Argentina $8,90 (recargo al interior $0,20) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

20

XAML

Características generales de XAML
Ejemplos / Controles simples

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Windows Presentation Foundation

Principales Namespaces / Herencia de clases / Objetos "freezables"

ISBN 978-987-1347-43-8
00020
9 789871 347438

RedUSERS

COMUNIDAD DE TECNOLOGIA

¡SUMATE!

EL SITIO Nº1 DE TECNOLOGIA

Noticias al instante // Entrevistas y coberturas exclusivas // Análisis y opinión de los máximos referentes // Reviews de productos // Trucos para mejorar la productividad // Registrate, participa, y comparte tus opiniones

SUSCRIBITE

SIN CARGO A CUALQUIERA DE NUESTROS **NEWSLETTERS** Y RECIBÍ EN TU CORREO ELECTRÓNICO TODA LA INFORMACIÓN DEL UNIVERSO TECNOLÓGICO ACTUALIZADA **AL INSTANTE**

INGRESÁ A
redusers.com/suscribirse-al-newsletter
¡Y REGISTRATE YA!

Foros · Encuestas · Tutoriales · Agenda de eventos · Videos · ¡Y mucho más!

redusers.com

Seguinos en:
www.facebook.com/redusers
www.twitter.com/redusers
www.youtube.com/redusersvideos

Elementos básicos

En el ejemplo que presentamos a continuación revisaremos la creación de elementos básicos para interfaz de usuario basada en WPF. Tenemos una lista de tabuladores (menús) dentro de los cuales podemos navegar y mostrar la información para cada tipo de menú; el menú que vamos a definir para el siguiente ejemplo es el correspondiente a botones de tipo formulario:

```
<TabItem Header="Buttons">
  <StackPanel  Orientation="Vertical">
    <Button VerticalAlignment="Top" Name=
    "btn" >
      Click inline!
    </Button>

    <StackPanel Orientation="Horizontal">
      <Button HorizontalAlignment="Center"
      VerticalAlignment="Center" Margin=
      "5">Button</Button>
      <RadioButton HorizontalAlignment=
      "Center" VerticalAlignment="Center"
      Margin="5">RadioButton</RadioButton>
      <CheckBox HorizontalAlignment="Center"
      VerticalAlignment="Center" Margin=
      "5">CheckBox</CheckBox>
    </StackPanel>

    <StackPanel Orientation="Horizontal">
      <Button HorizontalAlignment="Center"
      VerticalAlignment="Center" Margin="5">
        <StackPanel Orientation="Horizontal">
          <Line VerticalAlignment="Center"
          Y1="5" Y2="5" X2="12" Stroke
          Thickness="8" StrokeEndLineCap=
          "Triangle" Stroke="Black" />
          <TextBlock>Button</TextBlock>
          <Line VerticalAlignment="Center"
          Y1="5" Y2="5" X1="6" X2="18"
          StrokeThickness="8" StrokeStart
          LineCap="Triangle" Stroke="Green" />
        </StackPanel>
      </Button>

      <RadioButton HorizontalAlignment=
      "Center" VerticalAlignment="Center"
      Margin="5">
        <TextBlock Background="Red"
        Foreground="White" FontSize="16">
          RadioButton
        </TextBlock>
      </RadioButton>

      <CheckBox HorizontalAlignment="Center"
      VerticalAlignment="Center" Margin="5">
        <Grid>
          <Viewbox>
            <Path Stroke="RoyalBlue" Stroke
            Thickness="8" Data="M 8,8 L
            75,25 M 75,8 L 8, 25" />
          </Viewbox>
          <TextBlock HorizontalAlignment=
          "Center" VerticalAlignment=
          "Center" FontSize="18">
            CheckBox
          </TextBlock>
        </Grid>
      </CheckBox>
    </StackPanel>
```

