Text qui déclenche BoiteF_TextChanged...
Avec le flag, quand je vais modifier la propriété Text d'une TextBox, le met le flag à False, cela indique à l'autre évènement TextChanged de ne pas lui aussi convertir et afficher.

Enfin il faut écrire les procédures qui font la conversion : ConversionEF et ConversionFE dans un module standard. Ces procédures 'Function' appellent elles mêmes une autre fonction qui arrondi les résultats à 2 décimales.

Pour transformer des Euros en Francs, je les multiplie par 6.55957 puis j'arrondis.

On remarque que ces procédures reçoivent une string en paramètres et retourne un double.

```
Module Module1
Public Function ConversionEF(ByVal e As String) As Double
Dim somme As Double
Dim resultat As Double
somme = Val(e)
resultat = Arrondir(somme * 6.55957)
Return resultat
End Function

Public Function ConversionFE(ByVal e As String) As Double
Dim somme As Double
Dim resultat As Double
somme = Val(e)
resultat = Arrondir(somme / 6.55957)
Return resultat
End Function
```

Enfin la Function Arrondir arrondit à 2 décimales: pour cela on multiplie par 100, on arrondit à l'entier avec Round puis on divise par 100.

```
Public Function Arrondir(ByVal Valeur As Double) As Double
'arrondi a 2 chiffres après la virgule
Return (Math.Round(Valeur * 100)) / 100
End Function
End Module
```

A noter que l'on aurait pu utiliser une surcharge de Round qui arrondit directement à 2 décimales :

```
Return (Math.Round(Valeur, 2))
```

Exercice:
Quel code mettre dans la procédure Button_Click d'un bouton nommé 'Remise à zéro' qui met les 2 zones de saisie à zéro ?

(Pensez au flag)

E 3.2 Exemple : Mensualités d'un prêt

Comment créer un programme qui calcul les mensualités d'un prêt ?

Dans l'espace Microsoft.VisualBasic il existe des fonctions financières.
Pmt calcul les mensualités d'un prêt.
Remboursement mensuel= Pmt(Rate, NPer, PV, FV, Due)

Rate
: Obligatoire. Donnée de type Double indiquant le taux d'intérêt par période. Si taux d'intérêt annuel de 10 pour cent et si vous effectuez des remboursements mensuels, le taux par échéance est de 0,1/12, soit 0,0083.

NPer
: Obligatoire. Donnée de type Double indiquant le nombre total d'échéances. Par exemple, si vous effectuez des remboursements mensuels dans le cadre d'un emprunt de quatre ans, il y a 4 * 12 (soit 48) échéances.

PV
: Obligatoire. Double indiquant la valeur actuelle. Par exemple, lorsque vous empruntez de l'argent pour acheter une voiture, le montant du prêt correspond à la valeur actuelle (pour un emprunts il est négatif).

FV
: Facultatif. Double indiquant la valeur future ou le solde en liquide souhaité au terme du dernier remboursement. Par exemple, la valeur future d'un emprunt est de 0 F car il s'agit de sa valeur après le dernier remboursement. Par contre, si vous souhaitez économiser 70 000 F sur 15 ans, ce montant constitue la valeur future. Si cet argument est omis, 0 est utilisée par défaut.

Due
: Facultatif. Objet de type Microsoft.VisualBasic.DueDate indiquant la date d'échéance des paiements. Cet argument doit être DueDate.EndOfPeriod si les paiements sont dus à terme échu ou DueDate.BegOfPeriod si les paiements sont dus à terme à échoir (remboursement en début de mois).
Si cet argument est omis, DueDate.EndOfPeriod est utilisé par défaut.

Notez que si Rate est par mois NPer doit être en mois, si Rate est en année NPer doit être en année.

```
Sub CalculPret()
Dim PVal, Taux, FVal, Mensualite, NPerVal As Double
Dim PayType As DueDate

Dim Response As MsgBoxResult
Dim Fmt As String
Fmt = "###,###,##0.00"          ' format d'affichage.
FVal = 0 '0 pour un prêt.

PVal = CDbl(InputBox("Combien voulez-vous emprunter?"))
Taux = CDbl(InputBox("Quel est le taux d'intérêt annuel?"))

If Taux > 1 Then Taux = Taux / 100 ' Si l'utilisateur à tapé 4 transformer en 0.04.
        NPerVal =12* CDbl(InputBox("Durée du prêt (en années)?"))
        Response = MsgBox("Echéance en fin de mois?", MsgBoxStyle.YesNo)
        If Response = MsgBoxResult.No Then
        PayType = DueDate.BegOfPeriod
        Else
                PayType = DueDate.EndOfPeriod

End If

Mensualite = Pmt(Taux / 12, NPerVal, -PVal, FVal, PayType)
MsgBox("Vos mensualités seront de " & Format(Mensualite, Fmt) & " par mois") End
Sub
```

IPmt calcul les intérêts pour une période.

Calculons le total des intérêts :

```
Dim IntPmt, Total, P As Double
For P = 1 To TotPmts ' Total all interest.
IntPmt = IPmt(APR / 12, P, NPerVal, -PVal, Fval, PayType) Total
= Total + IntPmt
Next Perio
```

SECTION 4 : Ce qu'il faut savoir pour faire un Programme

4.1 Démarrer et Arrêter un programme

Quand vous démarrez votre programme, quelle partie du code va être exécutée en premier ?

Vous pouvez le déterminer en cliquant sur le menu Projet puis Propriétés de NomduProjet, une fenêtre Page de propriétés du projet s'ouvre.

Sous la rubrique Objet du démarrage, il y a une zone de saisie avec liste déroulante permettant de choisir :

- Le nom d'une fenêtre du projet

ou

- Sub Main()

Démarrer par une fenêtre

Si vous tapez le nom d'une fenêtre du projet, c'est celle-ci qui démarre : cette fenêtre est chargée au lancement du programme et la procédure Form_Load de cette fenêtre est effectuée.

Démarrer par Sub Main()

C'est cette procédure Sub Main qui s'exécute en premier lorsque le programme est lancé. Dans ce cas, il faut ajouter dans un module (standard ou d'une feuille) une Sub nommé Main(),

Exemple :
En mode conception Form1 a été dessinée, C'est le modèle 'la Classe' de la fenêtre qui doit s'ouvrir au démarrage.

Dans Sub Main(), on crée une fenêtre de départ que l'on nomme initForm avec le moule, la Class Form1 en 'instancant' la nouvelle fenêtre.

```
Public Shared Sub Main()
    Dim initForm As New Form1
    initForm.ShowDialog()
End Sub
```

Fenêtre Splash

Dans la Sub Main il est possible de gérer une fenêtre Splash.

C'est une fenêtre qui s'ouvre au démarrage d'un programme, qui montre simplement une belle image, pendant ce temps le programme initialise des données, ouvre des fichiers... ensuite la fenêtre 'Splash' disparaît et la fenêtre principale apparaît.

Exemple :
Je dessine Form1 qui est la fenêtre Spash.

Dans Form2 qui est la fenêtre principale, j'ajoute :

```
Public Shared Sub Main()
Dim FrmSplash As New Form1        'instance la fenêtre Splash
Dim FrmPrincipal As New Form2     'instance la feuille principale
FrmSplash.ShowDialog()     'affiche la fenêtre Splash en Modale
FrmPrincipal.ShowDialog()  'a la fermeture de Splash, affiche la fenêtre principale
End Sub
```

Dans Form1 (la fenêtre Splash)

```
Private Sub Form1_Activated
Me.Refresh() 'pour afficher totalement la fenêtre.

'Ici ou on fait plein de choses on ouvre des fichiers ou on perd du temps.
Me.Close()
End Sub
```

On affiche FrmSplash un moment (Ho! la belle image) puis on l'efface et on affiche la fenêtre principale. Word, Excel... font comme cela.

Comment arrêter le programme ?

```
Me.Close()     'Ferme la fenêtre en cours
```

Noter bien Me désigne le formulaire, la fenêtre en cours.

```
Application.Exit()      'Ferme l'application
```

Si des fichiers sont encore ouverts, cela les ferme. (Il vaut mieux les fermer avant, intentionnellement par une procédure qui ferme tous les fichiers.)

4.2 Ouvrir un autre formulaire (une fenêtre)

Rappel:Formulaire=fenêtre

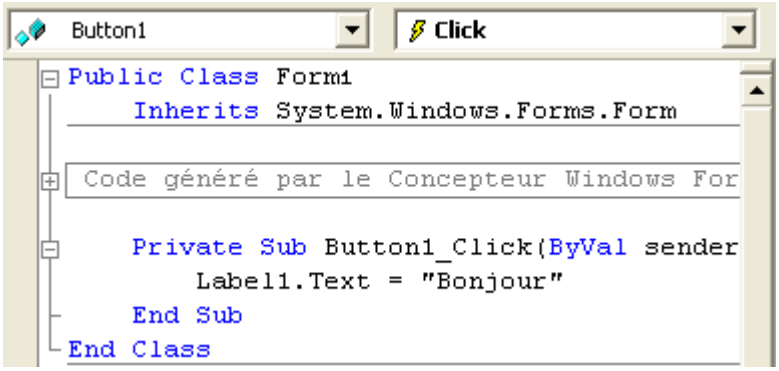
Comment à partir d'un formulaire Form1 ouvrir un second formulaire Form2 ?

Créer un formulaire

A- On va d'abord créer la Classe Form2

Ajouter un formulaire (Menu Projet, Ajouter un formulaire au projet) nommé Form2 .
On se rend compte que quand on ajoute un formulaire (Form2 par exemple), VB crée une nouvelle classe' Class Form2'qui hérite de System.Windows.Forms.Form , qui hérite donc de toutes les propriétés et méthodes de la Classe Form qui est la classe 'formulaire'.

```
Public Class Form2
End Class
```



Elle contient du code généré automatiquement par le concepteur Windows Forms et les procédures liées aux évènements.

Dessinez dans Form2 les contrôles nécessaires.

B- On va créer la fenêtre

Pour créer un nouveau formulaire dans le programme, il faut :

- Créer un formulaire à partir du moule, de la Classe Form2, cela s'appelle 'Instancer' un formulaire avec le mot New.
- Ouvrir ce formulaire, la faire apparaître, (avec ShowDialog, c'est un formulaire modal)

```
Dim f As New Form2()
f.ShowDialog()
```

En conclusion :
Le fait d'ajouter un formulaire et des contrôles à un projet crée une Class, (un moule) ce qui permet ensuite d'instancer un objet formulaire.

Dénomination des fenêtres après leur création

Une procédure crée un formulaire par Dim f As New Form2

- Dans le formulaire f créé:

Utiliser Me pour désigner le formulaire où on se trouve. (Form2 ou f ne sont pas acceptés)

Exemple :
Le formulaire f pourra être fermé par Me.close() dans le code du bouton Quitter par exemple.

- Hors du formulaire f, dans la procédure où a été instancé le formulaire:

Utiliser f pour désigner le formulaire.

Exemple :
Si la fenêtre appelante veut récupérer des informations dans le formulaire f (un texte dans txtMessage par exemple), il faudra écrire.

```
Text=f.txtMessage.Text
```

- Par contre, hors de la procédure qui a créée le formulaire, f n'est pas accessible.


En résumé: Attention donc, si vous instancez un formulaire dans une procédure, elle sera visible et accessible uniquement dans cette procédure.

Cela parait évident car un formulaire est un objet comme un autre et sa visibilité obéit aux règles habituelles (J'ai mis malgré tout un certains temps à le comprendre!!!).

Si vous voulez créer un formulaire qui soit visible dans la totalité du programme et dont les contrôles ou propriétés soient accessible par l'ensemble du programme, il faut l'instancer dans un module standard avec :

```
Public f As New Form2.
```


Un formulaire est un objet et sa visibilité obéit aux règles habituelles: Il peut être instancé dans une procédure, un module, précédé de 'Public','Public','Private'... Ce qui permet de gérer son accessibilité.

Un formulaire est un objet, et, on peut ajouter à un formulaire des méthodes et des membres

Pour ajouter une méthode à à un formulaire, il faut créer une Sub Public Public dans le corps de la fenêtre :

```
Public Sub Imprime()
        Code d'impression
End Sub
```

Si une instance de la fenêtre se nomme F, F.Imprime() exécute la méthode Imprime (donc la sub Imprime)

De même, pour définir un membre d'un formulaire, il faut ajouter une variable public.

```
Public Utilisateur As String
```

Permet d'utiliser en dehors du formulaire F.Utilisateur

Si le formulaire é été instancé dans un module de Classe et précédé de Public, les méthodes et propriétés de ce formulaire seront accessibles de partout.

Exemple plus complet

Avec récupération de données dans le formulaire créé, à partir d'une procédure :

Créer un formulaire en utilisant Form2.
L'ouvrir en formulaire modal.Quand l'utilisateur ferme cette fenêtre modale, récupérer le texte qui est dans txtMessage de cette fenêtre modale.

La ruse c'est de mettre dans le code du bouton Quitter de Form2 Me.Hide() pour rendre la fenêtre Form2 invisible mais accessible (et pas Me.Close() qui détruirait la fenêtre, le contrôle txtMessage et son contenu).

```
Dim f As New Form2()
f.ShowDialog()
Text=f.txtMessage.Text
f.Close()
```

Une fois que le texte à été récupéré, on faire disparaître la fenêtre f.
En réalité, curieusement, il semble que les propriétés de f soient accessibles même après un Close!!!

Autre problème, comment savoir si un formulaire existe, s'il n'existe pas le créer, s'il existe le rendre visible et lui donner la main :

```
If f Is Nothing Then    'Si f=rien
        f = New Form2
        f.ShowDialog()
    Else
    If f.Visible = False Then
        f.Visible = True
    End If
    f.Activate()
End If
```

Fenêtre modale ou non modale

Un formulaire modal est un formulaire qui une fois ouvert prend la main, interdit l'usage des autres fenêtres. Pour poursuivre, on ne peut que sortir de cette fenêtre.

Exemple typique : une MessageBox est un formulaire modal, les fenêtres d'avertissement dans Windows sont aussi modales.

Pour ouvrir un formulaire modal, il faut utiliser la méthode .ShowDialog

```
f.ShowDialog()
```

Noter, et c'est très important, que le code qui suit .showDialog est exécuté après la fermeture de la fenêtre modale.

Pour un formulaire non modal faire :

```
f.Show()
```

Dans ce cas le formulaire f s'ouvre, le code qui suit .Show est exécuté immédiatement, et il est possible de passer dans une autre fenêtre de l'application sans fermer f.

Owner

Comment savoir quel formulaire a ouvert le formulaire en cours ? (Quel est le formulaire parent?)

ShowDialog possède un argument facultatif, owner, qu'on peut utiliser afin de spécifier une relation parent-enfant pour un formulaire. Par exemple, lorsque le code de votre formulaire principal affiche une boîte de dialogue, vous pouvez passer Me comme propriétaire de la boîte de dialogue, afin de désigner votre formulaire principal comme propriétaire, comme le montre le code de l'exemple suivant :
Dans Form1

```
Dim f As New Form2
f.ShowDialog(Me)
```

Dans Form2 on peut récupérer le nom du 'propriétaire', du 'parent' qui a ouvert la fenêtre (il est dans Owner) et l'afficher par exemple :

```
Label1.text=Me.Owner.ToString
```

Cela affiche : NomApplication.Form1,text text=est le texte de la barre supérieure.

Récupération d'information par par DialogResult

On ouvre un formulaire modal, , comment, après sa fermeture, récupérer des informations sur ce qui s'est passé dans ce formulaire modale ?

Par exemple, l'utilisateur a t-il il cliqué sur le bouton Ok ou le bouton Cancel pour fermer le formulaire modale ?

Pour cela on va utiliser une propriété DialogResult des boutons, y mettre une valeur correspondant au bouton, quand l'utilisateur clique sur un bouton, on, la valeur de la propriété DialogResult du bouton est assignée à la propriété DialogResult du formulaire, on récupère cette valeur à la fermeture du formulaire modal.

Dans le formulaire modal Form2 on met :

```
ButtonOk.DialogResult= DialogResult.ok

ButtonCancel.DialogResult= DialogResult.Cancel
```

Dans le formulaire qui appelle :

```
Form2.ShowDialog()
If form2.DialogResult= DialogResult.ok then
        'l'utilisateur a cliquer sur le bouton ok
End if
```

Remarque :

1. On utilise comme valeur de DialogResult les constantes de l'énumération DialogResult:DialogResult.ok .Cancel .No .Yes .Retry .None
2. Si l'utilisateur clique sur la fermeture du formulaire modal (bouton avec X) cela retourne DialogResult.cancel
3. on peut aussi utiliser la syntaxe : If form2.ShowDialog(Me) = System.Windows.Forms.DialogResult.OK Then qui permet en une seule ligne d'ouvrir form2 et de tester si l'utilisateur a cliqué sur le bouton ok de form2.
4. La fermeture du formulaire modal par le bouton de fermeture ou l'appel de la méthode Close ne détruit pas toujours le formulaire modal, il faut dans ce cas utiliser la méthode Dispose pour le détruire.

Mon truc:De manière générale s'il y a des informations à faire passer d'un formulaire à un autre, j'utilise une variable Public (nommée BAL comme 'Boite aux lettres' par exemple) dans laquelle je met l'information à faire passer.

Bouton par défaut

Parfois dans un formulaire, l'utilisateur doit pouvoir, valider (taper sur la touche 'Entrée') pour accepter et quitter rapidement le formulaire (c'est l'équivalent du bouton 'OK') ou taper 'Echap' pour sortir du formulaire sans accepter (c'est l'équivalent du bouton 'Cancel').

Il suffit pour cela de donner aux propriétés AcceptButton et CancelButton du formulaire, le nom des boutons ok et cancel qui sont sur la feuille.

```
form1.AcceptButton = buttonOk
form1.CancelButton = buttonCancel
```

Si l'utilisateur tape la touche 'Echap' buttonCancel_Click est exécuté.

4.3 Traiter les erreurs

Il y a plusieurs types d'erreurs :

- Les erreurs de syntaxe.
- Les erreurs d'exécution.
- Les erreurs de logique.

Les erreurs de syntaxe

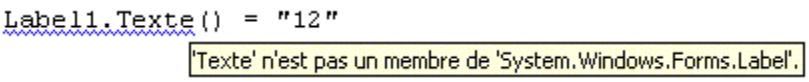
Elles surviennent en mode conception quand on tape le code :

Exemple :

```
A+1=B          'Erreur dans l'affectation
f.ShowDialogue 'Faute de frappe, il fallait taper ShowDialog
2 For... et un seul Next
```

Dans ces cas VB souligne en ondulé bleue le code. Il faut mettre le curseur sur le mot souligné, l'explication de l'erreur apparaît.

Exemple :
Propriété Text d'un label mal orthographiée.



Elles sont parfois détectées en mode Run.

Erreur dans une conversion de type de données par exemple.
Il faut les corriger immédiatement en tapant le bon code.

Les erreurs d'exécution

Elles surviennent en mode Run ou lors de l'utilisation de l'exécutable, une instruction ne peut pas être effectuée. Le Le logiciel s'arrête brutalement, c'est très gênant!! Pour l'utilisateur c'est un 'BUG'

L'erreur est:

- Soit une erreur de conception.

Exemple :
Ouvrir un fichier qui n'existe pas (On aurait du vérifier qu'il existe avant de l'ouvrir!).
Division par zéro.

Utiliser un index d'élément de tableau supérieur au plus grand possible :

```
Dim A(3) As String: A(5)="Toto"
```

- Soit une erreur de l'utilisateur.

Exemple :
On lui demande de taper un chiffre, il tape une lettre ou rien puis valide.

Il faut toujours vérifier ce que fait l'utilisateur et prévoir toutes les possibilités.

Exemple :
Si je demande à l'utilisateur de tapez un nombre entre 1 et 10, il faut:

- Vérifier qu'il a tapé quelque chose.
- Que c'est bien un chiffre (pas des lettres).
- Que le chiffre est bien entre 1 et 10.

Sinon il faudra reposer la question.

On voit bien que pour éviter les erreurs d'exécution il est possible :

- D'écrire du code gérant ces problèmes, contrôlant les actions de l'utilisateur…
- Une autre alternative est de capter l'erreur.

Capter les erreurs avec Try Catch Finally

Avant l'instruction supposée provoquer une erreur indiquez : Essayer (Try), si une erreur se produit Intercepter l'erreur (Catch) puis poursuivre (après Finally)

```
Try
        Instruction susceptible de provoquer une erreur
Catch
        Traitement de l'erreur
Finally
        Code toujours exécuté
End Try
```

Il faut pour que cela fonctionne avoir tapé au préalable Imports System.IO

Il est possible d'utiliser Catch pour récupérer l'objet 'Exception' qui est généré par l'erreur.

```
Catch ex As Exception
```

Cet objet Exception à des propriétés :
Message qui contient le descriptif de l'erreur.
Source qui contient l'objet qui a provoqué l'erreur....
ex.Message contient donc le message de l'erreur.

Cet objet Exception (de l'espace IO) à aussi des classes dérivées : StackOverFlowException; FileNotFoundExeption; EndOfStreamExeption; FileLoadExeption; PathTooLongExeption.

Enfin une exception peut provenir de l'espace System: ArgumentExceptions; ArithmeticException; DivideByZeroExeception.....

Il est possible d'écrire plusieurs instructions Catch avec pour chacune le type de l'erreur à intercepter. (Faisant partie de la classe Exceptions)

Exemple :
On ouvre un fichier par StreamReader , comment intercepter les exceptions suivantes?
Répertoire non valide
Fichier non valide
Autre.

```
Try
        sr= New StreamerReader (NomFichier)
Catch ex As DirectoryNotFoundException
        MsgBox("Répertoire invalide")
Catch ex As FileNotFoundException
        MsgBox("Fichier invalide")
Catch ex As Exception
        MsgBox(ex.Message)

End Try
```

Noter que le dernier Catch intercepte toutes les autres exceptions.
On peut encore affiner la gestion par le mot clé When qui permet une condition.

```
Catch ex As FileNotFoundException
    When ex.Message.IndexOf ("Mon Fichier.txt") >0
    MsgBox ("Impossible d'ouvrir Mon Fichier.txt")
```

Si le texte "Mon Fichier.txt" est dans le message, affichez que c'est lui qui ne peut pas être ouvert.
Exit Try permet de sortir prématurément.

Capter les erreurs avec On error

On peut aussi utiliser en VB.Net la méthode VB6 :
On Error Goto permet en cas d'erreur de sauter à une portion de code traitant l'erreur.
On peut y lire le numéro de l'erreur qui s'est produite, ce numéro est dans Err.Number.

Err.Description contient le texte décrivant l'erreur. Err.Source donne le nom de l'objet ou de l'application qui a crée l'erreur.

Quand l'erreur est corrigée, on peut revenir de nouveau effectuer la ligne qui a provoqué l'erreur grâce à Resume ou poursuivre à la ligne suivante grâce à Resume Next

Exemple :

```
On Error GoTo RoutinedErreur 'Si une erreur se produit se rendre à 'RoutineErreur'
Dim x As Integer = 33
Dim y As Integer = 0
Dim z As Integer
z = x / y ' Crée une division par 0 !!

RoutinedErreur: ' La Routine d'erreur est ici (remarquer le ':').
Select Case Err.Number ' On regarde le numéro de l'erreur.
Case 6 ' Cas : Division par zéro interdite
    y = 1 ' corrige l'erreur.
Case Else
    ' Autres erreurs....
End Select
Resume ' Retour à la ligne qui a provoqué l'erreur.
```

Pour arrêter la gestion des erreurs il faut utiliser :

```
On Error Goto 0
```

Parfois on utilise une gestion hyper simplifiée des erreurs:

Si une instruction 'plante', la sauter et passez à l'instruction suivante, pour cela on utilise:

```
On Error Resume Next
```

Exemple :
On veut effacer un fichier

```
On Error Resume Next
Kill (MonFichier)
On Error goto 0
```

Ainsi, si le fichier n'existe pas, cela ne plante pas (on aurait pu aussi vérifier qu'il existe avant de l'effacer).

On Error Gosub n'existe plus.

Les erreurs de logique

Le programme fonctionne, pas d'erreurs apparentes, mais les résultats sont erronés, faux.

Il faut donc toujours tester le fonctionnement du programme plusieurs fois dans les conditions réelles avec des données courantes, mais aussi avec des données remarquables (limites supérieures, inférieures, cas particuliers..) pour voir si les résultats sont cohérents et exacts.

Et avoir une armée de Bêta-testeurs.

Une fois l'erreur trouvée, il faut déterminer la cause et la corriger.

Ou bien elle **est évidente à la lecture du code** ou bien elle n'est pas évidente et c'est l'horreur.
Dans ce dernier cas il faut analyser le fonctionnement du programme pas à pas, instruction par instruction en surveillant la valeur des variables. (Voir la rubrique débogage)

Les erreurs les plus communes sont :

- Utilisation d'un mauvais nom de variable (La déclaration obligatoire des variables évite cela)
- Erreur dans la portée d'une variable.
- Erreur dans le passage de paramètres (Attention au ByVal et ByRef)
- Erreur dans la conception de l'algorithme.
- ...

