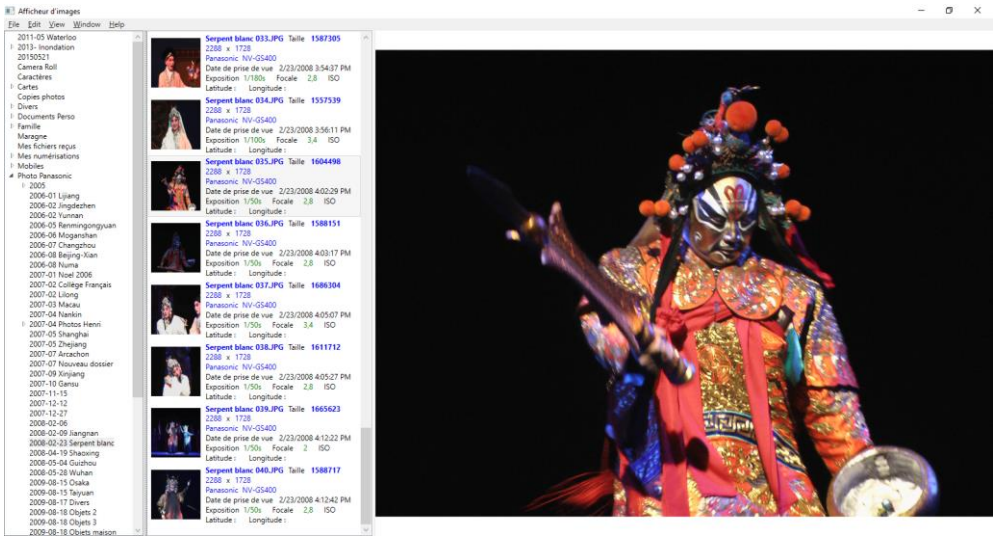
Afficheur d'images

Objectif

Il s'agit de créer une application permettant de parcourir le répertoire « Mes Images », d'afficher la liste des images qu'il contient, d'afficher en plus grand format une image sélectionnée et de créer un diaporama.



Pour pouvoir tester notre code, un jeu d'essai sera évidemment nécessaire, qui devra être situé sous ce répertoire et comporter au moins deux répertoires d'images. Si les images que l'on peut télécharger sur Internet sont évidemment utilisables, de « vraies » photos issues d'un appareil photo (ou même d'un téléphone) sont préférables car porteuses de beaucoup plus d'informations. Une partie de l'exercice exploitera le fait que ces photos sont au format *JPEG*. Si nécessaire, on trouvera sur le *moodle* du cours un jeu d'essai adapté, mais vous pouvez naturellement utiliser des photos personnelles.

Nous allons développer cette application en nous appuyant sur la technologie *Windows Presentation Foundation*, en utilisant un modèle de conception *Modèle-Vue/Vue-Modèle*, en utilisant largement la technique du *data binding*, et en écrivant le minimum de code possible.

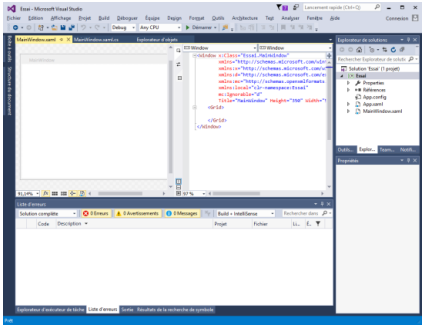
Assez différente de l'approche *Windows Forms* que nous avons suivi jusques ici, elle correspond mieux aux technologies actuelles utilisées pour les applications *Windows* modernes mais aussi dans le cadre du développement *Web* ou du développement pour appareil mobiles (*Windows Phone* ou *Android*).

Nous allons réaliser ce projet étapes par étapes. Ce document comporte comme d'habitude de nombreux extraits de code, très complet au début, de plus en plus parcellaires ensuite. Les recopier tels quels risque de poser plus de problèmes qu'ils ne sont susceptibles d'en résoudre !

Le texte comporte des *exercices*, essentiels et obligatoires, des *questions* sur lesquelles il est important de réfléchir (et qui peuvent nécessiter des recherches complémentaires), des *compléments* un peu plus difficiles et donc optionnels (les niveaux de difficultés relatifs sont symbolisés par les icônes ci-contre).

