TP .Net n° 3 : Application WPF

Objectif : Création d'un « PhotoViewer » avec classe C# et effets sous WPF

Partie 1 : Mise en place des fonctionnalités de l'application

Démarrez Visual Studio, et créez un nouveau projet de type « C# / **Application WPF (.Net Framework)** » en nommant votre répertoire de solution en local.

Il y a 2 fichiers relatifs à votre fenêtre principale. Quels sont-ils et quels types de code contiennent-ils ? :

_____

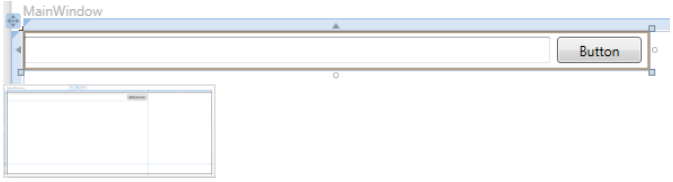
_____

A l'aide du code suivant, définissez 2 lignes et 2 colonnes dans votre « Grid » :

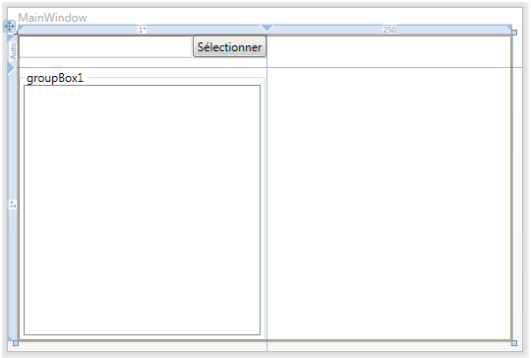
```
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="250" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
    <RowDefinition Height="Auto" />
    <RowDefinition Height="*" />
</Grid.RowDefinitions>
```

Dans votre première case (ligne 0, colonne 0) ajoutez un DockPannel contenant un bouton et un TextBox.
Forcez la position de votre DockPanel en lui fixant les propriétés : `Grid.Row="0" Grid.Column="0"`
Nommez votre bouton « Sélectionner », et positionnez le à droite de votre DockPanel : `DockPanel.Dock="Right"`
Positionnez votre TextBox à gauche du DockPannel : `DockPanel.Dock="Left"`
(Supprimez les tailles et les alignements autres du DockPannel et du TextBox)
Vous devez obtenir ceci :

```
<DockPanel Name="dockPanel1" ... >
  <Button Content="Sélectionner" ... />
  <TextBox DockPanel.Dock="Left" Name="…" Margin="1" />
</DockPanel>
```

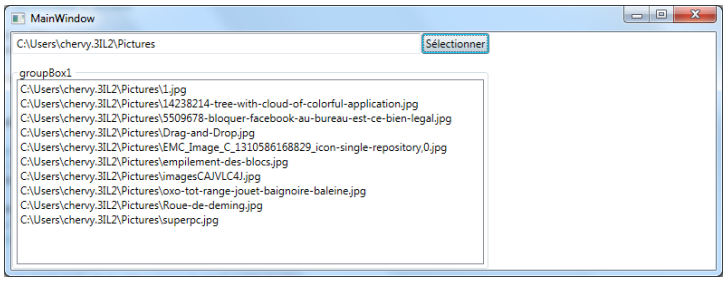


Dans la partie en dessous (ligne 1, colonne 0 de la Grid), ajoutez un contrôle GroupeBox qui contient un contrôle ScrollViewer qui contient lui-même un contrôle ListBox :
Mettez la propriété « `VerticalScrollBarVisibility` » du contrôle ScrollViewer à « `"Auto"` ».
Vérifiez que les éléments de la fenêtre se redimensionnent bien avec la fenêtre
Changez le nom du GroupBox en « Sélection »



