Présentation du Silverlight Toolkit - Partie 1

par Benjamin Roux (Retour aux articles)

Date de publication : 23/02/2009

Dernière mise à jour :

La première partie de cet article, destiné au Silverlight Toolkit vous présentera toutes les nouveautés au niveau contrôles.

Copyright 2009 - Benjamin Roux. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérets. Droits de diffusion permanents accordés à developpez LLC.

1 - Silverlight Toolkit

1-1 - Présentation

Le **Silverlight Toolkit** est un ensemble de composants Silverlight, développé par une communauté de passionnés.

Ce *Toolkit* met à notre disposition un certain nombre d'éléments non-présents en natif dans Silverlight (des contrôles, des systèmes de positionnement par exemple).

Dans la première partie de cet article je vais vous présenter les nouveaux contrôles, ainsi que le système de thème qui a été créé. Deux autres articles suivront, l'un sur les systèmes de positionnement, l'autre sur le système de graphique présent de le *Toolkit*.

Ce *Toolkit* évolue constamment, la version au moment où j'écris cet article est la release de Décembre 2008.

1-2 - Installation

Il faut tout d'abord télécharger le *toolkit* à cette **adresse**.

Le .zip contient une liste de dll, chacune contenant différentes choses. Par exemple, afin d'utiliser nos nouveaux contrôles, il faut référencer la dll *Microsoft.Windows.Controls.dll*.

1-3 - Préparation de l'article

Tout au long de cet article, nous allons utiliser 2 classes, **Town** et **Neighborhood**, afin de réaliser divers scénarios selon les contrôles.

Neighborhood.cs
```
public class Neighborhood
{
    public string Name { get; set; }
    public int Rate { get; set; }
}
```

Town.cs
```
public class Town
{
    public string Name { get; set; }
    public int Rate { get; set; }
    public List<Neighborhood> Neighborhoods { get; set; }

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

```
List<Town> towns = new List<Town>
{
    new Town
    {
        Name = "San Francisco",
        Rate = 5,
        Neighborhoods = new List<Neighborhood>
        {
            new Neighborhood { Name = "Union Square", Rate = 5 },
            new Neighborhood { Name = "Pacific Heights", Rate = 5 },
            new Neighborhood { Name = "Mission District", Rate = -1 },
            new Neighborhood { Name = "North Beach", Rate = 4 },
```


Copyright 2009 - Benjamin Roux. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérets. Droits de diffusion permanents accordés à developpez LLC.
http://broux.developpez.com/articles/csharp/silverlight-toolkit-partie-1

```
            new Neighborhood { Name = "Tenderloin", Rate = 1 }
        }
    },
    new Town
    {
        Name = "New York",
        Rate = -1,
        Neighborhoods = new List<Neighborhood>
        {
            new Neighborhood { Name = "Harlem" },
            new Neighborhood { Name = "Liberty Island" },
            new Neighborhood { Name = "Civic Center" },
            new Neighborhood { Name = "Queens" },
            new Neighborhood { Name = "Manhattan" }
        }
    },
    new Town { Name = "Berkeley", Rate = 4 },
    new Town { Name = "Oakland", Rate = -1 },
    new Town { Name = "San Jose", Rate = -1 },
    new Town { Name = "Seattle", Rate = -1 }
};
```

Nous allons également utiliser un *Converter* pour nos scénarios de DataBinding.

Rate2ImageConverter.cs

```
public class Rate2ImageConverter : IValueConverter
{

    #region IValueConverter Members

    public object Convert(object value, Type targetType, object parameter,
 System.Globalization.CultureInfo culture)
    {
        if (value is int)
        {
            int rate = (int)value;
            if (rate == -1) return null;
            else return new BitmapImage(new Uri(string.Format("Images/star{0}.png", rate),
 UriKind.Relative));
        }
        return null;
    }

    public object ConvertBack(object value, Type targetType, object parameter,
 System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }

    #endregion
}
```

Ce *Converter* permet de convertir un int en **BitmapImage**. On s'en servira pour afficher une image selon la notation (**Rate**) d'une ville ou d'un quartier.

Copyright 2009 - Benjamin Roux. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérets. Droits de diffusion permanents accordés à developpez LLC.

2 - Les contrôles

2-1 - TreeView

2-1-1 - Les bases

Première nouveauté dans ce pack de contrôles : le TreeView.

Pour arriver à ce résultat quelques lignes de XAML suffisent :