Construction du cadre

Même si nous n'avons pas immédiatement besoin de tous les éléments, nous allons tout de suite créer le cadre global de l'application.



Commençons par créer un projet *WPF* vide que l'on pourra appeler **`Afficheur`**.

Visual Studio s'ouvre comme d'habitude sur la fenêtre principale du programme. Mais cette fois-ci, elle est coupée en deux, avec d'un côté une représentation et de l'autre sa *description* sous la forme d'un fichier *XML*, plus précisément *XAML*. On pourra choisir entre un découpage horizontal (par défaut) ou vertical de cette fenêtre (comme sur l'illustration).

Le fichier créé par déclare un objet principal de type **`Window`** muni de divers attributs et contenant un seul et unique objet graphique, de type **`Grid`** :

```
<Window x:Class="Essai.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Essai"
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="525">
    <Grid>

    </Grid>
</Window>
```

Exercice 1. On peut d'ores et déjà changer le titre de la fenêtre pour : **`Afficheur d'images`**.

Remarque : pour commenter du code dans les pages XAML on devra utiliser : **`<!--…-->`**.

Découpage de la fenêtre

On peut bien sur modifier le contenu de la fenêtre grâce à l'interface graphique, mais il est souvent aussi simple de le faire directement *à la main* (le mode graphique réservant souvent des surprises).

Nous allons découper la fenêtre en plusieurs parties que nous remplirons progressivement. Pour cela nous définissons deux lignes (**`Row`**) et dans la deuxième ligne (**`Grid.Row="1"`**) nous définissons 5 colonnes (**`Column`**). Dans la première et la troisième colonne nous plaçons deux **`GridSplitter`**.

```
<Window x:Class="ImageViewer.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:ImageViewer"
        mc:Ignorable="d"
        Title="Afficheur d'images" Height="350" Width="525">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="20"></RowDefinition>
            <RowDefinition Height="*"></RowDefinition>
        </Grid.RowDefinitions>
        <Grid Grid.Row="1">
            <!-- #region Structure -->
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="150"></ColumnDefinition>
                <ColumnDefinition Width="5"></ColumnDefinition>
                <ColumnDefinition Width="100"></ColumnDefinition>
                <ColumnDefinition Width="5"></ColumnDefinition>
                <ColumnDefinition Width="*"></ColumnDefinition>
            </Grid.ColumnDefinitions>
            <!-- #endregion -->
            <GridSplitter Grid.Column="1" ResizeBehavior="PreviousAndNext" Width="3" />
            <GridSplitter Grid.Column="3" ResizeBehavior="PreviousAndNext" Width="3" />
        </Grid>
    </Grid>
</Window>
```

Exercice 2. Réalisez le découpage de la fenêtre. Compilez votre code et exécutez-le pour constater le fonctionnement des deux **`GridSplitter`**.

Question 1. Pourquoi les dimensions de certaines hauteurs ou largeurs ont-elles pour valeur **`*`** ?

Liste déroulante

Nous allons d'abord créer la liste déroulante qui apparaît au centre de la page, colonne deux, en rajoutant à notre page un objet de type **`ListBox`** (les autres colonnes seront remplies progressivement).

```
<ListBox Grid.Column="2" x:Name="listBox"
      HorizontalAlignment="Stretch"
      VerticalAlignment="Stretch"></ListBox>
```

On notera sur le code présenté ci-contre l'attribut **`Grid.Column`** qui précise le positionnement de la liste.

Exercice 3. Rajouter la boîte liste.

Question 2. Que signifie **`Stretch`** ?

Les données

Fournisseur de données

Afin de conserver un code bien organisé, nous allons créer une classe (*Projet|Ajouter une classe*) que nous appellerons **`DataModel`** et qui aura en charge toutes les actions nécessaires à la gestion des données.

Nous allons d'abord rajouter une seule propriété, qui contiendra les fichiers contenu dans le répertoire qui nous intéresse. Il sera initialisé par le constructeur qui récupère tous les fichiers de type **`jpg`** contenus dans un répertoire dont le nom est passé en paramètre (on notera la jolie cascade de *lambda-expressions*).

```
class DataModel
{
    public List<FileInfo> Fichiers { get; set; }
    public DataModel(String Chemin)
    {
        Fichiers = new List<FileInfo>();
        string[] fichiers = Directory.GetFiles(Chemin);
        fichiers.ToList().Where(c => c.ToLower().EndsWith("jpg"))
          .ToList().ForEach(c => Fichiers.Add(new FileInfo(c)));
    }
}
```

Exercice 4. Créez la classe **DataModel**.

