

Le Data Binding en Silverlight 2

par Benjamin Roux ([Retour aux articles](#))

Date de publication : 16/09/2008

Dernière mise à jour :

Cet article vous présentera le Data Binding en Silverlight 2.

1 - Data Binding ?.....	3
2 - Le Data Binding en Silverlight 2.....	4
2-1 - Comment faire.....	4
2-2 - Les Converters.....	5
2-3 - Les différents modes.....	7
2-4 - Afficher une liste d'objets.....	9
2-5 - Conclusion.....	12
3 - Remerciements.....	13

1 - Data Binding ?

Le Data Binding est simplement le moyen utilisé pour faire le lien entre votre interface et votre source de données.

Ce concept marque encore plus la séparation entre les différentes couches de votre application (ici entre la couche présentation et les autres).

On appelle l'interface **la cible** et le fournisseur de données, **la source**.

2 - Le Data Binding en Silverlight 2

2-1 - Comment faire

Pour illustrer cet article, nous allons créer une petite application de gestion de tutoriels.

Nous allons d'abord commencer par créer notre classe **Tutorial** :

```
public class Tutorial
{
    public string Title { get; set; }
    public string Author { get; set; }
    public string Url { get; set; }
    public int Rate { get; set; }
}
```

Un tutoriel a un titre, un auteur, une url et une note. Rien de bien compliqué.

Maintenant que nous avons notre objet métier, nous allons créer notre interface (avec simplement des **TextBlock**) :

```
<Grid x:Name="LayoutRoot" Background="White">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="75" />
        <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="25" />
        <RowDefinition Height="25" />
        <RowDefinition Height="25" />
        <RowDefinition Height="25" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <TextBlock Grid.Row="0" Text="Titre :" />
    <TextBlock Grid.Row="1" Text="Auteur :" />
    <TextBlock Grid.Row="2" Text="URL :" />
    <TextBlock Grid.Row="3" Text="Note :" />
    <TextBlock x:Name="TxtTitle" Grid.Row="0" Grid.Column="1" Text="{Binding Title}" />
    <TextBlock x:Name="TxtAuthor" Grid.Row="1" Grid.Column="1" Text="{Binding Author}" />
    <TextBlock x:Name="TxtUrl" Grid.Row="2" Grid.Column="1" Text="{Binding Url}" />
    <TextBlock x:Name="TxtRate" Grid.Row="3" Grid.Column="1" Text="{Binding Rate}" />
</Grid>
```

Et voilà. Vous apercevez par la même occasion la syntaxe à utiliser pour Binder une propriété de notre source, à une propriété de notre cible (ici des **TextBlock**) :

```
Property="{Binding PropertyName}"
```

Maintenant il faut dire à nos **TextBlock** qui est notre source. Nous allons d'abord créer un tutoriel.

Dans la méthode **InitializeComponent** de notre *Page.xaml*.

```
Tutorial templates = new Tutorial()
{
    Author = "Benjamin Roux",
    Title = "Templates avancées en Silverlight",
    Url = "http://broux.developpez.com/articles/csharp/templates-silverlight/",
    Rate = 10
};
```

Maintenant nous allons spécifier pour chacun de nos **TextBlock**, notre objet :

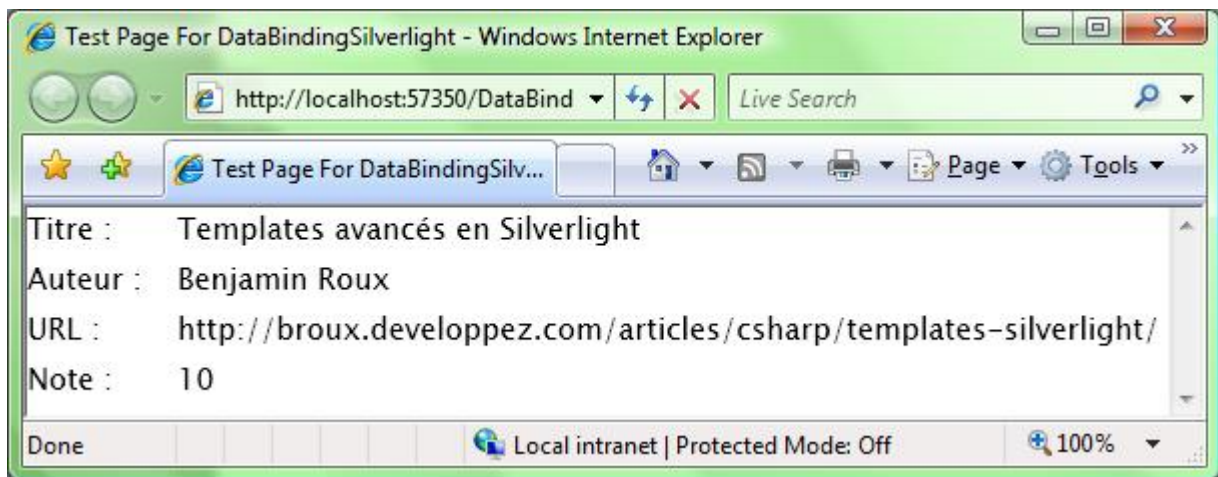
```
TxtAuthor.DataContext = templates;
TxtRate.DataContext = templates;
TxtTitle.DataContext = templates;
TxtUrl.DataContext = templates;
```

