

Mostrando datos

Contenidos

1.	ItemsControl.....	2
1.1.	ItemsControl con data binding.....	3
1.2.	Mostrando los elementos horizontalmente.....	5
1.3.	Barras de desplazamiento	6
2.	ListBox.....	7
2.1.	Enlazando datos	9
2.2.	Trabajando con la selección	10
3.	ComboBox	15
3.1.	Contenido a medida.....	15
3.1.	ComboBox editables	17
3.2.	Enlazando a datos	18
3.3.	Trabajando con la selección	19
4.	DataGrid	21
4.1.	Columnas	23
4.2.	Detalles en las filas	24

1. ItemsControl

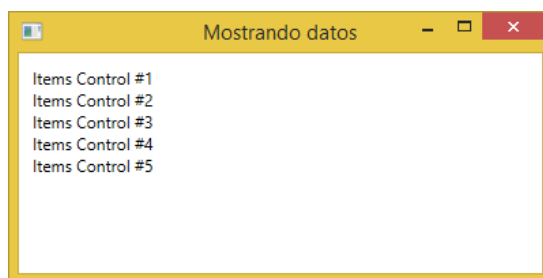
WPF ofrece un amplio rango de controles para mostrar listados de datos, que varían en cuanto a forma y complejidad.

La variante más simple es **ItemsControl**, que simplemente muestra un listado de elementos. Necesitamos aplicarle un estilo y una plantilla propias, pero en muchos casos es lo que necesitamos.

Comenzaremos por un ejemplo simple donde se comprueba cómo usar el control:

```
<Window x:Class="MostrandoDatos.VentanaPrincipal"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:system="clr-namespace:System;assembly=mscorlib"
        Title="Mostrando datos" Height="200" Width="400">
    <Grid Margin="10" >
        <ItemsControl>
            <system:String>Items Control #1</system:String>
            <system:String>Items Control #2</system:String>
            <system:String>Items Control #3</system:String>
            <system:String>Items Control #4</system:String>
            <system:String>Items Control #5</system:String>
        </ItemsControl>
    </Grid>
</Window>
```

Por ahora no hay demasiada diferencia entre usar un control como **ItemsControl** o añadir los elementos manualmente (por ejemplo, podríamos haber añadido 5 **TextBlock** de manera individual). Como vemos, por defecto el control **ItemsControl** no tiene ningún aspecto. Si pulsamos en alguno de los elementos, no sucede nada, porque no existe el concepto de elemento seleccionado ni similar.



1.1. ItemsControl con data binding

En principio el control no está pensado para ser usado con elementos definidos en el marcado, como hicimos en el primer ejemplo. Como cualquier otro control de WPF, **ItemsControl** está pensado para usar *data bindings*. Para ello tendremos que usar una **plantilla (template)** para definir cómo deben presentarse las clases *Code-Behind* al usuario.

Para ilustrar este comportamiento, a continuación se muestra un ejemplo donde se muestra una lista de tareas pendientes al usuario. Para mostrar la flexibilidad del modelo, vamos a usar una barra de progreso que muestra en qué porcentaje hemos superado la tarea:

```
<Window x:Class="MostrandoDatos.VentanaPrincipal"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Mostrando datos" Height="150" Width="300">
  <Grid Margin="10" >
    <ItemsControl
      Name="icListaTareas">
      <ItemsControl.ItemTemplate>
        <DataTemplate>
          <Grid Margin="0,0,0,5">
            <Grid.ColumnDefinitions>
              <ColumnDefinition Width="*" />
              <ColumnDefinition Width="100" />
            </Grid.ColumnDefinitions>

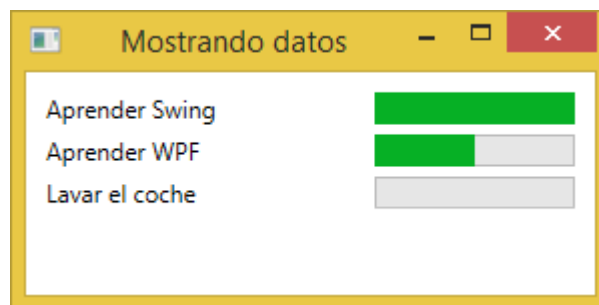
            <TextBlock Text="{Binding Titulo}"/>
            <ProgressBar Grid.Column="1"
              Minimum="0" Maximum="100" Value="{Binding Estado}" />
          </Grid>
        </DataTemplate>
      </ItemsControl.ItemTemplate>
    </ItemsControl>
  </Grid>
</Window>
```

```

namespace MostrandoDatos
{
    public partial class VentanaPrincipal : Window
    {
        public VentanaPrincipal()
        {
            InitializeComponent();
            List<Tarea> tareas = new List<Tarea>();
            tareas.Add(new Tarea(){Titulo="Aprender Swing", Estado=100});
            tareas.Add(new Tarea(){Titulo="Aprender WPF", Estado=50});
            tareas.Add(new Tarea(){Titulo="Lavar el coche", Estado=0});

            icListaTareas.ItemsSource = tareas;
        }
    }
    public class Tarea
    {
        public string Titulo { get; set; }
        public int Estado { get; set; }
    }
}

```



La parte más importante del ejemplo es la plantilla que especificamos dentro del **ItemsControl**, usando una etiqueta **DataTemplate** dentro de la propiedad **ItemsControl.ItemTemplate**. Esta plantilla especificará como se muestra cada uno de los elementos del control.

La plantilla consta de un **Grid** con dos columnas: La primera con un **TextBlock**, que muestra el título de las tareas, y la segunda con un **StatusBar** que muestra el progreso realizado. Ambos valores se vinculan a la propiedad correspondiente del tipo de objetos que componen los elementos de la lista (que será un tipo *Tarea* definido a medida en el *Code-Behind*).

En dicho archivo, además de definir la clase *Tarea* correspondiente, creamos una lista y la populamos la lista con varios objetos de este tipo. Para vincular la lista de objetos creados con el control definido en el XAML usamos la propiedad **ItemsSource**, que nos permite especificar la fuente de datos. El resto del trabajo se hace automáticamente, cada elemento se muestra usando la plantilla definida como puede verse en la captura correspondiente.

1.2. Mostrando los elementos horizontalmente

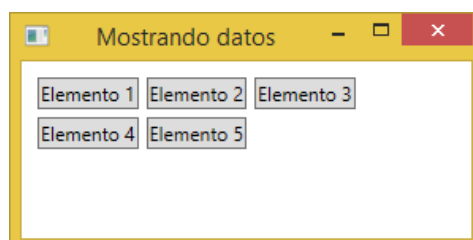
En los ejemplos anteriores, los elementos se muestran de arriba abajo, con cada elemento ocupando la fila entera. Esto sucede porque el elemento **ItemsControl** muestra todos los elementos en un **StackPanel** orientado verticalmente por defecto.

