TEMA

Tema 2: Creación de interfaces a partir de documentos XML

Desarrollo de aplicaciones multiplataforma

Desarrollo de interfaces

Autoría: David Forteza

Linkia FP

Tema 2: Creación de interfaces a partir de documentos XML

¿Qué aprenderás?

- Lenguajes de descripción de interfaces basados en XML
- Elementos, etiquetas y atributos XML
- Creación de controles y propiedades en XML
- Eventos y controladores
- Generación de aplicaciones basadas en interfaces XML

¿Sabías que…?

- XAML es la extensión del lenguaje XML utilizada para desarrollar interfaces en .NET

2.1. Introducción al diseño de interfaces basadas en XML

2.1.1. XML y XAML

El XML pertenece a la familia de los lenguajes de marcas. Esos lenguajes se caracterizan por desarrollarse en texto plano mediante palabras clave que luego son interpretadas por algún software en tiempo de ejecución. El más conocido de esos lenguajes probablemente se el HTML que, junto con otros como CSS se utiliza para desarrollar páginas web. En el caso del HTML el software que interpreta el fichero es el navegador web que convierte el código HTML en la página que vemos en nuestro navegador.

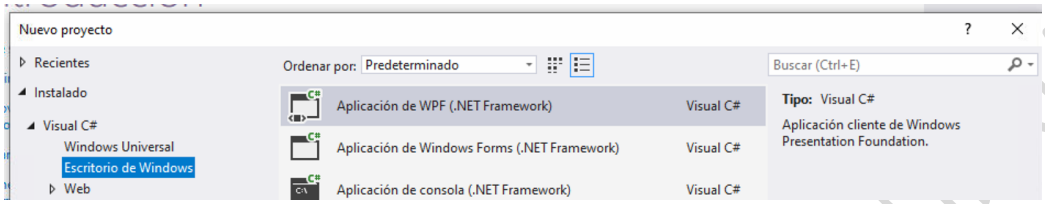
Algo parecido sucede con las interfaces basadas en XML. Partiendo de un fichero en texto plano codificado adecuadamente el entorno de ejecución del cliente interpreta la información contenida en él para renderizar la interfaz correspondiente y gestionar las interacción con el usuario. El XML es una especificación muy amplia de la que derivan subconjuntos específicos para determinados objetivos o entornos. Algo parecido a lo que sucede con el SQL, en el que cada sistema de bases de datos tiene pequeñas variantes para realizar las mismas tareas.

En nuestro caso utilizaremos una variante derivada de XML llamada XAML (**Extensible Application Markup Languaje**) creada para describir interfaces de usuario. XAML es un lenguaje que se puede aplicar al desarrollo de interfaces para escritorio y, además, suele utilizarse para web. Existen una serie de editores que permiten incorporar herramientas de edición y analizadores sintácticos, como pueden ser Visual Studio y Blend, entre otros.

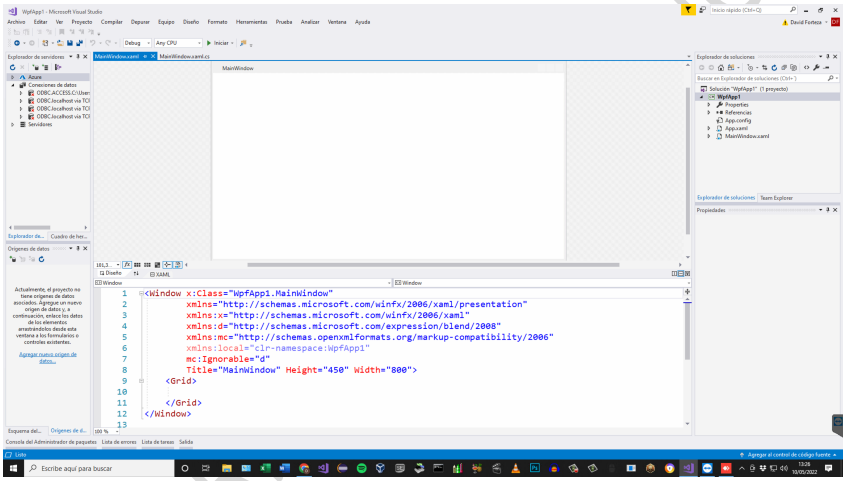
Uno de los principales objetivos que se pretende en el diseño de interfaces que están basadas en XAML es separar totalmente las capas de presentación de la capa lógica para conseguir evitar que se mezclen aquellos elementos que pertenezcan a distintas capas. Esto podría afectar a la distribución modular de la aplicación y al acoplamiento de esta. Entre las principales características de XAML se puede señalar que cada elemento gráfico se define mediante una etiqueta de apertura y otra de cierre, además de por un conjunto de atributos que definirán el aspecto y comportamiento de este. XAML cuenta con bastantes ventajas respecto a sus competidores, sobre todo al permitir desarrollar distintas interfaces mediante su asociación con .Net. En este lenguaje, tanto las etiquetas como los atributos se corresponden de forma directa con otros elementos que pertenecen al lenguaje .Net.

2.1.2. Introducción al desarrollo de interfaces con XAML

Para empezar a entender la mecánica del desarrollo de interfaces utilizando XAML vamos a realizar una interfaz sencilla con la que podremos aprender los elementos que intervienen en el proceso. Lo primero será crear un proyecto en .NET. Esta vez vamos a elegir un proyecto de tipo WPF (**Windows Presentation Foundation**) que es el entorno de desarrollo .NET basado en XAML.



Una vez creado el proyecto nos aparece la ventana inicial como se muestra en la imagen.



Es parecido al entrono usado en el tema anterior solo que esta vez vemos le vista gráfica de la interfaz y una ventana con el código XML en la parte inferior. Si hemos desarrollado en HTML anteriormente sería el equivalente a tener nuestro código HTML en un monitor y el navegador web con nuestra página en otro.