Rébarbatif non ?

On peut factoriser tout ça, en spécifiant le *DataContext* à l'élément qui englobe tous nos **TextBlock**, c'est-à-dire notre **Grid** qui se nomme *LayoutRoot* :

```
LayoutRoot.DataContext = templates;
```

Voici le résultat :



A noter qu'il existe une autre manière de spécifier le Binding, cette fois-ci par du code C#. Au lieu de faire par exemple

```
Text="{Binding Title}"
```

Il est possible de faire dans le code behind :

```
TxtTitle.SetBinding(TextBlock.TextProperty, new System.Windows.Data.Binding("Title"));
```

Voilà pour la version basique. Vous avez sûrement remarqué que pour afficher l'URL nous avons aussi utilisé un **TextBlock** et vous vous demandez peut-être pourquoi est-ce que nous n'avons pas utilisé un **HyperLinkButton**. Nous allons voir ça dans la prochaine partie.

2-2 - Les Converters

Alors pourquoi est-ce que nous n'avons pas utilisé un **HyperLinkButton** pour afficher mon URL ? Tout simplement car dans notre classe **Tutorial**, l'URL est stockée dans un string. Or la propriété *NavigateUri* veut un objet de type **Uri** : du coup cela n'aurait pas fonctionné.

Une solution aurait été de rajouter une propriété de type **Uri** à ma classe **Tutorial** et de binder cette propriété sur le *NavigateUri* de mon **HyperLinkButton**. Mais cela nous fait rajouter une propriété et, de plus, ce n'est pas super propre.

La solution : utiliser un **Converter** !

Les **Converters** sont simplement des classes qui permettent de convertir un objet d'un type en un autre.

C'est extrêmement simple à faire : il suffit d'implémenter l'interface **IValueConverter**.

Exemple avec un **Converter**, pour convertir un **string** en **Uri** :

```
public class String2UriConverter : IValueConverter
{
    #region IValueConverter Members

    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        if (value is string) return new Uri((string)value);
        else return value;
    }

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

    #endregion
}
```

Nous allons maintenant remplacer notre **TextBlock** par un **HyperLinkButton** :

```
<HyperlinkButton Grid.Row="2" Grid.Column="1" Content="{Binding Url}"
    NavigateUri="{Binding Url, Converter={StaticResource String2UriConverter}}"/> >
```

Voici la syntaxe pour utiliser un **Converter** en XAML :

```
Property="{Binding PropertyName, Converter={StaticResource String2UriConverter}}"
```

