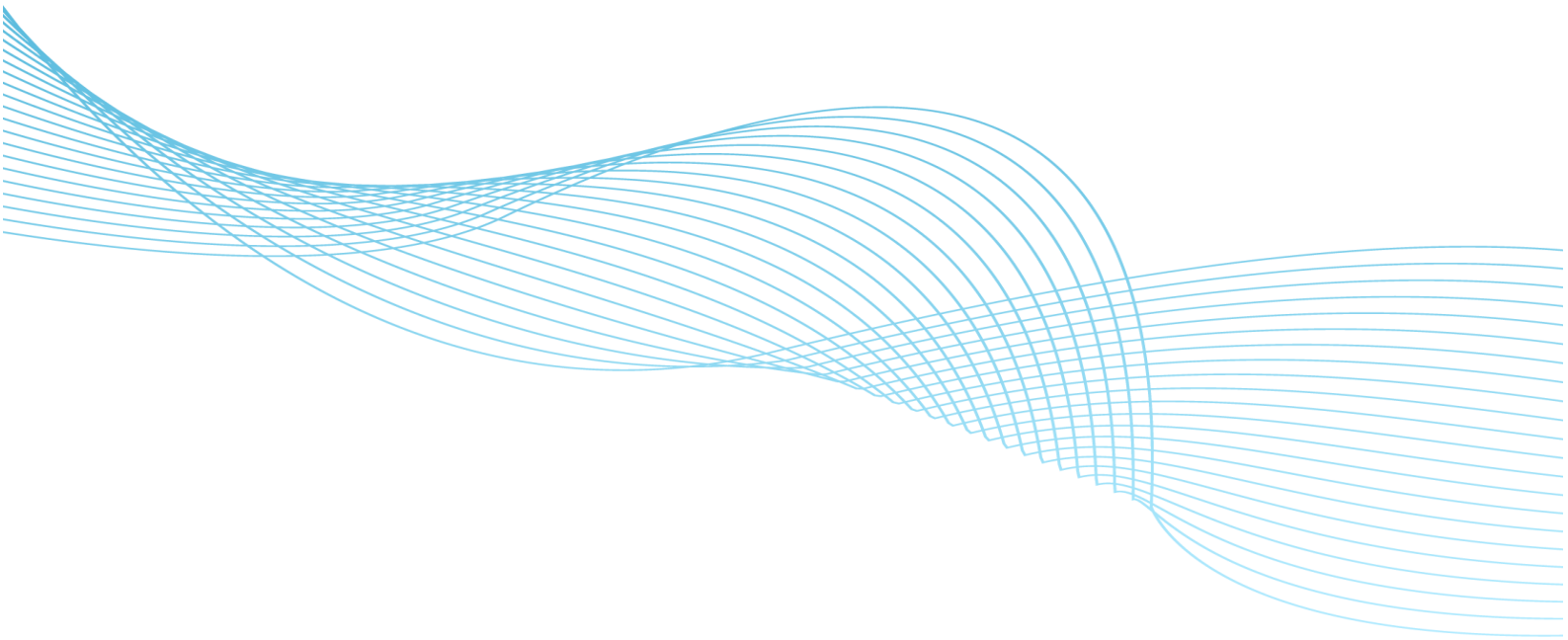
Dotnet France
Technologies Sharepoint, SQL Server & .NET

Association Dotnet France

Graphismes et Multimedia

Version 1.0



Sommaire

1 Introduction

Nous avons vu au cours des chapitres précédents que WPF est un formidable outil pour créer des applications aux interfaces riches. Néanmoins, nous n'avions pas encore abordé tous les points.

Dans ce chapitre nous allons voir comment créer des applications contenant des images, des formes, voir de la vidéo et du son.

Si le concept de graphisme 2D était déjà connu avec le Framework 2.0, la sortie du Framework 3.0 a permit de rajouter le support de la vidéo et du son de manière simple grâce à MediaElement.

De plus avec WPF, de nombreuses tâches graphiques autrefois pénible pour le programmeur et les graphistes sont simplifiées grâce à l'utilisation conjointe de Visual Studio et Expression Blend.

Enfin, vous aurez l'occasion en fin de chapitre d'apprendre ce qu'il faut savoir pour créer des animations fantastiques en seulement quelques lignes de code très claire.

2 Graphismes 2D

2.1 Les images

L'apport des images à une application semble évident, que ce soit pour ajouter le logo de l'entreprise ou créer une galerie d'image, il vous sera toujours utile de savoir comment manipuler les images. La manipulation d'images en WPF peut se faire totalement en XAML grâce à la balise <Image/>.

Dans l'exemple suivant nous affichons le logo de Dotnet-France dans une fenêtre :

```
<Window x:Class="Image2D.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300">
    <Grid>
        <Image Height="65" Width="200"
          Source="C:\Users\PFCH5\Documents\Visual Studio
          2008\Projects\Image2D\Image2D\logopetit.jpg" />

    </Grid>
</Window>
```

Note : Vous pouvez remarquer que la taille de l'image a été redéfinie grâce aux attributs Height et Width. Sans ces attributs l'image prend tout l'espace disponible, de la même manière que les autres contrôles.

Voici d'autres attributs qu'il est possible d'ajouter à votre balise <Image /> :

- Source sert à définir l'adresse de l'image sur l'ordinateur, l'adresse peut être absolue (voir exemple ci-dessus) ou relative.
- Height et widht permettent de contrôler la hauteur et la largeur (comme dans l'exemple ci-dessus).
- Stretch.None empêche l'image de s'étirer dans tous l'espace qu'elle peut occuper.
- Stretch.Fill étire l'image afin de remplir totalement la zone qu'elle peut occuper.
- Stretch.Uniform étire l'image en multipliant uniformément sa largeur et sa longueur.