Quelques règles permettent de les éviter : voir leçon 7.2

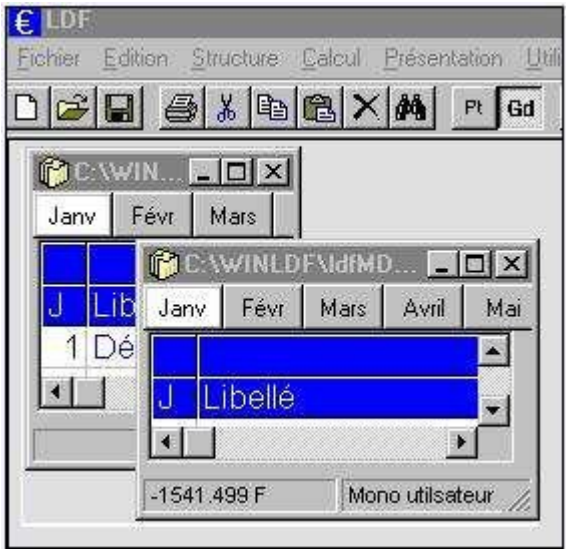
4.4 Travailler sur une fenêtre multidocument

Comment créer un programme MDI (Multi Document Interface) ?

Comprendre les programmes MDI

L'exemple de Word : la fenêtre principale (fenêtres MDI) contient les menus en haut, on peut ouvrir plusieurs documents dans des fenêtres filles.

Ci dessous l'exemple de LDF (Programme de comptabilité écrit par l'auteur) :



On a une fenêtre MDI (conteneur) contenant 2 fenêtres filles affichant chacune une année de comptabilité.

Dans VB.NET, un MDIForm (fenêtres principale MDI) est une fenêtre quelconque dont la propriété IsMDIContainer = true.
Dans la fenêtre fille, la propriété MDIParent indique le conteneur (C'est à dire le nom de la fenêtre MDI).

Les applications MDI peuvent avoir plusieurs conteneurs MDI.

Exemple d'un programme MDI.
On va créer une Form1 qui est le conteneur.
Une Form2 qui est la fenêtre fille.
Dans Form1 le menu principal contient la ligne '&Nouvelle' qui crée une nouvelle instance de la fenêtre fille.

Création de la fenêtre conteneur parent

Créer la fenêtre Form1 :

Dans la fenêtre Propriétés, affectez la valeur true à la propriété IsMDIContainer. Ce faisant, vous désignez la fenêtre comme le conteneur MDI des fenêtres enfants.

Remarque : Affecter la valeur Maximized à la propriété WindowState, car il est plus facile de manipuler des fenêtres MDI enfants lorsque le formulaire parent est agrandi. Sachez par ailleurs que le formulaire MDI parent prend la couleur système (définie dans le Panneau de configuration Windows).

Ajouter les menus du conteneur :
A partir de la boîte à outils, faites glisser un contrôle MainMenu sur le formulaire. Créez un élément de menu de niveau supérieur en définissant la propriété Text avec la valeur &File et des éléments de sous-menu appelés &Nouvelle et &Close. Créez également un élément de menu de niveau supérieur appelé &Fenêtre.

Dans la liste déroulante située en haut de la fenêtre Propriétés, sélectionnez l'élément de menu correspondant à l'élément &Fenêtre et affectez la valeur true à la propriété MdiList.

Vous activez ainsi le menu Fenêtre qui permet de tenir à jour une liste des fenêtres MDI enfants ouvertes et indique à l'utilisateur par une coche la fenêtre enfant active.

Il est conseillé de créer un module standard qui instance la fenêtre principale et qui contient une procédure Main qui affiche la fenêtre principale :

```
Module StandartGénéral
Public FrmMDI as Form1
Sub Main()
        FrmMDI.ShowDialog()
End sub
End Module
```

Noter bien que FrmMDI est donc la fenêtre conteneur et est Public donc accessible à tous.

Création des fenêtres filles

Pour créer une fenêtre fille, il suffit de donner à la propriété MDIParent d'une fenêtre le nom de la fenêtre conteneur.

Dessiner dans Form2 les objets nécessaires dans la fenêtre fille.

Comment créer une instance de la fenêtre fille à chaque fois que l'utilisateur clique sur le menu '&Nouvelle'?

En premier lieu, déclarez dans le haut du formulaire Form1 une variable MDIFilleActive qui contiendra la fenêtre fille active.

```
Dim MDIFilleActive As Form2
```

La routine correspondant au MenuItem &Nouvelle (dans la fenêtre MDI) doit créer une instance de la fenêtre fille :

```
Protected Sub MDIChildNouvelle_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MenuItem2.Click
        MDIFilleActive = New Form2()
        'Indique à la fenêtre fille son 'parent'.
        MDIFilleActive.MdiParent = Me
        'Affiche la fenêtre fille
        MDIFilleActive.Show()
End Sub
```

Comment connaître la fenêtre fille active ?

Quand on en a ouvert plusieurs ?
La fenêtre fille active est dans Me.ActiveMdiChild du conteneur

Comment voir s'il existe une fenêtre active ?

```
If Not (ActiveMdiChild=Nothing) then        'elle existe
```

En mettant dans la variable MDIFilleActive la fenêtre active, on est sur sur de l'avoir toujours à disposition : pour cela dans la procédure Form1_MdiActivate (qui se produit à chaque fois que l'on change de fenêtre fille) je récupère Me.ActiveMdiChild qui retourne la fenêtre fille active.
Dans Form1

```
Private Sub Form1_MdiChildActivate..
        MDIFilleActive=Me.ActiveMdiChild
End Sub
```

Il faut comprendre que peut importe le nom de la fenêtre fille activ ve, on sait simplement que la fenêtre fille active est dans MDIFilleActive, variable que l'on utilis ilise pour travailler sur cette fenêtre fille.

Comment avoir accès aux objets de la fenêtre fille à partir du conteneur ?

De la fenêtre conteneur j'ai accès aux objets de la fenêtre fille par l'intermédiaire de la variable MDIFilleActive précédemment mise à jour; par exemple le texte d'un label :

```
MDIFilleActive.label1.text
```

Comment parcourir toutes les fenêtres filles ?

La collection MdiChildren contient toutes les fenêtres filles, on peut les parcourir :

```
Dim ff As Form2
For Each ff In Me.MdiChildren
...
Next
```

Comment avoir accès aux objets du conteneur à partir de la fenêtre fille ?

En utilisant Me.MdiParent qui contient le nom du conteneur.
Dans la fenêtre fille le code Me.MdiParent.text ="Document 1" affichera 'Document 1' dans la barre de titre du conteneur.

Comment une routine du module conteneur appelle une routine dans la fenêtre fille active ?

Si une routine public de la fenêtre fille se nomme Affiche, on peut l'appeler par :

```
MDIFilleActive.Affiche()
```

Il n'est pas possible d'appeler les évènements liés aux objets.

Agencement des fenêtres filles

La propriété LayoutMdi de la fenêtre conteneur modifie l'agencement des fenêtres filles.

0 - MdiLayout.Cascade
1 - MdiLayout.TileHorizontal
2 - MdiLayout.TileVertical
3 - MdiLayout.ArrangeIcons

Exemple :
Le menu Item Cascade met les fenêtres filles en cascade.

```
Protected Sub CascadeWindows_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
        Me.LayoutMdi(System.Windows.Forms.MdiLayout.Cascade)
End Sub
```

4.5 Travailler sur le temps : dates, heure, Timers

On a vu qu'il existe un type de variable 'DateTime' pour gérer les dates et heures, comment l'utiliser ?

Nous verrons aussi comment utiliser les Timers pour déclencher des évènements à intervalle régulier.

Enfin comment perdre du temps ?

DateTime

```
Une variable DateTime        'Contient une date plus l'heure.
```
Elle occupe 8 octets. (64 bits)

Peut contenir une date comprises entre le 1er janvier de l'année 1 et le 31 décembre 9999 et des heures comprises entre 0:00:00 (minuit) et 23:59:59.

En fait ce qui est codé dans la variable DateTime est le nombre de graduations (Une graduation= 100 nanosecondes.) écoulées à compter de minuit, le 1er janvier de l'année 1 jusqu'a la date codée.

Nb: DateTime fait partie d'une Classe .Net , il existe aussi un type nommé Date qui contient aussi une date et l'heure et qui fait partie de VB mais qui n'est pas une classe.

Saisir une date, une heure

Pour saisir une valeur DateTime en utilisant un littéral: elle doit être placée entre des signes (#) et son format doit être de type d/m/yyyy, par exemple #31/5/1998#.

```
Dim DateNaissance As Date
DateNaissance= #02/12/1951#
```

Autre manière de saisir une date, une heure :

```
Dim date1 As New System.DateTime(1996, 6, 3, 22, 15, 0)     'Année,        mois,        jour,
```
heure,minute, seconde, et éventuellement millisecondes)

Afficher une date, une heure

Pour afficher les dates et heures simplement, il suffit d'utiliser .ToString

```
MsgBox(DateNaissance.ToString)  'Affichera 02/12/1951 11:00:00
```

C'est le format utilisé par l'ordinateur (en fonction du pays)
ToString peut comporter des arguments qui formatent l'affichage :

Voici quelques codes de formatage:

D	affiche le jour	2
Dd	affiche le jour sur 2 chiffres	02
Ddd	affiche le jour abrégé	Dim.
Dddd	affiche le jour complet	Dimanche
M	affiche le mois	12
MM	affiche le mois sur 2 chiffres	12
MMM	affiche le mois abrégé	déc
MMMM	affiche le mois complet	décembre
y, yy, yyyy	affiche 1 à 2 chiffres, deux chiffres ou quatre chiffre	51, 51, 1951

H affiche l'heure sur un ou deux chiffres (format 24h)
HH affiche l'heure sur 2 chiffres
h et hh font de même mais avec un format 12 h.
t, tt affiche l'heure en format 12h plus A ou P (pour matin, après midi)
m, mm, s, ss, f, ff font de même pour les minutes, secondes et millisecondes.
: et / sont les séparateurs heure et date.

Exemple :

```
MsgBox(DateNaissance.ToString("dddd d MMMM yyyy"))        'Affichera  Dimanche  2 décembre 1951
MsgBox(DateNaissance.ToString("hh:mm")        'Affichera 11:00
MsgBox(DateNaissance.ToString("d/MM/yy")      'Affichera  02/12/51
MsgBox(DateNaissance.ToString("%h)     'Affichera 11 le caractère % est utilisé quand on affiche une seule donnée.
```

On peut enfin utiliser les méthodes de la classe DateTime!!

```
DateNaissance.ToLongDateString          'dimanche 02 décembre 1951
DateNaissance.ToShortDateString         `02/12/1951
DateNaissance.ToLongTimeString          '11:00:00
DateNaissance.ToShortTimeString         '11:00
```

Variable « temps »

Un TimeSpan est une unité de temps exprimée en jours, heures, minutes, secondes;
Un TimeSpan initialisé avec 1.0e+13 graduations représente "11.13:46:40", ce qui correspond à 11 jours, 13 heures, 46 minutes et 40 secondes.
L'espace de nom System.DateTime. contient une multitude de membre :

Add Substract

On peut ajouter ou soustraire un TimeSpan à un DateTime, on obtient un DateTime.
En clair on peut ajouter à une date une durée, on obtient une date.
' Quel sera la date dans 36 jours ?

```
Dim today As System.DateTime
Dim duration As System.TimeSpan
Dim answer As System.DateTime

today = System.DateTime.Now
duration = New System.TimeSpan(36, 0, 0, 0)
answer = today.Add(duration)
```

On peut ajouter ou soustraire 2 dates, on obtient une TimeSpan

```
Dim diff1 As System.TimeSpan
diff1 = date2.Subtract(date1)
```

AddDay, addMonth, AddHours, AddSeconds, AddMiliseconds

Permet d'ajouter des jours,ou des mois, ou des heures, ou des secondes, ou des millisecondes à une date, on obtient une date.

```
Answer=today.AddDay(36)
```

Year, Month, Day, Hour, Minute, Second, Milisecond

Permettent d'extraire l'année, le mois, le jour, l'heure, les minutes, les secondes, les millisecondes d'une date :

```
I=DateNaissance.Year        ' => I=1951
I=System.DateTime.Now.Day       'donne le jour d'aujourd'hui (1 à 31)
```

DayOfWeek

Retourne le jour de la semaine (0 pour dimanche à 6 pour samedi)

```
I=DateNaissance.DayOfWeek       'I=0 car le 02/12/1951 est un dimanche.
```

DayForYear existe aussi.

Now, ToDay, TimeOfDay

Now est la date et l'heure du système.(Là, maintenant)
ToDay est la date du système avec l'heure à 0.
TimeOfDay est l'heure actuelle.

Ticks

Donne le nombre de graduations d'un DateTime.
AddTicks peut être utilisé.

Comparaison de DateTime

On utilise Compare: DateTime.Compare(t1, t2) retourne 0 si t1=t2, une valeur positive si t1>t2 négative si t1<t2.

```
Dim t1 As New DateTime(100)
Dim t2 As New DateTime(20)

If DateTime.Compare(t1, t2) > 0 Then
        Console.WriteLine("t1 > t2")
End If
If DateTime.Compare(t1, t2) = 0 Then
        Console.WriteLine("t1 = t2")
End If
If DateTime.Compare(t1, t2) < 0 Then
        Console.WriteLine("t1 < t2")
End If
```

On peut aussi utiliser la méthode op_Equality de l'espace de nom pour voir si 2 dates sont égales:

```
areEqual = System.DateTime.op_Equality(april19, otherDate)
```

Il existe aussi op_GreaterThan et beaucoup d'autres.

Comment saisir rapidement une date dans un programme?

En ajoutant à une fenêtre un contrôle DateTimePicker
En mode Run, il apparaît une zone rectangulaire avec la date système dedans :

Si l'utilisateur clique sur la flèche déroulante, il apparaît une fenêtre calendrier.

Il suffit pour l'utilisateur de cliquer sur la bonne date.
Le programmeur récupére la date dans DateTimePicker1.value
Il existe, bien sur, de multiples propriétés et plusieurs évènements, le plus remarquable étant : ValueChanged.

MonthCalendar est un contrôle similaire mais qui reste toujours ouvert.
De plus grâce à CalendarDimension on peut afficher plusieurs mois.

Les Timers

Pour déclencher un évènement à intervalle régulier, il faut utiliser les minuteries ou Timer.
Prendre le contrôle Timer dans le Boite à outils, l'ajouter à la fenêtre. Il apparaît en bas sous la fenêtre dans la barre d'état des composants.

Il n'apparaît pas à l'utilisateur dans la fenêtre en mode Run.
Il est très simple à utiliser.

La propriété Interval contient la périodicité de l'événement Ticks , évènement qui se déclenche régulièrement.

Interval est en millisecondes. Pour Interval=500 l'évènement Ticks se déclenche toutes les 1/2 secondes.
Start et Stop déclenche et arrête la minuterie. (De même Enabled active ou non)

Exemple :
Faire clignoter un label toutes les 1/2 secondes.
Créer le label1
Ajouter un Timer1 (qui se place en bas sous la fenêtre)

```
Private Sub Form3_Load(...)
        Timer1.Interval = 500
        Timer1.Start()
End Sub

Private Sub Timer1_Tick(..)
        Label1.Visible = Not (Label1.Visible)
End Sub
```

Un évènement Timer_Tick se produit toutes les 1/2 secondes et inverse la valeur de la propriété visible du label. (S'il était égal à True, il devient égal à False et vice versa.)

Mais attention : Timer à des restrictions de taille :

- Si votre application ou une autre demande beaucoup au système (boucles longues, calculs complexes, accès intensifs à un périphérique, un réseau ou un port, par exemple), les événements de minuterie peuvent être moins fréquent que spécifié dans la propriété Interval. Il n'est pas garanti que l'intervalle s'écoule dans le temps exact!!
- L'intervalle peut être compris entre 1 et 64 767 millisecondes : l'intervalle le plus long ne dépasse pas de beaucoup la minute (64,8 secondes).

- Le système génère 18 graduations à la seconde (même si la valeur de la propriété Interval est mesurée en millisecondes, la véritable précision d'un intervalle ne dépassera pas un dix-huitième de seconde).

Donc pour faire clignoter un label : OUI
Pour compter précisément un intervalle de temps:NON
Mais il y a d'autres méthodes, voir le cours 7.3

Perdre du temps

Parfois on a besoin de perdre du temps :
Exemple ne rien faire pendant 3 secondes puis poursuivre…

- Il est exclu de faire des boucles vides:
  ```
  For i=0 to 100000    ' le temps écoulé est variable en fonction des machines…
  Next i
  ```

- Autre méthode : on boucle tant que l'heure courante est inférieure à l'heure du départ+3s
  ```
  Dim t As DateTime=DateTime.Now
  Do While DateTime.Now <t.AddSeconds(3)
  Loop
  ```

Mais cela accapare le processeur.

- On peut utiliser un Timer et vérifier dans la procédure Tick si le temps est écoulé.
- On peut utiliser Thread.Sleep
  ```
  System.Threading.Thread.Sleep(3000)
  ```

Chronométrer

Parfois on a besoin de chronométrer un évènement :
Voir le cours 7.4 - Chronométrer

L'exemple E4.1sur l'horloge est aussi didactique.

4.6 Les fichiers

Comment lire et écrire dans des fichiers du texte, des octets, du XML du Rtf ?

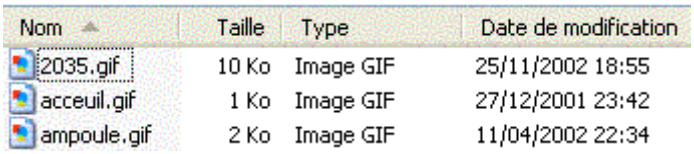
Généralités et rappels

Le mot 'fichier' est a prendre au sens informatique : ce n'est pas un ensemble de fiches mais plutôt un ensemble d'octets. Un fichier peut être un programme (Extension .EXE), du texte (Extension .TXT ou .DOC....), une image (Extension .BMP .GIF .JPG...), une base de données (.MDB..) du son, de la vidéo....

Pour travailler avec du texte, des octets, des données très simple (sans nécessité d'index, de classement..), on utilise les méthodes décrites dans cette page, travail direct dans les fichiers séquentiels, aléatoires, binaires. Mais dès que les informations sont plus structurées, il faut utiliser les bases de données (Il y a plusieurs chapitre plus loin traitant des Base de données).

Un fichier a un nom : 'Image.GIF', une extension : '.GIF' qui en indique généralement le type de contenu, des attributs (Longueur, Date de création, de modification, Fichier en lecture seule ou non..).

On voit cela dans l'explorer Windows :



Un fichier est composé d'enregistrements qui sont des 'paquets' de données; suivant le type de fichiers un enregistrement peut correspondre à une ligne, un octet, un groupe d'octets…

Comment utiliser les fichiers, voici le plan de cet article :

- Il est conseillé de travailler avec les Classes du Framework

Avec la Classe FileInfo.
On obtient des renseignements sur le fichier.
Pour lire écrire dans un fichier (en dehors des bases de données), il y a plusieurs méthodes.

Avec la Classe System.Io on a à notre disposition StreamReader StreamWriter BinaryReader BinaryWriter FileStream
Pour lire ou écrire dans un fichier, il faut l'ouvrir (Open), lire ou écrire en utilisant un flux de données (Stream) puis le refermer (Close).

Le Stream (flux, torrent, courant) est une notion générale, c'est donc un flux de données provenant ou allant vers un fichier, un port, une connexion TCP/IP...

L'accès est séquentiel : les données sont traitées du début à la fin du fichier.

- Il existe toujours la méthode classique du FileOpen

On ouvre le fichier en mode séquentiel, aléatoire, binaire, on lit X enregistrements, on referme le fichier.

- Avec certains objets, on gère automatiquement les lectures écritures sur le disque

Comme avec le RichTextBox par exemple.

En résumé, pour travailler sur les fichiers, on dispose :
- des instructions VB runtime traditionnelles: FileOpen WriteLine…
- des instructions du FSO (FileObjetSystem) pour la compatibilité avec les langages de script.
- de l'espace de nom System.IO avec les Class et objets .NET

Les 2 premiers font appel au troisième; donc pourquoi ne pas utiliser directement les Classe .NET?

Classe FileInfo et File, Stream

Pour travailler sur les fichiers, il faut au préalable taper :

```
Imports System.IO
```

La classe File est utilisée pour travailler sur un ensemble de fichier ou un fichier (sans instanciation préalable), la Classe FileInfo donne des renseignements sur un fichier particulier (Il faut instancer au préalable un objet FileInfo).

La Classe File possède les méthodes suivantes.

Exists	Teste si le fichier existe.
Create	Crée le fichier
Copy	Copie le fichier
Delete	Efface le fichier
GetAttributes , SetAttributes	Lire ou écrire les attributs
Move	Déplacement de fichier

Toutes les méthodes Open (pour un FileStream) OpenRead, OpenWrite, OpenText.

Exemple :
Un fichier existe-t-il? Afficher True s'il existe :

```
Label1.Text = File.Exists("vessaggi.gif").ToString
```

La Classe FileInfo possède les propriétés suivantes.

Name	Nom du fichier (sans extension)
FullName	Nom complet avec extension
Extension	Extension (.txt par exemple)
Length	Longueur du fichier
Directory	Répertoire parent
DirectoryName	Répertoire ou se trouve le fichier
Exists	
LastAccessTime	Date du dernier accès, LastWriteTime existe aussi
Attributes	Attributs

Il faut faire un AND entre Attributes et une valeur de l'énumération FileAttributes (Archive, Compressed, Directory, Encrypted, Hidden, Normal, ReadOnly, System, Temporaly).

Pour tester ReadOnly par exemple :

```
Fi.Attributes And FileAttributes.ReadOnly
'retourne True si le fichier est ReadOnly
```

Et les méthodes suivantes :
Create, Delete, MoveTo
AppendTex, CopyTo Open, OpenRead, OpenWrite, OpenText..

On voit que toutes les informations sont accessibles.

Exemple :
Pour un fichier, afficher successivement le nom, le nom avec répertoire, le répertoire, la longueur, la date de dernière écriture et si le fichier est en ReadOnly.

```
Dim sNom As String = "c:\monfichier.txt"
Dim Fi As FileInfo
Fi=New FileInfo( sNom)
        MsgBox("Nom="& Fi.Name)
        MsgBox("Nom complet ="& Fi.FullName)
        MsgBox("Répertoire="& Fi.DirectoryName)
        MsgBox("Longueur="& Fi.Lenght.ToString)
        MsgBox("Date der modification="& Fi.LastWriteTime.ToShortDateString)
        MsgBox("ReadOnly="& (Fi.Attributes And FileAttributes.ReadOnly).ToString)
```

Utiliser les « Stream »

Le Stream (flux, torrent, courant) est une notion générale, c'est donc un flux de données provenant ou allant vers un fichier, un port, une connexion TCP/IP...

Ici on utilise un Stream pour lire ou écrire dans un fichier.
L'accès est séquentiel: les données sont traitées du début à la fin du fichier.

Pour écrire dans un fichier texte :
Il faut instancier un objet de la classe StreamWriter. On écrit avec Write ou WriteLine (ajoute un saut de ligne). Enfin on ferme avec Close.

On peut instancer avec le constructeur de la classe StreamWriter et avec New, ou par la Classe File.

```
Dim SW As New StreamWriter ("MonFichier.txt") ' crée ou si existe écrase
```

Il existe une surcharge permettant d'ajouter à la fin du fichier :

```
Dim SW As New StreamWriter ("MonFichier.txt", True) ' crée ou si existe ajoute
```

Avec la classe File :

```
Dim SW As StreamWriter=File.CreateText ("MonFichier.txt") ' crée ou si existe écrase
Dim SW As StreamWriter = File.AppendText("MonFichier.txt") ' crée ou si existe ajoute
```

Ensuite pour écrire 2 lignes :

```
SW.WriteLine ("Bonjour")
SW.WriteLine ("Monsieur")
```

Enfin on ferme :

```
SW.Close()
```

Pour lire dans un fichier Texte :
Il faut instancier un objet de la classe StreamReader. On lit avec Read (un nombre d'octet) ReadLine (une ligne) ReadToEnd (de la position courante jusqu'à la fin). Enfin on ferme avec Close.

Avec le constructeur de la Classe Stream Reader :

```
Dim SR As New StreamReader ("MonFichier.txt")
```

Avec la Classe File :

```
Dim SR As StreamReader=File.OpenText ("MonFichier.txt")
```
Comment lire chaque ligne du fichier et s'arrêter à la fin ?
En effet on ne sait pas habituellement combien le fichier contient de ligne, si le fichier contient 2 lignes il faut en lire 2 et s'arrêter sinon on tente de lire après la fin du fichier, encore cela déclenche une erreur.

3 solutions :
1-Utiliser ReadToEnd qui lit en bloc jusqu'à la fin.
2-Avant ReadLine Taper un Try, quand l'erreur 'fin de fichier' survient elle est intercepté par Catch qui sort du cycle de lecture et ferme le fichier.
3-Utiliser Peek qui lit dans le fichier un caractère mais sans modifier la position courante de lecture.

