ACTIVIDAD GUIADA 5
Paneles

Contenidos

Los paneles son uno de los tipos de controles más importantes en WPF. Actúan como contenedores de otros controles y controlan la distribución de las ventanas. Dado que una ventana sólo puede contener ***un control hijo****, un panel habitualmente se utiliza para dividir el espacio en varias áreas, cada una de las cuales puede contener otro panel (que también es un control).*

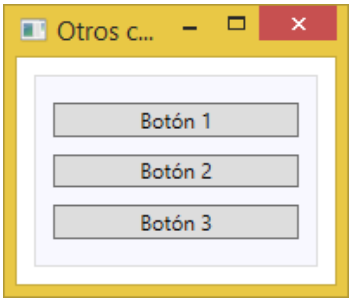
Hay varios tipos de paneles, cada uno con su propio modelo para tratar la distribución y los elementos hijos. Escoger el tipo de panel es importante para conseguir el comportamiento buscado, aunque en ocasiones puede ser un poco complicado.

Bordes y Fondos

Border es un control que nos permite dibujar un borde o un fondo (o ambos) alrededor de otro elemento. Así se vence la restricción de los paneles WPF, que no permiten dibujar un borde alrededor de las esquinas. A continuación se muestra un ejemplo de uso:

```
<Window x:Class="OtrosControles.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="Otros controles" Height="170" Width="200">

    <Grid Margin="10" >
        <Border Background="GhostWhite" BorderBrush="Gainsboro" BorderThickness="1">
            <StackPanel VerticalAlignment="Center" Margin="10">
                <Button>Botón 1</Button>
                <Button Margin="0,10">Botón 2</Button>
                <Button>Botón 3</Button>
            </StackPanel>
        </Border>
    </Grid>
</Window>
```

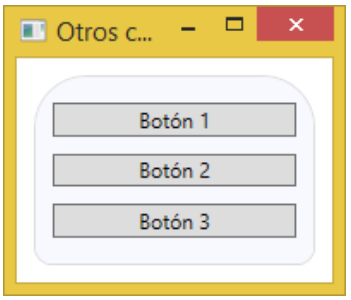
El Border no tiene estilo hasta que definimos un fondo (**Background**) o un borde (color [**BorderBrush**] y grosor [**BorderThickness**]).



Otros aspectos que podemos definir en los bordes (los aplicaremos sobre el ejemplo anterior) son:

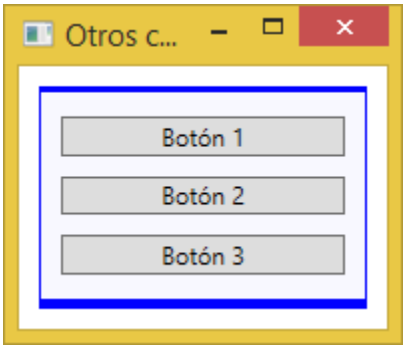
- **Esquinas redondeadas**. Se definen con la propiedad **CornerRadius**, que permiten especificar el radio de curvatura de las esquinas. Podemos especificar un único valor (4 esquinas), dos valores (uno para las esquinas superiores y otro para las inferiores) o 4 valores.

```
<Border Background="GhostWhite" BorderBrush="Gainsboro"
BorderThickness="1" CornerRadius="30,10">
```



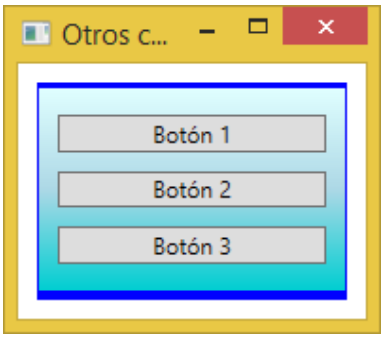
- **Color y grosor de los bordes**. El borde del ejemplo anterior es muy discreto, pero puede cambiar fácilmente regulando el color y/o la anchura. Dado que la propiedad **`BorderThickness`** es de tipo **`Thickness`**, podemos manipular cada anchura del borde de manera individual, de dos en dos o de 4 en 4.

```
<Border Background="GhostWhite" BorderBrush="Blue" BorderThickness="1,3,1,5" >
```



