L'interface Utilisateur

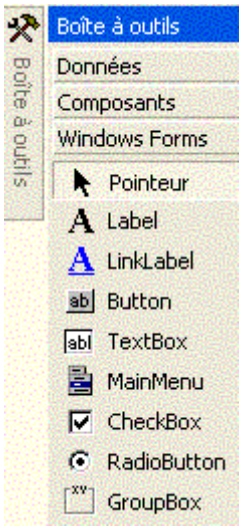
3.1 L'interface Utilisateur

Elle correspond aux fenêtres et contrôles que voit l'utilisateur.

On a vu que le développeur **dessine** cette interface en mode conception (Design) dans l'IDE.

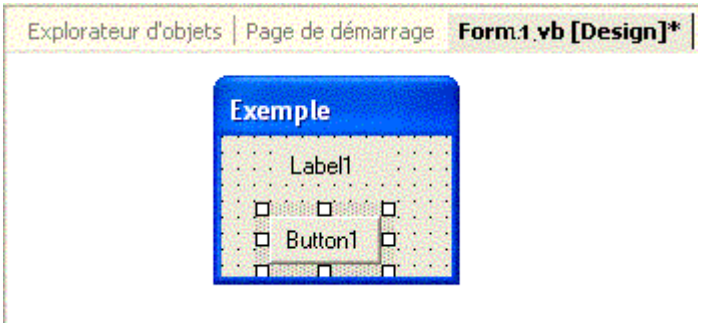
Rappel :
Comment créer une fenêtre ?
Menu **Projet**, **Ajouter un formulaire Windows**, cliquer sur **WindowsForm**, une fenêtre 'Form1' apparaît. On a bien crée une fenêtre avec la classe WindowsForms.Form (En fait on a crée une Classe 'Form1').

Comment ajouter un bouton ?
Cliquer sur '**Boite à Outils**' à gauche , bouton WindowsForms, puis bouton '**Button**',cliquer dans Form1, déplacer le curseur sans lâcher le bouton, puis lâcher le bouton : un bouton apparaît.



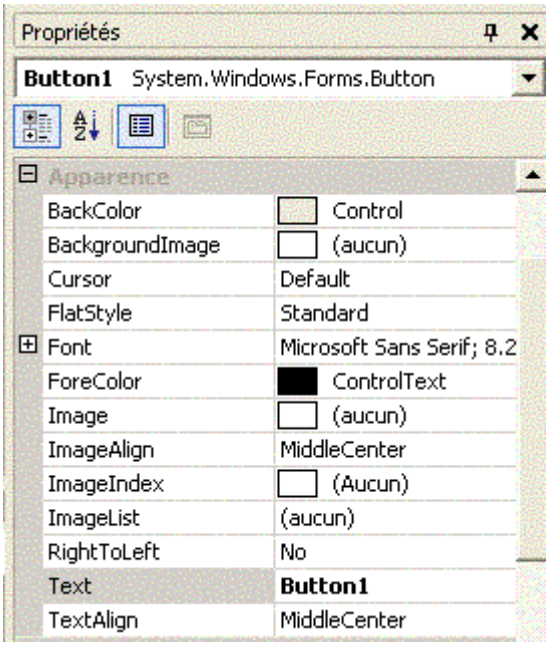
Comment ajouter un label ?
Un label est un contrôle qui permet d'afficher un texte.
Comme pour le bouton cliquer sur 'Boite à Outils' à gauche, bouton WindowsForms, bouton 'Label' et mettre un contrôle label sur la fenêtre.

On obtient dans la fenêtre principale :



Modifier les propriétés de l'objet.
Il suffit ensuite de modifier les propriétés de l'objet pointé (celui qui est entouré de petit carrés) pour lui donner l'aspect désiré. Les propriétés sont accessibles dans la **fenêtre de**

propriétés de droite.



Dans le code des procédures les propriétés des objets sont aussi accessibles.
Button1.Text="OK" 'permet par exemple de modifier la propriété **Text** d'un bouton.

Noter que **pour modifier la taille des objets**, on peut le faire très facilement à la souris en cliquant sur les petits carré entourant l'objet et en tirant les bords (On peut interdire les modifications de taille et de position des contrôles par le menu *Format* puis *verrouiller les contrôles* une fois que leurs tailles et positions est bien définies.*).*

Tous les objets ont des propriétés communes

Celles héritées de la Classe '**Control**' qu'il faut connaître:

Name : il s'agit du nom de l'objet tel qu'il est géré par l'application.

Par défaut, VB baptise tous les objets que vous créez de noms génériques, comme Form1, Form2, Form3 pour les fenêtres, List1, List2 pour les listes...
Accessible en mode conception uniquement.

Il est vivement conseillé, avant toute autre chose, de rebaptiser les objets que vous venez de créer afin de donner des noms plus évocateurs.

Le bouton sur lequel est écrit « OK » sera nommé BoutonOK.
La liste qui affiche les utilisateurs sera nommée ListUtilisateurs.

Il est conseillé de débuter le nom de l'objet par un mot évoquant sa nature :
BoutonOk ou BtOk ou ButtonOk, btnOk 'c'est comme vous voulez.

Microsoft conseille :

- Btn pour les Boutons
- Lst pour les ListBox
- Chk pour les CheckBox
- Cbo pour les combos
- Dlg pour les DialogBox
- Frm pour les Form
- Lbl pour les labels

- Txt pour les Textbox
- Tb pour les Toolsbar
- Rb pour les radiobutton
- Mm pour les menus
- Tmr pour les timers

Text : il s'agit du texte qui est associé à l'objet.
Dans le cas d'une fenêtre c'est le texte qui apparaît dans la barre de titre en haut.
Pour un TextBox ou un Label c'est évidement le texte qui est affiché.
On peut modifier cette propriété en mode conception ou dans le code :

Exemple : Avec du code comment faire pour que le bouton ButtonOk porte l'inscription 'Ok'

```
ButtonOk.Text= "Ok"
```

Enabled : accessible
Indique si un contrôle peut répondre à une interaction utilisateur.

La propriété Enabled permet l'activation ou la désactivation des contrôles au moment de l'exécution. Vous pouvez désactiver les contrôles ne s'appliquant pas à l'état actuel de l'application. Vous pouvez également désactiver un contrôle pour interdire son utilisation. Par exemple, un bouton peut être désactivé pour empêcher l'utilisateur de cliquer dessus. Si un contrôle est désactivé, il ne peut pas être sélectionné. Un contrôle désactivé est généralement gris.

Exemple : désactiver le ButtonOk

```
ButtonOk.Enabled=False
```

Visible :
Indique si un contrôle est visible ou non.

```
ButtonOk.Visible=False      'fait disparaître le bouton.
```

Attention pour rendre visible une fenêtre on utilisa la méthode .Show.

Font :
Permet le choix de la police de caractères affichée dans l'objet.

BackColor ForeColor :
Couleur du fond, Couleur de l'avant plan
Pour un bouton Forecolor correspond au cadre et aux caractères.

```
ButtonOk.ForeColor=  System.Drawing.Color.Blue
```

Tag :
Permet de stocker une valeur ou un texte lié à l'objet. Chaque objet a un Tag qui peut contenir du texte.
On l'utilise souvent comme un Flag lié à l'objet.

Par exemple: une liste peut contenir la liste des CD ou des DVD ou des K7, quand je charge la liste des CD, je rajoute List1.Tag="CD" cela permet ultérieurement de voir ce qu'il y a dans la liste.

Il y a bien d'autres propriétés.

Evènements liés aux objets

On a vu que les objets de l'interface utilisateur ont des **procédures déclenchées par les évènements de cet objet.**

2 exemples :
- Quand l'utilisateur clique sur un bouton Ok, la procédure ButtonOk_Click s'effectue.
- Quand l'état (coché ou non coché) d'une case à cocher nommée CouleurRouge change, la procédure CouleurRouge.CheckedChanged est activée.

La syntaxe **complète** de la procédure est:

```
Private Sub CouleurRougeCheckedChanges (ByVal sender As System.Objet, ByVal e
As System.EventArgs)  Handles CouleurRouge.CheckedChanged

End Sub
```

Détaillons :
La procédure évènement est **privée** (Private).

Après le nom Sub il y a **un nom de procédure** (CouleurRougeCheckedChanges)
Handles indique quel objet et évènement à déclenché la procédure. (On verra qu'il peut y en avoir plusieurs.)

A noter que **Sender** contient le contrôle ayant déclenché l'évènement et **e** l'évènement correspondant.
sender.Name 'contient par exemple le nom du contrôle ayant déclenché l'évènement. **On voit que quand on crée un objet, ses procédures évènements sont automatiquement crées.**

On se rend compte que dans une procédure évènement on peut modifier (en mode conception) ou lire (en mode Run) quel objet et quel évènement a déclenché la procédure. On peut même indiquer plusieurs objets liés à cette procédure.

Certains évènements sont communs à tous les contrôles :
- **Click**
- **DoubleClick**
- **GotFocus**
- **LostFocus**
- **KeyUp**
- **KeyPress**
- **KeyDown**

Il y a toujours des méthodes **Changed** déclenchées par un changement d'état : **CheckedChanged** pour une case à cocher, **TextChanged** pour un contrôle texte.

Pour ne pas alourdir les exemples, nous écrivons souvent une version simplifiée de l'en-tête de la procédure.

En résumé :

Le programmeur dessine les fenêtres et contrôles. Il
peut modifier les propriétés des objets dessinés :
- Par la **fenêtre de propriétés (en mode conception).**
- Par **du code (des instructions) dans les procédures.**

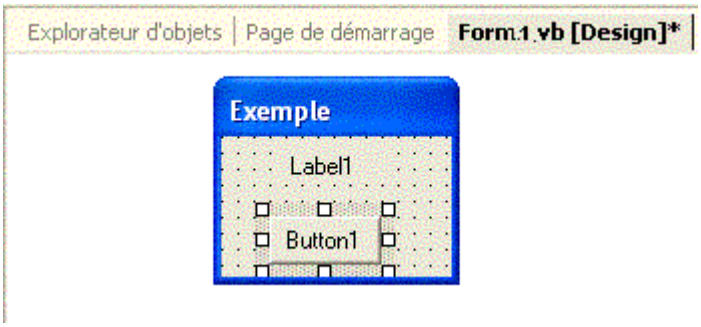
3.2 Les Forms

Elles correspondent aux fenêtres ou 'formulaires'.

Créer une fenêtre en mode conception :

Menu **Projet**, **Ajouter un formulaire Windows**, cliquer sur **WindowsForm**, une fenêtre 'Form1' apparaît. On a bien crée une fenêtre avec la classe WindowsForms.
Toute l'interface se trouve sur des fenêtres.

En VB.net on parle de **formulaire**.



Propriétés

Bien sur, la fenêtre possède des propriétés qui peuvent être modifiées en mode design dans la fenêtre 'Propriétés' à droite ou par du code:

Name : Nom du formulaire.
Donner un nom explicite, par exemple 'FrmDemarrage'

Dès qu'une fenêtre est créée on modifie immédiatement ses propriétés en mode conception pour lui donner l'aspect que l'on désire.

Text : C'est le texte qui apparaîtra dans la barre de titre en haut.
Text peut être modifié par le code : Form1.text= "Fenêtre"

Icon : propriété qui permet d'associer à la Form un fichier icône. Cette icône s'affiche dans la barre de titre, tout en haut à gauche. Si la Form est la Form par défaut du projet, c'est également cette icône qui symbolisera votre application dans Windows.

Comment créer une icône ?

Dans l'IDE de VB.
Menu Fichier>Nouveau>Fichier cliquez sur Icon, Vb ouvre une fenêtre Icon1 (dans l'éditeur d'images de Visual Studio.Net) Cela permet de créer ou modifier une icône (Fichier>Ouvrir>Fichier pour modifier).

Comment enregistrer ? Click droit dans l'onglet 'Icon1' ouvre un menu contextuel permettant d'enregistrer votre Icône.

WindowState :
Donne l'état de la fenêtre : Plein écran (FormWindowState.Maximized), normale (FormWindowState.Normal), dans la barre de tache (FormWindowState.Minimized).

Exemple : mettre une fenêtre en plein écran avec du code.

```
me.WindowState =FormWindowState.Maximized
```

(Quand on tape Me.WindowsState= Vb donne la liste, l'énumération)

ControlBox
Si cette propriété à comme valeur False, les boutons de contrôle situés à droite de la barre de la fenêtre n'apparaissent pas.

MaximizeBox
Si cette propriété à comme valeur False, le boutons de contrôle 'Plein écran' situés à droite de la barre de la fenêtre n'apparaît pas.

MinimizeBox
Si cette propriété à comme valeur False, le boutons de contrôle 'Minimize' situés à droite de la barre de la fenêtre n'apparaît pas.

FormBorderStyle
Permet de choisir **le type des bords de la fenêtre** : sans bord (None), bord simple (FixedSingle) ne permettant pas à l'utilisateur de modifier la taille de la fenêtre, bord permettant la modification de la taille de la fenêtre (Sizable)…

Exemple :

```
Me.FormBorderStyle =FormBorderStyle.Sizable
```

StartPosition :
Permet de choisir la position de la fenêtre lors de son ouverture.
Fenêtre au centre de l'écran ? À la position qui existait lors de la conception... ?

```
Me.StartPosition =FormStartPosition.CenterScreen
```

MinSize et MaxSize
Donne les dimensions minimums et maximums que l'on peut utiliser pour redimensionner une fenêtre.

Opacity
Allant de 0% à 100%, permet de créer un formulaire plus ou moins transparent.

Exemple:

```
Me.FormBorderStyle= Sizable
Me.ControlBox=False
```

Donne la fenêtre :

Ouvrir une fenêtre

On vient de dessiner une Form1 et une Form2 c'est donc les Class 'Form1 et 'Form2' (les moules) que l'on a dessiné.