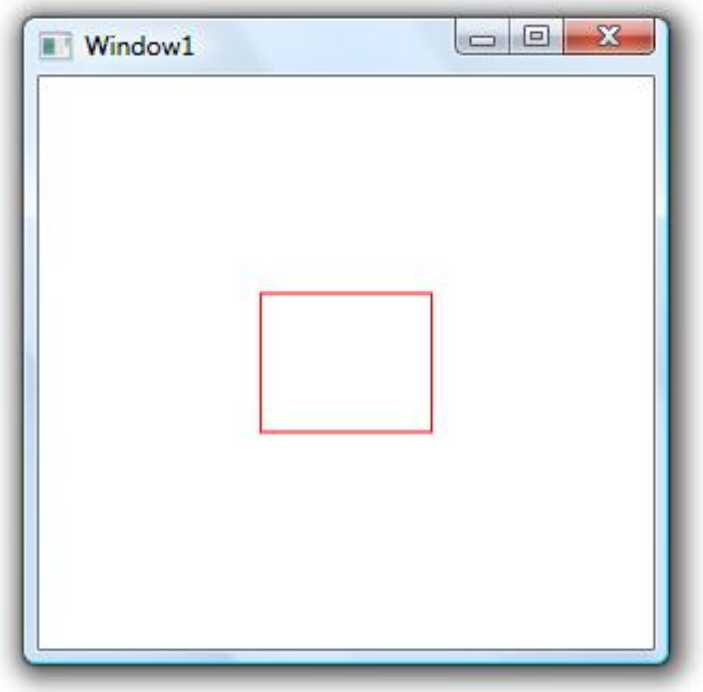
Comme vous pouvez le voir, l'ajout d'image est vraiment simple, n'hésitez pas à les utiliser.

2.2 Les formes géométriques

L'insertion de formes géométriques est une des possibilités 2D offerte par WPF que l'on retrouvait déjà en WinForm. Dans ce sous chapitre, nous allons réaliser l'insertion de forme géométrique dans notre fenêtre principale puis voir comment leur donner un aspect graphique différent.

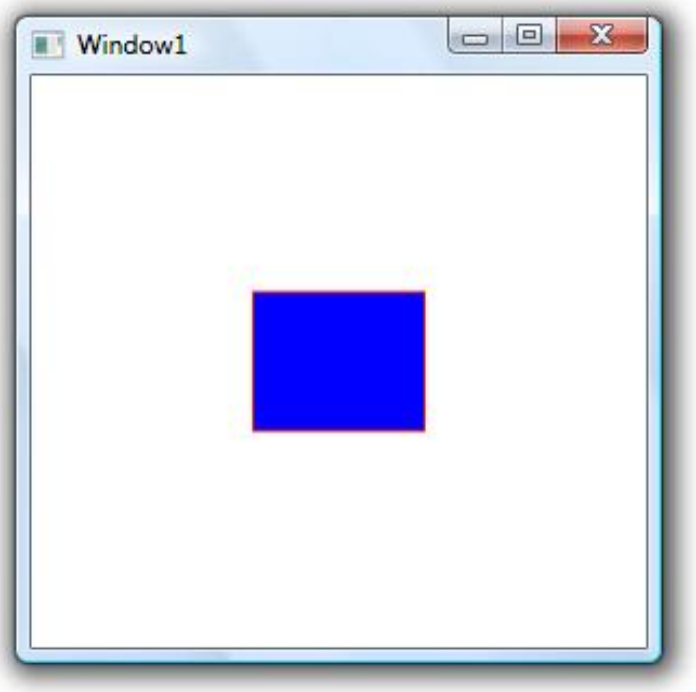
```
<Rectangle Height="65" Width="80" Name="monRectangle" Stroke="Red"/>
```

Ici notre rectangle se nommera « monRectangle » et sera de couleur rouge. Sa taille sera de 65 dip en hauteur et 80 dip en largeur.



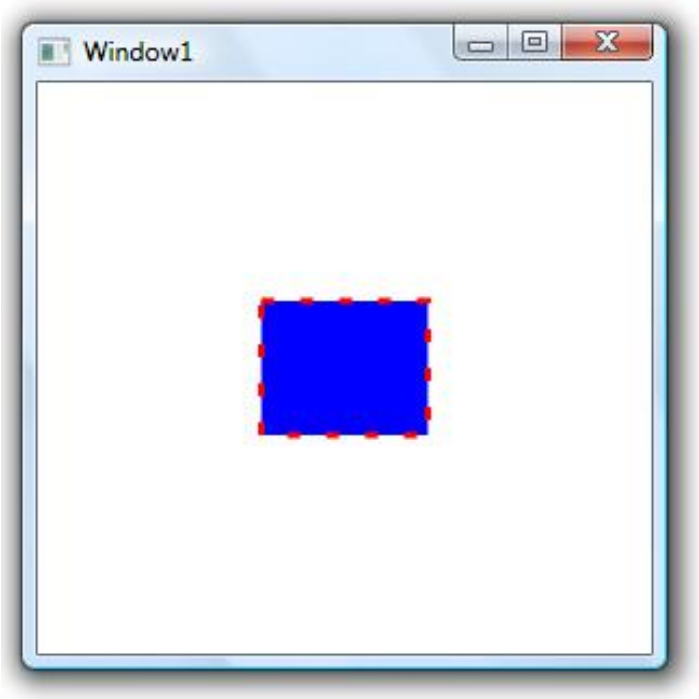
L'attribut Fill permet de colorer l'intérieur du rectangle (on lui ajoute comme valeur la couleur souhaitée).

```
<Rectangle Name="monRectangle" Height="65" Width="80" Stroke="Red"
Fill="Blue"/>
```



- L'attribut StrokeDashArray permet de transformer le contour en pointillés. Il prend 2 paramètres de type double. Le premier étant la longueur des pointillés, le deuxième l'espacement entre ceux-ci.
- L'attribut StrokeThickness sert à définir la largeur du contour

```
<Rectangle Name="monRectangle" Height="65" Width="80" Stroke="Red"
Fill="Blue" StrokeDashArray="2,4" StrokeThickness="3"/>
```


Window1

3 Musique Et Vidéo

3.1 Musique

Une des nouveautés dans le Framework 3.0 et 3.5 est l'introduction du contrôle multimédia MediaElement. Il réuni toutes les fonctions nécessaires au contrôle de flux média (son et vidéo) et se base sur les codecs installés sur la machine compatibles avec Windows Media Player.