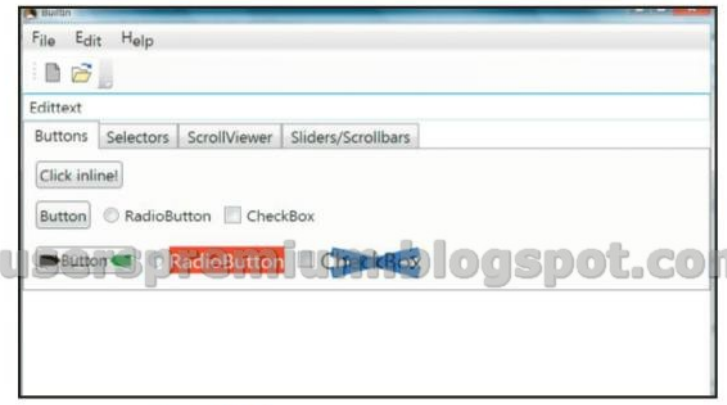


FIGURA 011 | Aplicar estilos a los diferentes elementos de ingreso de datos para formularios.

```
    </StackPanel>
</TabItem>
```

Si deseamos aumentar una opción del menú Nueva, únicamente debemos agregar otro nodo dentro del XAML con la etiqueta **<TabItem></TabItem>**. En esa etiqueta podemos observar los elementos **Button**, **RadioButton** y **CheckBox**, y los diferentes estilos, colores y tamaños que podemos aplicarles, tal como se muestra en la Figura 11.

Dibujar un elemento

Revisemos cómo podemos dibujar un elemento –en este caso, un botón– y visualizarlo de tal manera que sea posible agregarle un gráfico y redibujarlo para que la presentación de la aplicación sea perfecta:

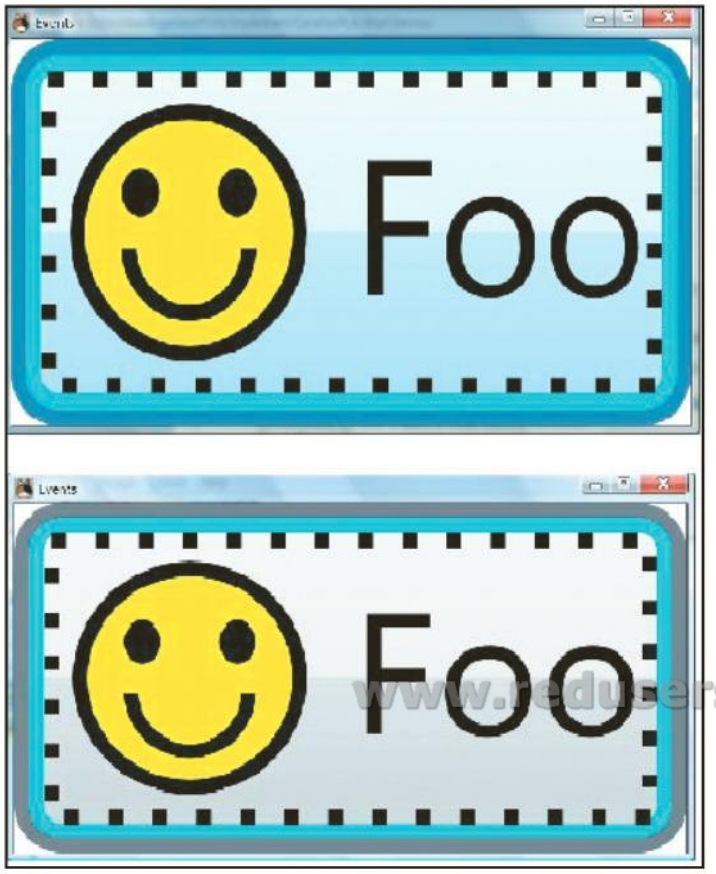


FIGURA 012 | Redibujar un elemento con XAML y code-behind.

