

SL228_POBJ Visual C#

Chapitre 3

Contrôles, propriétés et événements

Christian HUBER (CHR)
Version 1.0 février 2016

CONTENU DU CHAPITRE 3

3. Contrôles, propriétés et événements	3-1
3.1. Objectifs	3-1
3.2. Aide à disposition	3-1
3.3. Introduction aux contrôles	3-2
3.3.1. Nature d'un control	3-3
3.3.2. Accès aux élément du contrôle depuis le code	3-4
3.3.3. Propriétés communes a plusieurs contrôles	3-5
3.4. Classement des contrôles par fonction	3-6
3.5. Les Contrôles Windows Forms	3-14
3.5.1. Le contrôle Label	3-14
3.5.1.1. Label, propriété Text	3-14
3.5.1.2. Label, événement par défaut	3-14
3.5.2. Le contrôle Button	3-15
3.5.2.1. Button, événement par défaut	3-15
3.5.3. Le contrôle TextBox	3-16
3.5.3.1. TextBox, propriété Text	3-16
3.5.3.2. TextBox, événement par défaut	3-16
3.5.3.3. Saisie et affichage d'une valeur numérique	3-16
3.5.3.4. Affichage sur plusieurs lignes	3-17
3.5.3.5. TextBox, propriété Lines	3-18
3.5.3.6. Initialisation par les propriétés	3-18
3.5.3.7. Initialisation par le code de la propriété Lines	3-18
3.5.3.8. Initialisation par le code de la propriété Text.	3-19
3.5.3.9. Lecture d'un texte multi ligne	3-19
3.5.4. Le contrôle CheckBox	3-20
3.5.4.1. CheckBox, Evénement par défaut	3-20
3.5.4.2. CheckBox, propriété Checked	3-20
3.5.4.3. CheckBox, propriété ChecState	3-20
3.5.4.4. Exemple situation Checked et CheckState	3-20
3.5.4.5. CheckBox, propriété Enabled	3-22
3.5.5. Le contrôle RadioButton	3-23
3.5.5.1. RadioButton, propriété Checked	3-23
3.5.5.2. RadioButton, propriété Appearance	3-24
3.5.5.3. RadioButton, Evénement par défaut	3-24
3.5.5.4. RadioButton, Exemple	3-25
3.5.5.5. Désactivation d'un bouton d'option	3-25
3.5.5.6. Besoin des de groupes de RadioButton	3-26
3.5.6. Le contrôle GroupBox	3-28
3.5.7. Le contrôle PictureBox	3-29
3.5.7.1. La propriété Image	3-29
3.5.7.2. La propriété SizeMode	3-29
3.5.7.3. Chargement d'une image au moment du design	3-31
3.5.7.4. Effacement d'une image au moment du design	3-35
3.5.7.5. Chargement d'une image en exécution depuis les ressources	3-36

3.5.7.6.	Chargement d'une image en exécution, new BitMap	3-37
3.5.7.7.	Effacement d'une image en exécution	3-38
3.5.8.	Le contrôle ImageList	3-39
3.5.8.1.	Ajouter ou supprimer des images dans le Concepteur	3-39
3.5.8.2.	Ajouter des images par programme	3-40
3.5.8.3.	Pour supprimer des images par programme	3-40
3.5.8.4.	Dimension et résolution des images	3-40
3.5.8.5.	Redimensionnement des images	3-41
3.5.9.	Exemple utilisation de la liste d'image	3-42
3.5.11.	Le contrôle Panel	3-44
3.5.11.1.	Propriété BorderStyle	3-44
3.5.12.	Le contrôle NumericUpDown	3-45
3.5.12.1.	Propriétés principales	3-45
3.5.12.2.	NumericUpDown, Evénement par défaut	3-45
3.5.12.3.	NumericUpDown, Exemple	3-46
3.5.12.4.	Le type decimal	3-46
3.5.13.	Le contrôle TrackBar	3-47
3.5.13.1.	Propriétés principales	3-47
3.5.13.2.	Propriétés TickStyle	3-47
3.5.13.3.	TrackBar, Evénements	3-48
3.5.13.4.	TrackBar, Exemple	3-48
3.5.13.5.	Ajout manuel d'un événement	3-50
3.5.14.	Le contrôle ProgressBar	3-51
3.5.14.1.	Propriétés principales	3-51
3.5.14.2.	Propriétés Style	3-51
3.5.14.3.	ProgressBar, Evénement par défaut	3-51
3.5.14.4.	ProgressBar Exemple	3-52
3.5.15.	Le contrôle ListBox	3-54
3.5.15.1.	Ajout/suppression d'éléments de la liste par programme	3-54
3.5.15.2.	ListBox, propriété Items	3-54
3.5.15.3.	Effet des propriétés sur la ListBox	3-55
3.5.15.1.	ListBox, Evénement par défaut	3-55
3.5.15.2.	ListBox, Exemple	3-56
3.5.16.	Le contrôle ComboBox	3-57
3.5.16.1.	ComboBox, propriété DropDownStyle	3-57
3.5.16.2.	ComboBox, événement par défaut	3-57
3.5.16.3.	ComboBox, initialisation de la liste à la construction	3-58
3.5.17.	Le contrôle DomainUpDown	3-59
3.5.17.1.	DomainUpDown, événement par défaut	3-59
3.5.17.2.	Contrôle DomainUpDown, exemple	3-60
3.5.18.	ComboBox ou ListBox ?	3-61
3.5.19.	Ajout et suppression d'éléments d'une ListBox, ComboBox ou DomainUpDown	3-62
3.5.19.1.	Pour ajouter des éléments	3-62
3.5.19.2.	Pour enlever un élément	3-63
3.5.19.3.	Pour supprimer tous les éléments	3-64
3.5.20.	Accès aux éléments d'une Liste	3-64
3.5.20.1.	La propriété SelectedIndex	3-64
3.5.20.2.	La propriété SelectedItem	3-64
3.5.20.3.	La propriété Items.Count	3-65
3.5.20.4.	Utilisation de Items comme un tableau	3-65
3.6.	Les méthodes événementielles	3-66
3.6.1.	Démarrage de la Feuille	3-66
3.6.2.	Event Click	3-66

3.6.3.	Event Change	3-66
3.6.4.	Sélection d'un événement spécifique	3-67
3.7.	Exemple : convertisseur de devises	3-68
3.7.1.	Configuration du ComboBox	3-68
3.7.2.	Configuration des TextBox	3-68
3.7.2.1.	TextBox en ReadOnly	3-69
3.7.3.	Configuration des Labels	3-69
3.7.4.	Configuration du Form	3-69
3.7.5.	Configuration du Button	3-69
3.7.6.	Code lié au Button	3-69
3.7.7.	Code complet de l'exemple	3-70
3.8.	Historique	3-72
3.8.1.	Version 1.0 février 2016	3-72

3. CONTROLES, PROPRIETES ET EVENEMENTS

Ce chapitre a pour but d'approfondir l'usage des contrôles, leurs propriétés et les événements pouvant les activer.

Ce chapitre s'appuie en grande partie sur la section de l'aide *Controls to use on Windows Forms* qui est une partie de la section *Contrôles Windows Forms* du chapitre Visual Basic et Visual C#.

Les contrôles décrits dans ce chapitre dérivent de la classe de base Windows Forms.

3.1. OBJECTIFS

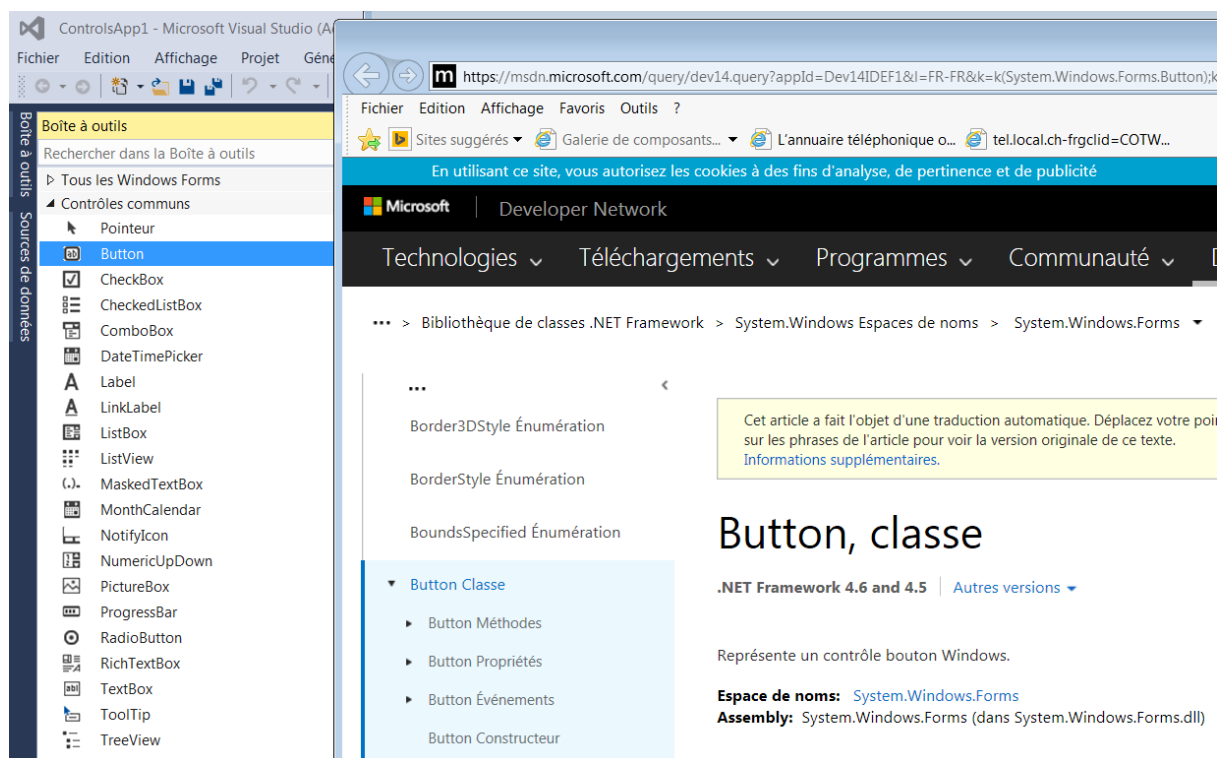
A l'issue de ce chapitre, l'étudiant sera capable de :

- Placer des contrôles dans une feuille
- Modifier les propriétés de ces contrôles
- Lire et modifier depuis le code les propriétés d'un contrôle
- Ajouter un traitement lié à un événement d'un contrôle

3.2. AIDE A DISPOSITION

Le plus simple pour obtenir de l'aide sur un control, c'est de sélectionner le control dans la boîte à outils et de presser sur la touche F1.

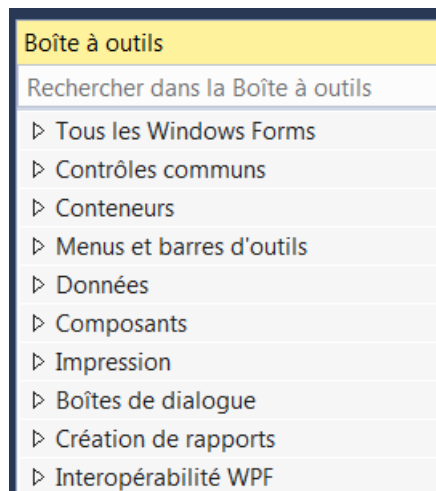
Exemple :



3.3. INTRODUCTION AUX CONTROLES

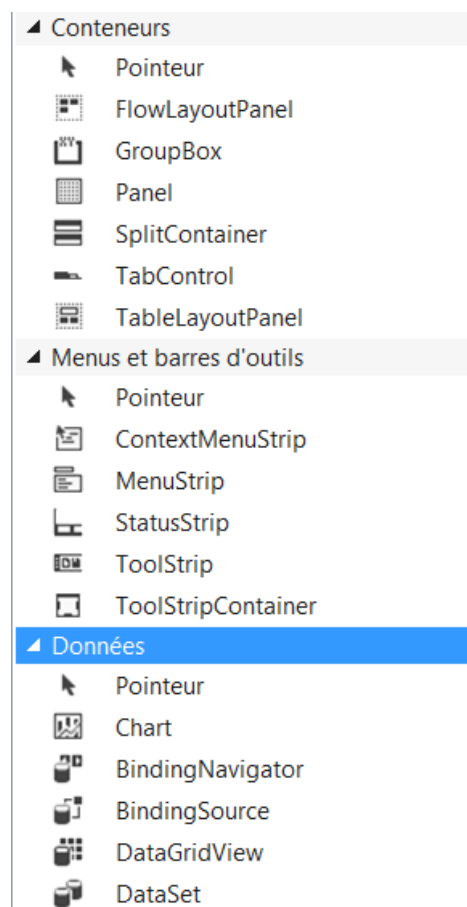
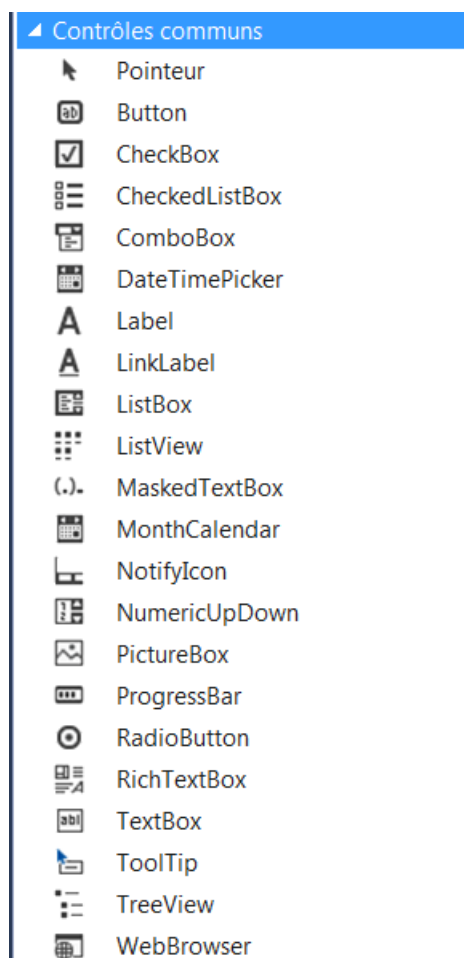
La boîte à outils du Visual C# 2015 contient les outils que vous pouvez utiliser pour dessiner des contrôles sur vos feuilles.

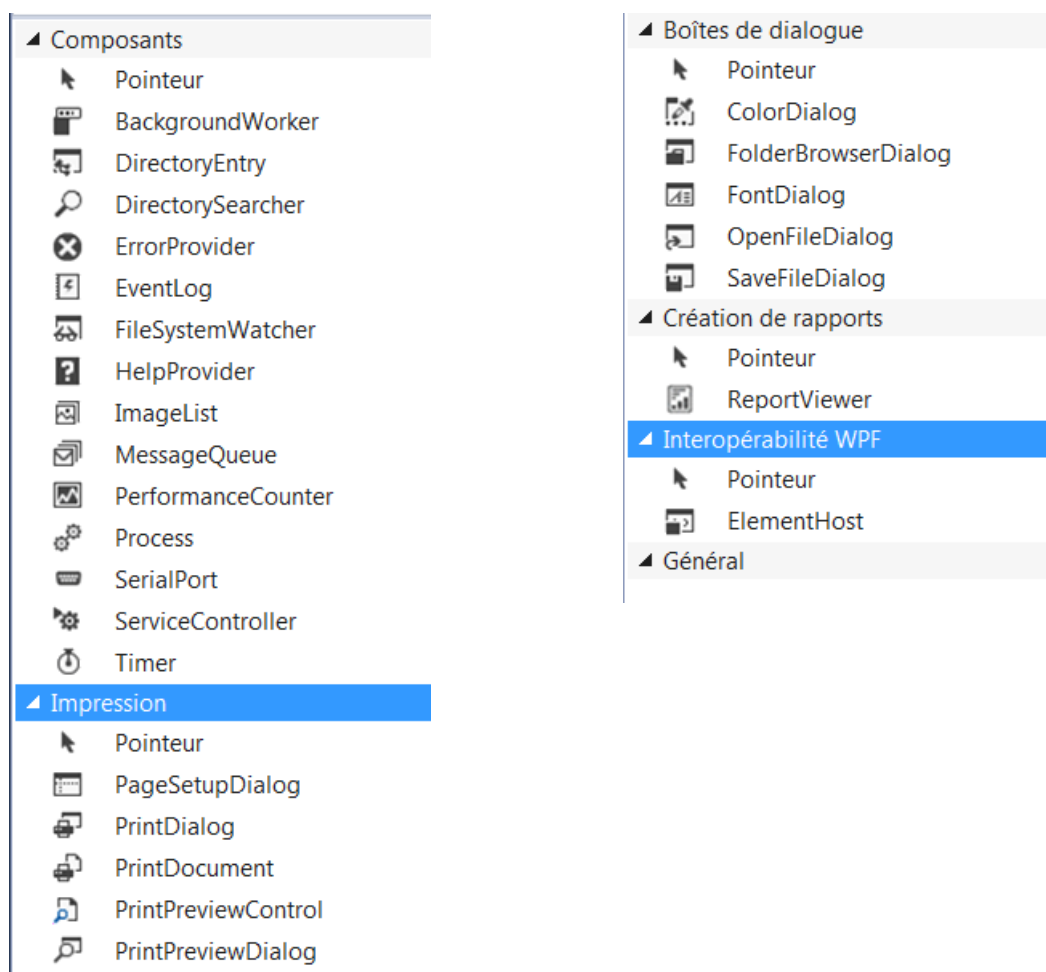
Voici les outils classé par catégories :



Tous les Windows Forms, n'est pas une catégorie, elle fournit l'ensemble des contrôles classé par ordre alphabétique.

Voici de détails de chaque catégorie :





La liste est déjà importante et ne correspond qu'à la liste dite Windows Forms. Il faut savoir qu'il est encore possible d'ajouter des composants externes ou de les créer soit même.

Ce chapitre traite essentiellement des contrôles du groupe **Contrôles communs**.

3.3.1. NATURE D'UN CONTROL

Un contrôle est un objet ayant essentiellement des propriétés ou attributs permettant de modifier l'apparence, la position, la couleur, le texte affiché, la police, etc... .

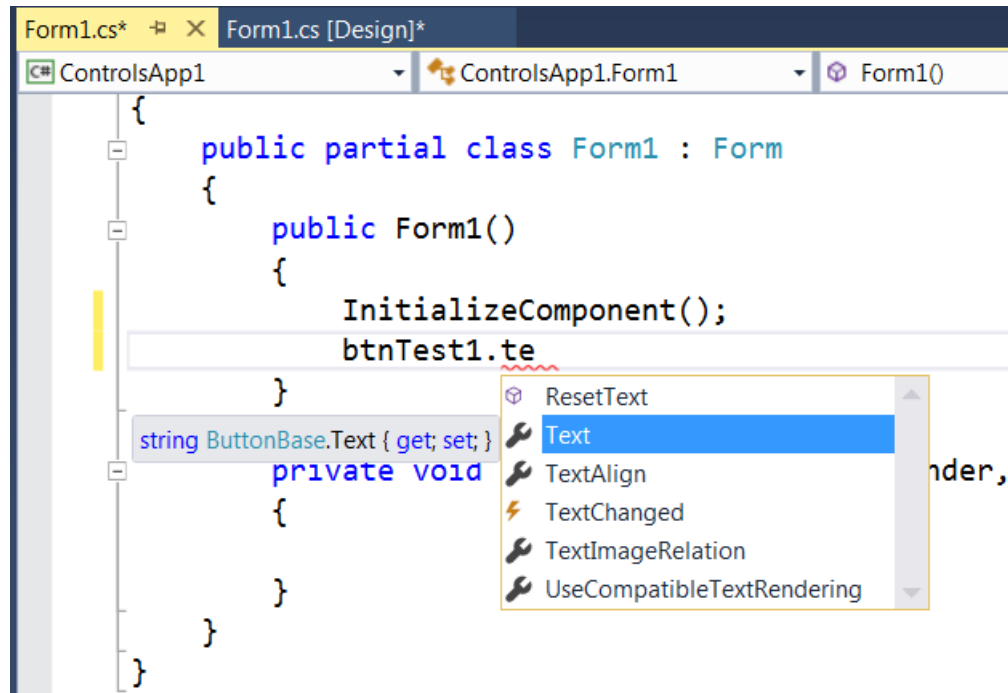
Les propriétés peuvent être établies lors de la conception du contrôle. Lorsque l'on sélectionne un contrôle dans la fenêtre de conception, les propriétés sont affichées dans la fenêtre de propriété.

Nous verrons par la suite qu'un contrôle possède aussi des méthodes (fonctions) et aussi des événements.

3.3.2. ACCES AUX ELEMENT DU CONTROLE DEPUIS LE CODE




Dans la partie code du Visual C#, lorsque l'on écrit le nom du contrôle (propriété (**Name**)) et que l'on ajoute un point (.) on obtient une liste déroulante de toutes les propriétés, méthodes et événement du contrôle.

Par exemple pour un contrôle Button appelé btnTest1:



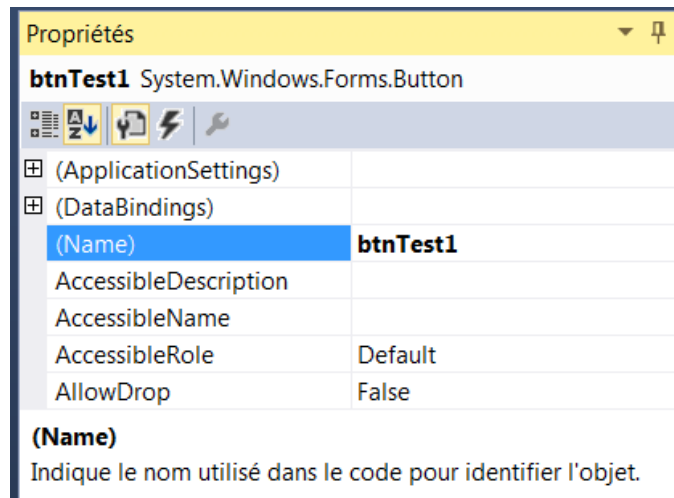
L'introduction de lettres après le . (.te) dans l'exemple ci-dessus positionne dans la liste. Lorsque l'élément est sélectionné dans la liste, une action sur la touche Tab insère automatiquement le nom du champ.