```
<control:TreeView x:Name="tvTowns">
    <control:TreeViewItem Header="San Francisco">
        <control:TreeViewItem Header="Union Square" />
        <control:TreeViewItem Header="Pacific Heights" />
        <control:TreeViewItem Header="Mission District" />
        <control:TreeViewItem Header="North Beach" />
        <control:TreeViewItem Header="Tenderloin" />
    </control:TreeViewItem>
    <control:TreeViewItem Header="New-York">
        <control:TreeViewItem Header="Harlem" />
        <control:TreeViewItem Header="Liberty Island" />
        <control:TreeViewItem Header="Civic Center" />
        <control:TreeViewItem Header="Queens" />
        <control:TreeViewItem Header="Manhattan" />
    </control:TreeViewItem>
</control:TreeView>
```

Il s'agit ni plus ni moins d'une suite de **TreeViewItem** imbriqués.

On peut également arriver à ce résultat en code-behind.

```
TreeViewItem sf = new TreeViewItem();
sf.Header = "San Francisco";
sf.Items.Add("Union Square");
sf.Items.Add("Pacific Heights");
sf.Items.Add("Mission District");
sf.Items.Add("North Beach");
sf.Items.Add("Tenderloin");

TreeViewItem ny = new TreeViewItem();
ny.Header = "New-York";
ny.Items.Add("Harlem");
```


Copyright 2009 - Benjamin Roux. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérets. Droits de diffusion permanents accordés à developpez LLC.

```
ny.Items.Add("Liberty Island");
ny.Items.Add("Civic Center");
ny.Items.Add("Queens");
ny.Items.Add("Manhattan");

tvTowns.Items.Add(sf);
tvTowns.Items.Add(ny);
```

Le problème est qu'ici les items sont hard-codés, il serait donc préférable d'avoir un TreeView basé sur des items dynamiques, avec un nombre variable... C'est là que le DataBinding du **TreeView** intervient.

2-1-2 - Le DataBinding

Nous allons utiliser la liste de **Town** précédemment créée dans la partie 1-3.

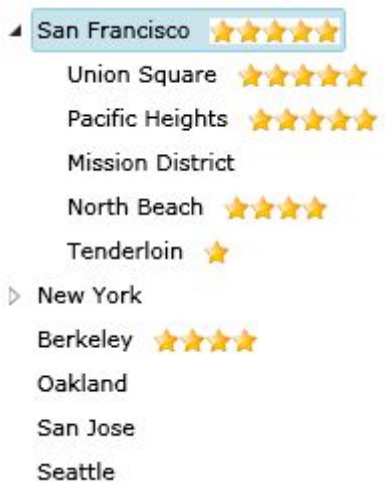
On pourrait tout d'abord penser qu'il faut faire comme pour tous les autres contrôles, mais ici il faut prendre en compte l'imbrication des éléments, et c'est là qu'un nouveau **DataTemplate** intervient : le **HierarchicalDataTemplate**

Ce **DataTemplate** est utilisé pour afficher des éléments sous une forme hiérarchique (ce qui est le cas pour un TreeView). Les éléments à afficher doivent obligatoirement avoir une collection d'enfants (**Neighborhoods** pour le cas de notre classe **Town**).

```
<control:TreeView x:Name="tvTowns">
    <control:TreeView.ItemTemplate>
        <control:HierarchicalDataTemplate ItemsSource="{Binding Neighborhoods}">
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="{Binding Name}" />

  <Image Source="{Binding Rate, Converter={StaticResource Rate2Image}}" Margin="10,0,0,0" />
            </StackPanel>
        </control:HierarchicalDataTemplate>
    </control:TreeView.ItemTemplate>
</control:TreeView>
```

Une nouvelle propriété apparaît dans le **HierarchicalDataTemplate** : **ItemsSource**. Il s'agit ni plus ni moins que du nom de la propriété contenant les éléments fils.

On affecte notre liste à notre TreeView et on obtient le résultat escompté :



Copyright 2009 - Benjamin Roux. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérets. Droits de diffusion permanents accordés à developpez LLC.

2-1-3 - Les problèmes connus

Voici une liste non exhaustive des problèmes connus avec le **TreeView**. Ils seront normalement corrigés dans la prochaine release.

- **DisplayMemberPath** ne fonctionne pas
- On ne peut pas imbriquer les **HierarchicalDataTemplate**.
- Utiliser à la fois **ItemStyle** et **ItemTemplate** empêche la visualisation dans Blend

Vous trouverez sûrement d'autre *Issues* sur le site officiel : **http://www.codeplex.com/Silverlight/WorkItem/List.aspx**

2-2 - AutoCompleteBox

2-2-1 - Les bases

L'**AutoCompleteBox** est une **TextBox** proposant des suggestions selon les entrées utilisateurs.

Ce résultat, assez simple, s'obtient très facilement.

On commence par ajouter notre contrôle dans notre XAML :