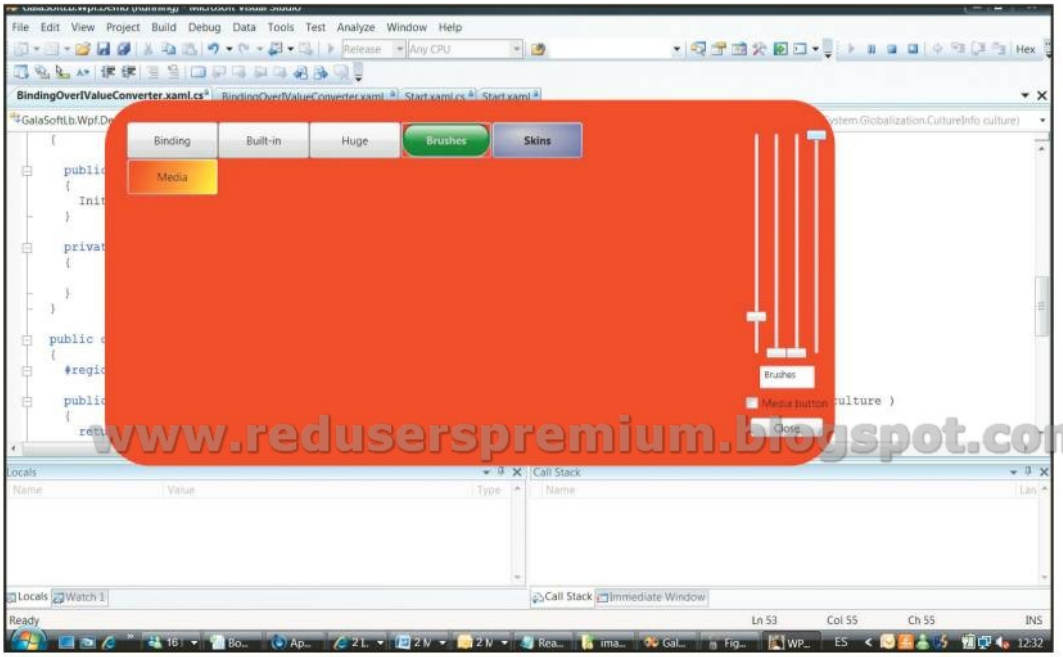
```
<Window x:Class="GalaSoftLb.Wpf.Demo.
ControlsHuge"
    xmlns="http://schemas.microsoft.com/
    winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/
    winfx/2006/xaml"
    Title="Events"
    SizeToContent="WidthAndHeight"
    Icon="GalaSoft.ico">
<Grid>
    <Grid.LayoutTransform>
      <ScaleTransform ScaleX="13" ScaleY=
      "13"/>
    </Grid.LayoutTransform>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="Auto" />
    </Grid.ColumnDefinitions>

    <Button MouseLeftButtonDown="MouseButton
    DownButton"
            PreviewMouseLeftButtonDown=
            "PreviewMouseButtonDownButton">

      <Grid MouseLeftButtonDown=
      "MouseButtonDownGrid"
            PreviewMouseLeftButtonDown=
            "PreviewMouseButtonDownGrid">

        <Grid.ColumnDefinitions>
          <ColumnDefinition />
          <ColumnDefinition />
        </Grid.ColumnDefinitions>

        <Canvas MouseLeftButtonDown=
        "MouseButtonDownCanvas"
PreviewMouseLeft
ButtonDown=
```

```
"PreviewMouse
                        ButtonDownCanvas"
                        Width="20"
                        Height="18"
                        VerticalAlignment=
                        "Center">

    <Ellipse MouseLeftButtonDown="Mouse
    ButtonDownEllipse"
    PreviewMouseLeftButtonDown=
    "PreviewMouseButtonDownEllipse"
                        Canvas.Left="1"
                        Canvas.Top=
                        "1" Width="16"
                        Height="16"

                        Fill="Yellow"
                        Stroke="Black" />
    <Ellipse Canvas.Left="4.5" Canvas.Top="5"
    Width="2.5" Height="3"

                        Fill="Black" />
    <Ellipse Canvas.Left="11" Canvas.Top="5"
    Width="2.5" Height="3"

                                Fill="Black" />
    <Path Data="M 5,10 A 3,3 0 0 0 13,10"
    Stroke="Black" />

        </Canvas>

  <TextBlock Grid.Column="1">Foo</TextBlock>

      </Grid>

    </Button>

  </Grid>
</Window>
```

El único evento en el que hemos ingresado código para obtener el efecto de cambiar el color del botón cuando el mouse está sobre él es el especificado a continuación:

```
private void MouseButtonDownEllipse(object
sender, RoutedEventArgs e)
{
  Debug.WriteLine("MouseButtonDownEllipse");
  #region
```

FIGURA 013 | Aplicación sin el efecto de opacidad o con opacidad a un 100%.



> Cualquier solución que construyamos sobre WPF puede abrirse en Visual Studio como en cualquier producto Expression.

```
    e.Handled = true;
    #endregion
}
```

Con este ejemplo podemos tener dos visualizaciones del botón: cuando el mouse esté sobre el objeto y cuando no lo esté. El redibujado de dicho elemento es totalmente transparente y, a nivel de interfaz gráfica, es perceptible sólo por el cambio del color sobre el botón.