- **Color de fondo**. El color de fondo es de tipo **`Brush`**, lo que abre un montón de posibilidades. Por ejemplo, podemos especificar gradientes. En este caso vamos a especificar un gradiente lineal (**`LinearGradientBrush`**). Sin entrar a detalles de definición del mismo, para entender el control nos basta con saber que podemos definir varias paradas de color, que son colores por los que pasará el gradiente:

```
<Border BorderBrush="Blue" BorderThickness="1,3,1,5" >
<Border.Background>
            <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
                  <GradientStop Color="LightCyan" Offset="0.0"/>
                  <GradientStop Color="LightBlue" Offset="0.5"/>
                  <GradientStop Color="DarkTurquoise" Offset="1"/>
            </LinearGradientBrush>
</Border.Background>
```



Tipos de paneles

A continuación se enuncian las características de los tipos de paneles existentes, para describirlos con más detalle en las secciones siguientes:

- **`Canvas`**

Un panel simple, que imita el modo de funcionamiento de **`WindowsForms`**. Permite asignar coordenadas específicas a cada uno de los controles hijos, dando control total sobre la distribución. Sin embargo, no es un enfoque muy flexible dado que hay que mover manualmente los controles para conseguir la alineación buscada. Sólo debería usarse cuando queremos tener un control muy preciso de la posición de los controles hijos.

- **`WrapPanel`**

El **`WrapPanel`** posiciona cada uno de sus controles hijos al lado del otro, ya sea horizontalmente (por defecto) o verticalmente, hasta que no hay más sitio, en cuyo caso se pasa a la siguiente línea y se continúa. Se utiliza cuando queremos controles en forma de lista (vertical u horizontal) que automáticamente pasa a la línea siguiente cuando no hay más espacio.

- **`StackPanel`**

Actúa de forma similar al **`WrapPanel`**, con la diferencia de que en lugar de pasar automáticamente a la siguiente línea cuando no hay espacio, simplemente se intenta expandir si es posible. Al igual que en el **`WrapPanel`**, la orientación puede ser vertical u horizontal, pero en lugar de ajustar la anchura o altura de los elementos hijos basándonos en el elemento más grande, cada elemento se ajusta para ocupar toda la anchura o altura posible. Se utiliza si queremos una lista de controles que toma todo el espacio posible, sin pasar de línea.

- **`DockPanel`**

Permite acoplar los controles hijos a la parte superior, inferior, izquierda o derecha. Por defecto el último control que se añade, si no se da una posición de acoplamiento concreta, ocupa todo el espacio disponible. Podemos lograr lo mismo con el **`GridPanel`**, pero para comportamientos simples el **`DockPanel`** es más sencillo de usar. Usamos el **`DockPanel`** cuando queremos acoplar uno o varios controles a los lados, por ejemplo para dividir la ventana en áreas específicas.

- **`Grid`**

Es posiblemente el más complicado de los paneles. Un **`Grid`** puede contener varias filas y columnas. Se define una altura para cada una de las filas y una anchura para cada una de las columnas, bien sea en una cantidad absoluta de píxeles, en porcentaje del espacio disponible o en auto, donde la fila o la columna automáticamente ajustan su tamaño dependiendo del contenido. Usamos el **`Grid`** cuando el resto de paneles no nos sirven, y habitualmente se usa en combinación con ellos.

- **`UniformGrid`**

Es similar al **`Grid`**, con la posibilidad de tener varias filas y columnas, pero una diferencia importante: Todas las filas y columnas tienen el mismo tamaño. Lo utilizamos cuando necesitamos el comportamiento del **`Grid`** y todos los elementos tienen que ser iguales.

El Canvas

Es el tipo de panel más simple ya que simplemente permite añadir controles y posicionarlos usando coordenadas explícitas. Aunque parezca cómodo tener control total sobre los elementos hijos, también significa que el panel no hará nada si redimensionamos la ventana o el contenido se escala. El siguiente ejemplo muestra cómo se comporta el **Canvas** por defecto:

```
< Window x:Class="Paneles.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Paneles" Height="100" Width="250">
    <Canvas>
        <Button>Botón 1</Button>
        <Button>Botón 2</Button>
    </Canvas>
</Window>
```

Como se puede ver, aunque tenemos dos botones, se colocan exactamente en el mismo lugar, de modo que sólo el último es visible.

El **Canvas** no hace nada si no damos coordenadas a los controles. Esto se hace usando las propiedades **`Left`**, **`Right`**, **`Top`** y **`Bottom`** del control.