```
<control:AutoCompleteBox x:Name="acTowns" />
```

On crée ensuite les suggestions (ici un simple tableau de string) que l'on affecte à notre contrôle.

```
acTowns.ItemsSource = new string[] { "San Francisco", "New-
York", "Berkeley", "Oakland", "San Jose", "Seattle" };
```

Et on obtient le résultat montré plus haut.

2-2-2 - Le DataBinding

Maintenant, il serait pratique de pouvoir changer la liste des suggestions à chaque fois que l'utilisateur ajoute ou supprime une lettre.

Pour cela nous allons voir comment binder nos suggestions sur un WebService (le service de suggestions de Live.com en l'occurrence).

On modifie légèrement le XAML pour s'abonner à l'évènement **Populating** qui est appelé chaque fois que le contenu de l'**AutoCompleteBox** est modifié.

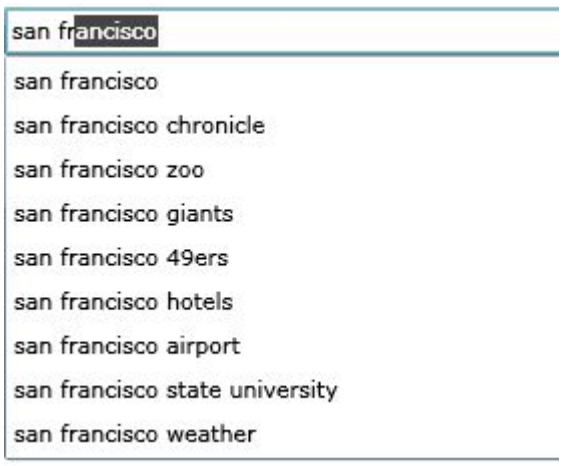
```
<control:AutoCompleteBox x:Name="acSearch" Populating="acSearch_Populating" />
```

```
private void acSearch_Populating(object sender, PopulatingEventArgs e)
{
    AutoCompleteBox box = sender as AutoCompleteBox;
```


Copyright 2009 - Benjamin Roux. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérets. Droits de diffusion permanents accordés à developpez LLC.
http://broux.developpez.com/articles/csharp/silverlight-toolkit-partie-1

```
    WebClient client = new WebClient();
    client.DownloadStringCompleted += new
 DownloadStringCompletedEventHandler(client_DownloadStringCompleted);
    // on interroge le service
    client.DownloadStringAsync(new Uri("http://api.search.live.net/qson.aspx?query=" + box.Text));
}

void client_DownloadStringCompleted(object sender, DownloadStringCompletedEventArgs e)
{
    if (e.Error == null)
    {
        try
        {
            List<string> suggests = new List<string>();

   // on parse le résultat qui est en Json
            JsonObject jso = ((JsonObject)JsonObject.Parse(e.Result))["SearchSuggestion"] as
 JsonObject;

   // on ajoute nos suggestions dans une liste
            foreach (JsonObject suggest in (JsonArray)jso["Section"])
            {
                suggests.Add(suggest.Values.First());
            }

   // on modifie les suggestions de notre contrôle
            acSearch.ItemsSource = suggests;
            acSearch.PopulateComplete();
        }
        catch { }
    }
}
```

Une fois les suggestions récupérées à partir du service Live, nous les affectons à notre **AutoCompleteBox**, et nous nous retrouvons avec ce résultat.



2-2-3 - Template

Comme pour beaucoup de contrôles, l'affichage est entièrement personnalisable. Ici nous allons afficher une image selon la notation pour chaque ville contenue dans notre liste.

