Templates avancés en Silverlight

par Benjamin Roux (Retour aux articles)

Date de publication : 05/05/2008

Dernière mise à jour : 17/06/2008

Dans cet article nous allons voir une utilisation avancée des templates en Silverlight.

1 - Template de contrôle avancé

1-1 - Introduction

Silverlight 2 fait apparaître avec lui de nombreuses nouveautés (introduites ici : **Introduction à Silverlight 2**) et avec ces dernières la notion de **Template** pour les contrôles.

Mais comment personnaliser un contrôle de A à Z ? C'est ce que nous allons voir immédiatement.

Depuis la beta 2 de Silverlight, la personnalisation des contrôles a été simplifiée grâce à Expression Blend, cet article s'adresse donc aux développeurs, qui, comme moi qui ne savent utiliser qu'une seule application de graphisme : Paint.

1-2 - Template de bouton

Pour illustrer ce concept, nous allons personnaliser un contrôle de type bouton.

Pour tout ce qui est réalisation de template, votre meilleure amie est sans nul doute la MSDN. En effet vous trouverez sur cette dernière le nom des différents états (par exemple MouseOver pour un bouton), ainsi que des exemples.
Voici le lien répertoriant les différentes pages qui sont utiles : **http://msdn2.microsoft.com/en-us/library/cc278075(VS.95).aspx**

Pour notre exemple de bouton le lien est celui-ci : **http://msdn2.microsoft.com/en-us/library/cc278069(VS.95).aspx**

On apprend le nom des différents états d'un bouton ainsi que le groupe auxquels ils appartiennent.

Etat	Groupe	Description
Normal	CommonStates	L'état normal du bouton
MouseOver	CommonStates	L'état du bouton lorsque l'utilisateur passe la souris dessus
Pressed	CommonStates	L'état du bouton que l'utilisation clique dessus
Disabled	CommonStates	L'état du bouton lorsqu'il est désactivé
Focused	FocusStates	L'état du bouton lorsqu'il prend le focus
Unfocused	FocusStates	L'état du bouton lorsqu'il perd le focus

Nous avons tout ce dont nous avons besoin pour réaliser le template de notre contrôle.

Pour cet article, nous allons rendre notre contrôle semblable visuellement aux boutons présents dans la suite Office 2007 (bleu et orange).

1-2-1 - Le visuel

Nous allons commencer par le visuel de notre bouton. Notre élément racine se nommera *RootElement*.

```
<Grid x:Name="RootElement">

</Grid>
```

Pour savoir où placer ce code, vous pouvez regarder ici : **http://broux.developpez.com/articles/csharp/introduction-silverlight-2/#L9**

Maintenant nous allons mettre un peu de couleur, à savoir une bordure bleue et un background en bleu dégradé.

```
<Grid x:Name="RootElement">

  <Border x:Name="VisualElement" Width="{TemplateBinding Width}" Height="{TemplateBinding Height}" CornerRadius="
        <Border.BorderBrush>
            <SolidColorBrush Color="#FF9BB7E0"/>
        </Border.BorderBrush>
        <Border.Background>
            <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                <GradientStop x:Name="Color1" Color="#FFC8DBEE" Offset="0.260"/>
                <GradientStop x:Name="Color2" Color="#FF84AFE6" Offset="0.530"/>
                <GradientStop x:Name="Color3" Color="#FFC8DBEE" Offset="0.80"/>
            </LinearGradientBrush>
        </Border.Background>
    </Border>
</Grid>
```

Voici le résultat de notre bouton avec ce code pour l'instant

```
<Button Width="155" Height="50" Content="Bouton Style Word 2007"
        Style="{StaticResource ButtonRectangle}" VerticalAlignment="Center"
        FontFamily="Verdana" FontSize="12" />
```

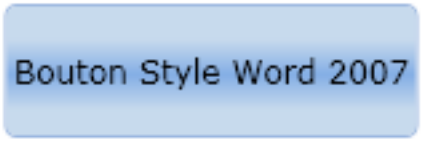
On voit que le contenu de notre bouton (ici du texte) n'est pas visible.