El comportamiento es fácil de cambiar, pues podemos cambiar el tipo de panel usado para mostrar los elementos usando la propiedad **ItemsPanel**, que admite un valor del tipo **ItemsPanelTemplate**.

```
<Window x:Class="MostrandoDatos.VentanaPrincipal"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:system="clr-namespace:System;assembly=mscorlib"
        Title="Mostrando datos" Height="150" Width="300">
    <Grid Margin="10" >
        <ItemsControl>
            <ItemsControl.ItemsPanel>
                <ItemsPanelTemplate>
                    <WrapPanel />
                </ItemsPanelTemplate>
            </ItemsControl.ItemsPanel>
            <ItemsControl.ItemTemplate>
                <DataTemplate>
                    <Button Content="{Binding}" Margin="0,0,5,5" />
                </DataTemplate>
            </ItemsControl.ItemTemplate>

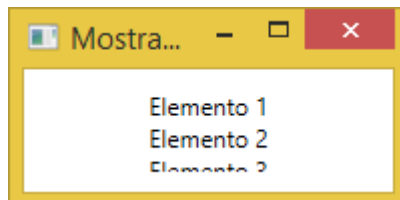
            <system:String>Elemento 1</system:String>
            <system:String>Elemento 2</system:String>
            <system:String>Elemento 3</system:String>
            <system:String>Elemento 4</system:String>
            <system:String>Elemento 5</system:String>
        </ItemsControl>
    </Grid>
</Window>
```

Especificamos que el **ItemsControl** debería usar un **WrapPanel** como platilla, especificándolo en **ItemsPanelTemplate**. Para comprobar las posibilidades, definimos un **ItemTemplate** que hace que los elementos de la lista (originalmente **String**) se usen como botones.



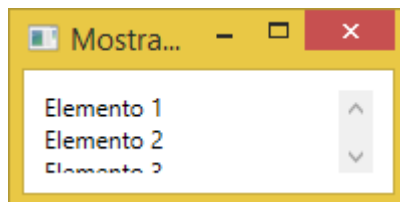
1.3. Barras de desplazamiento

Una vez que hemos comenzado a usar este tipo de controles, podemos entrar en un control muy común: Por defecto, no proporciona barras de desplazamiento, lo que implica que si el contenido no cabe en la ventana, simplemente se corta, como muestra el siguiente ejemplo:



Es muy fácil solucionar este problema mediante una barra de desplazamiento. Simplemente necesitamos añadir un elemento **ScrollView** sobre el **ItemsControl**

```
<Window x:Class="MostrandoDatos.VentanaPrincipal"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:system="clr-namespace:System;assembly=mscorlib"
    Title="Mostrando datos" Height="100" Width="200">
    <Grid Margin="10" >
        <ScrollView
            VerticalScrollBarVisibility="Auto" HorizontalScrollBarVisibility="Auto">
            <ItemsControl>
                <system:String>Elemento 1</system:String>
                <system:String>Elemento 2</system:String>
                <system:String>Elemento 3</system:String>
                <system:String>Elemento 4</system:String>
                <system:String>Elemento 5</system:String>
            </ItemsControl>
        </ScrollView>
    </Grid>
</Window>
```



Establecemos las dos opciones de visibilidad a Auto, haciendo que solo sean visibles cuando sea necesario.

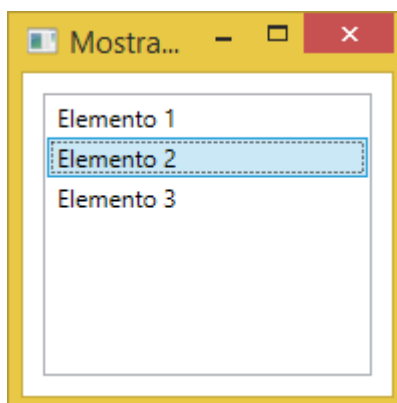
NOTA: Otra opción hubiera sido definir una barra de desplazamiento en la plantilla del control.

2. ListBox

El **ItemsControl** es una buena opción si queremos tener control completo acerca del modo en que se muestran los datos, y no necesitamos que el contenido sea seleccionable. Si por el contrario queremos que el usuario sea capaz de seleccionar elementos de la lista, es mejor usar controles como el **ListBox** o el **ListView**.

El **ListBox** añade algo de funcionalidad sobre el **ItemsControl**. Una de las principales diferencias es el hecho de que el **ListBox** trata con selecciones, permitiendo al usuario seleccionar uno o varios elementos de la lista y dando automáticamente retroalimentación.

```
<Window x:Class="MostrandoDatos.VentanaPrincipal"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:system="clr-namespace:System;assembly=mscorlib"
        Title="Mostrando datos" Height="200" Width="200">
    <Grid Margin="10" >
        <ListBox>
            <ListBoxItem>Elemento 1</ListBoxItem>
            <ListBoxItem>Elemento 2</ListBoxItem>
            <ListBoxItem>Elemento 3</ListBoxItem>
        </ListBox>
    </Grid>
</Window>
```



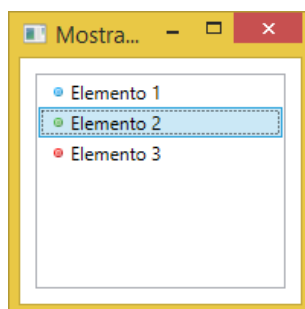
El ejemplo anterior es tan simple como parece: Declaramos un control **ListBox** y dentro del mismo declaramos tres **ListBoxItem**, cada uno con su texto. Sin embargo, dado que el **ListBoxItem** es en realidad un **ContentControl**, podemos definir contenido a medida para él:

```

<Window x:Class="MostrandoDatos.VentanaPrincipal"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:system="clr-namespace:System;assembly=mscorlib"
    Title="Mostrando datos" Height="200" Width="200">
    <Grid Margin="10" >
        <ListBox>
            <ListBoxItem>
                <StackPanel Orientation="Horizontal">
                    <Image
                        Source="/MostrandoDatos;component/Images/bullet_blue.png" />
                    <TextBlock>Elemento 1</TextBlock>
                </StackPanel>
            </ListBoxItem>
            <ListBoxItem>
                <StackPanel Orientation="Horizontal">
                    <Image
                        Source="/MostrandoDatos;component/Images/bullet_green.png" />
                    <TextBlock>Elemento 1</TextBlock>
                </StackPanel>
            </ListBoxItem>
            <ListBoxItem>
                <StackPanel Orientation="Horizontal">
                    <Image
                        Source="/MostrandoDatos;component/Images/bullet_red.png" />
                    <TextBlock>Elemento 1</TextBlock>
                </StackPanel>
            </ListBoxItem>
        </ListBox>
    </Grid>
</Window>

```

Para cada **ListBoxItem** añadimos un **StackPanel**, en el cual tenemos una imagen y un **TextBlock**. Esto nos da control total del contenido, como se ve en la siguiente captura:



De la captura también podemos notar una pequeña diferencia cuando comparamos con el **ItemsControl**, y es que por defecto se muestra un pequeño borde alrededor del control.