Opacar un elemento

El efecto de opacar un objeto para ver los elementos que están por detrás es muy sencillo de realizar mediante WPF. A través de este ejemplo, podemos lograr la visibilidad presentada por las ventanas en Windows Vista al momento de utilizar el componente de **AeroGlass**:

```
<Slider Name="sliderOpacity" Orientation=
"Vertical" Minimum="0.1" Maximum="1" Value=
"1" ToolTip="Opacity" HorizontalAlignment=
"Center" />
```

Incluyendo esta simple línea en XAML, podemos obtener en nuestras aplicaciones el efecto que se observa en las Figuras 13 y 14. Es importante mencionar que cualquier tipo de solución que construyamos sobre WPF puede abrirse tanto en Visual Studio como en cualquier producto de Microsoft Expression, sin necesidad de tener un convertidor. La idea es que tanto los desarrolladores como los diseñadores trabajen sobre el mismo producto con sus diferentes herramientas, sin temor de dañar la tarea hecha por los demás.

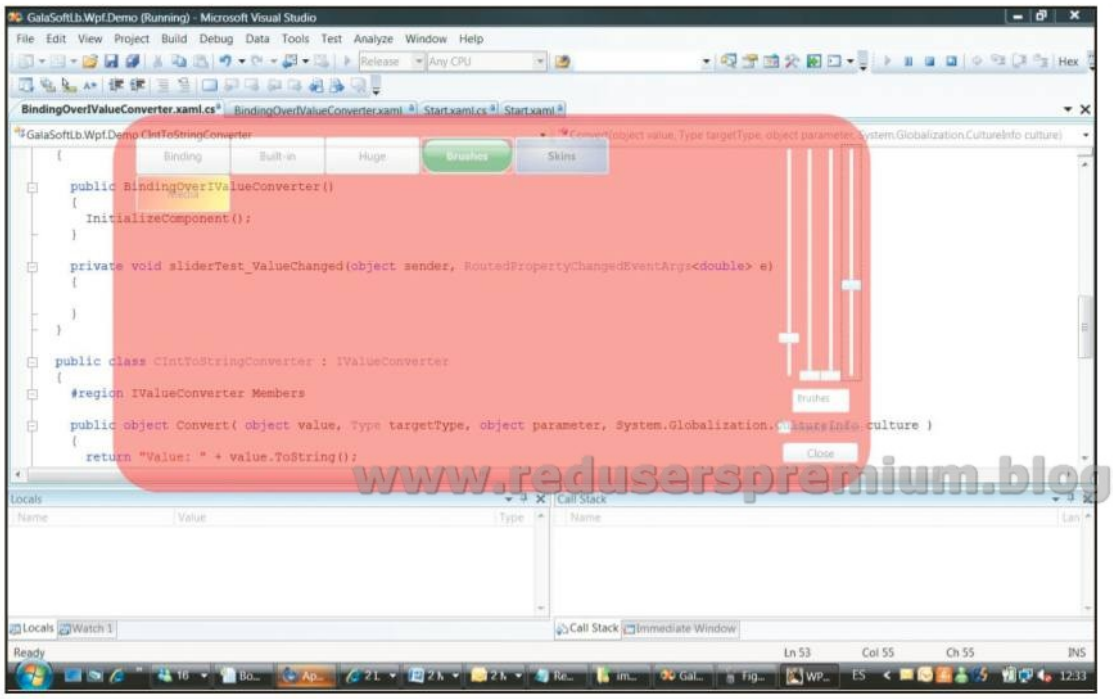


FIGURA 014 | Opacidad al 60% para la aplicación.

Controles simples

WPF nos provee de una serie de controles. Haremos referencia a los que usaremos con mayor frecuencia.

Existen diferentes tipos de controles en WPF, y aun cuando la mayoría es utilizada para tener interacción con el usuario, algunos permiten organizar el posicionamiento de los controles, realizar tareas en segundo plano, incrustar video, etc. Cuando creamos un nuevo proyecto del tipo **Windows Application (WPF)**, Visual Studio 2008 nos presentará en la barra de herramientas los diferentes controles de WPF agrupados en las siguientes categorías:

- Controles comunes
- Contenedores comunes
- Menús y barras de herramientas
- Todos los controles
- Todos los contenedores

Controles comunes

Como su nombre lo indica, son los controles más utilizados al momento de desarrollar una aplicación:

- Grid
- Button
- CheckBox
- ComboBox
- Image
- InkCanvas
- InkPresenter
- Label
- ListBox
- PasswordBox
- RadioButton
- Slider
- TextBox

En este apartado sólo haremos referencia a Grid, Label, Textbox, ComboBox, RadioButton e Image, ya que son los controles que exponen una funcionalidad típica en una aplicación.