Si dans une routine de la Form1 on veut **ouvrir une seconde fenêtre de type Form2**, il

faut :
Créer un Objet fenêtre (formulaire) avec le moule Form2 :

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

La nouvelle instance f de la Class 'form2' est un objet fenêtre.
Pour la faire apparaître j'utilise la méthode :.ShowDialog.

```
f.ShowDialog()
```

La fenêtre f est **modale car on a utilisé ShowDialog** : quand elle est ouverte, on ne peut pas aller dans une autre fenêtre de l'application avant de sortir de celle là. (A titre d'exemple les fenêtres MessageBox sont toujours Modales).

Utiliser .show pour ouvrir une feuille non modale.

Attention : une fenêtre est un objet et est 'visible' suivant les règles habituelles des objets.

Si on instance une fenêtre à partir d'une procédure, elle sera visible dans cette procédure. Si elle est 'Public' et instancée dans un module standard, elle sera visible partout.

Evènements

Au cours de l'exécution:
Quand la feuille est **chargée** la procédure **Form1_Load()** est activée**.**
On pourra donc y mettre le code initialisant la feuille.
Form1_Activated() est exécuté ensuite car la feuille deviendra active.
Form1.GotFocus() est enfin exécuté puisque la fenêtre prend le focus.
Form1.Enter () est exécuté lorsque l'utilisateur entre dans la fenêtre.

Dès qu'une propriété change de valeur un évènement 'PropriétéChanged' se déclenche :

- **Form1.BackColorChanged** se déclenche par exemple quand la couleur du fond change.
- **Form1.Resized** se déclenche quand on modifie la taille de la fenêtre. (C'est intéressant pour interdire certaines dimensions)

Form1.Leave survient dans il y a perte du focus.
Bien sur il existe aussi **Form1_Desactivate** quand la fenêtre perd le focus et n'est plus active.
Form1.Closing se produit pendant la fermeture de la fenêtre (on peut annuler cette fermeture en donnant à la variable Cancel la valeur True)

Form1.Closed se produit lorsque la fenêtre est fermée.

Il y en a beaucoup d'autres comme par exemple les évènements qui surviennent quand on utilise la souris (MouveUp, MouseDown, MouseMove) ou le clavier (KeyUp, KeyDown, KeyPress) sur la fenêtre, Left Right, Size, Position pour positionner la fenêtre ou définir sa taille…

Méthodes

On a déjà vu que pour faire apparaître une fenêtre il faut utiliser .ShowDialog (pour qu'elle soit modale) ou .Show (pour non modale).

```
Me.Close      `ferme le formulaire.
Me.Activate   `l'active s'il est visible
```

```
Me.Hide        `rend la fenêtre invisible.
```

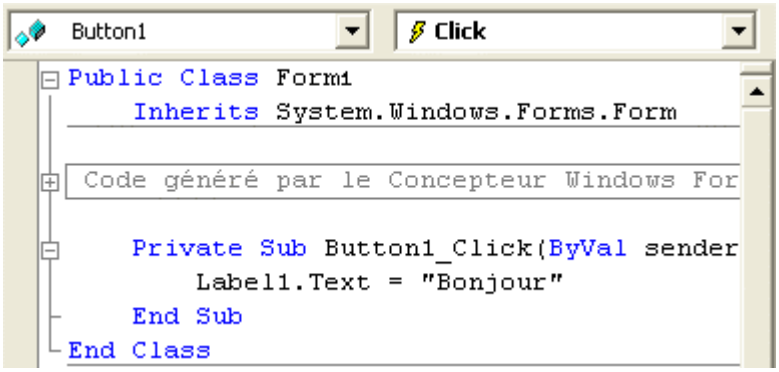
System.Windows.Forms.Form

On se rend compte que quand on dessine une fenêtre Form2 par exemple, VB crée une nouvelle classe **'Class Form2'**

```
Public Class Form2
End Class
```

Elle hérite de **System.Windows.Forms.Form**, on voit bien dans le code :

```
Inherits Systeme.Windows.Forms.Form
```



Elle contient :

- **du code généré automatiquement par le concepteur Windows Forms** (on peut le voir en cliquant sur le petit '+') et qui crée la fenêtre et ses contrôles.
- **les procédures liées aux évènements.**

Quand on tape Dim f As New Form2(), on crée une instance de la Class Form2.

Formulaire d'avant plan

Pour définir au moment de la conception un formulaire en tant que formulaire d'avant-plan d'une application :

- Dans la fenêtre **Propriétés**, attribuez à la propriété TopMost la valeur **true**.

Pour définir par code un formulaire en tant que formulaire d'avant-plan d'une application :

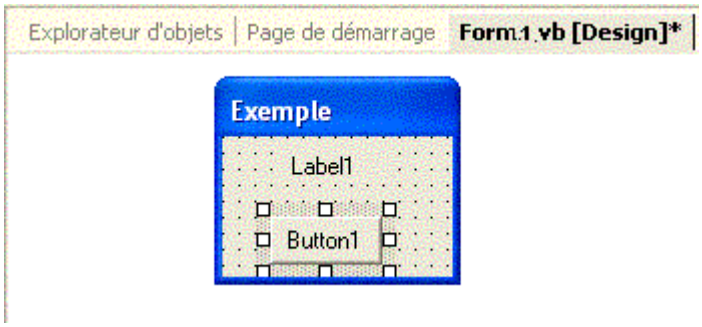
- Dans une procédure, attribuez à la propriété **TopMost** la valeur **true**.

```
Me.TopMost = True
```

3.3 Les Boutons

Ils sont omniprésents dans les 'formulaires'.

Créer un bouton :
Cliquer sur 'Boite à Outils' à gauche , bouton Windows Forms, puis bouton 'Button',cliquer dans Form1, déplacer le curseur sans lâcher le bouton, puis lâcher le bouton : un bouton apparaît.



Modifier ses propriétés :
Name est utilisé pour lui donner un nom explicite (BoutonOk BoutonCancel)

FlatStyle donne un aspect au bouton (Flat, standard, System)

Text contient le texte à afficher sur le bouton. **ForeColor** correspond à la couleur de ce texte (**BackColor** étant la couleur du fond)

Si on y inclut un « & » la lettre qui suit sera souligné et sert de raccourci clavier. &Ok donne sur le bouton Ok.

TextAlign permet de positionner le texte dans le bouton.
Image contient le nom de l'image à afficher sur le bouton (si on veut afficher une image, on le fait en mode Design; noter que quand on distribue l'application, il n'y a pas besoin de fournir le fichier contenant l'image avec l'application). (**AlignImage** permet de positionner l'image sur le bouton.)

On peut aussi puiser une image dans une **ImageList** grâce à la propriété **ImageList** et **ImageIndex**, on peut ainsi changer d'image.

La propriété **BackGroundImage** permet de mettre une image de fond.
Exemple :

```
button1.Text="Ok"   'affiche 'Ok' dans le bouton.
```

Utiliser les évènements :
L'évènement principalement utilisé est **Click()** : quand l'utilisateur clique sur le bouton la procédure est traitée.

```
Private Sub Button_Click(..)
End Sub
```

Cette procédure contient le code qui doit être exécuté lorsque l'utilisateur clique sur le bouton. Le bouton peut être sélectionné grâce à un clic de souris, à la touche ENTRÉE ou à la BARRE D'espacement si le bouton a le focus.

Créer un bouton Ok ou Cancel :
Parfois il faut permettre aux utilisateurs de sélectionner un bouton en appuyant sur la touche

ENTRÉE même si le bouton n'a pas le focus.

Exemple : Il y a sur la fenêtre un bouton "Ok" qui doit être enfoncé quand l'utilisateur tape 'Enter' au clavier, c'est le bouton qui 'valide' le questionnaire (et qui le ferme souvent).

Comment faire?
Définissez la propriété AcceptButton de la Form en lui donnant le nom du bouton.
Cela permet au formulaire d'avoir le comportement d'une boîte de dialogue.

Création d'un bouton par code :
L'exemple suivant crée un Button nommé Button1 sur lequel on voit "Ok", on modifie certaines de ses propriétés et l'ajoute à Form.

```
Private Sub InitializeMonButton()
Dim button1 As New Button1()
button1.Text="Ok"
' Ajouter le bouton à la Form
Controls.Add(button1)
End Sub
```

Il faut par code créer aussi les évènements liés à ce bouton: dans ce cas il faut déclarer le bouton plutôt avec la syntaxe contenant WithEvents et en haut du module.

```
Private WithEvents Button1 As Button
```

Puis écrire la sub évènement.

```
Sub OnClique ( sender As Objet, EvArg As EventArgs) Handles Button1
End Sub
```

Ainsi VB sait que pour un évènement sur le Button1, il faut déclencher la Sub OnClique.
(On reviendra sur cela)

Couleur transparente dans les images des boutons:
On a vu qu'on pouvait mettre une image dans un bouton, il faut pour cela donner à la propriété Image le nom du fichier contenant l'image, ceci en mode Design.
Mais l'image est souvent dans un carré et on voudrait ne pas voir le fond (rendre la couleur du fond **transparente**)

Voici l'image  , je voudrais ne pas afficher le 'jaune' afin de voir ce qu'il y a derrière et donner l'aspect suivant 

Dans Visual Basic 6.0, la propriété MaskColor était utilisée pour définir une couleur qui devait devenir transparente, permettant ainsi l'affichage d'une image d'arrière plan.

Dans Visual Basic .NET, il n'existe pas d'équivalent direct de la propriété MaskColor!!!

Cependant, on peut ruser et définir la transparence :
Dans le " Code généré par le Concepteur Windows Form " après la définition du bouton ou dans Form_Load ajouter:

```
Dim g As New System.Drawing.Bitmap(Button1.Image)
g.MakeTransparent(System.Drawing.Color.Yellow)
Button1.Image = g
```

On récupère le Bitmap de l'image du bouton, on indique que le jaune doit être transparent, on remet le BitMap.

Bien sur il y a intérêt à choisir une couleur (toujours la même) qui tranche pour les fonds de dessin et ne pas l'utiliser dans le dessin lui même.

3.4 Les TextBox

Les contrôles permettant de saisir du texte sont :
Les TextBox
Les RichTexteBox

Les contrôles TextBox

Contrôle qui contient du texte qui peut être modifié par l'utilisateur du programme.
C'est la propriété Text qui contient le texte qui a été tapé par l'utilisateur.

Exemple très simple : Comment demander son nom à l'utilisateur ?
Il faut créer un label dont la propriété Text contient "Tapez votre nom:", suivi d'un TextBox nommé txtNom avec une propriété Text="" (Ce qui fait que la TextBox est vide), enfin un bouton nommé btOk dont la propriété Text="Ok".

Cela donne :

Tapez votre nom: Ok

txtNom.Select() dans Form_Load donne le focus à la TextBox
Une fois que l'utilisateur a tapé son nom, il clique sur le bouton 'Ok'.

Dans la Sub btOk_Click il y a:

```
Dim Nom As String
Nom= txtNom.Text
```

La variable Nom contient bien maintenant le nom de l'utilisateur.

Un TextBox correspond à un **mini éditeur de texte**. (Mais sans enrichissement: sans gras, ni italique…) La police de caractères affectant la totalité du texte peut simplement être modifiée par la propriété **Font**. La couleur du texte peut être modifiée par **ForeColor**, mais la totalité du texte aura la même couleur.

La propriété **.text** permet aussi de modifier le texte visible dans le contrôle.

```
TextBox1.text="Bonjour"     'Affiche 'Bonjour' dans le contrôle.
```

Parmi les multiples propriétés de ce contrôle, signalons :
Multiline : autorise ou non l'écriture sur plusieurs lignes.
Scrollbars : fait figurer une barre de défilement horizontale ou verticale (ou les deux).
PaswordChar : crypte le texte entré sous forme d'étoiles.
MaxLength : limite le nombre de caractères qu'il est possible de saisir.

```
TextBox1.MaxLength= 3 'limite la saisie à 3 caractères.
TextBox1.MaxLength= 0 'ne limite pas la saisie.
```

TextLength : donne la longueur du texte

En mode **MultiLine** la collection **Lines** contient dans chacun de ses éléments une des lignes affichées dans le contrôle :

```
TexBox1.Lines(0)     'contient la première ligne
TexBox1.Lines(1)     'la seconde...
```

Les TextBox contiennent une méthode Undo, annulation de la dernière modification.
La propriété CanUndo du TextBox doit être à True.

Ensuite pour modifier:

```
If textBox1.CanUndo = True Then
textBox1.Undo()

' Vider le buffer Undo.
textBox1.ClearUndo()
End If
```

Ajouter au texte :
On peut ajouter du texte au texte déjà présent dans le TextBox

```
textBox2.AppendText(MonText)
```

C'est équivalent à textBox2.Text=textBox2.Text+MonText

Evènements liés aux TextBox :

- **KeyDown** survient quand on appuie sur la touche.
- **KeyPress** quand la touche est enfoncée.
- **KeyUp** quand on relâche la touche.

Ils surviennent dans cet ordre.
KeyPress permet de récupérer la touche tapée dans e.KeyChar.
KeyDown et **KeyUp** permettent aussi de voir si MAJ ALT CTRL ont été pressés.

On peut récupérer la touche pressé (dans e.KeyChar), mais impossible d'en modifier la valeur (e.KeyChar est en lecture seule par exemple)

Comment récupérer la totalité du texte qui est dans le TextBox ?

```
T= textBox1.Text
```

Comment mettre les lignes saisies par l'utilisateur dans un tableau ?

```
Dim tempArray() as String
tempArray = textBox1.Lines          'On utilise la collection Lines
```

Comment récupérer la première ligne ?

```
T= textBox1.Lines(0)
```

Si une partie du texte est **sélectionnée** par l'utilisateur, on peut la récupérer par :

```
T= TexBox1.SelectedText
```

Pour **sélectionner** une portion de texte on utilise

```
TextBox1.SelectionStart=3 'position de départ
TextBox1.SelectionLength=4 'nombre de caractère sélectionné
```

On peut aussi écrire :

```
TextBox1.Select(3,4)
```

puis

```
TexBox1.SelectedText="toto" 'remplace la sélection par 'toto'
```

Comment positionner le curseur après le troisième caractère ?
En donnant à la propriété SelectionStart la valeur 3

```
TextBox1.SelectionStart=3
```

SelectionLength doit avoir la valeur 0

Comment interdire la frappe de certains caractères dans une TextBox?