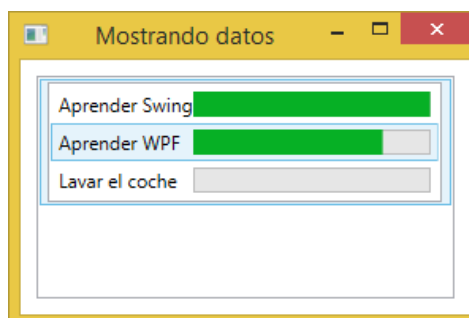
2.1. Enlazando datos

Al igual que en capítulo anterior, la definición manual de datos nos sirve para un primer ejemplo, pero lo más habitual es que los controles se generen a partir de elementos de una fuente de datos usando *data binding*.

Por defecto, al enlazar una lista de elementos a un **ListBox**, se llama a su método **ToString()** para representar el elemento. En otros casos no queremos este comportamiento, luego podemos declarar una plantilla **template** que se utiliza para mostrar el listado.

Reciclamos el ejemplo del listado de tareas usado anteriormente para aplicarlo a un **ListBox**:

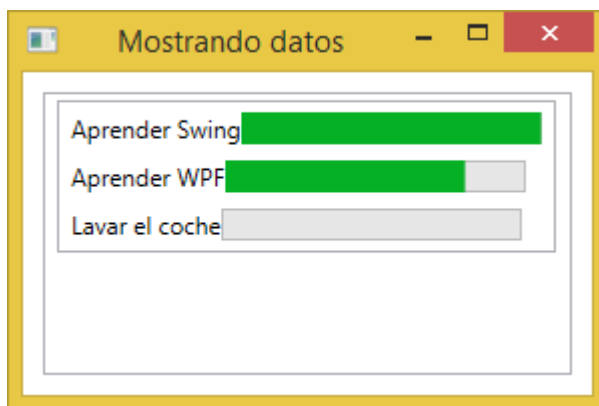
```
<Window x:Class="MostrandoDatos.VentanaPrincipal"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:system="clr-namespace:System;assembly=mscorlib"
    Title="Mostrando datos" Height="200" Width="300">
    <Grid Margin="10" >
        <ListBox>
            <ListBox Name="lbTareasPendientes"
                HorizontalContentAlignment="Stretch">
                <ListBox.ItemTemplate>
                    <DataTemplate>
                        <Grid Margin="0,2">
                            <Grid.ColumnDefinitions>
                                <ColumnDefinition Width="*" />
                                <ColumnDefinition Width="150" />
                            </Grid.ColumnDefinitions>
                            <TextBlock Text="{Binding Titulo}" />
                            <ProgressBar Grid.Column="1" Minimum="0" Maximum="100"
                                Value="{Binding Estado}" />
                        </Grid>
                    </DataTemplate>
                </ListBox.ItemTemplate>
            </ListBox>
        </Grid>
    </Window>
```



Como ya hemos visto, la mayor parte del trabajo ocurre en el **ItemTemplate** (la plantilla) que hemos definido en el **ListBox**. En la misma especificamos un **Grid** dividida en dos columnas, con un **TextBlock** mostrando el título en la primera y una barra de progreso mostrando el estado actual de la tarea en la segunda. Para mostrar los valores usamos un *data binding*.

En el archivo *Code-Behind* declaramos una clase **Tarea** para guardar el nombre y el estado de las tareas. En el constructor de la ventana inicializamos una lista, añadimos tres objetos de tipo **Tarea** y posteriormente asignamos la lista a la propiedad **ItemsSource** del **ListBox**. A continuación se usa la plantilla para mostrar automáticamente los elementos de la lista.

Nótese que la propiedad **HorizontalContentAlignment** está establecida a **Stretch**. Si no llegamos a establecerla, se hubiera respetado el valor por defecto en un **ListBox** que es **Left**, lo que implica cada elemento sólo toma el espacio horizontal que necesita. El resultado no hubiera quedado como queremos:



Al usar la propiedad **Stretch** los elementos toman todo el espacio disponible.

2.2. Trabajando con la selección

Como ya se mencionó, la principal diferencia entre el **ItemsControl** y el **ListBox** reside en que el **ListBox** trata y muestra la selección del usuario de manera automático.

Vamos a ver en el siguiente ejemplo como se puede trabajar con esta característica:

```

<Window x:Class="MostrandoDatos.VentanaPrincipal"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:system="clr-namespace:System;assembly=microsoft.windows.common-user-core-6.0.0"
    Title="Mostrando datos" Height="220" Width="450">
    <DockPanel Margin="10">

        <StackPanel DockPanel.Dock="Right" Margin="10,0">
            <TextBlock FontWeight="Bold" Margin="0,0,0,10">
                Selección del ListBox</TextBlock>
            <Button Name="btnMostrar" Click="btnMostrar_Click">Mostrar</Button>
            <Button Name="btnAnterior" Margin="0,5"
                Click="btnAnterior_Click">Anterior</Button>
            <Button Name="btnSiguiente" Click="btnSiguiente_Click">Siguiente</Button>
            <Button Name="btnWPF" Margin="0,5" Click="btnWPF_Click">WPF</Button>
            <Button Name="btnTodos" Click="btnTodos_Click">Seleccionar todos</Button>
        </StackPanel>
        <ListBox>
            <ListBox Name="lbTareasPendientes" SelectionMode="Extended"
                HorizontalContentAlignment="Stretch"
                SelectionChanged="lbTareasPendientes_SelectionChanged">
                <ListBox.ItemTemplate>
                    <DataTemplate>
                        <Grid Margin="0,2">
                            <Grid.ColumnDefinitions>
                                <ColumnDefinition Width="*" />
                                <ColumnDefinition Width="150" />
                            </Grid.ColumnDefinitions>
                            <TextBlock Text="{Binding Titulo}" />
                            <ProgressBar Grid.Column="1" Minimum="0" Maximum="100"
                                Value="{Binding Estado}" />
                        </Grid>
                    </DataTemplate>
                </ListBox.ItemTemplate>
            </ListBox>
        </DockPanel>
    </Window>

