DataBindings

Contenidos

1. Introducción

Un *Data Binding* (enlace de datos) es una técnica general que vincula dos fuentes de datos o información y mantiene automáticamente la sincronización entre ellos.

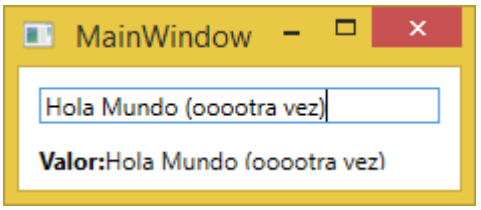
Con WPF se trata de un aspecto en el que se ha puesto bastante énfasis, pues es el método preferido para pasar datos desde el código a la capa de Interfaz de Usuario. Podemos establecer propiedades en un control (como **`ListBox`**) manualmente añadiendo elementos en un bucle, pero la forma más limpia de acuerdo a WPF es añadir un *data binding* entre el elemento fuente y el destino (correspondiente a la interfaz de usuario).

2. Un ejemplo simple

Como es habitual, vamos a utilizar un *data binding* en un ejemplo simple, que a continuación se irá desarrollando.

```
<Window x:Class="DataBinding.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="125" Width="300">
    <StackPanel Margin="10">
        <TextBox Name="txtValor" />
        <WrapPanel Margin="0,10">
            <TextBlock Text="Valor: " FontWeight="Bold" />
            <TextBlock Text="{Binding Path=Text, ElementName=txtValor}" />

        </WrapPanel>
    </StackPanel>
</Window>
```



Este ejemplo muestra como vinculamos el valor del **`TextBlock`** para que se corresponda con la propiedad **`Text`** del **`TextBox`**. Como puede verse, el **`TextBlock`** se actualiza automáticamente cuando introducimos texto en la caja de texto superior.

Sin la ayuda del *Data Binding*, esto requeriría que escucháramos al **`TextBox`** en un evento y luego actualizáramos el **`TextBlock`** cada vez que el texto cambia, pero con los *data binding* la conexión se establece de manera automática.

3. Sintaxis

XAML define una extensión del marcado mediante llaves.

Para *Data Binding* usamos la extensión **`Binding`**, que permite describir la relación con el **`Textbox`**. En su forma más simple, un **`Binding`** tiene el siguiente aspecto:

```
{Binding}
```

La forma anterior únicamente devuelve el **`DataContext`** (sobre el cual se hablará más adelante). Habitualmente lo que queremos es enlazar una propiedad a otra en el **`DataContext`**, por lo que la siguiente construcción se utiliza más:

```
{Binding Path=NombrePropiedad}
```

El **`Path`** indica la propiedad que quiero enlazar. Sin embargo, dado que **`Path`** es la propiedad por defecto para un **`Binding`**, puedo dejarlo sin nombrar con esta sintaxis (es indiferente usar una sintaxis o la otra):

```
{Binding NombrePropiedad}
```

Un **`Binding`** tiene otras propiedades, una de ellas es la propiedad **`ElementName`** que hemos usado en el ejemplo. Esto nos permite conectar directamente a otro elemento de la interfaz. Nótese que cada propiedad que añadimos al **`Binding`** se separa por una coma:

```
{Binding Path=Text, ElementName=txtValor}
```

4. Uso de DataContext

La propiedad **DataContext** es la fuente por defecto de los *bindings*, salvo que se declare específicamente otra fuente. Está definida en la clase **FrameworkElement**, de la cual heredan la mayor parte de los controles WPF, incluyendo **Window**.