La particularité de Peek est de retourner -1 s'il n'y a plus de caractère à lire sans déclencher d'erreur, d'exception.

La troisième solution est la plus générale et la plus élégante :

```
Do Until SR.Peek=-1
        Ligne=SR.ReadLine()
Loop
```

Enfin on ferme :

```
SR.Close()
```

Notion de 'Buffer', utilisation de Flush.
En fait quand on écrit des informations sur le disque, le logiciel travaille sur un buffer ou mémoire tampon qui est en mémoire vive. Si on écrit des lignes dans le fichier, elles sont 'écrites' dans le buffer en mémoire vive. Quand le buffer est plein,(ou que l'on ferme le fichier) l'enregistrement du contenu du buffer est effectué effectivement sur le disque.

Ce procédé est général à l'écriture et à la lecture de fichier mais totalement transparent car le programmeur ne se préoccupe pas des buffers.

Parfois, par contre, même si on a enregistré peu d'information, on veut être sûr qu'elle est sur le disque, il faut donc forcer l'enregistrement sur disque même si le buffer n'est pas plein, on utilise la méthode Flush.

```
SW.Flush()
```

Le fait de fermer un fichier par Close, appelle automatiquement Flush() ce qui enregistre des données du buffer.

Utiliser « FileOpen »

Visual Basic fournit trois types d'accès au fichier :

- l'accès séquentiel, pour lire et écrire des fichiers texte de manière continue, chaque donnée est enregistrée successivement du début à la fin ; les enregistrements n'ont pas la même longueur, ils sont séparés par des virgules ou des retours à la ligne.

On ne peut qu'écrire le premier enregistrement puis le second, le troisième, le quatrième...
Pour lire c'est pareil : on ouvre, on lit le premier, le second, le troisième, le quatrième....
Pour lire le troisième enregistrement, il faut lire avant les 2 premiers.

- l'accès aléatoire (Random), (on le nomme parfois accès direct) pour lire et écrire des fichiers texte ou binaire constitués d'enregistrements de longueur fixe, on peut

avoir directement accès à un enregistrement à partir de son numéro.

Philippe 1 place de la gare
Jean 35 rue du cloître
Pierre 14 impasse du musée
Louis sdf

Les enregistrements ont une longueur fixe, il faut prévoir!! Si on décide de 20 caractères pour le prénom, on ne pourra pas en mettre 21, le 21ème sera tronqué, à l'inverse l'enregistrement de 15 caractères sera complété par des blancs.

Il n'y a pas de séparateur entre les enregistrements.

Les enregistrements peuvent être constitués d'un ensemble de variables : une structure, ici prénom et adresse.

Ensuite on peut lire directement le second enregistrement, ou écrire sur le 3ème.

- l'accès binaire, pour lire ire et écrire dans tous les fichiers, on lit ou écrit un nombre d'octet désiré…

En pratique :

Les fichiers séquentiels sont bien pratiques pour charger une série de de ligne, (toujours la même) dans une ListBox par exemple.

Faut-il utiliser les fichiers séquentiels ou random (à accès aléatoire) pour créer par exemple un petit carnet d'adresse ?

Il y a 2 manières de faire :
- Créer un fichier random et lire ou écrire dans un enregistrement pour lire ou modifier une adresse.
- Créer un fichier séquentiel. A l'ouverture du logiciel lire séquentiellement toutes les adresses et les mettre dans un tableau (de structure). Pour lire ou modifier une adresse: lire ou modifier un élément du tableau. En sortant du programme enregistrer tous les éléments du tableau séquentiellement.(Enregistrer dans un nouveau fichier, effacer l'ancien, renommer le nouveau avec le nom de l'ancien).
- Bien sur s'il y a de nombreux éléments dans une adresse, un grand nombre d'adresse, il faut utiliser une base de données.

Si on ouvre un fichier en écriture et qu'il n'existe pas sur le disque, il est crée.
Si on ouvre un fichier en lecture et qu'il n'existe pas, une exception est déclenchée (une erreur). On utilisait cela pour voir si un fichier existait: on l'ouvrait, s'il n'y avait pas d'erreur c'est qu'il existait. Mais maintenant il y a plus simple pour voir si un fichier existe.
Si on ouvre un fichier et que que celui-ci est déjà ouvert par un autre autre programme, il se déclenche généralement une erreur (sauf si on l'ouvre en Binaire, c'était le cas en VB6, c'est à vérifier en VB.NET).

Pour ouvrir un fichier on utilise FileOpen
FileOpen (FileNumber, FileName, Mode, Access, Share, RecordLength)

Paramètres de FileOpen :
FileNumber

A tous fichier est affecté un numéro unique, c'est ce numéro que l'on utilisera pour indiquer sur quel fichier pratiquer une opération... Utilisez la fonction FreeFile pour obtenir le prochain numéro de fichier disponible.

FileName
Obligatoire. Expression de type String spécifiant un nom de fichier. Peut comprendre un nom de répertoire ou de dossier, et un nom de lecteur.

Mode
Obligatoire. Énumération OpenMode spécifiant le mode d'accès au fichier : Append, Binary, Input (séquentiel en lecture), Output (séquentiel en écriture) ou Random (accès aléatoire).

Access
Facultatif. Mot clé spécifiant les opérations autorisées sur le fichier ouvert : Read, Write ou ReadWrite. Par défaut, la valeur est OpenAccess.ReadWrite.

Share
Facultatif. Spécifiant si un autre programme peut avoir en même temps accès au même fichier : Shared (permet l'accès aux autres programmes), Lock Read (interdit l'accès en lecture), Lock Write (interdit l'accès en écriture) et Lock Read Write (interdit totalement l'accès). Le processus OpenShare.Lock Read Write est paramétré par défaut.

RecordLength
Facultatif. Nombre inférieur ou égal à 32 767 (octets). Pour les fichiers ouverts en mode Random, cette valeur représente la longueur de l'enregistrement. Pour les fichiers séquentiels, elle représente le nombre de caractères contenus dans la mémoire tampon.

Pour écrire dans un fichier on utilise :
Print, Write, WriteLine, dans les fichiers séquentiels
FilePut dans les fichiers aléatoires
Pour lire dans un fichier on utilise:
Input, LineInput dans les fichiers séquentiels
FileGet dans les fichiers aléatoires.
Pour fermer le fichier on utilise FileClose()

Numéro de fichier :
Pour repérer chaque fichier, on lui donne un numéro unique (de type Integer).
La fonction FreeFile retourne le premier numéro libre.

```
Dim No as integer
No= Freefile()
```

Ensuite on peut utiliser No

```
FileOpen( No, "MonFichier", OpenMode.Output)
Print(No,"toto")
FileClose (No)
```

1-Fichier séquentiel :
Vous devez spécifier si vous voulez lire (entrer) des caractères issus du fichier (mode Input), écrire (sortir) des caractères vers le fichier (mode Output) ou ajouter des caractères au fichier (mode Append).

Ouvrir le fichier 'MonFichier' en mode séquentiel pour y écrire :

```
Dim No as integer
No= Freefile
FileOpen( No, "MonFichier", OpenMode.Output)
```

Pour écrire dans le fichier séquentiel: on utilise Write ou WriteLine Print ou PrintLine:

- La fonction Print écrit dans le fichier sans aucun caractère de séparation.
  Print(1,"toto")
  Print(1,"tata")
  Print(1, 1.2)

Donne le fichier 'tototata1.2'

- La fonction Write insère des virgules entre les éléments et des guillemets de part et d'autre des chaînes au moment de leur écriture dans le fichier, les valeurs booléens et les variables DateTime sont écrites sans problèmes.
  Write(1,"toto")
  Write(1,"tata")
  Write(1, 1.2)

Donne le fichier '"toto";"tata";1.2"

Attention s'il y a des virgules dans les chaînes, elles seront considérées comme séparateurs!! Ce qui entraîne des erreurs à la lecture; il faut mettre la chaîne entre "" ou bien si c'est un séparateur décimal, le remplacer par un point. On peut aussi remplacer la virgule par un caractère non utilisé (# par exemple) avant de l'enregistrer puis après la lecture remplacer '#' par ','

Il faut utiliser Input pour relire ces données (Input utilise aussi la virgule comme séparateur.

- La fonction WriteLine insère un caractère de passage à la ligne, c'est-à-dire un retour chariot+ saut de ligne (Chr(13) + Chr(10)),On lira les données par LineInput.
  WriteLine(1,"toto")
  WriteLine(1,"tata")
  WriteLine(1, 1.2)

Donne le fichier
"toto"
"tata"
1.2

Il faut utiliser LineInput pour relire ces données car il lit jusqu'au retour Chariot, saut de ligne.
Toutes les données écrites dans le fichier à l'aide de la fonction Print respectent les conventions internationales, autrement dit les données sont mises en forme à l'aide du séparateur décimal approprié. Si l'utilisateur souhaite produire des données en vue d'une utilisation par plusieurs paramètres régionaux, il convient d'utiliser la fonction Write

EOF (NuméroFichier) veut dire 'End Of File', (Fin de Fichier) il prend la valeur True si on est à la fin du fichier et qu'il n'y a plus rien à lire.
LOF (NuméroFichier) veut dire 'Lenght Of File', il retourne la longueur du fichier.

Exemple, lire chaque ligne d'un fichier texte :

```
Dim Line As String
FileOpen(1, "MonFichier.txt", OpenMode.Input) ' Ouvre en lecture.
While Not EOF(1) ' Boucler jusqu'à la fin du fichier
Line = LineInput(1) ' Lire chaque ligne
Debug.WriteLine(Line) ' Afficher chaque ligne sur la console.

End While
FileClose(1) ' Fermer.
```

Ici on a utilisé une boucle While… End While qui tourne tant que EOF est Faux. Quand on est à la fin du fichier EOF (End of File) devient égal à True et on sort de la boucle.

2-Fichier à accès aléatoire

On ouvre le fichier avec FileOpen et le mode OpenMode.Random, ensuite on peut écrire un enregistrement grâce à FilePut() ou en lire un grâce à FileGet(). On peut se positionner sur un enregistrement précis (le 2eme, le 15ème) avec Seek.

Le premier enregistrement est l'enregistrement numéro 1
Exemple:
Fichier des adresses
Créer une structure Adresse, on utilise <VBFixedString( )> pour fixer la longueur.

```
Public Structure Adresse
        <VBFixedString(20)>Dim Nom As String
        <VBFixedString(20)>Dim Rue As String
        <VBFixedString(20)>Dim Ville As String
End Structure

'Ouvrir le fichier, comme il n'existe pas, cela entraîne sa création
Dim FileNum As Integer, RecLength As Long, UneAdresse As Adresse
' Calcul de la longueur de l'enregistrement
RecLength = Len(UneAdresse)
' Récupérer le premier numéro de fichier libre.
FileNum = FreeFile
' Ouvrir le fichier.
FileOpen(FileNum, "MONFICHIER.DAT", OpenMode.Random, , , RecLength)
```

Pour écrire des données sur le second enregistrement par exemple :

```
UneAdresse.Nom = "Philippe"
UneAdresse.Rue = "Grande rue"
UneAdresse.Ville = "Lyon"
FilePut(FileNum, UneAdresse,2 )
```

Dans cette ligne de code, FileNum contient le numéro utilisé par la fonction FileOpen pour ouvrir le fichier, 2 est le numéro de l'enregistrement ou sera copié la variable 'UneAdresse' (c'est un long si on utilise une variable) et UneAdresse, déclaré en tant que type Adresse défini par l'utilisateur, reçoit le contenu de l'enregistrement. Cela écrase l'enregistrement 2 s'il contenait quelque chose.

Pour écrire à la fin du fichier, ajouter un enregistrement il faut connaître le nombre d'enregistrement et écrire l'enregistrement suivant.

```
Dim last as long 'noter que le numéro d'enregistrement est un long
```

Pour connaître le nombre d'enregistrement, il faut diviser la longueur du fichier par la longueur d'un enregistrement.

```
last = FileLen("MONFICHIER.DAT") / RecLength
```

On ajoute 1 pour créer un nouvel enregistrement.

```
FilePut(FileNum, UneAdresse,last+1 )
```

Pour lire un enregistrement (le premier par exemple) :

```
FileGet(FileNum, UneAdresse, 1)
```

Attention Option Strict doit être à false.
Si option Strict est à True, la ligne qui précède génère une erreur car le second argument attendu ne peut pas être une variable 'structure'. Pour que le second argument de FileGet (Une adresse) soit converti dans une variable Structure automatiquement Option Strict doit donc être à false. (Il doit bien y avoir un moyen de travailler avec Option Strict On et de convertir explicitement mais je ne l'ai pas trouvé)
Remarque : si le fichier contient 4 enregistrements, on peut écrire le 10ème enregistrement, VB ajoute entre le 4ème et le 10ème, 5 enregistrements vides. On peut lire un enregistrement qui n'existe pas, cela ne déclenche pas d'erreur.
Le numéro d'enregistrement peut être omis dans ce cas c'est l'enregistrement courant qui est utilisé.

On positionne l'enregistrement courant avec Seek :

Exemple, lire le 8ème enregistrement :

```
Seek(FileNum,8)
FileGet(FileNum,Une Adresse)
```

Suppression d'enregistrements
Vous pouvez supprimer le contenu d'un enregistrement en effaçant ses champs (enregistrer à la même position des variables vides), mais l'enregistrement existe toujours dans le fichier.

Pour enlever un enregistrement supprimé

1. Créez un nouveau fichier.
2. Copiez tous les enregistrements valides du fichier d'origine dans le nouveau fichier (pas ceux qui sont vide).
3. Fermez le fichier d'origine et utilisez la fonction Kill pour le supprimer.
4. Utilisez la fonction Rename pour renommer le nouveau fichier en lui attribuant le nom du fichier d'origine.

3-Fichier binaire
Dans les fichiers binaires on travaille sur les octets.
La syntaxe est la même que pour les fichiers Random, sauf qu'on travaille sur la position d'un octet et non sur un numéro d'enregistrement.

Pour ouvrir un fichier binaire :

```
FileOpen(FileNumber, FileName, OpenMode.Binary)
```

FileGet et FilePut permettent de lire ou d'écrire des octets.

```
FileOpen(iFr, ReadString, OpenMode.Binary)
MyString = New String(" "c, 15)     'Créer une chaîne de 15 espaces
FileGet(iFr, MyString)          ' Lire 15 caractères dans MyString
FileClose(iFr)
MsgBox(MyString)
```

Le fait de créer une variable de 15 caractères et de l'utiliser dans FileGet permet de lire 15 caractères.

Utilisation du contrôle RichTextBox

On rappelle que du texte présent dans un contrôle RichTextBox peut être enregistré ou lu très simplement avec les méthodes .SaveFile et .LoadFile.

Le texte peut être du texte brut ou du RTF.

```
richTextBox1.SaveFile(FileName, RichTextBoxStreamType.PlainText)
```

Si on remplace.PlainText par .RichText c'est le texte enrichi et non le texte brut qui est enregistré.

Pour lire un fichier il faut employer .LoadFile avec la même syntaxe.
Simple, non!!!

Lire ou écrire des octets ou du XML

BinaryWriter et BinaryReader permettent d'écrire ou de lire des données binaires.
XMLTextWriter et XMLTextReader écrit et lit du Xml.

4.7 Travailler sur les répertoires

Comment créer, copier effacer des répertoires (ou dossiers) ?

Classe DirectoryInfo et la Classe Directory

Pour travailler sur les dossiers (ou répertoires), il faut au préalable taper :

```
Imports System.IO
```

La classe Directory est utilisée pour travailler sur un ensemble de dossier, la Classe directoryInfo donne des renseignements sur un dossier particulier (Après instanciation).

La Classe Directory possède les méthodes suivantes :

- Exists — Teste si le dossier existe.
- CreateDirectory — Crée le dossier
- Delete — Efface le dossier
- Move — Déplacement de dossier
- GetCurrentDirectory — Retourne le dossier de travail de l'application en cours
- SetCurrentDirectory — Définit le dossier de travail de l'application.
- GetDirectoryRoot — Retourne le dossier racine du chemin spécifié.
- GetDirectories — Retourne le tableau des sous dossiers du dossier spécifié.
- GetFiles — Retourne les fichiers du dossier spécifié.
- GetFilesSystemEntries — Retourne fichier et sous dossier avec possibilité d'un filtre.
- GetLogicalDrives — Retourne les disques
- GetParent — Retourne le dossier parent du dossier spécifié.

La Classe Directory est statique, on l'utilise directement.

Exemple:
Afficher dans une listeBox les sous dossiers du répertoire de l'application :

```
Dim SousDos() As String= Directory.GetDirectories(Directory.GetCurrentDirectory)
Dim Dossier As String
For Each Dossier In SousDos
    List1.Items.Add(Dossier)
Next
```

La Classe DirectoryInfo possède les propriétés suivantes :

Name	Nom du dossier (sans extension)
Full Name	Chemin et nom du dossier
Exists	
Parents	Dossier parent
Root	Racine du dossier

La Classe DirectoryInfo n'est pas statique, il faut instancer un dossier avant de l'utiliser.

Il y a aussi les méthodes suivantes :

Create, Delete, MoveTo	
CreateSubdirectory	
GetDirectories	Retourne les sous dossier
GetFiles	Retourne des fichiers
GetFileSystemInfos	

Exemple :
Afficher le répertoire parent d'un dossier :

```
Dim D As DirectoryInfo
D= New DirectoryInfo( MonDossier)
MsgBox(D.Parent.ToString)
```

Classe Path

La Classe statique Path a des méthodes simplifiant la manipulation des répertoires.

Exemple :
Si C= "C:\Windows\MonFichier.txt"
Path.GetDirectoryName(C) retourne "C:\Windows
Path.GetFileName(C) retourne "Monfichier.txt"
Path.GetExtension(C) retourne ".txt"
Path.GetFileNameWithoutExtension(C) retourne "MonFichier"
Path.PathRoot(C) retourne "c:\"

Il y a aussi les méthodes ChangeExtension, Combine, HasExtension...

Classe Environement

Fournit des informations concernant l'environnement et la plate-forme en cours ainsi que des moyens pour les manipuler.

Par exemple, les arguments de la ligne de commande, le code de sortie, les paramètres des variables d'environnement, le contenu de la pile des appels, le temps écoulé depuis le dernier démarrage du système ou le numéro de version du Common Language Runtime mais aussi certains répertoires.

```
Environment.CurrentDirectory      'donne le répertoire courant : ou le processus en cours démarre.
Environment.MachineName           'Obtient le nom NetBIOS de l'ordinateur local.
Environment.OsVersion             'Obtient un objet contenant l'identificateur et le numéro de version de la plate-forme en cours.
Environment.SystemDirectory       'Obtient le chemin qualifié complet du répertoire du système
Environment.UserName              'Obtient le nom d'utilisateur de la personne qui a lancé le thread en cours.
```

La fonction GetFolderPath avec un argument faisant partie de l'énumération SpecialFolder retourne le répertoire d'un tas de choses :

Exemples :
Quel est le répertoire Système ?

```
Environment.GetFolderPath(Environment.SpecialFolder.System)
```

Comment récupérer le nom des disques ?

```
Dim drives As String() = Environment.GetLogicalDrives()
```

Comment récupérer la ligne de commande ?

```
Dim arguments As String() = Environment.GetCommandLineArgs()
```

On peut aussi utiliser les anciennes méthodes VB

CurDir()
Retourne le chemin d'accès en cours.

```
MyPath = CurDir()
MyPath = CurDir("C"c)
```

Dir()
Retourne une chaîne représentant le nom d'un fichier, d'un répertoire ou d'un dossier qui correspond à un modèle ou un attribut de fichier spécifié ou à l'étiquette de volume d'un

lecteur.
'Vérifier si un fichier existe :
'Retourne "WIN.INI" si il existe.
```
MyFile = Dir("C:\WINDOWS\WIN.INI")
```

'Retourne le fichier spécifié par l'extension.
```
MyFile = Dir("C:\WINDOWS\*.INI")
```

'Un nouveau Dir retourne le fichier suivant
```
MyFile = Dir()
```

'On peut surcharger avec un attribut qui sert de filtre.
MyFile = Dir("*.TXT", vbHidden) ' affiche les fichiers cachés

'Recherche les sous répertoires.
```
MyPath = "c:\" ' Set the path.
MyName = Dir(MyPath, vbDirectory)
```

ChDrive
Change le lecteur actif. La fonction lève une exception si le lecteur n'existe pas.
```
ChDrive("D")
```

MkDir
Créé un répertoire ou un dossier. Si aucun lecteur n'est spécifié, le nouveau répertoire ou dossier est créé sur le lecteur actif.
```
MkDir("C:\MYDIR")
```

RmDir
Enleve un répertoire ou un dossier existant.
'Vérifier que le répertoire est vide sinon effacez les fichiers avec Kill.
```
RmDir ("MYDIR")
```

ChDir
Change le répertoire par défaut mais pas le lecteur par défaut.
```
ChDir("D:\TMP")
```

L'exécution de changements relatifs de répertoire s'effectue à l'aide de "..", comme suit :
```
ChDir("..")     'Remonte au répertoire parent.
```

FileCopy
Copier un fichier.
```
FileCopy(SourceFile, DestinationFile)
```

Rename
Renommer un fichier, un répertoire ou un dossier.
```
Rename (OldName, NewName)
```

FileLen
Donne la longueur du fichier, SetAttr et GetAttr modifie ou lit les attributs du fichier
Result = GetAttr(FName)

Result est une combinaison des attributs. Pour déterminer les attributs définis, utilisez l'opérateur And pour effectuer une comparaison d'opérations de bits entre la valeur retournée par la fonction GetAttr et la valeur de l'attribut. Si le résultat est différent de zéro, cet attribut est défini pour le fichier désigné.

Par exemple, la valeur de retour de l'expression And suivante est zéro si l'attribut Archive n'est pas défini :
Result = GetAttr(FName) And vbArchive

4.8 Afficher correctement du texte

Comment afficher du texte, du numérique suivant le format désiré ?

On a vu que pour afficher du texte il fallait l'affecter à la propriété 'text' d'un label ou d'un textBox (ou pour des tests l'afficher sur la fenêtre 'console').

Pas de problème pour afficher des chaînes de caractères, par contre, pour les valeurs numériques, il faut d'abord les transformer en String' et les formater (définir les séparateurs, le nombre de décimales...).

ToString

On a déjà vu que pour afficher une variable numérique, il fallait la transformer en string de la manière suivant :

```
MyDouble.ToString
```

Mais ToString peut être surchargé par un paramètre appelé chaîne de format. Cette chaîne de format peut être standard ou personnalisée.

- Chaîne de format standard :

Cette chaîne est de la forme 'Axx' ou A donne le type de format et xx le nombre de chiffre après la virgule.

```
Imports System
Imports System.Globalization
Imports System.Threading

Module Module1
Sub Main()

Thread.CurrentThread.CurrentCulture = New CultureInfo("en-us")
Dim UnDouble As Double = 123456789

Console.WriteLine("Cet exemple est en-US culture:")
Console.WriteLine(UnDouble.ToString("C"))        'format monétaire (C) affiche
$123,456,789.00

Console.WriteLine(UnDouble.ToString("E"))        'format scientifique (E) affiche
1.234568E+008

Console.WriteLine(UnDouble.ToString("P"))        'format % (P) affiche
12,345,678,900.00%

Console.WriteLine(UnDouble.ToString("N"))        'format nombre (N) affiche
123,456,789.00

Console.WriteLine(UnDouble.ToString("F"))        'format virgule fixe (F) affiche
123456789.00

End Sub
End Module
```

Autre exemple :

```
S=(1.2).ToString("C") 'retourne en CurrentCulture Français 1,2€
```

Il existe aussi D pour décimal, G pour général X pour hexadécimal.

- Chaîne de format personnalisé :

On peut créer de toute pièce un format, on utilise pour cela :
0 indique une espace réservé de 0

Chaque '0' est réservé à un chiffre. Affiche un chiffre ou un zéro. Si le nombre contient moins de chiffres que de zéros, affiche des zéros non significatifs.

Si le nombre contient davantage de chiffres à droite du séparateur décimal qu'il n'y a de zéros à droite du séparateur décimal dans l'expression de format, arrondit le nombre à autant de positions décimales qu'il y a de zéros.

Si le nombre contient davantage de chiffres à gauche du séparateur décimal qu'il n'y a de zéros à gauche du séparateur décimal dans l'expression de format, affiche les chiffres supplémentaires sans modification.

indique un espace réservé de chiffre.

Chaque '#' est réservé à un chiffre. Affiche un chiffre ou rien. Affiche un chiffre si l'expression a un chiffre dans la position où le caractère # apparaît dans la chaîne de format, sinon, n'affiche rien dans cette position.

Ce symbole fonctionne comme l'espace réservé au 0, sauf que les zéros non significatifs et à droite ne s'affichent pas si le nombre contient moins de chiffres qu'il n'y a de caractères # de part et d'autre du séparateur décimal dans l'expression de format.