```
<control:AutoCompleteBox x:Name="acTowns">
    <control:AutoCompleteBox.ItemTemplate>
        <DataTemplate>
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="{Binding Name}" />

  <Image Source="{Binding Rate, Converter={StaticResource Rate2Image}}" Margin="10,0,0,0" />
```

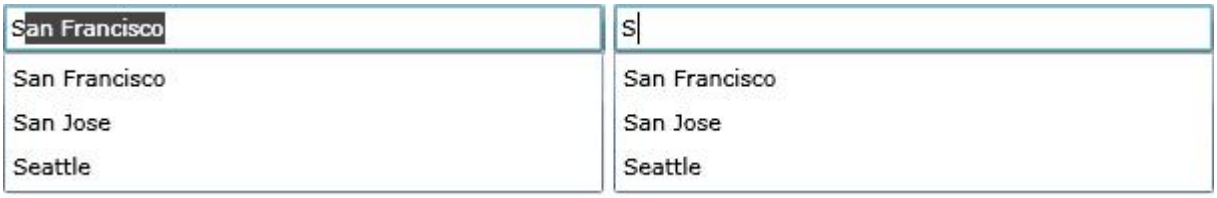

Copyright 2009 - Benjamin Roux. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérets. Droits de diffusion permanents accordés à developpez LLC.

```
            </StackPanel>
        </DataTemplate>
    </control:AutoCompleteBox.ItemTemplate>
</control:AutoCompleteBox>
```

2-2-4 - Utilisation avancée

Ce contrôle contient également un très grand nombre de propriété afin d'affiner l'utilisation de notre contrôle. Voici une liste non exhaustive.

IsTextCompletionEnabled : définit si, quand un résultat est trouvé, le texte est complètement affiché (avec la partie manquante en gras) ou pas.



SelectedItem : l'Item choisi parmi les possibilités

SearchMode : le mode de recherche (StartsWith, Contains, Equals, None ou Custom)

MinimumPrefixLength : la taille minimale du texte à rentrer avant que la recherche s'effectue

TextFilter : un délégué/lamba prenant 2 string : le texte du contrôle et une des possibilités parmis la liste des suggestions (ItemsSource).
Doit retourner un booléen indiquant si la possibilité et une suggestion à afficher.

```
acSearch.TextFilter = (s, s2) => s2.Contains(s);
```

Ce bout de code simule le mode de recherche *Contains*.

ItemFilter : similaire à **TextFilter**, excepté que le délégué/lambda prend en second paramètre un object. A vous de définir votre propre implémentation.

```
acSearch.ItemFilter = delegate(string s, object o)
{
    if (o is Town)
    {
        Town town = o as Town;
        return town.Name.StartsWith(s);
    }
    return false;
};
```

*En cas d'utilisation de **TextFilter** ou **ItemFilter** il faut mettre **SearchMode** à **Custom**.*

Copyright 2009 - Benjamin Roux. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérets. Droits de diffusion permanents accordés à developpez LLC.

Comme vous venez de le voir, ce contrôle est très puissant et permet d'améliorer grandement l'expérience utilisateur.

2-3 - Expander

Un **Expander** est un panel que l'on peut ouvrir ou fermer via un bouton, à la manière de ceux que l'on trouve sur les forums de Developpez.com.

J'ai d'ailleurs écrit un article sur l'utilisation des **templates en Silverlight**, où nous avons créé un contrôle similaire.

```
<control:Expander Header="Nouvelles discussions suivies: (0)"
                  Content="Il n'y a aucune discussion suivie à afficher dans ce dossier pour cette période de tem
```

Un clic sur le bouton fait disparaître le texte contenu dans **Content**.

On peut bien évidemment styler entièrement ce contrôle ou mettre ce que l'on désire dans **Header** et **Content**, nous ne sommes pas limités à du texte.

2-4 - NumericUpDown

Ce contrôle est simplement une *TextBox* avec deux boutons, l'un pour augmenter et l'autre pour diminuer la valeur contenue dans le contrôle.

*Ce contrôle se trouve dans la dll **Microsoft.Windows.Controls.Input.dll**, il faut bien veiller à l'ajouter en référence.*

```
<controlinput:NumericUpDown Maximum="100" Minimum="0" />
```

On peut également définir la valeur de l'incrément, le style des boutons, l'emplacement des boutons par quelques ajouts dans notre XAML...

2-5 - HeaderedContentControl & HeaderedItemsControl

Le *HeaderedContentControl* est le contrôle à la base de tous les contrôles possédant un **Header** et un **Content** (*Expander* par exemple).