Nous allons simplement rajouter un objet de type **ContentPresenter**, dont le contenu sera bindé sur le contenu de notre bouton, autrement dit, tout ce qui sera dans la propriété **Content** (du texte, un contrôle...) de notre bouton sera représenté dans notre **ContentPresenter**.

```
<Grid x:Name="RootElement">

  <Border x:Name="VisualElement" Width="{TemplateBinding Width}" Height="{TemplateBinding Height}" CornerRadius="
        <Border.BorderBrush>
            <SolidColorBrush Color="#FF9BB7E0"/>
        </Border.BorderBrush>
        <Border.Background>
            <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                <GradientStop x:Name="Color1" Color="#FFC8DBEE" Offset="0.260"/>
                <GradientStop x:Name="Color2" Color="#FF84AFE6" Offset="0.530"/>
                <GradientStop x:Name="Color3" Color="#FFC8DBEE" Offset="0.80"/>
            </LinearGradientBrush>
        </Border.Background>

  <ContentPresenter Content="{TemplateBinding Content}"
```

```
                        ContentTemplate="{TemplateBinding ContentTemplate}"
                        FontFamily="{TemplateBinding FontFamily}"
                        FontSize="{TemplateBinding FontSize}"
                        FontStretch="{TemplateBinding FontStretch}"
                        FontStyle="{TemplateBinding FontStyle}"
                        FontWeight="{TemplateBinding FontWeight}"
                        Foreground="Black"
                        Padding="{TemplateBinding Padding}"
                        TextAlignment="{TemplateBinding TextAlignment}"
                        TextDecorations="{TemplateBinding TextDecorations}"
                        TextWrapping="{TemplateBinding TextWrapping}"
                        VerticalContentAlignment="Center"
                        HorizontalContentAlignment="Center"
                        Margin="0" />
    </Border>
</Grid>
```

Et maintenant voici le résultat avec le même code que précédemment.



Il reste un problème : quand on passe la souris dessus ou même que l'on clique dessus il ne se passe rien du tout.

Pour ça nous allons maintenant spécifier ses comportements.

1-2-2 - Les animations

Le gestion des états avec les templates, se fait grâce au **VisualStateManager**, il nous faut donc l'utiliser.

On commencer par rajouter le namespace

```
xmlns:vsm="clr-namespace:System.Windows;assembly=System.Windows"
```

On rajoute maintenant les balises nécessaires :

```
<Grid x:Name="RootElement">
    <vsm:VisualStateManager.VisualStateGroups>
        <vsm:VisualStateGroup x:Name="CommonStates">
        </vsm:VisualStateGroup>
    </vsm:VisualStateManager.VisualStateGroups>
 [...]
</Grid>
```

*Dans la beta 2 de Silverlight, Visual Studio ne reconnait pas **VisualStateManager.VisualStateGroups**, mais cela fonctionne quand même.*

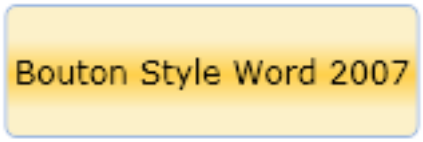
Et maintenant nous allons mettre nos étas uns à uns.

Commençons par l'état **MouseOver**.

```
<Grid x:Name="RootElement">
    <vsm:VisualStateManager.VisualStateGroups>
        <vsm:VisualStateGroup x:Name="CommonStates">
            <vsm:VisualState x:Name="MouseOver">
                <Storyboard>

  <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color1" Storyboard.TargetProperty="Color">
                        <DiscreteColorKeyFrame KeyTime="0:0:0" Value="#FFFDF2CA" />
                    </ColorAnimationUsingKeyFrames>

  <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color2" Storyboard.TargetProperty="Color">
                        <DiscreteColorKeyFrame KeyTime="0:0:0" Value="#FFFFD048" />
                    </ColorAnimationUsingKeyFrames>

  <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color3" Storyboard.TargetProperty="Color">
                        <DiscreteColorKeyFrame KeyTime="0:0:0" Value="#FFFDF2CA" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </vsm:VisualState>
            <vsm:VisualState x:Name="Disabled" />
        </vsm:VisualStateGroup>
    </vsm:VisualStateManager.VisualStateGroups>

</Grid>
```

Comme vous pouvez le voir c'est extrêmement simple, il suffit de déclarer un **VisualState** et d'associer à la propriété *x:Name* le nom de l'état auquel on veut qu'il soit associé. Puis il suffit de mettre un **Storyboard** pour notre animation.

Et voilà, désormais lorsque l'utilisateur passera la souris sur le bouton le dégradé bleu en fond changera en dégradé orange.