Vamos a echar un vistazo preliminar al código.

```
<Window x:Class="WpfApp1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WpfApp1"
```

```
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>

    </Grid>
</Window>
```

La primera etiqueta (o **Tag** en ingles) que vemos es **Window**. Esa etiqueta define el principio y fin de la definición de nuestra ventana. Podemos observar que en sus argumentos aparece una cadena con el nombre de nuestro proyecto (WpfApp1) seguido de el nombre del fichero XML de la interfaz (**MainWindow**).

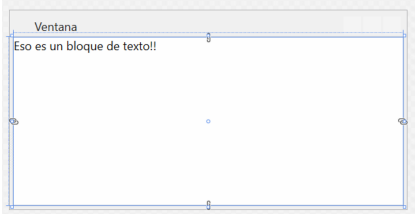
Luego aparecen un conjunto de espacios de nombres (**NameSpaces**) esos espacios de nombres tienen una función parecida a las librerías que importamos al principio en un programa. Luego aparece una etiqueta **Title** con el rotulo de la ventana y sus dimensiones. Y para terminar una etiqueta **Grid** que define la zona dónde vamos a situar nuestros controles. Para empezar, podemos modificar el título de la ventana y sus dimensiones (**Height** y **Width**) y observar cómo los cambios se aplican de forma inmediata en la vista previa de la parte superior.

2.1.2.1. El control TextBlock

Ahora el siguiente paso será añadir nuestro primer control a la ventana. Para ello usaremos el control más simple: un texto. Para ello usaremos la etiqueta **TextBlock**.

```
<Window x:Class="WpfApp1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WpfApp1"
        mc:Ignorable="d"
        Title="Ventana" Height="200" Width="400">
    <Grid>
        <TextBlock>Eso es un bloque de texto!!</TextBlock>
    </Grid>
</Window>
```

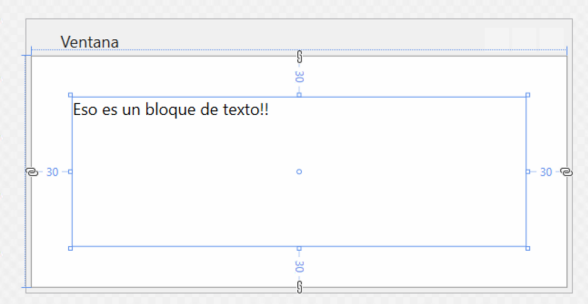
Ahora la ventana tendrá ese aspecto:



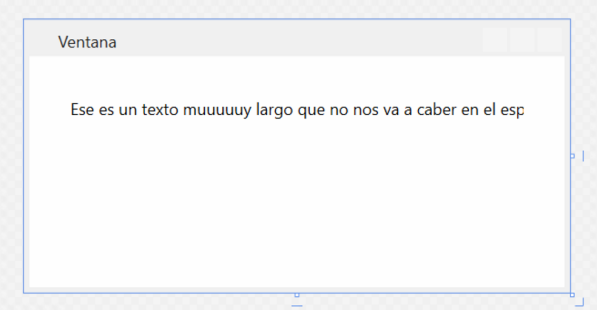
Luego vamos a jugar un poco con las propiedades del bloque de texto para ver como influyen en la forma en la que se muestra en la ventana. Lo primero será modificar la propiedad **Margin** que determina la distancia del bloque de texto con sus vecinos (en ese caso al no haber nada más serán los bordes de la ventana). Vamos a poner el valor 30 y a ver qué ocurre…

```
<Grid>
    <TextBlock Margin="30">Eso es un bloque de texto!!</TextBlock>
</Grid>
```

Vemos que se ha creado un espacio de 30 píxeles alrededor del cuadro de texto como se puede ver en la vista previa del editor.



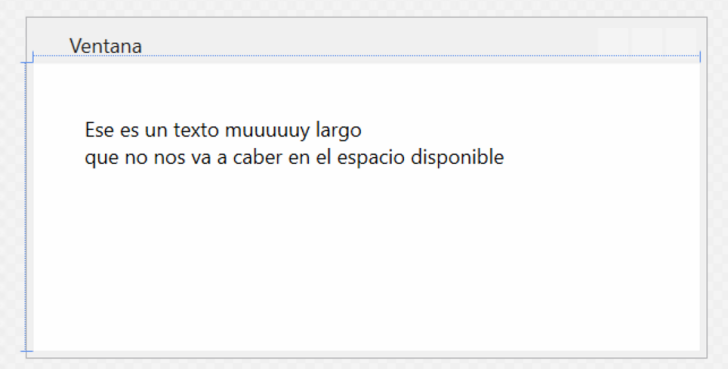
También podemos notar que el cuadro de texto se “expande” para ocupar todo el espacio posible, aunque no sea necesario para mostrar su contenido.




```
<Grid>
    <TextBlock Margin="30">Ese es un texto muuuuuy largo que no
        nos va a caber en el espacio disponible</TextBlock>
</Grid>
```

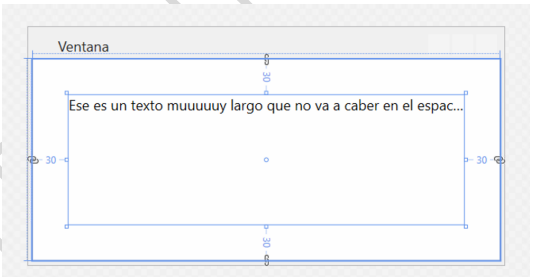
Si el texto es largo y no hay espació suficiente en el **TextBlock** podemos adoptar diferentes estrategias para solucionarlo:

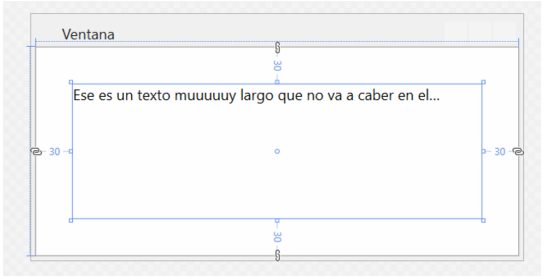
- Usar saltos de línea: Se puede utilizar la marca **<LineBreak/>** para forzar un salto de líne en nuestro texto. Podemos ver en el código anterior que, al igual que ocurre en el HTML los saltos de línea en el código no se traducen en saltos de línea en la vista previa. Hay que usar una marca específica para ello.




```
<Grid>
    <TextBlock Margin="30">Ese es un texto muuuuuy largo<LineBreak/>que no
        nos va a caber en el espacio disponible</TextBlock>
</Grid>
```

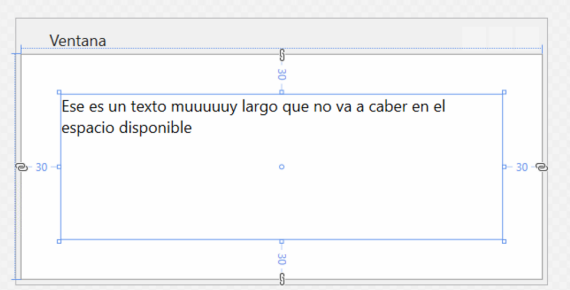
- Usar puntos suspensivos: Se puede utilizar el atributo **TextTrimming** para que aparezcan unos puntos suspensivos al final del mensaje si este es demasiado largo para el espacio disponible. Daremos el valor **WordEllipsis** o **CharacterEllipsis** a la propiedad mencionada según queramos mantener palabras completas o no respectivamente.





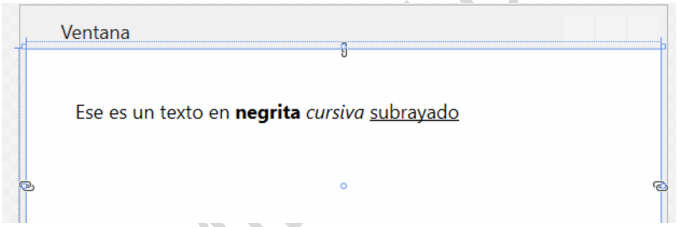

```
<Grid>
    <TextBlock Margin="30" TextTrimming="WordEllipsis">Ese es un texto muuuuuy
    largo que no va a caber en el espacio disponible</TextBlock>
</Grid>
```

- Usar separación de texto: Se puede utilizar el atributo **TextWrapping** con el valor **Wrap** para que el entorno agregue de forma automática saltos de línea cuando sea necesario. Esa suele ser la mejor opción.



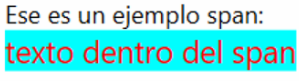

```
<Grid>
  <TextBlock Margin="30" TextWrapping="Wrap">Ese es un texto muuuuuy largo
   que no va a caber en el espacio disponible</TextBlock>
</Grid>
```

Para dar formato al texto (negrita, cursiva,….) lo podemos hacer mediante marcas (igual que en el código HTML).




```
        <TextBlock Margin="30" TextWrapping="Wrap">Ese es un texto en
<Bold>negrita</Bold><Italic>cursiva</Italic><Underline>subrayado</Underline>
        </TextBlock>
```

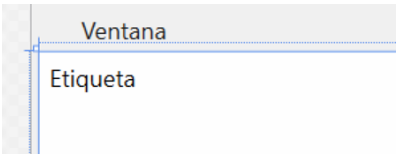
Si deseamos realizar un formato más complejo podemos usar la marca **span**. Esa marca nos premite definir una zona extensa de texto con diferentes formatos aplicados. Disponemos de atributos FontFamily, FontWeight, FontStyle, FontSize, Background, Foreground, TextDecoration,…




```
<TextBlock Margin="30" TextWrapping="Wrap">Ese es un ejemplo span:<LineBreak/>
    <Span FontSize="15" Background="Aqua" Foreground="Red">texto dentro del
span</Span>
</TextBlock>
```

2.1.2.2. El control Label

El control **Label** se parece mucho al **TextBlock** pero tiene algunas diferencias. La más importante es que se puede asociar a otro control de forma que el entorno los agrupa. Eso permite "etiquetar" controles como cajas de texto, listas, …

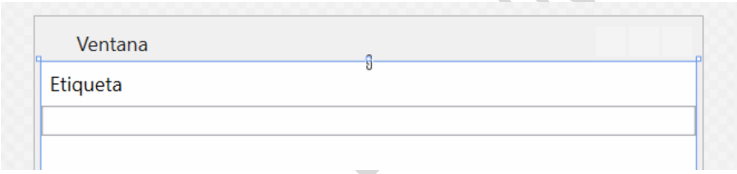


Vemos un ejemplo sencillo:

```
<Grid>
    <Label Content="Etiqueta"/>
</Grid>
```

Vemos que en ese caso el texto forma parte del atributo **Content**.

Ahora vamos a poner un cuadro de texto y asociarlo con la etiqueta.

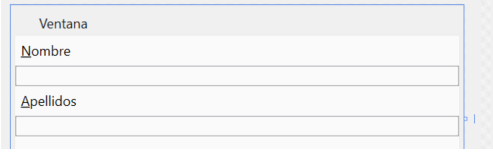



```
<StackPanel>
    <Label Content="Etiqueta" Target="{Binding ElementName=caja}"/>
    <TextBox Name="caja"/>
</StackPanel>
```

En primer lugar, nos fijamos que hemos cambiado el contenedor tipo **Grid** por otro de tipo **StackPanel**. Dedicaremos una sección para hablar de ellos más adelante. Por el momento el enlace entre la etiqueta y el cuadro de texto no parece muy relevante (se puede quitar y no ocurre nada). Sin embargo, ahora veremos su relevancia al definir un atajo de teclado a la etiqueta.