Grid

El control Grid pertenece a la categoría **contenedores comunes**, y como cualquier otro de su mismo grupo, permite definir el posicionamiento de los diferentes controles que se encuentren dentro de elemento **Window**. Si no definiéramos un control de Layout, perderíamos la posibilidad de representar gráficamente varios controles, como lo demuestra el siguiente ejemplo:

XAML:

```
<Window x:Class="Users_SimpleControls.
Window1"
    xmlns="http://schemas.microsoft.com/
    winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/
    winfx/2006/xaml"
    Title="Users_SimpleControls" Height="300"
    Width="300"
    >
  <Button Height="200" Width="200">Hola
  </Button>
</Window>
```


En este caso, se está creando una ventana con un alto y un ancho de 300 píxeles y, directamente como un elemento hijo, se le está agregando un control del tipo Button.
Sin embargo, si quisiéramos añadir un segundo botón, el diseñador marcaría un error, ya que

el elemento **Window** no está preparado para posicionar dos controles diferentes a ese mismo nivel.

Grid, como mencionamos anteriormente, nos permitirá organizar y desplegar nuestros controles pero en formato de cuadrícula (filas y columnas). En la Tabla 4, podemos observar los principales elementos del control Grid, y en la 5, sus propiedades más relevantes.

En el siguiente ejemplo, crearemos un control Grid que contenga 2 columnas de 150 píxeles de ancho cada una y 12 filas. El grid mostrará las líneas de división así como un color de fondo blanco:

```
<Window x:Class="Users_SimpleControls.
Window1"
    xmlns="http://schemas.microsoft.com/
    winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/
    winfx/2006/xaml"
    Title="Users_SimpleControls" Height="400"
    Width="300"
    >
  <Grid ShowGridLines="True" Background=
  "White">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="150"/>
      <ColumnDefinition Width="150"/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition/>
      <RowDefinition/>
      <RowDefinition/>
      <RowDefinition/>
      <RowDefinition/>
      <RowDefinition/>
      <RowDefinition/>
      <RowDefinition/>
      <RowDefinition/>
      <RowDefinition/>
      <RowDefinition/>
      <RowDefinition/>
    </Grid.RowDefinitions>

  </Grid>
</Window>
```

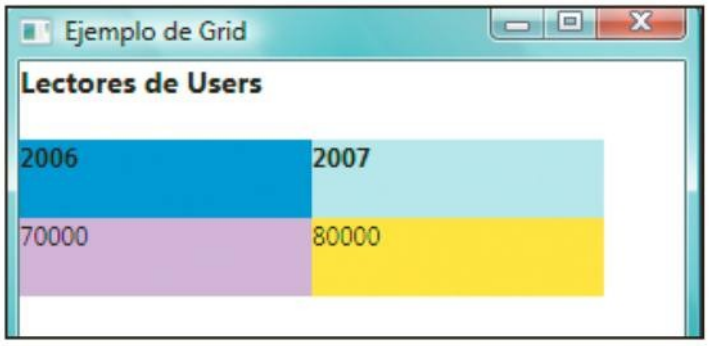


FIGURA 015 | Utilizando el diseñador de XAML en Visual Studio 2008, podremos minimizar los errores en el desarrollo.

Con la creación de las filas y las columnas especificamos el layout que deberá cumplir la ventana que estamos diseñando. Lo que haremos a continuación será indicar a los controles en qué posición de nuestro grid serán ubicados; es decir, en qué fila y columna se desplegarán. Todos

Tabla 4 | Principales elementos del control Grid

Elemento	Descripción
ColumDefinition	Permite definir las diferentes columnas que contendrá el grid, así como las propiedades de cada columna, alto, ancho, etc.
RowDefinition	Cada elemento RowDefinition creado especifica una fila dentro del grid. También permite determinar el comportamiento de cada fila, como tamaño máximo, mínimo, etc.