. (point) indique l'emplacement du séparateur décimal (celui affiché sera celui du pays)
Vous devriez donc utiliser le point comme espace réservé à la décimale, même si vos paramètres régionaux utilisent la virgule à cette fin. La chaîne mise en forme apparaîtra dans le format correct pour les paramètres régionaux.

, (virgule) indique l'emplacement du séparateur de millier.
Séparateur de milliers. Il sépare les milliers des centaines dans un nombre de quatre chiffres ou plus à gauche du séparateur décimal.

"Littéral" la chaîne sera affichée telle quelle.

% affichera en pour cent.
Multiplie l'expression par 100. Le caractère du pourcentage (%) est inséré

E0 affiche en notation scientifique.

: et / sont séparateur d'heure et de date.

; est le séparateur de section : on peut donner 3 formats (un pour les positifs, un pour les négatifs, un pour zéro) séparés par ;

Exemples :
Chaîne de format '0000', le chiffre 145 cela affiche '0145'
Chaîne de format '####', le chiffre 145 cela affiche '145'
Chaîne de format '000.00', le chiffre 45.2 cela affiche '045.20'
Chaîne de format '#,#', le chiffre 12345678 cela affiche '12,345,678'
Chaîne de format '#,,' le chiffre 12345678 cela affiche '12'

La chaîne de formatage' #,##0.00 ' veut dire obligatoirement 2 chiffres après le séparateur décimal et un avant :

Si on affiche avec ce format
1.1 cela donne 1,10

.5 cela donne 0,50
4563 cela donne 4 563,00

Exemples :

```
Dim N As Double = 19.95

Dim MyString As String = N.ToString("$#,##0.00;($#,##0.00);Zero")
```

' En page U.S. English culture, MyString aura la valeur: $19.95.
' En page Française, MyString aura la valeur: 19,95€.

Exemples :

```
Dim UnEntier As Integer = 42
MyString = UnEntier.ToString( "Mon nombre " + ControlChars.Lf + "= #" )
```

Affiche :
Mon nombre
= 42

Str() est toujours accepté

Il permet de transformer une variable numérique et String, qui peut ensuite être affichée.

```
MyString=Str(LeNombre)
Label1.Text=MyString
```

Pas de formattage…

String.Format

Permet de combiner des informations littérales à afficher sans modification et des zones formatées.

Les arguments de String.Format se décomposent en 2 parties séparées d'une virgule.

- Chaîne de formatage entre guillemets : Exemple "{0} + {1} = {2}": les numéros indique l'ordre des valeurs.
- Valeurs à afficher dans l'ordre, la première étant d'indice zéro. Exemple= A, B, A+B

Exemple :
Si A=3 et B=5

```
MsgBox(String.Format("{0} + {1} = {2}",A, B, A+B))
```

affiche 3+5=8

Autre exemple :

```
Dim MonNom As String = "Phil"
String.Format("Nom = {0}, heure = {hh}", MonNom, DateTime.Now)
```

Le texte fixe est « Nom = » et « ", heure = », les éléments de format sont « {0} » et « {hh} » et la liste de valeurs est MonNom et DateTime.Now.

Cela affiche : Nom = Phil Heure= 10

Là aussi on peut utiliser les formats :

- Prédéfinis: Ils utilisent là aussi les paramètres régionaux. Ils utilisent C, D, E, F, G, N, P, R, X comme ToSting.

```
MsgBox(String.Format("{0:C}",-456.45))        'Affiche -456,45€
MsgBox(String.Format("{0:D8}", 456))          'Affiche 00000456 Décimal 8 chiffres
MsgBox(String.Format("{0:P}", 0.14))          'Affiche 14% Pourcent
MsgBox(String.Format("{0:X}", 65535))         'Affiche FFFF Hexadécimal
```

- Personnalisé: avec des # et des 0
```
MsgBox(String.Format("{0:##,##0.00}", 6553.23))
```

La fonction Format (et pas la classe String.Format) fourni des fonctions similaires mais les arguments sont dans l'ordre inverse (valeur, chaîne de formatage) et il n'y a pas de numéro d'ordre et de{}!! C'est pratique pour afficher une seule valeur.

```
MyStr = Format(5459.4, "##,##0.00") ' Returns "5,459.40".
MyStr = Format(334.9, "###0.00") ' Returns "334.90".
MyStr = Format(5, "0.00%") ' Returns "500.00%".
```

CultureInfo

On se rend compte que l'affichage est dépendant de la CurrentCulture du Thread en cours.

Exemple :
Si la CurrentCulture est la CultureInfo Us et que j'affiche avec le format 'C' (monétaire) cela affiche un $ avant, si je suis en CurrentCulture Français cela affiche un € après.

Par défaut la CultureInfo est celle définie dans Windows.

On peut modifier le CurrentCulture par code (voir exemple plus haut).

En français par défaut :

Le séparateur de décimal numérique est le .
Exemple : 1.20

Le séparateur décimal monétaire est la ,
Exemple : 1,20€

4.9 Le curseur

Comment modifier l'apparence du curseur ?

Un curseur est une petite image dont l'emplacement à l'écran est contrôlé par la souris, un stylet ou un trackball. Quand l'utilisateur déplace la souris, le système d'exploitation déplace le curseur.

Différentes formes de curseur sont utilisées pour informer l'utilisateur de l'action que va avoir la souris.

Apparence du curseur

Pour modifier l'aspect du curseur il faut modifier l'objet Cursor.Current; l'énumération Cursors contient les différents curseurs disponibles :

```
System.Windows.Forms.Cursor.Current = System.Windows.Forms.Cursors.WaitCursor
```

Ou plus simplement pour afficher le sablier :

```
Cursor.Current = Cursors.WaitCursor
```

Pour revenir au curseur normal :

```
Cursor.Current = Cursors.Défault
```

Comme d'habitude il suffit de taper « Cursors. » pour voir la liste des curseurs.

Le curseur peut disparaître et être de nouveau affiché par Hide et Show.

Curseur sur un contrôle

Un contrôle dans une fenêtre possède une propriété Cursor; en mode design, si je donne une valeur autre que celle par défaut, CursorWait par exemple, cela modifie le curseur quand la souris passe au dessus de l'objet (met un sablier dans notre exemple).

4.10 Lancer une application, une page Web

Comment lancer une autre application ?

L'ancienne méthode toujours valable : Shell

Shell lance un programme exécutable.

```
Id=Shell (NomdeProgramme)        'lance l'application NomdeProgramme
```

on peut aussi utiliser :

```
Id=Shell(NomdeProgramme, TypedeFenetre, Wait, TimeOut)
```

TypedeFenêtre utilise l'énumération AppWinStyle pour définir le type de fenêtre de l'application lancé, AppWinStyle.MaximizedFocus ouvre par exemple l'application en plein écran.

Si vous souhaitez attendre la fin du programme avant de continuer, vous devez définir Wait à True.

TimeOut est le nombre de millisecondes à attendre pour la fin du programme si Wait est True.

Exemple :

```
ID = Shell("""C:\Program Files\MonFichier.exe"" -a -q", , True, 100000)
```

Dans une chaîne une paire de guillemets doubles adjacents ("") est interprétée comme un caractère de guillemet double dans la chaîne. Ainsi, l'exemple précédent présente la chaîne suivante à la fonction Shell :

```
"C:\Program Files\MonFichier.exe" -a -q
```

La fonction AppActivate rend active l'application ou la fenêtre définie par son nom ou son Id.

```
Dim ID As Integer
```

On peut utiliser :

```
AppActivate("Untitled - Notepad")
```

ou

```
ID = Shell(NOTEPAD.EXE", AppWinStyle.MinimizedNoFocus)
AppActivate(ID)
```

Avec la Classe Process

La Classe Process fournit l'accès à des processus locaux ainsi que distants, et vous permet de démarrer et d'arrêter des processus système locaux.

Classe de nom à importer : Imports System.Diagnostics

On peut maintenant instancer un Process.

```
Dim monProcess As New Process()
```

Ensuite il faut fournir à la classe fille StartInfo les informations nécessaires au démarrage.

```
monProcess.StartInfo.FileName = "MyFile.doc"
monProcess.StartInfo.Verb = "Print"
monProcess.StartInfo.CreateNoWindow = True
```

Enfin on lance le process :

```
monProcess.Start()
```

Noter la puissance de cette classe : on donne le nom du document et VB lance l'exécutable correspondant, on fait effectuer certaines action au programme.

Dans l'exemple du dessus on ouvre Word on y charge MyFile, on l'imprime, cela sans ouvrir de fenêtre.

On peut aussi utiliser la classe Process en statique (sans instanciation)

```
Process.Start("IExplore.exe")
Process.Start(MonPathFavori)
```

Ou en une ligne :

```
Process.Start("IExplore.exe", "www.microsoft.com")
```

En local on peut afficher un fichier html ou asp :

```
Process.Start("IExplore.exe", "C:\monPath\Fichier.htm")
Process.Start("IExplore.exe", "C:\monPath\Fichier.asp")
```

On peut enfin utiliser un objet StartInfo :

```
Dim startInfo As New ProcessStartInfo("IExplore.exe")
startInfo.WindowStyle = ProcessWindowStyle.Minimized
Process.Start(startInfo)
startInfo.Arguments = www.chez.com
Process.Start(startInfo)
```

Des propriétés du processus en cours permettent de connaître l'Id du processus (Id) les threads, les modules, les Dll, la mémoire, de connaître le texte de la barre de titre (MainWindowsTitle)…

On peut fermer le processus par Close ou CloseMainWindows

On peut instancer un processus sur une application déjà en cours avec GetProcessByName et GetProcessById :

```
Dim P As Process() = Process.GetProcessesByName("notepad")
```

On peut récupérer le processus courant :

```
Dim ProcessusCourant As Process = Process.GetCurrentProcess()
```

Récupérer toutes les instances de Notepad qui tourne en local :

```
Dim localByName As Process() = Process.GetProcessesByName("notepad")
```

Récupérer tous les processus en cours d'exécution grâce à GetProcesses :

```
Dim localAll As Process() = Process.GetProcesses()
```

Processus sur ordinateur distant.
Vous pouvez afficher des données statistiques et des informations sur les processus en cours d'exécution sur des ordinateurs distants, mais vous ne pouvez pas appeler Kill, Start, CloseMainWindows sur ceux-ci.

4.11Imprimer

Comment Imprimer ?

Prévoir une longue soirée, au calme, un bon siège, 1 g de paracétamol et un gros thermo de café !!!

On devra que l'on peut utiliser pour imprimer :
Soit un composant 'PrintDocument'.
Soit une instance de 'la Class PrintDocument'.

A-Imprimer 'Hello' avec le composant 'PrintDocument'.

L'utilisateur clique sur un bouton, cela imprime 'Hello'

Cet exemple utilise un 'composant PrintDocument'

Comment faire en théorie?
C'est le composant PrintDocument qui imprime.

En prendre un dans la boite à outils, le mettre dans un formulaire. Il apparaît sous le formulaire et se nomme PrintDocument1.

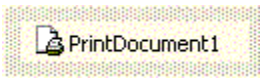
Pour imprimer il faut utiliser la méthode Print de ce composant PrintDocument, Il faut donc écrire l'instruction suivante :

```
PrintDocument1.Print
```

Cette instruction appelle la procédure évènement PrintDocument1_PrintPage du composant PrintDocument et qui contient la logique d'impression. Un paramètre de cet évènement PrintPage est l'objet graphique envoyé à l'imprimante. C'est à vous de dessiner dans l'objet graphique ce que vous voulez imprimer. En fin de routine, l'objet graphique sera imprimé (automatiquement).

En pratique :

- Je prends un PrintDocument dans la boite à outils, je le mets dans un formulaire. Il apparaît sous le formulaire et se nomme PrintDocument1.



- Si je double-clique sur PrintDocument1 je vois apparaître la procédure PrintDocument1_PrintPage (qui a été générée automatiquement) :

```
Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object, ByVal e As
System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage
End Sub
```

C'est cette procédure qui est fondamentale et qui contient les routines d'impression écrites par le programmeur. Les routines d'impression agissent sur l'objet graphique qui sera utilisé pour imprimer, cet objet graphique est fournit dans les paramètres de la procédure(ici c'est e qui est de type PrintPageEventArgs)

- Dans cette routine PrintPage, j'ajoute donc le code dessinant une texte (DrawString) sur l'objet graphique 'e':

```
e.Graphics.DrawString("Hello", New Font("Arial", 80, FontStyle.Bold), Brushes.Black, 150,
125)
```

- Enfin je dessine un bouton nommé 'ButtonPrint' avec une propriété Text contenant "Imprimer Hello" et dans la procédure ButtonPrint_Click j'appelle la méthode Print

```
PrintDocument1.Print()
```

Voici le code complet:

```
Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object, ByVal e As
System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage
e.Graphics.DrawString("Hello", New Font("Arial", 80, FontStyle.Bold), Brushes.Black,
150, 125)
End Sub
Private Sub ButtonPrint_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ButtonPrint.Click
PrintDocument1.Print()
End Sub
```

Si je clique sur le bouton 'Imprimer' cela affiche un gros 'Hello'.

La méthode Print d'un PrintDocument déclenche l'événement PrintPage de ce PrintDocument qui contient le code dessinant sur le graphique de la page à imprimer. En fin de routine PrintPage le graphique est imprimer sur la feuille de l'imprimante.

Toutes les méthodes graphiques permettant d'écrire, de dessiner, de tracer des lignes... sur un graphique permettent donc d'imprimer.

Imprimer du graphisme

Créons une ellipse bleue à l'intérieur d'un rectangle avec la position et les dimensions suivantes : début à 100, 150 avec une largeur de 250 et une hauteur de 250.

```
Private Sub PrintDocument1_PrintPage(ByVal sender As Object, ByVal e As
System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage
    e.Graphics.FillEllipse(Brushes.Blue, New Rectangle(100, 150, 250, 250))
End Sub
```

Imprimer un Message Box indiquant 'Fin d'impression'.

On a étudié l'évènement PrintPage, mais il existe aussi les évènements :
BeginPrint et EndPrint respectivement déclenchés en début et fin d'impression.

Il suffit d'utiliser l'évènement EndPrint pour prévenir que l'impression est terminée:

```
Private Sub PrintDocument1_EndPrint(ByVal sender As Object, ByVal e As
System.Drawing.Printing.PrintEventArgs) Handles PrintDocument1.EndPrint
    MessageBox.Show("Fin d'impression")
End Sub
```

On peut même fignoler et afficher "Fin d'impression de Nom du document"
Il faut avoir renseigné le DocumentName:

```
PrintDocument1.DocumentName = "MyTextFile"
```

Puis écrire :

```
Private Sub PrintDocument1_EndPrint(ByVal sender As Object, ByVal e As
System.Drawing.Printing.PrintEventArgs) Handles PrintDocument1.EndPrint
    MessageBox.Show( "Fin d'impression de "+PrintDocument1.DocumentName)
End Sub
```

B-Même programme : Imprimer 'Hello' mais avec la Class PrintDocument

L'utilisateur clique sur un bouton, cela imprime 'Hello'

Cet exemple utilise 'une instance de la Class PrintDocument'. On ne met pas de composant 'PritDocument' dans le formulaire.

Comment faire en théorie?
Il faut importer l'espace de nom 'Printing' par :

```
Imports System.Drawing.Printing
```

Il faut créez une instance de la Class PrintDocument dans le module.

```
Dim pd as PrintDocument = new PrintDocument()
```

Il faut créer une routine pd_PrintPage.

```
Private Sub pd_PrintPage(sender As object, ev As
System.Drawing.Printing.PrintPageEventArgs)
End sub
```

Il faut indiquer le "lien" entre l'objet pd et la routine évènement PrintPage

```
AddHandler pd.PrintPage, AddressOf Me.pd_PrintPage
```

Dans la procédure Button_Click d'un bouton "Imprimer" il faut appeler la méthode Print du PrintDocument pour effectuer l'impression du document.

```
pd.Print
```

Cela déclenche la procédure Private Sub pd_PrintPage précédemment écrite, dans laquelle on a ajouté :

```
ev.Graphics.DrawString ("Hello", printFont, Brushes.Black, leftMargin, yPos, new
StringFormat()).
```

Cela donne le code complet:

```
Imports System.Drawing.Printing

Public Class Form1
Inherits System.Windows.Forms.Form

Dim pd As PrintDocument = New PrintDocument 'Assumes the default printer
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    AddHandler pd.PrintPage, AddressOf Me.Pd_PrintPage
End Sub
Private Sub Pd_PrintPage(ByVal sender As System.Object, ByVal e As
System.Drawing.Printing.PrintPageEventArgs)
e.Graphics.DrawString("Hello", New Font("Arial", 80, FontStyle.Bold), Brushes.Black,
150, 125)
End Sub
Private Sub ButtonPrint_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ButtonPrint.Click
    pd.Print()
End Sub

End Class
```

Comment choisir l'imprimante ?

Le composant PrintDialog permet le choix de l'imprimante, de la zone à imprimer (tout, la sélection..) et donne accès aux caractéristiques de l'imprimante.

Comment l'utiliser ?
Il faut créer une instance de PrintDialog:

```
Dim dlg As New PrintDialog
```

Il faut indiquer au PrintDialog sur quel PrintDocument travailler :

```
dlg.Document = pd
```

Puis ouvrir la fenêtre PrintDialog avec la méthode ShowDialog.
L'utilisateur choisit son imprimante puis clique sur 'Ok'.
Si elle retourne Ok, on imprime.

Voici le code complet ou quand l'utilisateur clique sur le bouton ButtonPrint ('Imprimer') la fenêtre PrintDialog s'ouvre :

```
Private Sub ButtonPrint_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ButtonPrint.Click
Dim dlg As New PrintDialog
dlg.Document = pd
Dim result As DialogResult = dlg.ShowDialog()
If (result = System.Windows.Forms.DialogResult.OK) Then
        pd.Print()
End If

End Sub
```

Comment modifier la page à imprimer ?

Comment choisir d'imprimer en portrait ou paysage ? Modifier les marges...

Il faut utiliser un composant PageSetUpDialog.
Pour stocker les informations sur la page (marges...) il faut un PageSetting

Je lie le PageSetting au PageSetUpDialog en donnant à la propriété PageSettings du PageSetUpDialog le nom du PageSetting.
Puis j'ouvre le PageSetUpDialog.

Au retour le PageSetting contient les modifications, je les 'passe' au PrintDocument avant d'imprimer.

Cela donne :

```
Dim psDlg As New PageSetupDialog
Dim LePageSettings As New PageSettings
psDlg. PageSettings = LePageSettings
psDlg.ShowDialog()
pd.DefaultPageSettings = LePageSettings
```

Prévisualisation de la page à imprimer

On utilise pour cela un PrintPreviewDialog, on lui indique quel PrintDocument pré visualiser en l'assignant à sa méthode document puis on l'affiche par ShowDialog().

```
Dim dllg As New PrintPreviewDialog
dllg.Document = pd
dllg.ShowDialog()
```

Construction d'une application d'impression complexe

Comment imprimer le contenu d'un fichier texte vers une imprimante ?

Tous les didacticiels (Microsoft compris) donnent cet exemple.
La première chose que vous devez faire est d'écrire votre logique d'impression. Pour cela, quand la méthode PrintDocument.Print() est appelée, les événements suivants sont déclenchés.

- BeginPrint

- PagePrint (un ou plusieurs s'il y a plusieurs pages à imprimer)
- EndPrint

Le type d'arguments d'événement de PagePrint (PagePrintEventArgs) comprend une propriété HasMorePages. Si celle-ci a la valeur TRUE lors du retour de votre gestionnaire d'événements, PrintDocument définit une nouvelle page et déclenche de nouveau l'événement PagePrint.

Voyons la logique dans votre gestionnaire d'événements PagePrint :

- Imprimez le contenu de la page en utilisant les informations des arguments d'événement. Les arguments d'événement contiennent l'objet Graphics pour l'imprimante, le PageSettings pour cette page, les limites de la page, et la taille des marges.

Il faut dans PagePrint imprimer ligne par ligne en se déplaçant à chaque fois vers le bas d'une hauteur de ligne.

Pour 'simplifier', on considère que chaque ligne ne déborde pas à droite!!

- Détermine s'il reste des pages à imprimer.
- Si c'est le cas, HasMorePages doit être égal à TRUE.
- S'il n'y a pas d'autres pages, HasMorePages doit être égal à FALSE.

```
Public Class ExampleImpression
        Inherits System.Windows.Forms.Form

        ...

        private printFont As Font
        private streamToPrint As StreamReader

        Public Sub New ()
                MyBase.New
                InitializeComponent()
        End Sub

        'Evénement survenant lorsque l'utilisateur cloque sur le bouton 'imprimer'
        Private Sub printButton_Click(sender As object, e As System.EventArgs)

                Try
                        streamToPrint = new StreamReader ("PrintMe.Txt")
                        Try
                                printFont = new Font("Arial", 10)
                                Dim pd as PrintDocument = new PrintDocument() 'déclaration
                                du PrintDocument
                                AddHandler pd.PrintPage, AddressOf Me.pd_PrintPage
                                pd.Print()
                        Finally
                                streamToPrint.Close()
                        End Try

                Catch ex As Exception
                        MessageBox.Show("Une erreur est survenue: - " + ex.Message)
                End Try

        End Sub

        'Evènement survenant pour chaque page imprimer
        Private Sub pd_PrintPage(sender As object, ev As
        System.Drawing.Printing.PrintPageEventArgs)
```

```
        Dim lpp As Single = 0 'nombre de ligne par page

        Dim yPos As Single = 0 'ordonnée
        Dim count As Integer = 0 'numéro de ligne
        Dim leftMargin As Single = ev.MarginBounds.Left
        Dim topMargin As Single = ev.MarginBounds.Top
        Dim line as String

        'calcul le nombre de ligne par page
        ' hauteur de la page/hauteur de la police de caractère
        lpp = ev.MarginBounds.Height  / printFont.GetHeight(ev.Graphics)

        'lit une ligne dans le fichier
        line=streamToPrint.ReadLine()

        'Boucle affichant chaque ligne
        while (count < lpp AND line <> Nothing)

                yPos = topMargin + (count * printFont.GetHeight(ev.Graphics))

                'Ecrit le texte dans l'objet graphique
                ev.Graphics.DrawString (line, printFont, Brushes.Black, leftMargin, _
                        yPos, new StringFormat())

                count = count + 1

                if (count < lpp) then
                        line=streamToPrint.ReadLine()
                end if

        End While

        'S'il y a encore des lignes, on réimprime une page
        If (line <> Nothing) Then
                ev.HasMorePages = True
                Else
                        ev.HasMorePages = False
        End If

        End Sub

        ....

    End Class
```

On a vu que pour 'simplifier', on considère que chaque ligne ne déborde pas à droite.
Dans la pratique, pour gérer les retours à la ligne on peut dessiner dans un rectangle.
(Voir la page sur les graphiques.)

Propriétés du 'PrintDocument'

On peut sans passer par une 'boite de dialog' gérer directement l'imprimante, les marges, le nombre de copies…

Si pd est le PrintDocument :

- `pd.PrinterSetting` désigne l'imprimante en cours
- `pd.PrinterSetting.PrinterName` retourne ou définit le nom de cette imprimante
- `pd.PrinterSetting.Printerresolution` donne la résolution de cette imprimante.

pd.PrinterSetting.installedPrinted donne toutes les imprimantes installées.
La propriété DefaultPageSetting est en rapport avec les caractéristiques de la page.
pd.PrinterSetting.DefaultPageSetting.Margins donne les marges
pd.PrinterSetting.PrinttoFile permettrait d'imprimer dans un fichier (non testé)

Imprime le formulaire en cours

Exemple fournit par Microsoft :

```
Private Declare Function BitBlt Lib "gdi32.dll" Alias "BitBlt" (ByVal _ hdcDest
        As IntPtr, ByVal nXDest As Integer, ByVal nYDest As _ Integer,
        ByVal nWidth As Integer, ByVal nHeight As Integer, ByVal _ hdcSrc
        As IntPtr, ByVal nXSrc As Integer, ByVal nYSrc As Integer, _ ByVal
        dwRop As System.Int32) As Long
Dim memoryImage As Bitmap
Private Sub CaptureScreen()
        Dim mygraphics As Graphics = Me.CreateGraphics()
        Dim s As Size = Me.Size
        memoryImage = New Bitmap(s.Width, s.Height, mygraphics)
        Dim memoryGraphics As Graphics = Graphics.FromImage(memoryImage)
        Dim dc1 As IntPtr = mygraphics.GetHdc
        Dim dc2 As IntPtr = memoryGraphics.GetHdc
        BitBlt(dc2, 0, 0, Me.ClientRectangle.Width, _
                Me.ClientRectangle.Height, dc1, 0, 0, 13369376)
        mygraphics.ReleaseHdc(dc1)
        memoryGraphics.ReleaseHdc(dc2)
End Sub
Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object, _
        ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles _
        PrintDocument1.PrintPage
        e.Graphics.DrawImage(memoryImage, 0, 0)
End Sub
Private Sub PrintButton_Click(ByVal sender As System.Object, ByVal e As _
        System.EventArgs) Handles PrintButton.Click
        CaptureScreen()
        PrintDocument1.Print()
End Sub
```

Imprime un contrôle DataGrid

Exemple fournit par Microsoft :
Cet exemple nécessite :

- un contrôle Button, nommé ImprimerGrid, dans le formulaire ;
- un contrôle DataGrid nommé DataGrid1 ;
- un composant PrintDocument nommé PrintDocument1.