En Windows y otros sistemas operativos también, es una práctica común que tú puedas acceder a controles en un diálogo pulsando la tecla [Alt] y pulsando el carácter que corresponda al control que deseas acceder. El carácter a presionar será sobresaltado cuando tú pulses la tecla [Alt]. El control **TextBlock** no soporta esta funcionalidad, pero el **Label** sí, así que para acceder a controles **Label** es normalmente una excelente opción. Definimos la tecla de acceso poniendo una barra baja (_) antes del carácter.

No tiene por qué ser el primer carácter, puede estar antes de cualquiera de los caracteres en el contenido de tu **Label**. Lo normal es usarlo en el primer carácter que no está siendo usado como tecla de acceso en otro control.




```
<StackPanel>
     <Label Content="_Nombre" Target="{Binding ElementName=nombre}"/>
     <TextBox Name="nombre"/>
     <Label Content="_Apellidos" Target="{Binding ElementName=apellidos}"/>
     <TextBox Name="apellidos"/>
 </StackPanel>
```

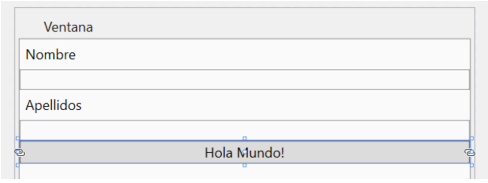
Al ejecutar veremos que el puntero se sitúa en el cuadro de texto correspondiente al pulsar las combinaciones **Alt+N** o **Alt+A**.

2.1.2.3. El control TextBox

Ese es el control más simple para recibir datos del usuario. El valor del atributo Text contiene el texto dentro de la caja y sirve en ambos sentidos: Para introducir un texto en el cuadro o para obtener un texto escrito por el usuario.

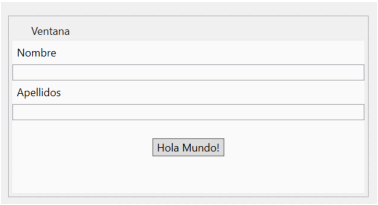
2.1.2.4. El control Button

Ese es el control más simple para activar algún evento. Vamos a añadir un botón a nuestra interfaz y a activar nuestro primer evento (El clásico “Hola Mundo!”).




```
<StackPanel>
     <Label Content="_Nombre" Target="{Binding ElementName=nombre}"/>
     <TextBox Name="nombre"/>
     <Label Content="_Apellidos" Target="{Binding ElementName=apellidos}"/>
     <TextBox Name="apellidos"/>
     <Button >Hola Mundo!</Button>
</StackPanel>
```

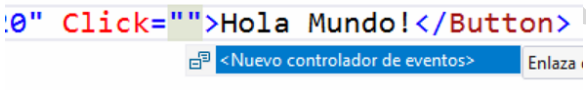
Vemos que el botón ocupa toda la anchura de la ventana y además está “pegado” al cuadro de texto. Eso ocurre porque el contenedor **StackPanel** apila los controle uno encima del otro sin margen ni espacio de separación. Eso se puede arreglar definiendo esos parámetros para nuestro botón.




```
<Button Width="80" Margin="20">Hola Mundo!</Button>
```

Una vez tenemos el botón perfectamente ubicado en la interfaz debemos definir su evento. En ese caso se tratará de un evento de tipo **Click.**

Así que añadimos una propiedad **Click** y al hacerlo aparece un elemento flotante.



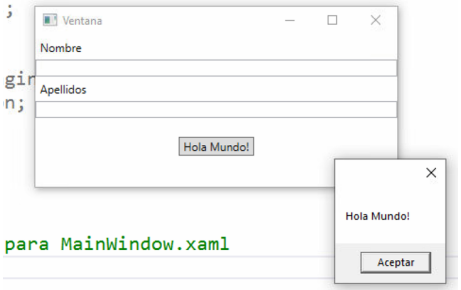

```
<Button Width="80" Margin="20" Click="Button_Click">Hola Mundo!</Button>
```

Si pulsamos en el cuadro flotante se nos genera un evento de forma automática el fichero con extensión **.xaml.cs**

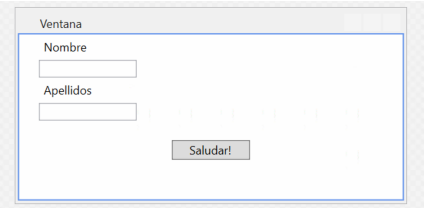
```
private void Button_Click(object sender, RoutedEventArgs e)
{

}
```

Es evidente que con la práctica podemos escribir esas cabeceras directamente si recurrir al autogenerado. Ahora solo nos queda escribir el código del evento y ejecutar.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Hola Mundo!");
}
```



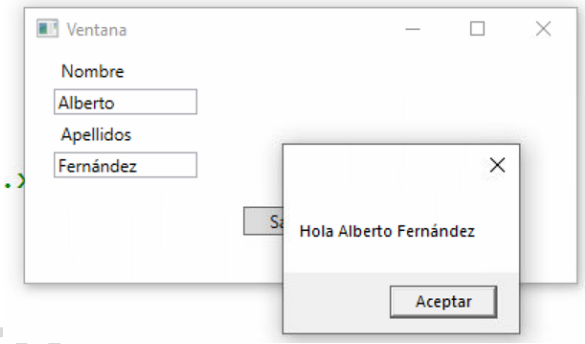
Ahora podemos enlazar el botón con los dos cuadros de texto para que la aplicación nos salude.