Pour notre exemple nous insérons une balise MediaElement (auquel nous ajoutons l'attribut Name et Source) et deux bouton Play et Stop dans notre fichier XAML.

```
<MediaElement Name="musique" Source=" C:\Users\PFCH5\Documents\Visual
    Studio 2008\Projects\MusiqueWPF\MusiqueWPF\mySong.mp3" />
<Button Margin="14,0,92,31" Name="Play" VerticalAlignment="Bottom"
    Height="21.96" Click="Play_Click">Play</Button>
<Button Margin="0,0,11,31" Name="Stop" VerticalAlignment="Bottom"
    Height="21.96" Click="Stop_Click" HorizontalAlignment="Right"
    Width="75">Stop</Button>
```

Puis ensuite nous définissons le comportement de nos trois contrôles dans le code C# :

```
private void Play_Click(object sender, RoutedEventArgs e)
{
    musique.LoadedBehavior = MediaState.Manual;
    musique.Play();
}

private void Stop_Click(object sender, RoutedEventArgs e)
{
    musique.LoadedBehavior = MediaState.Manual;
    musique.Stop();
}
```

L'attribution de la valeur de l'énumération MediaState à la propriété LoadedBehavior nous permet d'indiquer quel sera le comportement du MediaElement au chargement.

Ici nous indiquons que le MediaElement sera en mode Manual ce qui nous permettra de contrôler le flux grâce aux méthodes Play et Stop.

3.2 Vidéo

L'utilisation de MediaElement avec de la vidéo est similaire à celle avec de la musique.

Dans notre exemple nous rajoutons un bouton Pause, et changeons la source du MediaElement par notre vidéo.

```
<MediaElement Name="video" Margin="14,11,11,82"
     Source="C:\Users\PFCH5\Videos\Butterfly.wmv" />
<Button Margin="14,0,0,31" Name="Play" VerticalAlignment="Bottom"
     Height="21.96" Click="Play_Click" HorizontalAlignment="Left"
     Width="80">Play</Button>
<Button Margin="99,0,99,31" Name="Pause" VerticalAlignment="Bottom"
     Height="21.96" Click="Pause_Click">Pause</Button>
<Button Margin="0,0,11,31" Name="Stop" VerticalAlignment="Bottom"
     Height="21.96" Click="Stop_Click" HorizontalAlignment="Right"
     Width="80">Stop</Button>
```

Dans le code C# nous rajoutons simplement la méthode Pause_Click :

```
private void Pause_Click(object sender, RoutedEventArgs e)
        {
            video.LoadedBehavior = MediaState.Manual;
            video.Pause();
        }
```

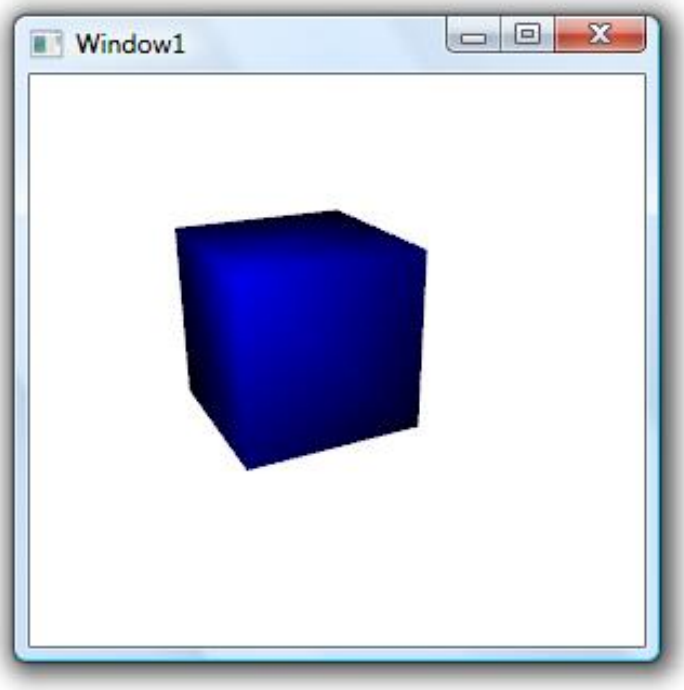
4 Un mot sur la 3D

Un autre des avantages apportés par WPF par rapport au WinForm est sa capacité à afficher des modèles 3D qui sont calculés par la carte graphique.

Pour créer un modèle 3D en XAML, la meilleure solution reste de posséder un plugin permettant l'export de votre travail en XAML. Il en existe pour la plupart des logiciels professionnels du marché. Voici un morceau de code permettant de dessiner un cube de couleur bleu avec des ombres portées, nous ne le commenterons pas dans ce chapitre.

```
<Window x:Class="_3D.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:p="http://www.codeplex.com/perspective"
    Title="Window1" Height="300" Width="300">
    <Grid>
        <Viewport3D Name="viewport3D1">
            <Viewport3D.Camera>
               <PerspectiveCamera x:Name="camMain" Position="4 2 2"
                        LookDirection="-4 -2 -2">
                    </PerspectiveCamera>
                  </Viewport3D.Camera>
                    <ModelVisual3D>
                     <ModelVisual3D.Content>
                      <DirectionalLight x:Name="dirLightMain"
                          Direction="-5,-2,-3">
                       </DirectionalLight>
                        </ModelVisual3D.Content>
                         <GeometryModel3D>
                          <GeometryModel3D.Geometry>
                           <MeshGeometry3D x:Name="meshMain"
                           Positions="0 0 0  1 0 0  0 1 0  1 1 0  0 0
                           1  1 0 1  0 1 1  1 1 1"
                          TriangleIndices="2 3 1  2 1 0  7 1 3  7 5 1
6 5 7  6 4 5  6 2 0  2 0 4  2 7 3  2 6 7  0 1 5  0 5 4">
                            </MeshGeometry3D>
                           </GeometryModel3D.Geometry>
                          <GeometryModel3D.Material>
                         <DiffuseMaterial x:Name="matDiffuseMain">
                          <DiffuseMaterial.Brush>
                           <SolidColorBrush Color="blue"/>
                          </DiffuseMaterial.Brush>
                        </DiffuseMaterial>
                    </GeometryModel3D.Material>
                </GeometryModel3D>
            </ModelVisual3D.Content>
        </ModelVisual3D>
    </Viewport3D>
  </Grid>
</Window >
```

Voici le résultat si nous compilons :



5 Animations

5.1 Storyboard

C'est une nouveauté de WPF, les Storyboard permettent de créer très facilement des animations, que ce soit de vos contrôles, d'images ou de vidéos. Un Storyboard est tout simplement un objet englobant des animations diverses. Il existe deux manières de remplir des Storyboard. Soit en les remplissant de balises de type From-To, soit de balises de type Key-Frame. Vous pouvez également mixer les deux.

Avant toute chose, il faut savoir qu'un Storyboard est considéré comme une ressource, il aura donc toute sa place dans la partie Ressource d'un contrôle, ou dans un fichier de ressources. Nous pouvons également utiliser les Event Triggers pour englober un Storyboard.