Comme d'habitude PrintPage imprime e.Graphics.

D'après ce que j'ai compris, l'évènement Paint redessine un contrôle mais on peut choisir le contrôle et l'endroit ou le redessiner,

Je redessine donc grâce à Paint, le DataGrid dans e.graphics.
PaintEventArgs Fournit les données pour l'événement Paint :
PaintEventArgs spécifie l'objet graphics à utiliser pour peindre le contrôle, ainsi que le ClipRectangle dans lequel le peindre.
InvokePaint déclenche l'évènement Paint

```
Private Sub ImprimerGrid_Click(ByVal sender As System.Object, ByVal e As _
        System.EventArgs) Handles PrintGrid.Click
```

```
        PrintDocument1.Print()
End Sub

Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object, _
        ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles _
        PrintDocument1.PrintPage
        Dim myPaintArgs As New PaintEventArgs(e.Graphics, New Rectangle(New _
                Point(0, 0), Me.Size))
        Me.InvokePaint(DataGrid1, myPaintArgs)
End Sub
```

4.12Dessiner

Avec GDI+ utilisé par VB.NET, on utilise des objets :
Graphics qui sont des zones de dessin
Image (BitMap ou MetaFile) contenant une image
Rectangle pour définir une zone
Pen correspondant à un Stylet
Font pour une police de caractères
Brush, c'est une brosse

Sur quoi dessiner ?

Il faut définir une zone de dessin, un objet Graphics. On peut y inclure des objets Image (des BitMap ou des MetaFile)

Pour obtenir un objet Graphics, il y a plusieurs façons :

- Soit on instance un objet Graphics

```
Dim g as Graphics    'Graphics contient Nothing, je ne peux rien en faire.
```

Il faut donc y mettre un objet Image (un BitMap ou un MetaFile) pour pouvoir travailler dessus.

Pour obtenir un BitMap par exemple, on peut :
Soit créer un objet BitMap vide :

```
Dim newBitmap As Bitmap = New Bitmap(600, 400)
Dim g as Graphics = Graphics.FromImage(newBitmap)
```

Paramètres= taille du BitMap mais il y a plein de surcharges
Soit créer un BitMap à partir d'un fichier sur disque

C'est pratique si on veut modifier une image qui est dans un fichier: on la lit dans un BitMap puis on la passe dans l'objet Graphics.

```
Dim myBitmap as New Bitmap("maPhoto.bmp") 'Charge maPhoto dans le BitMap
Dim g as Graphics = Graphics.FromImage(myBitmap)    'Crée un Graphics et y met le BitMap
```

g est un Graphics contenant l'image 'maPhoto.bmp' que je peux modifier.

Attention : le Graphics n'est pas 'visible', pour le voir il faut le mettre dans un composant (un PictureBox par exemple) qui lui sera visible. On verra cela plus bas.

- Soit on appelle la méthode CreateGraphics d'un contrôle ou d'un formulaire

On appelle la méthode CreateGraphics d'un contrôle ou d'un formulaire afin d'obtenir une référence à un objet Graphics représentant la surface de dessin de ce contrôle ou formulaire. Cette méthode est utilisée si vous voulez dessiner sur un formulaire ou un contrôle existant ;

```
Dim g as Graphics
g = Me.CreateGraphics        'Pour un formulaire
Dim g as Graphics
g = Panel1.CreateGraphics 'Pour un contrôle Panel
```

On peut ensuite dessiner sur g, cela sera immédiatement visible.
Il faut quand on n'utilise plus l'objet graphics, utiliser la méthode Dispose pour le libérer.

- Soit on récupère l'objet Graphics argument de l'évènement Paint d'un contrôle.

L'événement Paint pour des contrôles se déclenche lorsque le contrôle est redessiné, un objet Graphics est fourni comme une valeur de PaintEventArgs.

Pour obtenir une référence à un objet Graphics à partir des PaintEventArgs de l'événement Paint

1. Déclarez l'objet Graphics
2. Assignez la variable pour qu'elle référence l'objet Graphics passé dans les PaintEventArgs.
3. Dessinez dans l'objet Graphics.

```
Private Sub Form1_Paint(sender As Object, pe As PaintEventArgs) Handles _
    MyBase.Paint

    Dim g As Graphics = pe. Graphics
    ' Dessiner dans pe ici...
End Sub
```

Noter bien que pe est visible uniquement dans Form1_Paint
Pour déclencher l'évènement Paint et dessiner, on utilise la méthode OnPaint

Comment dessiner ?

La classe Graphics fournit des méthodes permettant de dessiner

```
DrawImage   'Ajoute une image (BitMap ou MetaFile)
DrawLine    'Trace une ligne
DrawString  'Ecrit un texte
DrawPolygon 'Dessine un polygone
...
```

En GDI+ on envoie des paramètres à la méthode pour dessiner :

Exemple :

```
MonGraphique.DrawEllipse( New Pen(Couleur),r)          'cela dessine une ellipse
```

Les 2 paramètres sont: la couleur et le rectangle dans lequel on dessine.

Pour travailler on utilise les objets :

Brush (Brosse)	Utilisé pour remplir des surfaces fermées avec des motifs, des couleurs ou des bitmaps. Elles peuvent être pleine et ne contenir qu'une couleur. Dim SB= New SolidBrush(Color.Red) TextureBrush utilise une image pour remplir. Dim SB= New TextureBrush(MonImage) LinéarGradientBrush permet des dégradés (passage progressif d'une couleur à une autre). Dim SB= New LinearGradientBrush(PointDébut, PointFin,Color1, Color2) Les HatchBrush sont des brosses hachurées Dim HatchBrush hb = new HatchBrush(HatchStyle.ForwardDiagonal, Color.Green,Color.FromArgb(100, Color.Yellow)) Les PathGradient sont des brosses plus complexes.

Pen (Stylet)	Utilisé pour dessiner des lignes et des polygones, tels que des rectangles, des arcs et des secteurs. Comment créer un Stylet? Dim blackPen As New Pen(Color.Black) on donne la couleur Dim blackPen As New Pen(Color.Black, 3)on donne couleur et epaisseur Dim blackPen As New Pen(MyBrush) on peut même créer un stylet avec une brosse Propriétés de ce Stylet: DashStyle permet de faire des pointillés. StartCap et EndCap définissent la forme du début et de la fin du dessin (rond, carré, flèche...)
Font (Police)	Utilisé pour décrire la police utilisée pour afficher le texte. Dim Ft= New Font("Lucida sans unicode",60) 'paramètres=nom de font et taille Il y a de nombreuse surcharge. Dim Ft= New Font("Lucida sans unicode",60, FontStyle.Bold)'pour écrire en gras
Color (Couleur)	Utilisé pour décrire la couleur utilisée pour afficher un objet particulier. Dans GDI+, la couleur peut être à contrôle alpha. System.Drawing.Color.Red pour le rouge
Point	Ils ont des cordonnées x, y Dim point1 As New Point(120, 120) ' avec des integer ou des PointF 'avec des Singles
Rectangle	Dim r As New RectangleF(0, 0, 100, 100) On remarque que le F après Point ou Rectangle veut dire 'Float', et necessite l'usage de Single.

Comment faire ?

Dessiner une ligne sur le graphique :

Pour dessiner une ligne, on utilise DrawLine.

```
Dim blackPen As New Pen(Color.Black, 3)          'créer un stylet noir d'épaisseur 3
' Créer des points
Dim point1 As New Point(120, 120)        'créer des points
Dim point2 As New Point(600, 100)
' Dessine la ligne
e.Graphics.DrawLine(blackPen, point1, point2)
```

On aurait pu utiliser une surcharge de Drawline en spécifiant directement les coordonnées des points.

```
Dim x1 As Integer = 120
Dim y1 As Integer = 120
Dim x2 As Integer = 600
Dim y2 As Integer = 100
e.Graphics.DrawLine(blackPen, x1, y1, x2, y2)
```

Dessiner une ellipse :
Définir un rectangle dans lequel sera dessiné l'ellipse.

```
Dim r As New RectangleF(0, 0, 100, 100)
g.DrawEllipse(New Pen(Color.Red), r)' Dessinons l' ellipse
```

Dessiner un rectangle :

```
myGraphics.DrawRectangle(myPen, 100, 50, 80, 40)
```

Comme d'habitude on peut fournir après le stylet des coordonnées(4), des points (2) ou un rectangle.

Dessiner un polygone :

```
Dim MyPen As New Pen(Color.Black, 3)
' Créons les points qui définissent le polygone
Dim point1 As New Point(150, 150)
Dim point2 As New Point(100, 25)
Dim point3 As New Point(200, 5)
Dim point4 As New Point(250, 50)
Dim point5 As New Point(300, 100)
Dim point6 As New Point(350, 200)
Dim point7 As New Point(250, 250)
Dim curvePoints As Point() = {point1, point2, point3, point4, _
point5, point6, point7}
' Dessinons le Polygone.
e.Graphics.DrawPolygon(MyPen, curvePoints
```

Dessiner un rectangle plein :

```
e.FillRectangle(new SolidBrush(Color.red), 300,15,50,50)
```

Il existe aussi DrawArc, DrawCurve, DrawBezier DrawPie…

Ecrire du texte sur le graphique :
Pour cela on utilise la méthode DrawString de l'objet graphique:

```
g.DrawString ("Salut", Me.Font, New SolidBrush (ColorBlack), 10, 10)
```

Paramètres:

Texte à afficher.
Police de caractères
Brosse, cela permet d'écrire avec des textures.
Coordonnées.

Si on spécifie un rectangle à la place des 2 derniers paramètres, le texte sera affiché dans le rectangle avec passage à la ligne si nécessaire :

```
Dim rectangle As New RectangleF (100, 100, 150, 150 )
Dim T as String= "Chaîne de caractères très longue"
g.DrawString (T, Me.Font, New SolidBrush (ColorBlack), Rectangle)
```

On peut même imposer un format au texte :

Exemple, centrer le texte :

```
Dim Format As New StringFormat()
Format.Aligment=StringAlignment.Center
g.DrawString (T, Me.Font, New SolidBrush (ColorBlack), Rectangle, Format)
```

On peut mesurer la longueur (ou le nombre de lignes) d'une chaîne :
Avec MeasureString

Exemple, centrer le texte : pour cela, calculer la logeur de la chaine, puis calculer le milieu de l'écran moins la 1/2 longueur de la chaîne :

```
Dim W As Double= Me.DisplayRectangle.Width/2
Dim L As SizeF= e.Graphics.MeasurString (Texte, TextFont)
Dim StartPos As Double = W - (L.Width/2)
g.Graphics.MeasueString (T, Me.Font, New SolidBrush (ColorBlack), Rectangle,
StartPos, 10)
```

Exemple, calculer le nombre de ligne et le nombre de caractères d'une chaîne :

```
g.Graphics.MeasueString (T, Me.Font, New SolidBrush (ColorBlack), Rectangle,Nex
StringFormat() NombredeCaractères, NombredeLignes)
```

Ajouter une image sur le graphique :
Pour cela on utilise la méthode DrawImage de l'objet graphique :

```
g.Graphics.DrawImage(New Bitmap("sample.jpg"), 29, 20, 283, 212)
```

On peut travailler avec des images .jpeg .png .bmp .Gif .icon .tiff .exif

Travailler sur un Objet Image

Charger une image
Si on veut afficher une image bitmap ou vectoriel, il faut fournir à l'objet Graphics un objet bitmap ou vectoriel. C'est la méthode DrawImage qui reçoit l'objet Metafile ou Bitmap comme argument. L'objet BitMap, si on le désire peut contenir le contenue d'un fichier qui sera affiché.

```
Dim myBMP As New BitMap ("MonImage.bmp")

myGraphics.DrawImage(myBMP, 10, 10)
```

Le point de destination du coin supérieur gauche de l'image, (10, 10), est spécifié par les deuxième et troisième paramètres.

```
myGraphics.FromImage(myBMP)  'est aussi possible
```

On peut utiliser plusieurs formats de fichier graphique : BMP, GIF, JPEG, EXIF, PNG, TIFF et ICON.

Cloner une image
La classe Bitmap fournit une méthode Clone qui permet de créer une copie d'un objet existant. La méthode Clone admet comme paramètre un rectangle source qui vous permet de spécifier la portion de la Bitmap d'origine à copier. L'exemple suivant crée un objet Bitmap en clonant la moitié supérieure d'un objet Bitmap existant. Il dessine ensuite les deux images.

```
Dim originalBitmap As New Bitmap("Spiral.png")'on charge un fichier png dans un
BitMap
Dim sourceRectangle As New Rectangle(0, 0, originalBitmap.Width, _
originalBitmap.Height / 2) 'on définit un rectangle

Dim secondBitmap As Bitmap = originalBitmap.Clone(sourceRectangle, _
PixelFormat.DontCare)'on définit un second BitMap Clonant une partie du 1ere BitMap
avec le rectangle

'On met les 2 BitMap dans un Graphics
myGraphics.DrawImage(originalBitmap, 10, 10)
myGraphics.DrawImage(secondBitmap, 150, 10)
```

Enregistrer une image sur le disque
On utilise pour cela la méthode Save.
Exemple: enregistrer le BitMap newBitMap dans 'Image1.jpg'

```
newBitmap.Save("Image1.jpg", ImageFormat.Jpeg)
```

Comment voir un Graphics ?

Si on a instance un objet Graphics, on ne le voit pas. Pour le voir il faut le mettre dans un PictureBox par exemple:

Exemple :

```
Dim newBitmap As Bitmap = New Bitmap(200, 200) 'créons un BitMap
Dim g As Graphics = Graphics.FromImage(newBitmap)'créons un Graphics et y mettre le BitMap
Dim r As New RectangleF(0, 0, 100, 100)' Dessinons une ellipse
g.DrawEllipse(New Pen(Color.Red), r)
```

Comment voir l'ellipse ?
Ajoutons un PictureBox au projet, et donnons à la propriété Image de ce PictureBox le nom du BitMap du Graphics:

```
PictureBox1.Image = newBitmap
```

L'ellipse rouge apparaît!! Si, Si!!

Paint si Resize

Par défaut Paint n'est pas déclenché quand un contrôle ou formulaire est redimensionné, pour forcer à redessiner en cas de redimensionnement, il faut mettre le style Style.Resizedraw du formulaire ou du contrôle à true.

```
SetStyle (Style.Resizedraw, true)
```

Cette syntaxe marche, la suivante aussi (pour le formulaire)

```
Me.SetStyle (Style.Resizedraw, true) 'pour tous les objets du formulaire?
```

Mais PictureBox1.SetStyle (Style.Resizedraw, true) n'est pas accepté!!

Afficher un texte en 3D

Afficher un texte en 3d.

```
PrivateSub TextEn3D(ByVal g As Graphics, ByVal position As PointF, ByVal text AsString, ByVal ft As Font, ByVal c1 As Color, ByVal c2 As Color)
    Dim rect AsNew RectangleF(position, g.MeasureString(text, ft))
    Dim bOmbre AsNew LinearGradientBrush(rect, Color.Black, Color.Gray, 90.0F)

    g.DrawString(text, ft, bOmbre, position)

    position.X -= 2.0F
    position.Y -= 6.0F

    rect = New RectangleF(position, g.MeasureString(text, ft))
    Dim bDegrade AsNew LinearGradientBrush(rect, c1, c2, 90.0F)

    g.DrawString(text, ft, bDegrade, position)
EndSub
```

Espace de nom

Pour utiliser les graphiques il faut que System.Drawing soit importé (ce qui est fait par défaut). (Systém.Drawing.DLL comme références de l'assembly)

4.13Ajouter une aide

Quand l'utilisateur utilise votre logiciel, il est parfois en difficultés, comment l'aider?
Avec des aides que le programmeur doit créer et ajouter au programme.

Généralités sur les 4 sortes d'aides

La Class Help permet d'ouvrir un fichier d'aide.

Le composant HelpProvider offre 2 types d'aide.
- Le premier consiste à ouvrir un fichier d'aide grâce à F1 que l'utilisateur doit consulter.
- Quant au second, il peut afficher une aide brève pour chacun des contrôles en utilisant le bouton d'aide (?). Il s'avère particulièrement utile dans les boîtes de dialogue modal.

Le composant ToolTip offre lui :
- une aide propre à chaque contrôle des Windows Forms.

Les fichiers d'aide

On peut utiliser les formats :
- HTML Fichier .htm
- HTMLHelp 1.x ou version ultérieure) Fichier .chm
- HLP Fichier .hlp les plus anciens.

Comment créer ces fichiers :
Pour les fichiers HTM:
Utiliser Word, ou FontPage, ou Netscape Composer...

Pour les fichiers HLP:
Utiliser Microsoft HelpWorkshop livré avec VB6

Pour les fichiers CHM:
Thierry AIM fournit sur le site developpez.com un excellent:
Cours pour créer un fichier CHM - http://thierry_aim.developpez.com/htmlhelp/

On conseille d'utiliser plutôt les fichiers chm.

Utilisation des fichiers d'aide !

Appel direct :

La classe Help permet d'ouvrir directement par code un fichier d'aide.

C'est ce qu'on utilise dans le menu '?' d'un programme (sous menu 'Aide'); dans la procédure correspondante (Sub Aide_Click) on écrit :

```
Help.ShowHelp (Me, "MonAide.html")
```

MonAide.html doit être dans le fichier de l'application (répertoire Bin)
Cela peut être un URL, l'adresse d'une page sur Internet!!

Il peut y avoir un 3ème paramètres: on verra cela plus bas (C'est le même paramètre que la propriété HelpNagigator de HelpProvider).

Appel par la touche F1 :

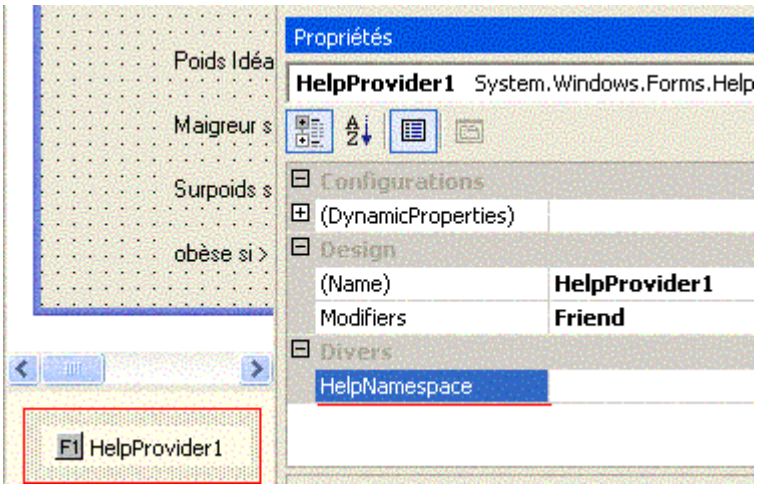
Vous pouvez utiliser le composant HelpProvider pour attacher des rubriques d'aide figurant dans un fichier d'aide (au format HTML, HTMLHelp 1.x ou ultérieur) à des contrôles spécifiques.

Quand on met un composant dans un formulaire (avec dans la propriété HelpNamespace, le nom de fichier d'aide), cela ajoute aux contrôles de ce formulaire les propriétés :

- HelpNavigator qui détermine le type d'appel (par numéro de rubrique, mot clé…)
- HelpKeyword qui contient le paramètre de recherche (le numéro de rubrique, le mot clé…)

Quand l'utilisateur est sur le contrôle et qu'il clique sur F1 la rubrique d'aide s'ouvre.

Pour créer cet aide :
Faites glisser un composant HelpProvider de la boîte à outils vers votre formulaire.
Le composant se place dans la barre d'état située au bas de la fenêtre.



Dans la fenêtre Propriétés du HelpProvider , donner à la propriété HelpNamespace, un nom de fichier d'aide .chm, col ou .htm.

Dans la fenêtre Propriétés du contrôle qui déclenchera l'aide, donner à la propriété HelpNavigator une valeur de l'énumération HelpNavigator.

Cette valeur détermine la façon dont la propriété HelpKeyword est passée au système d'aide.
HelpNagigator peut prendre la valeur :

AssociateIndex	Indique que l'index d'une rubrique spécifiée est exécuté dans l'URL spécifiée.
Find	Indique que la page de recherche d'une URL spécifiée est affichée.
Index	Indique que l'index d'une URL spécifiée est affiché.
KeywordIndex	Spécifie un mot clé à rechercher et l'action à effectuer dans l'URL spécifiée.
TableOfContents	Indique que le sommaire du fichier d'aide HTML 1.0 est affiché.
Topic	Indique que la rubrique à laquelle l'URL spécifiée fait référence est affichée.

Définissez la propriété HelpKeyword dans la fenêtre Propriétés. (la valeur de cette propriété sera passé au fichier d'aide afin de déterminer la rubrique d'aide à afficher).

Au moment de l'exécution, le fait d'appuyer sur F1 lorsque le contrôle (dont vous avez défini les propriétés HelpKeyword et HelpNavigator) a le focus ouvre le fichier d'aide associé à ce composant HelpProvider.

Remarque : Vous pouvez définir, pour la propriété HelpNamespace, une adresse http:// (telle qu'une page Web). Cela permet d'ouvrir le navigateur par défaut sur la page Web avec la chaîne indiquée dans la propriété HelpKeyword utilisée comme ancre (pour accéder à une section spécifique d'une page HTML).

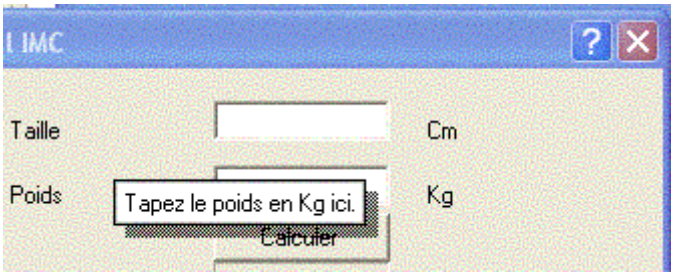
Dans le code il faut utiliser la syntaxe HelpProvider.SetHelpKeyword="..."

Exemple :
Pour afficher la page d'aide sur les formes ovales, sélectionnez la valeur HelpNavigator.KeyWordIndex dans la liste déroulante Help Navigator, dans la zone de texte HelpKeyword, 'tapez « ovales » (sans chevrons).

Utilisation du bouton d'aide :

Vous pouvez afficher l'aide pour un contrôle via le bouton Aide (?) situé dans la partie droite de la barre de titre.

Il faut que l'utilisateur clique sur le bouton d'aide (?) puis sur le contrôle qui nécessite une aide, ce qui entraîne l'ouverture d'un carré blanc contenant un message d'aide.



L'affichage de l'aide de cette façon convient particulièrement aux boîtes de dialogue. En effet, avec un affichage modal des boîtes de dialogue, il n'est pas facile d'ouvrir des systèmes d'aide externes, dans la mesure où les boîtes de dialogue modales doivent être fermées avant que le focus puisse passer à une autre fenêtre. Le bouton Réduire ou Agrandir ne doit pas être affiché dans la barre de titre. Il s'agit d'une convention pour les boîtes de dialogue alors que les formulaires disposent généralement de boutons Réduire et Agrandir.

Pour afficher l'aide contextuelle :
Faites glisser un composant HelpProvider de la boîte à outils vers votre formulaire.
Le contrôle est placé dans la barre d'état des composants située au bas de la fenêtre.
Attribuer aux propriétés Minimize et Maximize de la fenêtre la valeur false.

Puis,
Dans la fenêtre Propriétés de la fenêtre, donner à la propriété HelpButton la valeur true.
Cette configuration permet d'afficher dans la partie droite de la barre de titre du formulaire un bouton contenant un point d'interrogation.

Sélectionnez le contrôle pour lequel vous souhaitez afficher l'aide dans votre formulaire et mettre dans la propriété HelpString la chaîne de texte qui sera affichée dans une fenêtre de type ToolTip.

Essayer le bouton (?):
Appuyez surF5.
Appuyez sur le bouton Aide (?) de la barre de titre et cliquez sur le contrôle dont vous avez

défini la propriété HelpString. Le toolTip apparait.

Utilisation des infos bulle

Le composant ToolTip peut servir à afficher des messages d'aide courts et spécialisés relatifs à des contrôles individuels des Forms.

Cela ouvre une petite fenêtre indépendante rectangulaire dans laquelle s'affiche une brève description de la raison d'être d'un contrôle lorsque le curseur de la souris pointe sur celui-ci.

Il fournit une propriété qui précise le texte affiché pour chaque contrôle du formulaire.
En outre, il est possible de configurer, pour le composant ToolTip, le délai qui doit s'écouler avant qu'il ne s'affiche.