```
<control:HeaderedContentControl Header="Les quartiers de San Francisco">
    <control:HeaderedContentControl.Content>
        <ItemsControl x:Name="icSF">
            <ItemsControl.ItemTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding Name}" />
                </DataTemplate>
            </ItemsControl.ItemTemplate>
        </ItemsControl>
    </control:HeaderedContentControl.Content>
</control:HeaderedContentControl>
```


Copyright 2009 - Benjamin Roux. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérets. Droits de diffusion permanents accordés à developpez LLC.
http://broux.developpez.com/articles/csharp/silverlight-toolkit-partie-1

Le *HeaderedItemsControl* est la même chose que le *HeaderedContentControl* à la différence qu'il est basé sur l'affichage de plusieurs éléments. Le résultat du code précédent est donc identique à celui-ci.

```
<control:HeaderedItemsControl x:Name="hicSF" Header="Les quartiers de San Francisco">
    <control:HeaderedItemsControl.ItemTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding Name}" />
        </DataTemplate>
    </control:HeaderedItemsControl.ItemTemplate>
</control:HeaderedItemsControl>
```


Copyright 2009 - Benjamin Roux. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérets. Droits de diffusion permanents accordés à developpez LLC.

3 - Les thèmes

Ce *Toolkit* apporte également avec lui de nouveaux thèmes pour nos contrôles, ainsi qu'un système assez avancé de templatisation.

3-1 - Les nouveaux thèmes

Pour utiliser un des thèmes du contrôle, il suffit d'ajouter une référence sur la dll correspondante (**Microsoft.Windows.Controls.Theming.ExpressionDark.dll** par exemple), de référencer le namespace dans notre XAML.

