Concepteur Développeur en Informatique

Développer des composants d'interface

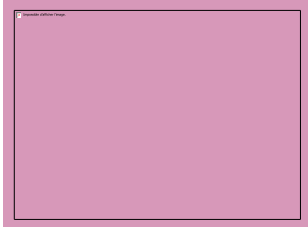
Formulaires Windows – Partie 4

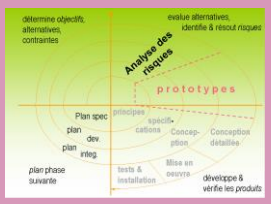
Accueil

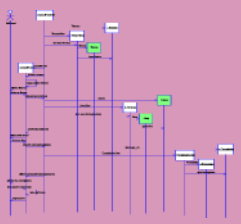
Apprentissage

PAE

Evaluation







Localisation : U03-E03-S02

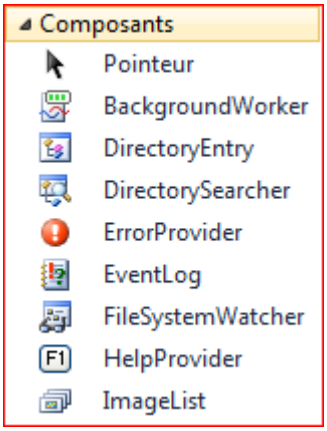
SOMMAIRE

1 Introduction

Ce document introduit de nouveaux contrôles de liste qui vous permettront d'apporter une plus grande richesse fonctionnelle à vos applications Windows.
Ce document ne saurait en aucun cas traiter de manière exhaustive des propriétés, méthodes et événements de ces différents contrôles ou composants.
Il ne s'agit pas d'un guide de référence mais d'une illustration de l'utilisation de ces contrôles.

2 Le composant ImageList

Vous trouverez ce composant dans l'onglet Composants de la boite à outils de Visual Studio



Utilisé avec des contrôles qui gèrent des listes d'objets associés à des images tels que les **ListView** ou les **TreeView** que vous allez découvrir dans ce document.

Le composant **ImageList** peut être alimenté durant la phase de conception ou lors de l'exécution du programme.

Les images « stockées » dans un composant **ImageList** sont accessibles par leur indice dans la collection Images.

La classe **Imagelist** est un composant et non un contrôle. Les instances de cette classe n'ont pas pour objet d'être affichées sur un formulaire.

Si vous observez le fichier généré en mode Design lors du dépôt de ce composant sur le formulaire, vous constaterez que l'instance de ce denier est associée à un conteneur de composants **components** et non à la collection des contrôles **controls** du formulaire.

Ce composant ne dispose pas de sa propre interface graphique et n'est pas ajouté à la classe du formulaire mais au conteneur.

```
this.components = new System.ComponentModel.Container();
this.ilDrapeaux = new System.Windows.Forms.ImageList(this.components);
this.SuspendLayout();
```

Observez la hiérarchie d'héritage pour

Ce contrôle permet de charger des listes d'images à partir d'un fichier de ressources ou d'un répertoire. Nous aborderons les fichiers de ressources plus en détails ultérieurement notamment lors de la régionalisation des interfaces.

2.1 Chargement des images RunTime

Dans cet exemple, j'ajouterai des images stockées sur disque.

La classe **ImageList** permet de gérer une collection d'images. Elle possède donc les méthodes et propriétés d'une collection.

Il est nécessaire de préciser la taille des images contenues dans la collection via la propriété **ImageSize**.

Chaque image est créée à partir d'un flux d'octets, ici en provenance d'un fichier sur le disque.