Le bouton « Sélection » doit permettre à l'utilisateur de sélectionner un répertoire contenant des images JPEG qui seront chargées dans la « listbox ». Ajoutez la méthode correspondant au clic du bouton en double cliquant dessus. Dans cette méthode, il faut :
- créer et instancier un « `FolderBrowserDialog` » en ajoutant la référence « SystemWindows.Forms » (Menu Projet/Ajouter une référence) et le « namespace » associé « `using System.Windows.Forms;` »
- appeler sa méthode « `ShowDialog` » en récupérant la valeur renvoyée de type « `DialogResult` », et tester cette valeur qui doit renvoyer la chaîne « OK »
- créer et instancier un objet de type « `DirectoryInfo` » qui pointe sur le répertoire sélectionné (propriété « `SelectedPath` »), et créer une boucle « foreach » qui instancie un objet fichier de type « `FileInfo` » renvoyé par la méthode « `GetFiles` » de l'objet « `DirectoryInfo` » en ne sélectionnant que les fichiers d'extension « jpg »
- ajouter à chaque itération un « Item » à la « listbox » (méthode « `Add` ») contenant le chemin du fichier (méthode « `FullName` »).

Le « TextBox » permet juste d'afficher le répertoire sélectionné.
Testez la partie sélection du répertoire de votre application pour obtenir le résultat suivant :



Les fichiers affichés par la « Listbox » le sont sous forme de chemins et non sous forme d'images. Il faut donc modifier la « listbox » pour qu'elle travaille sur les images et non sur les chemins.
Pour cela nous allons travailler sur la partie XAML et en particulier sur les styles des contrôles. Il faut dans un premier temps ouvrir une zone de ressources dans le code XAML dans laquelle seront définis les styles des éléments à afficher. Avant la partie des balises « Grid » définissez une balise « `<Window.Resources>` » et sa jumelle « `</Window.Resources>` ». Entre ces balises ajoutez le code suivant :

```
<Style TargetType="{x:Type ListBoxItem}">
   <Setter Property="Background" Value="Transparent" />
   <Setter Property="MaxHeight" Value="75" />
   <Setter Property="Margin" Value="5" />
   <Setter Property="Template">
      <Setter.Value>
          <ControlTemplate TargetType="{x:Type ListBoxItem}" >
                 <Border SnapsToDevicePixels="True" HorizontalAlignment="Stretch"
                        VerticalAlignment="Stretch" Background="{TemplateBinding Background}">
                        <Image  Source="{Binding Mode=OneWay}" Opacity="1" Cursor="Hand" x:Name="image"/>
                 </Border>
          </ControlTemplate>
      </Setter.Value>
   </Setter>
</Style>
```

A quel type d'élément s'adresse ce style : _____

Que se passe-t-il si vous commentez la ligne « `<Setter Property="MaxHeight" Value="75" />` » ? :

_____

En dessous de « `Border` » dans «`ControlTemplate` » ajoutez le code suivant :

```
<ControlTemplate.Triggers>
    <Trigger Property="IsSelected" Value="True">
        <Setter Property="Background" Value="#445B6249"   />
    </Trigger>
</ControlTemplate.Triggers>
```

Quel effet a été ajouté ? : _____

Ajouter le style suivant dans la balise « `<Window.Resources>` » :

```
<Style TargetType="{x:Type ListBox}" >
    <Setter Property="Foreground" Value="White" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type ListBox}" >
                <WrapPanel Margin="5" IsItemsHost="True" Orientation="Horizontal"
                           VerticalAlignment="Top" HorizontalAlignment="Stretch" />
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

Quel va être l'effet sur la ListBox ? : _____

Nommez votre style « `PhotoListBoxStyle` » :

```
<Style TargetType="{x:Type ListBox}" x:Key="PhotoListBoxStyle" >
```

L'effet s'applique t-il toujours sur la ListBox ? et Pourquoi ? :

Que faut t-il préciser dans la balise Listbox pour que ce style soit de nouveau appliqué ? :

Dans les propriétes de la Listbox ajoutez :