```

```
namespace MostrandoDatos
{
    public partial class VentanaPrincipal : Window
    {
        public VentanaPrincipal()
        {
            InitializeComponent();
            List<Tarea> tareas = new List<Tarea>();
            tareas.Add(new Tarea() { Titulo = "Aprender Swing", Estado = 100 });
            tareas.Add(new Tarea() { Titulo = "Aprender WPF", Estado = 80 });
            tareas.Add(new Tarea() { Titulo = "Lavar el coche", Estado = 0 });

            lbTareasPendientes.ItemsSource = tareas;
        }

        private void lbTareasPendientes_SelectionChanged(object sender,
            SelectionChangedEventArgs e)
        {
            if (lbTareasPendientes.SelectedItem != null)
                this.Title = (lbTareasPendientes.SelectedItem as Tarea).Titulo;
        }

        private void btnAnterior_Click(object sender, RoutedEventArgs e)
        {
            int indiceAnterior = 0;
            if (lbTareasPendientes.SelectedIndex > 0)
            {
                indiceAnterior = lbTareasPendientes.SelectedIndex - 1;
                lbTareasPendientes.SelectedIndex = indiceAnterior;
            }
        }

        private void btnMostrar_Click(object sender, RoutedEventArgs e)
        {
            foreach (Tarea tarea in lbTareasPendientes.SelectedItems)
                MessageBox.Show(tarea.Titulo);
        }
    }
}
```

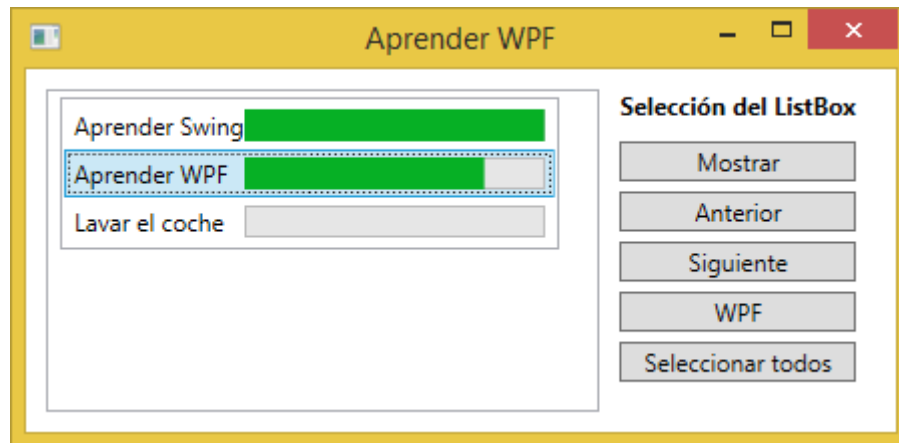
```
private void btnSiguiente_Click(object sender, RoutedEventArgs e)
{
    int indiceSiguiente = 0;
    if ((lbTareasPendientes.SelectedIndex >= 0)
    && (lbTareasPendientes.SelectedIndex < (lbTareasPendientes.Items.Count - 1)))
    {
        indiceSiguiente = lbTareasPendientes.SelectedIndex + 1;
        lbTareasPendientes.SelectedIndex = indiceSiguiente;
    }
}

private void btnWPF_Click(object sender, RoutedEventArgs e)
{
    foreach (Tarea tarea in lbTareasPendientes.Items)
    {
        if (tarea.Titulo.Contains("WPF"))
        {
            lbTareasPendientes.SelectedItem = tarea;
            break;
        }
    }
}

private void btnTodos_Click(object sender, RoutedEventArgs e)
{
    foreach (object o in lbTareasPendientes.Items)
        lbTareasPendientes.SelectedItems.Add(o);
}

}

public class Tarea
{
    public string Titulo { get; set; }
    public int Estado { get; set; }
}
}
```



Se ha definido un conjunto de botones a la derecha del **ListBox** para acceder o manipular la selección. También se ha cambiado el atributo **SelectionMode** a **Extended**, para permitir la selección de varios elementos. Esto puede hacerse de manera programática (como se hace con el último botón), o puede hacerlo el usuario pulsando [Control] o [Shift] mientras se pulsa en los elementos.

Para cada botón se ha definido un manejador del evento **Click**, además del evento **SelectionChanged** del **ListBox** que cambia el título de la ventana al cambiar la selección.

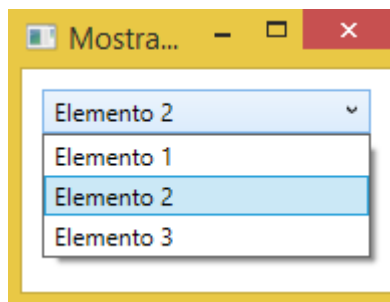
3. ComboBox

Un **ComboBox** es muy parecido al **ListBox**, aunque toma menos espacio, pues la lista de elementos se oculta cuando no se necesita. El **ComboBox** se utiliza en muchos sitios en Windows, pero para ir familiarizándonos con su uso vamos a introducir un ejemplo simple:

```
<Window x:Class="MostrandoDatos.VentanaPrincipal"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:system="clr-namespace:System;assembly=mscorlib"
        Title="Mostrando datos" Height="150" Width="200">

    <StackPanel Margin="10">
        <ComboBox>
            <ComboBoxItem>Elemento 1</ComboBoxItem>
            <ComboBoxItem IsSelected="True">Elemento 2</ComboBoxItem>
            <ComboBoxItem>Elemento 3</ComboBoxItem>
        </ComboBox>
    </StackPanel>
</Window>
```

En el ejemplo, se ha activado el control pulsando sobre él, haciendo que el listado de elementos se muestre. Lo único que hemos hecho es añadir algunos elementos, haciendo que uno de ellos se seleccione por defecto estableciendo la propiedad **IsSelected** en el mismo.