Rajoutons ensuite les autres états.

```
<Grid x:Name="RootElement">
    <vsm:VisualStateManager.VisualStateGroups>
        <vsm:VisualStateGroup x:Name="CommonStates">
            <vsm:VisualState x:Name="Normal">
                <Storyboard>

  <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color1" Storyboard.TargetProperty="Color">
                        <LinearColorKeyFrame KeyTime="0:0:0.35" Value="#FFC8DBEE" />
                    </ColorAnimationUsingKeyFrames>

  <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color2" Storyboard.TargetProperty="Color">
                        <LinearColorKeyFrame KeyTime="0:0:0.35" Value="#FF84AFE6" />
                    </ColorAnimationUsingKeyFrames>

  <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color3" Storyboard.TargetProperty="Color">
                        <LinearColorKeyFrame KeyTime="0:0:0.35" Value="#FFC8DBEE" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </vsm:VisualState>
            <vsm:VisualState x:Name="MouseOver">
```

```
                <Storyboard>

  <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color1" Storyboard.TargetProperty="Color">
                        <DiscreteColorKeyFrame KeyTime="0:0:0" Value="#FFFDF2CA" />
                    </ColorAnimationUsingKeyFrames>

  <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color2" Storyboard.TargetProperty="Color">
                        <DiscreteColorKeyFrame KeyTime="0:0:0" Value="#FFFFD048" />
                    </ColorAnimationUsingKeyFrames>

  <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color3" Storyboard.TargetProperty="Color">
                        <DiscreteColorKeyFrame KeyTime="0:0:0" Value="#FFFDF2CA" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </vsm:VisualState>
            <vsm:VisualState x:Name="Pressed">
                <Storyboard>

  <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color1" Storyboard.TargetProperty="Color">
                        <DiscreteColorKeyFrame KeyTime="0:0:0" Value="#FFFCB16D" />
                    </ColorAnimationUsingKeyFrames>

  <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color2" Storyboard.TargetProperty="Color">
                        <DiscreteColorKeyFrame KeyTime="0:0:0" Value="#FFFF8D06" />
                    </ColorAnimationUsingKeyFrames>

  <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color3" Storyboard.TargetProperty="Color">
                        <DiscreteColorKeyFrame KeyTime="0:0:0" Value="#FFFCB16D" />
                    </ColorAnimationUsingKeyFrames>

  <ColorAnimationUsingKeyFrames Storyboard.TargetName="VisualElement" Storyboard.TargetProperty="(Border.BorderBr
                        <DiscreteColorKeyFrame KeyTime="0:0:0" Value="#FF7B6645" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </vsm:VisualState>
            <vsm:VisualState x:Name="Disabled" />
        </vsm:VisualStateGroup>
    </vsm:VisualStateManager.VisualStateGroups>
</Grid>
```

Lorsque l'utilisateur cliquera sur le bouton la couleur de ce dernier passera en orange foncé (*Discrete* animation), lorsque le bouton repassera dans l'état normal (MouseOut par exemple), la couleur reviendra progressivement (*Linear* animation) au bleu et pour finir je n'ai pas mis d'animation pour l'état désactivé.