Estas propiedades nos permiten especificar una posición relativa a los bordes del **Canvas**. Por defecto, se establecen a **NaN** (*Not a Number*), que hacen que el **Canvas** los coloque en la esquina superior izquierda. Sin embargo esto es fácil de cambiar, como muestra el siguiente ejemplo:

```
<Window x:Class="Paneles.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Paneles" Height="100" Width="250">

    <Canvas>
        <Button Canvas.Left="10" Canvas.Top="10">Arriba Izquierda</Button>
        <Button Canvas.Left="10" Canvas.Bottom="10">Abajo Izquierda</Button>
        <Button Canvas.Right="10" Canvas.Top="10">Arriba Derecha</Button>
        <Button Canvas.Right="10" Canvas.Bottom="10">Abajo Derecha</Button>
    </Canvas>
</Window>
```

Dado que el **Canvas** nos da control total, no se controla si no hay sitio para albergar los elementos o si se superponen. Esto hace que sea una mala opción para diseñar diálogos, pero el **Canvas**, como su nombre indica (lienzo), se suele utilizar para pintar. WPF añade algunos controles que nos permiten colocarlos en un **Canvas** para hacer ilustraciones.

El WrapPanel

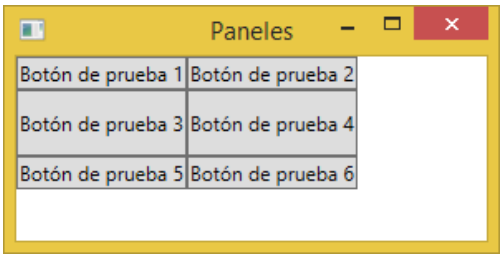
En el **WrapPanel** cada control hijo se posiciona al lado del otro horizontalmente (por defecto) o verticalmente, hasta que no hay más espacio, en cuyo caso se pasa a la siguiente línea y se continúa.

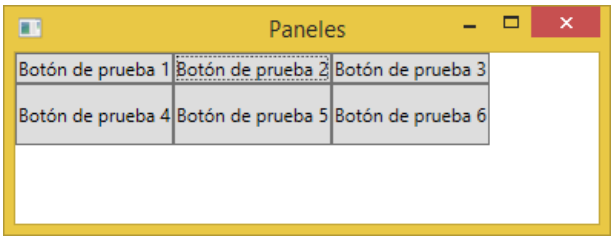
Cuando se usa la orientación **Horizontal**, los controles hijos reciben la misma altura, basándonos en el elemento más alto. Cuando se usa la orientación **Vertical**, los controles hijos tendrán la misma anchura, basándonos en el elemento más ancho.

En el siguiente ejemplo tenemos un **WrapPanel** con la orientación por defecto (Horizontal):

```
<Window x:Class="Paneles.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Paneles" Height="150" Width="300">

    <WrapPanel>
        <Button>Botón de prueba 1</Button>
        <Button>Botón de prueba 2</Button>
        <Button>Botón de prueba 3</Button>
        <Button Height="40">Botón de prueba 4</Button>
        <Button>Botón de prueba 5</Button>
        <Button>Botón de prueba 6</Button>
    </WrapPanel>
</Window>
```

Al especificar una altura determinada para uno de los botones en la segunda fila, el resto de los botones de la fila tienen la misma altura. Si redimensionamos la ventana veremos que el contenido se ajusta automáticamente al ancho de la misma:

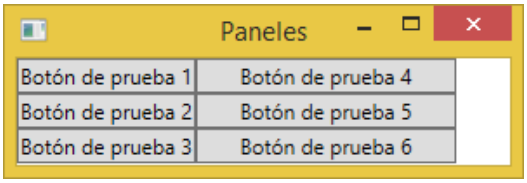


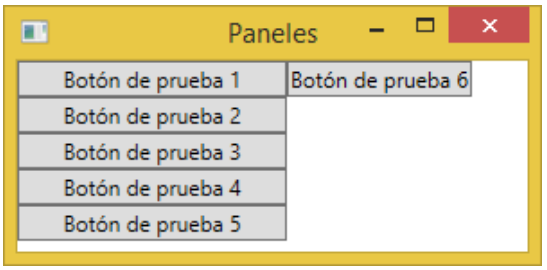


El comportamiento es similar si establecemos la orientación a **Vertical**:

```
<Window x:Class="Paneles.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Paneles" Height="100" Width="300">
    <WrapPanel Orientation="Vertical">
        <Button>Botón de prueba 1</Button>
        <Button>Botón de prueba 2</Button>
        <Button>Botón de prueba 3</Button>
        <Button Width="150">Botón de prueba 4</Button>
        <Button>Botón de prueba 5</Button>
        <Button>Botón de prueba 6</Button>
    </WrapPanel>
</Window>
```