```
xmlns:theme="clr-
namespace:Microsoft.Windows.Controls.Theming;assembly=Microsoft.Windows.Controls.Theming.ExpressionDark"
```

Puis d'englober tous nos contrôles dans une balise.

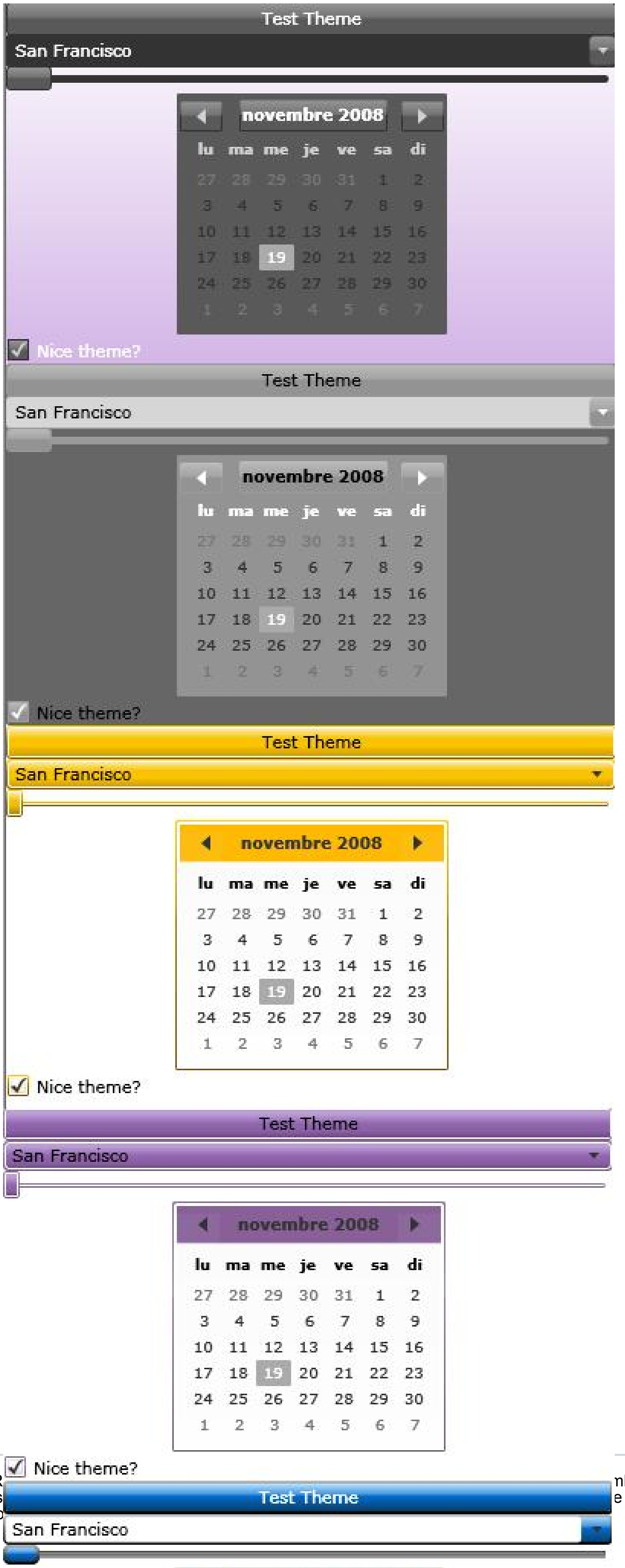
```
<theme:ExpressionDarkTheme>
    <StackPanel Width="400">
        <Button Content="Test Theme" />
        <ComboBox>
            <ComboBoxItem Content="San Francisco" IsSelected="True" />
        </ComboBox>
        <Slider />
        <base:Calendar />
        <CheckBox Content="Nice theme?" IsChecked="true" />
    </StackPanel>
</theme:ExpressionDarkTheme>
```

Le *Toolkit* comporte un actuellement 9 thèmes.

- ExpressionDark
- ExpressionLight
- RainierOrange
- RainierPurple
- ShinyBlue
- ShinyRed
- Whistler Blue
- Bureau Black
- Bureau Blue

Voici les 6 premiers en image.

Copyright 2009 - Benjamin Roux. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérets. Droits de diffusion permanents accordés à developpez LLC.



Copyright 2009 - Benjamin R ... nble de son contenu : textes, documents, images, etc sans ... e prison et jusqu'à 300 000 E de dommages et intérets. Dro

*Lors de l'ajout du namespace pour les thèmes RainierOrange et RainierOrange, une erreur vient se glisser, il faut rajouter le **i** dans Rainier.*

3-2 - Le ImplicitStyleManager

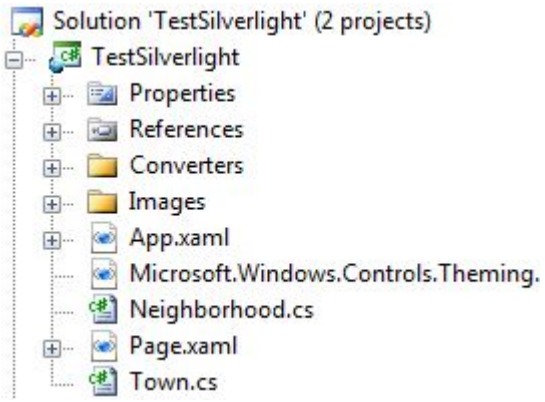
L'*ImplicitStyleManager* est un tout nouveau système de templatisation, permettant de styler un contrôle, plusieurs ou tout simplement les contrôles d'un conteneur.

Pour cela il suffit d'indiquer à notre manager, l'URI du XAML contenant la définition des thèmes.

Réalisons un exemple avec le thème *BureauBlue*. Le XAML se trouve dans le répertoire *Themes\XAML* là où vous avez placé votre *Toolkit*.

Avant toute chose il faut référence la dll **Microsoft.Windows.Controls.Theming.dll** dans votre projet, cette dernière contient tout ce dont nous avons besoin ici.

On commence par ajouter le XAML du thème à notre projet en tant que fichier existant.



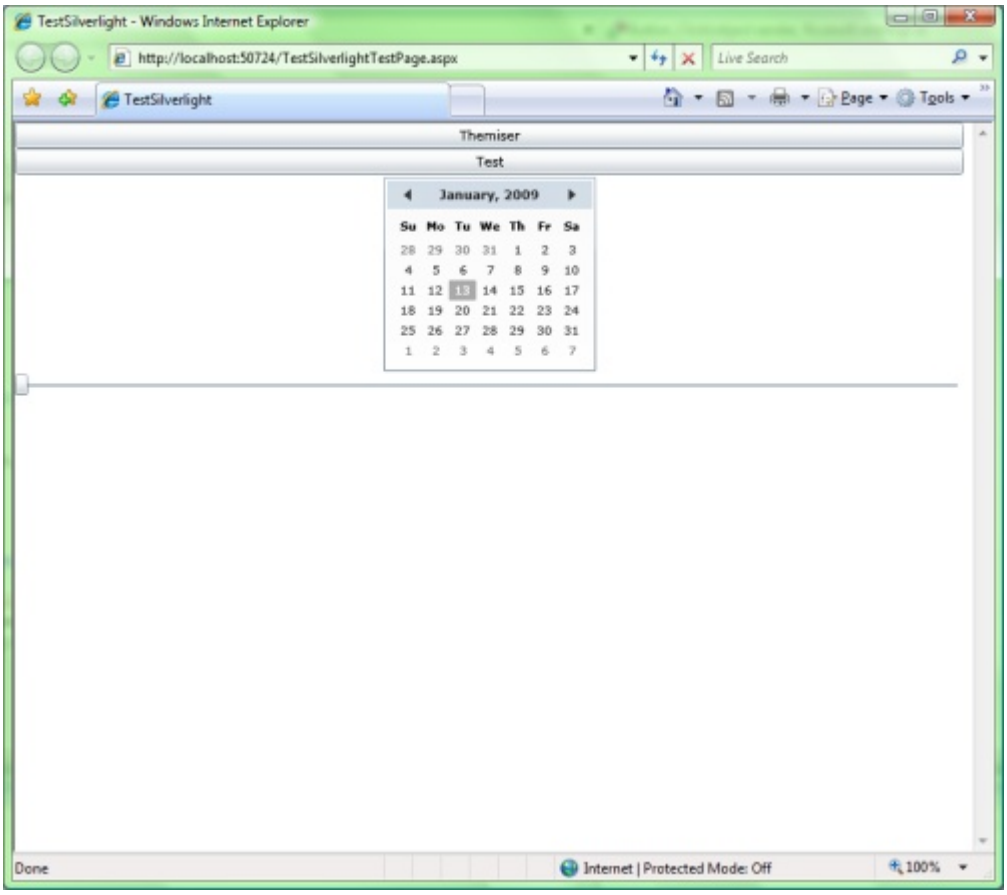
On crée maintenant, un design de test.