La classe **Image** dispose d'une méthode **FromFile.**

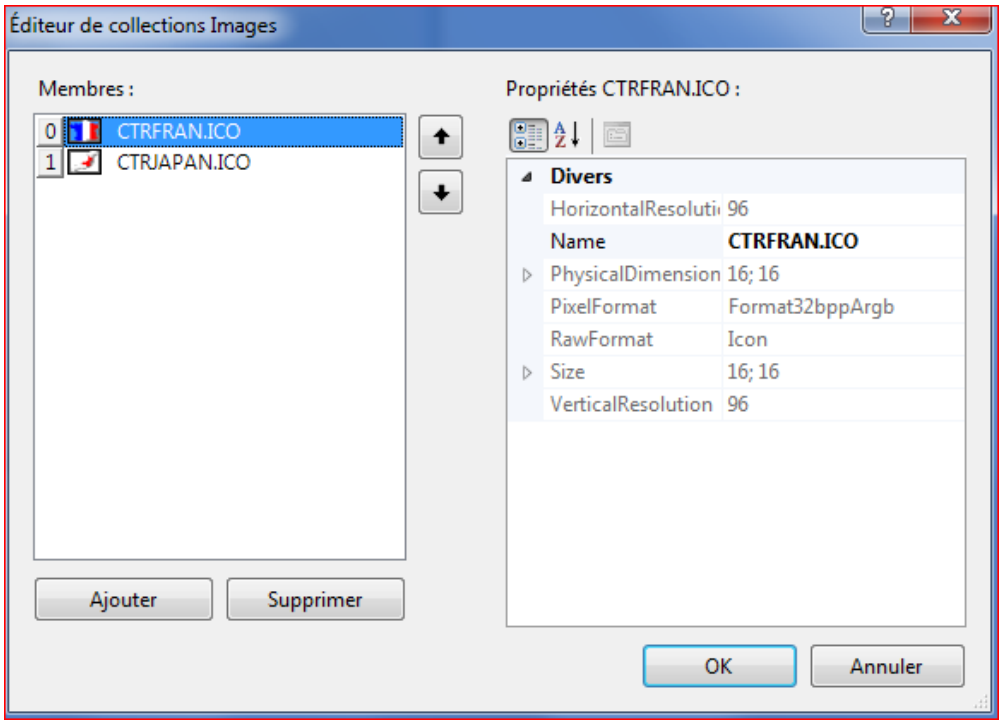
Il vous faudra donc alimenter la liste des images utiliser les techniques de gestion de fichiers de l'espace System.IO.

```
ilDrapeaux.ImageSize = new Size(75, 75);
string[] images = System.IO.Directory.GetFiles(@"D:\Drapeaux");
foreach (string pathImage in images)
{
    ilDrapeaux.Images.Add(Image.FromFile(pathImage));
}
```

2.2 Chargement des images lors de la conception (Designtime).

Il est aussi possible d'utiliser l'éditeur de collections d'images, accessible à partir de la fenêtre propriétés du composant ImageList.

Si les images sont chargées lors de la conception,en DesignTime, elles seront stockées dans un fichier de ressources et embarquées avec l'assemblage lors du déploiement de l'application.



3 Le contrôle ListView

Le contrôle **ListView** permet d'afficher une liste d'éléments avec, pour chaque élément, une icône et plusieurs colonnes de texte.

Le volet droit de l'explorateur Windows illustre bien les possibilités offertes par ce contrôle : l'icône représente le type de document et les colonnes textuelles ses propriétés.

Un contrôle **ListView** comporte une collection d'éléments **ListViewItem** qui représente chacun un élément dans la liste.

Chaque élément **ListViewItem** peut contenir une collection de sous-éléments **ListViewSubItem** qui complètent sa définition.

Les éléments de la liste peuvent être affichés de quatre manières différentes.

Ils peuvent apparaître sous la forme de :

- grandes icônes,
- petites icônes,
- petites icônes dans une liste verticale
- vue de détails.

Le quatrième mode d'affichage, la vue Détails, permet d'afficher l'élément et ses sous-éléments dans une grille munie d'en-têtes de colonne qui identifient les données affichées pour les sous-éléments.

Le texte des éléments peut être directement modifiable dans la liste.

Ce n'est toutefois pas l'usage courant, la liste étant plutôt utilisée en consultation seule.

Comme les autres contrôles de type liste, **ListView** prend en charge la sélection unique ou multiple.

La fonctionnalité de sélection multiple permet à l'utilisateur de sélectionner des éléments de la liste d'une manière analogue à un contrôle **ListBox**.

On peut par ailleurs, grâce à la propriété **CheckBoxes**, offrir la possibilité à l'utilisateur de sélectionner des éléments en cochant ces derniers.

Le contrôle **ListView** peut être utilisé dans maints contextes et être couplé avec le composant **ImageList** ou, comme dans le cas de l'explorateur, associé à un contrôle **TreeView**.

Pour plus d'informations sur le contrôle ListView, reportez –vous au chapitre 3 du livre du livre Windows Forms Application Development, page 98 ou consultez l'aide de msdn.