3.1. Contenido a medida

Como en los capítulos anteriores, se muestra un ejemplo de contenido a medida:

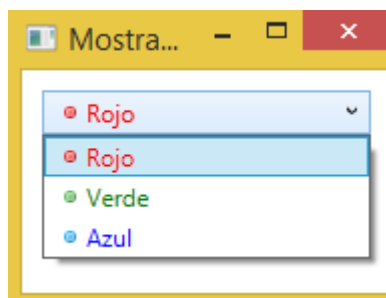
```

<Window x:Class="MostrandoDatos.VentanaPrincipal"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:system="clr-namespace:System;assembly=mscorlib"
    Title="Mostrando datos" Height="150" Width="200">

    <StackPanel Margin="10">
        <ComboBox>
            <ComboBoxItem IsSelected="True">
                <StackPanel Orientation="Horizontal">
                    <Image
                        Source="/MostrandoDatos;component/Images/bullet_red.png" />
                    <TextBlock Foreground="Red">Rojo</TextBlock>
                </StackPanel>
            </ComboBoxItem>
            <ComboBoxItem>
                <StackPanel Orientation="Horizontal">
                    <Image
                        Source="/MostrandoDatos;component/Images/bullet_green.png" />
                    <TextBlock Foreground="Green">Verde</TextBlock>
                </StackPanel>
            </ComboBoxItem>
            <ComboBoxItem>
                <StackPanel Orientation="Horizontal">
                    <Image
                        Source="/MostrandoDatos;component/Images/bullet_blue.png" />
                    <TextBlock Foreground="Blue">Azul</TextBlock>
                </StackPanel>
            </ComboBoxItem>
        </ComboBox>
    </StackPanel>
</Window>

```

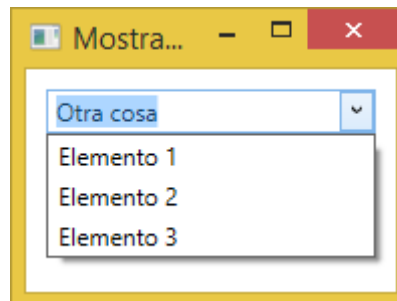
Como ya hemos hecho en otros capítulos, en cada elemento del control añadimos un `StackPanel` que contiene una imagen y un bloque de texto.



3.1. ComboBox editables

En el ejemplo anterior el usuario sólo podía seleccionar un valor de la lista, pero una de las cosas que aporta el **ComboBox** es que soporta la posibilidad de dejar que el usuario introduzca su propio valor. Esto es útil en situaciones donde queremos ayudar al usuario dando un conjunto pre-definido de opciones, dando igualmente la opción de introducir manualmente el valor deseado. Todo esto es controlado por la propiedad **IsEditable**, que cambia el comportamiento y el aspecto del **ComboBox**:

```
<ComboBox IsEditable="True">
```



Podemos introducir un valor completamente distinto o tomar uno de la lista. Además, al escribir el texto se autocompleta (basándose en el resto de elementos de la lista). Si queremos desactivar esta característica podemos establecer **IsTextSearchEnabled** a False.

3.2. Enlazando a datos

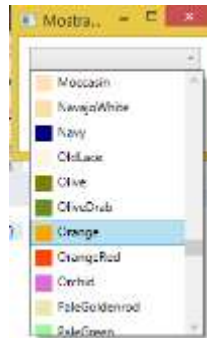
El presente ejemplo muestra como enlazar un **ComboBox** mediante un *binding*, a la par que aplicamos un template a medida que nos va a permitir mostrar un listado de colores.

```
<Window x:Class="MostrandoDatos.VentanaPrincipal"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:system="clr-namespace:System;assembly=mscorlib"
        Title="Mostrando datos" Height="150" Width="200">

    <StackPanel Margin="10">
        <ComboBox Name="cbColores">
            <ComboBox.ItemTemplate>
                <DataTemplate>
                    <StackPanel Orientation="Horizontal">
                        <Rectangle Fill="{Binding Name}"
                                Width="16" Height="16" Margin="0,2,5,2" />
                        <TextBlock Text="{Binding Name}" />
                    </StackPanel>
                </DataTemplate>
            </ComboBox.ItemTemplate>
        </ComboBox>
    </StackPanel>
</Window>
```

```
namespace MostrandoDatos
{
    public partial class VentanaPrincipal : Window
    {
        public VentanaPrincipal()
        {
            InitializeComponent();
            cbColores.ItemsSource = typeof(Colors).GetProperties();
        }
    }
}
```

En el *Code Behind*, obtenemos una lista de todos los colores con la clase **Colors**. La lista se asigna a la propiedad **ItemsSource** del **ComboBox**, que muestra cada color basándonos en el template definidos en el XAML. Cada ítem está compuesto por un rectángulo que se colorea con el color correspondiente y un nombre de color.



3.3. Trabajando con la selección

Una parte importante de trabajar con un **ComboBox** es ser capaz de leer la selección del usuario y usarla con algún fin. En el siguiente ejemplo, reutilizamos el ejercicio anterior, añadiendo algunos botones para controlar la selección. También se usa el evento **SelectionChanged** para determinar cuando la selección ha cambiado (bien sea por código o por parte del usuario) y actuar sobre ella.

```
<Window x:Class="MostrandoDatos.VentanaPrincipal"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:system="clr-namespace:System;assembly=mscorlib"
    Title="Mostrando datos" Height="125" Width="250">
    <StackPanel Margin="10">
        <ComboBox Name="cbColores" SelectionChanged="cbColores_SelectionChanged">
            <ComboBox.ItemTemplate>
                <DataTemplate>
                    <StackPanel Orientation="Horizontal">
                        <Rectangle Fill="{Binding Name}" Width="16" Height="16"
                            Margin="0,2,5,2" />
                        <TextBlock Text="{Binding Name}" />
                    </StackPanel>
                </DataTemplate>
            </ComboBox.ItemTemplate>
        </ComboBox>
        <WrapPanel Margin="15" HorizontalAlignment="Center">
            <Button Name="btnAnterior"
                Click="btnAnterior_Click" Width="55">Anterior</Button>
            <Button Name="btnSiguiente"
                Click="btnSiguiente_Click" Margin="5,0"
                Width="55">Siguiente</Button>
            <Button Name="btnAzul" Click="btnAzul_Click" Width="55">Azul</Button>
        </WrapPanel>
    </StackPanel>
</Window>
```

```
using System.Reflection;

namespace MostrandoDatos
{
    public partial class VentanaPrincipal : Window
    {
        public VentanaPrincipal()
        {
            InitializeComponent();
            cbColores.ItemsSource = typeof(Colors).GetProperties();
        }

        private void btnAnterior_Click(object sender, RoutedEventArgs e)
        {
            if (cbColores.SelectedIndex > 0)
            {
                cbColores.SelectedIndex = cbColores.SelectedIndex - 1;
            }
        }

        private void btnSiguiente_Click(object sender, RoutedEventArgs e)
        {
            if (cbColores.SelectedIndex < cbColores.Items.Count - 1)
            {
                cbColores.SelectedIndex = cbColores.SelectedIndex + 1;
            }
        }

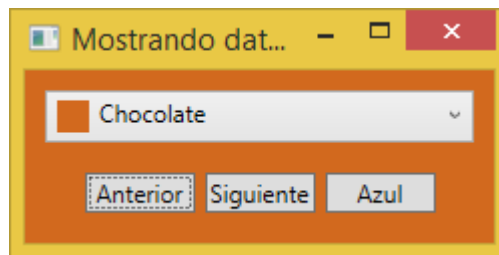
        private void btnAzul_Click(object sender, RoutedEventArgs e)
        {
            cbColores.SelectedItem = typeof(Colors).GetProperty("Blue");
        }

        private void cbColores_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            Color seleccion = (Color)(cbColores.SelectedItem as PropertyInfo).GetValue(null, null); this.Background = new SolidColorBrush(seleccion);
        }
    }
}
```

La parte interesante del ejemplo son los tres manejadores de eventos para los tres botones, así como el manejador de evento **SelectionChanged**. En los dos primeros, seleccionamos el elemento anterior o el siguiente leyendo la propiedad **SelectedIndex** y añadiendo o restando uno.