```
<UserControl x:Class="TestSilverlight.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:basic="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls"
    Width="800" Height="600">
    <StackPanel x:Name="LayoutRoot" Background="White">
        <Button Content="Themiser" Click="Button_Click" />
        <StackPanel>
            <Button Content="Test" />
            <basic:Calendar />
            <Slider />
        </StackPanel>
    </StackPanel>
</UserControl>
```

Un clic sur le bouton *Themiser* changera à la volée tout le thème de notre application.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    Uri uri = new Uri(@"TestSilverlight;component/Microsoft.Windows.Controls.Theming.BureauBlue.xaml",
 UriKind.Relative);
    ImplicitStyleManager.SetResourceDictionaryUri(LayoutRoot, uri);
    ImplicitStyleManager.SetApplyMode(LayoutRoot, ImplicitStylesApplyMode.Auto);
    ImplicitStyleManager.Apply(LayoutRoot);
}
```


Copyright 2009 - Benjamin Roux. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérets. Droits de diffusion permanents accordés à developpez LLC.



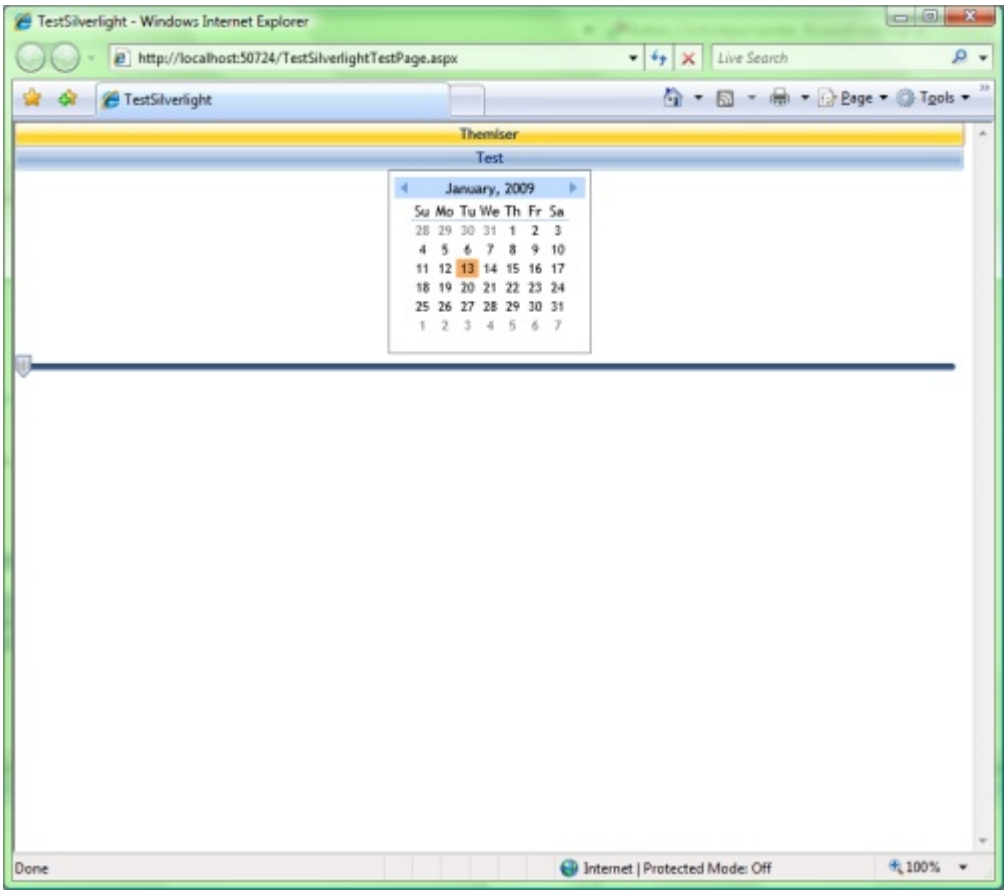
On crée une URI vers le XAML de notre thème, on assigne ce thème à un élément (ici notre StackPanel principal), puis on applique le thème à notre élément.