Quelques propriétés remarquables

Propriétés	
AllowColumnReorder	Indique si l'utilisateur peut réorganiser la présentation des colonnes
MultiSelect	Sélection multiple autorisée ou non
FullRowSelect	Indique si la sélection d'un item provoque la sélection de ses items enfants (sub-items)
GridLines	Affiche ou non un séparateur de colonnes
Sorting	Définir ou obtenir l'ordre de tri
View	Définir ou obtenir la liste de vue
SmallImageList	Composant à utiliser lors de l'affichage des éléments sous forme de petites icônes dans le contrôle.
LargeImageListe	Composant à utiliser lors de l'affichage des éléments sous forme de grandes icônes dans le contrôle.

3.1 Liste avec une seule colonne

Dans cet exemple, nous afficherons la liste des images associées à une colonne textuelle qui précisera le nom de l'image.
Nous souhaitons pouvoir afficher les quatre modes proposées par la liste. Nous devrons donc disposer de deux composants **ImageList**, qui seront respectivement affectés aux propriétés **SmallImageList** pour les petites icônes, et **LargeImageListe** pour les grandes icônes.

Nous ajouterons un entête de colonne qui permettra de préciser la nature des informations affichées dans la liste et la manière dont elles sont présentées. L'entête de colonne d'une ListView est une instance de la classe **ColumnHeader**.
Parmi les propriétés principales de ce type, la propriété Width qui peut être définie en pixels ou à :

- -1 pour adapter la largeur de l'élément le plus long dans la colonne
- -2 pour dimensionner automatiquement la largeur du titre de colonne

Définition des propriétés de base de la liste

```
lvDrapeaux.LabelEdit = false;
lvDrapeaux.AllowColumnReorder = false;
lvDrapeaux.MultiSelect = false;
lvDrapeaux.FullRowSelect = true;
lvDrapeaux.GridLines = false;
lvDrapeaux.Sorting = SortOrder.None;
```

Ajout d'un entête de colonne.

```
ColumnHeader enteteColonne = new ColumnHeader();
enteteColonne.Text = "Nom Image";
enteteColonne.TextAlign = HorizontalAlignment.Left;
enteteColonne.Width = 100;
lvDrapeaux.Columns.Add(enteteColonne);
```

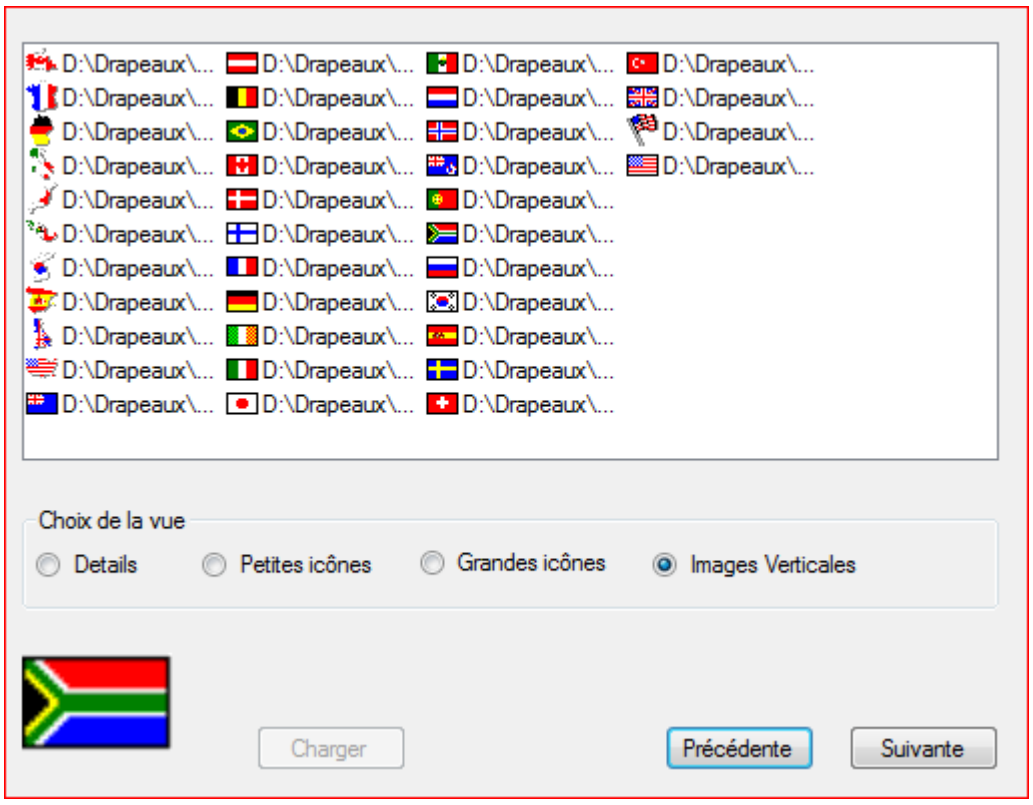
Ajout des éléments à la liste. Il s'agit ici du nom de l'image. Les images sont chargées à partir des fichiers d'un répertoire dans les deux listes d'images. L'association entre l'élément de la **ListeView** et l'image est réalisée lors de l'instanciation du **ListViewItem** via **l'index i**. Ici, chaque élément de la liste est associé à une image différente. Il s'agit d'un cas particulier.