Si nous voulons utiliser un Storyboard en tant que ressource de la fenêtre principale, et ainsi pouvoir l'appliquer à toute sorte de contrôle, il suffit de placer le Storyboard dans les ressources de la fenêtre :

```
<!--XAML-->
<Window
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      x:Class="WPF.Window1" x:Name="Window" Title="Window1"
    xmlns:local="clr-namespace:WPF"
      Width="300" Height="300">
    <Window.Resources>
        <Storyboard x:Name="MonAnimation">
            <!-- Code de l'animation -->
        </Storyboard>
    </Window.Resources>
    <StackPanel>
        <!-- Le contenu de ma fenêtre -->
    </StackPanel>
</Window>
```

Nous verrons dans la sous partie suivante comment utiliser ce Storyboard.

Si par contre nous souhaitons appliquer le Storyboard qu'à un seul contrôle en fonction de son état, nous allons utiliser les Event Triggers :

```
<!--XAML-->
<Window
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      x:Class="WPF.Window1" x:Name="Window" Title="Window1"
    xmlns:local="clr-namespace:WPF"
      Width="300" Height="300">
    <Window.Resources>
        <Style TargetType="Button">
            <Style.Triggers>
                <EventTrigger RoutedEvent="MouseEnter">
                    <BeginStoryboard>
                        <Storyboard>
                        </Storyboard>
                    </BeginStoryboard>
                </EventTrigger>
            </Style.Triggers>
        </Style>
    </Window.Resources>
    <StackPanel>
        <Button Height="100" Width="100" x:Name="MonBouton">
            <Button.Triggers>
                <EventTrigger RoutedEvent="Button.Click">
                    <BeginStoryboard>
                        <Storyboard>
                        </Storyboard>
                    </BeginStoryboard>
                </EventTrigger>
            </Button.Triggers>
            Coucou
        </Button>
    </StackPanel>
</Window>
```

Comme vous pouvez le remarquer nous pouvons placer les EventTriggers en tant que Style ou en tant que Triggers d'un contrôle. Dans le cas des styles, vous ne pourrez pas définir les cibles qui seront modifiées par l'animation. Faites donc attention et préférez utiliser les propriétés Triggers de vos contrôles.

Comme vous pouvez le voir, nous avons utilisé dans nos exemples les balises <BeginStoryboard> à l'intérieur de nos Triggers. Celles-ci vont permettre d'indiquer l'animation à jouer quand le Trigger est déclenché, vous pouvez aussi définir des comportements différents grâce aux balises suivantes :

Comportement	Balise
Mettre l'animation en pause	<PauseStoryboard>
Reprendre l'animation	<ResumeStoryboard>
Changer la vitesse de l'animation	<SetStoryboardSpeedRatio>
Passer l'animation courante	<SkipStoryboardToFill>
Arrêter l'animation	<StopStoryboard>
Supprimer le Storyboard et libérer la mémoire	<RemoveStoryboard>

5.1.1 From-To

Nous l'avions dit, nous pouvons utiliser deux types d'animations dans nos Storyboard. Nous allons tout d'abord parler des animations de type From-To. Les animations From-To permettent de créer des animations basiques en modifiant la valeur d'une propriété en fonction d'une durée.

Il existe trois types de balises afin de créer des animations From-To afin de pouvoir gérer tous les cas possibles :

Balise	Comportement	Exemple
`<DoubleAnimation>`	Permet de modifier une propriété de type double	`-Height` `-Width`
`<ColorAnimation>`	Permet de modifier une propriété de type Color	`-Background` `-Foreground`
`<PointAnimation>`	Permet de modifier une propriété de type Point	`-Centre d'une forme Géométrique` `-Angle d'une image`

Une fois que nous avons choisi la bonne balise, nous devons maintenant lui indiquer ce qu'elle devra faire. Pour cela nous allons lui rajouter :

- Une durée
- Les valeurs de début (From) et de fin (To)
- Le contrôle à animer
- La propriété à modifier

Voici un tableau récapitulant tous les attributs que vous allez pouvoir utiliser :

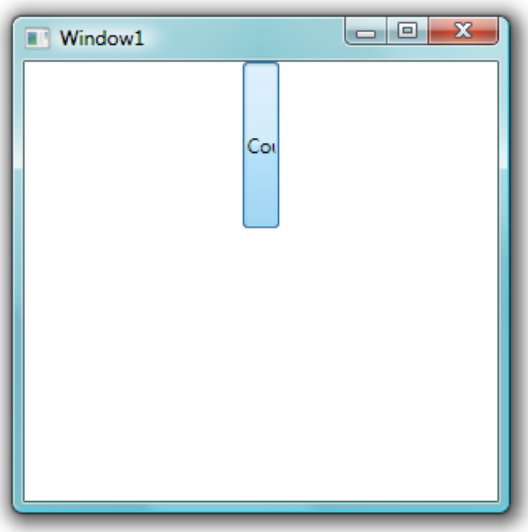
Attribut	Comportement	Valeur Possibles
Valeur de début et de fin		
From	Indique la valeur de départ	Toutes les valeurs que peut prendre la propriété
To	Indique la valeur de fin	Idem
By	Permet de modifier la valeur initiale en fonction de la valeur de By (Ex, si la valeur initiale est 10 et By 20, l'animation partira de 10 et ira jusqu'à 30).	Idem
From et To	Permet de jouer l'animation entre la valeur de From et la valeur de To	Idem
From et By	Permet de jouer l'animation entre la valeur de From et la valeur de From+By	Idem
Autres		
Duration	Permet de définir la durée de l'animation (C'est également un attribut de la balise Storyboard)	hh :mm:ss
Storyboard.TargetName	Permet d'indiquer le contrôle qui sera animé (C'est également un attribut de la balise Storyboard)	Le nom du contrôle.
Storyboard.TargetProperty	Permet d'indiquer la propriété à modifier	Plusieurs formes sont possibles, par exemple : - (Button.Width) - (Fill).(Color)
Autoreverse	Permet de faire repartir l'animation en arrière lorsqu'elle est terminée	True : L'animation repart vers le début False : Valeur par défaut
BeginTime	Permet de rajouter un délais avant que l'animation commence effectivement.	hh :mm :ss
RepeatBehavior	Permet d'indiquer si l'animation devra être répétée ou non.	Un nombre entier n : L'animation sera répétée n fois hh :mm:ss : L'animation sera répétée tant que le compteur tourne Forever : L'animation est répétée en boucle
SpeedRatio	Permet de changer la vitesse de l'animation en la multipliant par la valeur de SpeedRatio.	Un nombre entier n