Question 3. La ligne **fichiers.ToList(…)** est un peu imprudente. Comment peut-on la corriger ? Cherchez comment un simple point d'interrogation pourrait l'améliorer.

Question 4. Que signifie Modèle-Vue/Vue-Modèle ?

Complément 1. Rajouter une méthode *d'extension* permettant de se dispenser de l'appel à la méthode **ToList**().

Liaison de données

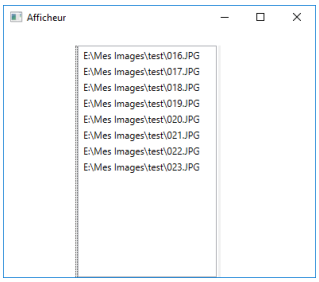
Pour réaliser la liaison entre notre **`ListBox`** et notre modèle de données, nous allons simplement devoir lui préciser la source de ses données en lui rajoutant l'attribut qui précise qu'elle est *liée* à l'objet **`Fichiers`** :

```
ItemsSource="{Binding Fichiers}"
```

Il suffit maintenant de créer une nouvelle instance du modèle dans le constructeur de notre fenêtre pour afficher dans la boîte liste la liste des fichiers chargés en réalisant la *liaison* (le *binding*) entre la boîte liste et la liste de fichiers.

```
public MainWindow()
{
    InitializeComponent();
    string path = Environment.GetFolderPath(Environment.SpecialFolder.MyPictures);
    DataContext = new DataModel(path);
}
```

Exercice 5. Réalisez l'affichage des fichiers.



Question 5. Que désigne **`DataContext`** ? Quel est son type ?

Question 6. Pourquoi l'application affiche-t-elle simplement la liste des noms de fichiers contenus dans le répertoire (illustration ci-contre) ?

Mise en forme de l'affichage

La mise en forme de l'affichage va se faire en décrivant, toujours en *XAML*, un format adapté sous la forme d'un **`DataTemplate`** (ici contenu dans un **`ItemTemplate`**) qui indiquera comment doit être affiché chacun des éléments de la source (**`ItemSource`**).

Le **`Template`** ci-dessous va permettre d'afficher pour chaque fichier image une vignette la représentant et, à sa droite, des informations comme le nom du fichier et sa taille (illustration de droite).


```
<ListBox.ItemTemplate>
    <DataTemplate>
        <StackPanel Orientation="Horizontal">
            <Image Width="80" Height="60"
                     HorizontalAlignment="Center">
                <Image.Source>
                    <BitmapImage DecodePixelWidth="80"
                          DecodePixelHeight="60"
                          UriSource="{Binding Path=FullName}" />
                </Image.Source>
            </Image>
            <StackPanel Orientation="Vertical">
                <StackPanel Orientation="Horizontal">
                    <TextBlock Text="{Binding Path=Name}" />
                    <TextBlock Text="Taille" />
                    <TextBlock Text="{Binding Path=Length}" />
                </StackPanel>
             </StackPanel>
        </StackPanel>
    </DataTemplate>
</ListBox.ItemTemplate>
```

On notera la structure arborescente de ce **`Template`**, organisé à l'aide d'objets de type **`StackPanel`** alternativement horizontaux et verticaux, qui construisent une structure imbriquée sur trois niveaux où vont être placés l'image et les deux zones de texte (illustration ci-contre).

On notera aussi la façon dont les diverses zones (image ou texte) sont liées aux données grâce à la notion de **`Binding`** qui précise quel attribut de l'élément de la liste doit-être représenté par un élément graphique donné.

Exercice 6. Réalisez la mise en forme qui vient d'être présentée. Adaptez le code pour que les données ne soient pas collées les unes aux autres.