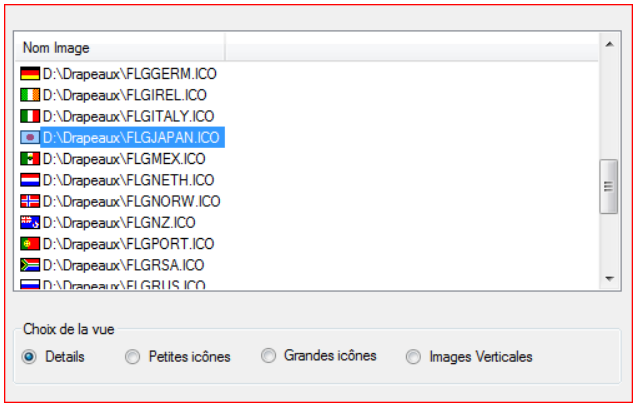
```
images = System.IO.Directory.GetFiles(@"D:\Drapeaux");
for (int i = 0; i < images.Length;i++)
{
    ilDrapeaux.Images.Add(Image.FromFile(images[i]));
    ilDrapeauxPetits.Images.Add(Image.FromFile(images[i]));
    lvDrapeaux.Items.Add(new ListViewItem(images[i].TrimEnd(),i));
}
```

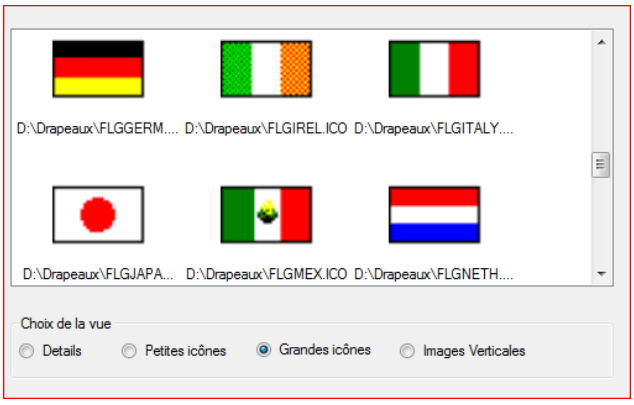
Association des listes d'images :

```
lvDrapeaux.SmallImageList = ilDrapeauxPetits;
lvDrapeaux.LargeImageList = ilDrapeaux;
```

Les différentes options d'affichage : Ici, disposition verticale sur plusieurs colonnes