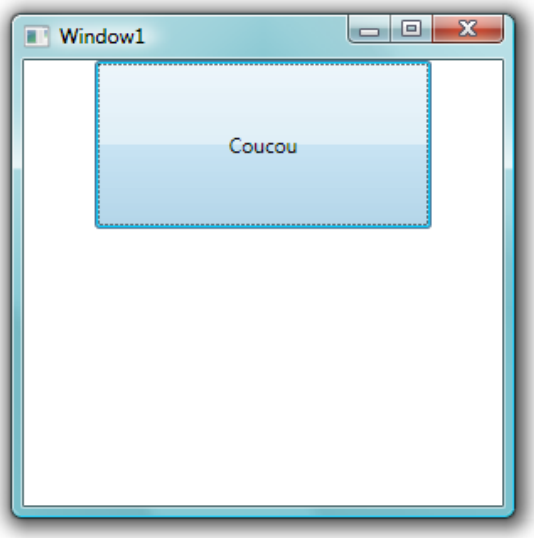
Nous avons tout en main pour construire une animation simple. Par exemple, nous allons rajouter à notre code précédent utilisant les triggers une animation afin de grossir un bouton quand on cliquera dessus :

```
<!--XAML-->
<Window
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      x:Class="WPF.Window1" x:Name="Window" Title="Window1"
      xmlns:local="clr-namespace:WPF"
      Width="300" Height="300">
    <StackPanel>
        <Button Height="100" Width="100" x:Name="MonBouton">
            <Button.Triggers>
                <EventTrigger RoutedEvent="Button.Click">
                    <BeginStoryboard>
                        <Storyboard x:Name="Agrandir">
                            <DoubleAnimation From="10" To="200"
                              Duration="0:0:1"
                              Storyboard.TargetName="MonBouton"
                              Storyboard.TargetProperty="(Button.Width)"
                              />
                        </Storyboard>
                    </BeginStoryboard>
                </EventTrigger>
            </Button.Triggers>
            Coucou
        </Button>
```

Pour créer cette animation, nous avons utilisé la balise DoubleAnimation car nous allons modifier la propriété Width du button. A cette balise nous lui ajoutons les valeurs de départ, 10 et d'arrivée, 200. Ensuite nous indiquons la durée de l'animation, soit une seconde. Enfin nous définissons la propriété ciblée ainsi que le nom du contrôle (ici facultatif car nous utilisons les Triggers du button).

Si nous lançons et cliquons sur le bouton, il va alors diminuer sa largeur à 10 puis la faire grossir jusqu'à 200 en une seconde :





5.1.2 Key-Frame

A la différence des animations de type From-To, les Key-Frame vont nous permettre d'établir plusieurs paliers dans notre animation. Si nous reprenons l'exemple du bouton précédent, nous préférerions peut être que celui-ci diminue progressivement de sa taille par défaut vers la taille 10, puis grossir jusqu'à une largeur de 200 et enfin retourner à sa valeur initiale.

Tout comme avec les balises de type From-To nous avons une balise par type de valeur à modifier. Voici les trois principales balises que nous pouvons utiliser :

- DoubleAnimationUsingKeyFrames
- ColorAnimationUsingKeyFrames
- PointAnimationUsingKeyFrames

Comme vous pouvez vous en douter au vu de leur nom, elles ont respectivement les mêmes champs d'actions que les balises DoubleAnimation, ColorAnimation et PointAnimation.

Dans les animations de type Key-Frame, nous allons pouvoir faire englober aux balises précédente trois nouvelles balises qui vont nous permettre d'effectuer trois animations au comportement différents :

- LinearXKeyFrame
- DiscreteXKeyFrame
- SplineXKeyFrame

Note : X Doit être remplacée par le type de la propriété à modifier, par exemple, si on utilise une animation Linear avec une DoubleAnimationUsingKeyFrames, on utilisera la balise LinearDoubleKeyFrame.

Afin de comprendre le comportement de chacune de ces balises, nous allons étudier trois exemples différents.

5.1.2.1 Linear

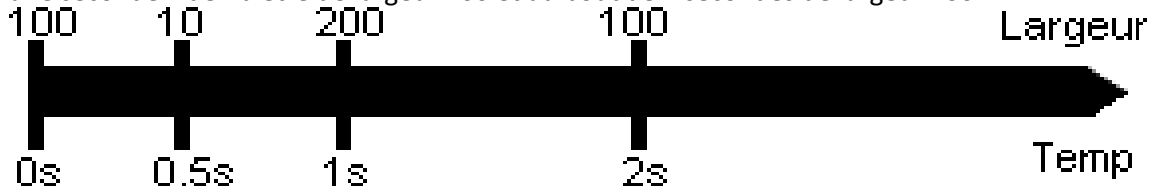
L'animation Linear se rapproche des animations From-To. Les valeurs changent de manière linéaire en fonction du temps.

Nous allons donc faire en sorte que notre button rapetisse à une largeur de 10 avant de grossir à une valeur de 200, puis de revenir à 100 :

```
<!--XAML-->
<Storyboard x:Name="Agrandir">
    <DoubleAnimationUsingKeyFrames Storyboard.TargetName="MonBouton"
     Storyboard.TargetProperty="(Button.Width)">
        <LinearDoubleKeyFrame Value="10" KeyTime="00:00:0.5"/>
        <LinearDoubleKeyFrame Value="200" KeyTime="00:00:1"/>
        <LinearDoubleKeyFrame Value="100" KeyTime="00:00:2"/>
    </DoubleAnimationUsingKeyFrames>
</Storyboard>
```

Nous retrouvons les balises et attributs que nous avons déjà vus. Seul l'attribut KeyTime devrait attirer votre attention. KeyTime permet de définir à quel moment la valeur indiquée dans l'attribut Value devra être assignée à la propriété.

Ici par exemple, à 0,5 secondes notre button devra impérativement être de largeur 10. Au bout d'une seconde il devra être de largeur 200 et au bout de 2 secondes de largeur 100.