En este caso los botones van verticalmente, y cuando llegan a la parte inferior de la ventana pasan a la siguiente columna. Como podemos ver, al ser el cuarto botón más ancho, el resto de botones de dicha columna reciben la misma anchura. De manera equivalente al ejemplo anterior, si cambiamos la altura de la ventana los botones se adaptan automáticamente:



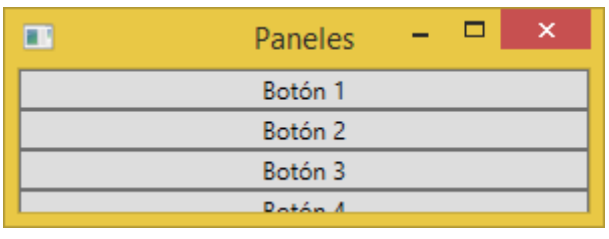


El StackPanel

El **StackPanel** es muy similar al **WrapPanel**, pero con una importante diferencia: El **StackPanel** no adapta el contenido. En su lugar, coloca el contenido en una dirección, permitiéndonos apilar (stack) un elemento tras otro. Veamos en primer lugar un ejemplo simple:

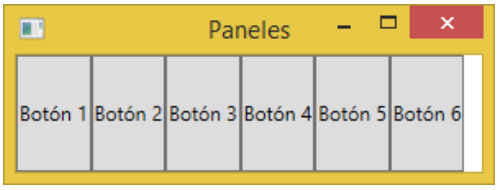
```
<Window x:Class="Paneles.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Paneles" Height="110" Width="300">

    <StackPanel>
        <Button>Botón 1</Button>
        <Button>Botón 2</Button>
        <Button>Botón 3</Button>
        <Button>Botón 4</Button>
        <Button>Botón 5</Button>
        <Button>Botón 6</Button>
    </StackPanel>
</Window>
```

En primer lugar, es conveniente darnos cuenta de que el **StackPanel** no se preocupa de si tenemos suficiente espacio para el contenido. Éste no se adapta, y tampoco se nos ofrece un Scroll (aunque hay un componente denominado **ScrollViewer** que podríamos usar).



También hay que tener en cuenta que la orientación por defecto del **StackPanel** es vertical, al contrario que en el **WrapPanel**, y se cambia usando la propiedad **Orientation**:

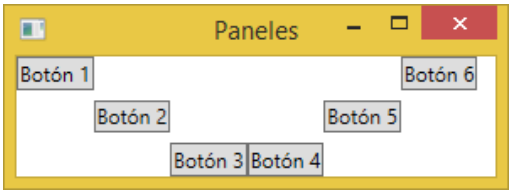
```
<StackPanel Orientation="Horizontal">
```



El **StackPanel** por defecto redimensiona sus controles hijos. En un **StackPanel** alineado verticalmente (como el primer ejemplo), todos los controles hijos se redimensionan horizontalmente para ocupar todo el ancho de pantalla., mientras que en un **StackPanel** orientado horizontalmente los controles se redimensionan verticalmente para ocupar todo el alto de pantalla. El **StackPanel** hace esto estableciendo la propiedad **HorizontalAlignment** o **VerticalAlignment** de sus controles hijos a **Stretch**, si bien es fácil redefinir esta propiedad manualmente.

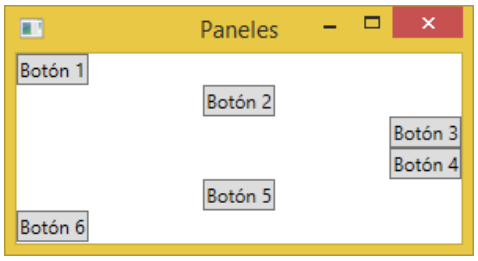
Por ejemplo, en el siguiente ejemplo usamos el mismo marcado que para la captura anterior, pero establecemos la propiedad **VerticalAligment** de los controles hijos, usando los valores **Top**, **Center** y **Bottom** para colocar los botones en un patrón determinado:

```
<StackPanel Orientation="Horizontal">
    <Button VerticalAlignment="Top">Botón 1</Button>
    <Button VerticalAlignment="Center">Botón 2</Button>
    <Button VerticalAlignment="Bottom">Botón 3</Button>
    <Button VerticalAlignment="Bottom">Botón 4</Button>
    <Button VerticalAlignment="Center">Botón 5</Button>
    <Button VerticalAlignment="Top">Botón 6</Button>
</StackPanel>
```