On découvre les indications graphique :

-  méthode
-  propriété ou attribut
-  événement

3.3.3. PROPRIETES COMMUNES A PLUSIEURS CONTROLES

Voici quelques propriétés que l'on retrouve dans la plus part des contrôles :



Il est possible de modifier l'organisation de l'affichage en sélectionnant :



par catégorie



classement alphabétique



permet d'obtenir la liste des événements



permet de revenir à la liste des propriétés.

Propriétés	Description
Apparence	
BackColor	Spécifie la couleur de l'arrière-plan du composant. Choix dans une palette accessible en ouvrant la liste déroulante de la propriété.
Cursor	Le curseur qui s'affiche lorsque la souris passe sur le contrôle
Font	Spécifie la police et la taille des caractères.
ForeColor	Spécifie la couleur de premier plan du Contrôle (Foreground Color)
Image	L'image qui sera affichée sur le control.
Text	Contient le texte affiché sur le contrôle
TextAlign	Détermine l'alignement du texte affiché sur le contrôle.

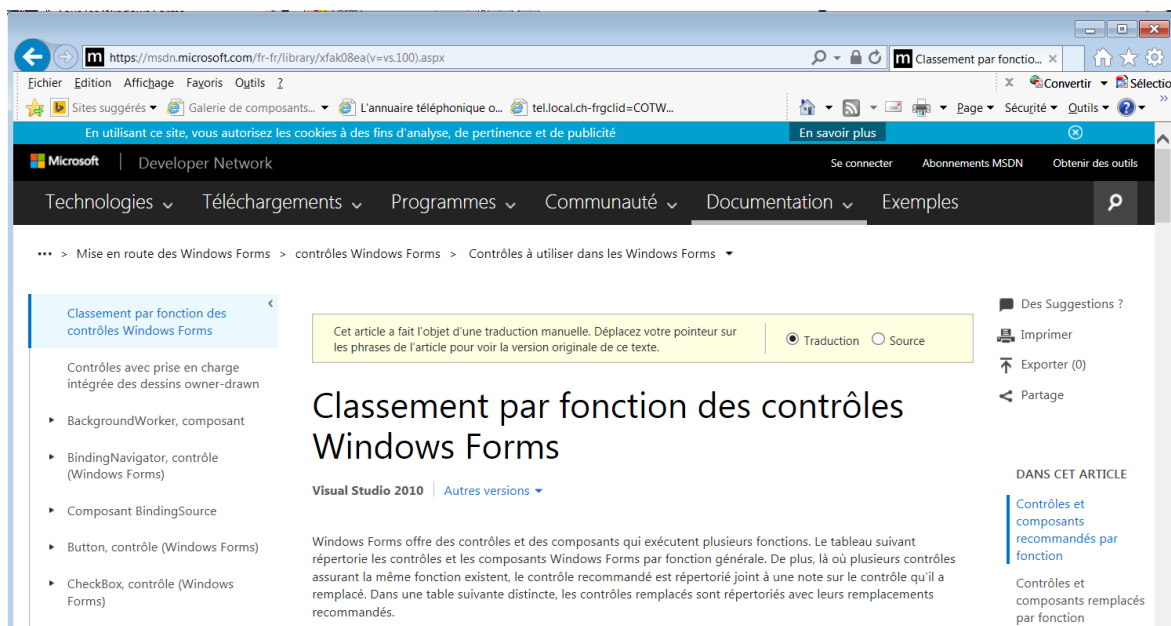
Comportement	
Enabled	Permet d'activer et de désactiver les contrôles selon qu'on les veut disponibles ou non lors de l'exécution.
Visible	si true le contrôle est visible, si false le contrôle est masqué.

Design	
(Name)	Spécifie le nom du contrôle, c'est le nom utilisé pour identifier l'objet dans le code.

<i>Disposition</i>	
Anchor	Top, Left. Les ancrs définissent à quel bord du conteneur le contrôle est lié.
Location	Position X et Y du contrôle
Size	Dimensions Width et Height du contrôle

3.4. CLASSEMENT DES CONTROLES PAR FONCTION

Le tableau suivant répertorie les contrôles Windows Forms selon leur fonction principale. Il est obtenu de la section **Classement par fonction des contrôles Windows Forms**.



la recherche conduit à une information Visual Studio 2010 !



Function	Control	Description
Affichage des données	DataGridView control	<p>The DataGridView control provides a customizable table for displaying data. La classe DataGridView active la personnalisation des cellules, lignes, colonnes et bordures.</p> <p>Remarque Le contrôle DataGridView fournit de nombreuses fonctionnalités de base et avancées qui sont absentes du contrôle DataGrid. Pour plus d'informations, consultez Differences Between the Windows Forms DataGridView and DataGrid Controls</p>


Function	Control	Description
Liaison de données et navigation	Composant BindingSource	Simplifie la liaison entre les contrôles sur un formulaire et les données en assurant la gestion des devises, la notification des modifications et d'autres services.
	Contrôle BindingNavigator	Fournit une interface de type de barre d'outils pour naviguer et manipuler des données sur un formulaire.
Édition de texte	Contrôle TextBox	Affiche un texte entré au moment du design et pouvant être modifié par les utilisateurs au moment de l'exécution ou par programme.
	Contrôle RichTextBox	Permet l'affichage du texte au format texte brut ou RTF.
	Contrôle MaskedTextBox	Contraint le format d'entrée d'utilisateur
Affichage d'informations (lecture seule)	Contrôle Label	Affiche du texte que les utilisateurs ne peuvent pas directement modifier.
	Contrôle LinkLabel	Affiche le texte sous la forme d'un lien de style Web et déclenche un événement lorsque l'utilisateur clique sur le texte spécial. En général, le texte est un lien vers une autre fenêtre ou vers un site Web.


Function	Control	Description
	Contrôle StatusStrip	Affiche des informations sur l'état actuel de l'application dans une zone à frame généralement située au bas d'un formulaire parent.
	Contrôle ProgressBar	Affiche la progression actuelle d'une opération pour l'utilisateur.
Affichage de page Web	Contrôle WebBrowser	Permet à l'utilisateur de naviguer dans des pages Web à l'intérieur de votre formulaire.
Sélection dans une liste	Contrôle CheckedListBox	Affiche une liste déroulante d'éléments accompagnés chacun d'une case à cocher.
	Contrôle ComboBox	Affiche une liste déroulante d'éléments.
	DomainUpDown control	Affiche une liste d'éléments que les utilisateurs peuvent faire défiler vers le haut ou le vers le bas à l'aide de boutons.
	Contrôle ListBox	Affiche une liste d'éléments texte et graphiques (icônes).
	Contrôle ListView	Affiche des éléments dans une vue parmi quatre vues différentes. Les différents types de vue sont : texte seul, texte avec petites icônes, texte avec grandes icônes et détails.

Function	Control	Description
	Contrôle ListView	Affiche des éléments dans une vue parmi quatre vues différentes. Les différents types de vue sont : texte seul, texte avec petites icônes, texte avec grandes icônes et détails.
	Contrôle NumericUpDown	Affiche une liste de nombres que les utilisateurs peuvent faire défiler vers le haut ou le vers le bas à l'aide de boutons.
	Contrôle TreeView	Affiche une collection hiérarchique d'objets nœud qui peut être constituée de texte éventuellement associé à des cases à cocher ou des icônes.
Affichage des graphismes	Contrôle PictureBox	Affiche dans un frame des fichiers graphiques tels qu'images bitmap et icônes.
Stockage des graphismes	Contrôle ImageList	Serves as a repository for images. ImageList controls and the images they contain can be reused from one application to the next.
Définition de valeurs	Contrôle CheckBox	Affiche une case à cocher et une étiquette pour le texte. Généralement utilisé pour définir des options.
	Contrôle CheckedListBox	Affiche une liste déroulante d'éléments accompagnés chacun d'une case à cocher.
	Contrôle RadioButton	Affiche une case d'option qui peut être activée ou désactivée.

Function	Control	Description
	TrackBar control	Permet aux utilisateurs de définir des valeurs sur une échelle par le déplacement d'un curseur.
Définition de dates	Contrôle DateTimePicker	Affiche un calendrier graphique permettant aux utilisateurs de sélectionner une date ou une heure.
	Contrôle MonthCalendar	Affiche un calendrier graphique permettant aux utilisateurs de sélectionner une plage de dates.
Boîtes de dialogue	Contrôle ColorDialog	Affiche la boîte de dialogue du sélecteur de couleurs qui permet aux utilisateurs de définir la couleur d'un élément d'interface.
	Contrôle FontDialog	Affiche une boîte de dialogue qui permet aux utilisateurs de définir une police et ses attributs.
	Contrôle OpenFileDialog	Affiche une boîte de dialogue qui permet aux utilisateurs de naviguer et de sélectionner un fichier.
	Contrôle PrintDialog	Affiche une boîte de dialogue qui permet aux utilisateurs de sélectionner une imprimante et de définir ses attributs.
	Contrôle PrintPreviewDialog	Affiche une boîte de dialogue qui montre un composant PrintDocument de contrôle tel qu'il se présentera une fois imprimé.

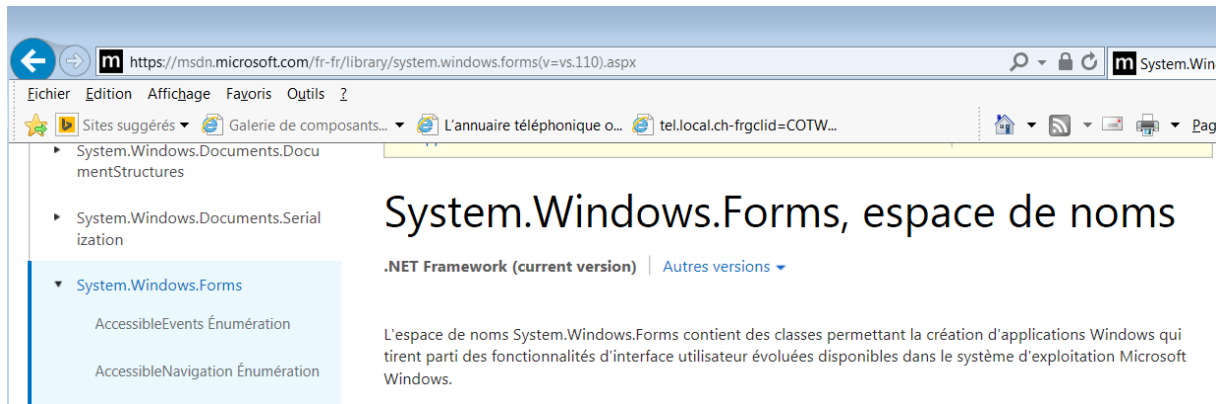
Function	Control	Description
	Contrôle FolderBrowserDialog	Affiche une boîte de dialogue qui permet aux utilisateurs de parcourir, créer et éventuellement sélectionner un dossier
	Contrôle SaveFileDialog	Affiche une boîte de dialogue qui permet aux utilisateurs d'enregistrer un fichier.
Contrôles de menu	Contrôle MenuStrip	<p>Crée des menus personnalisés.</p> <p> Remarque The MenuStrip is designed to replace the MainMenu control.</p>
	Contrôle ContextMenuStrip	<p>Crée des menus contextuels personnalisés.</p> <p> Remarque The ContextMenuStrip is designed to replace the ContextMenu control.</p>
Commands	Contrôle Button	Démarre, arrête ou interrompt un processus.
	LinkLabel control	Affiche le texte sous la forme d'un lien de style Web et déclenche un événement lorsque l'utilisateur clique sur le texte spécial. En général, le texte est un lien vers une autre fenêtre ou vers un site Web.
	Contrôle NotifyIcon	Affiche dans la zone d'état de la barre des tâches une icône qui représente une application s'exécutant en arrière-plan.

Function	Control	Description
	Contrôle ToolStrip	<p>Crée des barres d'outils qui peuvent avoir une apparence ou un comportement de type Microsoft Windows XP, Microsoft Office, Microsoft Internet Explorer ou personnalisé, avec ou sans thèmes, et avec prise en charge du dépassement de capacité et le reclassement des éléments au moment de l'exécution.</p> <p> Remarque Le contrôle ToolStrip est conçu pour remplacer le contrôle ToolBar.</p>
Aide utilisateur	Composant HelpProvider	Fournit une aide contextuelle ou en ligne pour les contrôles.
	Composant ToolTip	Fournit une fenêtre indépendante qui affiche une brève description de la fonction d'un contrôle lorsque l'utilisateur place le pointeur au-dessus du contrôle.
Groupement d'autres contrôles	Contrôle Panel	Groupe plusieurs contrôles à l'intérieur d'un frame déroulant dépourvu d'étiquette.
	Contrôle GroupBox	Groupe plusieurs contrôles (tels que des cases d'option) dans un frame non déroulant doté d'une étiquette.
	Contrôle TabControl	Fournit une page d'onglets permettant d'organiser efficacement les objets groupés et d'y accéder.

Function	Control	Description
	Contrôle SplitContainer	<p>Fournit deux panneaux séparés par une barre mobile.</p> <p> Remarque Le contrôle SplitContainer est conçu pour remplacer le contrôle Splitter.</p>
	Contrôle TableLayoutPanel	Représente un panneau qui dispose dynamiquement son contenu dans une grille composée de lignes et de colonnes.
	Contrôle FlowLayoutPanel	Représente un panneau qui présente dynamiquement son contenu, horizontalement ou verticalement.
Audio	Contrôle SoundPlayer	Lit des fichiers audio au format .wav. Les sons peuvent être chargés ou peuvent l'être de façon asynchrone.

3.5. LES CONTROLES WINDOWS FORMS

Les différents contrôles sont des classes faisant partie de l'espace de nom System.Windows.Forms.



Voici la description des contrôles les plus courants de la liste Windows Forms.

3.5.1. LE CONTROLE LABEL

A Label Aide voir Label, classe

Les contrôles Étiquette (Label) permettent d'afficher du texte que l'utilisateur ne peut pas modifier. Par contre le texte affiché est modifiable depuis l'application.

🔔 Pour le nom du contrôle, propriété (Name), utilisez le préfixe **lbl**.

3.5.1.1. LABEL, PROPRIETE TEXT

Elle permet de définir le texte à afficher (état initial). On utilise la même propriété pour modifier depuis le code le texte :

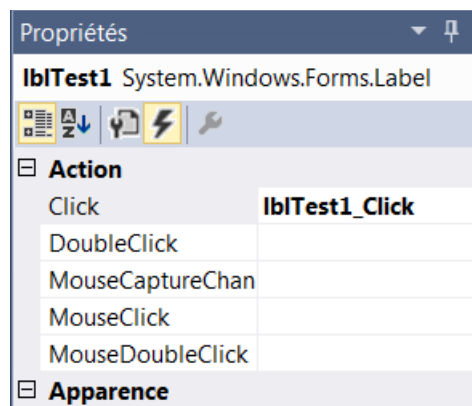
```
// Modifie la propriété Text du contrôle lblTest1
lblTest1.Text = "Hello !";
```

3.5.1.2. LABEL, EVENEMENT PAR DEFAUT

L'événement par défaut est un événement **Click**. Lors d'un double click sur le control cet événement est généré. On obtient la méthode :

```
private void lblTest1_Click(object sender, EventArgs e)
{
}
}
```

Au niveau des propriétés, en sélectionnant la liste des événement on obtient :



La section action nous montre les événements possibles et le lien entre Click et le nom de la méthode.

3.5.2. LE CONTROLE BUTTON

Button Aide voir **Button, classe**

Le contrôle **Button** de Windows Forms permet à l'utilisateur d'effectuer une action en cliquant dessus. Lorsque l'utilisateur clique sur le bouton, ce dernier réagit comme s'il était enfoncé puis relâché. Chaque fois qu'un utilisateur clique sur un bouton, le gestionnaire d'événements **Click** est appelé. Vous devez donc placer du code dans ce gestionnaire d'événements **Click** afin d'exécuter l'action de votre choix.

Pour le nom du contrôle, propriété (Name), utilisez le préfixe **btn**

La propriété **Text** permet de définir le texte affiché sur le bouton.

3.5.2.1. BUTTON, EVENEMENT PAR DEFAULT

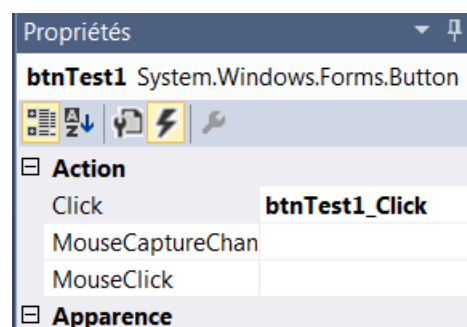
Par un double clic sur le bouton, la fenêtre code apparaît avec une méthode liée à l'événement **Click**.

Si le bouton est nommé btnTest1, on obtient :


```
private void btnTest1_Click(object sender, EventArgs e)
{
}
}
```

Au niveau des propriétés, en sélectionnant la liste des événement on obtient :

La section action nous montre les événements possibles et le lien entre Click et le nom de la méthode.



3.5.3. LE CONTROLE TEXTBOX

 **TextBox** Aide voir **TextBox, classe**

Le contrôle Zone de texte (TextBox) permet d'afficher des données saisies par l'utilisateur au moment de l'exécution ou affectées à la propriété **Text** du contrôle au moment de la création ou de l'exécution.

Pour le nom du contrôle, propriété (Name), utilisez le préfixe **txt**

3.5.3.1. TEXTBOX, PROPRIETE TEXT

Le texte entré dans le contrôle TextBox est contenu dans la propriété Text.

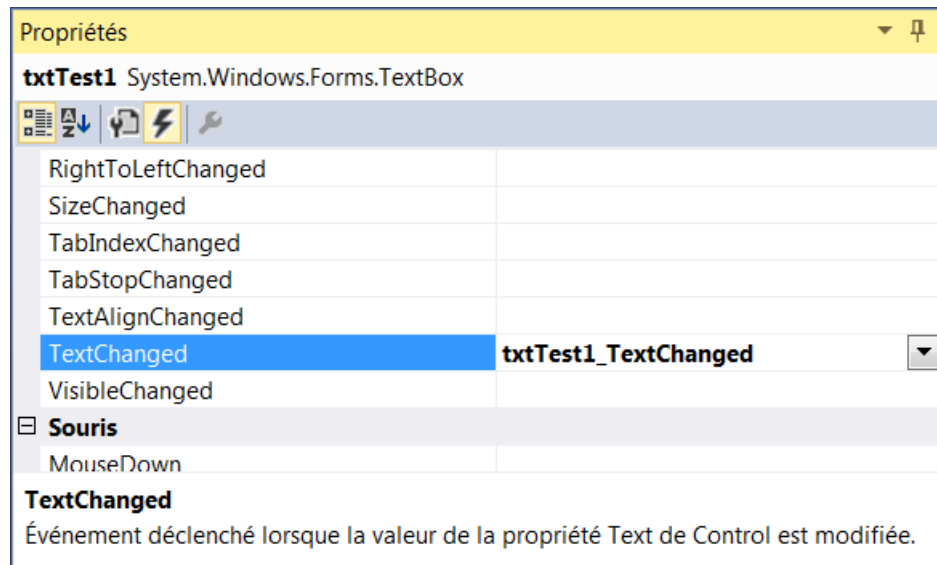
La propriété **MaxLength** définit le nombre maximal de caractères qui peuvent être entrés dans le contrôle d'édition. La taille par défaut est de 32767.

3.5.3.2. TEXTBOX, EVENEMENT PAR DEFAUT

L'événement par défaut est **TextChanged**.

```
private void txtTest1_TextChanged(object sender, EventArgs e)
{
}
}
```

Au niveau des propriétés, en sélectionnant la liste des événements on obtient :



L'événement **TextChanged** se trouve dans la section Propriété modifiée. On peut voir les nombreux événements possibles et le lien entre **TextChanged** et le nom de la méthode.

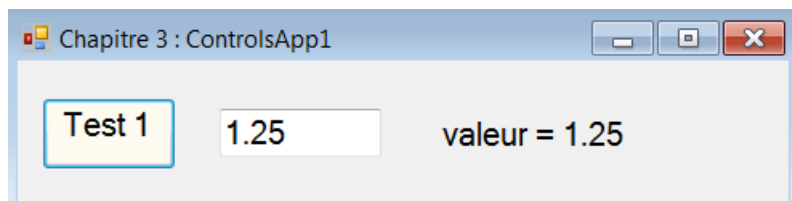
3.5.3.3. SAISIE ET AFFICHAGE D'UNE VALEUR NUMERIQUE

Dans cet exemple, dans l'événement Click d'un bouton on convertit le string en une valeur de type double. Ensuite pour afficher on convertit la valeur numérique en string.

```
private void btnTest1_Click(object sender, EventArgs e)
{
    double valeur;
```

```
// Lecture contenu TextBox et affichage avec lblTest1
valeur = double.Parse(txtTest1.Text);
lblTest1.Text = "valeur = " + valeur.ToString();
}
```

Résultat :



3.5.3.4. AFFICHAGE SUR PLUSIEURS LIGNES

Par défaut, le contrôle **TextBox** Windows Forms affiche une seule ligne de texte et n'affiche pas les barres de défilement. Si le texte est plus long que l'espace disponible, seule une partie du texte est visible. Vous pouvez modifier ce comportement par défaut en attribuant les valeurs appropriées aux propriétés **MultiLine**, **WordWrap** et **ScrollBars**.