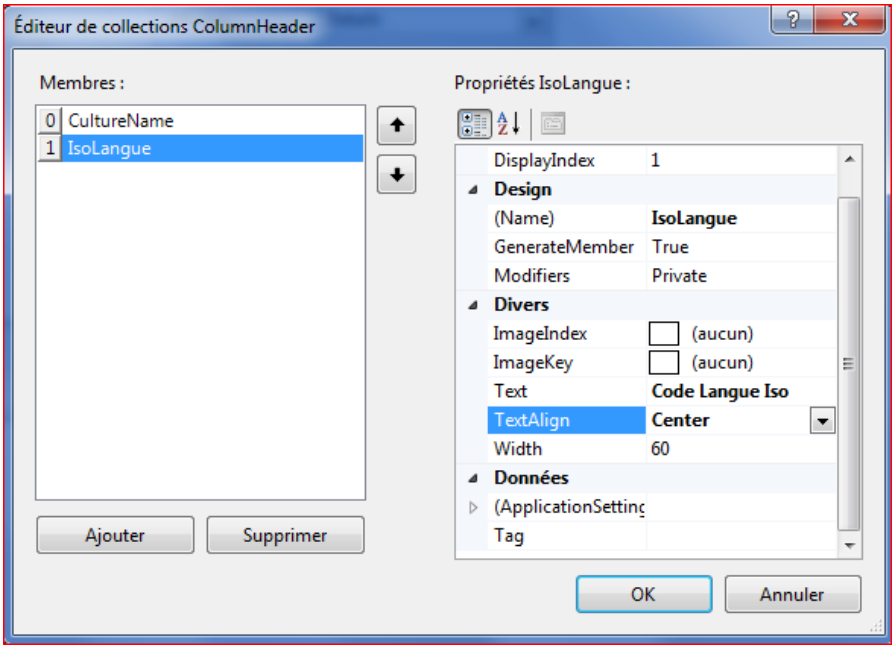
3.2 Liste multi-colonnes

Dans ce nouvel exemple, nous afficherons une vue des détails des cultures présentes sur notre ordinateur qui pourront être sélectionnées par l'intermédiaire d'une case à cocher.

3 propriétés seront affichées : le nom de la culture, le code langue de la culture, son codePage ANSI.

L'affichage de plus d'une propriété est réalisé par l'adjonction d'éléments enfants à l'élément principal. Ici, l'élément principal sera le nom de la culture.

Notez qu'il faut toujours définir les entêtes de colonnes avant de charger les éléments de la liste.
Si vous ne respectez pas cette règle, votre vue ne sera pas correctement présentée.



Ici les colonnes sont définies en Mode Design.

L'affichage de plus d'une propriété est réalisé par l'adjonction d'éléments enfants à l'élément principal :

- L'élément principal de la liste est un objet de type **ListViewItem**
- L'élément secondaire est un **ListViewItem.ListViewSubItem**. Il ne peut exister en dehors de l'item principal.

```
foreach (System.Globalization.CultureInfo culture in tabCultures)
{
    ListViewItem item = new ListViewItem(culture.DisplayName);
    item.SubItems.Add(culture.TwoLetterISOLanguageName);
    item.SubItems.Add(culture.TextInfo.ANSICodePage.ToString());
    listView1.Items.Add(item);
}
```

Les éléments dans la liste pourront être sélectionnés par click ou en cochant la case (propriété **CheckBoxes**) et la sélection de plusieurs éléments sera autorisée.

La réorganisation de l'ordre des colonnes (propriété **AllowColumnReorder**) sera autorisée afin d'en montrer les impacts.

```
System.Globalization.CultureInfo[] tabCultures =
    System.Globalization.CultureInfo.GetCultures(System.Globalization.CultureTypes.AllCultures);

lvCultures.View = View.Details;
lvCultures.AllowColumnReorder = true;
lvCultures.CheckBoxes = true;
lvCultures.MultiSelect = true;

foreach (System.Globalization.CultureInfo culture in tabCultures)
{
    ListViewItem item = new ListViewItem(culture.DisplayName);
    item.SubItems.Add(culture.TwoLetterISOLanguageName);
    item.SubItems.Add(culture.TextInfo.ANSICodePage.ToString());
    lvCultures.Items.Add(item);
}
```

Le résultat suivant est alors obtenu :

Nom de la culture	Code Langue Iso	Code Page
☐ Arabe	ar	1256
☐ Bulgare	bg	1251
☑ Catalan	ca	1252
☐ Chinois (Simplifié)	zh	936
☐ Tchèque	cs	1250
☑ Danois	da	1252
☐ Allemand	de	1252
☑ Grec	el	1253
☐ Anglais	en	1252
☐ Espagnol	es	1252
☐ Finnois	fi	1252
☐ Français	fr	1252
☐ Hébreu	he	1255
☐ Hongrois	hu	1250
☐ Islandais	is	1252
☐ Italien	it	1252

Vous pourriez souhaiter trier éventuellement les éléments en fonction de la colonne sélectionnée.

Pour le tri, seul le tri selon le texte de l'élément principal est fourni de base.

Mais vous pouvez, en vous appuyant sur les techniques de programmation objet, proposer votre propre mécanisme de tri. Pour cela il vous fait mettre en place une méthode de comparaison des valeurs présentes dans la colonne sur laquelle le tri portera.