```
<StackPanel>
    <Label Content="_Nombre" Margin="20,0" Target="{Binding ElementName=nombre}"/>
    <TextBox Name="nombre" Width="100" Margin="20,0" HorizontalAlignment="Left"/>
    <Label Content="_Apellidos" Margin="20,0" Target="{Binding ElementName=apellidos}"/>
    <TextBox Name="apellidos" Width="100" Margin="20,0" HorizontalAlignment="Left"/>
    <Button Width="80" Margin="20" Click="Button_Click">Saludar!</Button>
</StackPanel>
```

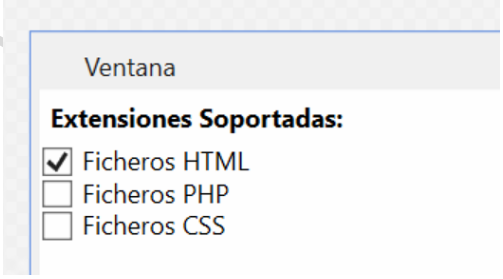
```
    private void Button_Click(object sender, RoutedEventArgs e)
    {
        MessageBox.Show("Hola "+nombre.Text+" "+apellidos.Text);
    }
```

Y al ejecutar tenemos esto:



2.1.2.3. El control CheckBox

Hasta el momento ya hemos aprendido la mecánica básica de uso de controles y eventos en una interfaz XAML. Así que vamos a proceder de una forma más ágil. En el control **CheckBox** la propiedad que contiene su estado se llama **IsChecked** y el evento que se activa cuando el usuario la marca **Checked** o desmarca **unChecked**. Veamos un ejemplo:

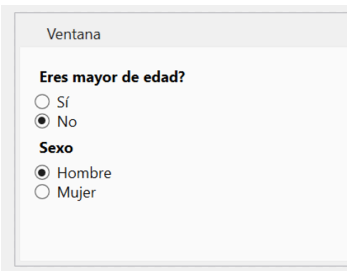


```
 <StackPanel>
    <Label FontWeight="Bold">Extensiones Soportadas:</Label>
    <CheckBox Name="html" IsChecked="True" Checked="html_Checked"
Unchecked="html_Checked">Ficheros HTML</CheckBox>
    <CheckBox Name="php" Checked="php_Checked" Unchecked="php_Checked">Ficheros
PHP</CheckBox>
    <CheckBox Name="css" Checked="css_Checked" Unchecked="css_Checked">Ficheros
CSS</CheckBox>
</StackPanel>
```

```
        private void html_Checked(object sender, RoutedEventArgs e)
        {
            if (html.IsChecked==true) MessageBox.Show("HTML soportado");
            else MessageBox.Show("HTML NO soportado");
        }

        private void php_Checked(object sender, RoutedEventArgs e)
        {
            if (php.IsChecked == true) MessageBox.Show("PHP soportado");
            else MessageBox.Show("PHP NO soportado");
        }

        private void css_Checked(object sender, RoutedEventArgs e)
        {
            if (css.IsChecked == true) MessageBox.Show("CSS soportado");
            else MessageBox.Show("CSS NO soportado");
        }
```

Fijarse que hemos usado el mismo controlador de evento para los eventos de check y uncheck. Y luego en el código hemos usado un condicional para establecer la acción a realizar.

2.1.2.4. El control RadioButton



Este control es casi idéntico al anterior. Aunque como ya sabemos los **RadioButton** se desactivan automáticamente de forma que solo uno puede estar activo a la vez. Para poder crear distintos grupos independientes (y por lo tanto un botón activo por grupo) basta con usar un atributo **GroupName** especificando un nombre de grupo.

Los controles con el mismo nombre de grupo se desactivan mutuamente. Veamos un ejemplo:

```
<Label FontWeight="Bold">Eres mayor de edad?</Label>
<RadioButton GroupName="mayor">Sí</RadioButton>
<RadioButton GroupName="mayor" IsChecked="True">No</RadioButton>
<Label FontWeight="Bold">Sexo</Label>
<RadioButton GroupName="sexo" IsChecked="True">Hombre</RadioButton>
 <RadioButton GroupName="sexo">Mujer</RadioButton>
```

2.1.3. Contenedores, paneles y controles de disposición

Los paneles son uno de los tipos de controles más importantes de WPF. Actúan como contenedores para otros controles, y manejan el layout de tus ventanas. A diferencia de una ventana, que sólo puede contener UN control hijo, un panel es usado comúnmente para dividir el espacio en áreas, donde cada área puede contener un control u otro panel (que es, al fin y al cabo, otro control).

Hay distintos tipos de panel, con cada uno de ellos teniendo su propio modo de lidiar con el diseño y los controles hijos. Elegir el panel correcto es esencial para obtener el comportamiento y el diseño que deseas, y especialmente al inicio este puede ser un trabajo difícil. La próxima sección describirá cada uno de los paneles y te dará una idea de cuándo usarlo.

2.1.3.1. Canvas

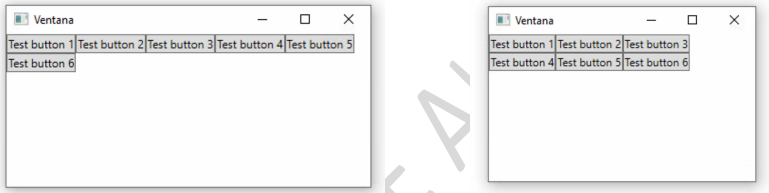
Un panel simple, que imita la forma de hacer las cosas de WinForms. Te permite asignar coordenadas específicas a cada uno de los controles secundarios, lo que te da un control total del diseño. Sin embargo, esto no es muy flexible, porque tienes que mover manualmente los controles secundarios y asegurarte de que se alineen de la forma que deseas. Úsalo (solo) cuando desees un control completo de las posiciones de los controles hijos.