5.1.2.2 Discrete

L'animation Discrete se déroule de la même manière que Linear à la différence que les valeurs sont changée de manière ponctuelle a chaque pallié. Cela donne une sensation de changement par à-coups.

```
<!--XAML-->
<Storyboard x:Name="Agrandir">
    <DoubleAnimationUsingKeyFrames Storyboard.TargetName="MonBouton"
     Storyboard.TargetProperty="(Button.Width)">
        <DiscreteDoubleKeyFrame Value="10" KeyTime="00:00:0.5"/>
        <LinearDoubleKeyFrame Value="200" KeyTime="00:00:1"/>
    </DoubleAnimationUsingKeyFrames>
</Storyboard>
```

Dans cet exemple, nous avons remplacé la première balise LinearDoubleKeyFrame par DiscreteDoubleKeyFrame et supprimé la dernière. Le comportement de notre bouton ressemblera ainsi à l'exemple que nous avions vu avec From-To, à savoir que le bouton passe brusquement d'une largeur de 100 à une largeur de 10, puis grossit progressivement vers une largeur de 200.

L'avantage de Discrete est de pouvoir l'utiliser conjointement avec des propriétés de type objets. En effet il existe en plus des balises d'animation standard d'autres balises un peu moins usitées telles qu'ObjectAnimationUsingKeyFrame qui nous permet d'utiliser toute sorte d'objets ne pouvant pas voir leur valeur modifié de manière progressive, ou dont le type est trop complexe. Par exemple, nous pourrons grâce à cela dessiner un cercle remplit d'un dégradé, et donc utiliser des Brush en tant que valeur.

5.1.2.3 Spline

Une animation de type Spline est une animation Linear qui progresse de frame en frame selon une courbe de bézier.

Pour configurer les paramètres de la courbe de bézier, nous allons utiliser l'attribut KeySpline qui prend un couple de paramètre pour chaque point de contrôle de votre courbe :

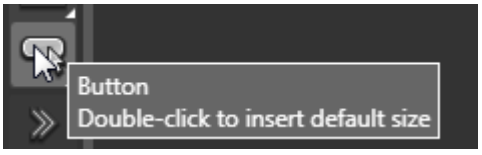
```
<!--XAML-->
<Storyboard x:Name="Agrandir">
    <DoubleAnimationUsingKeyFrames  Storyboard.TargetName="MonBouton"
Storyboard.TargetProperty="(Button.Width)">
        <SplineDoubleKeyFrame Value="10" KeyTime="00:00:0.5"/>
        <SplineDoubleKeyFrame Value="200" KeyTime="00:00:1"
KeySpline="0.8,0 0.1,0"/>
        <SplineDoubleKeyFrame Value="100" KeyTime="00:00:2"
KeySpline="0.5,0.2 0.3,0"/>
    </DoubleAnimationUsingKeyFrames>
</Storyboard>
```

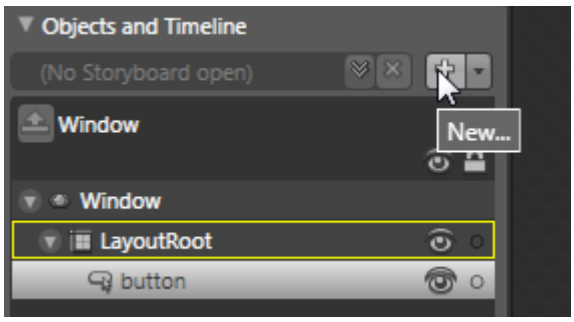
A moins de faire les calculs manuellement, définir les KeySpline a la main est très contraignant, aussi, nous verrons dans le point suivant comment faire une animation de type Spline avec Expression Blend.

5.1.3 Storyboard avec Expression Blend

Nous avons vu comment créer un storyboard en XAML complètement « à la main ». C'est une opération qui peut s'avérer très contraignante, surtout pour des animations complexes. Avec Expression Blend, les animations sont facilitées, nous allons voir comment les construire :

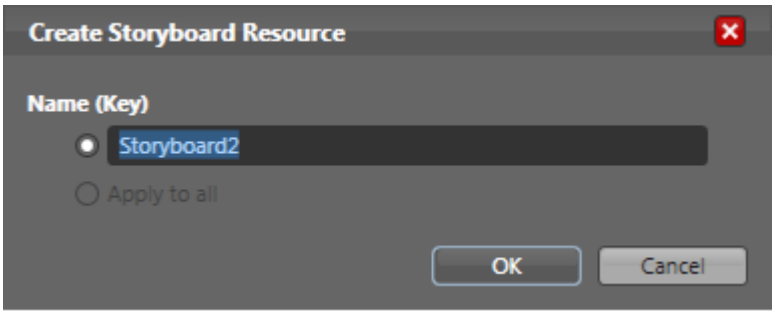
Tout d'abord, nous allons ajouter à notre fenêtre un bouton simple.

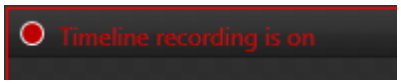




Ensuite, dans la partie de gauche de l'écran, nous appuyons sur la petite croix suivant la case (No Storyboard open).

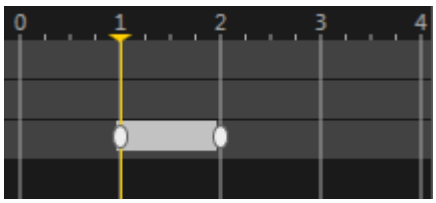
Ensuite indiquez un nom à votre Storyboard

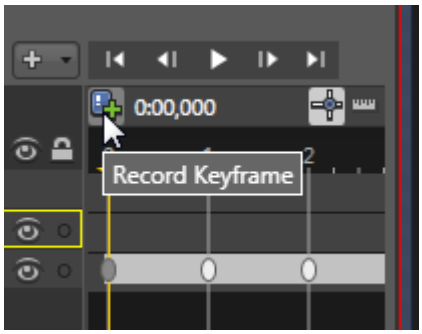




A partir de là, Expression Blend change d'aspect. Un point rouge apparait en haut de l'écran et indique qu'il va enregistrer les mouvements et changements que vous appliquez à vos contrôles dans la KeyFrame actuelle.