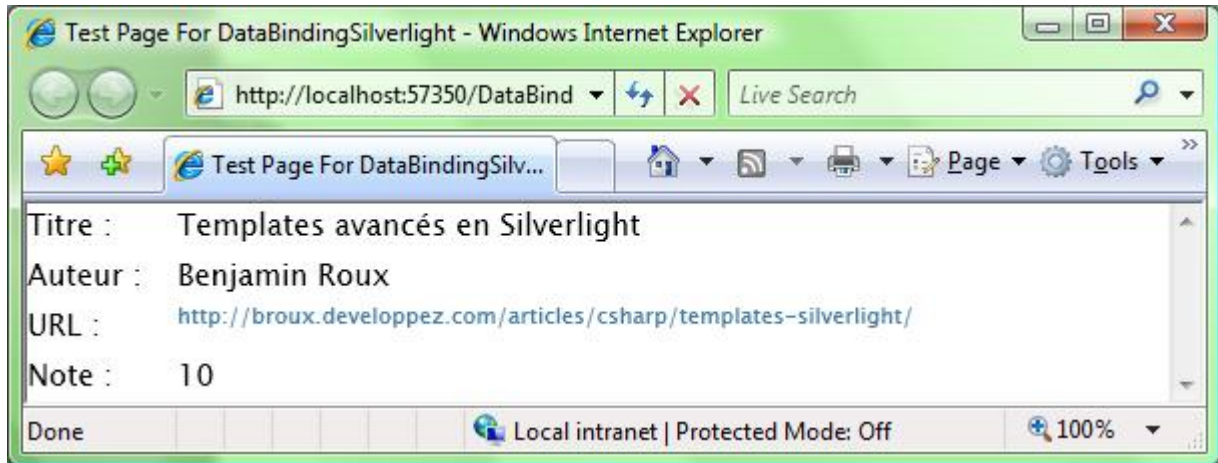
En C# :

```
System.Windows.Data.Binding binding = new System.Windows.Data.Binding("PropertyName");
binding.Converter = new Converters.String2UriConverter();
```

On n'oublie pas de rajouter notre **Converter** en ressource :


```
<UserControl x:Class="DataBindingSilverlight.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:conv="clr-namespace:DataBindingSilverlight.Converters"
    Width="600" Height="300">
    <UserControl.Resources>
        <conv:String2UriConverter x:Key="String2UriConverter"/> >
    </UserControl.Resources>
    [...]
</UserControl>
```

Et voilà le tour est joué. Au moment du Binding, notre **string** est transformé en **Uri**, objet attendu pour la propriété **NavigateUri**.



Nous allons maintenant voir les différents mode de Data Binding.

2-3 - Les différents modes

Il faut également savoir qu'il existe plusieurs mode de Binding (enum C#). Ces derniers sont répertoriés sur la page de la  **MSDN**.

Pour rappel, on appelle l'interface **la cible** et le fournisseur de données **la source**.


Type	Description
OneTime	Met à jour la propriété de la cible .
OneWay	Met à jour la propriété de la cible . Une modification sur la source est répercutée sur la cible (valeur par défaut).
TwoWay	Met à jour la propriété de la cible. Une modification sur la source est répercutée sur la cible et inversement.

Pour choisir le mode, il suffit de le spécifier lors du Binding :

```
Property="{Binding PropertyName, Mode=OneWay/TwoWay/OneTime}"
```

En C# :

```
System.Windows.Data.Binding binding = new System.Windows.Data.Binding("PropertyName");
binding.Mode = BindingMode.OneTime;
```

Pour les modes **OneWay** et **TwoWay**, les changements ne seront pas effectués tout seuls, il faut que la classe utilisée pour le Binding implémente l'interface  **INotifyPropertyChanged**.

Prenons l'exemple avec notre gestionnaire de tutoriels.

Nous allons rajouter un bouton qui modifie la note de notre tutoriel, pour voir le résultat :

```
<Button Grid.Row="4" Grid.ColumnSpan="2" Content="Changer note" Height="50" Width="100" Click="Button_Click" />
```

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    templates.Rate = 15;
}
```

Un clic sur le bouton modifie effectivement la note de notre tutoriel, mais aucun changement visuel n'apparaît. Pour le faire on pourrait éventuellement définir notre *DataContext* à *null* avant de le redéfinir sur notre objet, mais cela reste un bidouillage.

La solution est donc, comme expliqué plus haut, d'implémenter l'interface **INotifyPropertyChanged**, dans notre classe **Tutorial** :

```
public class Tutorial : INotifyPropertyChanged
{
    private string mTitle;
    private string mAuthor;
    private string mUrl;
    private int mRate;

    public string Title
    {
        get { return mTitle; }
        set
        {
            mTitle = value;
            NotifyPropertyChanged("Title");
        }
    }

    public string Author
    {
        get { return mAuthor; }
        set
        {
            mAuthor = value;
            NotifyPropertyChanged("Author");
        }
    }

    public string Url
    {
        get { return mUrl; }
        set
        {
            mUrl = value;
            NotifyPropertyChanged("Url");
        }
    }

    public int Rate
    {
        get { return mRate; }
        set
        {
            mRate = value;
            NotifyPropertyChanged("Rate");
        }
    }