Comment faire :
Ajoutez le contrôle ToolTip au formulaire.

Chaque contrôle à maintenant une propriété ToolTip ou on peut mettre le texte a afficher dans l'info bulle.
Utilisez la méthode SetToolTip du composant ToolTip.
On peut aussi le faire par code :

```
ToolTip1.SetToolTip(Button1, "Save changes")
```

Par code créons de toute pièce un ToolTip.

```
Dim toolTip1 As New ToolTip()

' Modifions les délais du ToolTip.
toolTip1.AutoPopDelay = 6000
toolTip1.InitialDelay = 2000
toolTip1.ReshowDelay = 500
' Force le ToolTip a être visible que la fenêtre soir active ou non.
toolTip1.ShowAlways = True

' donne le texte de l'info bulle à 2 contrôles.
toolTip1.SetToolTip(Me.button1, "My button1")
toolTip1.SetToolTip(Me.checkBox1, "My checkBox1")
```

4.14Appel d'une API

Les Api (Application Programming Interface) sont des bibliothèques de liaisons dynamiques (DLL, Dynamic-Link Libraries), se sont des fonctions (généralement écrites en C) et qui sont compilées dans une DLL.

Elles font :

- soit partie intégrante du système d'exploitation Windows. (API Windows)

Se sont ces Api (Kernel32.Dll=coeur du système, User32Dll= fonctionnement des applications, gdi32.dll=interface graphique) que Windows utilise pour fonctionner.

Les fonctions sont donc écrites pour Windows, parfois on n'a pas d'équivalent VB, aussi, plutôt que de les réécrire quand on en a besoin, on appelle celles de Windows.

Elles permettent d'effectuer des tâches lorsqu'il s'avère difficile d'écrire des procédures équivalentes. Par exemple, Windows propose une fonction nommée FlashWindowEx qui vous permet de varier l'aspect de la barre de titre d'une application entre des tons clairs et foncés.

Il faut avouer que, le Framework fournissant des milliers de classes permettant de faire pratiquement tout ce que font les Api Windows, on a très peu à utiliser les Api Windows.

Chaque fois que cela est possible, vous devez utiliser du code managé à partir du .NET Framework plutôt que les appels API Windows pour effectuer des tâches.

- soit partie de dll spécifiques fournit par des tiers pour permettre d'appeler des fonctions n'existant pas dans VB ni Windows.

Par exemple, il existe des Api MySql qui donnent accès aux diverses fonctions permettant d'utiliser une base de données MySql. (Ces Api contiennent 'le moteur' de la base de données.)

Les Api sont en code non managé. De plus elles n'utilisent souvent pas les mêmes types de données que VB. L'appel des Api se faisant avec des passages de paramètres, il y a des précautions à prendre!! Sinon cela plante!!! Cela plante vraiment.

Les API Windows

L'avantage de l'utilisation d'API Windows dans votre code réside dans le gain de temps de développement, car elles contiennent des douzaines de fonctions utiles déjà écrites et prêtes à être utilisées. L'inconvénient des API Windows est qu'elles peuvent être complexes à utiliser et implacables lorsqu'une opération se déroule mal.

Pour plus d'informations sur les API Windows, consultez la documentation du kit de développement Win32 SDK dans les API Windows du kit de développement Platform SDK. Pour plus d'informations sur les constantes utilisées par les API Windows, examinez les fichiers d'en-tête, tels que Windows.h, fournis avec le kit de développement Platform SDK.

MSDN donne aussi une description des Api : http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winprog/winprog/overview_of_the_windows_api.asp

Appels API avec Declare
La façon la plus courante d'appeler les API Windows consiste à utiliser l'instruction Declare.

Exemple (repris de chez Microsoft) : appel de la fonction Windows 'MessageBox' qui est dans user32.dll et qui affiche une MessageBox.

- Rechercher de la documentation de la fonction:

Le site MSDN donne la définition de la fonction MesssageBox :

```
Int MessageBox( HWND
        hWnd, LPCTSTR
        lpText, LPCTSTR
        lpCaption, UINT
        uType
);
```

Parameters:

hWnd

[in] Handle to the owner window of the message box to be created. If this parameter is NULL, the message box has no owner window.

lpText

[in] Pointer to a null-terminated string that contains the message to be displayed.

lpCaption

[in] Pointer to a null-terminated string that contains the dialog box title. If this parameter is NULL, the default title Error is used.

uType

[in] Specifies the contents and behavior of the dialog box. This parameter can be a combination of flags from the following groups of flags.

Constantes API Windows : Vous pouvez déterminer la valeur numérique de des constantes utiliser dans les Api par l'examen des instructions #define dans le fichier WinUser.h. Les valeurs numériques sont généralement affichées au format hexadécimal. Par conséquent, vous pouvez les convertir au format décimal.

Par exemple, si vous voulez combiner les constantes pour le style exclamation MB_ICONEXCLAMATION 0x00000030 et le style Oui/Non MB_YESNO 0x00000004, vous pouvez ajouter les nombres et obtenir un résultat de 0x00000034, ou 52 décimales.

Return Value

IDABORT	Abort button was selected.
IDCANCEL	Cancel button was selected.
IDCONTINUE	Continue button was selected.
IDIGNORE	Ignore button was selected.
IDNO	No button was selected.
IDOK	OK button was selected.
IDRETRY	Retry button was selected.
IDTRYAGAIN	Try Again button was selected.
IDYES	Yes button was selected.

- Il faut déclarer la procédure DLL

Ajoutez la fonction Declare suivante à la section de déclaration du formulaire de départ de votre projet ou à celle de la classe ou du module où vous voulez utiliser la DLL :

```
Declare Auto Function MBox Lib "user32.dll" _
Alias "MessageBox" (ByVal hWnd As Integer, _
        ByVal txt As String, ByVal caption As String, _
        ByVal Typ As Integer) As Integer
```

Declare comprend les éléments suivants.
Le modificateur Auto indique de suivre les règles du Common Language Runtime.
Le nom qui suit Function est celui que votre programme utilise pour accéder à la fonction

importée.

Le mot clé Alias indique le nom réel de cette fonction.

Lib suivi du nom et de l'emplacement de la DLL qui contient la fonction que vous appelez. Vous n'avez pas besoin d'indiquer le chemin d'accès des fichiers situés dans les répertoires système Windows.

Utilisez le mot clé Alias si le nom de la fonction que vous appelez n'est pas un nom de procédure Visual Basic valide ou est en conflit avec le nom d'autres éléments de votre application. Alias indique le nom réel de la fonction appelée.

Les types de données que Windows utilise ne correspondent pas à ceux de Visual Studio. Visual Basic effectue la plupart des tâches à votre place en convertissant les arguments en types de données compatibles, processus appelé marshaling. Vous pouvez contrôler de manière explicite la façon dont les arguments sont marshalés en utilisant l'attribut MarshalAs défini dans l'espace de noms System.Runtime.InteropServices.

Remarque : Les versions antérieures de Visual Basic vous autorisaient à déclarer des paramètres As Any (tout type). Visual Basic.NET ne le permet pas.

Ajoutez des instructions Const a la section des déclarations de votre classe ou module pour rendre ces constantes disponibles pour l'application.

Par exemple :

```
Const MB_ICONQUESTION = &H20L
Const MB_YESNO = &H4
Const IDYES = 6
Const IDNO = 7
```

Pour appeler la procédure DLL

```
Dim RetVal As Integer ' Valeur de retour.

RetVal = MBox(0, "Test DLL", "Windows API MessageBox", _
              MB_ICONQUESTION Or MB_YESNO)
If RetVal = IDYES Then
    MsgBox("Vous avez cliqué sur OUI")
Else
    MsgBox("Vous avez cliqué sur NON")
End If
```

Visual Basic.NET convertit automatiquement les types de données des paramètres et valeurs de retour pour les appels API Windows, mais vous pouvez utiliser l'attribut MarshalAs pour indiquer de façon explicite les types de données non managés attendus par une API.

On peut aussi appeler une API Windows à l'aide de l'attribut DllImport mais c'est compliqué.

Autre exemple classique

Utilisation de la routine BitBlt qui déplace des octets.
La documentation donne les renseignements suivants :

```
Declare Function BitBlt Lib "gdi32" ( _
    ByVal hDestDC As Long, _
    ByVal x As Long, _
    ByVal y As Long, _
    ByVal nWidth As Long, _
```

```
    ByVal nHeight As Long, _
    ByVal hSrcDC As Long, _
    ByVal xSrc As Long, _
    ByVal ySrc As Long, _
    ByVal dwRop As RasterOps _
    ) As Long
```

Parameter Information

☐ hdcDest
Identifies the destination device context.

☐ nXDest
Specifies the logical x-coordinate of the upper-left corner of the destination rectangle.

☐ nYDest
Specifies the logical y-coordinate of the upper-left corner of the destination rectangle.

☐ nWidth
Specifies the logical width of the source and destination rectangles.

☐ nHeight
Specifies the logical height of the source and the destination rectangles.

☐ hdcSrc
Identifies the source device context.

☐ nXSrc
Specifies the logical x-coordinate of the upper-left corner of the source rectangle.

☐ nYSrc
Specifies the logical y-coordinate of the upper-left corner of the source rectangle.

☐ dwRop
Specifies a raster-operation code.

Les Constantes dwRop

```
' Copies the source bitmap to destination bitmap
SRCCOPY = &HCC0020

' Combines pixels of the destination with source bitmap using the Boolean AND
operator.
SRCAND = &H8800C6

' Combines pixels of the destination with source bitmap using the Boolean XOR
operator.
SRCINVERT = &H660046

' Combines pixels of the destination with source bitmap using the Boolean OR operator.
SRCPAINT = &HEE0086

' Inverts the destination bitmap and then combines the results with the source bitmap
' using the Boolean AND operator.
SRCERASE = &H4400328

' Turns all output white.
```

```
WHITENESS = &HFF0062
' Turn output black.
BLACKNESS = &H42
```

Return Values

If the function succeeds, the return value is nonzero.

Ici on va utiliser cette routine pour copier l'image de l'écran dans un graphics.

```
Private Declare Function BitBlt Lib "gdi32.dll" Alias "BitBlt" (ByVal _ hdcDest
        As IntPtr, ByVal nXDest As Integer, ByVal nYDest As _ Integer,
        ByVal nWidth As Integer, ByVal nHeight As Integer, ByVal _
        hdcSrc As IntPtr, ByVal nXSrc As Integer, ByVal nYSrc As Integer, _
        ByVal dwRop As System.Int32) As Long
Dim memoryImage As Bitmap

Private Sub CaptureScreen()
        Dim mygraphics As Graphics = Me.CreateGraphics()
        Dim s As Size = Me.Size
        memoryImage = New Bitmap(s.Width, s.Height, mygraphics)
        Dim memoryGraphics As Graphics = Graphics.FromImage(memoryImage)
        Dim dc1 As IntPtr = mygraphics.GetHdc
        Dim dc2 As IntPtr = memoryGraphics.GetHdc

        BitBlt(dc2, 0, 0, Me.ClientRectangle.Width, Me.ClientRectangle.Height,
        dc1, 0, 0, 13369376)

        mygraphics.ReleaseHdc(dc1)
        memoryGraphics.ReleaseHdc(dc2)
End Sub
```

Le dernier paramètre a pour valeur= 13369376= SRCCOPY = &HCC0020 et correspond à 'Copies the source bitmap to destination bitmap'.

4.15Drag and Drop

L'exécution d'opérations glisser-déplacer (Drag ans Drop) peut être ajoutée dans un programme.

La méthode DoDragDrop du contrôle de départ autorise la collecte des données au début de l'opération.

Les évènements DragEnter, DragLeave et DragDrop. permettent de 'poser' les données dans le contrôle d'arrivée.

Exemple n°1 (simple) :

Le contrôle de départ est un contrôle Button, les données à faire glisser sont la chaîne représentant la propriété Text du contrôle Button, et les effets autorisés sont la copie ou le déplacement. Le texte sera déposé dans un textBox :

Le contrôle de départ

La fonctionnalité qui autorise la collecte des données au début de l'opération glisser dans la méthode DoDragDrop.

L'événement MouseDown du contrôle de départ est généralement utilisé pour démarrer l'opération glisser parce qu'il est le plus intuitif (la plupart des glisser-déplacer commencent par un appuie sur le bouton de la souris).

Mais, souvenez-vous que n'importe quel événement peut servir à initialiser une procédure glisser-déplacer.

Remarque : Les contrôles ListView et TreeView, , ont un événement ItemDrag qui est spécifique.

```
Private Sub Button1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles Button1.MouseDown
        Button1.DoDragDrop(Button1.Text, DragDropEffects.Copy Or
        DragDropEffects.Move)
End Sub
```

Le premier argument indique les données à déplacer.
Le second les effets permis = copier ou déplacer.

Le contrôle d'arrivée

Toute zone d'un Windows Form ou d'un contrôle peut être configurée pour accepter les données déplacées en définissant la propriété AllowDrop et en gérant les événements DragEnter et DragDrop.

Dans notre exemple, c'est un controle TextBox1 qui est le contrôle d'arrivée.
TextBox1.AllowDrop =True 'autorise le contrôle TextBox à recevoir

Dans l'événement DragEnter du contrôle qui doit recevoir les données déplacées.
Vérifier que le type des données est compatible avec le contrôle d'arrivée (ici, vérifier que c'est bien du texte).

Définir ensuite l'effet produit lorsque le déplacement a lieu en lui attribuant une valeur de l'énumération DragDropEffects. (Ici il faut copier).

```
Private Sub TextBox1_DragEnter(ByVal sender As Object, ByVal e As
```

```
System.Windows.Forms.DragEventArgs) Handles TextBox1.DragEnter
  If (e.Data.GetDataPresent(DataFormats.Text)) Then
       e.Effect = DragDropEffects.Copy
  Else
  e.Effect = DragDropEffects.None
  End If
End Sub
```

Dans l'événement DragDrop du contrôle d'arrivée, utilisez la méthode GetData pour extraire les données que vous faites glisser.

```
Private Sub TextBox1_DragDrop(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles TextBox1.DragDrop
  TextBox1.Text = e.Data.GetData(DataFormats.Text).ToString
End Sub
```

Exemple n°2 (plus complexe) :

Glisser déplacer une ligne d'une listBox 'ListBox1' vers une listBox 'ListBox2'.
Créer une ListBox1
Créer une listBox2 avec sa propriété AllowDrop=True 'listBox2 accepte le 'lâcher'

Dans l'en-tête du module ajouter :

```
Public IndexdInsertion As Integer          ' Variable contenant l'index ou sera inséré la
ligne
```

`Eventuellement pour l'exemple charger les 2 ListBox avec des chiffres pour pouvoir tester.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase
Dim i As Integer
For i = 1 To 100
ListBox1.Items.Add(i.ToString)
Next
For i = 1 To 100
ListBox2.Items.Add(i.ToString)
Next
End Sub
```

'Dans le listBox de départ, l'évènement MouseDrown déclenche le glisser déplacer par DoDragDrop.

```
Private Sub ListBox1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles ListBox1.MouseDown
ListBox1.DoDragDrop(ListBox1.Items(ListBox1.IndexFromPoint(e.X, e.Y)),
DragDropEffects.Copy Or DragDropEffects.Move)
End Sub
```

'ListBox1.IndexFromPoint(e.X, e.Y) retourne l'Index de l'item ou se trouve la souris à partir des coordonnées e.x et e.y du pointeur)
'DoDragDrop a 2 arguments: l'élément à draguer et le mode

'DragOver qui survient quand la souris se ballade sur le contrôle d'arrivé, vérifie si le Drop reçoit bien du texte et met dans IndexdInsertion le listItem qui est sous la souris.
'Noter que e.x et e.y sont les coordonnées écran, il faut les transformer en coordonnées client (du contrôle) par PointToClient afin d'obtenir l'index de l'item ou se trouve la souris (en utilisant IndexFromPoint.

```
Private Sub ListBox2_DragOver(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles ListBox2.DragOver
```

```
        If Not (e.Data.GetDataPresent(GetType(System.String))) Then
        e.Effect = DragDropEffects.None
        Else
        IndexdInsertion = ListBox2.IndexFromPoint(ListBox2.PointToClient(New Point(e.X,
        e.Y)))
        e.Effect = DragDropEffects.Copy
        End If
        End Sub

'Enfin dans DragDrop, on récupère le texte dans Item et on ajoute un item après l'item pointé.
        Private Sub ListBox2_DragDrop(ByVal sender As Object, ByVal e As
        System.Windows.Forms.DragEventArgs) Handles ListBox2.DragDrop
        Dim item As Object = CType(e.Data.GetData(GetType(System.String)), System.Object)
        ListBox2.Items.Insert(IndexdInsertion + 1, item)
        End Sub
```

4.16 Multithreads :

4.16-1 - Un Thread, c'est quoi ?

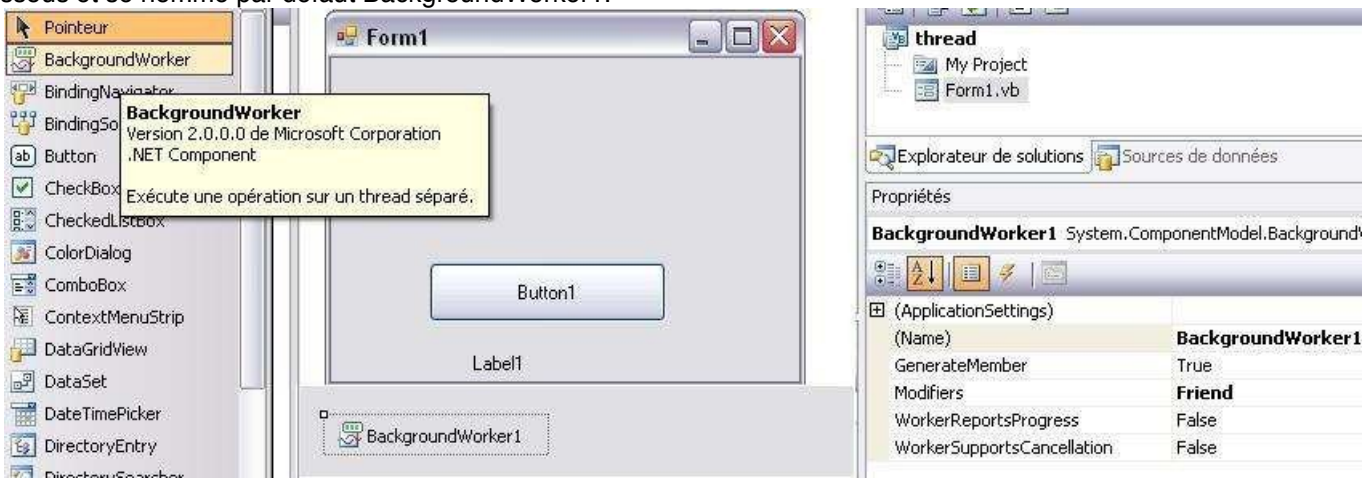
Le **thread** représente l'exécution d'un processus en mémoire. Un système multithread tel que Windows offre la capacité d'exécuter en parallèle plusieurs threads et donc plusieurs traitements en simultané.

On peut utiliser **la Classe Thread**, créer autant de thread que l'on veut, mais il faut gérer un tas de chose et c'est l'horreur.

On peut aussi (Framework 2) utiliser un **Thread d'arrière plan** (et un seul) qui est très simple d'utilisation. Son intérêts est que lorsqu'on a une tache très longue (très long calcul par exemple), il est possible d'effectuer le calcul long en arrière plan, pendant ce temps, on peut continuer à travailler dans le formulaire (thread principal); quand le thread d'arrière plan est terminé, on affiche les résultats.

4.16-2 - Comment ajouter un Thread d'arrière plan ?

Il faut aller chercher un composant BackgroundWorker dans la boite à outils et le déposer sur le formulaire, il apparaît en dessous et se nomme par défaut BackgroundWorker1.



La propriété WorkerReportsProgress donne à notre BackgroundWorker la possibilité de nous informer ou non de son état d'avancement.

La propriété WorkerSupportsCancellation nous permet d'autoriser l'annulation de la tâche en cours du BackgroundWorker.

Dans le code:

BackGroundWorker1.RunWorkerAsync(Objet) permet de déclencher le thread darrière plan.

BackGroundWorker1.DoWork : est
l' évènement qui se déclenche lorsque nous faisons appel au BackgroundWorker.
C'est cette routine qui tourne en arrière plan.

ProgressChanged : Cet évènement, si la propriété WorkerReportsProgress est activée,
se déclenche lorsque nous voulons indiquer que létat d'avancement du BackgroundWorker change.

RunWorkerCompleted : Une fois le traitement du BackgroundWorker terminé cet événement est

déclenché.

Exemple:

Si on clique sur un bouton cela crée un thread d'arrière plan qui effectue un calcul long.

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

    'La méthode RunWorkerAsync() du BackgroundWorker déclenche le thread d'arrière plan.

    BackgroundWorker1.RunWorkerAsync()

End Sub
```

```
'La procédure DoWork contient le code effectué en arrière plan.

Private Sub BackgroundWorker1_DoWork(ByVal sender As System.Object, _
  ByVal e As System.ComponentModel.DoWorkEventArgs) Handles BackgroundWorker1.DoWork

    'mes calculs très long

End Sub

'Quand le code d'arrière plan est terminé la procédure RunWorkerCompleted est exécutée.

Private Sub BackgroundWorker1_RunWorkerCompleted(ByVal sender As Object, _
  ByVal e As System.ComponentModel.RunWorkerCompletedEventArgs) _
  Handles BackgroundWorker1.RunWorkerCompleted

' ici, elle affiche un message indiquant de le thread d'arrière plan est terminé.

Label1.Text = "terminé"

End Sub
```

La méthode RunWorkerAsync peut avoir un paramètre qui sera transmis au thread d'arrière plan.

Mais un seul; ce paramètre étant de type objet, vous pouvez passer un tableau d'objets (string, int, etc...) ou même une structure

Ici dans l'exemple, on a un paramètre numérique, utilisé dans le thread d'arrière plan pour faire un calcul.

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

   BackgroundWorker1.RunWorkerAsync(180)

End Sub
```

Le paramètre , dans DoWork, se retrouve dans e.Argument , comme c'est un Objet, il faut le convertir en Integer pour l'utiliser:

```
Private Sub BackgroundWorker1_DoWork(ByVal sender As System.Object, _
  ByVal e As System.ComponentModel.DoWorkEventArgs) Handles

    BackgroundWorker1.DoWork a=a + Ctype (e.Argument, Integer)

End Sub
```

Le thread d'arrière plan peut appeler une Sub.

```
Private Sub BackgroundWorker1_DoWork(ByVal sender As System.Object, _
  ByVal e As System.ComponentModel.DoWorkEventArgs) Handles BackgroundWorker1.DoWork

    Calcul()

End Sub

Sub Calcul ()

'Mes calculs

End Sub
```

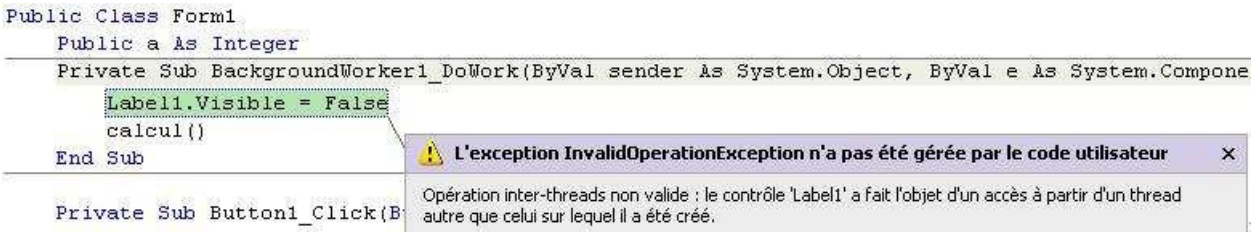
(Le thread principal peut lui aussi appeler la routine Calcul.)

Les variables sont accessibles dans le thread d'arrière plan:

```
'MyVar  par exemple qui est  Public  et  déclarée  en tête  de module.

Public  MyVar  As  Integer = 1
Private Sub  BackgroundWorker1_DoWork(ByVal  sender As  System.Object,  _
  ByVal e  As  System.ComponentModel.DoWorkEventArgs) Handles  BackgroundWorker1.DoWork

        MyVar=Myvar  +1

End  Sub
```

Par contre les objets de l'interface (du thread principal) ne sont pas accessibles dans le thread d'arrière plan:

Cela déclenche une exception si on tente d'y accéder.



4.16-3 - État d'avancement

Si la tâche d'arrière plan est très longue, il peut être intéressant de montrer dans l'interface utilisateur, l'état d'avancement de cette tâche.

Mais on rappelle que la tâche de fond ne peut pas intervenir sur l'interface.

Il faut donc:

Mettre la propriété WorkerReportsProgress de notre BackgroundWorker à True.

Dans le thread d'arrière plan, il faut, a chaque que l'on veut indiquer la progression, appeler la méthode ReportProgress en indiquant l'état d'avancement avec un paramètre.