Pour afficher plusieurs lignes dans le contrôle TextBox :

1. Définissez la propriété **MultiLine** à **true**. Si **WordWrap** est **true** (valeur par défaut), le texte du contrôle apparaîtra comme un ou plusieurs paragraphes ; sinon, il apparaîtra comme une liste dans laquelle certaines lignes peuvent être tronquées au niveau du bord du contrôle.
2. Définissez la propriété **ScrollBars** à la valeur appropriée.

Valeur	Description
None (Aucun)	Utilisez cette valeur si le texte est un paragraphe qui correspond presque toujours au contrôle. L'utilisateur peut utiliser le pointeur de la souris pour se déplacer dans le contrôle si le texte est trop long pour être affiché en entier en une seule fois.
Horizontal	Utilisez cette valeur si vous voulez afficher une liste de lignes, dont certaines peuvent être plus longues que la largeur du contrôle TextBox .
Both (Les deux)	Utilisez cette valeur s'il est possible que la liste soit plus longue que la hauteur du contrôle.

3. Attribuez à la propriété **WordWrap** une valeur appropriée.

Valeur	Description
false	Le texte dans le contrôle ne sera pas automatiquement renvoyé à la ligne, donc il défilera vers la droite jusqu'au prochain saut de ligne. Utilisez cette valeur si vous avez choisi les barres de défilement Horizontal ou Both (Les deux) ci-dessus.
true (valeur par défaut)	La barre de défilement horizontale n'apparaîtra pas. Utilisez cette valeur si vous avez choisi les barres de défilement Vertical ou None (Aucun) ci-dessus, afin d'afficher un ou plusieurs paragraphes.


3.5.3.5. TEXTBOX, PROPRIETE LINES

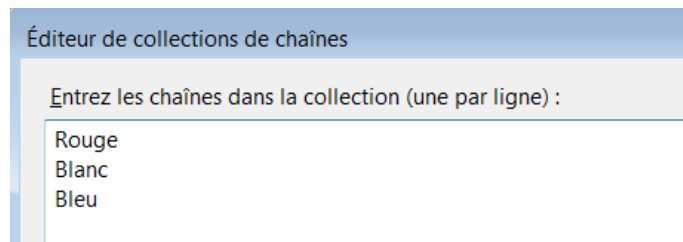
Elle permet dans la situation MultiLine, de voir le texte comme un tableau de chaînes de caractères (string).

Lines	Tableau de String[]
-------	---------------------

Cette propriété permet d'initialiser le texte multi ligne, ligne par ligne.

3.5.3.6. INITIALISATION PAR LES PROPRIETES

Lorsque l'on sélectionne le tableau de String[] le  apparaît et permet d'accéder à l'Editeur de collection de chaînes. Ce qui permet d'entrer des chaînes (enter effectue le changement de ligne).



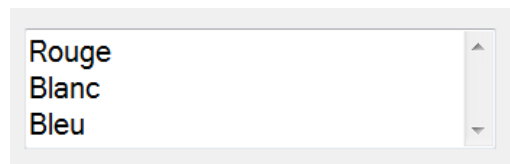
On obtient le détail du contenu de Lines :

Lines	Tableau de String[]	
[0]	Rouge	
[1]	Blanc	
[2]	Bleu	

La propriété Text elle devient :

Text	RougeBlancBleu
------	----------------

Au niveau du TextBox on obtient (en exécution) :



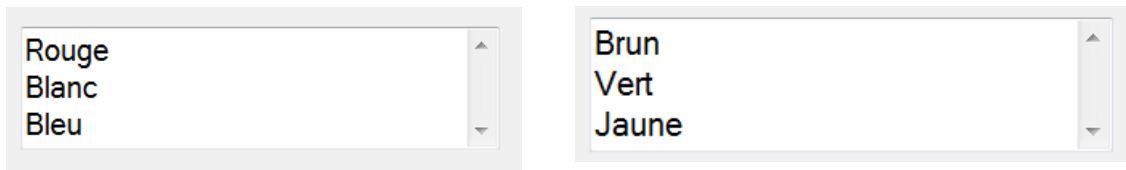
3.5.3.7. INITIALISATION PAR LE CODE DE LA PROPRIETE LINES

Dans l'action du button Fill on effectue :

```
private void btnFill_Click(object sender, EventArgs e)
{
    string[] tempArray = new string[3];
    tempArray[0] = "Brun";
    tempArray[1] = "Vert";
    tempArray[2] = "Jaune";
    txtMulti.Lines = tempArray;
}
```

Il est nécessaire d'utiliser un tableau de string temporaire pour l'affecter à la propriété Lines. On remarquera la syntaxe spécifique au C# de la déclaration du tableau de string.

Situation en exécution avant et après l'action sur le bouton Fill :



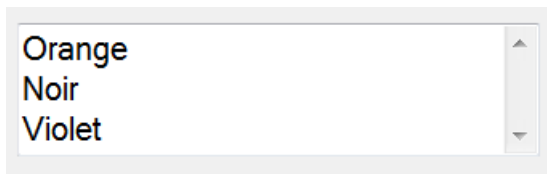
3.5.3.8. INITIALISATION PAR LE CODE DE LA PROPRIETE TEXT.

Il est possible en introduisant des caractères d'échappement dans un string d'obtenir le renvoi à la ligne. il faut utiliser "`\r\n`", le "`\n`" ne suffit pas.

Exemple en utilisant l'action du bouton Fill2.

```
private void btnFill2_Click(object sender, EventArgs e)
{
    txtMulti.Text = "Orange \r\n" + "Noir \r\n" + "Violet";
}
```

On obtient l'affichage sur 3 lignes.

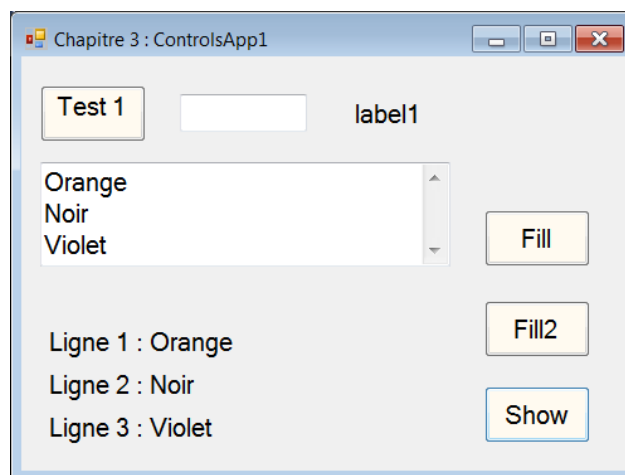


3.5.3.9. LECTURE D'UN TEXTE MULTI LIGNE


Dans l'exemple ci-dessous, dans l'action Click du bouton Show on affect chaque élément de la propriété Lines à un label.

```
private void btnShow_Click(object sender, EventArgs e)
{
    lblLine1.Text = "Ligne 1 : " + txtMulti.Lines[0];
    lblLine2.Text = "Ligne 2 : " + txtMulti.Lines[1];
    lblLine3.Text = "Ligne 3 : " + txtMulti.Lines[2];
}
```

Le résultat est correct quel que soit la façon d'introduire le texte multi lignes.



3.5.4. LE CONTROLE CHECKBOX

 [CheckBox](#) Aide voir **CheckBox, classe**

Utilisés en groupes, ces contrôles permettent d'afficher plusieurs choix à l'utilisateur qui peut en sélectionner un seul ou plusieurs.

Une coche s'affiche en regard du contrôle Case à cocher (CheckBox) lorsque celui-ci est sélectionné.

Pour le nom du contrôle, propriété (Name), utilisez le préfixe **chk**.

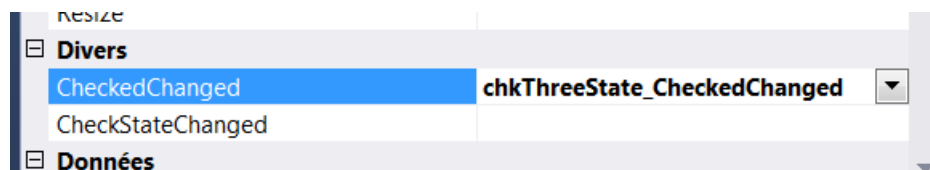
3.5.4.1. CHECKBOX, EVENEMENT PAR DEFAULT

L'événement par défaut est **CheckedChanged**.

Un double clic sur le CheckBox nommé chkThreeState génère le code suivant :

```
private void chkThreeState_CheckedChanged(object sender,
                                         EventArgs e)
{
    chkTest.ThreeState = chkThreeState.Checked;
}
```

👉 code auquel on a ajouté l'établissement de la propriété ThreeState du CheckBox chkTest à partir de la propriété Checked du CheckBox chkThreeState.



Il est à remarquer que cet événement ne concerne que le changement de la propriété **Checked**. Si on s'intéresse dans la situation où **ThreeState** a la valeur true à la propriété CheckState, alors il faut utiliser l'événement **CheckStateChanged**.

3.5.4.2. CHECKBOX, PROPRIETE CHECKED

La propriété **Checked** du contrôle CheckBox retourne la valeur **true** ou **false**. Avec true la case est cochée.

3.5.4.3. CHECKBOX, PROPRIETE CHECKSTATE

La propriété **CheckState** retourne **CheckState.Checked** ou **CheckState.Unchecked** ; ou, si la propriété **ThreeState** a la valeur **true**, **CheckState** peut aussi retourner **CheckState.Indeterminate**. Dans l'état indéterminé (**CheckState.Indeterminate**), la case est affichée avec une apparence estompée qui indique que l'option n'est pas disponible.

3.5.4.4. EXEMPLE SITUATION CHECKED ET CHECKSTATE

L'exemple proposé affiche dans des label la valeur des propriété **Checked** et **CheckState**, avec la possibilité de modifier la propriété **ThreeState**.

3.5.4.4.1. Code de l'exemple

Traitement dans l'événement Click du Button Test1.

```
private void btnTest1_Click(object sender, EventArgs e)
{
    lblCheck.Text = "Checked = " + chkTest.Checked.ToString();
    lblCheckState.Text = "CheckState = " +
        chkTest.CheckState.ToString();
}
```

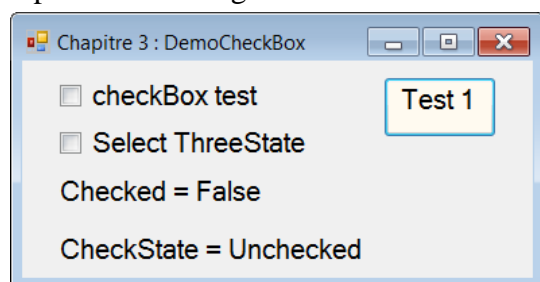
Traitement dans l'événement CheckedChanged du CheckBox chkThreeState pour établir à true ou false la propriété ThreeState.

```
private void chkThreeState_CheckedChanged(object sender,
    EventArgs e)
{
    chkTest.ThreeState = chkThreeState.Checked;
}
```

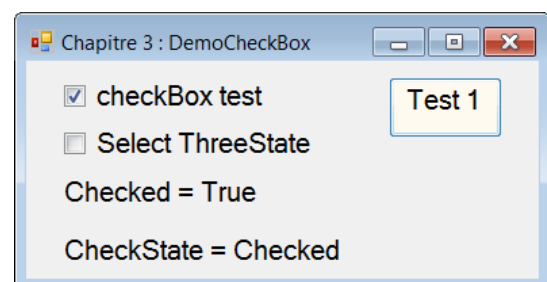
3.5.4.4.2. Résultats de l'exemple

Voici les résultats qui illustrent le comportement :

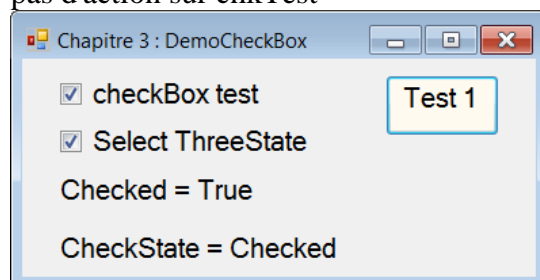
Après le démarrage



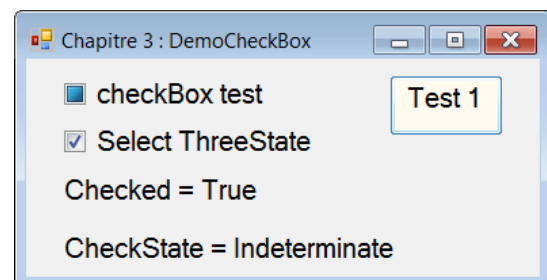
CheckBox checked



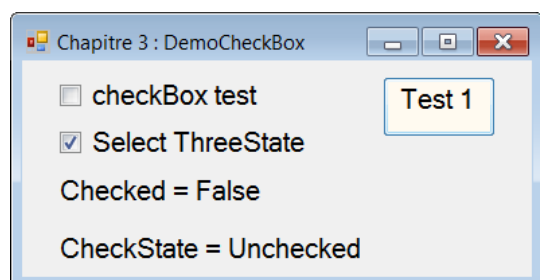
Situation avec ThreeState = true
pas d'action sur chkTest



Clic => état indéterminé



nouveau clic => unchecked



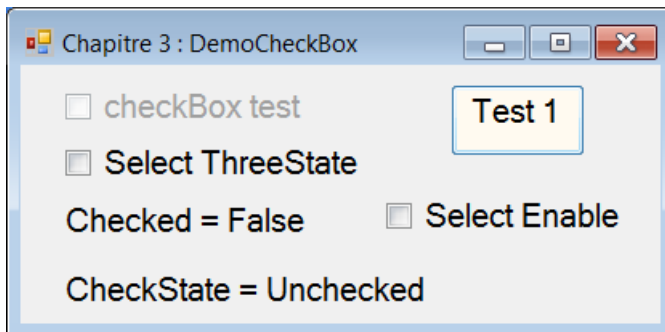
3.5.4.5. CHECKBOX, PROPRIETE ENABLED

Par contre, si on établit la propriété Enabled = False, le CheckBox devient grisé.

En utilisant un 3^{ème} CheckBox et l'action liée à l'événement CheckChanged.

```
private void chkEnable_CheckedChanged(object sender,
                                     EventArgs e)
{
    chkTest.Enabled = chkEnable.Checked;
}
```

On obtient (après 2 actions sur select Enable) :



Il n'est plus possible de modifier le CheckBox. Le CheckBox conserve l'état qu'il avait avant.

3.5.5. LE CONTROLE RADIOBUTTON

[RadioButton](#) Aide voir **RadioButton, classe**

Ce contrôle permet d'afficher des choix contenus dans des groupes de boutons d'options parmi lesquels l'utilisateur ne peut effectuer qu'une seule sélection.

Les contrôles **RadioButton** et **CheckBox** peuvent apparaître comme étant similaires. Il existe pourtant une différence importante entre ces deux contrôles : dans un groupe de **RadioButton**, lorsque l'utilisateur sélectionne un **RadioButton**, les autres deviennent automatiquement indisponibles. En revanche, l'utilisateur peut sélectionner autant de contrôles **CheckBox** qu'il le souhaite.

Pour le nom du contrôle, propriété (Name), utilisez le préfixe **rbtn**

Lorsqu'un utilisateur clique sur un contrôle **RadioButton**, sa propriété **Checked** prend la valeur **true** et le gestionnaire de l'événement **Click** est appelé. L'événement **CheckedChanged** est déclenché lorsque la valeur de la propriété **Checked** change (dû à l'action du clic sur le **RadioButton**).

3.5.5.1. RADIOBUTTON, PROPRIETE CHECKED

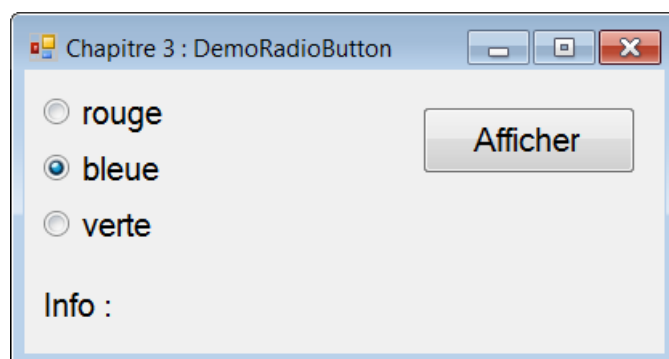
La propriété **Checked** du contrôle **RadioButton** indique si le bouton est sélectionné. Si tel est le cas, la valeur devient **true**.

3.5.5.1.1. Modification de la propriété Checked

Au niveau du code, il est possible d'imposer la valeur au **RadioButton**. Par exemple dans le constructeur de **Form1**.

```
public Form1()
{
    InitializeComponent();
    rbtnBleu.Checked = true;
}
```

Ce qui a pour effet de modifier la situation d'affichage au démarrage (En général le 1^{er} **RadioButton** est coché).



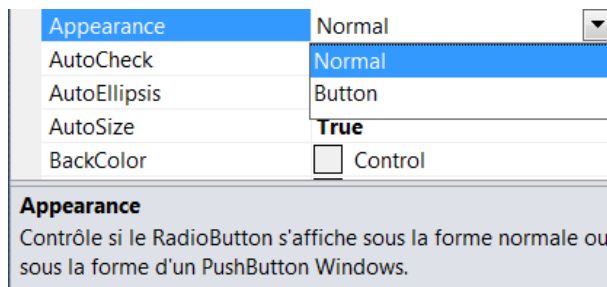
3.5.5.1.2. Lecture de la propriété Checked

Il est aussi possible de tester l'état du RadioButton. Comme dans l'exemple ci-dessous. Lors de l'action sur le bouton afficher on compose un message en testant la situation de chacun des RadioButton.

```
private void btnAfficher_Click(object sender, EventArgs e)
{
    string colorName = "";
    if (rbnRouge.Checked == true)
        colorName = rbnRouge.Text;
    if (rbnBleu.Checked == true)
        colorName = rbnBleu.Text;
    if (rbnVert.Checked == true)
        colorName = rbnVert.Text;
    lblInfo.Text = "La couleur est " + colorName;
}
```

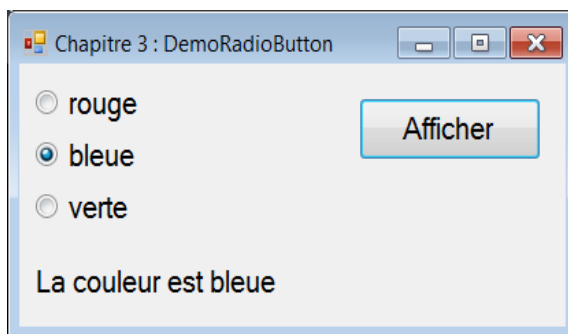
3.5.5.2. RADIOBUTTON, PROPRIETE APPEARANCE

Cette propriété a par défaut la valeur Normal. Il est possible de la modifier en Button.

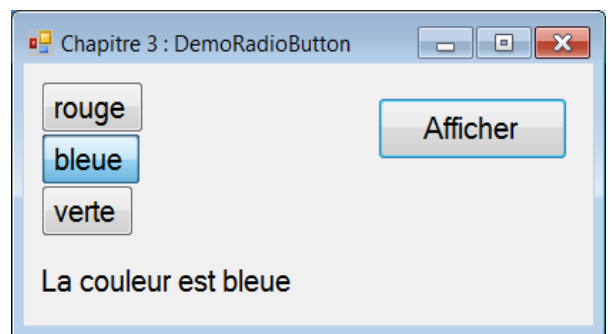


Voici les deux apparences des RadioButtons.

Appearance = Normal



Appearance = Button

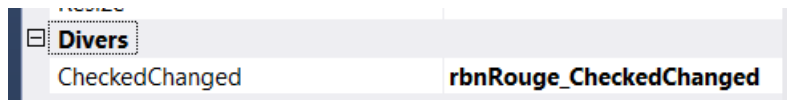


3.5.5.3. RADIOBUTTON, EVENEMENT PAR DEFAULT

L'événement par défaut est **CheckedChanged**. Un double clic sur le CheckBox génère la méthodes suivante :

```
private void rbnRouge_CheckedChanged(object sender, EventArgs e)
{
}
```

Dans la liste des événements on trouve la relation entre l'événement et le nom de la méthode.



🔔 Lorsque l'on active un RadioButton au sein d'un groupe on obtient 2 événements :

- 1) Un événement pour checked qui passe de True à False (Pour le RadioButton qui était Checked)
- 2) Un événement pour checked qui passe de False à True (Pour le RadioButton qui devient Checked)

3.5.5.4. RADIOBUTTON, EXEMPLE

Voici un exemple de traitement par les événements de 3 RadioButton permettant de composer le message et d'attribuer la couleur correspondante au texte d'un Label.

```
private void rbnRouge_CheckedChanged(object sender, EventArgs e)
{
    if (rbnRouge.Checked == true)
    {
        lblInfo.Text = "La couleur est " + rbnRouge.Text;
        lblInfo.ForeColor = Color.Red;
    }
}
```

```
private void rbnBleu_CheckedChanged(object sender, EventArgs e)
{
    if (rbnBleu.Checked == true)
    {
        lblInfo.Text = "La couleur est " + rbnBleu.Text;
        lblInfo.ForeColor = Color.Blue;
    }
}
```

```
private void rbnVert_CheckedChanged(object sender, EventArgs e)
{
    if (rbnVert.Checked == true)
    {
        lblInfo.Text = "La couleur est " + rbnVert.Text;
        lblInfo.ForeColor = Color.Green;
    }
}
```

☹️ à cause des 2 événements générés lors du passage du Checked de l'un à l'autre des RadioButton, on effectue un test pour n'agir que si Checked est true.