En el tercer manejador de evento (botón *Azul*), usamos **SelectedItem** para seleccionar un elemento específico basándonos en el valor. Aquí necesitamos usar reflexión para sacar la lista de colores.

El último manejador se corresponde a un cambio en el elemento seleccionado. Leemos el color correspondiente y lo usamos para cambiar el color de fondo de la ventana, como puede verse en la captura:



4. DataGrid

Un **DataGrid** nos permite visualizar información en forma de tabla. Habitualmente se usa en combinación con una base de datos o incluso un archivo XML, pero al igual que la mayoría de controles WPF, funciona bien con un listado de objetos en memoria. Es el enfoque que usaremos en este capítulo.

En un primer ejemplo crearemos un **DataGrid** sin establecer propiedades:

```
<Window x:Class="MostrandoDatos.VentanaPrincipal"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:system="clr-namespace:System;assembly=mscorlib"
    Title="Mostrando datos" SizeToContent="WidthAndHeight">

    <Grid Margin="10">
        <DataGrid Name="dgGente"></DataGrid>
    </Grid>

</Window>
```

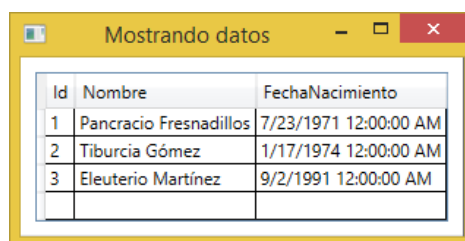
```

namespace MostrandoDatos
{
    public partial class VentanaPrincipal : Window
    {
        public VentanaPrincipal()
        {
            InitializeComponent();

            List<Usuario> users = new List<Usuario>();
            users.Add(new Usuario() { Id = 1, Nombre = "Pancracio Fresnadillos",
                                      FechaNacimiento = new DateTime(1971, 7, 23) });
            users.Add(new Usuario() { Id = 2, Nombre = "Tiburcia Gómez",
                                      FechaNacimiento = new DateTime(1974, 1, 17) });
            users.Add(new Usuario() { Id = 3, Nombre = "Eleuterio Martínez",
                                      FechaNacimiento = new DateTime(1991, 9, 2) });

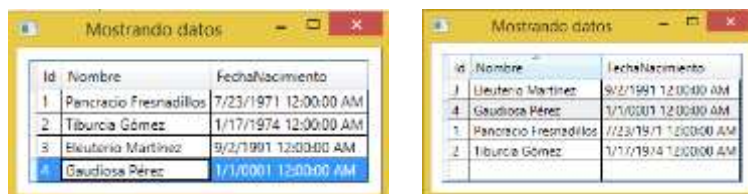
            dgGente.ItemsSource = users;
        }
    }
    public class Usuario
    {
        public int Id { get; set; }
        public string Nombre { get; set; }
        public DateTime FechaNacimiento { get; set; }
    }
}

```



Id	Nombre	FechaNacimiento
1	Pancracio Fresnadillos	7/23/1971 12:00:00 AM
2	Tiburcia Gómez	1/17/1974 12:00:00 AM
3	Eleuterio Martínez	9/2/1991 12:00:00 AM

Como vemos, el control se dibuja de manera automática. Si pulsamos en una de las celdas, podemos comprobar que podemos editar los datos, ya sean los correspondientes a un usuario existente o a un nuevo usuario y añadirlos a la tabla. Además, si pulsamos en una de las columnas veremos que se ordenan los datos en función de dicha columna.



Id	Nombre	FechaNacimiento
1	Pancracio Fresnadillos	7/23/1971 12:00:00 AM
2	Tiburcia Gómez	1/17/1974 12:00:00 AM
3	Eleuterio Martínez	9/2/1991 12:00:00 AM
4	Gaudiosa Pérez	1/1/0001 12:00:00 AM

Id	Nombre	FechaNacimiento
3	Eleuterio Martínez	9/2/1991 12:00:00 AM
4	Gaudiosa Pérez	1/1/0001 12:00:00 AM
1	Pancracio Fresnadillos	7/23/1971 12:00:00 AM
2	Tiburcia Gómez	1/17/1974 12:00:00 AM

4.1. Columnas

En algunos casos queremos definir manualmente las columnas que se muestran en el **DataGrid**, bien porque no queremos que se muestren todas las columnas de la fuente de datos, porque queremos controlar el nombre de las mismas o el tipo de datos que se van a mostrar.

Podemos seleccionar el tipo de columna basándonos en el tipo de datos que queremos mostrar/editar. Tenemos los siguientes tipos disponibles:

- **DataGridTextColumn**
- **DataGridCheckBoxColumn**
- **DataGridComboBoxColumn**
- **DataGridHyperlinkColumn**
- **DataGridTemplateColumn**

Como es lógico, el último tipo nos permite definir cualquier tipo de contenido, lo que abre la puerta a añadir controles a medida. Veamos el ejemplo anterior modificado (el *Code-Behind* no cambia):