Exemple :

Ne permettre de saisir que des chiffres.
Pour cela il faut utiliser l'évènement KeyPress du textBox qui retourne un objet e de type KeyPressEventArgs. e.KeyChar contient le caractère pressé, mais il est en lecture seule!! on ne peut le modifier. Pour annuler la frappe (dans notre exemple si le caractère n'est pas un chiffre) il faut faire e.handled=True.
IsNumeric permet de tester si le caractère est numérique.

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs) Handles TextBox1.KeyPress
If IsNumeric(e.KeyChar) Then
        e.Handled = False
Else
        e.Handled = True
End If
End Sub
```

Compter combien de fois on a tapé certains caractères?

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs) Handles TextBox1.KeyPress
Select Case e.KeyChar

' Compte les  backspaces.
Case ControlChars.Back
        Nombrebackspace = Nombrebackspace + 1

' Compte les  'ENTER'.
Case ControlChars.Lf
        Nombrereturn = Nombrereturn + 1

' Compte les  ESC.
Case Convert.ToChar(27)
        NombreEsc = NombreEsc + 1

' Compte les autres.
Case Else
        keyPressCount = keyPressCount + 1
End Select
End Sub
```

Petite parenthèse:
Pour comparer les caractères il y a 2 méthodes:

```
if e.KeyChar=Convert.ToChar(27) then
```

ou

```
if AscW(e.Keychar)=27 then
```

Différentes manières de récupérer ce qui a été tapé :

On a vu que TextBox.text contient la totalité du texte; si on l'utilise dans l'évènement TextBox1_TextChanged, on récupère le nouveau texte dès que l'utilisateur à tapé quelque chose.

TextBox1_KeyPress() et TextBox1_KeyUp() permettent de récupérer le caractère qui à été tapé.

Y a t-il un moyen de modifier le caractère tapé ? Les propriétés de e comme e.KeyChar (dans KeyPress) ou e.KeyCode, e.KeyData e.KeyValue dans les évènements KeyPress et KeyDown sont en lecture seule!!!

Une solution est de modifier directement le texte :

Exemple :
Si l'utilisateur tape ',' afficher '.' à la place.

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs) Handles TextBox1.KeyPress
Dim pos As Integer
pos = TextBox1.SelectionStart       'on mémorise la position du curseur
If e.KeyChar = "," Then
e.Handled = True     'on ne valide pas le caractère ',' qu n'apparaîtra pas.
TextBox1.Text = TextBox1.Text.Insert(pos, ".")  'on insère un '.'
TextBox1.SelectionStart = pos + 1'on avance le curseur d'un caractère
End If
End Sub
```


Autre solution?

Le contrôle RichTextBox

Si vous êtes débutant, passez à un rubrique suivante, vous reviendrez plus tard à la gestion du code RTF.

Rich Text veut dire 'Texte enrichi'.

Le contrôle RichTextBox permet d'afficher, d'entrer et de manipuler du texte mis en forme. Il effectue les mêmes tâches que le contrôle TextBox, mais il peut également afficher des polices, des couleurs et des liens, charger du texte et des images incorporées à partir d'un fichier, ainsi que rechercher des caractères spécifiques.


Le contrôle RichTextBox a les possibilités d'un traitement de texte comme Word.

Qu'est ce que RTF ?
Le format du texte que l'on peut mettre dans une RichTextBox est le format RTF (Rich Text Format = Format de Texte Enrichi)

Explication : un texte peu être enregistré en brut (sans enrichissements) en RTF (Utilisable dans la plupart des traitements de texte), au format Word (.doc)…

Pour utiliser les fonctionnalités du RichTextBox il faut utiliser la propriété .Rtf.
Quand j'affecte un texte à la propriété .**Text** il est affiché tel quel, sans tenir compte de l'enrichissement.

Quand j'affecte un texte à la propriété **.Rtf** du contrôle pour l'afficher, s'il contient des enrichissements au format RTF, l'enrichissement est affiché.

Les bases du codage RTF
Le texte doit débuter par '{' et se terminer par '}', il peut aussi débuter par "{\rtf1\ansi "
Ensuite les enrichissements s'effectuent par des **balises** qui indiquent le début et la fin de l'attribut, une balise commence par le caractère '\' :
Entre \b et \b0 le texte sera en gras (Bold)

Exemple :
Ajoute le texte "Ce texte est en **gras**." à un contrôle RichTextBox existant.

```
RichTextBox1.Rtf = "{\rtf1\ansi Ce texte est en \b gras\b0.}"
```

Voici les principaux attributs :

\b	\b0	ce qui est entre les 2 balises est en gras
\i	\i0	ce qui est entre les 2 balises est en italique
\par		fin paragraphe (passe à la ligne)
\f font \f1 .. \f0		font numéro 1 entre les 2 balises
\plain		ramène les caractères par défaut
\tab		caractère de tabulation
\fs		taille de caractère \fs28 = taille 28

Mettre un espace après la balise.

Comment afficher un texte enrichir ?

```
RichTextBox1.RTF= T        'T étant le texte enrichi
```

Mettre un texte en couleurs, utiliser plusieurs polices :
Mettre la **table des couleurs** en début de texte :

```
{ \colortbl \red0\green0\blue0;\red255\green0\blue0;\red0\green255\blue0;}
```

Après Colortbl (Color Table) chaque couleur est codée avec les quantités de rouge vert et bleu.
Les couleurs sont repérées par leur ordre: couleur 0 puis 1 puis 2... et séparées par un ';'

Dans notre exemple couleur 0=noir; couleur 1=rouge; couleur 2=vert

Pour changer la couleur dans le texte on utilise \cf puis le numéro de la couleur :

« \cf1 toto \cf0 } » 'toto est affiché en rouge.

Pour **modifier les polices de caractère**, le procédé est similaire avec une **Font Table** :

```
{\fonttbl
{\fo\froman Symbol;}
{\f1\fswiss Arial;}
}
```

Pour passer en Arial \f1 ..\f0

Exemple complet :

```
"{\rtf1\ansi
{ \colortbl
\red0\green0\blue0;
\red255\green0\blue0;
\red0\green255\blue0;}
{\fonttbl
{\fo\froman Symbol;}
{\f1\fswiss Arial;}
}
Ce qui suit est en \f1 \cf1 \i Arial Rouge Italique \f0 \cf0 \i0
}»
```

Cela donne :

Nb : Si vous copier coller l'exemple pour l'essayer, enlever les sauts à la ligne.

Comment modifier l'aspect du texte qui a été sélectionné ?
On n'est plus dans le cas ou on affiche d'emblée la totalité du texte, mais dans le cas ou l'utilisateur veut modifier son texte qui est déjà dans le contrôle.

Exemple :
L'utilisateur sélectionne une portion du texte dans le contrôle puis clique sur une bouton nommé 'Rouge' pour mettre la sélection en rouge.

Dans **BoutonRouge_Click()** écrire :

```
RichTextBox1.SelectionColor = System.Drawing.Color.Red
```

De même pour modifier la police, la hauteur de la police, l'aspect gras ou non :

```
RichTextBox1.SelectionFont = New Font("Tahoma", 12, FontStyle.Bold)
```

Enfin le texte peut être enregistré dans un fichier :

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

Si on remplace **.RichText** par .PlainText c'est le texte brut et non le texte enrichi qui est enregistré.
Pour lire un fichier il faut employer **.LoadFile** avec la même syntaxe.

Comment faire une recherche dans le texte?
La fonction Find permet de rechercher une chaîne de caractères dans le texte :

```
richTextBox1.Find(searchText, searchStart, searchEnd, RichTextBoxFinds.MatchCase)
```

La méthode retourne l'emplacement d'index du premier caractère du texte recherché et met en surbrillance ce dernier, sinon, elle retourne la valeur -1.

3.5 Les labels

Il y a 2 sortes de Label :

- **Les Label**
- **Les LinkLabel**

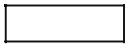
Les 'Label' :

On en a déjà utilisé pour **afficher du texte non modifiable par l'utilisateur**.

Les contrôles Label sont généralement utilisés pour **fournir un texte descriptif à un contrôle**. Vous pouvez par exemple utiliser un contrôle Label pour ajouter un texte descriptif à un contrôle TextBox.

Ceci a pour but d'informer l'utilisateur du type de données attendu dans le contrôle.

Exemple hyper simple :

Donner votre nom: 

La légende qui s'affiche dans l'étiquette est contenue dans la propriété **Text** du label1.

Pour modifier le texte du label par du code :

```
Label1.Text="Donner votre Prénom"
```

La propriété **Alignement** vous permet de définir l'alignement du texte dans l'étiquette (centré, à droite, à gauche), **BorderStyle** permet de mettre une bordure (un tour) ou non..

Il est également possible d'y afficher une image avec la propriété **.Image**

Remarquez que la mise à jour de l'affichage du Label (comme les autres contrôles d'ailleurs) est effectuée en fin de Sub:

Si on écrit :

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

La variable i prend les valeurs 1 à 100, mais à l'affichage rien ne se passe pendant la boucle, VB affiche uniquement 100 à la fin.

Si on désire voir les chiffres défiler avec affichage de 0 puis 1 puis 2..., il faut rafraîchir l'affichage à chaque boucle avec la méthode Refresh() :

```
Dim i As Integer
For i = 0 To 100
        Label1.Text = i.ToString: Label1.Refresh()
Next i
```

Les LinkLabel :

Permettent de créer un **lien** sur un label.

Text Indique le texte qui apparaît.

LinkArea défini la zone de texte qui agira comme un lien, dans la fenêtre de propriété taper 11 ;4 (on verra que c'est plus simple que de le faire par code)
Les 4 caractères à partir du 11ème seront le lien, ils seront soulignés.

Ne pas oublier comme toujours que le premier caractère est le caractère 0.

L'événement **LinkClicked** est déclenché quand l'utilisateur clique sur le lien. Dans cette procédure on peut permettre le saut vers un site Internet ou toute autre action.

Exemple :

```
LinkLabel1.text= "Visitez le site LDF"
LinkLabel1.LinkArea = New System.Windows.Forms.LinkArea(11, 4)
'Pourquoi faire simple !!
```

Cela affiche :

Visitez le site LDF

Si l'utilisateur clique sur le mot site, la procédure suivante est déclenchée :

```
Private Sub LinkLabel1.LinkClicked...
```

Il est possible de modifie la couleur du lien pour indiquer qu'il a été utilisé :

Si **VisitedLinkColor** contient une couleur e.visited=True modifie la couleur.
(e est l'élément qui a envoyer l'évènement, j'en modifie la propriété Visited.)

On peut y inclure une action quelconque, en particulier un saut vers un site Web :

```
System.diagnostics.Process.Start(" http://plasserre.developpez.com/ ")
'correspond au code qui ouvre un browser Internet (Internet Explorer ou Netscape) et
qui charge la page dont l'adresse est indiquée.
```

La collection Links permet d'afficher plusieurs liens dans un même texte, mais cela devient vite très compliqué.

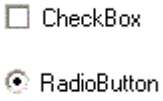
3.6 Les Cases à cocher

Il y a 2 sortes de case à cocher :

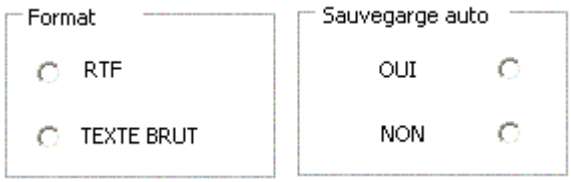
- **Les CheckBox**
- **Les RadioButton**

- Les " cases à cocher " (**CheckBox**) : Elles sont carrées, et indépendantes les unes des autres, si l'utilisateur coche une case , cela n'a pas d'influence sur les autres cases du formulaire, qu'elles soient regroupées dans un cadre pour faire plus joli ou non.

- Les " boutons radio " (**RadioButton**) : Ils sont ronds et font toujours partie d'un groupe (Ils sont dans une fenêtre ou dessinés dans un objet **GroupBox**). Ce groupe est indispensable, car au sein d'un groupe de RadioButton, un seul bouton peut être coché à la fois : si l'utilisateur en coche un, les autres se décochent.



Il faut **regrouper les radios boutons** dans des 'GroupBox' par exemple pour rendre les groupes indépendants :



Ici si je clique sur le bouton 'OUI' à droite, cela décoche 'NON' mais n'a pas d'influence sur le cadre Format.

La propriété Text, bien sur, permet d'afficher le libellé à coté du bouton, on peut aussi mettre une image avec la propriété Image**.** CheckAlign permet de mettre la case à cocher à droite ou à gauche du texte, TextAlign permet d'aligner le texte.

Exemple pour le bouton en haut à droite :

```
RadioButton3.Text = "OUI"
RadioButton3.TextAlign = MiddleCenter     'Middle=hauteur, center = horizontale
RadioButton3.CheckAlign = MiddleRight
```

La propriété la plus intéressante de ces cases est celle qui nous permet de savoir si elle est cochée ou non. Cette propriété s'appelle Checked. Sa valeur change de False à True si la case est cochée.