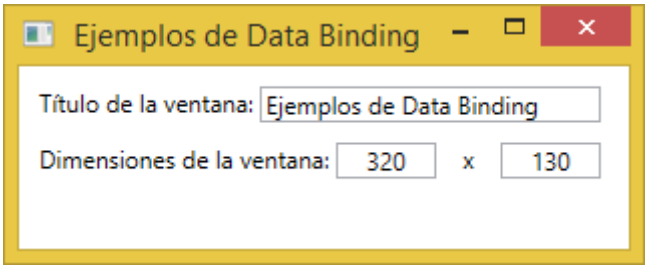
No hay fuente de datos por defecto para la propiedad **DataContext**, pero dado que se hereda a través de la jerarquía de controles, podemos establecer un **DataContext** para la ventana y usarlo en los controles hijos. Veamos un ejemplo:

```
<Window x:Class="DataBinding.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Ejemplos de Data Binding" Height="130" Width="320">
    <StackPanel Margin="10">
        <WrapPanel>
            <TextBlock Text="Título de la ventana: " />
            <TextBox  Text="{Binding  Title,
                      UpdateSourceTrigger=PropertyChanged}" Width="170" />
        </WrapPanel>
        <WrapPanel Margin="0,10,0,0">
            <TextBlock Text="Dimensiones de la ventana: "></TextBlock>
            <TextBox Text="{Binding Width}" Width="50" TextAlignment="Center" />
            <TextBlock Text=" x " Margin="10,0" />
            <TextBox Text="{Binding Height}" Width="50"  TextAlignment="Center"/>
        </WrapPanel>
    </StackPanel>
</Window>
```

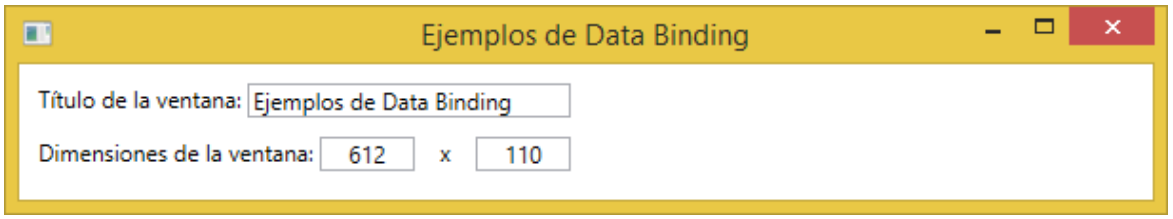
```
namespace DataBinding
{
    /// <summary>
    /// Lógica de interacción para MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            this.DataContext = this;
        }
    }
}
```

El *Code-Behind* de este ejemplo únicamente añade una línea de código: Tras la llamada estándar a InitializeComponent(), se asigna la referencia al propio objeto (**`this`**) al **`DataContext`**, lo que básicamente informa a la ventana de que él mismo será el **`DataContext`**

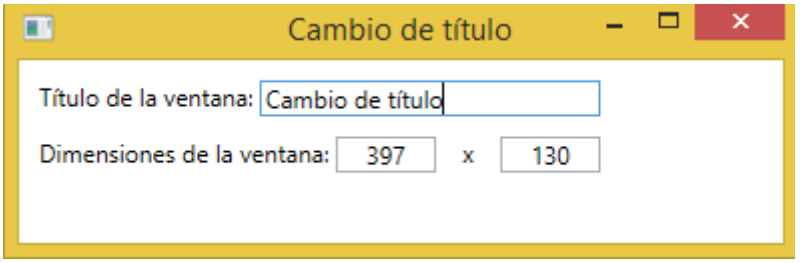
En XAML, usamos esta característica para vincular varias propiedades de **`Window`**, incluyendo el **`Title`**, **`Width`** y **`Height`**. Dada que **`Window`** es el **`DataContext`**, que se pasa a cada uno de los controles hijos, no es necesario definir una fuente en cada uno de los *Bindings* (simplemente se utiliza como si estuviera disponible globalmente).



Si redimensionamos la ventana veremos que los cambios de dimensión se reflejan inmediatamente en las cajas de texto.



A la inversa también funciona: Si cambiamos el título o las dimensiones, los cambios se reflejarán automáticamente en la fuente del *binding*.