los controles cuentan que las propiedades Grid.Colum y Grid.Row, que indican en qué columna y en qué fila se posicionará el control. Partiendo del ejemplo anterior, crearemos un elemento Label (el control será explicando más adelante), que será ubicado en la columna 0 fila 2:

```
<Label Grid.Column="0" Grid.Row="2">Apellido
Materno</Label>
```

Si no se especifica ningún valor de dimensión (Height o Width) en el control de Grid, ni en su elemento Grid.ColumnDefinitions, entonces significa que el control ocupará el 100% del tamaño de la ventana. En caso de que las dimensiones de la ventana cambien, Grid reajustará su tamaño para adaptarse de forma automática.

Label

El control **Label** permite desplegar texto y posicionarlo en la región que especifiquemos. Sus principales propiedades se describen en la Tabla 6.

En el siguiente ejemplo crearemos una etiqueta que desplegaá el texto "Apellido Materno", y le especificaremos un borde de 2 puntos de ancho de color rojo:

```
<Label Grid.Column="0" Grid.Row="2" Content=
 "Apellido Materno" BorderBrush="Red">
            <Label.BorderThickness>2</Label.
            BorderThickness>
     </Label>
```

TextBox

Permite capturar la información en texto plano por parte del usuario. Sus principales propiedades se describen en la Tabla 7.

En el siguiente ejemplo creamos un control TextBox con un color de fondo Azure, el texto "Casador", y la comprobación gramatical activada:

```
<TextBox Grid.Column="1" Grid.Row="2"
SpellCheck.IsEnabled="True" Background=
"Azure">Casador</TextBox>
```

El corrector gramatical marcará con una línea en rojo la palabra "Casador". Al hacer

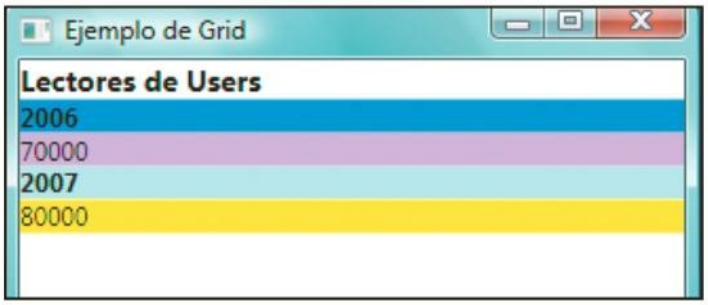


FIGURA 016 | Control Label posicionado dentro del control Grid.

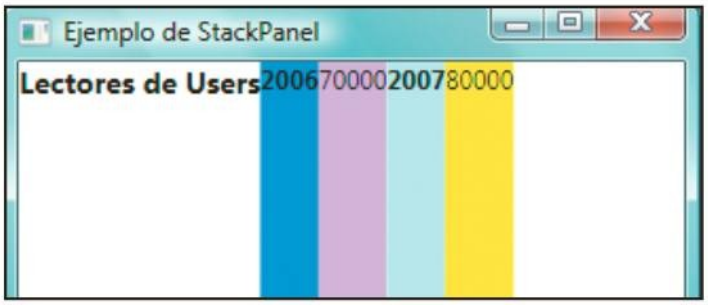


FIGURA 017 | Control Label con sus propiedades establecidas.

Tabla 5 | Principales propiedades del control Grid

Elemento	Descripción
ShowGridLines	Indica si se mostrarán las líneas del grid; de manera predeterminada es false.
BackGround	Especifica un color de fondo.

El control Grid podría ser el control más utilizado al momento de realizar el Layout de nuestra ventana, ya que permite posicionar los controles como si se tratara de una tabla.