Lo mismo podemos hacer para la orientación vertical usando la propiedad **HorizontalAligment** de los controles hijos:

```
<StackPanel>
    <Button HorizontalAlignment="Left">Botón 1</Button>
    <Button HorizontalAlignment="Center">Botón 2</Button>
    <Button HorizontalAlignment="Right">Botón 3</Button>
    <Button HorizontalAlignment="Right">Botón 4</Button>
    <Button HorizontalAlignment="Center">Botón 5</Button>
    <Button HorizontalAlignment="Left">Botón 6</Button>
</StackPanel>
```

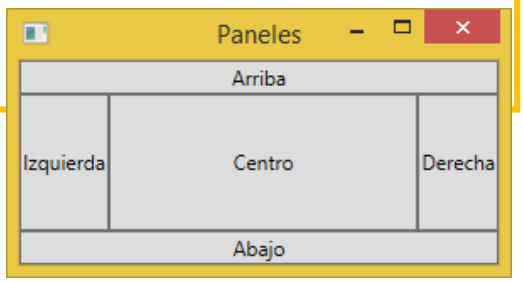


El DockPanel

El **DockPanel** facilita acoplar (dock) contenido en las cuatro direcciones (**Top**, **Bottom**, **Left** y **Right**). Esto lo hace una buena opción en ocasiones en las que queremos dividir la ventana en distintas áreas.

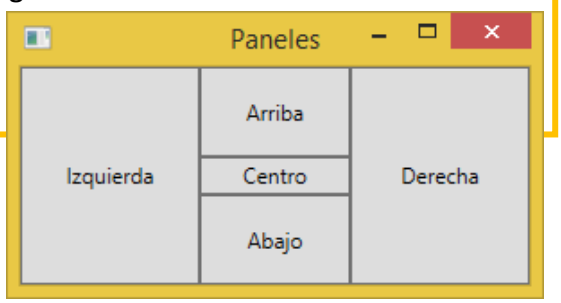
Los controles hijos usan la propiedad **DockPanel.Dock** para decidir en qué dirección se acoplan. Si no lo usamos, el primer control será acoplado hacia la izquierda, mientras que el último control añadido toma todo el espacio que resta (aunque se puede deshabilitar usando la propiedad **LastChildFill)**. Veamos un ejemplo de uso:

```
<Window x:Class="Paneles.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Paneles" Height="160" Width="300">

    <DockPanel >
        <Button DockPanel.Dock="Bottom">Abajo</Button>
        <Button DockPanel.Dock="Top">Arriba</Button>
        <Button DockPanel.Dock="Left">Izquierda</Button>
        <Button DockPanel.Dock="Right">Derecha</Button>
        <Button>Centro</Button>
    </DockPanel>
</Window>
```



El orden en que los botones están definidos también influye. Por ejemplo, los botones de *Arriba y Abajo* toman espacio de los botones de *Izquierda* y *Derecha* porque se definieron antes. Cambiando el orden de los botones cambiaríamos el espacio que se "quitan". Esto se hace en el siguiente ejemplo, donde además establecemos el tamaño de los elementos:

```
<Window x:Class="Paneles.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Paneles" Height="160" Width="300">

    <DockPanel >
        <Button DockPanel.Dock="Left" Width="100">Izquierda</Button>
        <Button DockPanel.Dock="Right" Width="100">Derecha</Button>
        <Button DockPanel.Dock="Bottom" Height="50">Abajo</Button>
        <Button DockPanel.Dock="Top" Height="50">Arriba</Button>
        <Button>Centro</Button>
    </DockPanel>
</Window>
```



El Grid

Un **Grid** (rejilla) puede contener múltiples filas y columnas. Se define una altura para cada fila y una anchura para cada columna, ya sea mediante un valor absoluto en píxeles, un porcentaje del espacio disponible o auto, en cuyo caso la fila o columna automáticamente ajustará el tamaño dependiendo del contenido.

En su forma más básica, el **Grid** simplemente toma todos los controles que le añadimos, los estira para usar el máximo tamaño posible y los coloca uno encima del otro.