La troisième ligne définit le mode qui sera utilisé pour appliquer notre thème. Ce mode peut avoir 3 valeurs différentes :

- None : spécifie de ne pas appliquer le thème implicitement (valeur par défaut)
- OneTime : le thème sera appliqué implicitement à tous les contrôles présents. Tous les contrôles ajoutés par la suite garderont leur thème original
- Auto : le thème sera appliqué implicitement à tous les contrôles, même ceux ajoutés dynamiquement par la suite (mauvais pour les performances)

Le résultat est immédiat après le clic.

Copyright 2009 - Benjamin Roux. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérets. Droits de diffusion permanents accordés à developpez LLC.
http://broux.developpez.com/articles/csharp/silverlight-toolkit-partie-1



Tout peut aussi être fait en XAML :

```
<UserControl x:Class="TestSilverlight.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:basic="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls"
    xmlns:t="clr-
namespace:Microsoft.Windows.Controls.Theming;assembly=Microsoft.Windows.Controls.Theming"
    Width="800" Height="600">
    <StackPanel x:Name="LayoutRoot" Background="White" t:ImplicitStyleManager.ApplyMode="Auto"
t:ImplicitStyleManager.ResourceDictionaryUri="TestSilverlight;component/
Microsoft.Windows.Controls.Theming.BureauBlue.xaml">
        <StackPanel>
            <Button Content="Test" />
            <basic:Calendar x:Name="test" />
            <Slider />
        </StackPanel>
    </StackPanel>
</UserControl>
```

Ici le thème sera appliqué implicitement (mode Auto). Si le mode avait été *None*, il aurait fallu appeler explicitement la méthode **Apply** en code behind.

Comme nous venons de le constater, ce nouveau système de templatisation est plutôt puissant, il nous permet de modifier nos thèmes à la volée, simplement en pointant vers un nouveau fichier de thème.

Il répond parfaitement à une problématique que je trouvais souvent sur les forums officiels de Silverlight, qui était de pouvoir modifier les thèmes au runtime. La solution proposée est élégante et ravira tous les développeurs.

Copyright 2009 - Benjamin Roux. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérets. Droits de diffusion permanents accordés à developpez LLC.

4 - Conclusion

Cet article sur les nouveaux contrôles et thèmes contenus dans le Silverlight Toolkit se termine. Nous avons passé en revue les différents contrôles, qui nous simplifieront la tâche, à nous, développeurs Silverlight.

Comme dit précédemment, 2 autres articles sont en cours d'écriture sur les autres nouveautés du *Toolkit*.

Copyright 2009 - Benjamin Roux. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérets. Droits de diffusion permanents accordés à developpez LLC.

5 - Remerciements

Merci à toute l'équipe .NET pour leurs relectures ainsi qu'à Thomas Levesque et Joris Crozier pour leurs corrections orthographiques.

Copyright 2009 - Benjamin Roux. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérets. Droits de diffusion permanents accordés à developpez LLC.