Voici donc le code final.

```
<Grid x:Name="RootElement">
    <vsm:VisualStateManager.VisualStateGroups>
        <vsm:VisualStateGroup x:Name="CommonStates">
            <vsm:VisualState x:Name="Normal">
                <Storyboard>

  <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color1" Storyboard.TargetProperty="Color">
                        <LinearColorKeyFrame KeyTime="0:0:0.35" Value="#FFC8DBEE" />
                    </ColorAnimationUsingKeyFrames>

  <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color2" Storyboard.TargetProperty="Color">
```

```
                        <LinearColorKeyFrame KeyTime="0:0:0.35" Value="#FF84AFE6" />
                    </ColorAnimationUsingKeyFrames>

    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color3" Storyboard.TargetProperty="Color">
                        <LinearColorKeyFrame KeyTime="0:0:0.35" Value="#FFC8DBEE" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </vsm:VisualState>
            <vsm:VisualState x:Name="MouseOver">
                <Storyboard>

    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color1" Storyboard.TargetProperty="Color">
                        <DiscreteColorKeyFrame KeyTime="0:0:0" Value="#FFFDF2CA" />
                    </ColorAnimationUsingKeyFrames>

    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color2" Storyboard.TargetProperty="Color">
                        <DiscreteColorKeyFrame KeyTime="0:0:0" Value="#FFFFD048" />
                    </ColorAnimationUsingKeyFrames>

    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color3" Storyboard.TargetProperty="Color">
                        <DiscreteColorKeyFrame KeyTime="0:0:0" Value="#FFFDF2CA" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </vsm:VisualState>
            <vsm:VisualState x:Name="Pressed">
                <Storyboard>

    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color1" Storyboard.TargetProperty="Color">
                        <DiscreteColorKeyFrame KeyTime="0:0:0" Value="#FFFCB16D" />
                    </ColorAnimationUsingKeyFrames>

    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color2" Storyboard.TargetProperty="Color">
                        <DiscreteColorKeyFrame KeyTime="0:0:0" Value="#FFFF8D06" />
                    </ColorAnimationUsingKeyFrames>

    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Color3" Storyboard.TargetProperty="Color">
                        <DiscreteColorKeyFrame KeyTime="0:0:0" Value="#FFFCB16D" />
                    </ColorAnimationUsingKeyFrames>

    <ColorAnimationUsingKeyFrames Storyboard.TargetName="VisualElement" Storyboard.TargetProperty="(Border.BorderBr
                        <DiscreteColorKeyFrame KeyTime="0:0:0" Value="#FF7B6645" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </vsm:VisualState>
            <vsm:VisualState x:Name="Disabled" />
        </vsm:VisualStateGroup>
      </vsm:VisualStateManager.VisualStateGroups>

    <Border x:Name="VisualElement" Width="{TemplateBinding Width}" Height="{TemplateBinding Height}" CornerRadius="
          <Border.BorderBrush>
              <SolidColorBrush Color="#FF9BB7E0"/>
          </Border.BorderBrush>
          <Border.Background>
              <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                  <GradientStop x:Name="Color1" Color="#FFC8DBEE" Offset="0.260"/>
                  <GradientStop x:Name="Color2" Color="#FF84AFE6" Offset="0.530"/>
                  <GradientStop x:Name="Color3" Color="#FFC8DBEE" Offset="0.80"/>
              </LinearGradientBrush>
          </Border.Background>

    <ContentPresenter Content="{TemplateBinding Content}"
                          ContentTemplate="{TemplateBinding ContentTemplate}"
                          FontFamily="{TemplateBinding FontFamily}"
                          FontSize="{TemplateBinding FontSize}"
                          FontStretch="{TemplateBinding FontStretch}"
                          FontStyle="{TemplateBinding FontStyle}"
                          FontWeight="{TemplateBinding FontWeight}"
                          Foreground="Black"
                          Padding="{TemplateBinding Padding}"
                          TextAlignment="{TemplateBinding TextAlignment}"
                          TextDecorations="{TemplateBinding TextDecorations}"
                          TextWrapping="{TemplateBinding TextWrapping}"
```

```
                        VerticalContentAlignment="Center"
                        HorizontalContentAlignment="Center"
                        Margin="0" />
    </Border>
</Grid>
```

Vous pouvez aussi voir que je n'ai pas utilisé l'élément pour indiquer le focus sur le bouton (en général une bordure en pointillés).

Voici enfin un lien vers un test online : **Test Online**

1-2-3 - Conclusion

Comme vous venez de le constater, réaliser un template pour un contrôle n'a rien de compliqué, il vous faut simplement quelques talents de designers, et connaitre le nom des différents éléments, rien de plus.

Nous allons maintenant passer à la réalisation d'un contrôle templatisable.

2 - Création d'un contrôle templatisable

Dans cette partie, nous allons créer un contrôle templatisable. Ce contrôle sera un panel extensible à la manière de ceux que l'on trouve sur Developpez.

Pour la création de ce contrôle, je me suis inspiré du contrôle présenté dans **l'excellent article** de Shawn Burk.

2-1 - La classe

Tout d'abord nous allons créer notre classe **ExpanderPanel**. Pour que ce soit un contrôle, nous allons la faire hériter de **ContentControl**, cette classe définit un contrôle contenant une propriété **Content** qui contiendra le contenu de notre contrôle.

Notre **ExpanderPanel** aura une propriété **Content** qui contiendra le texte principal.

```
public class ExpanderPanel : ContentControl
{

}
```

Notre contrôle utilisant les templates, il faut rajouter une ligne dans le constructeur.

```
public ExpanderPanel() : base()
{
    DefaultStyleKey = typeof(ExpanderPanel);
}
```

2-2 - Eléments templatisables

Nous allons maintenant ajouter nos élément templatisables.