```
Margin="5" IsSynchronizedWithCurrentItem="True" SelectionMode="Extended" ItemsSource="{Binding}"
```

Modifiez ensuite le style « ListBoxItem » comme cela :

```
<Border Padding="4" Background="White" BorderBrush="#22000000" BorderThickness="1">
   <StackPanel Orientation="Vertical" Width="200" Height="200">
       <Border SnapsToDevicePixels="True" HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
                                          Background="{TemplateBinding Background}">
            <Image Width="150" Height="150" Source="{Binding Mode=OneWay}"
                                          Opacity="1" Cursor="Hand" x:Name="image"/>
       </Border>
   </StackPanel>
</Border>
```

Et modifiez : `<Setter Property="MaxHeight" Value="150" />`

A quel élément s'applique le Trigger ajoutant un fond gris à l'Item sélectionné ? :

Vous devez obtenir une Listbox d'images entièrement configurée et utilisable :

Ajoutez dans la partie droite (colonne 1 et ligne 1) un GroupeBox contenant un ScrollViewer contenant lui-même un StackPanel. Supprimez toute référence de taille (hauteur et largeur) de façon à ce que l'ensemble occupe la taille de la case de la grille. Mettez la propriété « `VerticalScrollBarVisibility` » du ScrollViewer à auto. Votre GroupBox doit afficher « Propriétés ».

Ajoutez le style suivant à votre fenêtre :

```
<Style x:Key="LabelHeader" TargetType="{x:Type Label}">
    <Setter Property="Background">
```

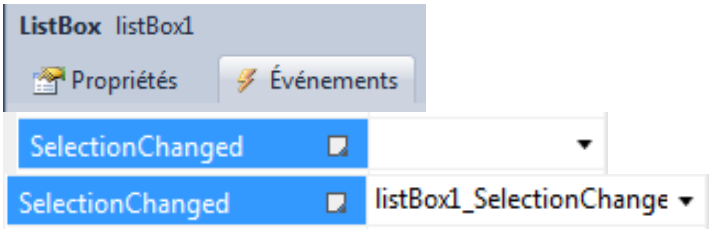
```
            <Setter.Value>
                <LinearGradientBrush StartPoint="0,0.5" EndPoint="1,0.5" >
                    <LinearGradientBrush.GradientStops>
                        <GradientStop Offset="0.5" Color="{x:Static SystemColors.AppWorkspaceColor}" />
                        <GradientStop Offset="2" Color="Transparent" />
                    </LinearGradientBrush.GradientStops>
                </LinearGradientBrush>
            </Setter.Value>
        </Setter>
        <Setter Property="Foreground" Value="White" />
        <Setter Property="FontWeight" Value="Bold" />
    </Style>
```

Ajoutez un contrôle Label affichant le texte « Source » dans le StackPanel. Que devez-vous ajouter pour qu'il applique le style précédent ? :

_____

Ajoutez un 2° Label dans le StackPanel de la partie droite, sans texte mais avec un nom identifiable (lb1 par ex) et un style « LabelPhoto » que vous allez définir. Ce style doit fixer la couleur de fond (`GhostWhite`), la couleur du texte (`DarkRed`), la hauteur du Label, et imposé le slyle de police en gras.

Sélectionnez la ListBox de gauche et dans sa fenêtre « Propriétes » cliquez sur « Evénements ». Recherchez l'évenement « SelectionChanged » et faites un double clic sur la partie droite.



Qu'est qui a été ajouté à la balise de la ListBox ? :

_____

Qu'est qui a été ajouté au code behind C# ? :

_____

Que devez-vous rajouter dans cette méthode pour que le chemin de l'image soit ajouté au 2° Label de droite ? :

_____

Vous devez obtenir ceci :

Partie 2 : Utilisation de classes C#

Faites « Ajouter un nouvel élément » et sélectionnez une classe C# que vous nommerez « Photo.cs ».
Ajoutez à votre classe, deux variables privées (« `private` ») de type String et Uri nommée respectivement _path et _source. Ajoutez le constructeur de votre classe :

```
public Photo(string path)
{
}
```

Dans votre constructeur initialisez les variables privées _path et _source.