```
RadioButton.Checked = True                'Coche le bouton
If RadioButton.Checked = True Then ' Teste si le bouton est coché.
End If
```

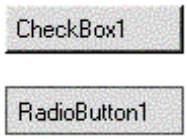
La procédure **RadioButton.CheckedChange()** permet d'intercepter le **changement d'état** d'un bouton.

Pour le CheckButton **TreeState** permet de définir 3 états au lieu de 2 (coché, indéterminé = grisé, non coché)

CheckedState indique un des 3 états (alors que **Checked** n'en indique que deux.)

Appearance peut aussi donner une **apparence de bouton** à la case à cocher. Il est enfoncé

ou pas en fonction de la valeur de Checked.



Ici les 2 boutons ont une Appearance = Button, celui du haut n'est pas coché, l'autre est coché (enfoncé).

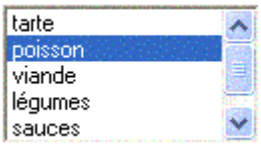
3.7 Les Contrôles 'liste'

Il y a 4 sortes de listes :

- **Les ListBox**
- **Les CheckedListBox**
- **Les Combos**
- **Les ListView**

Les ListBox

Le contrôle **ListBox** affiche une liste d'éléments (d'objets) dans laquelle l'utilisateur peut faire un ou plusieurs choix.



La liste contient "tarte", "poisson", "viande", "légumes", "sauces".
Ici l'élément "poisson" est sélectionné, il est en bleu.

La listBox contient une collection d'"Item":

Elle n'a pas de nombre initialement défini d'élément.
Si j'ajoute un élément à la ListBox, cela ajoute un élément à la collection Items

Items est une *collection* contenant tous les éléments (les objets) chargés dans la liste.
ListBox1.Items est la collection du contrôle ListBox1

La propriété **Items.Count** indique le nombre d'éléments contenus dans la liste.

Attention **le premier élément est toujours l'élément 0**, aussi le nombre d'éléments est égal au numéro de l'élément le plus haut plus un.

Pour ajouter ou supprimer des éléments dans un contrôle ListBox.
Utilisez la méthode **Items.Add**, **Items.Insert**, **Items.Clear** ou **Items.Remove**. En mode conception, vous pouvez également utiliser la propriété **Items**.

Exemples :

Vider la ListBox :

```
ListBox1.Items.Clear()
```

Ajouter l'élément "poisson"

```
ListBox1.Items.Add("poisson")
```

Ajouter '4'

```
ListBox1.Items.Add(4.Tostring)
```

ou

```
ListBox1.Items.Add(4) 'accepté car les items sont des objets.
```

Insérer 'lulu en 4éme position

```
ListBox1.Items.Insert(4, "lulu")
```

Les listBox acceptent des objets, elles affichent généralement ce qu'il y a dans la propriété 'Text' de l'objet.

Charger dans une ListBox1 les nombres de 1 à 100 :

```
For i = 1 To 100
        ListBox1.Items.Add(i.ToString)
Next i
```

Comment enlever des éléments ?

```
ListBox1.Items.RemoveAt(5)          ' Enlever l'élément d'index 5.
ListBox1.Items.Remove(ListBox1.SelectedItem)  ' Enlever l'élément sélectionné
ListBox1.Items.Remove("Tokyo")   ' Enlever l'élément "Tokyo".
```

Comment lire l'élément 3 ?

```
T=ListBox1.Items(3).ToString
```

Comment rechercher l'élément qui contient une chaîne de caractères ?

```
List1.FindString("pa")          ` retourne le numéro du premier élément commençant par 'pa'.
x=List1.FindString("pa",12)          `retourne le numéro de l'élément commençant par 'pa' en cherchant à partir du 12 éme élément.
x=List1.FindStringExact("papier") 'permet de rechercher l'élément correspondant exactement à la chaîne.
```

Comment sélectionner un élément par code ?

```
ListBox1.SetSelected(x, True)
```

L'utilisateur double-clique sur un des éléments, comment récupérer son numéro ?
Grâce à **SelectedIndex**.

```
Private Sub ListBox_DoubleClick.
        N=ListBox1.SelectedIndex
End If
```

`N contient le numéro de l'élément sélectionné. Attention comme d'habitude, si je sélectionne « 3 » c'est en faite l'élément numéro 2.

SelectedIndex retourne donc un entier correspondant à l'élément sélectionné dans la zone de liste. Si aucun élément n'est sélectionné, la valeur de la propriété **SelectedIndex** est égale à -1.

La propriété **SelectedItem** retourne l'élément sélectionné ("poisson" dans l'exemple si dessus).

Et la multi sélection, quels éléments ont été sélectionnés ?
La propriété **SelectionMode** indique le nombre d'éléments pouvant être sélectionnés en même temps.

Lorsque plusieurs éléments sont sélectionnés, la valeur de la propriété **SelectedIndex** correspond au rang du premier élément sélectionné dans la liste. Les collections **SelectedItems** et **SelectedIndices** contiennent les éléments et les numéros d'index sélectionnés.

Si la propriété Sorted est à True, la liste est triée automatiquement.

On peut **'charger' une ListBox automatiquement avec un tableau** en utilisant Datasource :

```
Dim LaList() As String = {"one", "two", "three"}
ListBox1.DataSource = LaList
```

On peut aussi utiliser AddRange :

```
Dim Ite(9) As System.Object
```

```
Dim i As Integer
For i = 0 To 9
Ite(i) = "Item" & i
Next i

ListBox1.Items.AddRange(Ite)
```

Comment connaître l'**index de l'élément que l'on vient d'ajouter** ? (Et le sélectionner)

```
Dim x As Integer
x = List1.Items.Add("Hello")
List1.SelectedIndex = x
```

On utilise la valeur retournée (x dans notre exemple) par la méthode Add.
(NewIndex n'existe plus en VB.NET)

Comment affecter à chaque élément de la liste un numéro, une clé ?
Exemple : je charge dans une ListBox la liste des utilisateurs mais quand on clique sur la liste, je veux récupérer le numéro de l'utilisateur.

Comment donc, à chaque élément de la listbox, donner un numéro (différent de l'index).
En VB6 on utilisait une propriété (ListBox.ItemData()) pour lier à chaque élément de la listBox un nombre (une clé); cela n'existe plus en VB.Net!!

Il faut utiliser les fonctions de compatibilité :

```
VB6.SetItemData(ListBox1, 0, 123) 'pour lier à l'élément 0 la valeur 123.
```

Ce n'est pas simple!!

Une alternative, pas très élégante :

Ajouter l'élément "toto"+chr$(9)+chr$(9)+ clé (clé n'est pas visible car les caractères tabulation l'ont affichée hors de la listbox)

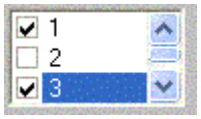
Quand l'utilisateur clique sur la ligne, on récupère la partie droite donc la clé.

Quand on charge une ListBox directement avec une base de données, il y a une solution pour gérer une clé.

Lorsque la propriété **MultiColumn** a la valeur **true**, la liste s'affiche avec une barre de défilement horizontale. Lorsque la propriété **ScrollAlwaysVisible** a la valeur **true**, la barre de défilement s'affiche, quel que soit le nombre d'éléments.

CheckedListBox

C'est une Listbox mais avec une **case à cocher** sur chaque ligne



Attention : SelectedItems et **SelectedIndices** ne déterminent pas les éléments qui sont cochés, mais ceux qui sont en surbrillance.

La collection **CheckedItems** vous donne par contre les éléments cochés. La méthode **GetItemChecked** (avec comme argument le numéro d'index) détermine si l'élément est coché.

Exemple :

Pour déterminer les éléments cochés dans un contrôle CheckedListBox :
Tester chaque élément de la collection **CheckedItems**, en commençant par 0. Notez que cette méthode fournit le numéro que porte l'élément dans la liste des éléments cochés, et non dans la liste globale. Par conséquent, si le premier élément de la liste n'est pas coché alors que le deuxième l'est, le code ci-dessous affiche une chaîne du type « Item coché 1 = Dans la liste : 2 ».

```
    If CheckedListBox1.CheckedItems.Count <> 0 Then

    'S'il y a des éléments cochés une boucle balaye les éléments cochés
    '(Collection CheckedItems) et affiche le numéro de l'élément DANS LA LISTE toutes lignes.
    Dim x As Integer
    Dim s As String = ""
    For x = 0 To CheckedListBox1.CheckedItems.Count - 1
        s = s & "Item coché " & (x+1).ToString & " = " & « Dans la liste : »&
        CheckedListBox1.CheckedItems(x).ToString & ControlChars.CrLf
    Next x
    MessageBox.Show(s)
End If
```

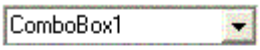
On rappelle comme toujours que quand on parle du 3eme élément cela correspond à l'index 2.

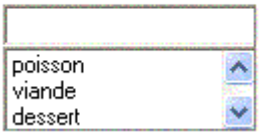
Les ComboBox

Les listes Combo (Liste combiné) possèdent deux caractéristiques essentielles par rapport aux ListBox.

Elles sont ***modifiables*** : c'est-à-dire que l'utilisateur a la possibilité d'entrer un élément qui ne figure pas au départ dans la liste. Cette caractéristique concerne donc les données proprement dites, cela se traduit par la présence d'une zone de texte en haut de la liste.

Elles peuvent être ***déroulantes*** ou déjà déroulée, c'est-à-dire qu'on ne voit qu'un seul élément de la liste à la fois et qu'il faut cliquer sur la flèche du côté pour " déplier " la liste, ou bien que la liste est déjà visible. C'est la propriété **DropDownList** qui gère cela.





La combos du bas a sa DropDownList=Simple
L'utilisateur peut donc cliquer dans la liste (ce qui met le texte cliqué dans la zone texte), ou taper un nouveau texte.

Items.Add	(méthode) ajoute un élément à une liste.
Items.Clear	(méthode) efface tous les éléments d'une liste.
Items.Count	(propriété) renvoie le nombre d'éléments d'une liste.
Multiselect	(propriété) permet la sélection multiple.
Item.Remove	(méthode) supprime un élément de la liste.
Sorted	(propriété) trie les éléments d'une liste.

Comment récupérer la zone texte quand elle change ?
Elle est dans la propriété **Text**.

On utilise l'évènement **TextChanged** qui se déclenche quand le texte est modifié.

```
Private Sub ComboBox1_TextChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles ComboBox1.TextChanged
        Label1.Text = ComboBox1.Text
End Sub
```

Le Contrôle ListView

De plus en plus puissant, le contrôle ListView permet d'afficher des listes multi colonnes, ou des listes avec icône ou case à cocher.

En mode conception :
La propriété **View** permet de déterminer l'aspect général du contrôle, elle peut prendre les valeurs :

- **Details** permet une liste avec sous éléments et titre de colonnes.
- Liste utilise un ascenseur horizontal.
- **LargeIcon**
- **SmallIcone**

Par programmation cela donne :

```
ListView1.View= View.Details
```

Utilisons le mode détails (Appelé mode Rapport)

Nombre	Carré	Cube
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216

Comment remplir les en-têtes de colonnes ?

En mode conception il y a une ligne **Columns**, le fait de cliquer sur le bouton d'expansion (…) ouvre une fenêtre, cliquer sur 'Ajouter' permet d'ajouter une colonne, la propriété **Text** permet de donner un libellé qui apparaîtra en haut de la colonne. On peut ainsi nommer les 3 colonnes (« Nombre », « Carré », « Cube » dans notre exemple).

Par programmation cela donne :

```
ListView1.Colonns.Add (« Nombre », 60, HorizontalAlignment.Left)
…
```

Pour remplir le tableau, on pourrait, sur la ligne Items de la fenêtre des propriétés, cliquer sur … et rentrer les valeurs 'à la main'.


En pratique on crée les colonnes, le nom des colonnes en mode conception, on remplit le tableau par programmation :

Exemple : Faire un tableau de 3 colonnes, mettre les nombres de 1 à 100 dans la première, leur carré dans la seconde, leur cube dans la troisième.

Pour chaque ligne je crée un objet **ListViewItem**, sa propriété **Text** contient le texte de la première colonne, j'ajoute à cet objet des **SubItems** qui correspondent aux colonnes suivantes. Enfin j'ajoute le **ListViewItem** au contrôle **ListView.**

```
Dim i As Integer
For i = 1 To 100
```

```
            Dim LVI As New ListViewItem
            LVI.Text = i.ToString
            LVI.SubItems.Add((i * i).ToString)
            LVI.SubItems.Add((i * i * i).ToString)
            ListBox1.Items.Add(LVI)
    Next i
```

Comment intercepter le numéro de la ligne qui a été cliquée par l'utilisateur (et l'afficher) ?