3.5.5.5. DESACTIVATION D'UN BOUTON D'OPTION

Pour désactiver un bouton d'option, affectez la valeur **false** à sa propriété **Enabled**. Au moment de l'exécution, le RadioButton apparaîtra ombré, ce qui signifie qu'il est indisponible.

Il n'est plus possible de sélectionner la couleur verte

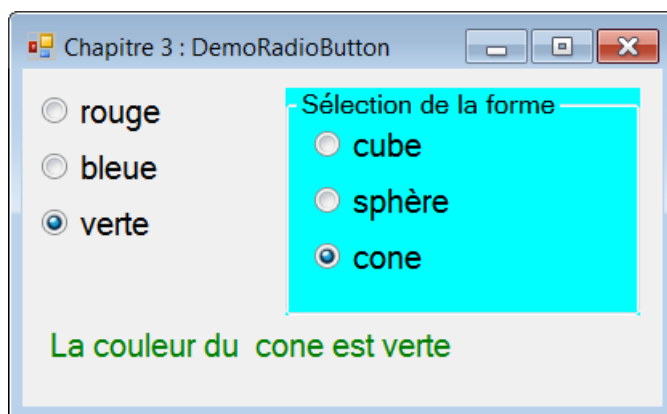


3.5.5.6. BESOIN DES DE GROUPES DE RADIOBUTTON

Dans le formulaire l'ensemble des RadioButton sont couplé. Si par exemple on souhaite réalisé la sélection de couleurs et de formes, il est nécessaire de regrouper les contrôles RadioButton en les plaçant à l'intérieur d'un conteneur, tel qu'un contrôle **GroupeBox**, **Panel**, ou une feuille.

Voici un complément à l'exemple précédant, avec l'introduction dans un GroupBox de 3 RadioButton permettant de sélectionner une forme.

Pour bien marquer le GroupBox sa couleur de fond a été modifiée en Aqua.



Voici le code complet de l'exemple qui a évolué en introduisant une méthode pour mettre à jour le message. En plus introduction de 3 attributs pour manipuler la couleur, le nom de la couleur et la forme.

```
namespace DemoRadioButton
{
    public partial class Form1 : Form
    {
        Color myColor;
        string myForme;
        string myColorName;

        public Form1()
        {
            InitializeComponent();
            rbnBleu.Checked = true;
        }
    }
}
```



```
        myColor = Color.Blue;
        myColorName = "bleue";
        rbnCube.Checked = true;
        myForme = "cube";
    }

    private void rbnRouge_CheckedChanged(object sender,
                                         EventArgs e)
    {
        if (rbnRouge.Checked == true)
        {
            myColorName = rbnRouge.Text;
            myColor = Color.Red;
            MajMess();
        }
    }

    private void rbnBleu_CheckedChanged(object sender,
                                         EventArgs e)
    {
        if (rbnBleu.Checked == true)
        {
            myColorName = rbnBleu.Text;
            myColor = Color.Blue;
            MajMess();
        }
    }

    private void rbnVert_CheckedChanged(object sender,
                                         EventArgs e)
    {
        if (rbnVert.Checked == true)
        {
            myColorName = rbnVert.Text;
            myColor = Color.Green;
            MajMess();
        }
    }

    private void rbnCube_CheckedChanged(object sender,
                                         EventArgs e)
    {
        if (rbnCube.Checked == true)
        {
            myForme = rbnCube.Text;
            MajMess();
        }
    }
}
```

```

private void rbnSphere_CheckedChanged(object sender,
                                     EventArgs e)
{
    if (rbnSphere.Checked == true)
    {
        myForme = rbnSphere.Text;
        MajMess();
    }
}

private void rbnCone_CheckedChanged(object sender,
                                    EventArgs e)
{
    if (rbnCone.Checked == true)
    {
        myForme = rbnCone.Text;
        MajMess();
    }
}

private void MajMess()
{
    lblInfo.Text = "La couleur du " + myForme +
                  " est " + myColorName;
    lblInfo.ForeColor = myColor;
}
}

```

3.5.6. LE CONTROLE GROUPBOX



GroupBox Aide voir

GroupBox, classe

Les contrôles **GroupBox** sont utilisés pour fournir un groupement identifiable d'autres contrôles. Par exemple, vous pouvez utiliser des contrôles GroupBox pour subdiviser une feuille de manière fonctionnelle, afin de séparer par exemple des contrôles RadioButton.

La légende de la zone de groupe est définie par la propriété **Text**.

Il est aussi possible d'utiliser le GroupBox comme zone pour réaliser un graphique.

3.5.7. LE CONTROLE PictureBox

 **PictureBox** Aide voir **PictureBox, classe**

Pour le nom du contrôle, propriété (Name), utilisez le préfixe **pic**.

Le contrôle **PictureBox** Windows Forms sert à afficher des graphiques au format bitmap, GIF, JPEG, métafichier ou icône.

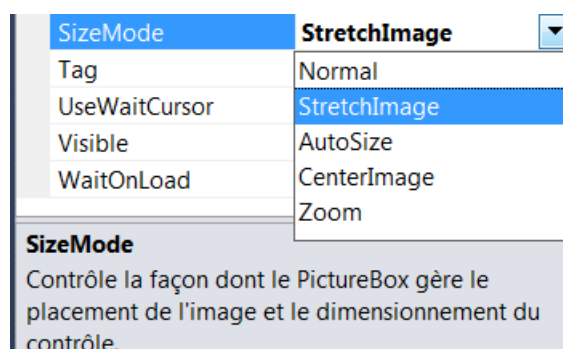
3.5.7.1. LA PROPRIETE IMAGE

L'image affichée est déterminée par la propriété **Image**, laquelle peut être définie au moment de l'exécution ou à celui du design.

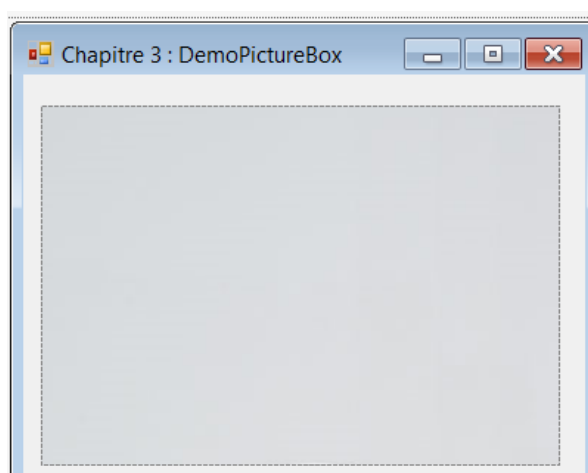
```
// Set the image property.
this.PictureBox1.Image = new Bitmap(typeof(Button), "Button.bmp");
```

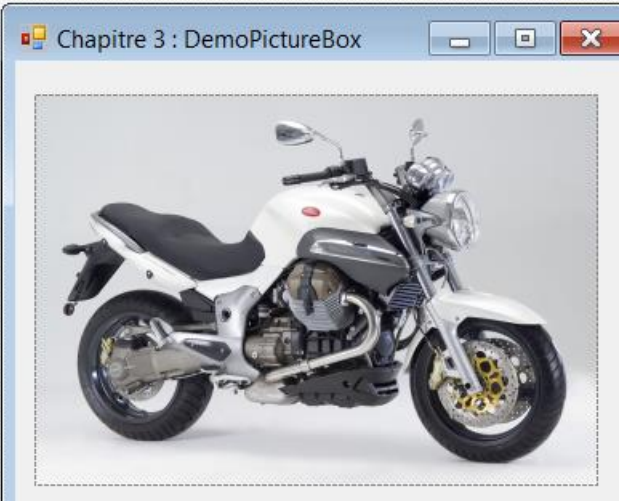
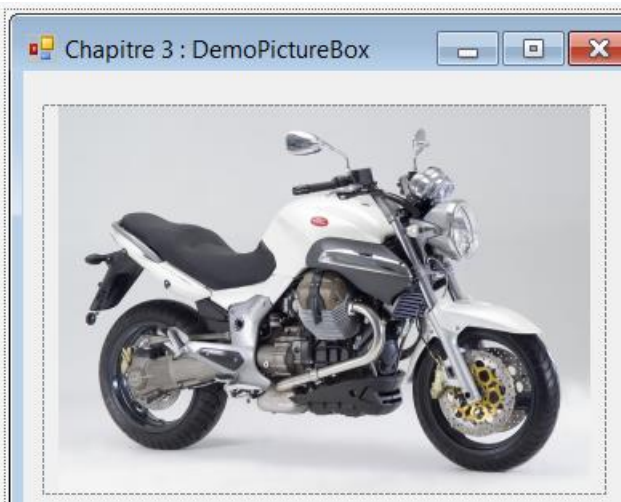

3.5.7.2. LA PROPRIETE SIZEMODE

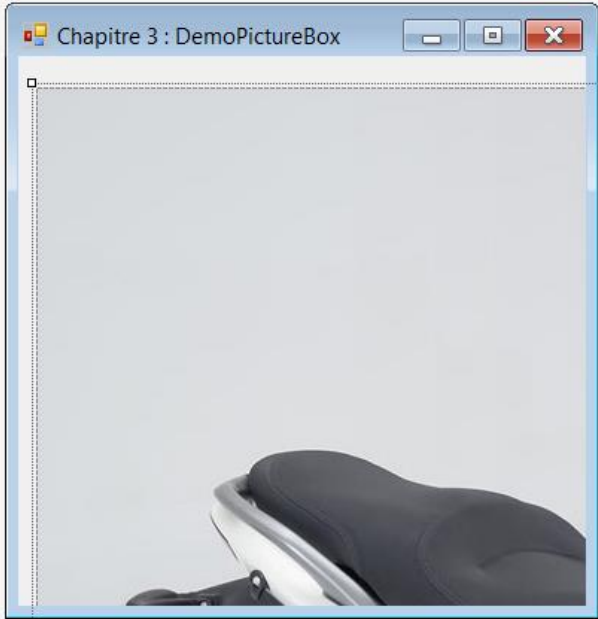
La propriété **SizeMode** détermine la façon dont l'image et le contrôle se dimensionnent l'un par rapport à l'autre. Il y a 5 modes :



Le résultat est donné avec une image bien plus grande que la zone PictureBox




Valeur de SizeMode	Effet	Résultat
Normal	Alignement du coin supérieur gauche de l'image avec celui du contrôle. La taille de l'image n'est pas modifiée. (On ne voit que la couleur de fond)	

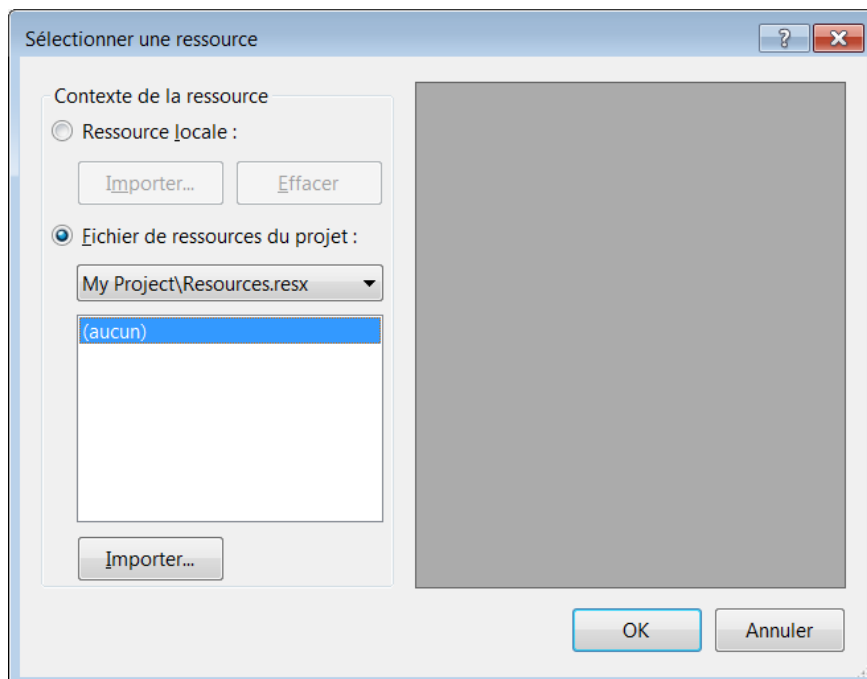
StretchImage	Etirement/réduction de l'image pour l'adapter à la taille du contrôle qu'elle contient	
Zoom	Réduit l'image en conservant le rapport hauteur/largeur, avec redimensionnement de la zone de manière à ne pas dépasser la dimension de départ	
CenterImage	Affiche la partie centrale de l'image dans le contrôle, OU centre l'image dans le cas d'une petite image.	

<p>AutoSize</p>	<p>Ajustement de la taille du contrôle pour l'adapter à celle de l'image qu'il affiche.</p> <p>L'image s'affiche en taille réelle.</p> <p>Dans notre cas la Picture Box s'agrandi jusqu'à la limite de la feuille.</p> <p>Remarque : si on revient à un autre mode, la zone reprend les dimensions initiales.</p>	
------------------------	--	--

☞ C'est la situation `SizeMode = Zoom` qui donne le meilleur résultat, car l'image garde ses proportions et la dimension de la PictureBox n'est qu'un peu modifiée.

3.5.7.3. CHARGEMENT D'UNE IMAGE AU MOMENT DU DESIGN

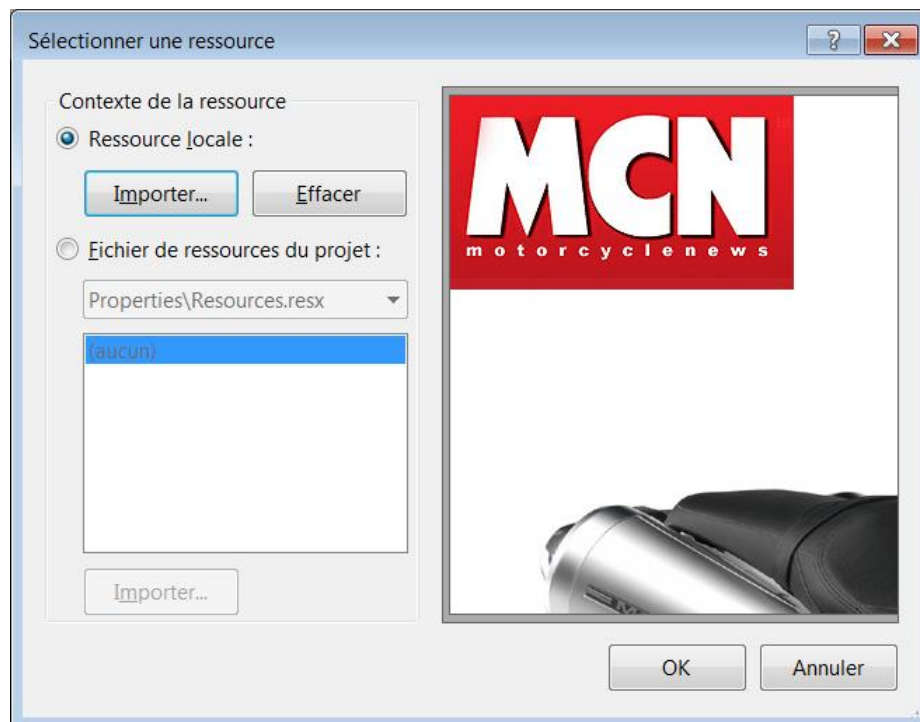
Dans la fenêtre Propriétés, sélectionnez la propriété **Image**  (aucun) , puis cliquez sur le bouton Sélection (), pour afficher la boîte de dialogue **Sélectionner une ressource**.



Il y a deux contextes possibles :

3.5.7.3.1. Ressource locale

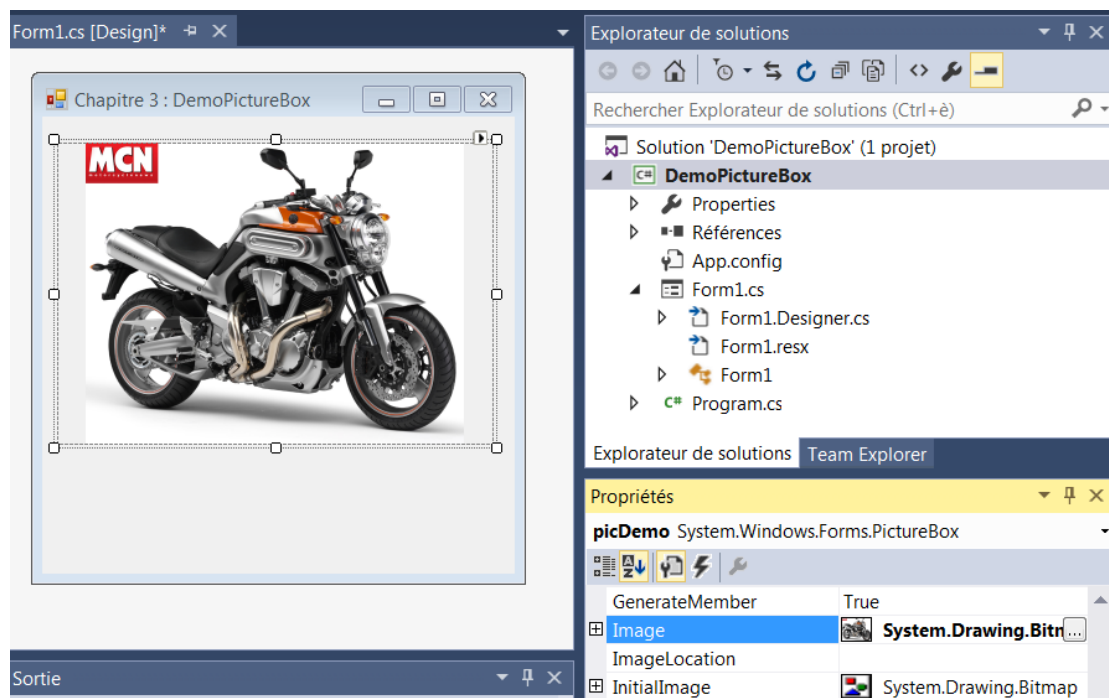
Dans ce contexte, il suffit d'activer le bouton Importer et de rechercher un fichier.



L'indication du nom du fichier n'apparaît pas, l'image devient du type **System.Drawing.Bitmap**.



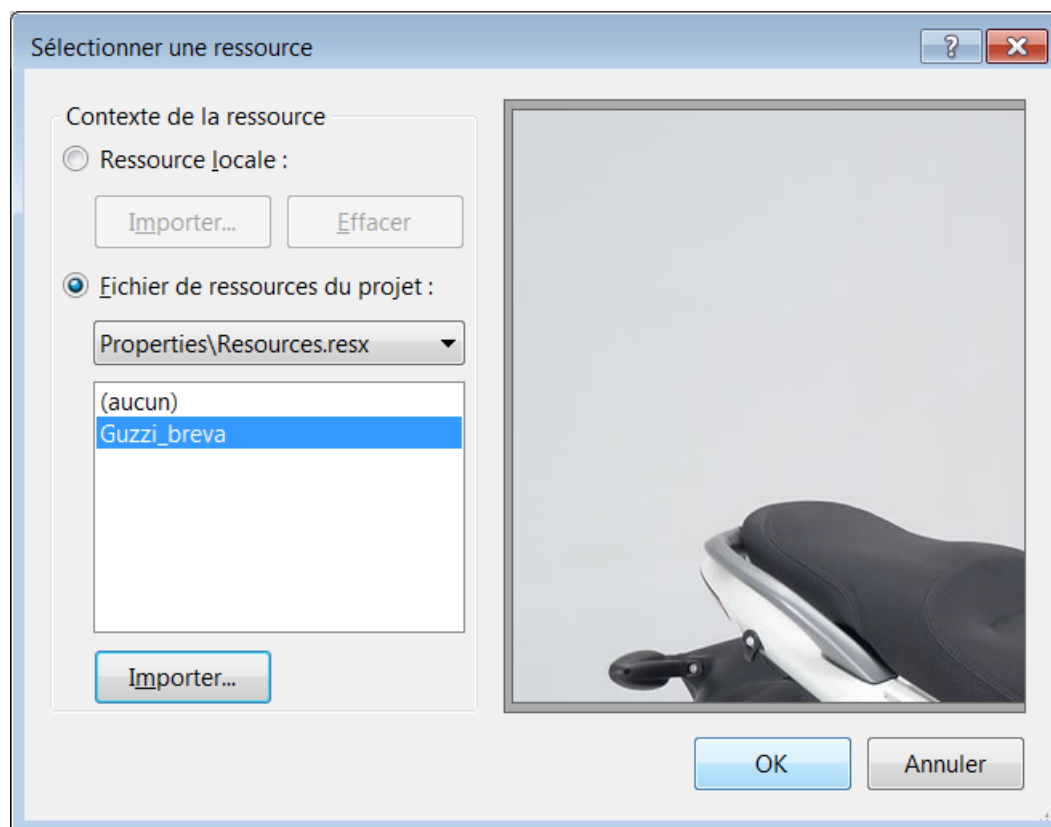
Il n'est possible d'importer qu'une seule image !