```
_path = path;
_source = null;
if (_path != "")
{ _source = new Uri(path); }
```

Ajoutez le méthode « `ToString()` » suivante :

```
public override string ToString()
{
    return _source.ToString();
}
```

Votre classe hérite de la classe Object, et il vaut mieux surcharger (`override`) la méthode « `ToString()` » de la classe « object » pour disposer d'une méthode plus adaptée à l'utilisation de cette classe.
Enfin ajoutez une propriété Source qui définit le chemin de la photo :

```
public string Source { get { return _path; } }
```

Dans le code de votre fenêtre, initialisez une variable de type « Photo ».

```
Photo ph;
```

Dans la methode correspondant à la sélection d'une photo de la liste, instanciez-y votre objet Photo

```
ph = new Photo(listBox1.SelectedItem.ToString());
```

Dans cette même méthode, afficher le Label en dessous de Source avec la propriété de l'object de la classe « Photo » :

lb1.Content = _____

Nous allons maintenant compléter les informations affichées sur la photo sélectionnée. Chaque photo prise avec un appareil photographique numérique comporte des données disponibles insérées dans le fichier lui-même (JPEG, TIFF …) comme par exemple les données au format Exif.

Ajoutez dans votre classe Photo :
- le namespace : `using System.Windows.Media.Imaging;`
- une classe interne <u>publique</u> « `PhotoMetadata` »

Ajoutez à cette nouvelle classe :
- une variable « `_metadata` » de type « `BitmapMetadata` »
- un constructeur ayant comme argument « `Uri imageUri` » dont le code est :

```
BitmapFrame frame = BitmapFrame.Create(imageUri, BitmapCreateOptions.DelayCreation,
                                                                    BitmapCacheOption.None);
_metadata = (BitmapMetadata)frame.Metadata;
```

Dans la classe Photo :
- creez une variable « `PhotoMetadata` » nommée « `_phmetadata` »
- dans le constructeur, intanciez cette variable en appelant le constructeur de sa classe

```
_phmetadata = new PhotoMetadata(_source);
```

- ajoutez une méthode publique « `Metadata` » retournant cette variable

```
public PhotoMetadata Metadata { get { return _phmetadata; } }
```

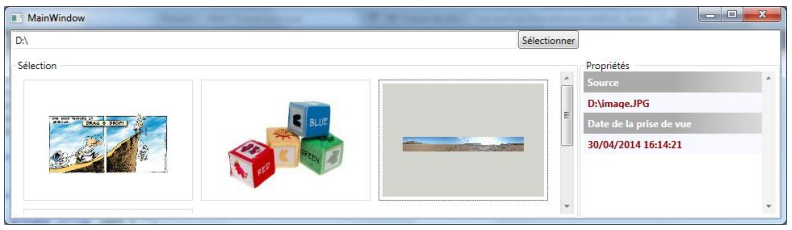
Pour rendre accessible toute propriété du metadata, il suffit maintenant de créer une propriété publique accessible de la classe « PhotoMetadata » correspondant à la propriété recherchée. Par exemple, pour publier la date de la prise de vue, il suffit d'ajouter la propriété suivante :

```
public DateTime? DateTaken
{
    get
    {
        Object val = _metadata.DateTaken;
        if (val != null)
        {
            return Convert.ToDateTime(_metadata.DateTaken.ToString());
        }
        else
        {
            return null;
        }
    }
}
```

Dans ce cas la valeur associée sera lisible dans un object « ph » de la classe « Photo » par :