```
Private Sub  BackgroundWorker1_DoWork()
  Dim  MyThread  As  BackgroundWorker = CType(sender,
  BackgroundWorker)'récupération  du   thread d'arrière  plan

  MyThread.ReportProgress(pourcent)'pourcent  est un   Integer indiquant l'état d'avancement.

End  Sub
```

Noter que c'est au programmeur de créer la logique calculant d'état d'avancement (et donc la valeur de la variable pourcent)

Enfin dans le thread principal, la Sub BackgroundWorker1_ProgressChanged() s'exécute à chaque fois que le thread d'arrière plan le demande et met à jour un index visuel sur l'interface.

```
Private Sub  BackgroundWorker1_ProgressChanged(  _
ByVal sender  As  Object,  _
ByVal e  As  ProgressChangedEventArgs)  _
Handles  BackgroundWorker1.ProgressChanged

MyProgressBarr.Value =  e.ProgressPercentage
End  Sub
```

4.16-4 - Arrêter le thread en cours

Il suffit de faire dans le thread principal:

```
BackgroundWorker1.CancelAsync()
```

Dans le thread d'arrière plan, il faut vérifier si l'arrêt à été demandé:

Dans DoWork on récupère le thread d'arrière plan qui est le sender, on regarde si sa propriété CancellationPending est à True, si oui on met e.cancel à True ce qui arrête le thread d'arrière plan.

```
Dim MyThread As BackgroundWorker = CType(sender, BackgroundWorker)

If MyThread.CancellationPending Then e.Cancel = True 'ce qui arrête le thread d'arrière plan.
```

Si on veut tester la demande d'arrêt dans une Sub, il faut envoyer en paramètre à cette sub MyThread et e.

4.16-5 - Résultat retourné par le thread d'arrière plan

Il peut y avoir plusieurs types de résultat à la fin, on peut le voir dans l'argument e de type RunWorkerCompletedEventArgs retourné par la procédure BackgroundWorker1.RunWorkerCompleted.

* Il y a eu une erreur pendant le traitement. Dans ce cas la propriété e.Error est différente de null.

* Le traitement a été annulé. Dans ce cas la propriété e.Canceled est à true.

* Le traitement s'est déroulé normalement. Le résultat se trouve dans la propriété e.Result .(Bien sur ,dans DoWork il faut avoir mis le résultat des calculs dans e.Result)

Exemple de traitement:

```
Private Sub BackgroundWorker1_RunWorkerCompleted( _
 ByVal sender As Object, _
 ByVal e As RunWorkerCompletedEventArgs) _ Handles
 BackgroundWorker1.RunWorkerCompleted

 If Not (e.Error Is Nothing) Then
  lblResult.Text = "Il y a eu une erreur : " + e.Error.Message
 ElseIf e.Cancelled Then
  lblResult.Text = "Opération annulée "
 Else
  lblResult.Text = "Opération Ok Résultat : " + e.Result.ToString
 End If

End Sub
```

4.20Déboguage

Le déboguage est la recherche des bugs. (Voir 4.3 Traiter les erreurs)

Pour déboguer, il faut lancer l'exécution du programme, suspendre l'exécution à certains endroits du code et voir ce qui se passe puis faire avancer le programme pas à pas :

Pour démarrer et arrêter l'exécution, on utilise les boutons suivants :

On lance le programme avec le premier bouton, on le suspend avec le second, on l'arrête définitivement avec le troisième…

On peut suspendre (l'arrête temporairement) le programme :

- avec le second bouton
- grâce à des points d'arrêt (pour définir un point d'arrêt en mode de conception, cliquez en face d'une ligne dans la marge grise : la ligne est surlignée en marron. Quand le code est exécuté, il s'arrête sur cette ligne marron).

```
For i= 1 To 6
Tableau(i)=i*i
Next i
```

En plus si on clique sur le rond de gauche avec le bouton droit de la souris, on ouvre un menu permettant de modifier les propriétés de ce point d'arrêt (il y a la possibilité d'arrêter au premier ou au Xième passage sur le point d'arrêt, ou arrêter si une expression est à True ou à changé)

- en appuyant sur Ctrl-Alt-Pause
- en incluant dans le code une instruction Stop

Attention : Si vous utilisez des instructions Stop dans votre programme, vous devez les supprimer avant de générer la version finale.

Les transformer en commentaire :

```
' Stop
```

Ou utiliser des instructions conditionnelles :

```
#If DEBUG Then
Stop
#End If
```

Déboguage

Quand le programme est suspendu, on peut observer les variables, déplacer le point d'exécution, on peut aussi faire marcher le programme pas à pas (instruction par instruction) et observer l'évolution de la valeur des variables, on peut enfin modifier la valeur d'une variable afin de tester le logiciel avec cette valeur.

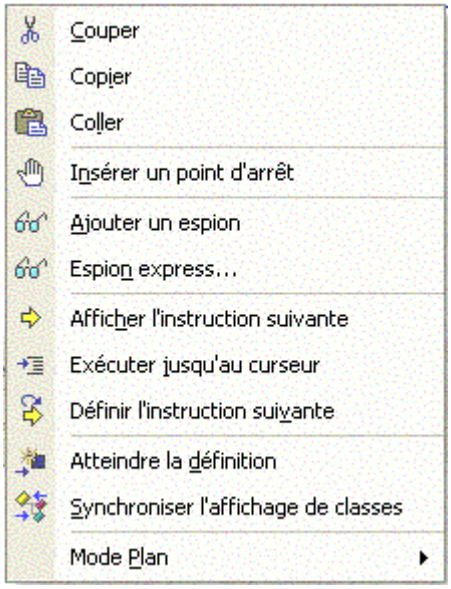
F11 permet l'exécution pas à pas (y compris des procédures appelées : si il y a appel à une autre procédure, le pas à pas saute dans l'autre procédure)

F10 permet le pas à pas (sans détailler les procédures appelées : exécute la procédure appelée en une fois)

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

On peut afficher ou définir l'instruction suivante, exécuter jusqu'au curseur, insérer un point d'arrêt ou un espion en cliquant sur le bouton droit de la souris et en

choisissant une ligne du menu.



Espion express permet de saisir une expression (variable, calcul de variables) et de voir ensuite dans une fenêtre 'espion' les modifications de cet expression au cours du déroulement du programme.

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

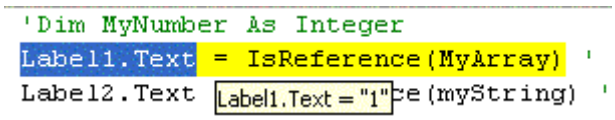
- Automatique, qui affiche les valeurs des variables de l'instruction en cours et des instructions voisines.
- Immédiat où il est possible de taper des instructions ou expressions pour les exécuter ou voir des valeurs.
  Taper "?I" (c'est l'équivalent de "Print I" qui veut dire: écrire la valeur de la variable I) puis valider, cela affiche la valeur de la variable I.

  Autre exemple, pour voir le contenu d'un tableau A(), tapez sur une seule ligne : "For i=0 to 10: ?i: Next i"

  Enfin, il est possible de modifier la valeur d'une variable: Tapez" I=10" puis validez, cela modifie la valeur de la variable.
- Espions permettant d'afficher le contenu de variables ou d'expressions.
- Espions Express permet d'afficher la valeur de l'expression sélectionnée.
- Points d'arrêts permet de modifier les propriétés des points d'arrêts. On peut mettre un point d'arrêt en cliquant dans la marge grise à gauche: l'instruction correspondante s'affiche en marron et l'exécution s'arrêtera sur cette ligne.
- Me affiche les données du module en cours.
- Variables locales affiche les variables locales.
- Modules affiche les dll ou .exe utilisés.
- Mémoire, Pile d'appels, Thread, Registres, Code Machine permettent d'étudier le fonctionnement du programme à un niveau plus spécialisé et technique.

Comment voir rapidement la valeur de propriétés ou de variables.
Il est toujours possible de voir la valeur d'une propriété d'un objet en la sélectionnant avec la souris:

Exemple on sélectionne label1.Text et on voit apparaître sa valeur.



Pour les variables, il suffit que le curseur soit sur une variable pour voir la valeur de cette variable.

On peut aussi copier une expression dans la fenêtre 'immédiat', mettre un ? avant et valider pour voir la valeur de l'expression.

Attention à l'affichage :
Parfois en mode pas à pas on regarde le résultat d'une instruction dans la fenêtre du programme. Par exemple on modifie la propriété text d'un label et on regarde si le label a bien changé.

Parfois la mise à jour n'est pas effectuée car le programme met à jour certains contrôles seulement en fin de procédure. Pour palier à cela et afficher au fur et à mesure, même si la procédure n'est pas terminée, on utilise la méthode Refresh de l'objet qui 'met à jour'.

Exemple :

```
Label1.text="A"
Label1.Refresh
```

Cela ne semble pas toujours fonctionner. Avez-vous une explication ?

Objet Console

On peut écrire sur la console, quand on a parfois besoin d'afficher des informations, mais uniquement pour le programmeur:

```
Console.WriteLine( myKeys(i))
```

Mais dans un programme Windows, il n'y a pas de console!! la sortie est donc envoyée vers la fenêtre de sortie (voir Debug)

Objet Debug

L'espace de noms Systems.Diagnostics est nécessaire.

Pour déboguer du code, on a parfois besoin d'afficher des informations, mais uniquement pour le programmeur, en mode debug afin de suivre le cheminement du programme ou la valeur d'une variable ou si une condition se réalise; pour cela on utilise une fenêtre nommée 'Sortie'(Output). (Menu Affichage>Autres fenêtres>Sortie)

Pour écrire dans la fenêtre Output (sans arrêter le programme):

- Du texte :
  `Debug.Write(Message)`
- Ajouter un passage à la ligne :
  `Debug.WriteLine(Message)`
- Le contenu d'une variable :
  `Debug.Write(Variable)`
- Les propriétés d'un objet :
  `Debug.Write(Objet)`

Exemple :

```
Debug.Write("ça marche")  'Affiche 'ça marche'
Dim A as Integer=2
Debug.Write(A)        'Affiche 2
Debug.Write(A+2)    'Affiche 4
```

On voit que s'il y a une expression, elle est évaluée.
On peut aussi afficher un message si une condition est remplie en utilisant WriteLineIf ou WriteIf :

```
Debug.WriteLineIf(i = 2, "i=2")
```

Affiche 'i=2' si i=2

Cela vous permet, sans arrêter le programme (comme le fait Assert), d'être informé quand une condition est vérifiée.

Debug.Assert par contre affiche une fenêtre Windows et stoppe le programme si une assertion (une condition) passe à False.

```
Debub.Assert(Assertion)
Debub.Assert(Assertion, Message1)
Debub.Assert(Assertion, Message1, Message2)
```

L'exemple suivant vérifie si le paramètre 'type' est valide. Si le type passé est une référence null (Nothing dans Visual Basic), Assert ouvre une boite de dialogue nommé 'Echec Assertion' avec 3 boutons 'Abandonner, Recommencer' 'Ignorer'... La liste des appels est affichée dans la fenêtre (procédure en cours en tête de liste, module et numéro de ligne en première ligne)

```
Public Shared Sub UneMethode (type As Type, Typedeux As Type)
Debug.Assert( Not (type Is Nothing), "Le paramètre Type est=Nothing ", "Je ne peux
pas utiliser un Nothing")
....
End Sub UneMethode

Debug.Fail
```

Fait pareil mais sans condition.

Objet Trace

Trace possède les mêmes fonctions que Debug (Write, WriteIf, Assert, Fail..) mais la différence est que Trace permet d'afficher à l'utilisateur final par défaut.

Trace est activé par défaut. Par conséquent, du code est généré pour toutes les méthodes Trace dans les versions release et debug. Ceci permet à un utilisateur final d’activer le traçage pour faciliter l’identification du problème sans que le programme ait à être recompilé.

Par opposition, Debug est désactivé par défaut dans les versions release, donc aucun code exécutable n’est généré pour les méthodes Debug.

SECTION 7 : Règles de bonne programmation et d'optimisation

7.2 Règles de bonne programmation

Pour faire un code solide, éviter les bugs, avoir une maintenance facile, il faut suivre quelques règles.

Au niveau du projet

Découper un traitement complexe en plusieurs petites routines effectuant chacune une fonction précise.

Découper les différentes fonctions du logiciel en Module et procédures, ou en Objet (Créer des Classes dont les méthodes seront les diverses routines). (Voir la leçon 5.10)

Séparer l'interface utilisateur et l'applicatif.

Exemple, pour un formulaire affiche les enregistrements d'une base de données :

Créer :
- Les fenêtres dont le code gère uniquement l'affichage. C'est l'interface utilisateur ou IHM (Interface Homme Machine)
- une Classe gérant uniquement l'accès aux bases de données.

Cela facilite la maintenance : si on désire modifier l'interface, on touche au fenêtre et pas du tout à la Classe base de données.

Architecture à 3 niveaux.
Elle peut être nécessaire dans certains programmes, les 3 niveaux sont :
- Application, interface.
- Logique.
- Données.

Exemple, un formulaire affiche certains enregistrements d'une base de données.
- L'interface affiche les enregistrements.
- Les classes ou modules 'logiques' déterminent les bons enregistrements.
- Les classes ou modules données vont chercher les données dans la base de données.

Si au lieu de travailler sur une base Access, je travaille sur une base SQLServer, il suffit de réécrire la troisième couche.

Dans un module

Respecter l'ordre suivant :
1. Instructions Option
2. Instructions Imports
3. Procédure Main
4. Instructions Class, Module et Namespace, le cas échéant

Dans une Class
1. Instructions Declare
2. Déclaration des variables (Public Private)
3. Sub ou Function

Sous peine d'erreurs à la compilation.

Rendre le code lisible

- Ajoutez des commentaires

Pour vous, pour les autres.

Au début de chaque routine, Sub, Function, Classe, noter en commentaire ce qu'elle fait et quelles sont les caractéristiques des paramètres :
- Le résumé descriptif de la routine, la Sub ou Function.
- Une description de chaque paramètre.
- La valeur retournée s'il y en a une.
- Une description de toutes les exceptions…
- Un exemple d'utilisation
- Une explication sur le fonctionnement de la routine.

Ne pas ajouter de commentaire en fin de ligne (une partie ne sera pas visible) mais plutôt avant la ligne de code. Seule exception ou on utilise la fin de ligne: les commentaires après les déclarations de variable.

```
Dim i As Integer       'Variable de boucle
        'Parcours du tableau à la recherche de…
For i=0 To 100
...
```

Paradoxalement, trop de commentaires tue le code autant que le manque de commentaires.

Pour éviter de tomber dans le tout ou rien, fixons nous quelques règles :
- Commentez le début de chaque Sub, Fonction, Classe
- Commentez toutes les déclarations de variables
- Commentez toutes les branches conditionnelles
- Commentez toutes les boucles

- Choisissez des noms de procédures et de variables avec soins : leur nom doit être explicite. Microsoft propose quelques règles :

Routines
Utilisez la casse Pascal (CalculTotal) pour les noms de routine (la première lettre de chaque mot est une majuscule).

Évitez d'employer des noms difficiles pouvant être interprétés de manière subjective, notamment Analyse() pour une routine ou YYB8 pour une variable.

Dans les objets, il ne faut pas inclure des noms de classe dans les noms de propriétés Patient.PatientNom est inutile, utiliser plutôt Patient.Nom.

Utilisez les verbe/nom pour une routine : CalculTotal().

Variables
Pour les noms de variables, utilisez la casse selon laquelle la première lettre des mots est une majuscule, sauf pour le premier mot (iNombrePatient); noter ici que la première lettre indique le type de la variable (Integer), elle peut aussi indiquer la portée (gTotal pour une variable globale).

Ajoutez des méthodes de calcul (Min, Max, Total) à la fin d'un nom de variable, si nécessaire. Les noms de variable booléenne doivent contenir Is qui implique les valeurs True/False, par

exemple fileIsFound.

Évitez d'utiliser des termes tels que Flag lorsque vous nommez des variables d'état, qui différent des variables booléennes car elles acceptent plus de deux valeurs. Plutôt que documentFlag, utilisez un nom plus descriptif tel que documentFormatType.
Même pour une variable à courte durée de vie utilisez un nom significatif. Utilisez des noms de variable d'une seule lettre, par exemple i ou j, pour les index de petite boucle uniquement.

N'utilisez pas des nombres ou des chaînes littérales telles que For i = 1 To 7. Utilisez plutôt des constantes par exemple For i = 1 To DAYSINWEEK, pour simplifier la maintenance et la compréhension.

Tables
Pour les tables, utilisez le singulier. Par exemple, utilisez table 'Patient' plutôt que 'Patients'.
N'incorporez pas le type de données dans le nom d'une colonne.

Divers
Minimisez l'utilisation d'abréviations.
Lorsque vous nommez des fonctions, insérez une description de la valeur retournée, notamment GetCurrentWindowDirectory().

Évitez de réutiliser des noms identiques pour divers éléments.
Évitez l'utilisation d'homonymes et des mots qui entraînent souvent des fautes d'orthographe.
Évitez d'utiliser des signes typographiques pour identifier des types de données, notamment $ pour les chaînes ou % pour les entiers.

Un nom doit indiquer la signification plutôt que la méthode.

- Eclaircir, aérer le code:

Eviter plusieurs instructions par ligne.
Ajouter quelques lignes blanches.
Décaler à droite le code contenu dans une boucle ou une section If... End If :

Une mise en retrait simplifie la lecture du code, par exemple :

```
If ... Then
        If ... Then
                ...
        Else
                ...
        End If
Else
...
End If
```

Forcer la déclaration des variables et les conversions explicites

Option Explicit étant par défaut à On, toute variable utilisée doit être déclarée. Conserver cette option. Cela évite les erreurs liées aux variables mal orthographiées.

Si Option Strict est sur On, seules les conversions de type effectuées explicitement sur les variables seront autorisées. Le mettre sur On.

Voir la leçon 1.7 à ce sujet

Utilisez des constantes ou des énumérations

L'usage de constantes facilite les modifications.

Exemple : un programme gère des utilisateurs, faire :

Créer une constante contenant le nombre maximum d'utilisateurs.

```
Const NombreUtilisateur= 20
Dim VariableUtilisateur (NombreUtilisateur)        'on utilise NombreUtilisateur et non 20
For i = 0 To NombreUtilisateur-1
Next i
```

Plutôt que :

```
Dim VariableUtilisateur (20)
For i = 0 To 19
Next i
```

Si ultérieurement on veut augmenter le nombre d'utilisateurs possibles à 50, il suffit de changer une seule ligne :

```
Const NombreUtilisateur= 50
```

Utiliser les constantes VB, c'est plus lisible :

```
Form1.BorderStyle=2 'est à éviter
Form1.BorderStyle= vbSizable 'c'est mieux
```

Vérifier la validité des paramètres que reçoit une Sub ou Function

Vous pouvez être optimiste et ne pas tester les paramètres reçus par votre Sub. Les paramètres envoyés seront toujours probablement bons!! Bof un jour vous ferez une erreur, ou un autre n'aura pas compris le type de paramètre à envoyer et cela plantera !!!

Donc, il faut vérifier la validité des paramètres.

On peut le faire au fur et à mesure de leur utilisation dans le code, il est préférable de faire toutes les vérifications en début de Sub.

Se méfier du passage de paramètres 'par valeur' ou par 'référence'

Par défaut les paramètres sont envoyés 'par valeur' vers une procédure. Aussi, si la variable contenant le paramètre est modifiée, cela ne modifie pas la valeur de la variable de la procédure appelante.

Si on a peur de se tromper utilisons 'ByVal' et 'ByRef' dans l'en-tête de la Sub ou de la Fonction.

Les Booléens sont des Booléens

Utiliser une variable Integer pour stocker un Flag dont la valeur ne peut être que 'vrai' ou 'faux' et donner la valeur 0 ou -1 est à proscrire.

Faire :

```
Dim Flag As Boolean
Flag=True
```

(Utiliser uniquement True et False)

Eviter aussi d'abréger à la mode Booléens ce qui n'en est pas.

```
Dim x,y As Integer
If x And y then
```

(pour tester si x et y sont = 0) est à éviter.

Faire plutôt :

```
If x<>0 And y <>0
```

Utilisez les variables Date pour stocker les dates

Ne pas utiliser de type Double.

```
Dim LaDate As Date
LaDate=Now
```

Ne faire aucune confiance à l'utilisateur du logiciel

Si vous demandez à l'utilisateur de saisir un entier entre 1 et 7.

Vérifiez :

- qu'il a tapé quelque chose!!
- Qu'il a tapé une valeur numérique.
- Que c'est un entier.
- Que c'est supérieur à 0 et inférieur à 8.

Accorder les moindres privilèges :
Ne permettre de saisir que ce qui est nécessaire de saisir.

7.3 Optimiser en vitesse

VB.NET est t-il rapide ?

Comment VB.NET est situé en comparaison avec les autres langages de programmation ?

Le site OsNews.com publie les résultats d'un petit benchmark comparant les performances d'exécution sous Windows de plusieurs langages de programmation.

Les langages .NET - et donc le code managé en général - n'ont pas à rougir devant Java, pas plus que face au langage C compilé grâce à GCC. Voici un aperçu des résultats chiffrés (valeurs les plus faibles = les meilleures performances) :

	int	long	double	trig	I/O	TOTAL
Visual C++	9.6	18.8	6.4	3.5	10.5	48.8
Visual C#	9.7	23.9	17.7	4.1	9.9	65.3
gcc C	9.8	28.8	9.5	14.9	10.0	73.0
Visual Basic	9.8	23.7	17.7	4.1	30.7	85.9
Visual J#	9.6	23.9	17.5	4.2	35.1	90.4
Java 1.3.1	14.5	29.6	19.0	22.1	12.3	97.6
Java 1.4.2	9.3	20.2	6.5	57.1	10.1	103.1
Python/Psyco	29.7	615.4	100.4	13.1	10.5	769.1
Python	322.4	891.9	405.7	47.1	11.9	1679.0

Lire l'article complet à l'adresse : http://www.osnews.com/story.php?news_id=5602 - Nine Language Performance Round-up: Benchmarking Math & File I/O [OsNews.com]

Article publié également sur www.DotNet-fr.org

VB.NET est il plus rapide que VB6 ?

Exemple No 1 :

Sur une même machine P4 2.4 G faisons tourner un même programme: 2 boucles imbriquées contenant une multiplication, l'addition à un sinus et l'affichage dans un label :

En Vb.Net:

```
Imports System.Math
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
Dim i As Integer
Dim j As Integer
Dim k As Integer
For i = 0 To 100
For j = 0 To 1000
Label5.Text = (k * 2 + Sin(4)).ToString : Label5.Refresh()
k = k + 1
Next
Next
End Sub
```

35 secondes dans l'IDE, 25 secondes avec un exécutable après compilation.
En utilisant des 'Integer' ou des 'Long', il y a peu de différence.

En VB6

```
Private Sub Command1_Click()
Dim i As Long
Dim j As Long
Dim k As Long
For i = 0 To 100
For j = 0 To 1000
Label1.Caption = Str(k * 2 + Sin(4)): Label1.Refresh
k = k + 1
Next
Next
End Sub
```

9 secondes dans l'IDE , 7 secondes avec un exécutable après compilation.
Dur, dur 25 s pour VB.NET, 7 s pour VB6.

Exemple No 2 :
Sur une même machine P4 2.4 G faisons tourner un même programme: On crée un tableau de 10000 String dans lequel on met des chiffres Puis on trie le tableau.

En Vb.Net

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
Dim i As Integer
Dim A(10000) As String
For i = 9999 To 0 Step -1
        A(i) = (9999 - i).ToString
Next i
Array.Sort(A)
Label1.Text = "ok"
End Sub
```

< 1 seconde

En VB6

```
Private Sub Command1_Click()

Dim i As Integer
Dim A(10000) As String
Dim j As Integer
Dim N As Integer
Dim Temp As String
N = 9999
'remplir le tableau
For i = 9999 To 0 Step -1
        A(i) = Str(9999-i)
Next i

'trier
For i = 0 To N – 1
        For j = 0 To N - i – 1

                If A(j) > A(j + 1) Then
                        Temp = A(j): A(j) = A(j + 1): A(j + 1) = Temp
                End If

        Next j
```

```
        Next i
        End Sub
```

35 secondes

Moins d'une seconde avec VB.NET, 35 secondes en VB6.
La méthode 'Sort' est hyper plus rapide que la routine de tri !!!

En conclusion :
La couche du Framework semble ralentir considérablement la vitesse du code.

Mais, en VB.net, il faut raisonner différemment et utiliser judicieusement les classes et les méthodes au lieu de taper de longues routines.

Cela fait que en VB.Net :

- Le code est plus court et compact (moins de temps de développement)
- Le code est plus rapide.

Comment accélérer une application VB.NET ?

Utilisation des nouvelles fonctionnalités

Il faut raisonner différemment et utiliser judicieusement les classes et les méthodes au lieu de taper de longues routines.

Exemple :
On l'a vu plus haut La méthode 'Sort' d'un tableau est hyper plus rapide que la routine de tri écrite en code.