Nous allons donc nous appuyer sur l'interface **IComparer** qui permet de comparer les éléments d'un tableau et sur laquelle s'appuie les méthodes **Sort().**

Nous créons un nouveau type interne à notre formulaire **ListViewItemComparer** avec deux constructeurs :

- Un constructeur vide, tri par défaut sur la colonne 0.
- Un constructeur avec en argument l'index de la colonne

Quand la méthode **Sort** sera invoquée, elle réalisera la comparaison entre deux éléments se situant à la position communiquée au constructeur.

```
class ListViewItemComparer : IComparer
{
    private int _col;

    public ListViewItemComparer()
    {
        _col = 0;
    }
    public ListViewItemComparer(int column)
    {
        _col = column;
    }
    public int Compare(object x, object y)
    {
        return String.Compare(((ListViewItem)x).SubItems[_col].Text,
            ((ListViewItem)y).SubItems[_col].Text);
    }
}
```

Il nous reste à programmer la demande de tri sur événement clic de la colonne.

```
private void lvCultures_ColumnClick(object sender, ColumnClickEventArgs e)
{
   this.lvCultures.ListViewItemSorter = new ListViewItemComparer(e.Column);
   lvCultures.Sort();
}
```

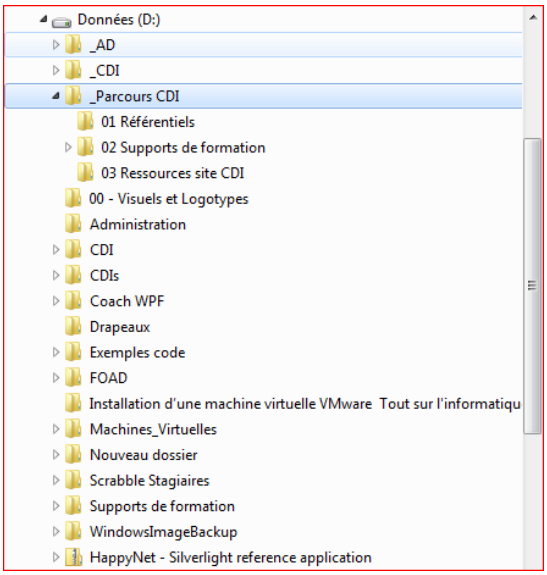
Cette approche est valide pour tout tri sur des collections ou des tableaux disposant d'une méthode Sort(). Le tri peut toutefois être plus complexe à mettre en place : ici, il s'agit de la comparaison de 2 chaînes que nous n'avons pas à reprogrammer…

4 Le contrôle TreeView

Comme son nom l'indique, le **TreeView** est à l'image d'un arbre. Il est constitué de ramifications complexes **TreeNode** et d'éléments terminaux (les feuilles) qui n'ont pas d'autres subdivisions.

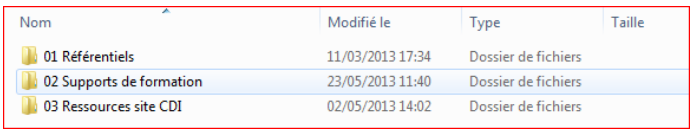
Entre les différents nœuds de l'arbre, des relations hiérarchiques de type parent / enfants existent. Un enfant n'appartient qu'à un seul parent mais les descendants, à la manière d'un arbre généalogique, peuvent être nombreux.

L'utilisation du **TreeView** peut nécessiter le recours à des processus récursifs du type : déterminer l'ensemble des éléments ayant un lien de filiation avec l'élément communiqué et ce, quel que soit le niveau de la filiation. Ce traitement se poursuivra donc jusqu'à ce que le dernier élément de l'arbre n'est plus d'enfant.



L'image de la fenêtre gauche de l'explorateur de documents de Windows permet de se faire une idée précise de son apparence et de son fonctionnement.

La partie droite est constituée d'un contrôle ListView :



Un exemple courant d'utilisation en informatique de gestion d'un TreeView est le traitement des nomenclatures d'articles (composant / composés) dont le niveau d'arborescence (profondeur) n'est pas défini.

Nous verrons qu'il faut prendre des options particulières pour synchroniser les éléments qui constituent l'arborescence et les informations stockées dans la base de données afin de conserver de bonnes performances à l'ensemble.

Comme pour le contrôle ListView, il est possible d'associer une image fournie par un composant ImageList à chaque élément de l'arbre.

Pour plus d'informations sur le contrôle TreeView, reportez –vous au chapitre 3 du livre du livre Windows Forms Application Development, page 100.

4.1 Organisation

La collection **Nodes** contient tous les objets **TreeNode** assignés à **TreeView**.

Les nœuds d'arbre dans cette collection sont référencés comme nœuds d'arbre racine.