```
ph.Metadata.DateTaken.ToString();
```

Vous pouvez ainsi ajouter la date de la prise de vue à votre fenêtre dans la partie « Propriétés » de la photo actuellement sélectionnée en utilisant la propriété « DateTaken » du « Metadata » de l'objet photo.



De la même façon ajoutez les propriétés :
- Titre qui retourne le titre de photo
- Camera qui retourne la description de l'appareil photos par les valeurs concaténées de « CameraManufacturer » et « CameraModel »
- Application qui retourne le nom de l'application ayant généré le fichier

Pour retourner une valeur de type « string » utilisez :

```
public string Propriete
```

et

```
return (val != null ? (string)val : String.Empty);
```

(sinon vous aurez une erreur, une chaîne ne peut pas être nullable)

Il existe d'autres valeurs qui sont incluses dans les fichiers photos, généralement au format Exif (Exchangeable image file format) qui est une spécification de format d'informations photographiques pour les images utilisées par les appareils photographiques numériques. Les balises de métadonnées définies dans le format Exif standard couvrent un large éventail de données précisant les propriétés de la prise de vue, de la photo, des droits et de son auteur. Vous allez en extraire quelques unes.

Ajoutez à la classe « PhotoMetadata » les 3 méthodes privées « QueryMetadata », « ParseUnsignedRational » et « ParseUnsignedRational » (disponibles sur Moodle).

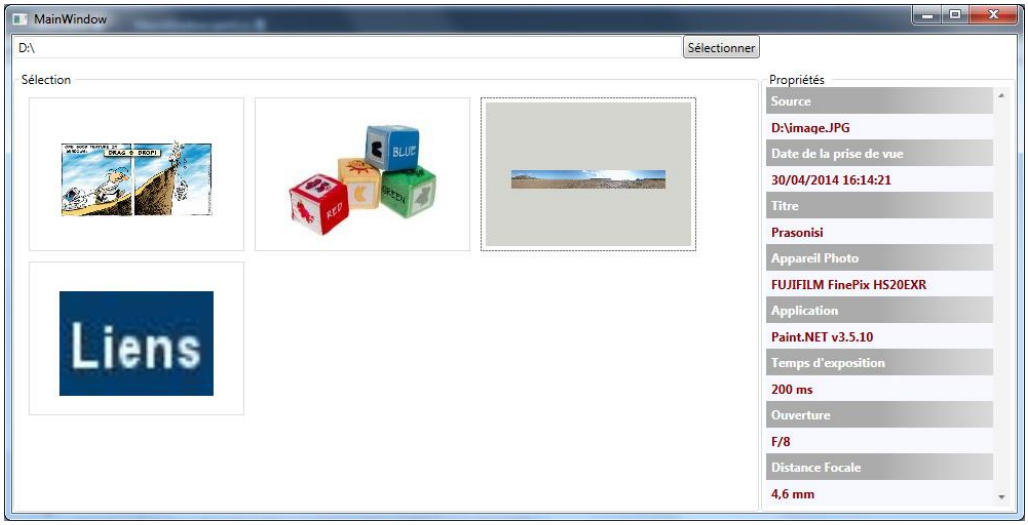
Ajoutez ensuite la propriété « IsoSpeed » qui retourne le temps d'exposition (en ms). Sa valeur est déterminée par :

```
QueryMetadata("/app1/ifd/exif/subifd:{uint=34855}");
```

Ajoutez ensuite la propriété de la valeur d'ouverture au format decimal : `"/app1/ifd/exif/subifd:{uint=33437}"` qu'il faut ensuite traiter par :

```
ParseUnsignedRational((ulong)val);
```

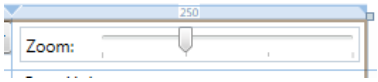
Finissez par la distance focale (en mm) : `"/app1/ifd/exif/subifd:{uint= 37386}"`



Partie 3 : Ajout d'effets WPF

WPF permet de réaliser directement des actions sur des éléments de l'interface à partir d'autres éléments de cette même interface, sans code behind.
Dans la partie libre en haut à droite de votre Grid, ajoutez un « DockPanel » avec dedans à gauche un « Label » (`DockPanel.Dock="Left"`) et un contrôle « Slider ». Pour ce dernier, mettre les propriétés du curseur comme cela :