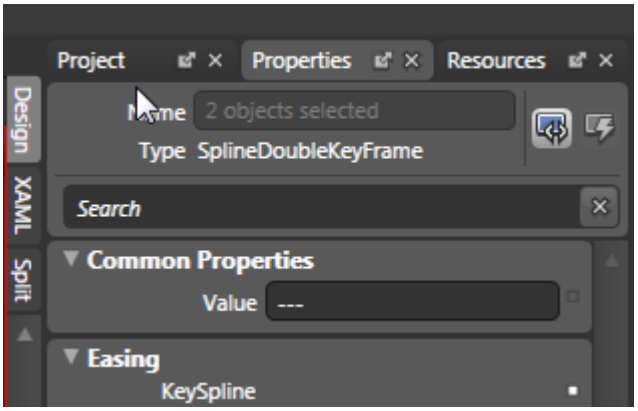
Pour changer de KeyFrame, il suffit de déplacer sur la Timeline la barre jaune en fonction du temps qu'il faut a votre animation pour se dérouler. Vous pourrez ensuite déplacer votre contrôle et voir qu'un point blanc apparait sur la Timeline. Si vous effectuez des modifications à deux temps différents, le point est remplacé par une barre.



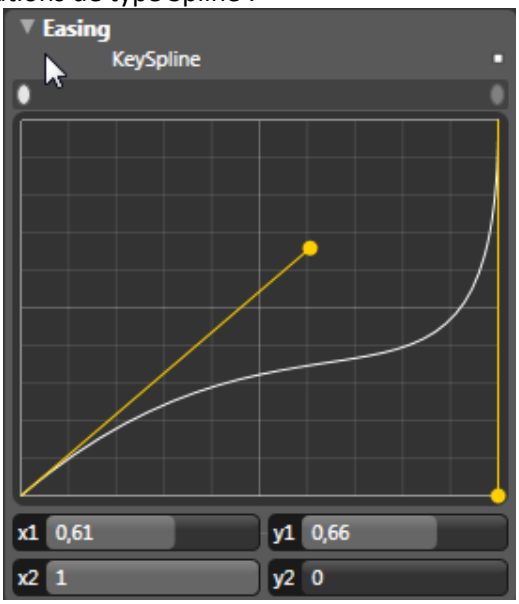


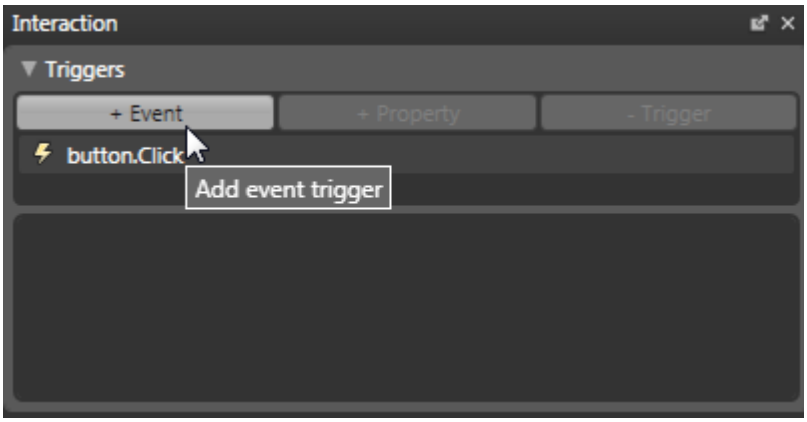
Vous pouvez ajouter à la main une Keyframe en appuyant sur le bouton situé au dessus de la Timeline.

Si vous souhaitez que votre animation soit de type Spline, vous pouvez cliquer sur un point dans la Timeline (et non pas sur une barre) et ensuite cliquer sur l'onglet Properties en haut à droite de votre écran.



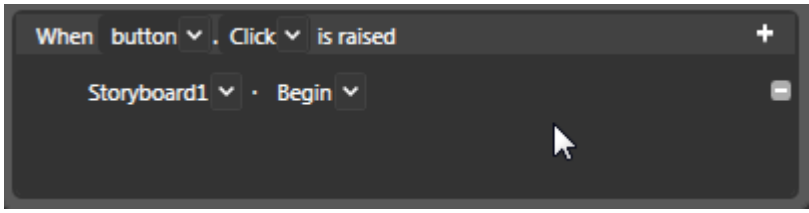
A partir de là, cliquez sur le header Easing pour dérouler la courbe de bézier nous permettant de gérer facilement les animations de type Spline :



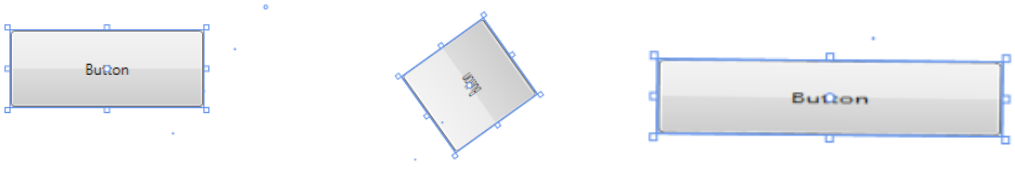


Enfin, il ne nous reste plus qu'a créer notre déclencheur de Storyboard, grâce à un EventTrigger. Pour cela il suffit de cliquer sur le bouton « + Event » dans la partie Trigger en haut à gauche de votre écran

Puis ensuite de configurer votre Trigger grâce aux options qui nous sont fournies
Par exemple, quand le contrôle button est clické, on lance le Storyboard1



Maintenant, si nous cliquons sur notre Button, l'animation se déroule :



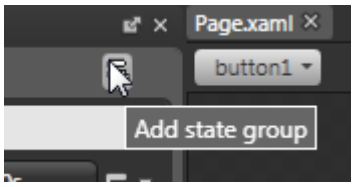
En utilisant Expression Blend, vous vous simplifierez grandement la vie lorsqu'il sagira de créer des animations, des EventTriggers, changer des couleurs, des formes, les tailles d'un contrôle. Si l'on rajoute à cela la puissance du moteur d'animation, le temps que le programmeur et les graphistes vont gagner à concevoir l'application en utilisant Blend lui donne un avantage certain sur ce que l'on connaissait en WinForm.

5.2 Silverlight et le VisualStateManager

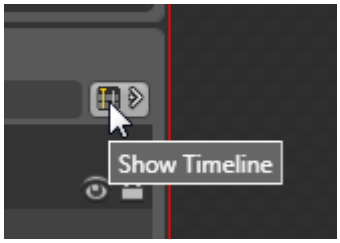
Jusqu'à présent, nous avons vu comment lancer des Storyboard avec des EventTriggers en WPF. Malheureusement, en Silverlight, il n'est pas possible d'utiliser les Triggers, ceux-ci ont été retirés pour des raisons de lourdeur. Afin de pallier à ce manque, nous disposons de VisualStateManager.