```
<Canvas>
    <Button Canvas.Left="20" Canvas.Top="30">Boton</Button>
</Canvas>
```

2.1.3.2. WrapPanel

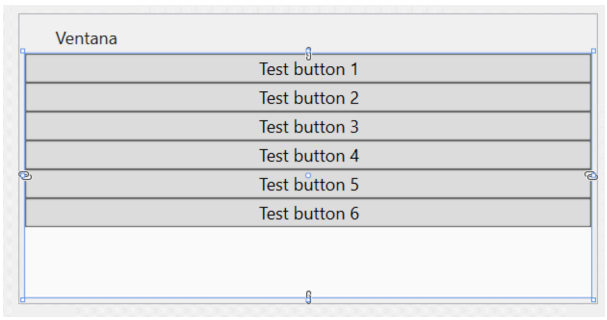
El **WrapPanel** colocará cada uno de sus controles hijos uno junto al otro, horizontalmente (predeterminado) o verticalmente, hasta que no haya más espacio, donde se ajustará a la siguiente línea y luego continuará. Úsalo cuando desees un control de lista vertical u horizontal que se ajusta automáticamente cuando no hay más espacio. La ventaja es que esto se produce de forma dinámica cuando la aplicación está en ejecución creando un entorno "**responsive**".

```
<WrapPanel>
    <Button>Test button 1</Button>
    <Button>Test button 2</Button>
    <Button>Test button 3</Button>
    <Button>Test button 4</Button>
    <Button>Test button 5</Button>
    <Button>Test button 6</Button>
</WrapPanel>
```



2.1.3.3. StackPanel

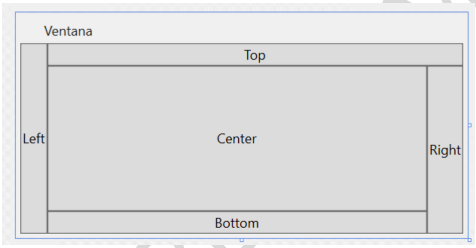
El **StackPanel** actúa de manera muy parecida al **WrapPanel**, pero en lugar de adaptarse si los controles secundarios ocupan demasiado espacio, simplemente se expande, si es posible. Al igual que con **WrapPanel**, la orientación puede ser horizontal o vertical, pero en lugar de ajustar el ancho o la altura de los controles hijos en función del elemento más grande, cada elemento se estira para abarcar todo el ancho o la altura de la ventana. Use el **StackPanel** cuando desee una lista de controles que ocupe todo el cuadrante disponible, sin adaptarlos.



```
<StackPanel>
        <Button>Test button 1</Button>
        <Button>Test button 2</Button>
        <Button>Test button 3</Button>
        <Button>Test button 4</Button>
        <Button>Test button 5</Button>
        <Button>Test button 6</Button>
</StackPanel>
```

2.1.3.4. DockPanel

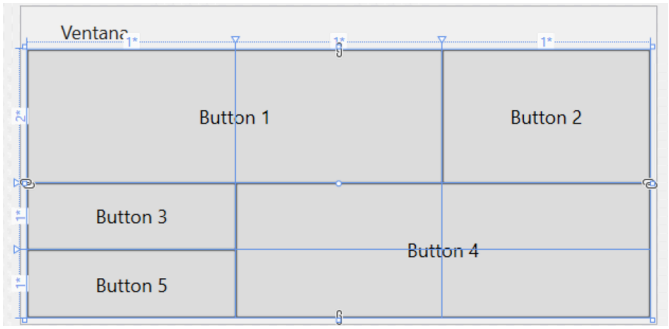
El **DockPanel** te permite acoplar los controles hijos a la parte superior, inferior, izquierda o derecha. Por defecto, el último control, si no tiene una posición de acoplamiento específica, llenará el espacio restante. Puedes lograr lo mismo con el panel Cuadrícula (**Grid**), pero para las situaciones más simples, **DockPanel** será más fácil de usar. Usa **DockPanel** siempre que necesites acoplar uno o varios controles a uno de los lados, como dividir la ventana en áreas específicas.




```
<DockPanel>
    <Button DockPanel.Dock="Left">Left</Button>
    <Button DockPanel.Dock="Top">Top</Button>
    <Button DockPanel.Dock="Right">Right</Button>
    <Button DockPanel.Dock="Bottom">Bottom</Button>
    <Button>Center</Button>
</DockPanel>
```

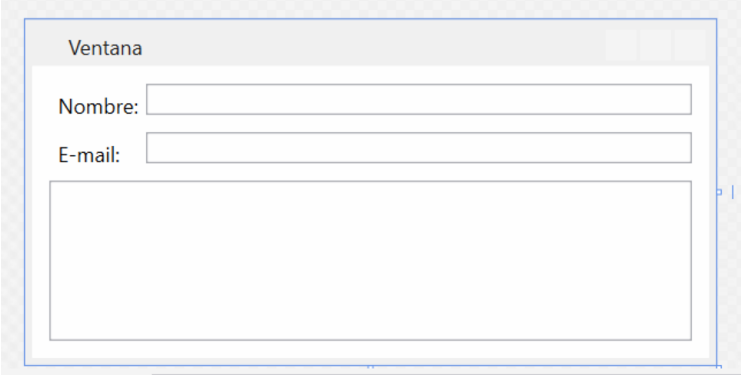
2.1.3.5. Grid

La rejilla (**Grid**) es probablemente el más complejo de los tipos de paneles. Una cuadrícula puede contener múltiples filas y columnas. Defines una altura para cada una de las filas y un ancho para cada una de las columnas, ya sea en una cantidad absoluta de píxeles, en un porcentaje del espacio disponible o como automático, donde la fila o columna ajustará automáticamente su tamaño según el contenido. Usa la cuadrícula cuando los otros paneles no hagan el trabajo, por ejemplo: cuando necesita varias columnas y, a menudo, en combinación con los otros paneles.