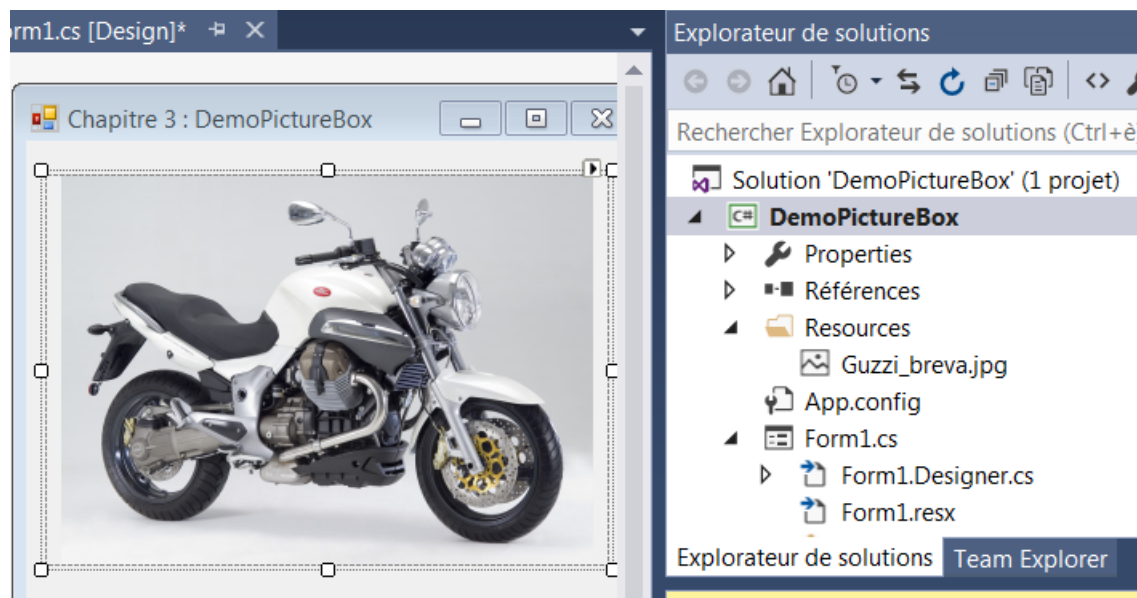
L'image n'apparaît pas dans les ressources et il n'y a pas de création d'un répertoire ressources.

3.5.7.3.2. Fichier de ressources du projet

Dans ce contexte, il suffit d'activer le bouton Importer et de rechercher un fichier.

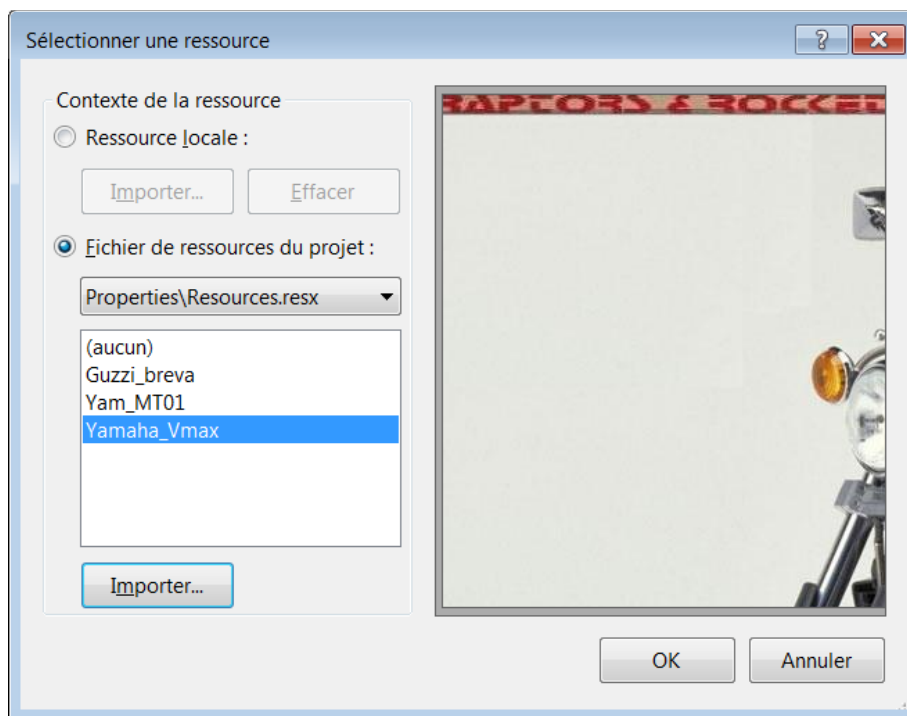


Après avoir activé le bouton OK, on observe l'image dans le PictureBox, ainsi que l'indication du nom du fichier dans le répertoire ressources de l'explorateur de solution.

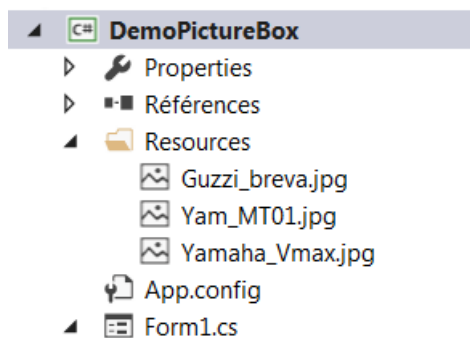


Le fichier image est copié dans le répertoire **Resources** du projet.

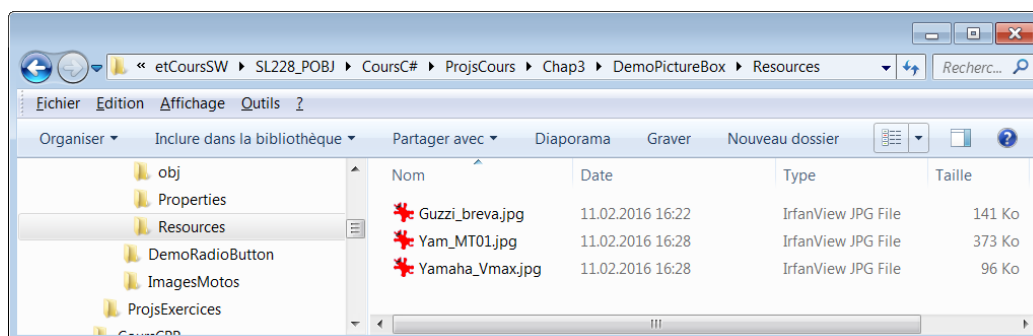
Il est possible de charger plusieurs images :



Que l'on voit apparaître dans le répertoire **Resources**.



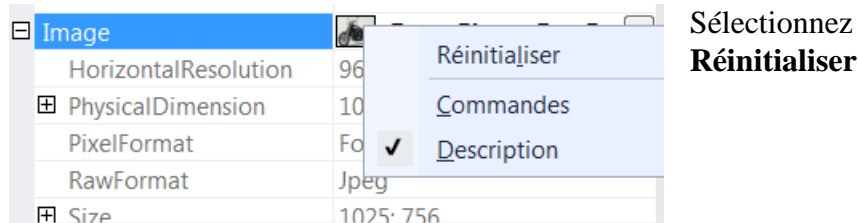
Et bien sûr dans la structure du projet en utilisant l'explorer.



3.5.7.4. EFFACEMENT D'UNE IMAGE AU MOMENT DU DESIGN

Dans la fenêtre Propriétés, sélectionnez la propriété **Image**. Il y a 2 possibilités d'action:

1) Cliquez avec le bouton droit de la souris sur l'image miniature affichée à gauche du nom de l'objet image. Un petit menu apparaît :



La propriété image devient :

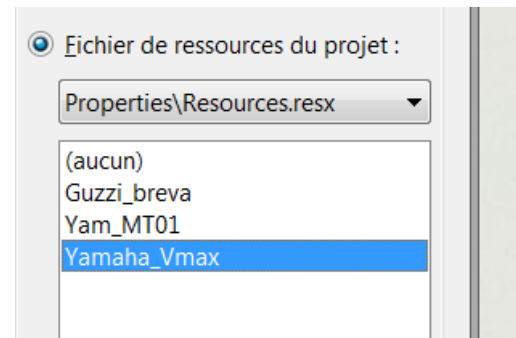
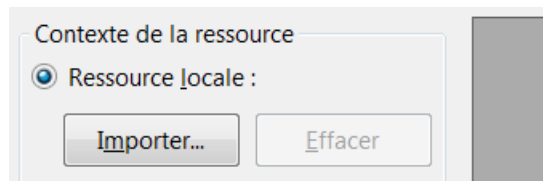


⊗ mais les fichiers restant dans les ressources !.

2) Cliquez sur le bouton sélection (). Il y a 2 possibilités selon le contexte.

Pour le Contexte "Ressource locale", utilisez le bouton Effacer

Pour le Contexte "Ressource du projet", Sélectionnez (**aucun**)



L'image est supprimée de l'affichage, mais pas des ressources.

Comme les images sont dans les ressources il est possible de faire réapparaître une image en cliquant sur son nom.

3.5.7.5. CHARGEMENT D'UNE IMAGE EN EXECUTION DEPUIS LES RESSOURCES

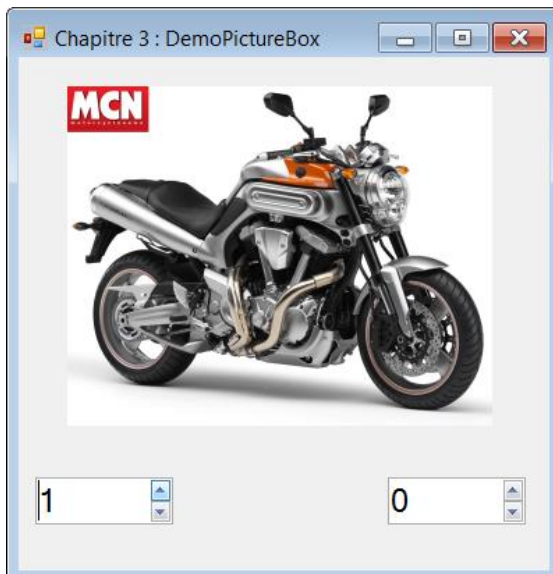
Dans la situation où on dispose de quelques images dans les ressources, il est possible de modifier la propriété image en exécution.

Pour charger une image ou changer d'image en exécution, il est possible comme ci-dessous de modifier la propriété **Image**, dans cet exemple sur la base de l'événement de changement de valeur d'un NumericUpDown.

```
private void nudSelect_ValueChanged(object sender, EventArgs e)
{
    int Select = (int)nudSelect.Value;
    switch (Select)
    {
        case 0 :
            picDemo.Image = Properties.Resources.Guzzi_breva;
            break;
        case 1:
            picDemo.Image = Properties.Resources.Yam_MT01;
            break;
        case 2:
            picDemo.Image = Properties.Resources.Yamaha_Vmax;
            break;
    }
}
```

😊 Dans ce cas on utilise directement **Properties.Resources**. La notion de nom de fichier et de chemin n'est plus nécessaire.

Exemple en exécution.



3.5.7.6. CHARGEMENT D'UNE IMAGE EN EXECUTION, NEW BITMAP

Pour charger une image ou changer d'image en exécution, il est possible comme ci-dessous de modifier la propriété **Image**. Utilisation du constructeur de Bitmap en en spécifiant soit le chemin complet, soit seulement le nom du fichier.

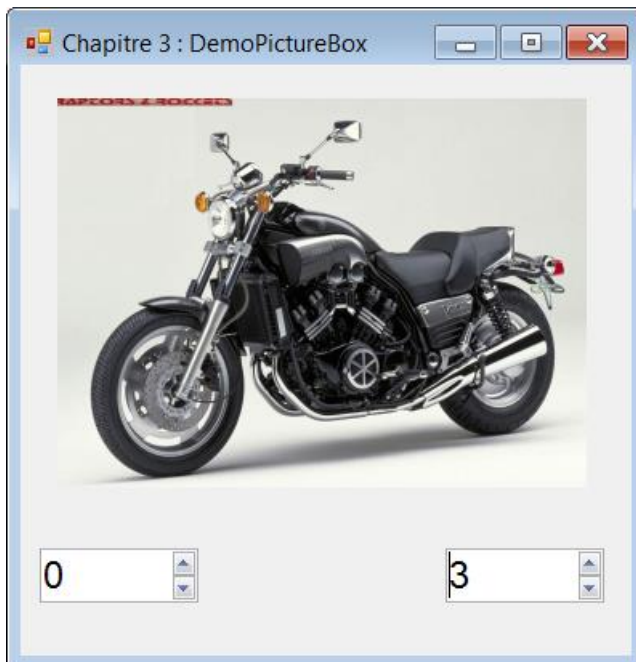


Dans ce cas le fichier doit être copié sous bin\Debug ou bin\Release selon la situation.

Exemple en utilisant le 2^{ème} NumericUpDown.

```
private void nudSelectBis_ValueChanged(object sender,
                                     EventArgs e)
{
    int Select = (int)nudSelectBis.Value;
    switch (Select)
    {
        case 0:
            picDemo.Image = new Bitmap("Guzzi_breva.jpg");
            break;
        case 1:
            picDemo.Image = new Bitmap("Yam_MT01.jpg");
            break;
        case 2:
            picDemo.Image = new Bitmap("Yamaha_Vmax.jpg");
            break;
    }
}
```

Exemple en exécution.

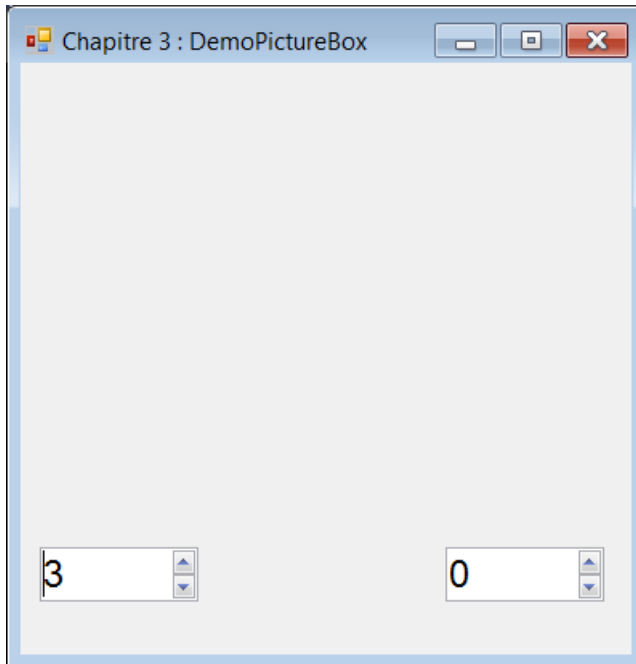


3.5.7.7. EFFACEMENT D'UNE IMAGE EN EXECUTION


Définissez la propriété **Image** en lui attribuant **null**. Exemple :

```
case 3:
    picDemo.Image = null;
break;
```

Le graphique sera supprimé du contrôle PictureBox, même s'il a été chargé dans la propriété Image au moment de la création.



3.5.8. LE CONTROLE IMAGELIST

 **ImageList** Aide voir **ImageList, classe**

Pour le nom du contrôle, propriété (Name), utilisez le préfixe **ili**

Le composant **ImageList** Windows Forms permet de stocker des images, qui peuvent ensuite être affichées par d'autres contrôles. ☞ Ce contrôle n'apparaît pas sur la feuille mais dans une zone annexe.

☞ Pour les projets **Smart Device** ce control est nécessaire pour manipuler les images !

Vous pouvez utiliser une liste d'images avec n'importe quel contrôle doté d'une propriété **Image**.

Images est la principale propriété du composant **ImageList** ; **Images** (Collection) elle contient les images à utiliser dans le contrôle associé. Chaque image est accessible par sa valeur d'index.

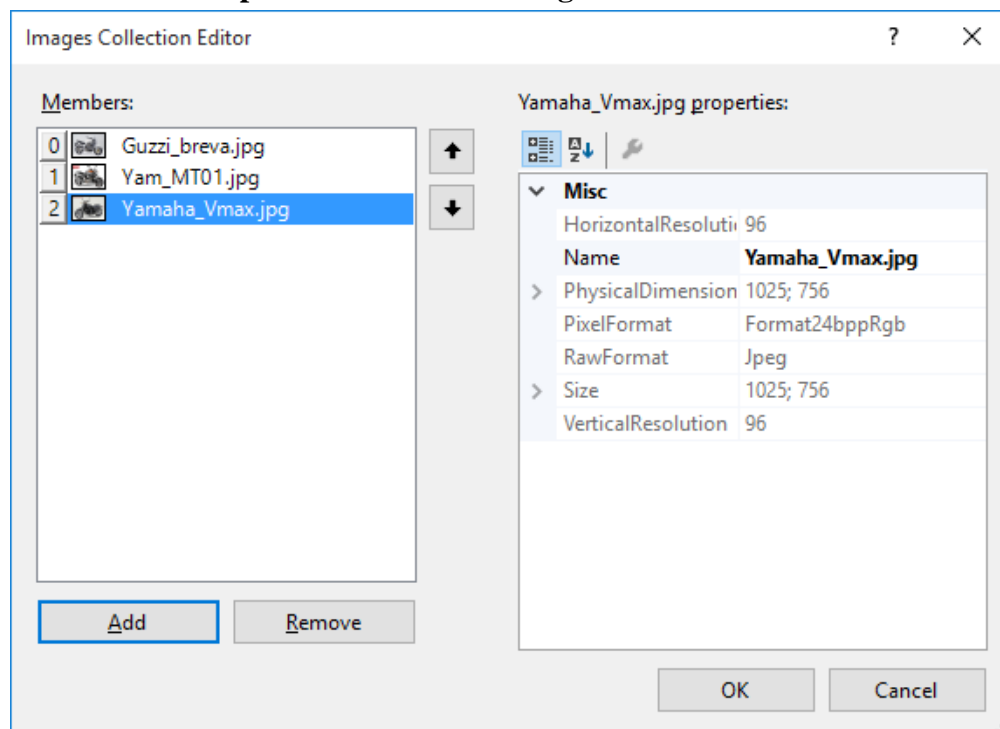
3.5.8.1. AJOUTER OU SUPPRIMER DES IMAGES DANS LE CONCEPTEUR

Dans la fenêtre Propriétés du composant **ImageList**, cliquez sur le bouton  en regard de la propriété **Images**.

Vous voyez s'afficher l'**Éditeur de la collection Image**.

Utilisez les boutons **Ajouter** et **Supprimer** pour ajouter et supprimer des images de la liste.

3.5.8.1.1. Exemple de collections d'images:



☞ Cette action ne place pas les images dans les ressources !

3.5.8.2. AJOUTER DES IMAGES PAR PROGRAMME

Utilisez la méthode **Add** de la propriété **Images** de la liste d'images.

Pour ajouter une image il est possible de créer un bitmap à partir du fichier qui doit être placé sous bin\debug.

```
public Form1()
{
    InitializeComponent();
    iliMotos.Images.Add(new Bitmap("Yamaha_FJR1300AS_001.jpg"));
    iliMotos.Images.Add(new Bitmap("bmw-r1200rt.jpg"));
}
```

3.5.8.3. POUR SUPPRIMER DES IMAGES PAR PROGRAMME

Utilisez la méthode **RemoveAt** pour supprimer une image ou la méthode **Clear** pour supprimer toutes les images de la liste.

```
// supprime la 1ere image
iliMotos.Images.RemoveAt(0);

// Efface toute les images de la liste
iliMotos.Images.Clear();
```

3.5.8.4. DIMENSION ET RESOLUTION DES IMAGES

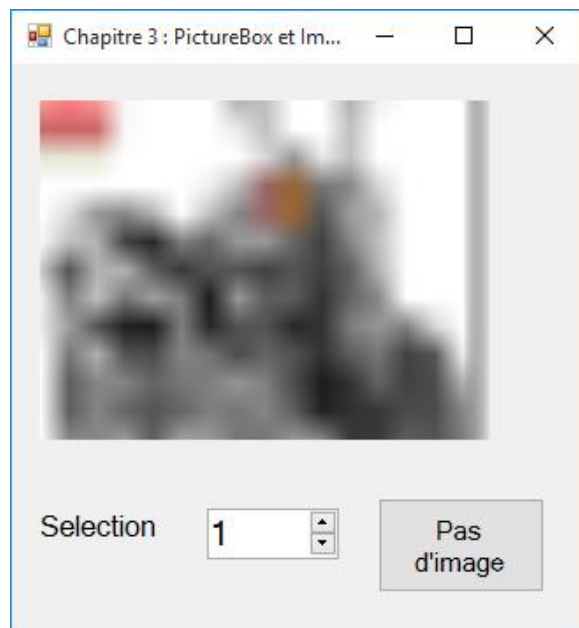
Pour éviter des surprises quant à la résolution et l'apparence des images, il faut établir comme suit les propriétés **ImageSize** et **ColorDepth** du contrôle **ImageList**

Ceci lorsque l'on charge des images couleur d'une taille correcte.

iliMotos System.Windows.Forms.ImageList	
<div> <div> <div></div> <div></div> <div></div> </div> <div> <div></div> <div></div> <div></div> </div> </div>	
⊕ (ApplicationSettings)	
(Name)	iliMotos
ColorDepth	Depth32Bit
GenerateMember	True
Images	(Collection)
⊖ ImageSize	256; 256
Width	256
Height	256
Modifiers	Private

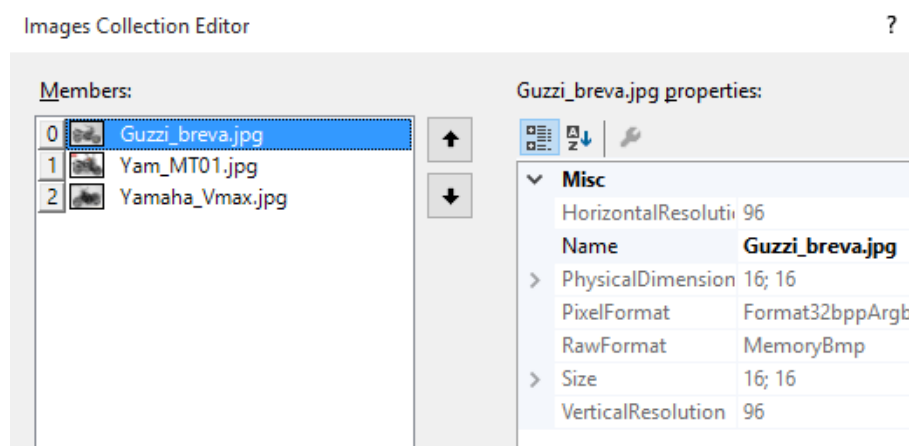
3.5.8.5. REDIMENSIONNEMENT DES IMAGES

Les images qui ont été chargées dans la liste avec la propriété établie à 16x16 donnent un résultat flou.