```
    Private Sub ListBox1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Handles ListBox1.Click
        Label1.Text = ListBox1.SelectedIndices(0).ToString
    End Sub
```

Si la propriété **MultiSelect** est à False il y a, bien sur, une seule ligne sélectionnée, sinon les lignes sélectionnées sont dans la collection SelectedIndices().

Si on voulait récupérer le texte de la ligne sélectionnée, il aurait fallu utiliser :

```
    ListBox1.SelectedItems(0)
```

Si la propriété **GridLine** est à True, des lignes matérialisant les cases apparaissent.
Si la propriété **CheckedBox** est à True, des cases à cocher apparaissent.

Attention : si la somme des colonnes est plus large que le contrôle, un ascenseur horizontal apparaît !!

Pour ne pas voir cet ascenseur, rusez sur la largeur des colonnes (c'est le 2eme paramètre de la méthode .Columns.Add).

3.8 Les fenêtres toutes faites

Il existe :

- **Les MessageBox.**
- **Les InputBox**

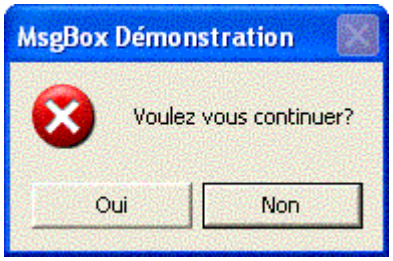
Ces fenêtres toutes faites facilitent le travail :

MessageBox :

Ouvre une fenêtre qui présente un message.
C'est une fonction qui affiche un message dans une boîte de dialogue, attend que l'utilisateur clique sur un bouton (Ok ou Oui-Non..), puis retourne une information qui indique le bouton cliqué par l'utilisateur.

Reponse= **MessageBox.show**(TexteAAfficher,Titre, TypeBouton etIcone, BoutonParDéfaut)

Exemple :



Paramètres :

TexteAAfficher

Obligatoire. Expression **String** affichée comme message de la boîte de dialogue (longueur maximale 1 024 caractères). N'oubliez pas d'insérer un retour chariot si le texte est long, cela crée 2 lignes.

Titre

Expression **String** affichée dans la barre de titre de la boîte de dialogue. Si l'argument *Titre* est omis, le nom de l'application est placé dans la barre de titre.

TypeBouton et Icons

Expression numérique qui représente la somme des valeurs spécifiant
-le nombre et le type de boutons à afficher :

MessageBoxButtons.OKOnly	Un seul bouton 'Ok'
MessageBoxButtons.YesNo	Deux boutons 'Oui' 'Non'
MessageBoxButtons.OkCancel	'Ok' et 'Annuler'
MessageBoxButtons.AbortRetryIgnore	'Annule' 'Recommence' 'Ignore'

..

-le style d'icône à utiliser :
MessageBox.Icons.Critical
MessageBox.Icons.Exclamation
MessageBox.Icons.Question
MessageBox.Icons.Information

L'identité du bouton par défaut

MessageBox.DefaultButtons.DefaultButton1
MessageBox.DefaultButtons.DefaultButton2

Retour de la fonction :
Retourne une constante qui indique quel bouton à été pressé.

DialogResult.Yes

```
DialogResult.No
DialogResult.Cancel
DialogResult.Retry
DialogResult.Ok
```

L'ancienne syntaxe VB avec MsgBox est conservée :

```
Reponse= MsgBox(TexteAAfficher, TypeBouton, Titre)
```

Dans ce cas il faut utiliser MsgBoxStyle MggBoxIcons et MsgBoxResult pour le retour. De plus les arguments ne sont pas dans le même ordre!!
Il est conseillé d'utiliser MessageBox.Show (qui est VB.NET) plutôt que MsgBox qui est de la compatibilité avec VB

Exemple :

```
Reponse=MessageBox.Show(«Bonjour»)
```

'Affiche le message 'Bonjour' avec un simple bouton 'Ok'
Cela sert à fournir un message à l'utilisateur sans attendre de choix de sa part.
Autre exemple en ancienne syntaxe :

```
R=MsgBox("Continuer"& chr$(13)& "l'application?", MsgBoxStyle.YesNo, "Attention"
```

Affiche une MessageBox avec dans la barre de titre « Attention »
'Affiche dans la boite :
« Continuer
l'application » (sur 2 lignes)
'La boite a 2 boutons : 'Oui' 'Non'

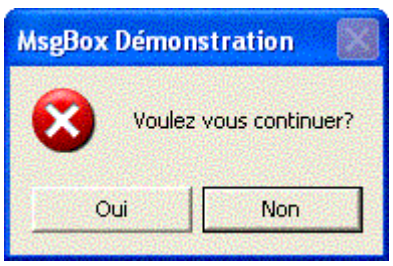
Exemple complet :

```
Dim msg As String
Dim title As String
Dim style As MsgBoxStyle
Dim response As MsgBoxResult
msg = "Voulez vous continuer?"    ' Définition du message à afficher.
style = MsgBoxStyle.DefaultButton2 Or _
        MsgBoxStyle.Critical Or MsgBoxStyle.YesNo 'On affiche Oui Non
title = "MsgBox Démonstration"    ' Définition du  titre.
' Affiche la boite MsgBox.

response = MsgBox(msg, style, title)

If response = MsgBoxResult.Yes Then      ' L'utilisateur a choisi Oui.
' code si l'utilisateur à cliquer sur Oui
Else
' code si l'utilisateur à cliquer sur No.
End If
```

Voila ce que cela donne :



'On remarque que dans l'exemple, on crée des variables dans lesquelles on met le texte ou les constantes adéquates, avant d'appeler la fonction MsgBox.

InputBox

C'st une fonction qui permet d'ouvrir une fenêtre qui pose une question :

Elle retourne la réponse tapée par l'utilisateur :
Le retour est effectué dans une variable String.

```
Dim Nom As String
Nom = InputBox("Bonjour","Tapez votre nom ?")
```

Cela donne :

On pourrait rajouter un 3eme argument=la réponse par défaut.
Si l'utilisateur clique sur le bouton annuler, une chaîne vide est retournée.

OpenFileDialog

Comment afficher une boite de dialogue permettant de sélectionner un fichier (ou des fichiers) à ouvrir par exemple ?

Dans la boite à Outils, cliquez sur OpenFileDialog puis cliquez sur la fenêtre en cours : un contrôle OpenFileDialog1 apparaît sous le fenêtre.

Ouvre une boite de dialogue permettant de choisir un nom et un chemin de fichier, au programmeur d'écrire le code lisant les fichiers.

Dans le code à l'endroit ou doit s'ouvrir la fenêtre, tapez :

```
OpenFileDialog1.ShowDialog()
```

C'est suffisant pour créer une fenêtre montrant l'arborescence des fichiers et répertoires et pour que l'utilisateur choisisse un fichier, mais le plus souvent on a besoin que la boite de dialogue propose un type de fichier et un répertoire précis.

Par exemple je veux ouvrir un fichier .TXT dans le répertoire c:\MesTextes
Il faut dans ce cas, **AVANT** le ShowDialog renseigner certaines propriétés du contrôle OpenFileDialog1 :

```
With OpenFileDialog1
        .Filter="Fichiers txt|*.txt" ' on travaille uniquement sur les .txt
                'S'il y a plusieurs filtre les séparer par |
        .Multiselect=False          'sélectionner 1 seul fichier
        .CheckFileExists=True       'Message si nom de fichier qui n'existe pas.
                'Permet d'ouvrir uniquement un fichier qui existe
End With
```

Comment afficher la boite et vérifier si l'utilisateur à cliqué sur ouvrir ?

```
If OpenFileDialog1.ShowDialog= DialogResult.Ok Then
end if
```

Maintenant, OpenFileDialog1.FileName contient le nom du fichier sélectionné (avec extension et chemin)

Path.GetFileName(OpenFileDialog1.FileName) donne le nom du fichier sans chemin.

SaveFileDialog

Boite de dialogue fonctionnant de la même manière que OpenFileDialog mais avec quelques propriétés spécifiques.

Ouvre une boite de dialogue permettant à l'utilisateur de choisir un nom et un chemin de fichier, au programmeur d'écrire le code enregistrant les fichiers.

```
SaveFileDialo1.CreatePrompt= True        ' Message de confirmation si
                                         'création d'un nouveau fichier
SaveFileDialo1.OverwritePrompt=True      'Message si le fichier existe déjà
                                         'évite l'effacement d'ancienne données
SaveFileDialo1.DefaultExt="txt"          'extension par défaut
```

On récupère aussi dans **.FileName** le nom du fichier si la propriété **.ShowDialog** à retourné **DialogResult.Ok**.

Il existe aussi :

- **LoadDialog**
- **PrintDialog**

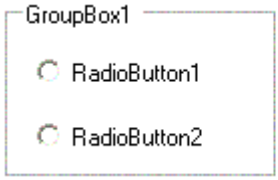
3.9 Regroupement de contrôles

On peut regrouper des contrôles dans :

- **Les GroupBox.**
- **Les Panels.**
- **Les PictureBox.**
- **Les TabControl.**

GroupBox et Panel

Il est possible de regrouper des contrôles dans un **container**, on peut par exemple regrouper plusieurs RadioButton. Le container peut être un **GroupBox** ou un **Panel**.



GroupBox **Panel** avec AutoScroll=True et BorderStyle=Single

Pour l'utilisateur, le fait que toutes les options soient regroupées dans un panneau est un indice visuel logique (Tous les RadioButton permettrons un choix dans une même catégorie de données).

Au moment de la conception, tous les contrôles peuvent être déplacés facilement, si vous déplacez le contrôle GroupBox ou Panel, tous les contrôles qu'il contient sont également déplacés. Les contrôles regroupés dans un panneau ou un GroupBox sont accessibles au moyen de la propriété Controls du panneau.

Le contrôle **Panel** est similaire au contrôle **GroupBox,** mais seul le contrôle Panel peut disposer de barres de défilement et seul le contrôle **GroupBox** peut afficher une légende.
La légende de la **GroupBox** est définie par la propriété **Text**.

Pour faire apparaître les barres de défilement dans le **Pannel** mettre AutoScroll =True et AutoScrollMinSize =100;100

Dans un Panel, pour afficher des barres de défilement, attribuez à la propriété **AutoScroll** la valeur **true**.. La propriété **BorderStyle** détermine si la zone est entourée d'une bordure invisible (**None**), d'une simple ligne (**FixedSingle**) ou d'une ligne ombrée (**Fixed3D**).

Comment créer un contrôle Panel ?

Faites glisser un contrôle **Panel** de l'onglet **Windows Forms** de la boîte à outils jusqu'à un formulaire.
Ajoutez des contrôles au panneau en les déposant dans le panneau.
Si vous voulez mettre dans le panneau des contrôles existants, sélectionnez-les tous, coupez-les dans le Presse-papiers, sélectionnez le contrôle **Panel** et collez-les.

PictureBox

Le contrôle **PictureBox** peut afficher une image mais peu aussi servir de conteneur à d'autres contrôles.
Retenons la **notion de conteneur** qui est le contrôle **parent**.

TabControl

Ce contrôle permet de créer des onglets comme dans un classeur, onglets entièrement gérés par VB. Chaque page peut contenir d'autres contrôles.

En mode conception, en passant par la propriété **TabPages,** on ajoute des onglets dont la propriété **Text** contient le texte à afficher en haut (Ici : Page 1..). il suffit ensuite de cliquer sur chaque onglet et d'y ajouter les contrôles.

En mode run les onglets fonctionnent automatiquement: cliquez sur Page 2 affiche la page correspondante (et déclenche l'événement **Click** de cet objet **TabPage)**…

La propriété **Alignment** permet de mettre les onglets en haut, en bas, à droite, à gauche.

Evènement commun

Exemple :
3 cases à cocher permettent de colorer un label en vert rouge ou bleu.

Comment gérer les évènements ?

On peut écrire 3 routines complètes pour chaque case à cocher.
Il est aussi toujours possible dans chacune des 3 procédures CouleurX.checkedChanged de vérifier si la case est cochée et de modifier la couleur.
C'est plus élégant d'avoir une procédure unique qui, en fonction de la case à cocher qui a déclenché l'évènement, change la couleur.

On désire donc parfois que l'**évènement de plusieurs contrôles différents soit dirigé sur une seule et même procédure**.

Mais, la notion de groupe de contrôle comme en VB6 n'existe plus!!!

Par contre par l'intermédiaire du Handles, il est possible d'associer plusieurs évènements à une seule procédure :

```
Private Sub CouleurCheckedChanges (ByVal sender As System.Objet, ByVal e As
System.EventArgs)
Handles CouleurVert.CheckedChanged, CouleurRouge.CheckedChanged,
CouleurBleu.CheckedChanged

End Sub
```

Cette procédure est activée quand les cases à cocher CouleurVert CouleurBleu, CouleurRouge changent d'état.

A noter que **Sender** est le contrôle ayant déclenché l'évènement et **e** l'évènement correspondant.

Pour modifier la couleur il faut ajouter dans la procédure :

```
Select Case sender.Name
        Case "CouleurRouge"
                Lbl.BackColor= ..Rouge
```

```
.....
```

Je ne suis pas certain que cela fonctionne, il faut plutôt mettre :

```
Select Case sender
    Case CouleurRouge
```

Enfin la ligne suivante marche !