```
<Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="2*" />
            <RowDefinition Height="1*" />
            <RowDefinition Height="1*" />
        </Grid.RowDefinitions>
        <Button Grid.ColumnSpan="2">Button 1</Button>
        <Button Grid.Column="3">Button 2</Button>
        <Button Grid.Row="1">Button 3</Button>
        <Button Grid.Column="1" Grid.Row="1" Grid.RowSpan="2"
Grid.ColumnSpan="2">Button 4</Button>
        <Button Grid.Column="0" Grid.Row="2">Button 5</Button>
</Grid>
```

Podemos utilizar lo que hemos visto con anterioridad para crear una ventana con un formulario de contacto para pedir información.



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

Conceptos clave

- **Lenguaje XAML:** En este tema, hemos aprendido crear una interfaz usando el lenguaje de marcas XAML.
- **Eventos:** También hemos aprendido a enlazar el código C# que controla los eventos con los controles de la interfaz
- **Paneles:** Hemos aprendido a utilizar los distintos tipos de paneles para colocar nuestros controles en la ventana de la forma más adecuada.

Test de autoevaluación

1. Cuál es la variante de XML creada para describir interfaces de usuario
    a. CSS.
    b. HTML.
    c. XAML.
    d. UML.

2. Cuál es el entorno de .NET que permite trabajar con XAML
    a. ASP
    b. WPF
    c. C#
    d. NET Framework

3. Como se define un control en lenguaje XML
    a. Usando etiquetas y atributos
    b. Referenciando su librería
    c. Creando una instancia de clase
    d. Se añade desde la paleta de controles al formulario

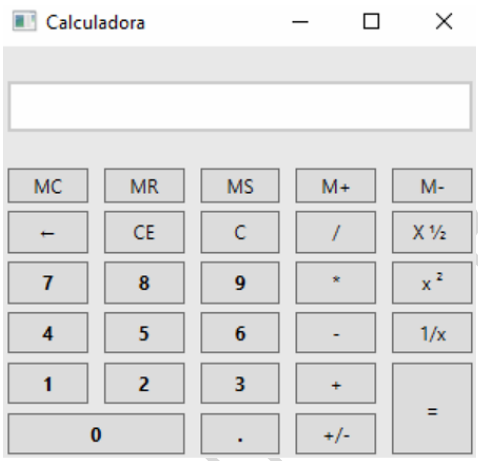
4. Cuál de los siguientes elementos no es un contenedor
    a. Canvas
    b. Grid
    c. StackPanel
    d. DataGrid

5. Cuál de los siguientes atributos sirve para manejar textos largos
    a. TextTrimming
    b. Margin
    c. Width
    d. Length

6. Qué atributo debemos usar para vincular una etiqueta con su control
    a. Target
    b. Binding
    c. Control
    d. bonded

Ponlo en práctica

Vamos a crear una interfaz para una calculadora usando XML. Pero solamente la interfaz sin definir eventos ni código. La interfaz deberá tener un aspecto lo más parecido al que se muestra en la imagen.



Solucionarios

Test de autoevaluación

1. Cuál es la variante de XML creada para describir interfaces de usuario
    a. CSS.
    b. HTML.
    c. **XAML.**
    d. UML.

2. Cuál es el entorno de .NET que permite trabajar con XAML
    a. ASP
    b. **WPF**
    c. C#
    d. NET Framework

3. Como se define un control en lenguaje XML
    a. **Usando etiquetas y atributos**
    b. Referenciando su librería
    c. Creando una instancia de clase
    d. Se añade desde la paleta de controles al formulario

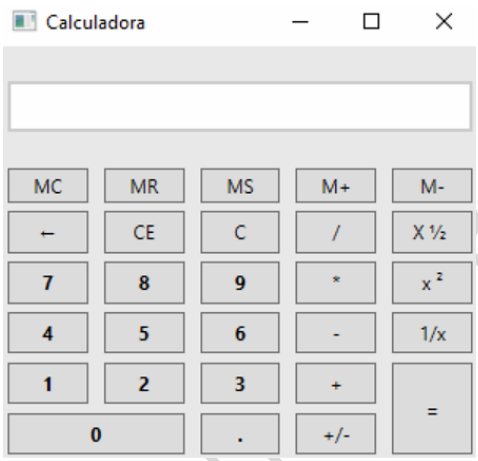
4. Cuál de los siguientes elementos no es un contenedor
    a. Canvas
    b. Grid
    c. StackPanel
    d. **DataGrid**

5. Cuál de los siguientes atributos sirve para manejar textos largos
    a. **TextTrimming**
    b. Margin
    c. Width
    d. Length

6. Qué atributo debemos usar para vincular una etiqueta con su control
    a. **Target**
    b. Binding
    c. Control
    d. bonded

Ponlo en práctica

Vamos a crear una interfaz para una calculadora usando XML. Pero solamente la interfaz sin definir eventos ni código. La interfaz deberá tener un aspecto lo más parecido al que se muestra en la imagen.