Notre contrôle aura un élément principal (*RootElement*) de type **FrameworkElement**, un élément de header (*HeaderElement*) de type **ContentControl**, un élément qui contiendra notre contenu principal (*ContentElement*) de type **ContentControl**, un élément cliquable qui permettra de dérouler ou d'enrouler le contenu principal (*ButtonElement*) de type **ToggleButton**, puis deux états (*Opened* et *Closes*).

```
[TemplatePart(Name = ExpanderPanel.RootElement, Type = typeof(FrameworkElement))]
[TemplatePart(Name = ExpanderPanel.HeaderElement, Type = typeof(ContentControl))]
[TemplatePart(Name = ExpanderPanel.ContentElement, Type = typeof(ContentControl))]
[TemplatePart(Name = ExpanderPanel.ButtonElement, Type = typeof(ToggleButton))]
[TemplateVisualStateAttribute(Name = ExpanderPanel.Opened, GroupName = "CommonStates")]
[TemplateVisualStateAttribute(Name = ExpanderPanel.Closed, GroupName = "CommonStates")]
public class ExpanderPanel : ContentControl
{
```

```
    private const string RootElement = "RootElement";
    private const string HeaderElement = "HeaderElement";
    private const string ContentElement = "ContentElement";
    private const string ButtonElement = "ButtonElement";
    private const string Opened = "Opened";
    private const string Closed = "Closed";
}
```

Une fois le contrôle chargé, son interface est remplie à partir de son template. Pour ce faire, la méthode **OnApplyTemplate** est appelée. Nous allons mettre le code permettant de charger nos éléments via le template générique ou celui défini par l'utilisateur.

Pour chaque partie de notre template nous allons la stocker dans une variable.

```
[TemplatePart(Name = ExpanderPanel.RootElement, Type = typeof(FrameworkElement))]
[TemplatePart(Name = ExpanderPanel.HeaderElement, Type = typeof(ContentControl))]
[TemplatePart(Name = ExpanderPanel.ContentElement, Type = typeof(ContentControl))]
[TemplatePart(Name = ExpanderPanel.ButtonElement, Type = typeof(ToggleButton))]
[TemplateVisualStateAttribute(Name = ExpanderPanel.Opened, GroupName = "CommonStates")]
[TemplateVisualStateAttribute(Name = ExpanderPanel.Closed, GroupName = "CommonStates")]
public class ExpanderPanel : ContentControl
{
    private const string RootElement = "RootElement";
    private const string HeaderElement = "HeaderElement";
    private const string ContentElement = "ContentElement";
    private const string ButtonElement = "ButtonElement";
    private const string Opened = "Opened";
    private const string Closed = "Closed";

    FrameworkElement mRootElement = null;
    ToggleButton mButtonElement = null;

    public override void OnApplyTemplate()
    {
        base.OnApplyTemplate();

        mRootElement = this.GetTemplateChild(ExpanderPanel.RootElement) as FrameworkElement;

        if (mRootElement != null)
        {
            mButtonElement = this.GetTemplateChild(ExpanderPanel.ButtonElement) as ToggleButton;
        }

        ChangeState();
    }
}
```

2-3 - Propriétés de binding

Comme déjà expliqué notre contrôle contiendra deux parties : le header et le contenu principal.

Notre contrôle héritant de **ContentControl**, il possède déjà une propriété **Content**. Nous allons créer une propriété bindable.

Pour ça, Visual Studio nous aide beaucoup grâce à un snippet : *propdp*.

```
// Using a DependencyProperty as the backing store for HeaderContent.  This enables animation, styling, binding,
public static readonly DependencyProperty HeaderContentProperty =
    DependencyProperty.Register("HeaderContent", typeof(object), typeof(ExpanderPanel), null);

public object HeaderContent
{
    get { return (object)GetValue(HeaderContentProperty); }
```

```
    set { SetValue(HeaderContentProperty, value); }
}
```

Nous allons également rajouter une propriété **IsOpened**, permettant de spécifier et de modifier l'état du contrôle.