```
    If sender Is CouleurRouge Then...
```

3.10 Positionnons les contrôles

On peut :
- **Dimensionner les contrôles**
- **Les positionner.**

Tous les contrôles héritent donc tous de la classe Windows Forms**.**
Les Windows Forms ont des propriétés, que tous les contrôles récupèrent :

Concernant la taille :

On peut utiliser :
- **Left, Top** coordonnées du coin supérieur droit et **Bottom, Right** inférieur gauche.
- **Location** : coordonnées X, Y du coin supérieur droit du contrôle en pixels.
- **Height, Width** pour la hauteur et la largeur du contrôle en pixels.
- **Size** : hauteur, largeur peut aussi être utilisé.

Exemple :

```
Button.left=188
Button.Top=300
Ou
Button.Location= New System.Drawing.Point(188,300)
```

System.Drawing.Point() positionne un point dans l'espace.

En mode conception il est bien plus simple de dimensionner les contrôles à la main dans la fenêtre Design.

Pour le redimensionnement de fenêtre :
Pour que l'utilisateur puisse redimensionner la fenêtre qu'il a sous les yeux (en cliquant sur les bords) il faut que la propriété **FormBorderStyle** de la fenêtre **= Sizable.**

Mais si l'utilisateur modifie la taille de la fenêtre qui contient les contrôles, la taille des contrôles ne suit pas.

(Avant cette version VB.net, il fallait dans l'événement Form_Resize, déclenché par la modification des dimensions de la fenêtre, écrire du code modifiant les dimensions et positions des contrôles afin qu'il s'adaptent à la nouvelle fenêtre.)

En VB.Net c'est plus simple grâce à :
Anchor :
Permet d'**ancrer** les bords.
Un bord ancré reste à égale distance du bord du conteneur quand le conteneur (la fenêtre) est redimensionné.

En mode conception il suffit de cliquer sur '. . .' en face de Anchor pour voir s'ouvrir une fenêtre, cliquer sur les bords que vous voulez ancrer.

Par défaut les bord Top (haut) et left(gauche) sont ancrés.

Expliquons :
Left est ancré, si je déplace le bord droit de la fenêtre, le contrôle n'est pas déplacé car la distance bord gauche de la fenêtre et bord gauche du contrôle est fixe. Par contre si je déplace le bord gauche de la fenêtre, le contrôle suit.

Exemple :
Prenons 2 contrôles dans une fenêtre, celui de gauche avec Anchor à left et celui de droite à left et right.

Si je déplace le bord droit (ou le gauche d'ailleurs) : le contrôle droit est redimensionné, les 2 contrôles restent cote à cote.

Dock

Amarre aux bords. Il y a même possibilité d'amarrer aux 4 bords (Fill) pour remplir le conteneur, et de modifier la propriété **DockPaddind** afin se s'éloigner légèrement des bords pour faire joli.

Spliter

Le contrôle **Splitter** servent à redimensionner des contrôles **au moment de l'exécution.**
Le contrôle **Splitter** est utilisé dans les applications dont les contrôles présentent des données de longueurs variables, comme l'Explorateur Windows.

Pour permettre à un utilisateur de redimensionner un contrôle ancré au moment de l'exécution, ancrez le contrôle à redimensionner au bord d'un conteneur, puis ancrez un contrôle Splitter sur le même côté de ce conteneur.

3.11 MainMenu et ContextMenu

MainMenu :

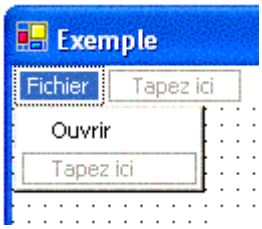
On peut ajouter un menu dans une fenêtre.
Beaucoup d'applications contiennent un menu.

Exemple de menu :
Fichier
    Ouvrir
    Fermer
    Imprimer
    Quitter
Edition
    Couper
    Copier
    Coller
    ...

On remarque que le contenu des menus est **standardisé** afin que l'utilisateur s'y retrouve sans aide (L'utilisateur lit, à mon avis, rarement les aides !!!)

Comment créer un menu ?
En allant dans la boite à outils, chercher un **main menu** et en le déposant sur la fenêtre : il apparaît en dessous de la fenêtre.

Pour 'dessiner' le menu, il suffit de mettre le curseur sur le menu en haut de la fenêtre, ou est écrit 'Taper ici', à cet endroit tapez le texte du menu, 'Fichier' par exemple.



Il apparaît automatiquement un 'Tapez Ici' pour les lignes dessous ou le menu suivant.
Les lignes du menu sont nommées automatiquement MenuItem1, MenuItem2…

Quand le curseur est sur une ligne du menu, la fenêtre de propriété donne les propriétés de la ligne :
La propriété **ShortKey** permet de créer un raccourci.
La propriété **Checked** permet de cochez la ligne
La propriété **Visible** permet de faire apparaître ou non une ligne.

Si vous double-cliquez sur la ligne du menu vous voyez apparaître :

```
Private Sub MenuItem1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MenuItem1.Click
End Sub
```

C'est la procédure événement liée à la ligne du menu.

Quand l'utilisateur clique sur une ligne du menu c'est le code contenu dans cette procédure qui est effectué.

ContextMenu :

C'est un menu qui s'ouvre quand, sur un objet, on clique sur le bouton droit de la souris.

En allant dans la boite à outils, chercher un **Context menu**, on le dépose sur la fenêtre, il apparaît en dessous de la fenêtre.

Si on le sélectionne avec la souris, il apparaît en haut et comme pour le menu principal, on peut ajouter des lignes.

Il faut ensuite affecter ce Context Menu à un contrôle, pour cela donner à la propriété ContextMenu du contrôle le nom du ContextMenu.

```
TextBox1.ContextMenu = ContextMenu1
```

Si vous double-cliquez sur une ligne du menu vous voyez apparaître les procédures évènement correspondantes.

3.12Avoir le Focus

Nous allons étudier comment un objet de l'interface devient actif.

Lorsqu'une fenêtre ou un contrôle est actif on dit qu'il a le **focus.**

Si une fenêtre prend le focus, sa barre de titre en haut prend la couleur active, si c'est un contrôle texte, le curseur apparaît dedans.

Comment donner le focus à une fenêtre ?

Si une fenêtre est visible la méthode **Activate** lui donne le focus.
Form1.Activate()

Dans ce cas l'évènement Form1_Activated survient.
La méthode Desactivate est déclenchée quand la fenêtre perd le focus.

Comment donner le focus à un contrôle ?

Avec la méthode Focus
TxtNom.Focus()

Avec la méthode Select :
TxtNom.Select() 'donne le focus à la zone de texte Txnom et met le curseur dedans.

On peut la surcharger et en plus sélectionner une portion du texte :
TxtNom.Select(2,3) 'donne le focus et sélectionne 3 caractères à partir du second.
Ou forcer à **ne rien sélectionner** (second argument à 0).

On peut interdire à un contrôle le focus en donnant la valeur False à sa propriété **CanFocus**.

Aussi avant de donner le focus il est préférable de vérifier s'il peut le prendre :

```
If TxtNom.CanFocus then
    TxtNom.Focus()
End If
```

L'évènement **GotFocus** se produit quand le contrôle **prend** le focus.
Private Sub TxtNom_GotFocus..
End Sub

Cascade d'évènement quand on prend ou on perde le focus

Enter
Se produit quand l'utilisateur entre dans le contrôle.

GotFocus
Se produit quand le contrôle prend le focus.

Leave
Se produit quand le focus quitte le contrôle.

Validating
Se produit lorsque le contrôle est en cours de validation. La validation c'est vous qui devez la faire!!!
Pour un bouton par exemple se produit lorsque l'on quitte le bouton, cela permet de contrôler la validité de certaines données et si nécessaire d'interdire de quitter le contrôle si certaines

conditions ne sont pas remplies :

```
Private Sub Button1_Validating ((ByVal sender As Object, ByVal e As
System.ComponentModel.CancelEventArgs) Handles Button1.Validating
If condition then
        e.Cancel = True       'Annuler la perte du focus
End If
End Sub
```

Validated
Se produit lorsque le contrôle à terminé sa validation

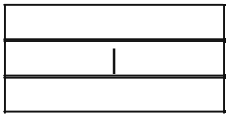
LostFocus
L'évènement **LostFocus** se produit quand le contrôle **perd** le focus.

Si la propriété CauseValidating du contrôle a la valeur **false**, les événements **Validating** et **Validated** sont supprimés.
Les événements **Enter** et **Leave** sont supprimés dans les formulaires (fenêtres).
Les événements équivalents dans la classe **Form** sont les événements Activated et Desactivate.

Certains contrôles ne peuvent pas avoir le focus, comme les labels par exemple.

Usage du clavier pour passer d'un contrôle à l'autre

Dans une application ou un utilisateur saisi beaucoup de données dans de multiples contrôles, il passe souvent d'un contrôle (TextBox par exemple) au suivant avec la touche TAB.



Comment permettre cela ?
Chaque contrôle à une propriété **TabIndex** qui s'incrémente automatiquement de 0 à 1, 2, 3...quand on ajoute des contrôles sur une fenêtre.

Lorsque l'utilisateur appuie sur TAB, le focus passe au contrôle qui a le TabIndex immédiatement supérieur.

On peut modifier le TabIndex des contrôles pour modifier l'ordre de tabulation.

Quand **TabStop** a la propriété False (au niveau d'un contrôle) celui ci est exclu de l'ordre de tabulation et le focus ne s'arrête pas.

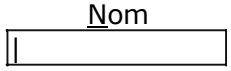
Raccourcis clavier

Dans beaucoup d'applications certains contrôles ont un raccourci clavier :
Exemple : **Nouveau** est une ligne de menu. N étant souligné, **ALT-N** déclenche la ligne de menu, donne le focus au contrôle.

Comment faire cela :
Dans la propriété Text du contrôle, quand on tape le texte en mode conception, il faut mettre un '&' avant la lettre qui servira de raccourci clavier.

'&Nouveau' dans notre exemple donnera bien **Nouveau et ALT N** fonctionnera.
Pour une **TextBox**, la propriété text ne peut pas être utilisée, aussi il faut mettre devant la textBox un contrôle label (qui lui ne peut pas avoir le focus), si le TabIndex du label et du

TextBox se suivent, le fait de faire le raccourci clavier du label donnera le focus au TextBox.



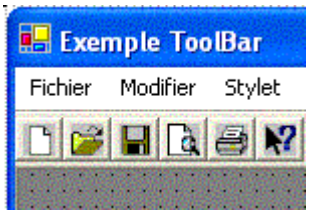
Exemple quand l'utilisateur tapera Alt-N, le focus ira dans le TextBox dessous.

3.13ToolBar et StatusBar

Comment mettre une barre de bouton en haut et une barre d'état en bas?

La barre de bouton

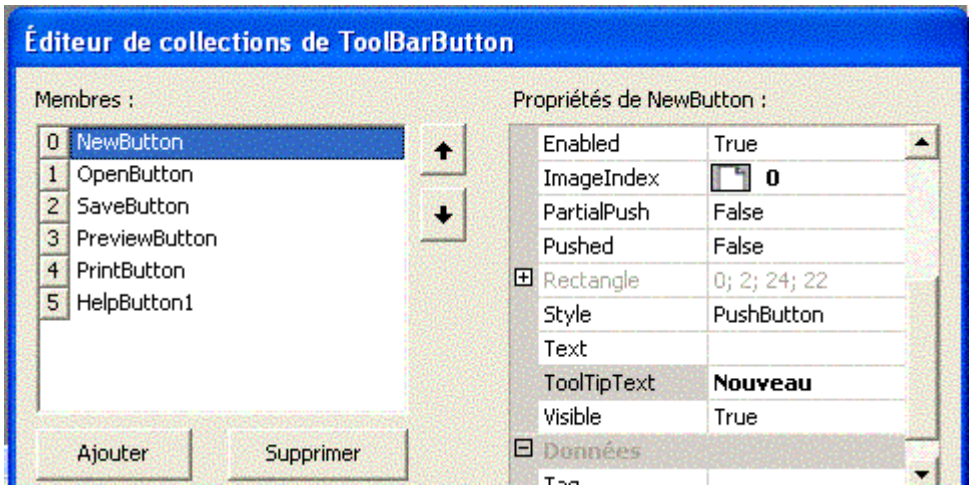
Voici un exemple classique, sous le menu il y a une barre de bouton: **Nouveau, Ouvrir, Enregistrer, Chercher, Imprimer...**



Aller chercher dans la boite à outils un contrôle **ToolBar**, il se place en haut, sous le menu. Mettre aussi un **ImageList**. (Un contrôle ImageList est un contrôle qui stocke des images, chaque image étant chargés en mode conception et repérées par un numéro (0, 1, 2, 3...)

Dans les propriétés du ToolBar mettre dans la propriété **ImageList** le nom du contrôle **ImageList** qui contient les images des boutons.

Ouvrir la **collection Buttons** dans la fenêtre des propriétés de la ToolBar pour pouvoir ajouter ou modifier les boutons :



Vous pouvez ajouter ou enlever des boutons.

Chaque bouton a **ses propriétés** affichées à droite :

- **Name** Nom du Bouton. Exemple : NewButton.
- **ImageIndex** donne le numéro de l'image (contenue dans l'imagelist) qui doit s'afficher dans le bouton.
- **ToolTipText** donne le texte du ToolTip (Carré d'aide qui apparaît quand on est sur le bouton) Il faut aussi que la propriété ShowToolTip de la ToolBar soit à True

L'évènement déclenché par le click de l'utilisateur sur un bouton est :
ToolBar1_ButtonClick

L'argument e contient les arguments de l'évènement click de la ToolBar. e.Button contient le nom du bouton qui a déclenché l'évènement. Pour chaque nom de bouton on appellera la procédure correspondante: NewDoc(), Open()...

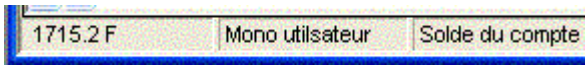
Comme d'habitude il suffit de double-cliquer sur la ToolBar pour faire apparaître
ToolBar1_ButtonClick

Voici le code complet :

```
Private Sub ToolBar1_ButtonClick(ByVal Sender As System.Object, ByVal e As
System.Windows.Forms.ToolBarButtonClickEventArgs) Handles toolBar1.ButtonClick
If e.Button Is NewButton Then
        NewDoc()
ElseIf e.Button Is OpenButton Then
        Open()
ElseIf e.Button Is SaveButton Then
        Save()
ElseIf e.Button Is PreviewButton Then
        PrintPreview()
...
End If
End Sub
```

Contrôle StatusBar

La barre d'état se trouve en bas de la fenêtre et **affiche des informations relatives aux opérations en cours**.



Dans la fenêtre des propriétés du StatusBar, **la collection Panels** contient les zones d'affichage du StatusBar.

Dans le code, pour modifier le texte d'une zone faire :

```
StatusBar1.Panels(0).Text="1715.2F"
```

3.14Les Images

Comment afficher des images ?

Le contrôle PictureBox

Le contrôle **PictureBox** sert à afficher des graphismes au format bitmap, GIF, JPEG, métafichier ou icône (Extension **.BMP .GIF .JPG .WMF .ICO**)

L'image affichée est déterminée par la propriété **Image**, laquelle peut être définie au moment de l'exécution ou du design. La propriété **SizeMode** détermine la façon dont l'image et le contrôle se dimensionnent l'un par rapport à l'autre.

On peut charger une image en mode conception ou dans le code :

```
PictureBox1.Image = Image.FromFile("vimage.gif)
```

(L'objet de la Classe Image charge une image d'un fichier puis l'affecte à la propriété Image.)
ou par
PictureBox1.Image.FromFile("vcar1.gif") **cette syntaxe ne marche pas!!!Pourquoi?**

Ho! Merveille, les GIF animés sont acceptées et s'animent sous VB.

Comment effacer une image?