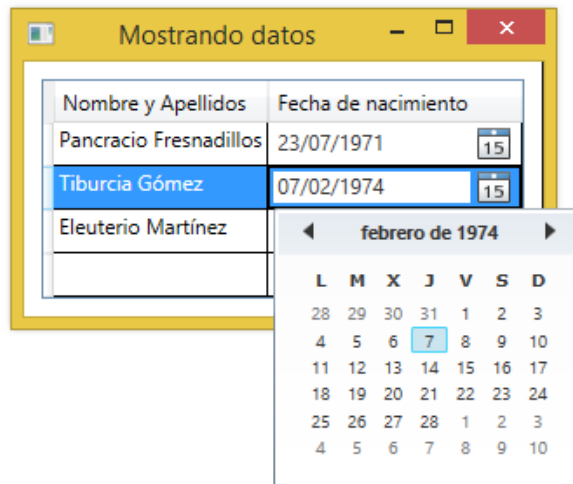
```
<Window x:Class="MostrandoDatos.VentanaPrincipal"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Mostrando datos" SizeToContent="WidthAndHeight">
    <Grid Margin="10">
        <DataGrid Name="dgGente" AutoGenerateColumns="False">
            <DataGrid.Columns>

                <DataGridTextColumn Header="Nombre y Apellidos"
                    Binding="{Binding Nombre}" />

                <DataGridTemplateColumn Header="Fecha de nacimiento">
                    <DataGridTemplateColumn.CellTemplate>
                        <DataTemplate>
                            <DatePicker SelectedDate="{Binding FechaNacimiento}"
                                BorderThickness="0" />
                        </DataTemplate>
                    </DataGridTemplateColumn.CellTemplate>
                </DataGridTemplateColumn>
            </DataGrid.Columns>
        </DataGrid>
    </Grid>
</Window>
```

En el marcado, hemos añadido la propiedad **AutoGenerateColumns** al **DataGrid**, que se ha establecido a **False** para así tener control de las columnas utilizadas. Como se puede ver, no se ha incluido la columna de ID (pues habitualmente no tiene importancia para el usuario final).

Para la propiedad **Nombre**, se ha usado una columna simple basada en texto. Para la columna correspondiente a la fecha de nacimiento se ha definido una columna a medida usando **DataGridTemplateColumn** con un control **DatePicker** dentro de él. Esto nos permite tomar la fecha de un calendario en lugar de hacerlo manualmente:



4.2. Detalles en lasfilas

Un escenario muy común cuando usamos un control **DataGrid** es la posibilidad de mostrar detalles de cada fila, habitualmente encima o debajo de la propia fila. El control nos permite hacer esto de manera muy sencilla:

```
<Window x:Class="MostrandoDatos.VentanaPrincipal"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Mostrando datos" SizeToContent="WidthAndHeight">
    <Grid Margin="10">
        <DataGrid Name="dgGente" AutoGenerateColumns="False">
            <DataGrid.Columns>
                <DataGridTextColumn Header="Nombre y apellidos"
                    Binding="{Binding Nombre}" />
                <DataGridTextColumn Header="Fecha de nacimiento"
                    Binding="{Binding FechaNacimiento}" />
            </DataGrid.Columns>
            <DataGrid.RowDetailsTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding Detalles}" Margin="10" />
                </DataTemplate>
            </DataGrid.RowDetailsTemplate>
        </DataGrid>
    </Grid>
</Window>
```



```

namespace MostrandoDatos
{
    public partial class VentanaPrincipal : Window
    {
        public VentanaPrincipal()
        {
            InitializeComponent();

            List<Usuario> users = new List<Usuario>();
            users.Add(new Usuario() { Id = 1, Nombre = "Pancracio Fresnadillos",
                FechaNacimiento = new DateTime(1971, 7, 23) });
            users.Add(new Usuario() { Id = 2, Nombre = "Tiburcia Gómez",
                FechaNacimiento = new DateTime(1974, 1, 17) });
            users.Add(new Usuario() { Id = 3, Nombre = "Eleuterio Martínez",
                FechaNacimiento = new DateTime(1991, 9, 2) });
            dgGente.ItemsSource = users;

        }
    }

    public class Usuario
    {
        public int Id { get; set; }

        public string Nombre { get; set; }

        public DateTime FechaNacimiento { get; set; }

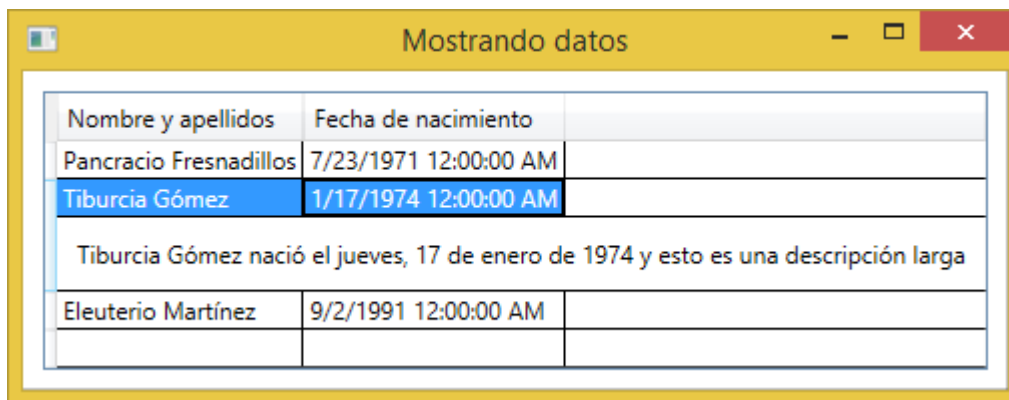
        public String Detalles
        {
            get
            {
                return this.Nombre + " nació el " + this.FechaNacimiento.ToLongDateString()+
                    " y esto es una descripción larga";
            }
        }
    }
}

```

Se ha expandido el ejemplo del apartado anterior con una nueva propiedad en la clase **User** llamada **Detalles**, que simplemente devuelve información acerca del usuario en forma de **String**.

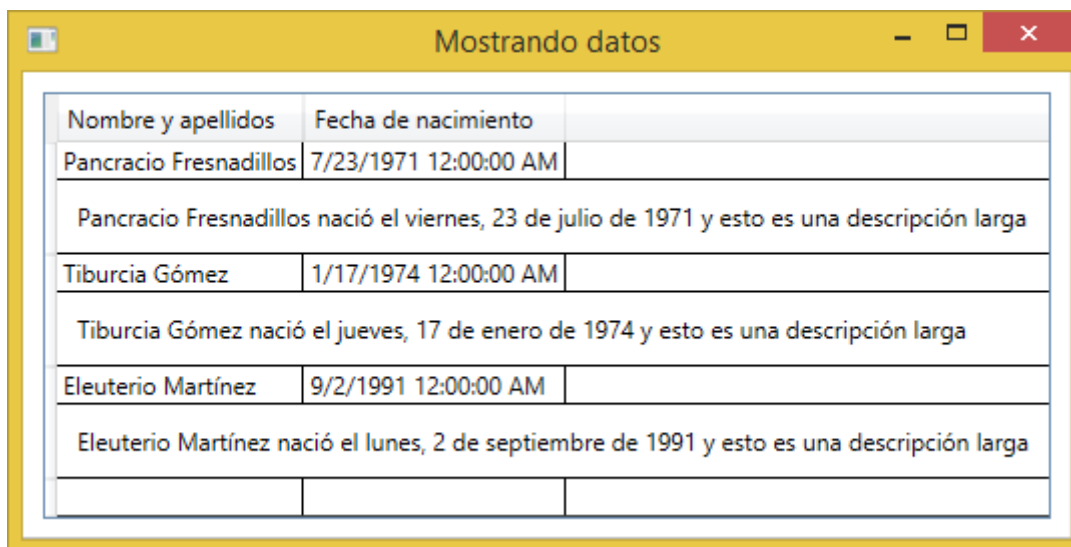
En el XAML, hemos definido un par de columnas y luego usamos **RowDetailsTemplate** para especificar una plantilla para los detalles (que también será de tipo **DataTemplate** y en la cual introducimos los el control correspondiente (en este caso un **TextBlock** con la descripción).