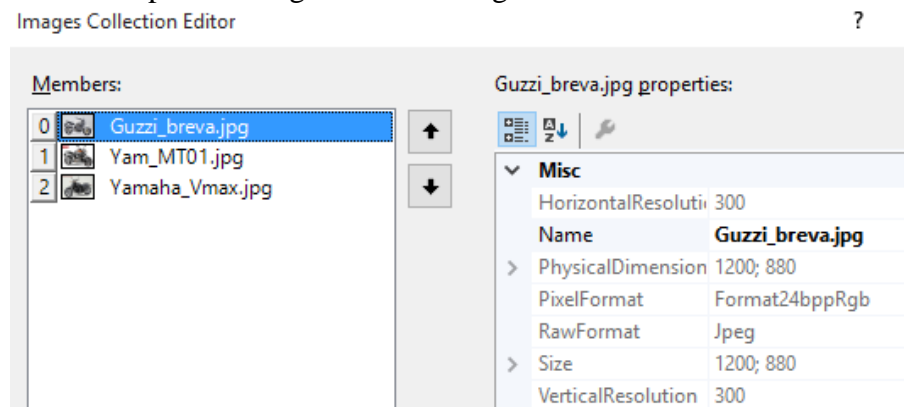


☹ Il faut dans l'éditeur de collection supprimer les images et les ajouter à nouveau.

On constate que l'image a été enregistrée avec une taille de 16 x 16.



Situation après rechargement de l'image.



3.5.9. EXEMPLE UTILISATION DE LA LISTE D'IMAGE

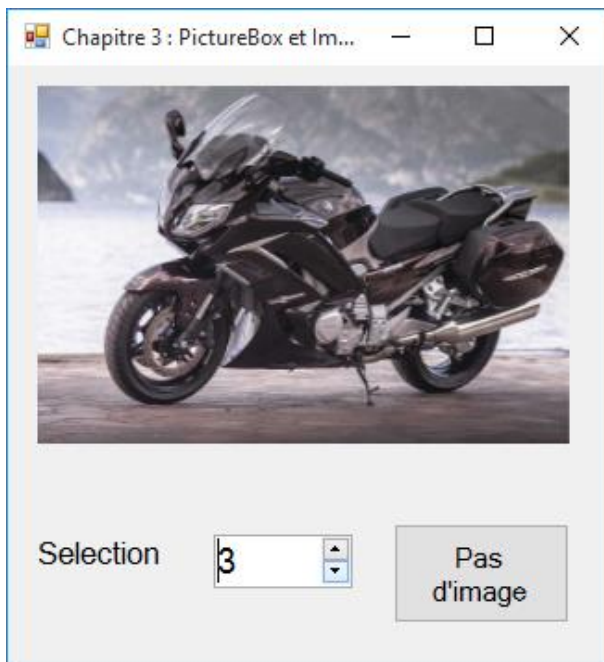
Dans l'exemple ci-dessous, nous allons utiliser une liste d'image, que nous allons afficher dans un contrôle PictureBox. En plus nous utiliserons le contrôle NumericUpDown, avec la propriété maximum limité au nombre d'images - 1.

Une partie de la liste est introduite dans la collection, le complément est réalisé dans le code dans le constructeur.

Nous utilisons l'événement du changement de valeur du contrôle NumericUpDown, pour charger une image de la liste dans le contrôle PictureBox.

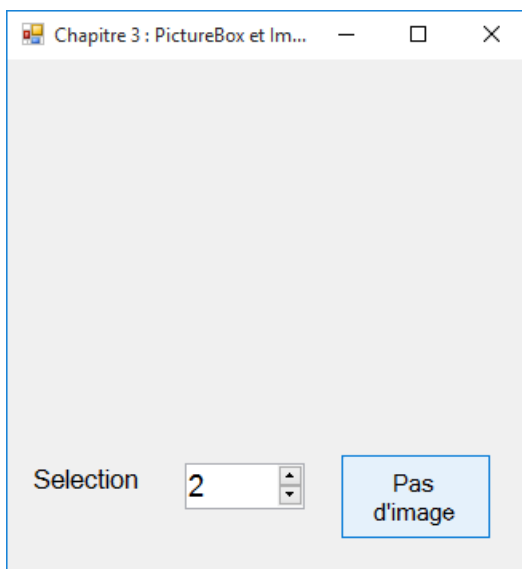
L'événement du bouton Pas d'image est utilisé pour effacer l'image et la liste.

3.5.9.1.1. Situation en exécution

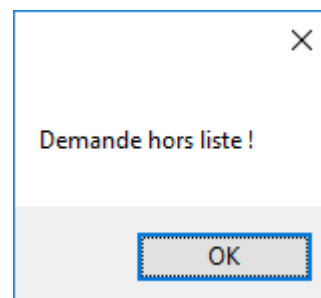


Remarque : la propriété SizeMode de la PictureBox a été établie à **StretchImage**.

Action du bouton Pas d'image :



Affichage du message lorsque l'on modifie la valeur de sélection, car il y a effacement de l'image et suppression des éléments de la liste :



3.5.9.1.2. Code de l'exemple de gestion d'images

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        iliMotos.Images.Add(new
            Bitmap("Yamaha_FJR1300AS_001.jpg"));
        iliMotos.Images.Add(new Bitmap("bmw-r1200rt.jpg"));
        //Configure la valeur maximum du Nud
        nudSelect.Maximum = iliMotos.Images.Count - 1;
    }

    private void nudSelect_ValueChanged(object sender,
        EventArgs e)
    {
        int Idx = (int)nudSelect.Value;
        if (Idx < iliMotos.Images.Count) {
            picMotos.Image = iliMotos.Images[Idx];
        } else
        {
            MessageBox.Show("Demande hors liste !");
        }
    }

    private void btnNoImage_Click(object sender,
        EventArgs e)
    {
        // Affiche aucune image
        picMotos.Image = null;

        // Efface toute les images de la liste
        iliMotos.Images.Clear();
    }
}
```

3.5.11. LE CONTROLE PANEL

Panel Aide voir **Panel, classe**

Le contrôle **Panel** de Windows Forms permet de fournir un mode de groupement identifiable pour les autres contrôles. Les zones sont principalement utilisées pour diviser un formulaire par fonctions. Le contrôle **Panel** est similaire au contrôle **GroupBox** ; toutefois, seul le contrôle **Panel** peut disposer de barres de défilement et seul le contrôle **GroupBox** peut afficher une légende.

Le Panel dispose de la méthode **CreateGraphics**, qui permet de réaliser des graphiques sur le panel.

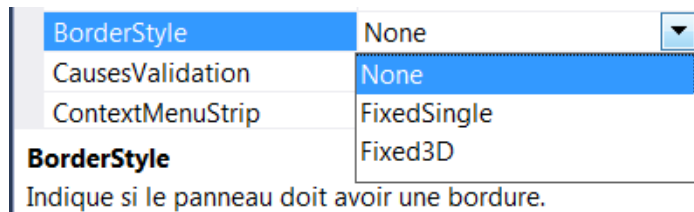
Pour rendre le Panel bien visible il est possible de modifier sa couleur de fond, propriété **BackColor**.

Par exemple au niveau du code :

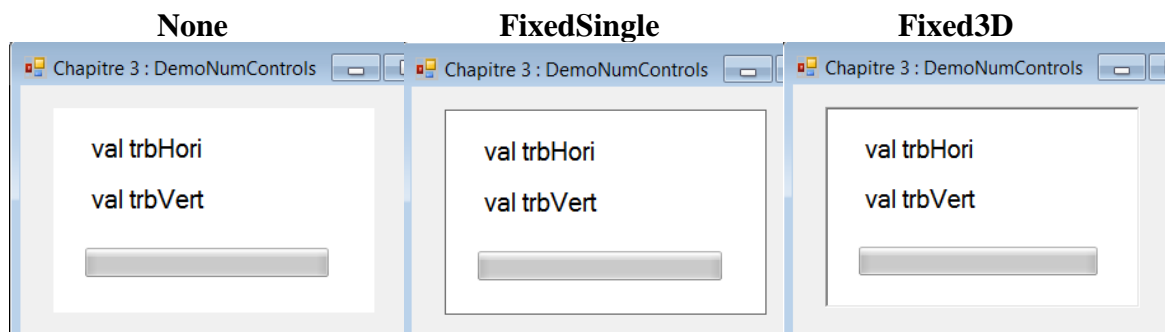
```
public Form1()
{
    InitializeComponent();
    panel1.BackColor = Color.White;
}
```

3.5.11.1. PROPRIETE BORDERSTYLE

La propriété **BorderStyle**, dont la valeur par défaut est **None**, permet d'ajouter un cadre simple (**FixedSingle**) ou relief (**Fixed3D**)




Effet de la propriété **BorderStyle** :




Remarque : lorsque l'on a placé des contrôles sur le Panel, si on déplace le Panel, les contrôles se déplacent avec lui.

3.5.12. LE CONTROLE NUMERICUPDOWN

 [NumericUpDown](#) Aide voir **NumericUpDown, classe**

Pour le nom du contrôle, propriété (Name), utilisez le préfixe **nud**

Le contrôle **NumericUpDown** Windows Forms se présente comme une zone de texte à laquelle sont associées deux flèches permettant l'incrément et le décrément de la valeur

numérique 

L'utilisateur peut augmenter et diminuer cette valeur numérique en cliquant sur les boutons Haut et Bas, en appuyant sur les touches ▲ et ▼ ou en tapant le nombre de leur choix. La touche ▲ rapproche ce nombre de la valeur maximale ; à l'inverse, la touche ▼ le rapproche de la valeur minimale.

3.5.12.1. PROPRIETES PRINCIPALES

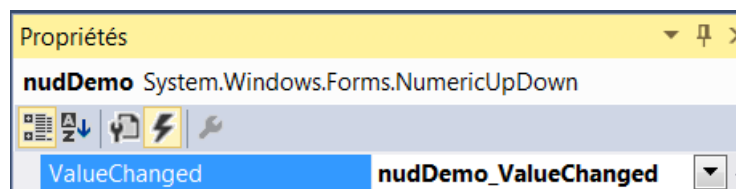
Value :	la valeur affichée
Maximum :	fixe la limite maximale de la valeur (par défaut 100)
Minimum :	fixe la limite minimale de la valeur (par défaut 0)
Increment :	valeur du pas (par défaut 1)
Hexadecimal :	si True affiche en hexadécimal, si False en décimal (par défaut False)
DecimalPlaces	Indique le nombre de décimales à afficher (défaut 0)
ThousandSeparator :	si True ajoute un séparateur de millier (par défaut False)

3.5.12.2. NUMERICUPDOWN, EVENEMENT PAR DEFAUT

L'événement par défaut est le changement de la valeur (**ValueChanged**). Un double clic sur le contrôle génère la méthode suivante :

```
private void nudDemo_ValueChanged(object sender, EventArgs e)
{
}
}
```

Au niveau de la liste des événements dans les propriété on trouve cet événement dans la catégorie action :

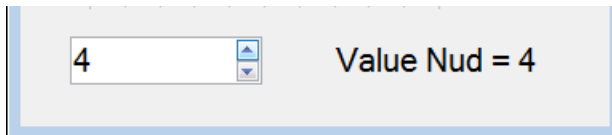


3.5.12.3. NUMERICUPDOWN, EXEMPLE

Dans cet exemple, lorsque l'événement ValueChanged se produit, on utilise la propriété Value pour l'afficher avec un label.

```
private void nudDemo_ValueChanged(object sender, EventArgs e)
{
    lblNud.Text = "Value Nud = " + nudDemo.Value.ToString();
}
```

Exemple de résultat :



⊗ la propriété Value est du type decimal, ce qui implique un cast si on souhaite l'affecter à un entier.

```
int ValNud = nudDemo.Value;
lblNud.Text = "Value Nud = " + ValNud.ToString();
```

(champ) NumericUpDown Form1.nudDemo

Impossible de convertir implicitement le type 'decimal' en 'int'. Une conversion explicite existe (un cast est-il manquant ?)

D'où finalement :


```
int ValNud = (int)nudDemo.Value;
lblNud.Text = "Value Nud = " + ValNud.ToString();
```

3.5.12.4. LE TYPE DECIMAL

Le mot clé **decimal** indique un type de données 128 bits. Par rapport aux types virgule flottante, le type **decimal** fournit une plus grande précision et une plage de valeurs plus réduite ; il est donc particulièrement approprié aux calculs financiers et monétaires. Le tableau suivant indique la plage de valeurs approximative et la précision fournies par le type **decimal**.

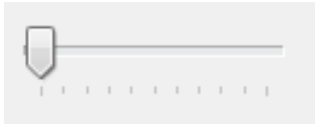
Type	Plage approximative	Précision	Type .NET Framework
decimal	$(-7.9 \times 10^{28} \text{ à } 7.9 \times 10^{28}) / (10^0 \text{ à } 28)$	28-29 chiffres significatifs	Decimal

3.5.13. LE CONTROLE TRACKBAR

 **TrackBar** Aide voir **TrackBar, classe**

Pour le nom du contrôle, propriété (Name), utilisez le préfixe **trb**

Le contrôle **TrackBar** Windows Forms (parfois appelé « contrôle Slider ») permet de naviguer dans un grand volume d'informations ou d'ajuster visuellement un paramètre numérique.



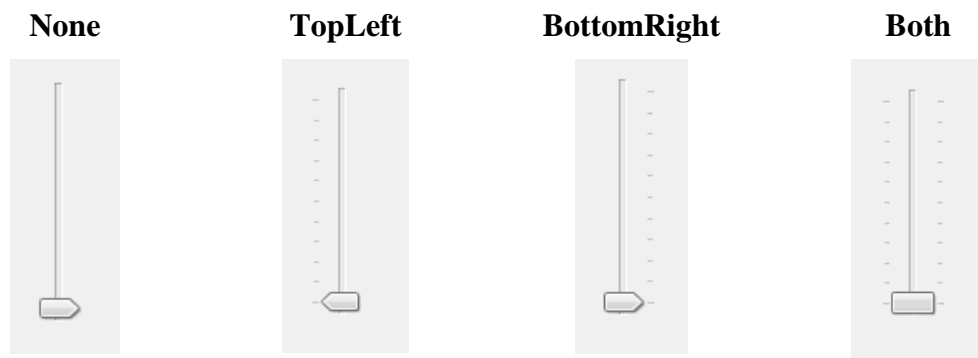
Le contrôle **TrackBar** est également constitué de deux parties : le curseur (de défilement) et les graduations. Le curseur de défilement peut être ajusté. Sa position correspond à la propriété **Value**. Les graduations sont des indicateurs visuels. Elles sont placées à intervalles réguliers. La barre de suivi se déplace selon les incréments que vous avez spécifiés. Elle peut être alignée horizontalement ou verticalement. Une barre de suivi peut être utilisée, par exemple, pour définir la fréquence de clignotement du curseur ou la vitesse de la souris.

3.5.13.1. PROPRIETES PRINCIPALES

Value :	la valeur sélectionnée
TickFrequency	détermine l'espacement des graduations. Le nombre de graduation correspond à la plage de valeur divisée par TickFrequency. (1 par défaut)
Maximum :	fixe la limite maximale de la valeur représentée (10 par défaut)
Minimum :	fixe la limite minimale de la valeur représentée (0 par défaut)
SmallChange	détermine le nombre de positions dont le curseur se déplace lorsque l'utilisateur appuie sur la touche GAUCHE ou DROITE
LargeChange	détermine le nombre de positions dont le curseur se déplace lorsque l'utilisateur appuie sur la touche HAUT ou BAS ou clique sur la barre de suivi d'un côté ou de l'autre du curseur.
Orientation	Orientation du TrackBar, Horizontal ou Vertical

3.5.13.2. PROPRIETES TICKSTYLE

La propriété **TickStyle** définit l'apparence du TrackBar. Voici un exemple pour les quatre valeurs d'un TrackBar Vertical. .



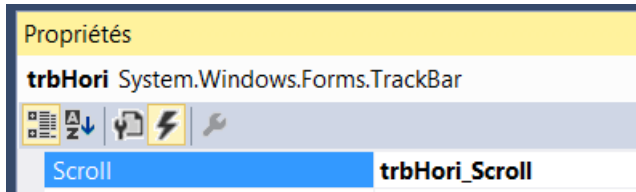
3.5.13.3. TRACKBAR, EVENEMENTS

L'événement par défaut est **Scroll**, (déplacement du curseur). Voici la méthode générée lors d'un double clic sur le contrôle.

```
private void trbHori_Scroll(object sender, EventArgs e)
{

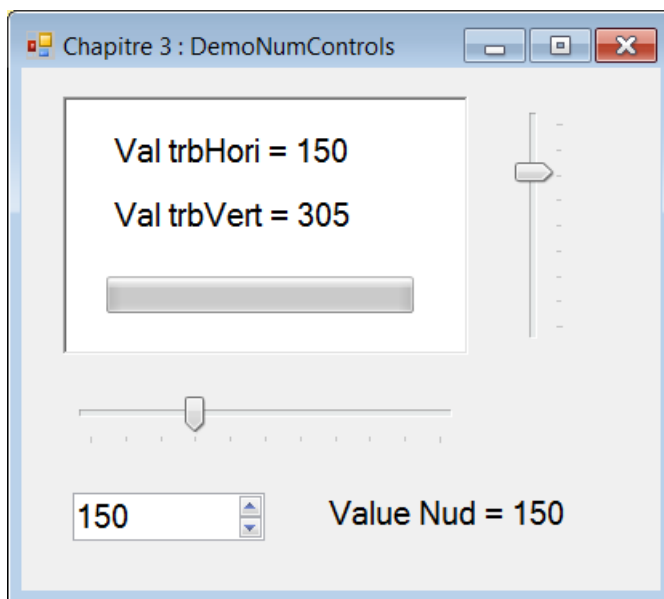
}
```

Au niveau de la liste des événements dans les propriétés on trouve cet événement dans la catégorie Comportement :



☞ Un autre événement intéressant est **ValueChanged**. (Changement de la valeur).

3.5.13.4. TRACKBAR, EXEMPLE



L'exemple ci-dessous utilise 2 TrackBars, pour établir les valeurs en X et Y, qui sont affichées par 2 labels placé dans le panel.

3.5.13.4.1. Code de l'exemple

```
namespace DemoNumControls
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            panel1.BackColor = Color.White;
            // Configuration du trackBar Hori
            trbHori.Maximum = 500;
            trbHori.TickFrequency = trbHori.Maximum / 10;
        }
    }
}
```

```
// Configuration du trackBar Vert
trbVert.Maximum = 400;
trbVert.TickFrequency = trbHori.Maximum / 10;
trbVert.Orientation = Orientation.Vertical;
trbVert.TickStyle = TickStyle.BottomRight;
// Synchro valeur Nud ave trbHori
nudDemo.Maximum = trbHori.Maximum;
}

private void trbVert_Scroll(object sender, EventArgs e)
{
    lblVert.Text = "Val trbVert = " +
                    trbVert.Value.ToString();
}

private void nudDemo_ValueChanged(object sender,
                                   EventArgs e)
{
    int ValNud = (int)nudDemo.Value;
    lblNud.Text = "Value Nud = " + ValNud.ToString();
    // Impose la valeur au trbHori
    trbHori.Value = (int)nudDemo.Value; ;
}

private void trbHori_Scroll(object sender, EventArgs e)
{
    lblHori.Text = "Val trbHori = " +
                    trbHori.Value.ToString();
}

// Événement supplémentaire nécessaire pour mise à jour
// du label
private void trbHori_ValueChanged(object sender,
                                   EventArgs e)
{
    lblHori.Text = "Val trbHori = " +
                    trbHori.Value.ToString();
}
}
```

3.5.13.4.2. remarques à propos l'exemple

C'est en essayant d'imposer la valeur du NumericUpDown à celle du TrackBar que l'on découvre 2 choses :

```
// Impose la valeur au trbHori
trbHori.Value = (int)nudDemo.Value; ;
```

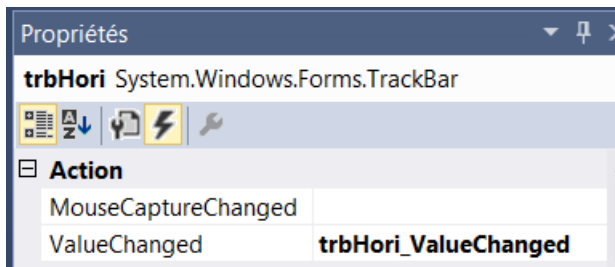
La propriété Value du TrackBar est du type **int**, d'où la nécessité d'un cast.


Lorsque on modifie la valeur du `TrackBar`, cela ne produit pas un événement `Scroll`, ce qui fait que le label n'est pas modifié. D'où la nécessité d'introduire un événement supplémentaire **ValueChanged** pour effectuer le même traitement que dans l'événement `Scroll`.