Exercice 7. Récupérez sur *moodle* la bibliothèque **`ExifLibrary`**, ajoutez une référence, remplacez partout où c'est nécessaire **`FileInfo`** par **`JPGFileInfo`** et complétez l'affichage pour obtenir quelque chose comme ce qui est représenté sur l'illustration ci-contre (ou mieux ☺) en utilisant les propriétés adaptées de cette classe (et des photos contenant suffisamment d'informations !).[1]

Question 7. Comment n'afficher les libellés que si les données sont disponibles (par exemple, beaucoup de photos n'ont pas de données GPS et il est donc inutile d'afficher la ligne) ?

Affichage de l'image en plus grand format

Pour afficher l'image dans un format plus adapté (partie droite de l'illustration), nous allons rajouter dans la colonne de droite une zone de type **`Image`**, à laquelle on permettra de remplir tout l'espace disponible.

```
<Image Grid.Column="4" x:Name="image" HorizontalAlignment="Stretch"
           VerticalAlignment="Stretch"
    Source="{Binding ElementName=listBox, Path=SelectedItem.FullName}"/>
```

Exercice 8. Ajoutez ce contrôle image.

Question 8. Expliquez pourquoi le clic sur un élément de la liste suffit à afficher l'image sans que l'on ait eu à écrire la moindre ligne de code et en particulier aucune méthode de traitement de l'évènement.

Création d'un diaporama

Dans cette section nous allons permettre à l'application de lancer un diaporama, c'est-à-dire d'afficher successivement toutes les photos du répertoire et ce à intervalle régulier.

Ajout d'un menu

Pour que l'utilisateur puisse lancer notre diaporama, nous allons avoir besoin d'un élément de menu et d'une méthode de réponse à l'événement associé. La création d'un menu se fait par :

```
<Menu Grid.Row="0" IsMainMenu="True">
    <MenuItem Header="_Diaporama" Click="Diaporama_Click" />
    <MenuItem Header="_Edit" Click="EditItem_Click" />
</Menu>
```

Exercice 9. Créez le menu et la méthode de réponse.

Question 9. A quoi sert le caractère "_" au début des libellés de menu ?

Diaporama

Nous allons maintenant créer une nouvelle fenêtre, par l'option de menu *Projet|Ajouter une fenêtre*. C'est cette fenêtre qui nous permettra d'afficher en plein écran le diaporama.

Exercice 10. Créez une fenêtre sans bordure occupant toute la surface de l'écran et contenant un simple contrôle image. A la création, la fenêtre recevra en paramètre la liste de fichiers qu'elle devra afficher.

Exercice 11. Modifiez la fenêtre principale pour que le menu Diaporama ouvre une fenêtre diaporama.

Exercice 12. Rajoutez un **`Timer`** pour que le diaporama s'exécute automatiquement.

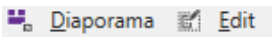
Exercice 13. Arrêtez le diaporama en appuyant sur la touche *Esc*

Question 10. A quoi set l'attribut ToolTip ?

Complément 2. Permettre de mettre le diaporama en pause et de le relancer, d'avancer et de reculer.

Complément 3. Ajouter des *Tooltips* à chaque élément de menu

Complément 4. Ajoutez des icônes aux éléments de menu :



Amélioration du parcours de fichiers

Création d'un contrôle utilisateur

[1] WPF possède bien une classe *BitmapMetadata* permettant d'obtenir le même type d'informations mais elle est d'un usage beaucoup plus délicat.

Pour pouvoir remplir la partie gauche qui est présentée sur notre première illustration, nous allons devoir créer une structure permettant de parcourir les répertoires.

Pour cela, nous aurons besoin d'une classe accessoire destinée à représenter un répertoire et ses sous-répertoires. Très simple, elle est décrite sur le code ci-contre.

```
public class DirectoryItem
{
    public string Name { get; set; }
    public string Path { get; set; }
    public List<DirectoryItem> Items { get; set; }

    public DirectoryItem()
    {
        Items = new List<DirectoryItem>();
    }
}
```

En rajoutant une méthode à notre modèle de données, il nous servira de fournisseur pour obtenir la liste des répertoires :

```
public List<Item> GetItems(string path)
{
    var items = new List<Item>();
    var dirInfo = new DirectoryInfo(path);
    foreach (var directory in dirInfo.GetDirectories())
    {
        var item = new DirectoryItem
        {
            Name = directory.Name,
            Path = directory.FullName,
            Items = GetItems(directory.FullName)
        };
        items.Add(item);
    }
    return items;
}
```

Nous pourrions naturellement rajouter dans notre page principale tous les composants nécessaires au parcours des répertoires, mais il nous paraît préférable d'écrire un code qui pourra être facilement réutilisé dans une autre application.