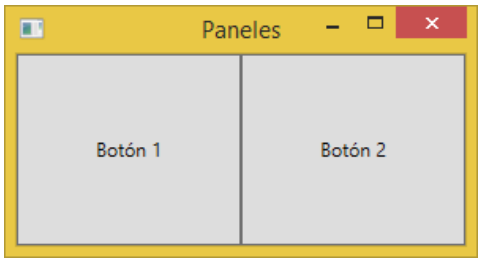
```
<Window x:Class="Paneles.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Paneles" Height="160" Width="300">
    <Grid>
        <Button>Botón 1</Button>
        <Button>Botón 2</Button>
    </Grid>
</Window>
```

Como vemos, el último control toma la posición superior, lo que en este caso implica que no se ve el primer botón, lo que no es demasiado útil.

Para dividir el espacio en filas y columnas usaremos **ColumnDefinitions** y **RowDefinitions**:

```
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <Button>Botón 1</Button>
        <Button Grid.Column="1">Botón 2</Button>
    </Grid>
```

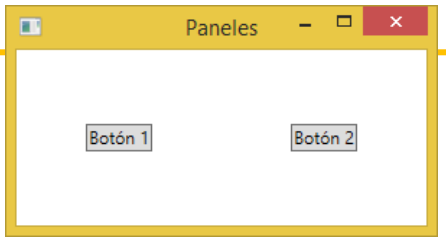
En este ejemplo simplemente hemos dividido el espacio disponible en dos columnas, que dividen el espacio de forma equitativa (lo que se especifica con un *****). En el segundo botón, usamos la propiedad **Grid.Column** para especificar que queremos colocarlo en la segunda columna, teniendo en cuenta que el índice de filas y columnas comienza en 0. Podríamos haber usado la propiedad en la primera columna, lo que no es necesario dado que es donde se coloca por defecto.



Como se puede ver, los controles toman todo el espacio disponible, que es el comportamiento por defecto, Esto se hace estableciendo las propiedades **HorizontalAligment** y **VerticalAligment** de los controles hijos al valor **Stretch**.

En algunos casos queremos que los controles tomen únicamente el espacio que necesitan. Lo más sencillo en estos casos es establecer directamente los valores **HorizontalAligment** y **VerticalAligment** de los controles que queremos manipular. Si en el ejemplo anterior modificamos dichos valores en los botones:

```
<Button
      HorizontalAlignment="Center"
      VerticalAlignment="Center">Botón 1
</Button>
<Button
      HorizontalAlignment="Center"
      VerticalAlignment="Center" Grid.Column="1">Botón 2
</Button>
```

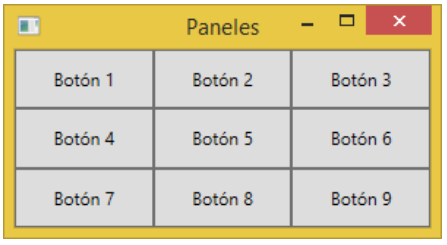


El resultado será que los botones ocupan únicamente el espacio que deberían de acuerdo a su contenido, y en este caso aparecen centrados en la columna correspondiente.

Filas y columnas

El siguiente ejemplo divide el espacio en 3 filas y 3 columnas que ocupan lo mismo. Añadimos un total de 9 botones que vamos añadiendo a la rejilla. Omitimos la posición de los controles que se ubican en la primera fila o la primera columna, pues su valor por defecto es 0.

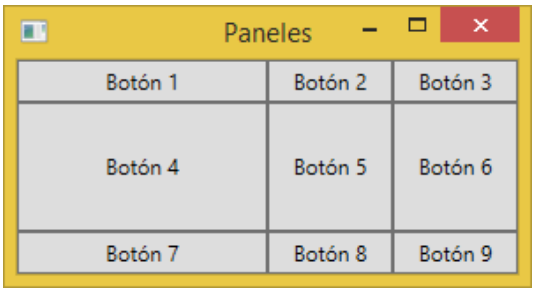
```
<Window x:Class="Paneles.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Paneles" Height="160" Width="300">

    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>

        <Button >Botón 1</Button>
        <Button Grid.Column="1" >Botón 2</Button>
        <Button Grid.Column="2" >Botón 3</Button>
        <Button Grid.Row="1">Botón 4</Button>
        <Button Grid.Column="1" Grid.Row="1">Botón 5</Button>
        <Button Grid.Column="2" Grid.Row="1">Botón 6</Button>
        <Button Grid.Row="2">Botón 7</Button>
        <Button Grid.Row="2" Grid.Column="1">Botón 8</Button>
        <Button Grid.Row="2" Grid.Column="2">Botón 9</Button>
    </Grid>
</Window>
```