El modo en que los cambios se transmiten de nuevo a la fuente se determina a partir de la propiedad UpdateSourceTrigger, que puede tomar los siguientes valores:

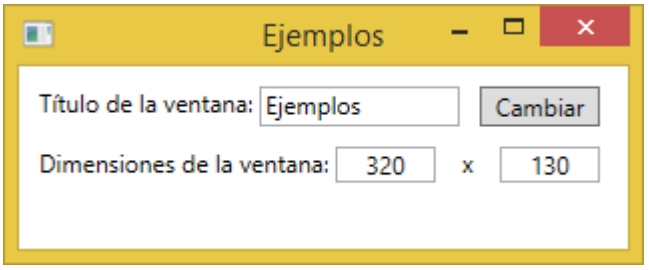
- **`Default`** (valor por defecto). La fuente se actualiza en función de la propiedad a la que se vincula.
- **`PropertyChanged`**. La propiedad cambia cuando se actualiza. Es lo mismo que por defecto para todas las propiedades salvo **`Text`**.
- **`LostFocus`**. La propiedad sólo cambia cuando el control pierde el foco. Es lo mismo que por defecto para la propiedad **`Focus`**.
- **`Explicit`**. Hay que forzar la actualización manualmente, utilizando una llamada a **`UpdateSource`** en el *Binding.*

El siguiente ejemplo muestra cómo funcionan las propiedades mencionadas:

```
<Window x:Class="DataBinding.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Ejemplos" Height="130" Width="320">
    <StackPanel Margin="10">
        <WrapPanel>
            <TextBlock Text="Título de la ventana: " />
            <TextBox Name="txtTitulo"
              Text="{Binding Title, UpdateSourceTrigger=Explicit}" Width="100" />
            <Button Margin="10,0,0,0" Width="60" Content="Cambiar"
               Name="btnCambiaTitulo" Click="btnCambiaTitulo_Click"  />
        </WrapPanel>
        <WrapPanel Margin="0,10,0,0">
            <TextBlock Text="Dimensiones de la ventana: "></TextBlock>
            <TextBox Text="{Binding Width, UpdateSourceTrigger=LostFocus}"
               Width="50" TextAlignment="Center" />
            <TextBlock Text=" x " Margin="10,0" />
            <TextBox Text="{Binding Height, UpdateSourceTrigger=PropertyChanged}"
               Width="50"  TextAlignment="Center"/>
        </WrapPanel>
    </StackPanel>
</Window>
```

```
namespace DataBinding
{
    /// <summary>
    /// Lógica de interacción para MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            this.DataContext = this;
        }

        private void btnCambiaTitulo_Click(object sender, RoutedEventArgs e)
        {
          BindingExpression binding = txtTitulo.GetBindingExpression(TextBox.TextProperty);
            binding.UpdateSource();
        }
    }
}
```



Cada una de las cajas de texto utiliza un **`UpdateSourceTrigger`** distinto:

- El primero está establecido a **`Explicit`**. Por ello, añadimos un botón de texto junto al **`TextBox`** que permite actualizar la fuente a demanda. En el *Code-Behind* encontramos el manejador para el evento **`Click`**, donde usamos un par de líneas de código para encontrar el *binding* y posteriormente llamar al método **`UpdateSource()`**
- El segundo **`TextBox`** usa el valor **`LostFocus`**, que en realidad sería el valor por defecto para un control de tipo **`Text`**. Esto implica que el valor fuente será actualizado cada vez que el control destino pierda el foco.
- El tercer y último **`TextBox`** usa el valor **`PropertyChanged`**, que implica que el valor fuente será actualizado cada vez que la propiedad destino cambia, lo que en este caso ocurre en cuanto cambia el texto.

Como se ha visto en los ejemplos, usando la propiedad **`DataContext`** permite establecer a base de nuestros *bindings* a lo largo de la jerarquía de controles. Esto nos ahorra la incomodidad de tener que definir manualmente una fuente para cada *binding*.

Sin embargo, no significa que tengamos que utilizar el mismo **`DataContext`** para todos los controles de una ventana. Dado que cada control tiene su propia propiedad **`DataContext`**, se puede fácilmente romper la cadena de herencia y sobrescribir el **`DataContext`** con un nuevo valor. Esto nos permite tener un **`DataContext`** global en la ventana y otro más específico por ejemplo en un panel.