Les nœuds d'arbre ajoutés par la suite à un nœud d'arbre racine sont référencés comme nœuds enfants.

Étant donné que chaque **TreeNode** peut contenir une collection d'autres objets **TreeNode**, il peut être difficile de déterminer votre position dans l'arborescence quand vous itérez sur la collection.

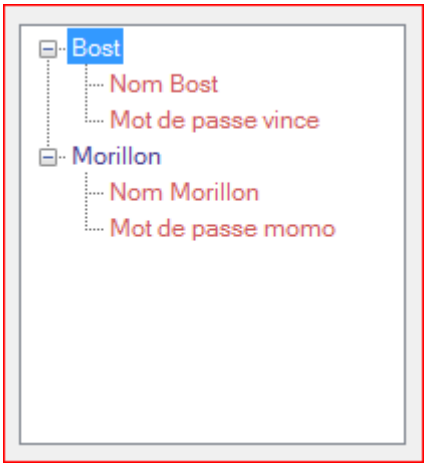
Un objet **TreeNode** est doté d'une étiquette définie par la propriété Text.

Vous pouvez, lors de la construction d'un nœud, utiliser un constructeur auquel vous passez la valeur de l'étiquette.
Vous devez ensuite l'ajouter à la collection **Nodes** racine ou à la collection **Nodes** d'un enfant.

4.2 Mise en œuvre

Cet exemple met en œuvre un contrôle arborescent présentant les utilisateurs du système et les propriétés de ces derniers.

Chaque utilisateur est défini comme un nœud parent, chacune de ses propriétés étant un nœud enfant.




```
foreach (Utilisateur utilisateur in utilisateurs)
{
    TreeNode oParent = new TreeNode();
    oParent.ForeColor = Color.DarkSlateBlue;
    oParent.Text = utilisateur.ID;
    tvUtilisateurs.Nodes.Add(oParent);
    TreeNode oEnfant = new TreeNode();
    oEnfant.ForeColor = Color.IndianRed;
    oEnfant.Text = String.Format("{0} {1}",
        "Nom", utilisateur.Nom);
    oParent.Nodes.Add(oEnfant);
    oEnfant = new TreeNode();
    oEnfant.ForeColor = Color.IndianRed;
    oEnfant.Text = String.Format("{0} {1}", "Mot de passe",
        utilisateur.MotDePasse);
    oParent.Nodes.Add(oEnfant);
}
```

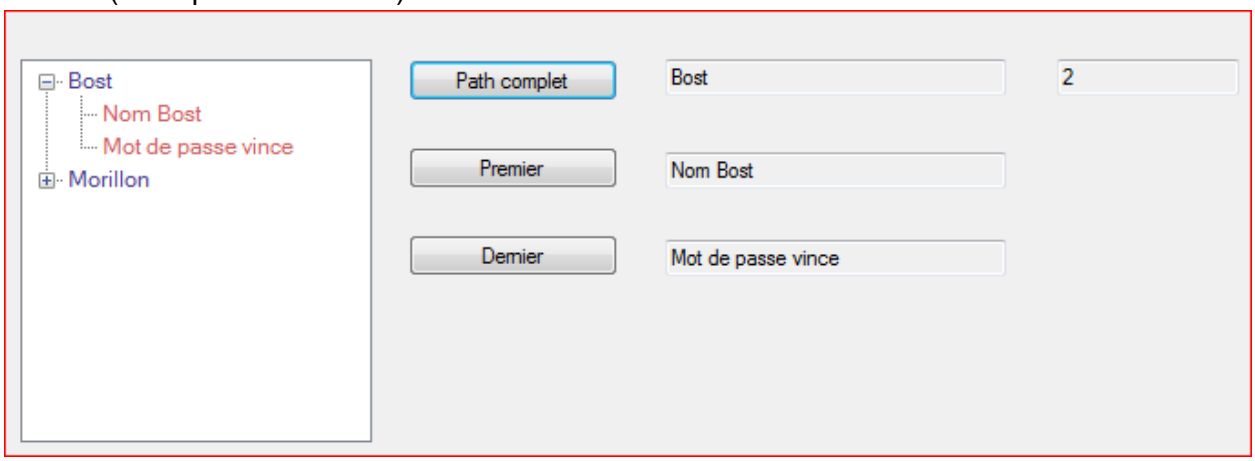
Quelques propriétés singulières.