```
Minimum="80" Maximum="320" Value="160" TickFrequency="80" TickPlacement="BottomRight"
                                                    SmallChange="5" LargeChange="20"
```



Nommez le « `ZoomSlider` ».

Vous allez relier la valeur renvoyée par le contrôle « Slider » à la taille des images.
Dans le WrapPanel du style « `PhotoListBoxStyle` » de la ListBox, ajoutez les 2 propriétés suivantes :

```
ItemHeight="{Binding ElementName=ZoomSlider, Path='Value'}"
ItemWidth="{Binding ElementName=ZoomSlider, Path='Value'}"
```

Et dans le style des « `ListBoxItem` », supprimez les tailles de l'image, du StackPanel et le MaxHeight.
Quel est l'effet obtenu ? :

Dans la partie « Ressources », ajoutez le code suivant (avant le style relatif aux « ListBoxItem ») :

```
<ToolTip x:Key="PreviewScreen" x:Shared="True" Background="Transparent"
         Placement="Right" Name="previewToolTip">
        <Border BorderBrush="RoyalBlue" BorderThickness="2" CornerRadius="2">
                <Image Source="{Binding Mode=OneWay}" Opacity="0.5" />
        </Border>
</ToolTip>
```

Et entre les balises « `<ControlTemplate.Triggers>` » relatives aux « items » de la « listbox », ajoutez :

```
<Trigger Property="IsMouseOver" Value="True">
        <Trigger.Setters>
                <Setter Property="ToolTip" Value="{StaticResource PreviewScreen}"/>
        </Trigger.Setters>
</Trigger>
```

Que fait cet ajout de code ? :

Qu'est ce qu'un élément « ToolTip » ? :

Quel est l'événement intercepté ? : _____
Quel est le lien entre les « items » et le « ToolTip » ? : _____

Pour terminer vous allez rajouter un Diaporama animé de présentation des images présentes dans la ListBox.
Ajoutez à votre projet une nouvelle fenêtre WPF nommée Diaporama. Ajoutez-y 2 champs Images non liés entre eux :

```
<Image x:Name="Image1" >    et    <Image x:Name="Image2" >
```

Sur votre fenêtre principale, ajoutez un bouton « Diaporama » à coté de votre contrôle « Slider », et interceptez sa méthode « Click ».
Dans cette méthode, créez et intanciez un nouvel objet « `Diaporama` » (opérateur new) et affichez-le avec la méthode « `ShowDialog` ».

Pour afficher un diaporama des images de la ListBox, il faut disposer de ces images dans la fenêtre Diaporama. Dans la classe de cette fenêtre, créez une liste publique de chaînes qui va contenir les chemins d'accès des images :

```
public List<String> sDiapo = new List<String>();
```

Après avoir créé votre objet fenêtre « `Diaporama` » et avant de l'afficher, remplissez la liste des chemins d'images en parcourant la ListBox avec par exemple une boucle « `foreach` ». Vous utiliserez :

```
Votre_Fenetre.sDiapo.Add(chemin_image.ToString());
```

En interceptant « Window_Activated », vérifiez que votre liste est bien initialisée :

```
ImageSourceConverter s = new ImageSourceConverter();
Image1.Source = (ImageSource)s.ConvertFromString(sDiapo[0]);
```

Ajoutez les options suivantes à « Image1 » :

```
<Image x:Name="Image1" >
    <Image.OpacityMask>
        <LinearGradientBrush StartPoint="0,0" EndPoint="1,0">
            <GradientStop Offset="1" Color="Black" x:Name="BlackStop"/>
            <GradientStop Offset="1" Color="Transparent" x:Name="TransparentStop"/>
        </LinearGradientBrush>
    </Image.OpacityMask>
</Image>
```

Que définissent ces 2 objects :

Créez une zone « `<Window.Resources>` » et ajoutez-y le scénario suivant :