    #region INotifyPropertyChanged Members

    public event PropertyChangedEventHandler PropertyChanged;

    public void NotifyPropertyChanged(string propertyName)
    {

```



```
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
        }
    }

    #endregion
}
```

Chaque fois qu'une propriété est modifiée, nous appelons la méthode **NotifyPropertyChanged**, qui se charge de lancer l'évènement **PropertyChanged**.

Désormais, lorsque notre source de données sera modifiée, toutes les propriétés de nos objets visuels bindées sur des propriétés de notre objet seront mises à jour.

Nous venons d'illustrer le mode **OneWay** : la source de donnée a été modifiée.

Pour illustrer le mode **TwoWay**, nous allons remplacer le **TextBlock** de la note par une **TextBox**. On n'oublie pas de modifier le mode de Binding également :

```
<TextBox x:Name="TxtRate" Grid.Row="3" Grid.Column="1" Text="{Binding Rate, Mode=TwoWay}" />
```

Désormais, lorsque nous modifierons la valeur de la **TextBox** et que cette dernière perdra le focus, la source de donnée sera mise à jour !

Voici pour les différents mode de Binding, c'est à vous de choisir le mode qui vous convient selon ce que vous voulez faire :

- **OneTime** pour des données qui ne changeront pas
- **OneWay** pour des données qui sont susceptibles d'être modifiées ailleurs
- **TwoWay** pour des données que l'utilisateur peut modifier et qui peuvent aussi être modifiées ailleurs...

2-4 - Afficher une liste d'objets

Nous avons vu comment afficher un seul objet, nous allons maintenant voir comment afficher une liste d'objets.

Tout d'abord, nous allons créer une liste de tutoriaux :

```
List<Tutorial> tutoriels = new List<Tutorial>();

Tutorial templates = new Tutorial()
{
    Author = "Benjamin Roux",
    Title = "Templates avancés en Silverlight",
    Url = "http://broux.developpez.com/articles/csharp/templates-silverlight/",
    Rate = 10
};

Tutorial animations = new Tutorial()
{
    Author = "Benjamin Roux",
    Title = "Les animations en Silverlight",
    Url = "http://broux.developpez.com/articles/csharp/animations-silverlight/",
    Rate = 10
};
[...]
tutoriels.Add(templates);
tutoriels.Add(animations);
```

Nous voici avec une liste de **Tutoriel**.


Pour les afficher, nous avons plusieurs possibilités. Tout d'abord une **ListBox**, ou encore un **ItemsControl** (équivalent du **Repeater** en ASP.NET). C'est ce dernier que nous allons utiliser.