Nous allons donc créer un contrôle utilisateur que nous appellerons **`BrowserControl`**, grâce à l'option de menu *Projet|Ajouter un contrôle utilisateur*.

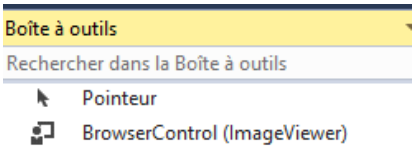
Dans ce contrôle, on insère un **`TreeView`** et on décrit son organisation dans la section **`Resources`** en s'appuyant sur un **`HierarchicalDataTemplate`** qui spécifie la façon dont se feront les liaisons en fonction du type des données liées. Ces types seront décrits un peu plus loin.

```
<UserControl x:Class="ImageViewer.BrowserControl"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:Model="clr-namespace:ImageViewer"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
    <UserControl.Resources>
        <HierarchicalDataTemplate DataType="{x:Type Model:DirectoryItem}"
                                  ItemsSource="{Binding Items}">
            <TextBlock Text="{Binding Path=Name}" ToolTip="{Binding Path=Path}" />
        </HierarchicalDataTemplate>
    </UserControl.Resources>
    <Grid>
        <TreeView ItemsSource="{Binding}" />
    </Grid>
</UserControl>
```

Il suffit maintenant de compléter le constructeur pour faire l'acquisition des données nécessaires à remplir l'arborescence :

```
string path = Environment.GetFolderPath(Environment.SpecialFolder.MyPictures);
DataContext = new DataModel().GetItems(path);
```

Une fois compilé, ce code fournit un contrôle qui apparaît dans la boîte à outils et que l'on peut insérer dans la colonne de gauche de la fenêtre principale par simple glisser-déplacer.



Exercice 14. Réalisez le contrôle permettant de parcourir les répertoires.

Question 11. Comment faire en sorte qu'il occupe tout l'espace dans la colonne de gauche ?

Question 12. Le contrôle que nous avons construit est parfaitement opérationnel mais n'a pour le moment aucun effet. Pourquoi ?

Création d'un évènement

Nous allons maintenant rajouter un événement à notre classe **`BrowserControl`**.

```
public event RoutedEventHandler ItemChanged;
```

Puis, dans notre réponse à l'événement **`SelectedItemChanged`** du **`TreeView`**, nous allons « lever l'évènement » :

```
private void TreeView_SelectedItemChanged(object sender,
                        RoutedPropertyChangedEventArgs<object> e)
{
    if (ItemChanged != null)
        ItemChanged(sender, e);
}
```

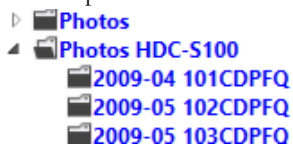
Question 13. Expliquez le code de traitement de l'évènement **`SelectedItemChanged`**.

Exercice 15. Rajoutez l'évènement à la classe et le traitement de l'évènement sur le **`TreeView`**.

Après avoir compilé le contrôle, on constate que dans la fenêtre de ses propriétés est apparu un nouvel événement.

Exercice 16. Dans la fenêtre principale, traitez l'évènement pour que le changement de répertoire modifie la colonne présentant la liste des images contenues dans ce répertoire.

Question 14. Quelles modifications apparaissent dans le fonctionnement du programme ?



Complément 5. Modifier l'affichage de l'arbre pour rajouter des icônes de dossier (illustration ci-contre).

Pour être réutilisable, il faudrait que le contrôle utilisateur soit créé dans un autre projet de la solution. Il faut pour cela ajouter à la solution un projet de nom **`Controls`** de type *Bibliothèque de Classe* et créer le contrôle dans ce projet.

Pour l'insérer dans le projet initial il faudra alors rajouter une référence à la bibliothèque de classe.

Complément 6. Transférer le contrôle dans une bibliothèque de classe.

Triggers

WPF permet aussi de gérer de façon simple, et toujours par liaison de données, des modifications de l'environnement. C'est la notion de **`Trigger`** (*déclencheur*) qui va le permettre.