```
<Storyboard x:Key="VisibleToInvisible" >
    <DoubleAnimation Storyboard.TargetName="TransparentStop"
                     Storyboard.TargetProperty="Offset" To="0"  Duration="0:0:2"   />
    <DoubleAnimation Storyboard.TargetName="BlackStop"
                     Storyboard.TargetProperty="Offset" To="0" Duration="0:0:2"  />
</Storyboard>
```

Créez une zone « `<Window.Triggers>` » et ajoutez-y le déclencheur suivant

```
<EventTrigger RoutedEvent="Window.Loaded">
    <EventTrigger.Actions>
        <BeginStoryboard Storyboard="{StaticResource  VisibleToInvisible}"/>
    </EventTrigger.Actions>
</EventTrigger>
```

Quel est le rôle de chacun de ces éléments et quel va être l'enchaînnement des événements ? :

Cette animation s'effectue avec la première image, mais il serait intéressant de passer à l'image suivante ou de lancer une autre animation à la fin du scénario. Pour exécuter du code après le scénario, définissez dans votre scénario une méthode appelée à cet effet :

```
<Storyboard x:Key="VisibleToInvisible" Completed="VisibleToInvisible_Completed" >
```

Et ajoutez-la à votre code behind :

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

Ajoutez le scénario inverse au premier :

```
<Storyboard x:Key="InvisibleToVisible" >
    <DoubleAnimation Storyboard.TargetName="TransparentStop"
                     Storyboard.TargetProperty="Offset" To="1"  Duration="0:0:2"   />
    <DoubleAnimation Storyboard.TargetName="BlackStop"
                     Storyboard.TargetProperty="Offset" To="1" Duration="0:0:2"   />
</Storyboard>
```

Puis dans votre méthode de fin de premier scénario, lancez-le avec le code suivant :

```
Storyboard sb = (Storyboard)this.FindResource("InvisibleToVisible");
sb.Begin();
(Ajoutez le namespace : using System.Windows.Media.Animation;)
```

Votre image doit réapparaître après les 2 scénarios.
Maintenant interceptez la méthode de fin de 2° scénario que vous nommerez «`InvisibleToVisible_Completed`» et faites qu'elle appelle le premier de façon à ce que l'animation soit continue en passant d'un scénario à l'autre en permanence.

C'est animé mais cela reste figé sur la première image. Dans « `InvisibleToVisible_Completed` », vous allez faire varier l'image à chaque appel de cette méthode. Pour cela :
- définissez une variable globale représentant l'indice de l'image affichée (int)
- dans la méthode chargez dans Image1 l'image de l'indice en cours avec un « `ImageSourceConverter` » comme vous avez fait dans « Window_Activated »
- en dessous, faites varier l'indice de 1 (attention si l'indice dépasse le nombre d'image de la liste, il doit repasser à zéro)

L'animation doit passer d'une image à l'autre.

Cette animation fonctionne bien mais on peut très facilement l'améliorer. Imaginez que la disparition de l'image corresponde à l'apparition de l'image suivante. Pour cela, il suffit dans «`VisibleToInvisible_Completed`» de charger Image2 par l'image de l'indice courant puis d'augmenter l'indice de 1. En vérité Image2 va être charger par l'image suivante (car l'indice a été modifié à la fin du scénario suivant) et va se découvrir à la disparition de Image1, ensuite Image1 va recouvrir Image2 avec une 3° image (car l'indice aura été augmenté dans cette méthode à la fin de ce scénario). Dans le XAML, mettez Image2 avant Image1 car c'est Image1 qui recouvre et découvre Image2.
Utilisez des images de même proportion, sinon il faudra « Stretcher » les images pour obtenir un effet propre.