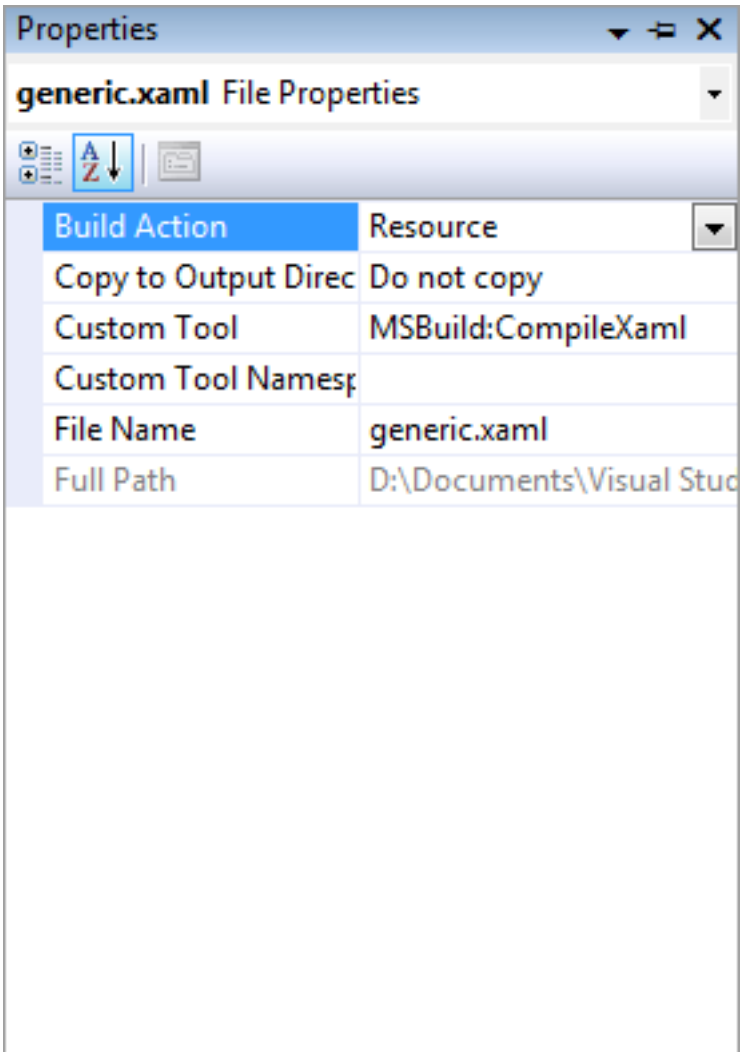
```
// Using a DependencyProperty as the backing store for IsOpened.  This enables animation, styling, binding, etc..
public static readonly DependencyProperty IsOpenedProperty =
    DependencyProperty.Register("IsOpened", typeof(bool), typeof(ExpanderPanel), new
 PropertyMetadata(new PropertyChangedCallback(NotifyChangedState)));

public bool IsOpened
{
    get
    {
        return (bool)GetValue(IsOpenedProperty);
    }
    set
    {
        SetValue(IsOpenedProperty, value);
    }
}

private static void NotifyChangedState(DependencyObject sender, DependencyPropertyChangedEventArgs
 args)
{
    ((ExpanderPanel)sender).IsOpened = (bool)args.NewValue;
}
```

Nous avons juste créé la propriété, le visuel de notre contrôle ne change pas. Nous modifierons le code par la suite.

2-4 - Template de test

Nous allons maintenant créer un template de test pour tester notre contrôle sans pour autant perdre du temps sur le design.

Pour stocker le design, nous allons créer un fichier generic.xaml, qui contiendra notre template. Ce fichier devra être spécifié en tant que Resource dans ses propriétés.



A l'exécution, le runtime regarde dans ce fichier pour récupérer le template par défaut pour afficher le contrôle.

Ce fichier aura plus ou moins cette tête.

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:my="clr-namespace:SilverlightApplication1;assembly=SilverlightApplication1">

    <Style TargetType="my:ExpanderPanel">
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="my:ExpanderPanel">
                 <!-- xml here -->
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>

</ResourceDictionary>
```

Notre template de test ressemblera à ça :

Voici le code du template.

```
<Grid x:Name="RootElement">
    <Grid.RowDefinitions>
        <RowDefinition Height="25"/>
        <RowDefinition Height="auto"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="auto" />
        <ColumnDefinition Width="25" />
    </Grid.ColumnDefinitions>
    <ContentControl Grid.Column="0" Grid.Row="0"
                    x:Name="HeaderElement" HorizontalAlignment="Left"
                    VerticalAlignment="Center"
                    Content="{TemplateBinding HeaderContent}"/>


  <ToggleButton Grid.Row="0" Grid.Column="1" HorizontalAlignment="Center" x:Name="ButtonElement" Content="X" />

  <ContentControl Grid.Row="1" Grid.ColumnSpan="2" x:Name="ContentElement" Content="{TemplateBinding Content}" />
</Grid>
```

Nous allons maintenant mettre le XAML dans notre *Page.xaml* pour afficher notre contrôle.