```
// Événement supplémentaire nécessaire pour mise à jour  
// du label  
private void trbHori_ValueChanged(object sender,  
                                EventArgs e)  
{  
    lblHori.Text = "Val trbHori = " +  
                  trbHori.Value.ToString();  
}
```

3.5.13.5. AJOUT MANUEL D'UN ÉVÉNEMENT

Pour ajouter manuellement un événement ou plus précisément sa méthode événementielle, il faut agir au niveau des propriété du contrôle.



Dans les propriété en ayant sélectionné la liste des événements () , il suffit d'introduire le nom de la méthode voulue et le système la génère automatiquement.

3.5.14. LE CONTROLE PROGRESSBAR

ProgressBar Aide voir **ProgressBar, classe**

Pour le nom du contrôle, propriété (Name), utilisez le préfixe **prb**

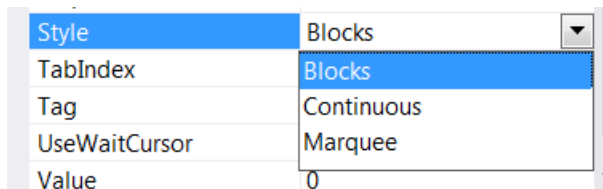
Le contrôle **ProgressBar** Windows Forms représente un contrôle de barre de progression Windows. Lorsque l'action est terminée, la barre est entièrement remplie.

3.5.14.1. PROPRIETES PRINCIPALES

Value :	la valeur à afficher
Maximum :	fixe la limite maximale de la valeur (100 par défaut)
Minimum :	fixe la limite minimale de la valeur (0 par défaut)
Step :	Valeur du pas lors de l'utilisation de la méthode PerformStep (10 par défaut)
Style :	Style de la présentation du ProgressBar.

3.5.14.2. PROPRIETES STYLE

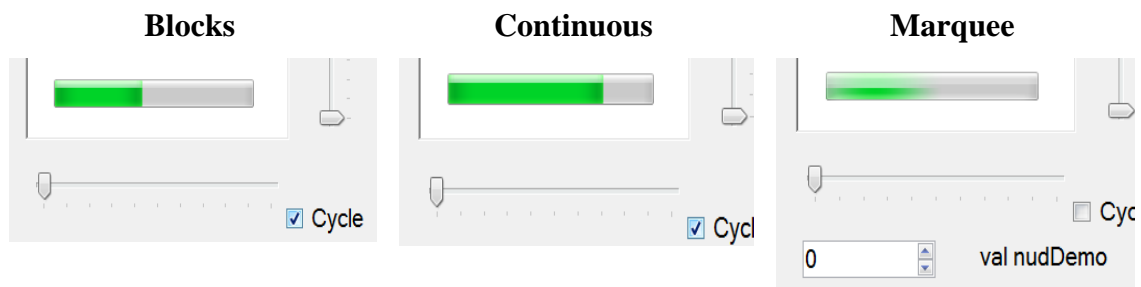
La propriété **Style** définit l'apparence du ProgressBar



Style

Cette propriété permet à l'utilisateur de définir le style ...

Voici un exemple pour les trois valeurs de cette propriété :



Avec le Style **Marquee**, on obtient un défilement montrant une progression mais sans relation avec l'état de la progression et sans modifier la valeur..

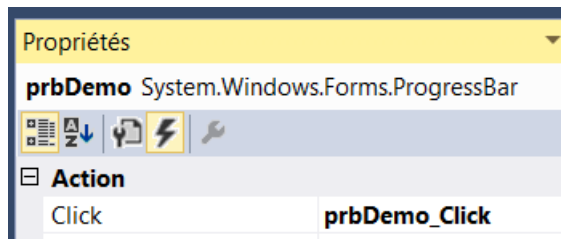
Le Style **Blocks** se comporte comme le **Continuous** depuis le Visual Studio 2010, cela est peut être lié à Windows 7.

3.5.14.3. PROGRESSBAR, EVENEMENT PAR DEFAULT

L'événement par défaut est **Click**. Voici la méthode générée lors d'un double clic sur le contrôle.

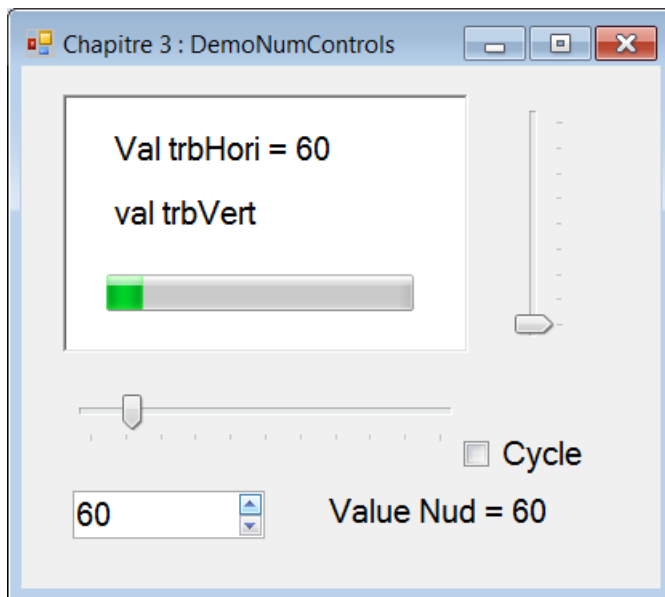
```
private void prbDemo_Click(object sender, EventArgs e)
{
}
}
```

Au niveau de la liste des événements dans les propriétés on trouve cet événement dans la catégorie Action.



Remarque : le progressBar est plus utilisé comme moyen d'affichage que comme moyen de fournir une valeur.

3.5.14.4. PROGRESSBAR EXEMPLE



Dans cet exemple la valeur affichée par le NumericUpDown est attribuée au ProgressBar. Si on coche la CheckBox Cycle la progression est obtenue par un Timer et la méthode PerformStep().

3.5.14.4.1. Code de l'exemple

Ajout dans le constructeur :

```
// Configuration du ProgressBar
prbDemo.Style = ProgressBarStyle.Continuous;
// Synchro valeur ProgressBar avec trbHori
prbDemo.Maximum = trbHori.Maximum;
```

Action dans événement ValueChanged du nudDemo :

```
// Impose la valeur au prbDemo
prbDemo.Value = (int)nudDemo.Value; ;
```

Action dans événement Tick du Timer :

```
private void timer1_Tick(object sender, EventArgs e)
{
    // Modifie valeur du ProgressBar par pas
    prbDemo.PerformStep();
}
```

Gestion du Timer1 dans l'événement CheckChanged du CheckBox :

```
private void chkCycle_CheckedChanged(object sender, EventArgs e)
{
    if (chkCycle.Checked == true)
    {
        // start le timer
        timer1.Start();
    } else
    {
        // stop le timer
        timer1.Stop();
    }
}
```

3.5.15. LE CONTROLE LISTBOX

ListBox Aide voir **ListBox, classe**

Pour le nom du contrôle, propriété (Name), utilisez le préfixe **lst**

Le contrôle **ListBox** de Windows Forms affiche une liste d'éléments dans laquelle l'utilisateur peut effectuer un ou plusieurs choix. Si le nombre d'éléments excède la capacité d'affichage, une barre de défilement est automatiquement ajoutée au contrôle **ListBox**.

Lorsque la propriété **MultiColumn** a la valeur **true**, la zone de liste affiche des éléments dans plusieurs colonnes et une barre de défilement horizontale s'affiche.

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

La propriété **SelectionMode** détermine le nombre d'éléments pouvant être sélectionnés en même temps.

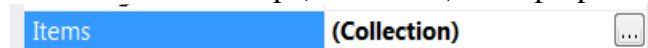
3.5.15.1. AJOUT/SUPPRESSION D'ELEMENTS DE LA LISTE PAR PROGRAMME


Pour ajouter ou supprimer des éléments dans un contrôle **ListBox**, utilisez la méthode **Items.Add**, **Items.Insert**, **Items.Clear**, **Items.Remove** ou **Items.RemoveAt**. Au moment du design, vous pouvez également ajouter des éléments à la liste en utilisant la propriété **Items**.

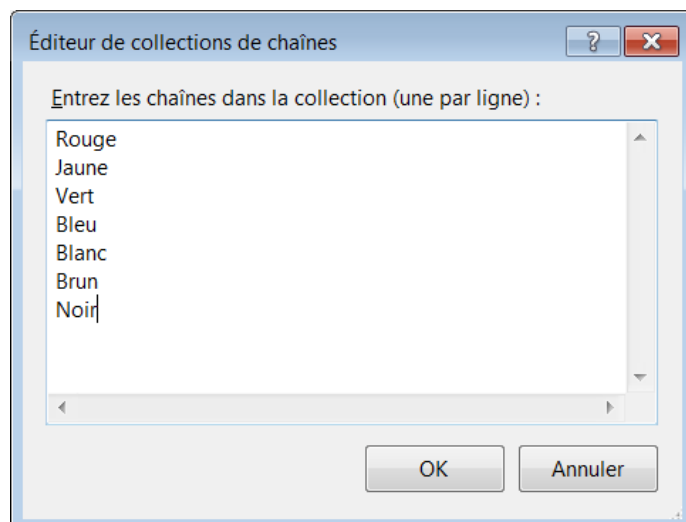
3.5.15.2. LISTBOX, PROPRIETE ITEMS

La propriété **Items** permet d'accéder à n'importe quel élément de la liste. Cette propriété contient une sorte de tableau (Collection) dont chacun des éléments correspond à un élément de la liste et est représenté sous forme de chaîne.

Il est possible d'introduire les éléments de la liste dans la propriété **Items**. Il faut sélectionner le champ (**Collection**) de la propriété **Items**.



L'activation du bouton  lance l'éditeur de collections de chaînes qui permet d'initialiser le contenu de la liste.



3.5.15.3. EFFET DES PROPRIETES SUR LA LISTBOX

Situation A : (En execution)

MultiColumn = False, SelectionMode = One



Zone plus haute que la longueur de la liste avec **ScrollAlwaysVisible = False**



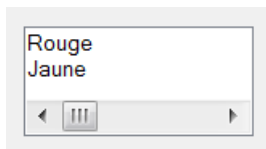
Idem mais **ScrollAlwaysVisible = True**



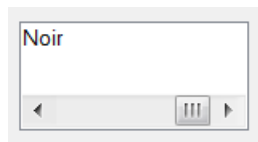
Zone plus courte que la longueur de la liste avec **ScrollAlwaysVisible = False**

Situation B : ScrollAlwaysVisible = False,

Déplacement par "tranche"

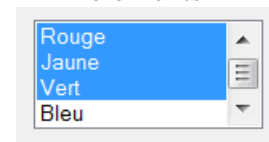


MultiColumn = **True**
(début de la liste)
SelectionMode = One



MultiColumn = **True**
(fin de la liste)
SelectionMode = One

Sélection de plusieurs éléments



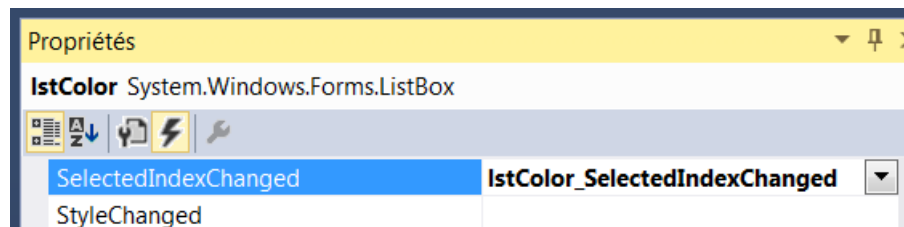
SelectionMode = **MultiSimple**
MultiColumn = False

3.5.15.1. LISTBOX, EVENEMENT PAR DEFAULT

L'événement par défaut est **SelectedIndexChanged**. Voici la méthode générée lors d'un double clic sur le contrôle.

```
private void lstColor_SelectedIndexChanged(object sender,
                                         EventArgs e)
{
}
}
```

Au niveau de la liste des événements dans les propriété on trouve cet événement dans la catégorie Comportement.

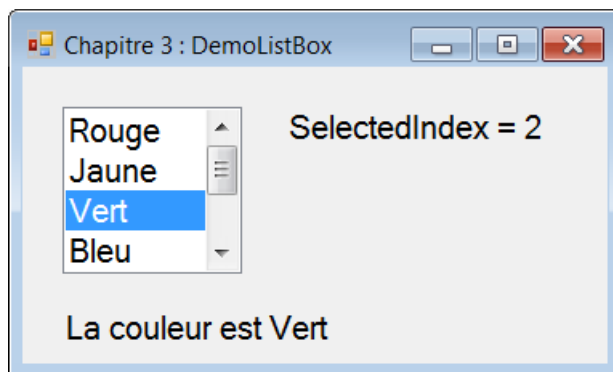


Ce qui correspond au changement de la position de l'élément sélectionné.

3.5.15.2. LISTBOX, EXEMPLE

Dans cet exemple on utilise l'événement **SelectedIndexChanged** pour afficher dans un Label la couleur sélectionnée et la valeur de **SelectedIndex** dans un autre Label.

```
private void lstColor_SelectedIndexChanged(object sender,
                                         EventArgs e)
{
    lblShow.Text = "La couleur est " + lstColor.SelectedItem;
    lblIndex.Text = "SelectedIndex = " +
                   lstColor.SelectedIndex.ToString();
}
```



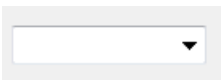
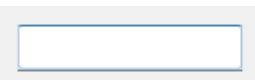

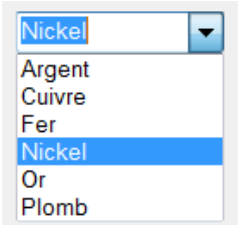
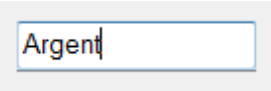
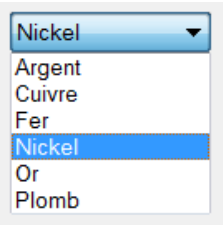
3.5.16. LE CONTROLE COMBOBOX

ComboBox Aide **ComboBox, classe**

Le contrôle **ComboBox** Windows Forms permet d'afficher des données dans une zone de liste déroulante fixe. Par défaut, le contrôle **ComboBox** comporte deux parties. Dans la partie supérieure figure une zone de texte dans laquelle l'utilisateur peut taper le nom d'un élément de la liste. Dans la deuxième partie, une zone de liste affiche une liste d'éléments dans laquelle l'utilisateur peut sélectionner un élément.

3.5.16.1. COMBOBOX, PROPRIETE DROPDOWNSTYLE

Elle définit le style de la zone de liste du ComboBox. Il y a 3 styles :

DropDown (défaut)	Simple	DropDownList
Liste modifiable déroulante	Liste modifiable simple	Zone de liste déroulante
En création :	En création :	En création :
		
En exécution	En exécution	En exécution
	 ⊖ pas de sélection possible ?	
Il est possible de modifier le texte affiché	Il est possible de modifier le texte affiché	N'est pas modifiable

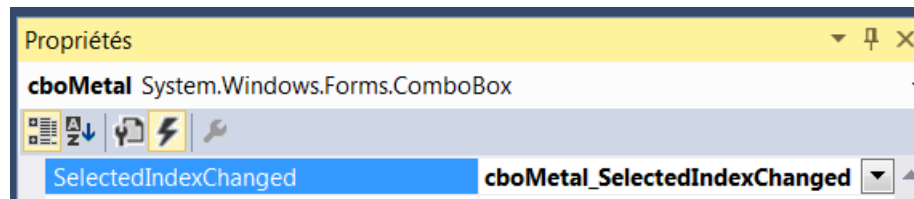
La différence entre DropDown et DropDownList est qu'il est possible de modifier la liste dans le cas du DropDown, alors que ce ne l'est pas pour le DropDownList.

3.5.16.2. COMBOBOX, EVENEMENT PAR DEFAUT

Comme pour la ListBox, l'événement par défaut est **SelectedIndexChanged**. Voici la méthode générée lors d'un double clic sur le contrôle. (compléter d'une action sur un Label).

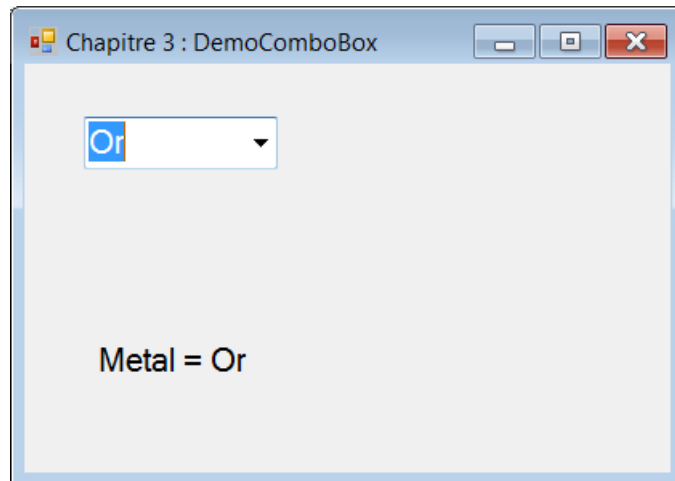
```
private void cboMetal_SelectedIndexChanged(object sender,
                                           EventArgs e)
{
    lblMetal.Text = "Metal = " + cboMetal.SelectedItem;
}
```

Au niveau de la liste des événements dans les propriétés on trouve cet événement dans la catégorie Comportement.



Ce qui correspond au changement de la position de l'élément sélectionné.

Exemple de résultat :

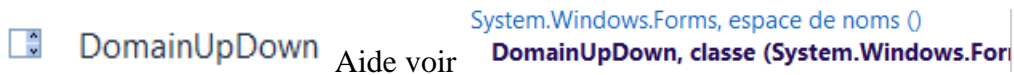


3.5.16.3. COMBOBOX, INITIALISATION DE LA LISTE A LA CONSTRUCTION

Voici le code ajouter au constructeur de Form1 pour créer la liste et se positionner sur le 1^{er} éléments.

```
public Form1()
{
    InitializeComponent();
    // Creation de la liste
    cboMetal.Items.Add("Argent");
    cboMetal.Items.Add("Cuivre");
    cboMetal.Items.Add("Fer");
    cboMetal.Items.Add("Nickel");
    cboMetal.Items.Add("Or");
    cboMetal.Items.Add("Plomb");
    cboMetal.SelectedIndex = 0;
}
```


3.5.17. LE CONTROLE DOMAINUPDOWN



Pour le nom du contrôle, propriété (Name), utilisez le préfixe **Dom**

Le contrôle **DomainUpDown** Windows Forms se présente comme une zone de texte à laquelle sont associés deux boutons permettant le déplacement vers le haut et vers le bas dans une liste. Il affiche et définit une chaîne de texte provenant d'une liste d'options. Pour sélectionner la chaîne, l'utilisateur peut se déplacer dans une liste en cliquant sur des boutons haut et bas, en appuyant sur les touches HAUT et BAS ou en tapant directement une chaîne correspondant à un élément de la liste. Ce contrôle peut notamment être utilisé pour sélectionner des éléments dans une liste de noms triés par ordre alphabétique (pour trier la liste, attribuez à la propriété **Sorted** la valeur **true**).

Ce contrôle a un fonctionnement très proche de celui des zones de liste simples ou déroulantes, mais il occupe un espace plus réduit.

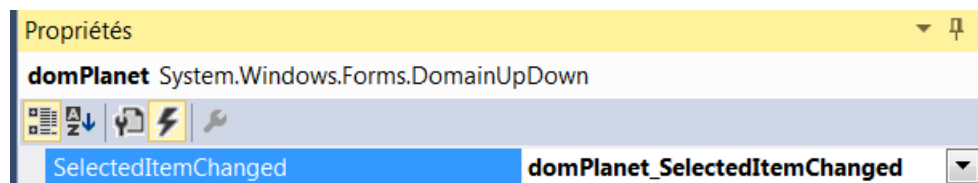
Les principales propriétés de ce contrôle sont **Items**, **ReadOnly** et **Wrap**. La propriété **Items** contient la liste des objets dont les valeurs sont affichées sous forme de texte dans le contrôle. Si la propriété **ReadOnly** a la valeur **false**, le contrôle complète automatiquement le texte tapé par l'utilisateur et l'associe à une valeur de la liste. Si la propriété **Wrap** a la valeur **true**, vous revenez au premier élément de la liste lorsque vous arrivez au dernier, et vice versa.

3.5.17.1. DOMAINUPDOWN, EVENEMENT PAR DEFAULT

Comme pour la **ListBox**, l'événement par défaut est **SelectedIndexChanged**. Voici la méthode générée lors d'un double clic sur le contrôle. (complétée d'une action sur deux **Label**).

```
private void domPlanet_SelectedIndexChanged(object sender,
                                           EventArgs e)
{
    lblSelected.Text = "SelectedItem : " +
                      domPlanet.SelectedItem;
    lblText.Text = "Text : " + domPlanet.Text;
}
```

Au niveau de la liste des événements dans les propriétés on trouve cet événement dans la catégorie Comportement.



Ce qui correspond au changement de la position de l'élément sélectionné.

3.5.17.2. CONTROLE DOMAINUPDOWN, EXEMPLE

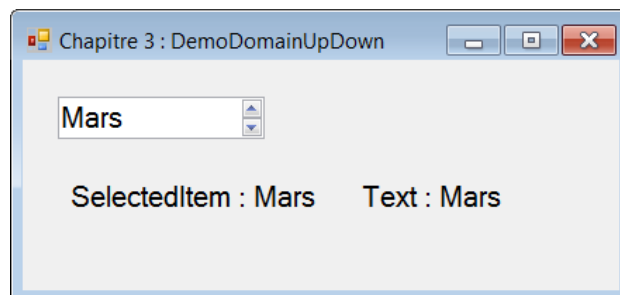
Pour illustrer l'utilisation de ce contrôle voici un exemple simple. Tout d'abord nous créons la liste des éléments non pas en éditant la collection, mais en utilisant du code placé dans le constructeur de la feuille. En exécution, sur la base de l'événement **SelectedItemChanged**, le texte sélectionné sera affiché dans deux Label, en utilisant la propriété `.Text` et la propriété `.SelectedItem`.

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        // Creation de la liste
        domPlanet.Items.Add("Mars");
        domPlanet.Items.Add("Mercure");
        domPlanet.Items.Add("Jupiter");
        domPlanet.Items.Add("Saturne");
        domPlanet.Items.Add("Terre");

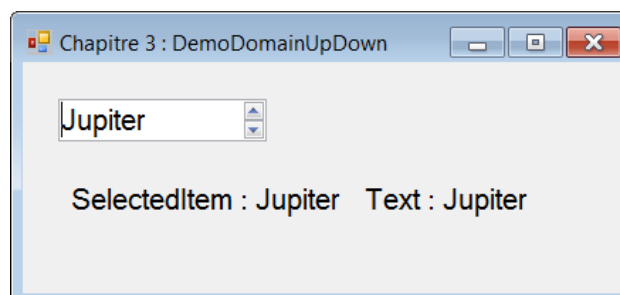
        // domPlanet.SelectedIndex = 0;
        // OU
        domPlanet.SelectedItem = "Mars";
    }

    private void domPlanet_SelectedItemChanged(
        object sender, EventArgs e)
    {
        lblSelected.Text = "SelectedItem : " +
                           domPlanet.SelectedItem;
        lblText.Text = "Text : " + domPlanet.Text;
    }
}
```

Situation au démarrage :



Situation après sélection :



3.5.18. COMBOBOX OU LISTBOX ?

Les contrôles **ComboBox** et **ListBox** ont des comportements similaires et sont dans certains cas interchangeables. Toutefois, dans certaines situations, l'un ou l'autre contrôle sera plus approprié à une tâche.