Como se puede ver si ejecutamos el ejemplo, los detalles se muestran bajo la fila escogida. Al cambiar de fila, se esconden los detalles que se estaban mostrando y pasan a mostrarse los de la nueva fila escogida.



Nombre y apellidos	Fecha de nacimiento	
Pancracio Fresnadillos	7/23/1971 12:00:00 AM	
Tiburcia Gómez	1/17/1974 12:00:00 AM	
Tiburcia Gómez nació el jueves, 17 de enero de 1974 y esto es una descripción larga		
Eleuterio Martínez	9/2/1991 12:00:00 AM	

Utilizando la propiedad **RowDetailsVisibilityMode**, podemos cambiar el comportamiento anterior. Por defecto la propiedad vale **VisibleWhenSelected**, donde los detalles sólo se muestran cuando el padre está seleccionado. Podemos cambiar la propiedad a **Visible** (en cuyo caso los detalles se muestran siempre) o **Collapsed** (en cuyo caso los detalles se ocultan siempre). La siguiente captura muestra el primer caso:



Nombre y apellidos	Fecha de nacimiento	
Pancracio Fresnadillos	7/23/1971 12:00:00 AM	
Pancracio Fresnadillos nació el viernes, 23 de julio de 1971 y esto es una descripción larga		
Tiburcia Gómez	1/17/1974 12:00:00 AM	
Tiburcia Gómez nació el jueves, 17 de enero de 1974 y esto es una descripción larga		
Eleuterio Martínez	9/2/1991 12:00:00 AM	
Eleuterio Martínez nació el lunes, 2 de septiembre de 1991 y esto es una descripción larga		

Podemos modificar la plantilla para mostrar los datos que veamos convenientes. A continuación extendemos el ejemplo anterior para dar una idea de las posibilidades que tenemos:

```

<Window x:Class="MostrandoDatos.VentanaPrincipal"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Mostrando datos" SizeToContent="WidthAndHeight">
    <Grid Margin="10">
        <DataGrid Name="dgGente" AutoGenerateColumns="False"
            RowDetailsVisibilityMode="Visible">
            <DataGrid.Columns>
                <DataGridTextColumn Header="Nombre y apellidos"
                    Binding="{Binding Nombre}" />
                <DataGridTextColumn Header="Fecha de nacimiento"
                    Binding="{Binding FechaNacimiento}" />
            </DataGrid.Columns>
            <DataGrid.RowDetailsTemplate>
                <DataTemplate>
                    <DockPanel Background="GhostWhite">
                        <Image DockPanel.Dock="Left" Source="{Binding NombreImagen}"
                            Height="64" Margin="10" />
                        <Grid Margin="0,10">
                            <Grid.ColumnDefinitions>
                                <ColumnDefinition Width="Auto" />
                                <ColumnDefinition Width="*" />
                            </Grid.ColumnDefinitions>
                            <Grid.RowDefinitions>
                                <RowDefinition Height="Auto" />
                                <RowDefinition Height="Auto" />
                                <RowDefinition Height="Auto" />
                            </Grid.RowDefinitions>

                            <TextBlock Text="ID: " FontWeight="Bold" />
                            <TextBlock Text="{Binding Id}" Grid.Column="1" />
                            <TextBlock Text="Nombre: " FontWeight="Bold" Grid.Row="1" />
                            <TextBlock Text="{Binding Nombre}" Grid.Column="1" Grid.Row="1" />
                            <TextBlock Text="Fecha Nacimiento: " FontWeight="Bold" Grid.Row="2" />
                            <TextBlock Text="{Binding FechaNacimiento, StringFormat=d}"
                                Grid.Column="1" Grid.Row="2" />
                        </Grid>
                    </DockPanel>
                </DataTemplate>
            </DataGrid.RowDetailsTemplate>
        </DataGrid>
    </Grid>
</Window>

```

```
namespace MostrandoDatos
{
    public partial class VentanaPrincipal : Window
    {
        public VentanaPrincipal()
        {
            InitializeComponent();

            List<Usuario> users = new List<Usuario>();
            users.Add(new Usuario() { Id = 1, Nombre = "Pancracio Fresnadillos",
            FechaNacimiento = new DateTime(1971, 7, 23), NombreImagen = "pancracio.gif" });
            users.Add(new Usuario() { Id = 2, Nombre = "Tiburcia Gómez",
            FechaNacimiento = new DateTime(1974, 1, 17), NombreImagen="tiburcia.gif" });
            users.Add(new Usuario() { Id = 3, Nombre = "Eleuterio Martínez",
            FechaNacimiento = new DateTime(1991, 9, 2), NombreImagen="eleuterio.gif" });

            dgGente.ItemsSource = users;
        }
    }

    public class Usuario
    {
        public int Id { get; set; }

        public string Nombre { get; set; }

        public DateTime FechaNacimiento { get; set; }

        private string nombreImagen;

        public string NombreImagen {

            get
            {
                return "/MostrandoDatos;component/Images/" + this.nombreImagen;
            }
            set
            {
                this.nombreImagen = value;
            }
        }
    }
}
```

Aquí se muestra el aspecto del **DataGrid**:

Mostrando datos		
Nombre y apellidos	Fecha de nacimiento	
Pancracio Fresnadillos	7/23/1971 12:00:00 AM	
	ID:	1
	Nombre:	Pancracio Fresnadillos
	Fecha Nacimiento:	7/23/1971
Tiburcia Gómez	1/17/1974 12:00:00 AM	
	ID:	2
	Nombre:	Tiburcia Gómez
	Fecha Nacimiento:	1/17/1974
Eleuterio Martínez	9/2/1991 12:00:00 AM	
	ID:	3
	Nombre:	Eleuterio Martínez
	Fecha Nacimiento:	9/2/1991