```
<Window x:Class="WpfApp1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WpfApp1"
        mc:Ignorable="d"
        Title="Calculadora" Height="300" Width="320">

    <Grid Name="MainGrid" Background="#E8E8E8">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="0.1*"/>
            <!-- Espaciado izquierdo. -->
            <ColumnDefinition Width="1.0*"/>
            <ColumnDefinition Width="0.2*"/>
            <!-- Espaciado Col 1-2  -->
            <ColumnDefinition Width="1.0*"/>
            <ColumnDefinition Width="0.2*"/>
            <!-- Espaciado Col 2-3  -->
            <ColumnDefinition Width="1.0*"/>
            <ColumnDefinition Width="0.2*"/>
            <!-- Espaciado Col 3-4 . -->
            <ColumnDefinition Width="1.0*"/>
            <ColumnDefinition Width="0.2*"/>
            <!-- Espaciado Col 4-5 -->
            <ColumnDefinition Width="1.0*"/>
```

```
            <ColumnDefinition Width="0.1*"/>
            <!-- Espaciado derecho. -->
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="22"/>
            <!-- espacio sobre pantalla -->
            <RowDefinition Height="32"/>
            <!-- Espacio pantalla -->
            <RowDefinition Height="18"/>
            <!-- Espacio bajo pantalla -->
            <RowDefinition Height="0.2*"/>
            <RowDefinition Height="0.8*"/>
            <RowDefinition Height="0.2*"/>
            <RowDefinition Height="1.0*"/>
            <RowDefinition Height="0.2*"/>
            <RowDefinition Height="1.0*"/>
            <RowDefinition Height="0.2*"/>
            <RowDefinition Height="1.0*"/>
            <RowDefinition Height="0.2*"/>
            <RowDefinition Height="1.0*"/>
            <RowDefinition Height="0.2*"/>
            <RowDefinition Height="1.0*"/>
            <!-- Espaciado teclas -->
            <RowDefinition Height="0.1*"/>
            <!-- Espaciado inferior -->
        </Grid.RowDefinitions>

        <TextBox Name="Entry" Grid.Row="1" Grid.Column="1" Grid.ColumnSpan="9"
                 TextAlignment="Right" FontSize="18" BorderThickness="2"
BorderBrush="#CCCCCC">
            <TextBox.Text >
                <Binding Path="CurrentValue" Mode="TwoWay"
                   UpdateSourceTrigger="PropertyChanged" />
            </TextBox.Text>
        </TextBox>

        <TextBlock Name="Memory" Grid.Row="2"  Grid.Column="1" Text="" />
```

```
        <TextBlock Name="Info" Grid.Row="2"  Grid.Column="3" Text=""
Grid.ColumnSpan="7" TextAlignment="Right" />
        <Button Name="MemoryClear" Grid.Row="4" Grid.Column="1" Content="MC" />
        <Button Name="MemoryRecall"  Grid.Row="4" Grid.Column="3" Content="MR" />
        <Button Name="MemorySave" Grid.Row="4" Grid.Column="5" Content="MS" />
        <Button Name="MemoryAdd" Grid.Row="4" Grid.Column="7" Content="M+" />
        <Button Name="MemorySubstract" Grid.Row="4" Grid.Column="9" Content="M-" />
        <Button Name="EditDeleteLast" Grid.Row="6" Grid.Column="1" Content="←" />
        <Button Name="EditClearEntry" Grid.Row="6" Grid.Column="3" Content="CE" />
        <Button Name="EditClearAll" Grid.Row="6" Grid.Column="5" Content="C" />
        <Button Name="CalculateDivide"  Grid.Row="6"  Grid.Column="7" Content="/"
/>
        <Button Name="CalculateSqrt"  Grid.Row="6"  Grid.Column="9" Content="X ½ "
/>
        <Button Name="NumberSeven" Grid.Row="8"  Grid.Column="1" >
            <TextBlock Text="7"  FontWeight="Bold" />
        </Button>
        <Button Name="NumberEight"  Grid.Row="8"  Grid.Column="3" >
            <TextBlock Text="8"  FontWeight="Bold" />
        </Button>
        <Button Name="NumberNine"  Grid.Row="8"  Grid.Column="5" >
            <TextBlock Text="9"  FontWeight="Bold" />
        </Button>
        <Button Name="CalculateMultiply" Grid.Row="8"  Grid.Column="7" Content="*"
/>
        <Button Name="CalculateSquare"  Grid.Row="8"  Grid.Column="9" Content="x ²"
/>
        <Button Name="NumberFour" Grid.Row="10" Grid.Column="1" >
            <TextBlock Text="4"  FontWeight="Bold" />
        </Button>
        <Button Name="NumberFife" Grid.Row="10" Grid.Column="3" >
            <TextBlock Text="5"  FontWeight="Bold" />
        </Button>
        <Button Name="NumberSix" Grid.Row="10" Grid.Column="5" >
            <TextBlock Text="6"  FontWeight="Bold" />
        </Button>
        <Button Name="CalculateSubtract"  Grid.Row="10" Grid.Column="7" Content="-"
/>
```

```
        <Button Name="CalculateReciprocal"  Grid.Row="10" Grid.Column="9"
Content="1/x" />
        <Button Name="NumberOne"  Grid.Row="12" Grid.Column="1" >
            <TextBlock Text="1"  FontWeight="Bold" />
        </Button>
        <Button Name="NumberTwo" Grid.Row="12" Grid.Column="3" >
            <TextBlock Text="2"  FontWeight="Bold" />
        </Button>
        <Button Name="NumberThree" Grid.Row="12" Grid.Column="5" >
            <TextBlock Text="3"  FontWeight="Bold" />
        </Button>
        <Button Name="CalculateAdd" Grid.Row="12" Grid.Column="7" Content="+" />
        <Button Name="CalculateResult" Grid.Row="12" Grid.Column="9" Content="="
Grid.RowSpan="3" />
        <Button Name="NumberZero" Grid.Row="14" Grid.Column="1" Grid.ColumnSpan="3"
>
            <TextBlock Text="0"  FontWeight="Bold" />
        </Button>
        <Button Name="NumberDecimalDot" Grid.Row="14" Grid.Column="5" >
            <TextBlock Text="."  FontWeight="Bold" />
        </Button>
        <Button Name="CalculateNegate" Grid.Row="14" Grid.Column="7" Content="+/-"
/>
    </Grid>
      </Window>
```

Los solucionarios están disponibles en la versión interactiva del aula.