```
<ItemsControl x:Name="Tutoriels">
  <ItemsControl.ItemTemplate>
    <DataTemplate>
      <Grid>
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="75" />
          <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
          <RowDefinition Height="25" />
          <RowDefinition Height="25" />
          <RowDefinition Height="25" />
          <RowDefinition Height="25" />
          <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <TextBlock Grid.Row="0" Text="Titre :" />
        <TextBlock Grid.Row="1" Text="Auteur :" />
        <TextBlock Grid.Row="2" Text="URL :" />
        <TextBlock Grid.Row="3" Text="Note :" />
        <TextBlock x:Name="TxtTitle" Grid.Row="0" Grid.Column="1" Text="{Binding Title}" />
        <TextBlock x:Name="TxtAuthor" Grid.Row="1" Grid.Column="1" Text="{Binding Author}" />
        <HyperlinkButton Grid.Row="2" Grid.Column="1" Content="{Binding Url}"
          NavigateUri="{Binding Url, Converter={StaticResource String2UriConverter}}" />
        <TextBox x:Name="TxtRate" Grid.Row="3" Grid.Column="1" Text="{Binding Rate}" />
      </Grid>
    </DataTemplate>
  </ItemsControl.ItemTemplate>
</ItemsControl>
```

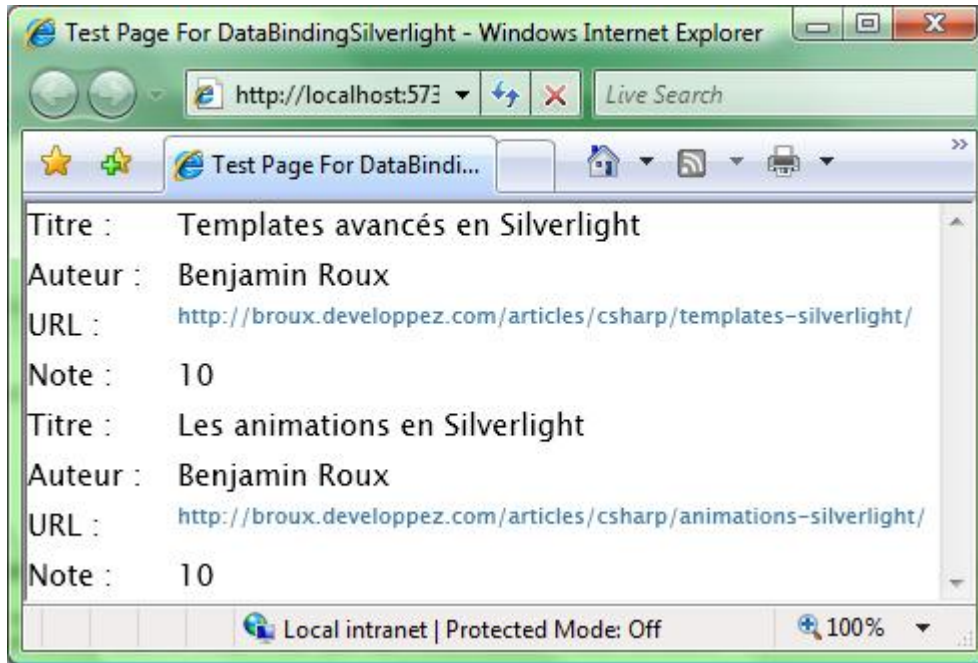
On reprend notre UI d'avant et on l'encadre par un contrôle de type **ItemsControl**.

On n'oublie pas de spécifier notre source de données à ce contrôle :

```
Tutoriels.ItemsSource = tutoriels;
```

 *Cette fois on utilise la propriété **ItemsSource** et non **DataContext**.*

Voyons le résultat :



Nous avons tous nos éléments les uns à la suite des autres. Vraiment pratique !

On peut également modifier les éléments comme précédemment : **TextBox** pour la note, ou modification de la note via un bouton. Tous les changements sont effectués visuellement.

En revanche, il y a un problème : l'ajout ou la suppression d'éléments dans notre liste.

Si l'on rajoute un bouton et que lors du clic sur le bouton on tente de rajouter un élément à notre **List**, comme ceci.


```

Tutorial databinding = new Tutorial()
{
    Author = "Benjamin Roux",
    Title = "Le Data Binding en Silverlight",
    Url = "http://broux.developpez.com/articles/csharp/databinding-silverlight/",
    Rate = 10
};

tutoriels.Add(databinding);
    
```

L'élément n'apparaît pas ! On pourrait comme précédemment mettre la propriété *ItemsSource* à *null* pour la remettre sur notre **List** par la suite mais c'est encore et toujours du bidouillage.

La solution est d'utiliser une collection qui implémente  **INotifyCollectionChanged**.

Ca tombe bien : Silverlight 2 met à notre disposition une classe qui l'implémente :  **ObservableCollection<T>**.

Cette classe fait tout comme la classe **List<T>**, sauf qu'elle implémente **INotifyCollectionChanged**.

On remplace dans notre code :

```

ObservableCollection<Tutorial> tutoriels = new ObservableCollection<Tutorial>();
    
```

Et désormais, lorsqu'un élément est ajouté ou supprimé de notre collection, un évènement est lancé. Il dit à notre **ItemsControl** que la source a changé et qu'il doit donc faire les modifications nécessaires !

2-5 - Conclusion

Voici la fin de notre article sur le Data Binding en Silverlight 2 qui, j'espère, vous sera très utile. Il est à noter que des nouveautés y seront apportées lors de la release de Silverlight 2.

3 - Remerciements

Je tiens à remercier toute l'équipe .NET pour leurs conseils et relectures, ainsi que **MaliciaR** pour sa correction orthographique et surtout grammaticale.