```
<my:ExpanderPanel x:Name="PanelExp" HeaderContent="Header">
    <my:ExpanderPanel.Content>
        <Grid Background="Red">
            <TextBlock Text="Content" />
        </Grid>
    </my:ExpanderPanel.Content>
</my:ExpanderPanel>
```

2-5 - Les états (ouvert/fermé)

Nous allons maintenant ajouter du code pour modifier le visuel de notre contrôle selon son état.

Tout d'abord la propriété.

```
public bool IsOpened
{
    get
    {
        if (mButtonElement != null) return mButtonElement.IsChecked ?? false;
        else return (bool)GetValue(IsOpenedProperty);
    }
    set
    {
        if (mButtonElement != null) mButtonElement.IsChecked = value;
        else SetValue(IsOpenedProperty, value);

        ChangeState();
    }
}s
```

Lorsque la propriété sera modifiée, nous changerons l'état de notre **ToggleButton** si ce dernier existe, sinon nous modifierons simplement la **DependencyProperty**.

Et lorsque nous voudrons récupérer la valeur, nous récupèrerons la valeur de l'état du **ToggleButton** s'il existe, sinon la valeur de la **DependencyProperty**.

Une fois fait, nous changerons l'état de notre contrôle au niveau visuel.

```
private void ChangeState()
{
    if (IsOpened) VisualStateManager.GoToState(this, ExpanderPanel.Opened, true);
    else VisualStateManager.GoToState(this, ExpanderPanel.Closed, true);
}
```

Selon la valeur de **IsOpened**, on va à tel ou tel état.

Ajoutons maintenant le code pour le clic sur le bouton (qui changera l'état de notre contrôle).

```
public override void OnApplyTemplate()
{
    base.OnApplyTemplate();

    mRootElement = this.GetTemplateChild(ExpanderPanel.RootElement) as FrameworkElement;

    if (mRootElement != null)
    {
        mButtonElement = this.GetTemplateChild(ExpanderPanel.ButtonElement) as ToggleButton;

        if (mButtonElement != null)
        {
            mButtonElement.IsChecked = IsOpened;
            mButtonElement.Click += new RoutedEventHandler(mButtonElement_Click);
        }
    }

    ChangeState();
}

void mButtonElement_Click(object sender, RoutedEventArgs e)
{
    IsOpened = mButtonElement.IsChecked.Value;
}
```

A l'application du template, si le bouton existe, on règle son état sur l'état de notre contrôle (ouvert/fermé), puis on s'abonne à l'évènement du clic.

Lorsque le bouton est cliqué, on modifie simplement l'état de notre contrôle par rapport à l'état du bouton (qui vient de changer).

2-6 - Template final

Nous allons maintenant rendre le design un peu plus agréable.

Voici le code final.

```
<Grid x:Name="RootElement">
    <vsm:VisualStateManager.VisualStateGroups>
        <vsm:VisualStateGroup x:Name="CommonStates">
            <vsm:VisualState x:Name="Opened">
                <Storyboard>

  <DoubleAnimationUsingKeyFrames Storyboard.TargetName="ContentParent" Storyboard.TargetProperty="(UIElement.Rend
                      <SplineDoubleKeyFrame KeyTime="0:0:0.5" Value="1.0"/>
```

```
                    </DoubleAnimationUsingKeyFrames>
                </Storyboard>
            </vsm:VisualState>
            <vsm:VisualState x:Name="Closed">
                <Storyboard>

<DoubleAnimationUsingKeyFrames Storyboard.TargetName="ContentParent" Storyboard.TargetProperty="(UIElement.Rend
                        <SplineDoubleKeyFrame KeyTime="0:0:0.5" Value="0"/>
                    </DoubleAnimationUsingKeyFrames>
                </Storyboard>
            </vsm:VisualState>
        </vsm:VisualStateGroup>
    </vsm:VisualStateManager.VisualStateGroups>
    <Grid.RowDefinitions>
        <RowDefinition Height="25"/>
        <RowDefinition Height="auto"/>
    </Grid.RowDefinitions>

    <Grid Grid.Row="0" Background="#FF607DAF">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="25"/>
        </Grid.ColumnDefinitions>
        <ContentControl x:Name="HeaderElement" HorizontalAlignment="Left"
                        Margin="5,0,0,0"
                        VerticalAlignment="Center"
                        Foreground="White" FontFamily="Verdana"
                        FontSize="13" FontWeight="Bold"
                        Content="{TemplateBinding HeaderContent}"/>