```
If Not (PictureBox1.Image Is Nothing) Then
        PictureBox1.Image.Dispose()
        PictureBox1.Image = Nothing
End If
```

Les objets de la Classe Image ont comme d'habitude des propriétés et des méthodes.

La méthode **RotateFlip** permet par exemple d'effectuer une rotation de l'image, quand on tape le code, VB donne automatiquement la liste des paramètres possible.

```
PictureBox1.Image.RotateFlip(RotateFlipType.Rotate90FlipX)
```

La méthode **Save** sauvegarde l'image dans un fichier.

```
PictureBox1.Image.Save("c:\image.bmp")
```

Noter bien que le nom de l'extension suffit à imposer le format de l'image.

On peut charger une image .GIF puis la sauvegarder en .BMP

Il y a bien d'autres propriétés gérant les dimensions, la palette de l'image.

La propriété 'Image' des contrôles :

De nombreux contrôles Windows Forms peuvent afficher des images. L'image affichée peut être une icône symbolisant la fonction du contrôle ou une image ; par exemple, l'image d'une disquette sur un bouton indique généralement une commande d'enregistrement. L'icône peut également être une image d'arrière-plan conférant au contrôle une certaine apparence. Pour tous les contrôles affichant des images, l'image peut être définie à l'aide des propriétés Image ou BackgroundImage.

Pour affecter à la propriété Image ou BackgroundImage un objet de type System.Drawing.Image en général, vous utiliserez la méthode **FromFile** de la classe Image pour charger une Image à partir d'un fichier.

Exemple pour un bouton:

```
button1.Image = Image.FromFile("C:\Graphics\MyBitmap.bmp")
' Aligne l'image.
button1.ImageAlign = ContentAlignment.MiddleRight
```

Exemple pour un label:

```
Dim Label1 As New Label()
Dim Image1 As Image

Image1 = Image.FromFile("c:\\MyImage.bmp")

' modifier la taille du label pour qu'il affiche l'image.

Label1.Size = Image1.Size

' Mettre l'image dans le label.

Label1.Image = Image1
```

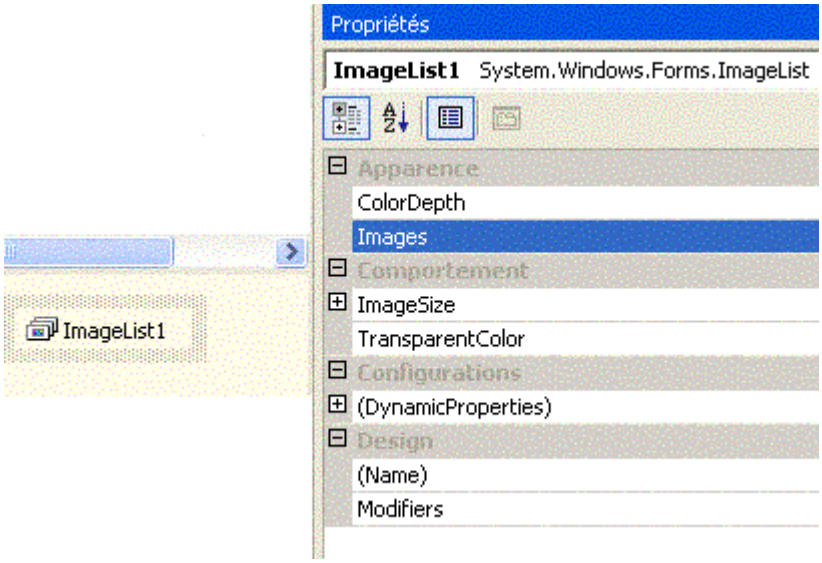
Si on renseigne la propriété Image, on ne peut pas utiliser en même temps la propriété ImageList décrite ci-dessous.

Le contrôle ImageList

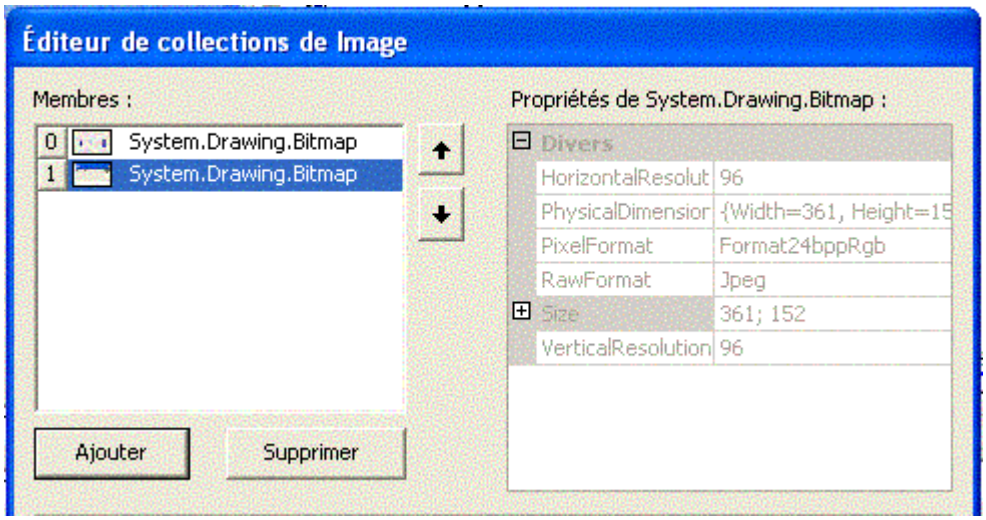
Il sert de containeur à image, c'est une **collection d'images**. les images qu'il contient seront utilisées par d'autres contrôles (PictureBox, Listview, TreeView, Button....)

Il n'est pas visible en exécution.

En conception il apparaît en bas sous la fenêtre. A droite figurent ses propriétés, en particulier, **la collection Images** qui contient les images et la propriété **TransparentColor** qui indique la couleur qui doit être transparent, c'est à dire non visible.



Si je clique sur le bouton en face de **Images**, **l'éditeur de collections d'image** s'ouvre.



On peut ajouter des images avec le bouton 'Ajouter'.
L'ImageList est ainsi chargée.

Ensuite pour utiliser une image de l'ImageList dans un autre contrôle, il faut modifier les propriétés de cet autre contrôle (un bouton par exemple).

La propriété **ImageList** du bouton doit contenir le nom du contrôle imageList et **ImageIndex** du bouton doit contenir l'index de l'image dans l'imageList.

```
btOk.ImageList = imagelist1
btOk.ImageIndex = 2
```

Un ImageList peut aussi être **chargée par code** :

```
imageList1.Images.Add(Image.FromFile(NomImage))
```

On ajoute à la collection Images une image venant d'un fichier nommé NomImage.

On peut surchargé la méthode Add en fournissant en plus la couleur transparente.

```
imageList1.Images.Add(Image.FromFile(imageToLoad), CouleurTransparente)
```

La taille des images peut aussi être modifiée par code :

```
imageList1.ImageSize = New Size(255, 255)
imageList1.TransparentColor = Color.White
```

Résumons
Révision pour y voir plus clair

3.30 Calcul de l'IMC. Révision++ Structuration des programmes

Ce chapitre permet de 'réviser' pas mal de notions.

Qu'est ce que l'IMC ?

L'index de masse corporelle est très utilisé par les médecins. Il est calculé à partir du poids et de la taille :
IMC=Poids/Taille au carré (avec Poids en Kg, Taille en mètres)

Cela permet de savoir si le sujet est :
- maigre (IMC<18.5)
- normal (IMC idéal=22)
- en surpoids (IMC>25)
- obèse (IMC>30).

On peut calculer le poids idéal par exemple PI= 22* T*T
Nous allons détailler ce petit programme :

Quel est le cahier des charges du programme ?

L'utilisateur doit pouvoir :
Saisir un poids, une taille, cliquer sur un bouton 'Calculer'

Les routines doivent :
Vérifier que l'utilisateur ne fait pas n'importe quoi.

Calculer et afficher les résultats : l'IMC mais aussi, en fonction de la taille, le poids idéal, les poids limites de maigreur, surpoids, obésité.

Création de l'interface

Il faut **2 zones de saisie** pour saisir le poids et la taille :
On crée 2 'TextBox' que l'on nomme :
- **TexBoxPoids**
- **TextBoxTaille**

(On laisse la propriété **Multiline** à **False**) pour n'avoir qu'une zone de saisie.

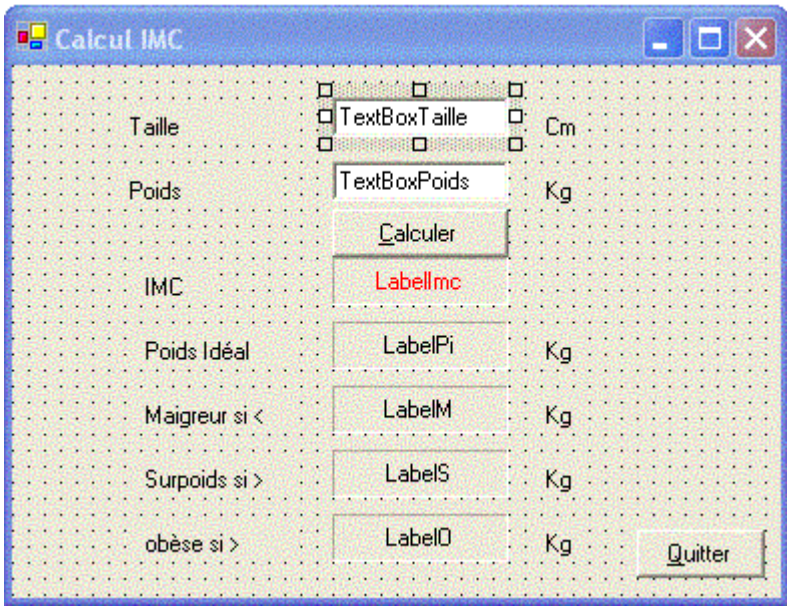
Pour afficher les résultats, on crée 5 'label' que l'on met les uns sous les autres. (Pour aller plus vite et que les labels fassent la même taille, on en crée un puis par un copier et des coller, on crée les autres) :
- **labelImc** 'pour afficher l'Imc
- **labelPi** 'pour afficher le poids idéal
- **labelM** 'pour afficher le poids limite de la maigreur.
- **labelS** 'pour afficher le poids limite du surpoids
- **labelO** 'pour afficher le poids limite de l'obésité.

Ensuite on ajoute **des labels** devant et derrière chaque TextBox et label pour indiquer devant, ce qu'ils contiennent et derrière, l'unité.

On ajoute 2 boutons :

- **ButtonCalcul**
- **ButtonQuitter**

Cela donne :



Pour faire beau :
La propriété Text de la fenêtre contient "Calcul IMC", pour afficher cela dans la barre de titre.
La propriété ForeColor de labelImc est en rouge.
La propriété BorderStyle des labels a la valeur 'Fixed3d' ce qui rend les bords visibles.

Ajout du Code

La procédure évènement Form1_Load qui s'effectue lorsque la fenêtre se charge **initialise** les zones d'affichage en les vidant :

```
Private Sub Form1_Load(..)
TextBoxTaille.Text = ""
TextBoxPoids.Text = ""
LabelImc.Text = ""
LabelPi.Text = ""
LabelM.Text = ""
LabelS.Text = ""
LabelO.Text = ""
End Sub
```

La procédure ButtonCalcul_Click qui se déclenche lorsque l'utilisateur clique sur le bouton 'Calculer' contient le code principal.

Voici la totalité du code, on le détaillera dessous.

```
Private Sub ButtonCalcul_Click(..)

Dim sPoids As Single 'Variable Single contenant le poids
Dim sTaille As Single 'Variable Single contenant la taille
```

```
'******Controle de validité des entrées***********

'Les valeurs saisies sont-elles numérique?
If Not (IsNumeric(TextBoxTaille.Text)) Then
        MsgBox("Entrez une valeur numérique pour la taille")
        Exit Sub
End If
If Not (IsNumeric(TextBoxPoids.Text)) Then
        MsgBox("Entrez une valeur numérique pour le poids")
        Exit Sub
End If

'Convertir les textes saisies en single
' et les mettre dans les variables
sTaille = CType(TextBoxTaille.Text, Single) / 100
sPoids = CType(TextBoxPoids.Text, Single)

'Les valeurs saisies sont-elles cohérentes?
If sTaille < 50 Or sTaille > 250 Then
        MsgBox("Entrez une taille valide")
        Exit Sub
End If
        If sPoids < 20 Or sPoids > 200 Then
        MsgBox("Entrez un poids valide")
Exit Sub
End If

'Effectuer les calculs et afficher les résultats.
LabelImc.Text = (Math.Round(sPoids / (sTaille * sTaille), 2)).ToString
LabelPi.Text = (Math.Round(22 * (sTaille * sTaille), 2)).ToString
LabelM.Text = (Math.Round(18.5 * (sTaille * sTaille), 2)).ToString
LabelS.Text = (Math.Round(25 * (sTaille * sTaille), 2)).ToString
LabelO.Text = (Math.Round(30 * (sTaille * sTaille), 2)).ToString
End Sub
```

Détaillons :

Quelles sont les différentes étapes ?

- On déclare les variables.
- On vérifie que ce qui a été tapé est numérique.
- On convertit le texte qui est dans la TextBox en Single
- On teste si les valeurs de poids et taille sont cohérentes.
- On fait le calcul et on affiche.

Déclaration de variables.