Propriétés	
PrevNode	La référence du nœud **frère** qui précède le nœud sélectionné
NextNode	La référence du nœud **frère** qui suit le nœud sélectionné
PrevVisibleNode	La référence du nœud visible qui précède le nœud sélectionné
NextVisibleNode	La référence du nœud visible qui suit le nœud sélectionné
FirstNode	Le premier enfant
LastNode	Le dernier enfant
Parent	Le nœud parent
FullPath	Le chemin complet du nœud
IsSelected	Définir si le nœud est ou non sélectionné
IsExpanded	Le nœud est-il développé
IsEditing	Le nœud est-il modifiable
IsVisible	Le nœud est-il visible

Pour réduire les nœuds enfants vous pouvez cliquer sur le signe – ou invoquer la méthode **Collapse**. Pour développer les nœuds enfants, la méthode **Expand** ou **expandAll**. pour développer tous les nœuds.

La propriété **FullPath** vous permet d'obtenir l'adresse complète du nœud dans l'arborescence.
Elle est associé à la proriété pathSeparator qui permet de préciser le séparateur de nœud.
Cette approche n'est pas sans similitude avec les méthodes de manipulation de flux XML avec Xpath et xQuery.

Exemple sur sélection du Nœud Bost : Le nombre qui s'affiche indique le nombre de nœuds enfants (et de petits enfants…)



Le code associé :

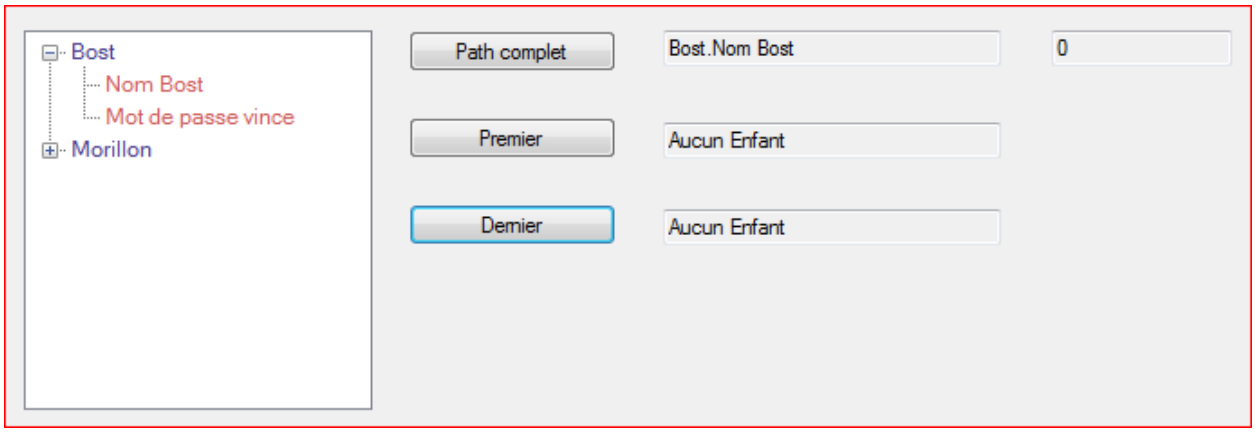
```
private void btnPath_Click(object sender, EventArgs e)
{
    tvUtilisateurs.PathSeparator = ".";
    if (tvUtilisateurs.SelectedNode != null)
    {
        txtPath.Text = tvUtilisateurs.SelectedNode.FullPath;
        txtNbre.Text = tvUtilisateurs.SelectedNode.GetNodeCount(true).ToString();
    }

}

private void btnFirst_Click(object sender, EventArgs e)
{
    if (tvUtilisateurs.SelectedNode != null)
        if (tvUtilisateurs.SelectedNode.FirstNode != null)
            txtPremier.Text = tvUtilisateurs.SelectedNode.FirstNode.Text;
        else txtPremier.Text = "Aucun Enfant";
}

private void btnLast_Click(object sender, EventArgs e)
{
    if (tvUtilisateurs.SelectedNode != null)
        if (tvUtilisateurs.SelectedNode.LastNode != null)
    txtDernier.Text = tvUtilisateurs.SelectedNode.LastNode.Text;
        else txtPremier.Text = "Aucun Enfant";
}
```

Le résultat est obtenu lorsque le nœud Nom est sélectionné :



Il est important de vérifier l'existence des objets avant d'accéder à leurs propriétés, notamment si vous travaillez sur les nœuds sélectionnés.