Choix des variables

Sur les ordinateurs actuels :
Pour les entiers les Integer sont les plus rapides car le processeur calcul en Integer . Viennent ensuite les Long, Short, et Byte.

Dans les nombres en virgule flottante, les Double sont les plus rapides car le processeur à virgule flottante calcul en Double, ensuite se sont les Single puis les Decimal.

Si c'est possible utiliser les entiers plutôt que les nombres en virgules flottantes.

Exemple pour stocker les dimensions d'une image, on utilisera les pixels: l'image aura un nombre entier de pixels et on peut ainsi utiliser une variable Integer, alors que si on utilise les centimètres on devra travailler sur des fractionnaires donc utiliser par exemple des Singles.

L'usage de constantes est plus rapide que l'usage de variable, car la valeur d'une constante est directement compilée dans le code.

Pour stocker une valeur, une variable est plus rapide qu'une propriété d'objet.

Tableau

Le CLR est optimisé pour les tableaux unidimensionnels.
L'usage des tableaux de tableau 'A(9),(9)' est plus rapide que les tableaux multidimensionnels 'A(9,9)'.

Pour rechercher un élément dans un ensemble l'élément à partir de son index, utilisez un tableau (l'accès à un élément d'index i est plus rapide dans un tableau que dans une collection)

Collections

Si on ne connaît pas le nombre d'éléments maximum et que l'on doit ajouter, enlever des éléments, il vaut mieux utiliser une collection (ListArray) plutôt qu'un tableau avec des Dim Redim Preserve. Mais attention une collection est composée d'objet, ce que est lent.

Pour rechercher un élément dans un ensemble l'élément à partir d'une clé (KeyIndex), utilisez une collection (l'accès à un élément ayant la clé X est plus rapide dans une collection que dans un tableau; dans une tableau il faut en plus écrire la routine)

Eviter la déclaration de variables 'Objet' et les liaisons tardives

Eviter de créer des variables Objet :
Pour créer une variable et y mettre une String:

```
Dim A 'crée un 'Objet' A
```

Il est préférable d'utiliser :

```
Dim A As String
```

La gestion des objets est plus lente que la gestion d'une variable typée.

Il faut aussi éviter les liaisons tardives : Une liaison tardive consiste à utiliser une variable Objet et à l'exécution, donc tardivement, lui assigner une String ou un Objet ListBox par exemple. Dans ce cas, à l'exécution, VB doit analyser de quel type d'objet il s'agit et le traiter, alors que si la variable a été déclarée d'emblée comme une String ou une ListBox, VB a déjà prévu le code nécessaire en fonction du type de variable. Utilisez donc des variables typées.

Utilisez les bonnes 'Option'

```
Option Strict On 'permet de convertir les variables de manière explicite et accélère le code.
Option Compare Binary 'accélère les comparaisons et les tris (la comparaison binaire consiste à comparer les codes unicode des chaînes de caractère).
```

Pour les fichiers utilisez System.IO

L'utilisation des System.IO classes accélère les opérations sur fichiers (en effet, les autres manières de lire ou d'écrire dans des fichiers comme les FileOpen font appel à System.IO : autant l'appeler directement!!) :

- Path, Directory, et File
- FileStream pour lire ou écrire
- BinaryReader and BinaryWriter pour les fichiers binaires
- StreamReader and StreamWriter pour les fichiers texte

Utiliser des buffers entre 8 et 64K

Opérations

Si possible :
Utiliser :"\"

Pour faire une vraie division on utilise l'opérateur '/'

Si on a seulement besoin du quotient d'une division (et pas du reste ou du résultat fractionnaire) on utilise '\', c'est beaucoup plus rapide.

Utiliser :"+="
A+= 2 est plus rapide que A= A+2

Utiliser : AndAlso et ElseOr
AndAlso et ElseOr sont plus rapide que And et Or.
(Puisque la seconde expression n'est évaluée que si nécessaire)

Utiliser :With End With
With.. End With accélère le code:

```
With Form1.TextBox1
        .BackColor= Red
        .Text="BoBo"
        .Visible= True
End With
```

Est plus rapide que :

```
Form1.TextBox1.BackColor= Red
Form1.TextBox1.Text="BoBo"
Form1.TextBox1.Visible= True
```

Car Form1.TextBox1 est 'évalué' 3 fois au lieu de 1 fois.

En mettre le moins possible dans les boucles

Soit un tableau J (100,100) d'entiers :
Soit un calcul répété 100 000 fois sur un élément du tableau, par exemple :

```
For i=1 to 100000
        R=i*J(1,2)
next i
```

On va 100000 fois chercher un élément d'un tableau, c'est toujours le même !

Pour accélérer la routine (c'est plus rapide de récupérer la valeur d'une variable simple plutôt d'un élément de tableau), on utilise une variable intermédiaire P :

```
Dim P as integer
P=J(1,2)
For i=1 to 100000
        R=i*P
next i
```

C'est plus rapide.

De la même manière si on utilise une propriété (toujours la même) dans une boucle, on peut stocker sa valeur dans une variable car l'accès à une variable simple est plus rapide que l'accès à une propriété.

Eviter aussi les Try Catch dans des grandes boucles.

Comment accélérer quand on utilise des 'String'

Exemple d'une opération coûteuse en temps :

```
Dim s As String = "bonjour";
```

```
s += "mon" + "ami";
```

En réalité le Framework va créer 3 chaînes en mémoires avec toutes les pertes en mémoire et en temps que cela implique.

Pour effectuer des opérations répétées sur les string, le framework dispose donc d'une classe spécialement conçue et optimisée pour ça : System.Text.StringBuilder.

Pour l'utiliser, rien de plus simple

```
System.Text.StringBuilder sb = new System.Text.StringBuilder();
sb.Append("bonjour");
sb.Append("mon ami");
string s = sb.ToString();
```

La méthode ToString de la classe StringBuilder renvoi la chaîne qu'utilise en interne l'instance de StringBuilder.

Comment accélérer l'affichage ?

Formater le plus vite possible :
Pour mettre en forme des nombres et les afficher Format est puissant, mais si on peut utiliser ToString c'est plus rapide (ToString est aussi plus rapide que Cstr).
ChrW utilisé pour afficher un caractère (et AscW) sont plus rapide que Chr et Asc car il travaille directement sur les Unicodes.

Précharger les fenêtres et les données.
Quand une fenêtre en ouvre une autre, le temps de chargement est long, l'utilisateur attend!

Solution :
En début de programme précharger les fenêtres en les rendant invisible. Lors de l'utilisation de ces fenêtres il suffira de les rendre visible, ce qui est plus rapide que de les charger.
Certaines données (liste...) doivent être chargées une fois pour toute, le faire en début de programme, lors de l'affichage de la fenêtre 'Splach' par exemple (Celle qui contient une belle image et qui s'ouvre en premier)

Afficher les modifications en une fois:
A chaque fois que l'on fait une modification de propriété (couleur, taille..) ou de contenu (texte dans un TextBox) Vb affiche chaque modification. Si on modifie tout et que l'on re-affiche tout cela va plus vite.

Rendre l'objet inactif, faire toutes les modifications puis réactiver.
Pour le cas du TextBox ne pas faire.

```
TextBox1.Text = TextBox1.Text + "Bonjour"
TextBox1.Text = TextBox1.Text + ""Monsieur"
```

Faire :

```
Dim T as string
T = "Bonjour"
T &= "Monsieur"
TextBox1.Text = T
```

Le texte est affiché en une fois.

Afficher en 2 fois :
A l'inverse pour ne pas faire attendre un affichage très long, afficher le début (l'utilisateur voit apparaître quelque chose à lire) il est occupé un temps, ce qui permet d'afficher le reste.
Exemple : remplir une listBox avec un grand nombre d'éléments long à préparer: en afficher 5

rapidement puis calculer et afficher les autres. L'utilisateur à l'impression que la ListBox se rempli immédiatement.

Pour faire patienter l'utilisateur lors d'une routine qui dure longtemps ? (Et lui montrer que l'application n'est pas bloquée) :

- Transformer le curseur en sablier en début de routine, remettre un curseur normal en fin de routine.
- Utiliser une ProgressBar (pour les chargements long par exemple)

Ce qui n’influence pas la rapidité du code

Les boucles For , Do ,While ont toutes une vitesse identique.

7.4 Chronométrer le code

Je veux comparer 2 routines et savoir laquelle est la plus rapide.

Pour chronométrer un évènement long

Entendons par évènement long, plusieurs secondes ou minutes.

Pas de problème, 2 solutions :

- On utilise un Timer, (dans l'évènement Ticks qui survient toutes les secondes, une variable s'incrémente comptant les secondes). (Partie 4.5 du cours).
- On peut utiliser l'heure Système.

```
Dim Debut, Fin As DateTime
Dim Durée As TimeSpan
Debut=Now
...Routine...
Fin=Now
Durée=Fin-Debut
```

Créer un compteur pour les temps très courts

C'est le cas pour chronométrer des routines dont la durée bien inférieure à une seconde.
Cela semblait à première vue facile!!!

J'ai en premier lieu utilisé un Timer, (dans l'évènement Ticks un compteur de temps s'incrémente) mais les intervalles de déclenchement semblent long et aléatoire

J'ai ensuite utilisé l'heure système :
Mais 'Durée' est toujours égal au 0 pour les routines rapides car il semble que Now ne retourne pas les millisecondes ou les Ticks.

J'ai trouvé la solution chez Microsoft :
Utilisation d'une routine de Kernel32 qui retourne la valeur d'un compteur (QueryPerformanceCounter). QueryPerformanceFrequency retourne le nombre de fois que le compteur tourne par seconde.

Exemple :
Comparer 2 boucles, l'une contenant une affectation de variable tableau (b=a(2)) l'autre une affectation de variable simple (b=c), on gagne 33%.

```
Declare Function QueryPerformanceCounter Lib "Kernel32" (ByRef X As Long) As Short
Declare Function QueryPerformanceFrequency Lib "Kernel32" (ByRef X As Long) As Short

Private Sub ButtonGo_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ButtonGo.Click
Dim debut As Long
Dim fin As Long
Dim i As Long
Dim a(5) As String
Dim b As String
Dim c As String
Dim d1 As Long
Dim d2 As Long
```

```
'**********première routine
QueryPerformanceCounter(debut)
For i = 0 To 10000
b = a(2) Next
QueryPerformanceCounter(fin)
d1 = fin - debut
Label1.Text = d1.ToString

'**********seconde routine
QueryPerformanceCounter(debut)
c = a(2)
For i = 0 To 10000
b = c
Next
QueryPerformanceCounter(fin)
d2 = fin - debut
Label2.Text = d2.ToString

Label5.Text = "Gain 2eme routine:" & 100 - Int(d2 / d1 * 100).ToString
End Sub
```

C'est cette routine qui est utilisée pour étudier l'optimisation du code.

Elle n'est pas parfaite, car sujette à variation : les valeurs sont différentes d'un essai à l'autre en fonction des processus en cours !

Y a-t-il mieux ?

SECTION 8 : Allons plus loin

8.1 Allons plus loin avec les procédures

On savait que les procédures pouvaient être Public ou Privée.

En fait une procédure peut être :

Public
Les procédures déclarées avec le mot clé Public ont un accès public. Il n'existe aucune restriction quant à l'accessibilité des procédures publiques.

Protected
Dans un module de classe, les procédures déclarées avec le mot clé Protected ont un accès protégé. Elles sont accessibles seulement à partir de leur propre classe ou d'une classe dérivée.

Friend
Les procédures déclarées avec le mot clé Friend ont un accès ami.
Elles sont accessibles à partir du programme contenant leur déclaration et à partir de n'importe quel autre endroit du même assembly.

Protected Friend
Les procédures déclarées avec les mots clés Protected Friend ont l'union des accès ami et protégé.Elles peuvent être utilisées par du code dans le même assembly, de même que dans les classes dérivées.
L'accès Protected Friend peut être spécifié uniquement pour les membres des classes.

Private
Les procédures déclarées avec le mot clé Private ont un accès privé.
Elles ne sont accessibles qu'à partir de leur contexte de déclaration, y compris à partir des membres de types imbriqués, tels que des procédures.

8.2 Comprendre le code créé par Visual Basic

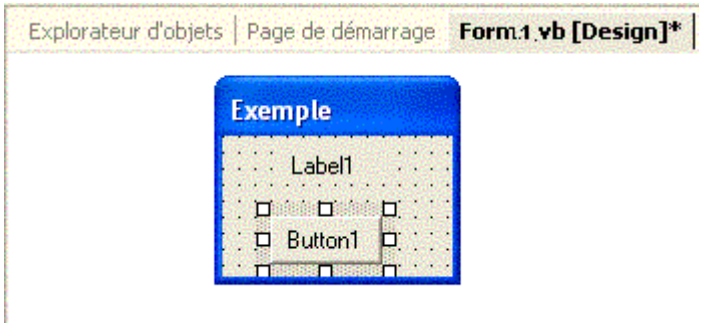
Comprendre le code généré automatiquement par Vb quand on crée une formulaire ou un contrôle.

Code généré automatiquement lors de la création d'un formulaire ou d'un contrôle

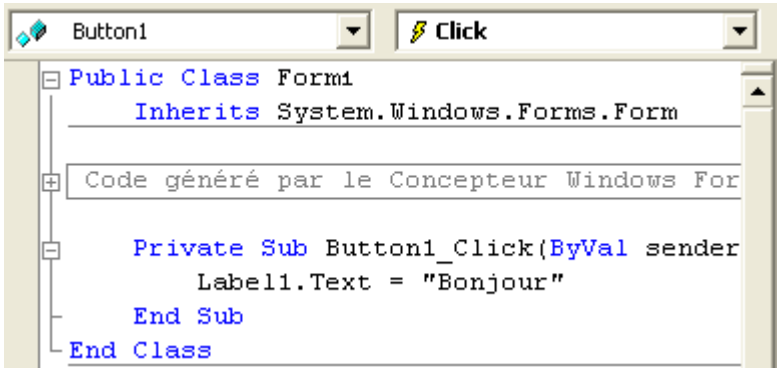
Une application 'Windows Forms' est principalement constituée de formulaires (ou fenêtre), de contrôles et de leurs événements.

Effectivement, pendant la création de l'interface utilisateur de votre application, vous créez généralement une fenêtre contenant des contrôles et des procédures évènements.

Quand vous créer un nouveau projet 'Windows Forms' cela dessine un formulaire, une fenêtre vide et le code correspondant, Ajoutons y un bouton cela donne l'interface utilisateur suivante :



Comme on l'a vu, VB crée le code correspondant et dans ce code une Classe correspondant à la fenêtre, cette classe dérive de la Classe Form.



(On rappelle que la véritable fenêtre, l'objet sera instancé à partir de cette classe)

Décortiquons le code :
Vb crée une Class nommé Form1, elle est public (accessible partout)

```
Public Class Form1
```

Cette Classe hérite des propriétés de la Classe Form (celle ci est fournis par le Frameworks)

```
Inherits System.Windows.Forms.Form
```

Ensuite il y a une région (partie du code que l'on peut 'contracter' et ne pas voir ou 'dérouler', cette région contient :" Le Code généré (automatiquement) par le Concepteur Windows Form ", si on le déroule en cliquant sur le '+'.

On voit :

- Le constructeur de la fenêtre: la routine Sub New

MyBase fait référence à la classe de base de l'instance en cours d'une classe,
MyBase.New 'construit' la Classe

- Le destructeur de la fenêtre : la routine Sub Dispose

- Le créateur des contrôles de la fenêtre par la procédure Sub InitializeComponent

Elle est nécessaire pour créer les contrôles et définir les propriétés de ces contrôles.

Exemple : création d'un label Me.Label1= NewSystem.Windows.forms.Label
Modification d'une propriété : Me.Label.Text="Hello"

Elle définie aussi les propriété du formulaire :

```
Me.Name = "Form1"
```

Exemple d'un formulaire vide nommé Form1 :

```
Public Class Form1
Inherits System.Windows.Forms.Form
#Region " Code généré par le Concepteur Windows Form
Public Sub New()
MyBase.New()

'Cet appel est requis par le Concepteur Windows Form.
InitializeComponent()

'Ajoutez une initialisation quelconque après l'appel InitializeComponent()
End Sub

'La méthode substituée Dispose du formulaire pour nettoyer la liste des composants.
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
If disposing Then
If Not (components Is Nothing) Then
components.Dispose()
End If
End If
MyBase.Dispose(disposing)
End Sub
'Requis par le Concepteur Windows Form
Private components As System.ComponentModel.IContainer

'REMARQUE : la procédure suivante est requise par le Concepteur Windows Form
'Elle peut être modifiée en utilisant le Concepteur Windows Form.
'Ne la modifiez pas en utilisant l'éditeur de code.
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
'
'Form1
'
Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
Me.ClientSize = New System.Drawing.Size(292, 266)
Me.Name = "Form1"
Me.Text = "Form1"
End Sub
#End Region

End Class
```

Si dans la fenêtre Design on ajoute un bouton Button1 cela ajoute le code :
Cette ligne contenant WithEvents indique qu'il y a une gestion d'évènement sur les boutons.

```
Friend WithEvents Button1 As System.Windows.Forms.Button
```

Cette ligne créée le bouton

```
Me.Button1 = New System.Windows.Forms.Button
```

Cette ligne le positionne

```
Me.Button1.Location = New System.Drawing.Point(56, 144)
```

Cette ligne lui donne un nom

```
Me.Button1.Name = "Button1"
```

Cette ligne détermine sa taille

```
Me.Button1.Size = New System.Drawing.Size(104, 24)
```

Cette ligne indique ce qui est affiché sur le bouton

```
Me.Button1.Text = "Button1"
```

Cela donne :

```
Private components As System.ComponentModel.IContainer
Friend WithEvents Button1 As System.Windows.Forms.Button
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
Me.Button1 = New System.Windows.Forms.Button
Me.SuspendLayout()
'
'Button1
'
Me.Button1.Location = New System.Drawing.Point(56, 144)
Me.Button1.Name = "Button1"
Me.Button1.Size = New System.Drawing.Size(104, 24)
Me.Button1.TabIndex = 0
Me.Button1.Text = "Button1"
```

Les procédures évènements correspondant au bouton sont automatiquement créées :

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

On constate qu'il y a une liaison entre la fenêtre Design et le code généré; on pourrait modifier dans le code l'interface utilisateur. C'est déconseillé d'aller trafiquer dans cette zone de "Code généré par le Concepteur Windows Form", il faut mieux faire des modifications dans la partie design et dans la fenêtre de propriété.

Substitution de procédures évènement

Il est possible de substituer une méthode (utiliser sa propre méthode à la place de la méthode normale qui existe normalement dans un contrôle)

Exemple créer un contrôle simple affichant toujours 'Bonjour' :
Il faut créer une classe héritant des 'Control', détourner sont évènement OnPaint (avec Overides) qui survient quand le contrôle se dessine pour simplement afficher 'Bonjour'

```
Public Class ControleAffichantBonjour
Inherits Control
Overrides Protected Sub OnPaint ( e As PaintEventArgs )
        e.Graphics.DrawString ("Bonjour", Font, nex SolidBrush(ForeColor)
End Sub
End Class
```

Cet exemple ne sert strictement à rien!! Pour une fois !!!
Il est aussi possible de détourner des évènements.

Dans le chapitre 4.11 'Impression' il y a un bel exemple de création de "lien" entre un objet printdocument et la routine évènement PrintPage (imprimer hello avec un printdocument)

Dans le chapitre suivant on va utiliser ces connaissances pour, dans le code, créer soi-même des contrôles et leurs évènements.

8.3 Créer des contrôles par code

Dans le code, on peut créer soi-même de toutes pièces, des contrôles et leurs évènements.

Créer des contrôles par code

Dans le code d'une procédure, il est possible de créer de toute pièce un contrôle, mais attention, il faut tout faire !!!

Créons le bouton.

```
Dim Button1 = New Button
```

Modifions ses propriétés :

```
Me.Button1.Location = New System.Drawing.Point(56, 144)
Me.Button1.Name = "Button1"
Me.Button1.Size = New System.Drawing.Size(104, 24)
Me.Button1.TabIndex = 0
Me.Button1.Text = "Button1"
```

Le bouton existe mais il faut l'ajouter à la collection Controls de la fenêtre (Cette collection contient tous les contrôles contenus dans la fenêtre) :

```
Me.Controls.Add(Button1)
```

Le bouton existe mais pour le moment, il ne gère pas les évènements.

Il faut inscrire le bouton dans une méthode de gestion d'évènements. En d'autres termes, Vb doit savoir quelle procédure évènement doit être déclenchées quand un évènement survient.

Pour cela, il y a 2 méthodes :

- Déclarer la variable avec le mot clé WithEvents ce qui permet ensuite d'utiliser le Handles du contrôle dans la déclaration d'une Sub

Déclaration dans la partie déclaration du module (en haut) (WithEvents n'est pas accepté dans une procédure) :

```
Private WithEvents Button1 As Button1
```

Remarque Button1 est accessible dans la totalité du module.
Puis écrire la sub évènement.

```
Sub OnClique ( sender As Objet, EvArg As EventArgs) Handles Button1.Click
End Sub
```

Ainsi VB sait que pour l'évènement Button1.Click, il faut déclencher la Sub OnClique.

Remarque : il pourrait y avoir plusieurs Handles sur une même sub, donc des évènements différents sur des objets différents déclenchant la même procédure.

- Utiliser AddHandler

Déclaration (possible dans une procédure) :

```
Dim Button1 As Button
```

Puis écrire la gestion de l'évènement. (L'évènement Button1.click doit déclencher la procédure dont l'adresse est BouttonClique)

```
AddHandler Button1.Click AddressOf  BouttonClique
```

Enfin on écrit la sub qui 'récupère ' l'évènement :

```
Private Sub BouttonClique (sender As Objet, evArgs As EventArgs)
End Sub
```

Ainsi VB sait que pour un évènement du Button1, il faut déclencher la Sub ButtonClique

Exemple avec AddHandler :

Créons un TextBox nommé TB et une procédure déclenchée par KeyUp de ce TextBox :

Dans une procédure (Button1_Click par exemple) : Je crée un TextBox nommé TB, je le positionne, je met dedans le texte 'ici une textbox'. Je l'ajoute aux Contrôles du formulaire.

Grâce à 'AddHandler', je lie l'évènement Keyup de cet objet TB à la sub que j'ai créée : TextboxKeyup.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
Dim TB As New System.Windows.Forms.TextBox
TB.Location = New System.Drawing.Point(2, 2)
TB.Text = "ici une textBox"
Me.Controls.Add(TB)
AddHandler TB.Keyup, AddressOf TextboxKeyup.
End sub

Sub TextboxKeyup.(ByVal sender As Object, ByVal e As KeyEventArgs)
...
End Sub
```

Si je crée un autre bouton TB2, j'ajoute de la même manière AddHandler TB2.Click, AddressOf TextboxKeyup2, ainsi chaque évènement de chaque contrôle à ses propres routines évènement et en cliquant sur le bouton TB2 on déclenche bien TextboxKeyup2.

Attention, la procédure TextboxKeyup doit recevoir impérativement les bons paramètres : un objet et un KeyEventArgs car ce sont les paramètres retournés par un KeyUp.

Autre exemple avec AddHandler mais avec 2 boutons :

Il est possible de créer plusieurs contrôles ayant la même procédure évènement:

Créons 2 boutons (BT1 et BT2) déclenchant une seule et même procédure (BoutonClique).

Dans ce cas, comment savoir sur quel bouton l'utilisateur à cliqué ?
En tête du module déclarons les boutons (Ils sont public):

```
Public BT1 As New System.Windows.Forms.Button
Public BT2 As New System.Windows.Forms.Button
```

Indiquons dans form_load par exemple la routine évènement commune (BoutonClique) grâce à AddHandler.

```
Form_Load
BT1.Location = New System.Drawing.Point(2, 2)
BT1.Text = "Bouton 1"
Me.Controls.Add(BT1)
BT2.Location = New System.Drawing.Point(100, 100)
BT2.Text = "Bouton 2"
Me.Controls.Add(BT2)
AddHandler BT1.Click, AddressOf BoutonClique
AddHandler BT2.Click, AddressOf BoutonClique
End Sub
```

Si c'est le bouton 1 qui a été cliqué, afficher "button1" dans une TextBox :

```
Sub BoutonClique(ByVal sender As Object, ByVal e As EventArgs)
If sender Is BT1 Then
        TextBox1.Text = "button 1"
ElseIf sender Is BT2 Then
        TextBox1.Text = "button 2"
End If
End Sub
```

La ruse est que déterminer quel objet (quel bouton) à déclenché l'évènement, pour cela on utilise le premier paramètre, le sender :

```
If sender Is BT1 Then        `Si le sender est le bouton1…
```

Les délégués

Pour la petite histoire, nous créons un délégué à chaque fois que nous créons une procédure gestionnaire d'évènement avec le mot Handles ou avec AddHandler.

En C on utilise des pointeurs de fonction, adresse en mémoire indiquant ou le logiciel doit sauter quand on appelle une fonction ou un évènement. En VB on parle de délégué.