```
Dim sPoids As Single 'Variable Single contenant le poids
Dim sTaille As Single 'Variable Single contenant la taille
```

Ce sont des variables 'privées' propres à la procédure (utilisation de Dim ou Private).

Contrôle de validité :
L'utilisateur est sensé avoir tapé un poids et une taille puis cliqué sur le bouton 'Calculer'. Mais **il ne faut absolument pas lui faire confiance :** il a peut-être oublié de taper le poids ou a donner une taille=0 (l'ordinateur n'aime pas diviser par 0!!), il a peut-être fait une faute de frappe et tapé du texte!!...

Donc il faut tester ce qui a été tapé, s'il y a erreur, on prévient l'utilisateur avec une 'MessageBox' puis on sort de la routine par (Exit Sub) sans effectuer de calculs.

Ici par exemple, on teste si le texte saisi dans la zone taille n'est pas numérique :

```
If Not (IsNumeric(TextBoxTaille.Text)) Then
    MsgBox("Entrez une valeur numérique pour la taille")
    Exit Sub
End If
```

Amélioration : On aurait pu automatiquement effacer la valeur erronée et placer le curseur dans la zone à ressaisir :

```
If Not (IsNumeric(TextBoxTaille.Text)) Then
    MsgBox("Entrez une valeur numérique pour la taille")
    TextBoxTaille.Text=""
    TextBoxTaille.Select()
    Exit Sub
End If
```

Conversion :
Si le texte est bien 'Numéric', on fait la conversion en réel simple précision (Single)

```
sTaille = CType(TextBoxTaille.Text, Single) / 100
```

On utilise CType pour convertir une String en Single.
On divise taille par 100 car l'utilisateur à saisie la taille en centimètres et les formules nécessite une taille en mètres.

Problème du séparateur décimal dans les saisies.
Pourquoi saisir la taille en Cm? C'est pour éviter d'avoir à gérer le problème du séparateur décimal si la taille est saisie en mètres: L'utilisateur va t-il taper "1.75" ou "1,75" si je lui demande des mètres.

On rappelle que pour convertir un texte en Single VB accepte le point et pas la virgule.

Pour ma part voici ma solution : si je demandais des mètres j'ajouterais en début de routine une instruction transformant les ',' en '.' :

```
TextBoxTaille.Text = Replace(TextBoxTaille.Text, ",", ".")
```

Faire les calculs et afficher les résultats.

Je fais le calcul:

```
sPoids / (sTaille * sTaille)
```

J'arrondis à 2 décimales après la virgule grâce à Math.Round(,2):

```
Math.Round(sPoids / (sTaille * sTaille), 2)
```

Je convertis en String:

```
(Math.Round(sPoids / (sTaille * sTaille), 2)).ToString
```

J'affiche dans le label 'labelImc':

```
LabelImc.Text = (Math.Round(sPoids / (sTaille * sTaille), 2)).ToString
```

(J'aurais pu aussi ne pas arrondir le calcul mais formater l'affichage pour que 2 décimales soient affichées)

La procédure ButtonQuitter_Click déclenchée quand l'utilisateur clique sur le bouton 'Quitter' ferme la seule fenêtre du projet (c'est Me , celle ou on se trouve), ce qui arrête le programme.

```
Private Sub ButtonQuitter_Click()
    Me.Close()
End Sub
```

Structuration

Ici on a fait simple : une procédure évènement calcul et affiche les résultats.
On aurait pu, dans un but didactique **'structurer' le programme**.

On aurait pu découper le programme en procédure :
- Une procédure faisant le calcul.
- Une procédure affichant les résultats.

Pour les variables il y a dans ce cas 2 possibilités :
- Mettre les variables en 'Public' dans un module Standard.
- Utiliser des variables privées et les passer en paramètres.

Première solution : Variables 'Public'
Créer dans un module standard des variables Public pour stocker les variables Poids et Taille, résultats (Public sIMC A Single par exemple), créer dans ce même module standard une procédure Public nommée 'Calculer' qui fait les calculs et met les résultats dans les variables public, enfin dans le module de formulaire créer une procédure 'AfficheResultat' affichant les résultats.

Module standard :

```
'Déclaration de variables Public
Public sPoids As Single
Public sTaille As Single
Public sIMC A Single
..
'Procedure Public de calcul
Public Sub Calculer
        sIMC=Math.Round(sPoids / (sTaille * sTaille), 2)
        ...
End Sub
```

Module de formulaire Form1 :

```
'Procédure évènement qui appelle les divers routines
Private Sub ButtonCalculer_Click
        ...
        sTaille = CType(TextBoxTaille.Text, Single) / 100
        Calculer()    'Appelle la routine de calcul
        AfficheResultat() 'Appelle la routine d'affichage
End Sub

'routine d'affichage
Private Sub AfficheResultat()

        LabelImc.Text = sIMC.ToString
        ...
End Sub
```

On voit bien que la routine de Calcul est générale et donc mise dans un module standard et d'accès 'Public', alors que la routine d'affichage affichant sur Form1 est privée et dans le module du formulaire.

Seconde solution : Variables 'Privées' et passage de paramètres
On aurait pu ne pas créer de variables 'public' mais créer des fonctions (CalculIMC par exemple) à qui on enverrait en paramètre le poids et la taille et qui retournerait le résultat du calcul. Une procédure AfficheResultatIMC recevrait en paramètres la valeur de l'IMC à afficher.
Module standard :

```
'Pas de déclaration de variables Public
..
'Function Public de calcul: reçoit en paramètre le poids et la taille
'retourne l'Imc
Public Function CalculerIMC (T as Single, P As Single) As Single
        Return Math.Round(P / (T*T), 2)
End Sub
```

Module de formulaire Form1 :

```
'Procédure évènement qui appelle les divers routines
Private Sub ButtonCalculer_Click
        ...
        sTaille = CType(TextBoxTaille.Text, Single) / 100

        'Appelle de la routine calcul avec l'envoie de paramètres sPoids et sTaille
        'Au retour on a la valeur de L'imc que l'on envoie à la routine d'affichage.
        AfficheResultatIMC(CalculerIMC(sTaille, sPoids)) 'Appelle la routine d'affichage
End Sub

'routine d'affichage
Private Sub AfficheResultatIMC(I As Single)
        LabelImc.Text = i.ToString
End Sub
```

Remarque :
La ligne `AfficheResultatIMC(CalculerIMC(sTaille, sPoids))` est équivalente à :

```
Dim s As single
s=(CalculerIMC(sTaille, sPoids)
AfficheResultatIMC(s))
```

mais on se passe d'une variable temporaire.

Conclusion :
Faut-il travailler avec des variables Public ou passer des paramètres ?

Réponses :
A mon avis, les 2 et "ça dépend"!!!(Bien la réponse).

Et votre avis ?

3.31 Ordre des instructions

Dans quel ordre écrire dans un module.

Contenu des modules

Le code Visual Basic est stocké dans des modules (modules de formulaires, module de classe ...), chaque module est dans un fichier ayant l'extension '.vb". Les projets sont composés de fichiers, lesquels sont compilés pour créer des applications.

Respecter l'ordre suivant :

1. Instructions **Option** toujours en premier (force des contraintes de déclaration de variables, de conversion de variables, de comparaison).
2. Instructions **Imports** (charge des espaces de noms)
3. Procédure **Main** (la procédure de démarrage si nécessaire)
4. Instructions **Class**, **Module** et **Namespace**, le cas échéant

Exemple :

```
Option Explicit On
Imports System.AppDomain
Imports Microsoft.VisualBasic.Conversion

Public Class Form1
Inherits System.Windows.Forms.Form
        Dim WithEvents m As PrintDocument1

#Region " Code généré par le Concepteur Windows Form "

        Public d As Integer
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles End Sub
        Dim A As integer
End Class
```

On remarque de nouveau l'importance de l'endroit ou les variables sont déclarées : Dans notre exemple A est accessible uniquement dans Form_Load, alors que d est public.

Si vous entrez les instructions dans un ordre différent, vous risquez de créer des erreurs de compilation.

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 Period
```

Ce qu'il faut savoir pour faire un vrai 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

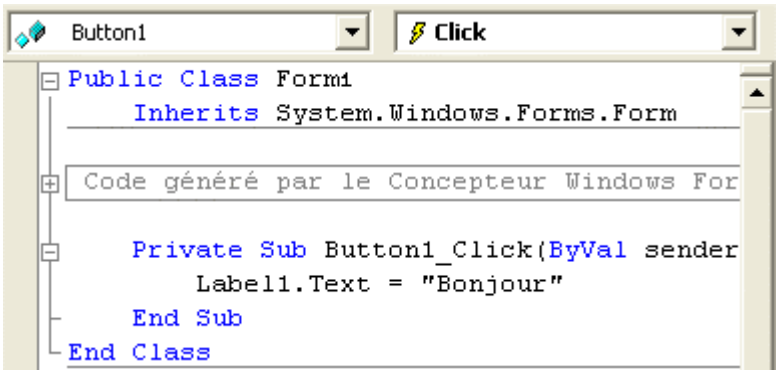
Comment à partir d'un formulaire Form1 ouvrir un second formulaire Form2 ?

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','Private'... Ce qui permet de gérer son accessibilité.

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

Pour ajouter une méthode à un formulaire, il faut créer une Sub 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 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 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, 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 Publique (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.

```
Label1.Texte() = "12"
```

'Texte' n'est pas un membre de 'System.Windows.Forms.Label'.

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 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 de multiples 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 en déterminer la cause et la corriger.

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

Les erreurs les plus communes sont :

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

Quelques règles permettent de les éviter : voir leçon 7.2

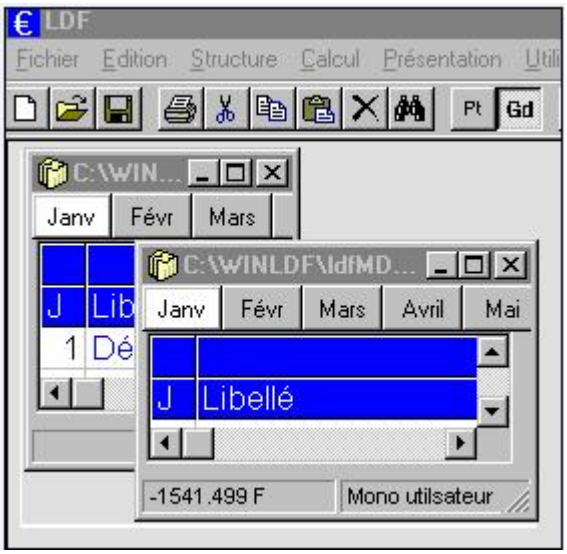
4.4 Travailler sur une fenêtre multidocument

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 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 active, on sait simplement que la fenêtre fille active est dans **MIDFilleActive**, variable que l'on utilise 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
...
Nex
t
```

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 :

Nom	Taille	Type	Date de modification
2035.gif	10 Ko	Image GIF	25/11/2002 18:55
acceuil.gif	1 Ko	Image GIF	27/12/2001 23:42
ampoule.gif	2 Ko	Image GIF	11/04/2002 22:34

Un fichier est composé d'**enregistrements** qui sont des 'paquets' de données; suivant le type de 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.

Philippe
Jean-François
Louis

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 3éme.

- **l'accès binaire**, pour lire 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 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 celui-ci est déjà ouvert par un autre programme, il se déclenche généralement une erreur (sauf si on l'ouvre en Binaire, c'était le cas en VB6, c'est à vérifier en VB.NET).

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

Paramètres de FileOpen :
FileNumber

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.

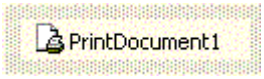
Pour imprimer il faut utiliser la méthode Print de ce composant PrintDocument, Il faut donc écrire l'instruction suivante :

```
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 'ImprimerHello' cela affiche un gros 'Hello'.


La méthode **Print** d'un **PrintDocument** déclenche l'évènement **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 ?** M**odifier 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 :

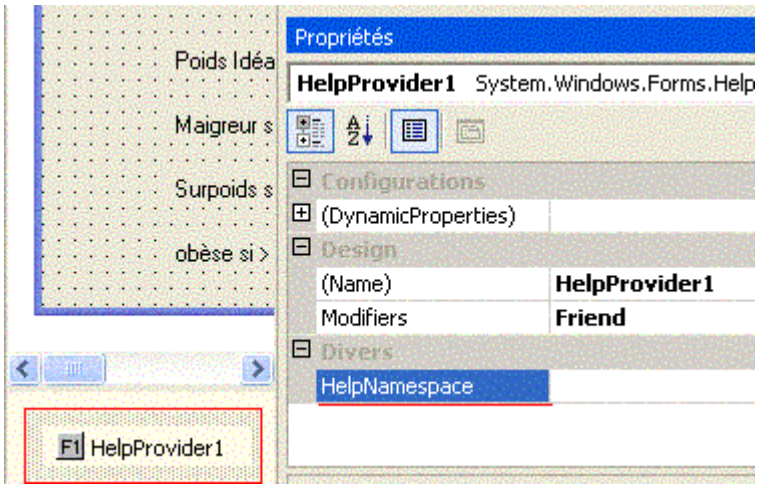
Vous pouvez utiliser le composant HelpProvider pour **attacher des rubriques d'aide** figurant dans un fichier d'aide (au format HTML, HTMLHelp 1.x ou ultérieur) **à des contrôles** spécifiques.

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).

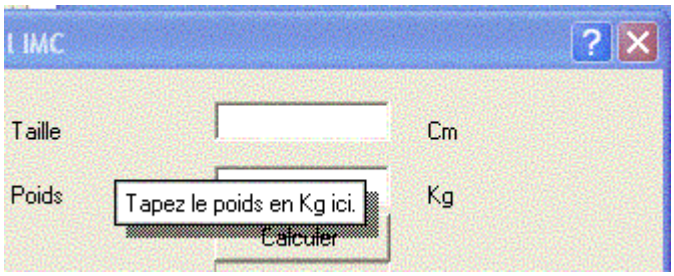
Dans le code il faut utiliser la syntaxe HelpProvider.SetHelpKeyword="..."

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 sur**F5**.
Appuyez sur le bouton Aide (?) de la barre de titre et cliquez sur le contrôle dont vous avez

défini la propriété **HelpString**. Le toolTip apparait.

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")
```