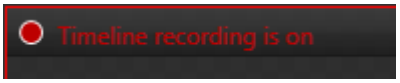
Nous allons voir comment nous servir de VisualStateManager avec Expression Blend et Visual Studio. Tout d'abord, créez un projet Silverlight. Une fois la solution chargée dans Expression Blend, rajoutez un Bouton dans Page.xaml.

Une fois cela fait, nous allons ajouter à notre Page.xaml un nouvel état, pour cela, nous allons dans la partie State située en haut a gauche. Cliquez d'abord sur la petite icône avec un « + » pour créer un groupe d'état, puis donnez-lui un nom.



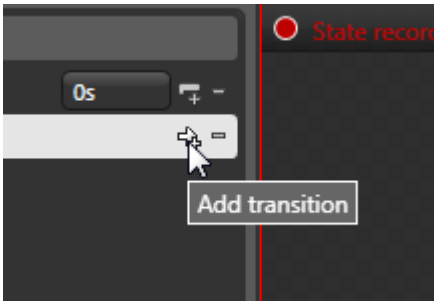
Ensuite nous allons créer dans ce groupe d'état, un nouvel état, que nous allons appeler DeplacerButton.



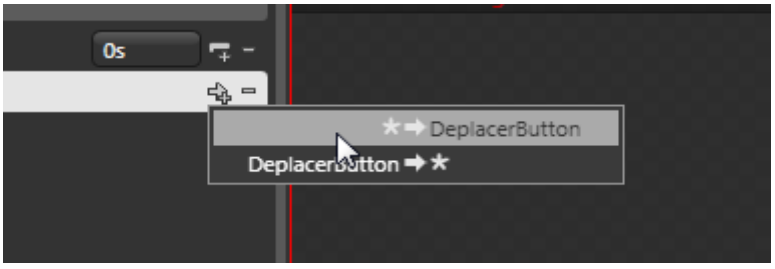


Nous avons maintenant définit dans notre UserControl Page.xaml un nouvel état appelé DeplacerButton. Dans cet état, nous allons simplement déplacer notre Button. Pour cela, cliquez sur l'état que vous venez de créer, vérifiez que l'indicateur d'enregistrement est allumé, et déplacez le bouton. Si vous voulez effectuer une animation plus complexe, vous pouvez dérouler la Timeline en cliquant sur cet icône situé dans l'espace Objects and Timeline.

Vous pouvez, si vous le souhaitez, indiquer des temps de transition entre vos états. Vous aurez le choix par exemple de mettre un temps de transition d'une seconde entre votre état de base et DeplacerButton, puis un autre temps de transition lorsque vous quittez DeplacerButton pour un troisième état. Pour cela, il vous faut cliquer sur l'icône « Add transition ».



A partir de là plusieurs choix vous sont proposés selon le type de transition que vous souhaitez :

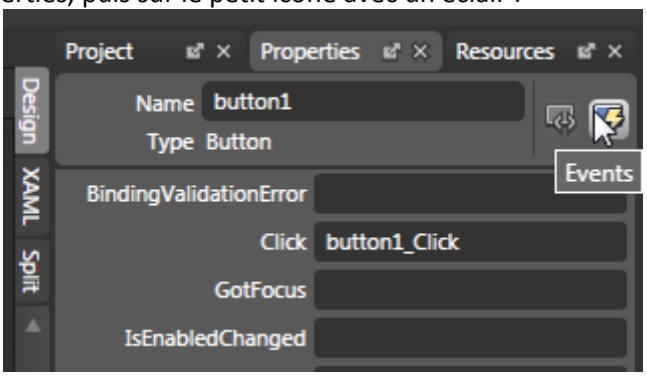


Ensuite vous pouvez indiquer un temps de transition en seconde.

Maintenant, nous voulons que notre bouton se déplace seulement lorsque nous cliquons dessus, nous allons associer une méthode à l'évènement OnClick du bouton. Vous pouvez soit rajouter dans votre XAML le morceau de code suivant à votre button :

```
<!--XAML-->
<Button Click="button1_Click" />
```

Sinon sous Expression Blend, cliquez sur le bouton, puis dirigez vous en haut à droite de l'écran, cliquez sur Properties, puis sur le petit icône avec un éclair :



De là, indiquez le nom de votre méthode dans le champ suivant Click, cela aura pour effet de lancer Visual Studio si ce n'est pas déjà fait. Vous n'aurez plus qu'à définir le comportement de la méthode. Dans notre cas, nous voulons que lorsque l'on clique sur le bouton, il se déplace, c'est-à-dire que l'on doit passer de l'état de base à l'état DéplacerButton. Pour faire cela, nous devons obligatoirement travailler en code behind. Voici le code que nous allons utiliser dans le fichier Page.xaml.cs :

```
//C#
private void button1_Click(object sender, RoutedEventArgs e)
{
    VisualStateManager.GoToState(this, "DeplacerButton", true);
}
```

VisualStateManager nous permet d'utiliser la méthode statique GoToState qui va faire transiter un état vers un autre. Le premier argument de cette méthode permet d'indiquer le contrôle qui contient l'état vers lequel transiter. Dans notre cas, c'est la Page qui contient l'état, nous mettons donc le mot clef this, ensuite vous devrez indiquer l'état vers lequel transiter, et enfin, le troisième argument permet d'indiquer si l'on veut prendre en compte les temps de transitions

Maintenant si nous cliquons sur notre bouton, celui-ci devrait se déplacer exactement comme nous le souhaitions.

Grâce à VisualStateManager vous devriez être en mesure de créer n'importe quelle animation dans votre application, que ce soit en Silverlight ou en WPF.

6 Conclusion

Dans ce chapitre nous avons vu l'essentiel de ce qu'il faut savoir pour créer des interfaces riches contenant des images, des animations et des éléments multimédias.

N'hésitez pas à utiliser les notions vues ici, en effet, les Storyboard et VisualStateManager sont des outils puissants et permettent de créer de grandes quantités d'interfaces pourvus d'effets impressionnants.

Néanmoins, nous n'avons pas vu en profondeur toutes les notions qui peuvent être utiles à savoir dans vos développements. Si vous voulez en savoir plus sur un point de ce cours, nous vous invitons à consulter MSDN.