Si quisiéramos botones de distinto ancho, podemos usar la siguiente notación para asignar los anchos:

```
<Grid.ColumnDefinitions>
     <ColumnDefinition Width="2*" />
     <ColumnDefinition Width="*" />
     <ColumnDefinition Width="*" />
 </Grid.ColumnDefinitions>
 <Grid.RowDefinitions>
     <RowDefinition Height="*" />
     <RowDefinition Height="3*" />
     <RowDefinition Height="*" />
 </Grid.RowDefinitions>
```

De este modo estamos indicando que los botones de la primera columna ocupan el doble (**2***) que los del resto de columnas. Lo mismo ocurre con los botones de la segunda fila, que ocupan el triple (**3***) que los botones del resto de filas:



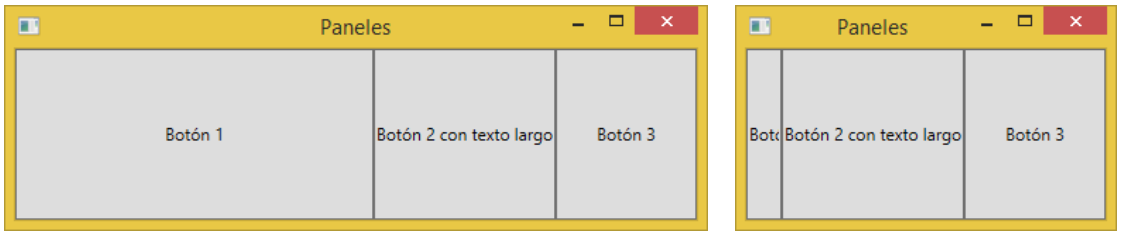
Unidades

Hasta ahora hemos usado como unidad el asterisco (*****), que especifica que una fila o columna debe ocupar el porcentaje que le toca del espacio asignado. Sin embargo hay otros dos modos de especificar el ancho/alto de una fila o columna: Unidades absolutas y Auto anchura/altura. Veamos el siguiente ejemplo donde se combinan ambos:

```
<Window x:Class="Paneles.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Paneles" Height="160" Width="300">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="100" />
        </Grid.ColumnDefinitions>

        <Button >Botón 1</Button>
        <Button Grid.Column="1" >Botón 2 con texto largo</Button>
        <Button Grid.Column="2" >Botón 3</Button>
    </Grid>
</Window>
```

En el ejemplo el primer botón tiene un asterisco (luego ocupa todo el espacio que les sobra al resto), el segundo tiene la anchura establecida a **`Auto`** (lo que implica que tomará el espacio que necesite según su contenido) y el tercero tiene una anchura fija de 100 píxeles.

Si redimensionamos la ventana veremos que quien adapta su ancho es el primer botón:



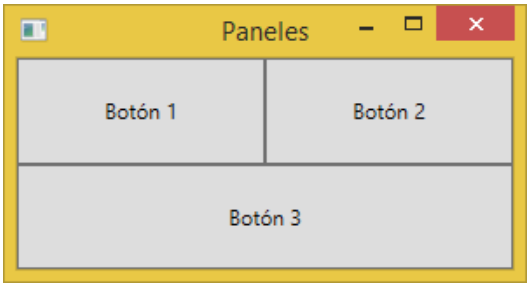
En un **`Grid`** donde varias columnas (o filas) tienen un ancho con *, comparten la anchura/altura no usada por las columnas/filas que usan anchura/altura absoluta o Auto.

Combinaciones

El comportamiento por defecto del **`Grid`** es que cada control toma una celda, pero en ocasiones queremos tomar el control de más de una celda o columna. Esto se puede hacer fácilmente con las propiedades **`ColumnSpan`** y **`RowSpan`**. El valor por defecto de las mismas es 1, pero podemos especificar un número mayor para combinar varias filas o columnas.

En el siguiente ejemplo definimos dos filas y dos columnas, donde el botón 3 ocupa la segunda fila completa:

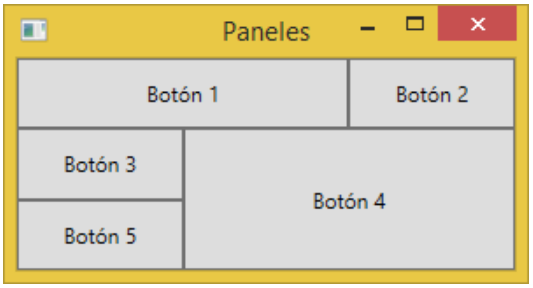
```
<Window x:Class="Paneles.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Paneles" Height="160" Width="300">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>

        <Button >Botón 1</Button>
        <Button Grid.Column="1" >Botón 2</Button>
        <Button Grid.Row="1" Grid.ColumnSpan="2" >Botón 3</Button>
    </Grid>
</Window>
```