En général, une zone de liste déroulante est mieux adaptée à une liste de choix *proposés*, tandis qu'une zone de liste convient mieux aux cas où le choix doit être limité aux seuls éléments contenus dans la liste. Une zone de liste déroulante contient un champ de zone de texte dans lequel il est possible de taper des choix. Toutefois, cette règle ne se vérifie pas lorsque la propriété **DropDownStyle** est définie avec la valeur **ComboBoxStyle.DropDownList**. Dans ce cas, le contrôle sélectionne un élément correspondant à la première lettre que vous tapez.

En outre, les zones de liste déroulante économisent de la place dans un formulaire. En effet, comme la liste n'est pas entièrement affichée tant que l'utilisateur ne clique pas sur la flèche vers le bas, la zone de liste déroulante peut aisément tenir dans un espace restreint qui ne serait pas suffisant pour une zone de liste. Toutefois, cette règle ne se vérifie pas lorsque la propriété **DropDownStyle** est définie avec la valeur **ComboBoxStyle.Simple** : dans ce cas, en effet, la liste complète est affichée et la zone de liste déroulante occupe alors plus de place que ne le ferait une zone de liste.

3.5.19. AJOUT ET SUPPRESSION D'ELEMENTS D'UNE LISTBOX, COMBOBOX OU DOMAINUPDOWN

Voici une rubrique de l'aide :

Ajout et suppression d'éléments d'un contrôle ComboBox, ListBox ou CheckedListBox Windows Forms

Il existe de nombreuses façons d'ajouter des éléments à une zone de liste déroulante, une zone de liste ou une zone de liste de cases à cocher, car ces contrôles peuvent être liés à une infinité de sources de données. Toutefois, cette rubrique illustre la méthode la plus simple et suppose l'absence de liaison de données. Les éléments affichés sont généralement des chaînes ; cependant, il est possible d'utiliser n'importe quel objet. Le texte affiché dans le contrôle correspond à la valeur retournée par la méthode **ToString** de l'objet.

Pour montrer la généralisation, nous appliquons ces principes à un **DomainUpDown**.

3.5.19.1. POUR AJOUTER DES ELEMENTS

Les méthodes d'ajout d'élément sont : **Add**, **AddRange** et **Insert**

3.5.19.1.1. Ajout avec **Items.Add**

Ajoutez la chaîne ou l'objet à la liste au moyen de la méthode **Add** de la classe **ObjectCollection**. La collection est référencée à l'aide de la propriété **Items** :

```
domPlanet.Items.Add("Terre");
```

3.5.19.1.2. Ajout avec **Items.AddRange**

Pour ajouter plusieurs éléments en une fois à la liste, il est possible de créer un tableau d'élément et de l'ajouter en une fois.

```
string[] tabPlanet = { "Uranus", "Pluton", "Neptune"};  
domPlanet.Items.AddRange(tabPlanet);
```

On obtient les 3 nouveaux éléments en fin de liste.

3.5.19.1.3. Ajout avec **Items.Insert**

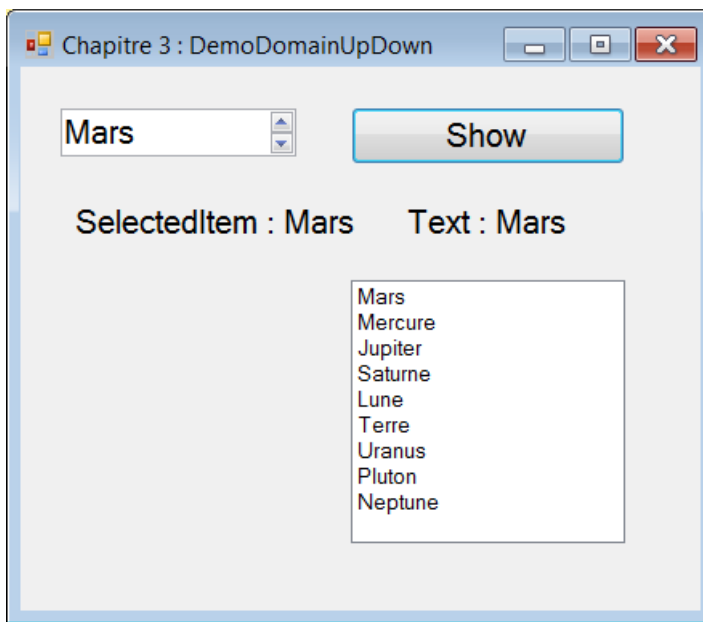
La méthode **Insert**, permet d'insérer un élément dans la liste, **avant** l'élément dont on donne l'indice.

Dans l'exemple ci-dessous on utilise la méthode **IndexOf** pour trouver l'indice d'un élément, puis on insert un nouvel élément.

```
int idx = domPlanet.Items.IndexOf("Terre");  
domPlanet.Items.Insert(idx, "Lune");
```

On obtiendra dans la liste Lune, Terre. Ce que l'on peut vérifier plus facilement en copiant le contenu de la collection **Items** de **domPlanet** dans une **ListBox**.

Le Button Show déclenche la copie est on obtient :



Le code pour effectuer la copie est le suivant :

```
private void btnShow_Click(object sender, EventArgs e)
{
    foreach (string element in domPlanet.Items)
    {
        lstPlanet.Items.Add(element);
    }
}
```

👉 cet exemple permet de découvrir la boucle **foreach** !

3.5.19.2. POUR ENLEVER UN ELEMENT

Appelez la méthode **Remove** ou **RemoveAt** pour supprimer des éléments.

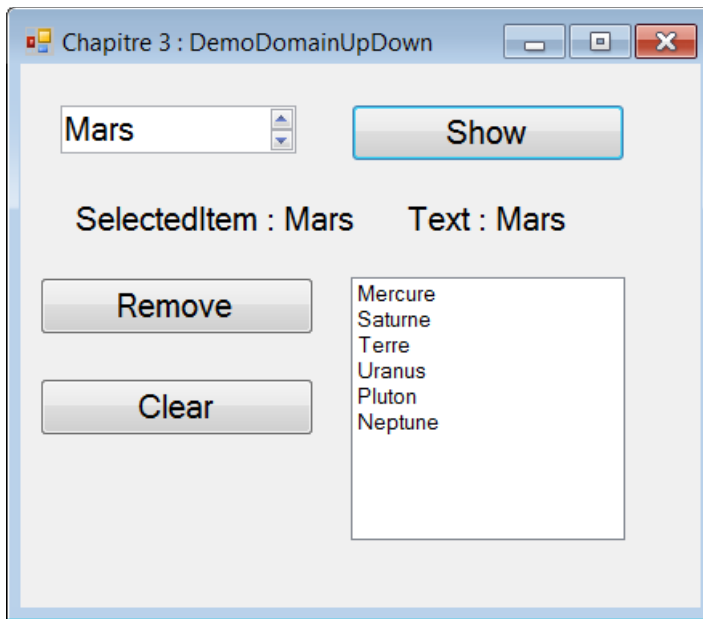
Remove possède un seul argument qui spécifie l'élément à supprimer.

RemoveAt supprime l'élément portant le numéro d'indice spécifié.

```
private void btnRemove_Click(object sender, EventArgs e)
{
    // Pour supprimer l'élément d'indice 2 (Jupiter):
    domPlanet.Items.RemoveAt(2);
    // Pour supprimer l'élément couramment sélectionné:
    domPlanet.Items.Remove(domPlanet.SelectedItem);
    // Pour supprimer l'élément "Lune":
    domPlanet.Items.Remove("Lune");
    lstPlanet.Items.Clear(); // pour permettre le control
}
```

Le résultat doit montrer une liste dans laquelle Mars, Jupiter et Lune ont été supprimés.

Résultat des suppression :



3.5.19.3. POUR SUPPRIMER TOUS LES ELEMENTS

Appelez la méthode **Clear** pour supprimer tous les éléments de la collection.

```
private void btnClear_Click(object sender, EventArgs e)
{
    domPlanet.Items.Clear();
    lstPlanet.Items.Clear(); // pour permettre le control
}
```

3.5.20. ACCES AUX ELEMENTS D'UNE LISTE

Quelques propriétés, uniquement disponibles lors de l'exécution, permettant l'accès à l'élément sélectionné ou aux éléments de la liste.

3.5.20.1. LA PROPRIETE **SELECTEDINDEX**

La propriété **SelectedIndex** retourne un entier (valeur de l'indice) correspondant au premier élément sélectionné dans la zone de liste. Vous pouvez modifier par programme l'élément sélectionné en changeant la valeur de la propriété **SelectedIndex** dans le code ; l'élément de liste correspondant est affiché en surbrillance dans le Windows Form.

Si aucun élément n'est sélectionné, la valeur de la propriété **SelectedIndex** est -1.

Si le premier élément est sélectionné, la propriété **SelectedIndex** prend la valeur 0.

Lorsque plusieurs éléments sont sélectionnés, la valeur de la propriété **SelectedIndex** correspond au rang du premier élément sélectionné dans la liste.

3.5.20.2. LA PROPRIETE **SELECTEDITEM**

La propriété **SelectedItem** est similaire à la propriété **SelectedIndex**, à cela près qu'elle retourne l'élément lui-même, en général une valeur de chaîne.

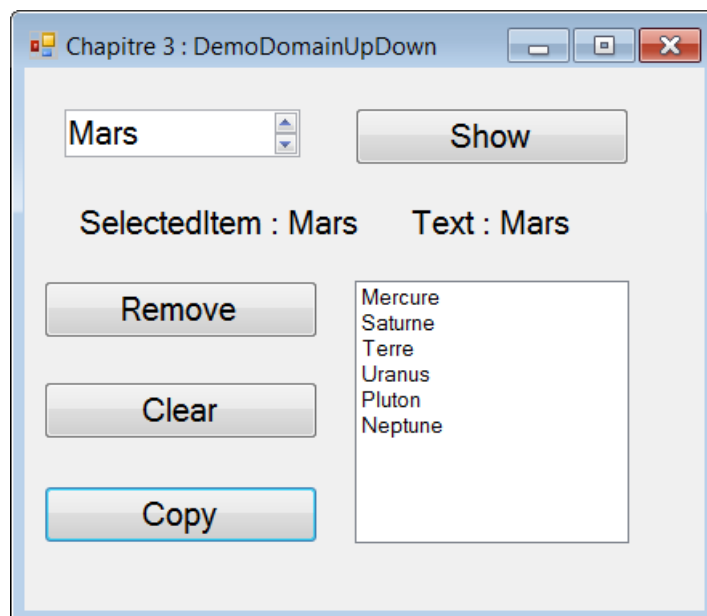
3.5.20.3. LA PROPRIETE ITEMS.COUNT

La propriété **Items.Count** indique le nombre d'éléments contenus dans la liste et sa valeur est toujours supérieure d'une unité à celle de la plus haute valeur de **SelectedIndex**, et cela parce que la propriété **SelectedIndex** est numérotée à partir de zéro.

3.5.20.4. UTILISATION DE ITEMS COMME UN TABLEAU

Voici un exemple montrant l'utilisation de la propriété **Items** comme un tableau en utilisant **Items.Count** pour déterminer l'indice maximum.

```
private void btnCopy_Click(object sender, EventArgs e)
{
    lstPlanet.Items.Clear(); // pour ne pas cumuler
    for (int idx = 0; idx < domPlanet.Items.Count; idx++)
    {
        lstPlanet.Items.Add(domPlanet.Items[idx]);
    }
}
```



3.6. LES METHODES EVENEMENTIELLES

Elles établissent le lien entre un événement qui survient à un contrôle et l'action que l'on veut réaliser lorsque celui-ci se produit.

Lorsque l'on double click sur un contrôle en mode création, Visual C# génère une méthode événementielle correspondante :

Voici quelques cas courant :

3.6.1. DEMARRAGE DE LA FEUILLE

Avec le C# l'événement de démarrage n'existe pas il est remplacé par le constructeur du formulaire.

Exemple de génération :

```
public Form1()
{
    InitializeComponent();
    // Creation de la liste
    domPlanet.Items.Add("Mars");
    domPlanet.Items.Add("Mercure");
}
```

☺ toute les initialisation à exécuter une seule fois sont à placer après l'appel de **InitializeComponent();**

3.6.2. EVENT CLICK

Se produit lorsque on clique sur un contrôle, le plus courant étant le Button.

Exemples de génération :

```
private void btnCopy_Click(object sender, EventArgs e)
{
}
}
```

3.6.3. EVENT CHANGE

Se produit lorsqu'on modifie le texte par exemple d'un TextBox. S'applique aussi au RadioButton et CheckBox.

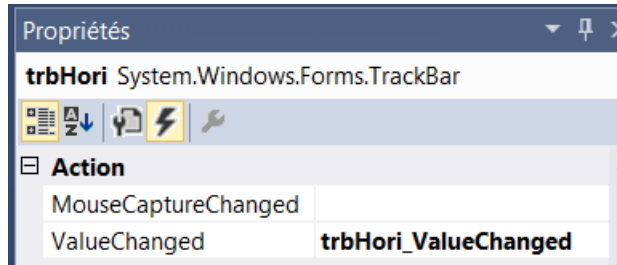
Exemples de génération :

```
private void txtTest1_TextChanged(object sender, EventArgs e)
{
}

private void chkThreeState_CheckedChanged(object sender,
                                           EventArgs e)
{
}
}
```


3.6.4. SELECTION D'UN EVENEMENT SPECIFIQUE

Pour ajouter manuellement un événement ou plus précisément sa méthode événementielle, il faut agir au niveau des propriété du contrôle.

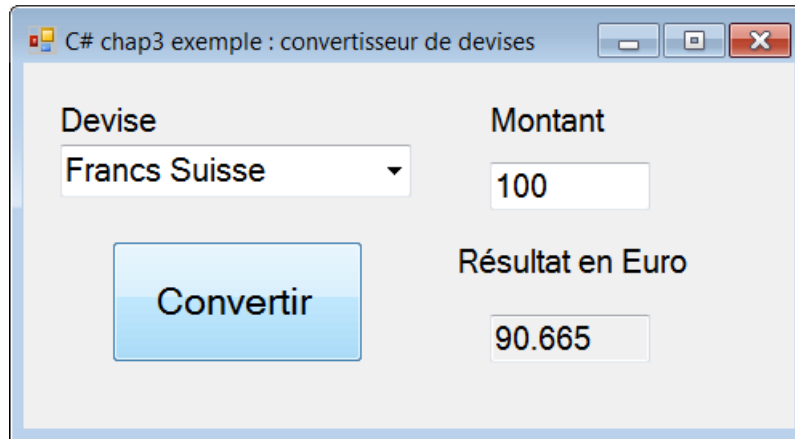


Dans les propriété en ayant sélectionné la liste des événements (⚡), il suffit d'introduire le nom de la méthode voulue et le système la génère automatiquement.

3.7. EXEMPLE : CONVERTISSEUR DE DEVISES

Pour mieux illustrer les éléments mentionnés dans ce chapitre, nous allons suivre pas à pas la réalisation d'une application Windows contenant une feuille avec quelques contrôles.

Voici un convertisseur de quelques devises en Euro :



La devise est sélectionnée par un ComboBox, le montant à convertir dans une TextBox. Le résultat en Euro est affiché dans une deuxième "TextBox" lorsqu'on active le Button libellé "Convertir". Trois Label sont utilisés comme légende.

3.7.1. CONFIGURATION DU COMBOBOX

Pour l'utiliser facilement par la suite on nomme le ComboBox, propriété (Name) = `cboDevises`. (**cbo** est le préfixe conforme aux conventions de codage).

Les différentes devises sont introduites dans la propriété **Items**, au niveau du code de la manière suivante:

```
public Form1()
{
    InitializeComponent();
    cboDevises.Items.Add("Francs Suisse");
    cboDevises.Items.Add("US Dollar");
    cboDevises.Items.Add("Livre Sterling");
    cboDevises.Items.Add("Yen japonais");
    cboDevises.SelectedIndex = 0; // positionne francs suisse
}
```

3.7.2. CONFIGURATION DES TEXTBOX

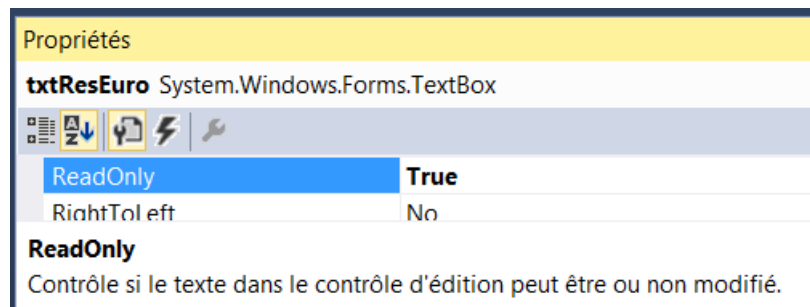
On configure uniquement la propriété (Name) en utilisant le préfixe **txt**.

Les noms choisis sont `txtMontant`, le montant de la devise à convertir et `txtResEuro` pour le résultat en Euro.

Remarque : lorsqu'on utilise le TextBox pour gérer des valeurs numériques, il faut configurer la propriété `Text = 0`. Pour `txtMontant` la valeur 100 convient bien pour éviter un montant nul.

3.7.2.1. TEXTBOX EN READONLY

Lorsque qu'un TextBox est utilisé uniquement pour de l'affichage il peut être configuré en lecture seule, en établissant à **True** la propriété **ReadOnly**.



3.7.3. CONFIGURATION DES LABELS

Il n'est pas nécessaire lorsque l'on ne prévoit pas d'accéder au label depuis le code de modifier la propriété (Name). Si on modifie il est recommandé d'utiliser le préfixe **lbl**.

C'est par la propriété `.Text` que l'on configure le texte à afficher.

3.7.4. CONFIGURATION DU FORM

Pour obtenir le titre dans la barre du dialogue, on configure la propriété `Text = "C# chap3 exemple : convertisseur de devises"`.

Remarque : c'est en positionnant le curseur dans la zone d'édition (sans sélectionner de contrôle) que l'on accède à la propriété de la "Form".

3.7.5. CONFIGURATION DU BUTTON

On configure la propriété (Name) en utilisant le préfixe **btn**, le nom choisi étant `btnConvertir`.

C'est par la propriété `Text` que l'on configure le texte affiché au centre du bouton, soit "Convertir". Il est possible de modifier la police en utilisant la propriété `.Font`.

3.7.6. CODE LIE AU BUTTON

Par un double clic sur le bouton on ouvre la fenêtre code est on obtient la méthode événementielle par défaut :

```
private void btnConvertir_Click(object sender, EventArgs e)
{
}
```

3.7.7. CODE COMPLET DE L'EXEMPLE

```

using System;
using System.Windows.Forms;

namespace ExampleDeviceConverter
{
    public partial class Form1 : Form
    {
        double[] tauxConversion = new double[4];

        public Form1()
        {
            InitializeComponent();
            cboDevises.Items.Add("Francs Suisse");
            cboDevises.Items.Add("US Dollar");
            cboDevises.Items.Add("Livre Sterling");
            cboDevises.Items.Add("Yen japonais");
            cboDevises.SelectedIndex = 0; // positionne SFR

            tauxConversion[0] = 0.90665; // taux SFR
            tauxConversion[1] = 0.894387; // taux USD
            tauxConversion[2] = 1.289078; // taux GBP
            tauxConversion[3] = 0.00775; // taux Yen japonais

            txtMontant.Text = "100"; // évite des problèmes
        }

        private void btnConvertir_Click(object sender,
                                         EventArgs e)
        {
            double montant, resEuro;
            if (cboDevises.SelectedIndex >= 0 &&
                cboDevises.SelectedIndex <= 3)
            {
                montant = double.Parse(txtMontant.Text);
                resEuro = montant *
                    tauxConversion[cboDevises.SelectedIndex];
                txtResEuro.Text = resEuro.ToString();
            }
        }
    }
}

```

La solution est basée sur l'usage d'une table de conversion et de sa correspondance avec l'index de la liste des textes de la ComboBox.

☹ à noter la déclaration spécifique du tableau :

```
double[] tauxConversion = new double[4];
```

Quelques remarques : on obtient le rang dans la liste de la ComboBox par la propriété **SelectedIndex**, le premier texte ayant l'indice 0. Ce qui permet d'écrire directement :

tauxConversion[cboDevises.SelectedIndex]

Il est nécessaire de convertir en numérique la valeur entrée sous forme de texte dans le TextBox txtMontant. D'où :

```
double montant;  
montant = double.Parse(txtMontant.Text);
```

Pour afficher la valeur en Euro il est nécessaire de convertir la valeur numérique en texte. D'où :

```
txtResEuro.Text = resEuro.ToString();
```

3.8. HISTORIQUE

3.8.1. VERSION 1.0 FEVRIER 2016

Traduction du chapitre 3 du VB 2010 avec suppression de certains éléments pour alléger. Version finale obtenue le 10.03.2016.