Le code suivant permet de ne pas afficher une zone de texte (**`TextBlock`**) si un champ de données n'est pas renseigné (valeur **`x:Null`**) :

```
<TextBlock Text="Exposition  " Margin="8,0,0,0">
    <TextBlock.Style>
        <Style TargetType="{x:Type TextBlock}">
            <Style.Triggers>
                <DataTrigger Binding="{Binding Path=Exposition}" Value="{x:Null}">
                    <Setter Property="Visibility" Value="Collapsed"/>
                    <Setter Property="Foreground" Value="Black"/>
                </DataTrigger>
            </Style.Triggers>
        </Style>
    </TextBlock.Style>
</TextBlock>
```

Exercice 17. Améliorez l'interface pour qu'elle n'affiche les données JPEG que si elles sont effectivement renseignées.

L'extrait suivant va afficher ou faire disparaître un **`StackPanel`** complet selon qu'un élément de menu de nom **`check`** sera validé ou non.

Pour cela, il traite de façon strictement déclarative la modification de l'état de l'élément de menu. Il permet aussi de fixer la taille d'affichage d'une image selon qu'elle est accompagnée de détails ou non :

```
<StackPanel.Style>
    <Style>
        <Style.Triggers>
            <DataTrigger Binding="{Binding ElementName=check, Path=IsChecked}" Value="False">
                <Setter Property="StackPanel.Width" Value="130"></Setter>
                <Setter Property="StackPanel.Height" Value="90"></Setter>
            </DataTrigger>
            <DataTrigger Binding="{Binding ElementName=check, Path=IsChecked}"  Value="True">
                <Setter Property="StackPanel.Width" Value="400"></Setter>
                <Setter Property="StackPanel.Height" Value="100"></Setter>
            </DataTrigger>
        </Style.Triggers>
    </Style>
</StackPanel.Style>
```

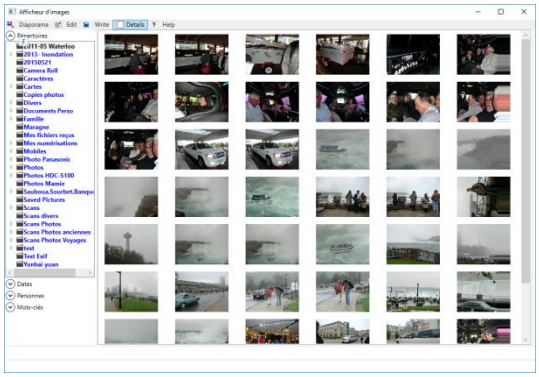
Exercice 18. Permettre à l'utilisateur de choisir s'il souhaite afficher les détails sur une image.

Autres composants d'interface

D'autres contrôles prédéfinis vont nous permettre d'améliorer la présentation de l'interface.

WrapPanel

Le code suivant permet d'afficher les images en grille, comme le montre l'illustration ci-contre. Elle indique comment doit être organisé le panneau d'affichage des images :



```
<ListBox.ItemsPanel>
    <ItemsPanelTemplate>
        <WrapPanel
             Orientation="Horizontal"
             IsItemsHost="True">
        </WrapPanel>
    </ItemsPanelTemplate>
</ListBox.ItemsPanel>
```

Complément 7. Réalisez la mise en forme représentée sur l'illustration.

Expander

Sur la même illustration, on peut voir que le contrôle de navigation a été inséré dans un nouveau contrôle qui permet de le réduire pour faire apparaître d'autres choix. C'est le contrôle **`Expander`** qui permet cela :

```
<Expander x:Name="mots" Grid.Row="3" IsExpanded="False"
          Header="Mots-clés">
    <ListBox VerticalAlignment="Stretch" ItemsSource="{Binding Keywords}">
        <ListBox.ItemTemplate>
            <DataTemplate>
                <TextBlock Text="{Binding Path=Name}" FontWeight="Bold"
                  Foreground="Blue" Margin="8,0,0,0" />
            </DataTemplate>
        </ListBox.ItemTemplate>
    </ListBox>
</Expander>
```

Complément 8. Complétez la mise en forme représentée sur l'illustration.

Synthèse

Cet exercice nous a montré comment écrire du code dans lequel le traitement des données est bien séparé de leur mise en forme et comment la liaison de données permet de réaliser une application relativement complexe en écrivant très peu de code.