Este ejemplo es muy simple (podríamos haber usado una combinación de paneles para llevarlo a cabo), pero en otros casos como el siguiente resulta muy potente.

```
<Window x:Class="Paneles.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Paneles" Height="160" Width="300">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <Button Grid.ColumnSpan="2">Botón 1</Button>
        <Button Grid.Column="3">Botón 2</Button>
        <Button Grid.Row="1">Botón 3</Button>
        <Button      Grid.Column="1"      Grid.Row="1"      Grid.RowSpan="2"
Grid.ColumnSpan="2">Botón 4</Button>
        <Button Grid.Column="0" Grid.Row="2">Botón 5</Button>
    </Grid>
</Window>
```

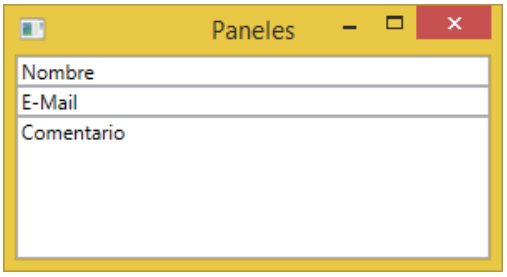
Con 3 filas y 3 columnas normalmente deberíamos tener 9 celdas, pero en este ejemplo usamos una combinación de filas y columnas para tener sólo 9 botones. Un botón puede ocupar varias filas, varias columnas o ambos, como es el caso del cuarto botón.



Utilizando el Grid: Formulario de contacto

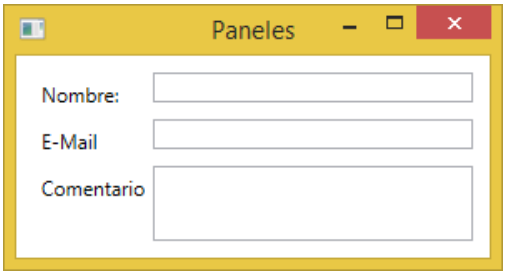
Vamos a aplicar lo visto con un ejemplo real como es un formulario. En primer lugar haremos un formulario simple con tres filas: Dos de ellas con alturas automáticas y la última que ocupará el resto del espacio disponible:

```
<Window x:Class="Paneles.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Paneles" Height="180" Width="300">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <TextBox>Nombre</TextBox>
        <TextBox Grid.Row="1">E-Mail</TextBox>
        <TextBox Grid.Row="2" AcceptsReturn="True">Comentario</TextBox>
    </Grid>
</Window>
```



En una siguiente iteración vamos a añadir etiquetas para designar para que vale cada uno de los campos, en lugar de ponerlo como un texto dentro del **`TextBox`**. Esto implica crear una rejilla de 3 filas x 2 columnas:

```
    <Grid Margin="10">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>

        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>

        <Label>Nombre:</Label>
        <TextBox Grid.Column="1" Margin="0,0,0,10"></TextBox>
        <Label Grid.Row="1">E-Mail</Label>
        <TextBox Grid.Row="1" Grid.Column="1" Margin="0,0,0,10"></TextBox>
        <Label Grid.Row="2">Comentario</Label>
        <TextBox Grid.Row="2"  Grid.Column="2" AcceptsReturn="True" />
    </Grid>
```



Tras analizar el formulario, podríamos querer añadir un botón de *Guardar* que ocupara la parte inferior del formulario. Para ello simplemente añadimos una fila nueva y hacemos que el botón correspondiente ocupe la misma:

```
<Window x:Class="Paneles.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Paneles" Height="180" Width="300">

    <Grid Margin="10">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>

        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>

        <Label>Nombre:</Label>
        <TextBox Grid.Column="1" Margin="0,0,0,10"></TextBox>
        <Label Grid.Row="1">E-Mail</Label>
        <TextBox Grid.Row="1" Grid.Column="1" Margin="0,0,0,10"></TextBox>
        <Label Grid.Row="2">Comentario</Label>
        <TextBox Grid.Row="2"  Grid.Column="2" AcceptsReturn="True"> </TextBox>
        <Button Grid.Row="3" Grid.ColumnSpan="2" Margin="0,10,0,10" >
                Guardar
        </Button>
    </Grid>
</Window>
```