        <ToggleButton Grid.Column="1" HorizontalAlignment="Center" x:Name="ButtonElement">
            <ToggleButton.Template>
                <ControlTemplate TargetType="ToggleButton">
                    <Grid x:Name="RootElement">
                        <vsm:VisualStateManager.VisualStateGroups>
                            <vsm:VisualStateGroup x:Name="CheckStates">
                                <vsm:VisualState x:Name="Checked">
                                    <Storyboard>
                                        <DoubleAnimation Storyboard.TargetName="BorderButton"
                                                         Storyboard.TargetProperty="(Border.RenderTransform).Angl
                                                         From="0" To="180" Duration="0:0:0.5" />
                                    </Storyboard>
                                </vsm:VisualState>
                                <vsm:VisualState x:Name="Unchecked">
                                    <Storyboard>
                                        <DoubleAnimation Storyboard.TargetName="BorderButton"
                                                         Storyboard.TargetProperty="(Border.RenderTransform).Angl
                                                         From="180" To="360" Duration="0:0:0.5" />
                                    </Storyboard>
                                </vsm:VisualState>
                            </vsm:VisualStateGroup>
                        </vsm:VisualStateManager.VisualStateGroups>
                        <Border x:Name="BorderButton" Width="15" Height="15" Background="#FF314977"
                                BorderThickness="1" BorderBrush="White" CornerRadius="15">
                            <Border.RenderTransform>
                                <RotateTransform CenterX="7.5" CenterY="7.5" Angle="0" />
                            </Border.RenderTransform>
                            <Canvas>

<Line X1="2" Y1="6" X2="6.5" Y2="2" Stroke="White" StrokeThickness="1" />

<Line X1="6.5" Y1="2" X2="11" Y2="6" Stroke="White" StrokeThickness="1" />

<Line X1="2" Y1="10" X2="6.5" Y2="6" Stroke="White" StrokeThickness="1" />

<Line X1="6.5" Y1="6" X2="11" Y2="10" Stroke="White" StrokeThickness="1" />
                            </Canvas>
                        </Border>
                    </Grid>
                </ControlTemplate>
            </ToggleButton.Template>
        </ToggleButton>
```

```
    </Grid>
    <Grid x:Name="ContentParent" Grid.Row="1" Background="#FFF5F5FF">
        <Grid.RenderTransform>
            <ScaleTransform />
        </Grid.RenderTransform>

  <ContentControl x:Name="ContentElement" Content="{TemplateBinding Content}" HorizontalAlignment="Center">
            <ContentControl.RenderTransform>
                <ScaleTransform ScaleY="1" ScaleX="1" />
            </ContentControl.RenderTransform>
        </ContentControl>
    </Grid>
</Grid>
```

Et voici le code à mettre dans le *Page.xaml*.

```
<my:ExpanderPanel x:Name="PanelExp" HeaderContent="Nouvelles discussions suivies: (0)" IsOpened="true">
    <my:ExpanderPanel.Content>
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="auto"/>
                <RowDefinition Height="auto"/>
            </Grid.RowDefinitions>

  <TextBlock Text="Il n'y a aucune discussion suivie Ã  afficher dans ce dossier pour cette pÃ©riode de temp
                       FontFamily="Verdana" FontSize="13" FontWeight="Bold" Margin="30,10,30,10" />
            <Grid Grid.Row="1" Background="#FF3E5C92">

  <HyperlinkButton Grid.Row="1" Content="Voir toutes les discussions suivies" NavigateUri="http://
broux.developpez.com"
                                HorizontalAlignment="Right" Margin="0,5,5,5" TextDecorations="Underline"
                                FontFamily="Verdana" FontSize="11" Foreground="#FFAEE0D5"/>
            </Grid>

        </Grid>
    </my:ExpanderPanel.Content>
</my:ExpanderPanel>
```

Notre contrôle ressemble désormais à ca.

2-7 - Test online

Et voici maintenant notre contrôle **testable en ligne**.

2-8 - Conclusion

Comme vous pouvez le voir, la création de contrôles templatisables est relativement simple et ne nécessite pas des connaissances approfondies sur Silverlight ou WPF en général.

3 - Conclusion

Et ainsi se termine cet article sur l'utilisation avancée des templates en Silverlight. Nous avons vu comment templatiser les contrôles déjà existants (bouton, textbox...) mais aussi comment créer nos propres contrôles templatisables afin de les rendre customisables à volonté.

4 - Remerciements

Je tiens à remercier toute l'équipe .NET, ainsi que **Romain Valeri** pour sa correction orthographique.