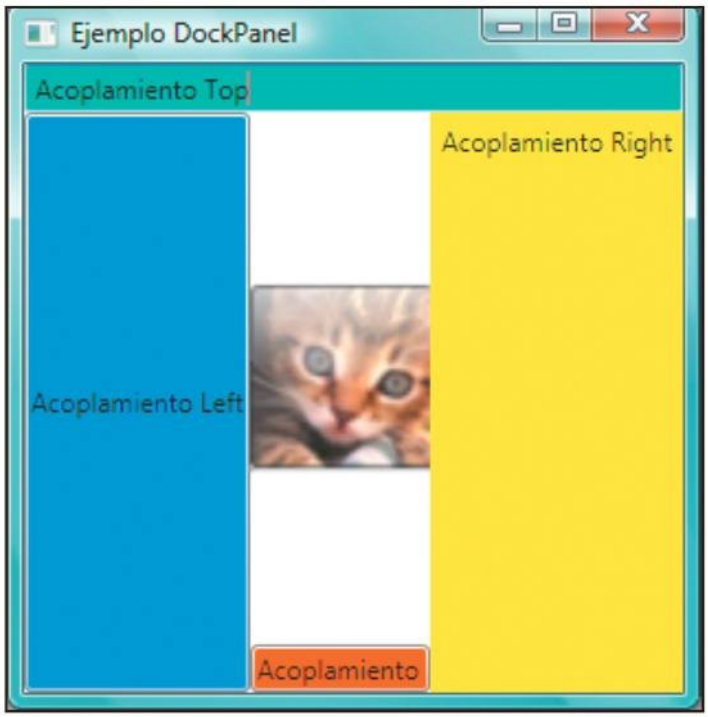


FIGURA 018 | Algunos controles contienen propiedades que nos permiten, por ejemplo, comprobar la gramática mientras se escribe.

un clic con el botón derecho, se muestra un menú contextual sugiriendo la palabra correcta: "Cazador".

ComboBox

Permite desplegar información en forma de lista seleccionable. Cada uno de estos elementos tiene un nombre asignado, el cual sirve para conocer cuál es el valor que el usuario ha seleccionado. En el siguiente ejemplo, crearemos un ComboBox con el nombre "cboRangoEdad", que contendrá tres elementos del tipo ComboBoxItem; cada uno de ellos corresponde a un rango de edad, al último se le asignará un color de fondo verde:

```
<ComboBox Grid.Column="1" Grid.Row="3"
Name="cboRangoEdad">
        <ComboBoxItem  Name="Rango1">18 a 30
        años</ComboBoxItem>
        <ComboBoxItem  Name="Rango2">31 a 50
        años</ComboBoxItem>
        <ComboBoxItem  Name="Rango3"
        Background="Green">50 en adelante
        </ComboBoxItem>
    </ComboBox>
```

En tiempo de diseño, nos asociaremos al evento SelectionChanged, que se ejecutará cuando el usuario seleccione un elemento dentro del

Tabla 6 | Principales propiedades del control Label

Propiedad	Descripción
Content	Asigna el texto que será mostrado dentro del control.
Background	Especifica un color de fondo.
Label.BorderThickness	Asigna el ancho del borde de la etiqueta.
BorderBrush	Especifica el tipo de brocha que se utilizará para dibujar el borde.
Height	Especifica el alto de la etiqueta.
Width	Asigna el ancho de la etiqueta.

ComboBox. Primero, crearemos el código que se ejecutará cuando el usuario seleccione un elemento:

```
void cboRangoEdad_SelectionChanged(object
sender, SelectionChangedEventArgs e)
        {
            ComboBoxItem elemento
            Seleccionado = cboRangoEdad.
            SelectedItem as ComboBoxItem;
            string mensaje = string.Format
            ("El valor seleccionado es '{0}'
            con el texto '{1}'",elemento
            Seleccionado.Name,elemento
            Seleccionado.Content);
            MessageBox.Show(mensaje);
        }
```

El código anterior mostrará un mensaje con el nombre y el valor del elemento seleccionado. La propiedad SelectItem es del tipo object, por lo cual es necesario convertirla al tipo ComboBoxItem y, de esa manera, obtener la información de dicho objeto.

Lo único que falta hacer, es asociar la función cboRangoEdad_SelectionChanged con el evento SelectionChanged:

```
<ComboBox Grid.Column="1" Grid.Row="3" Name=
"cboRangoEdad" SelectionChanged="cboRango
Edad_SelectionChanged">
```

RadioButton

El control RadioButton permite al usuario seleccionar un valor de manera excluyente dentro de un grupo de elementos; es decir, sólo permite la selección de un valor dentro de un determinado grupo. ComboBox y RadioButton comparten el mismo objetivo; sin embargo, el primero se presenta como una lista desplegable y el segundo se muestra con el comportamiento de un botón, con lo cual se logra una experiencia de usuario diferente.
En el siguiente ejemplo crearemos dos controles del tipo RadioButton, que estarán asociados utilizando la propiedad GroupName. Si el nombre

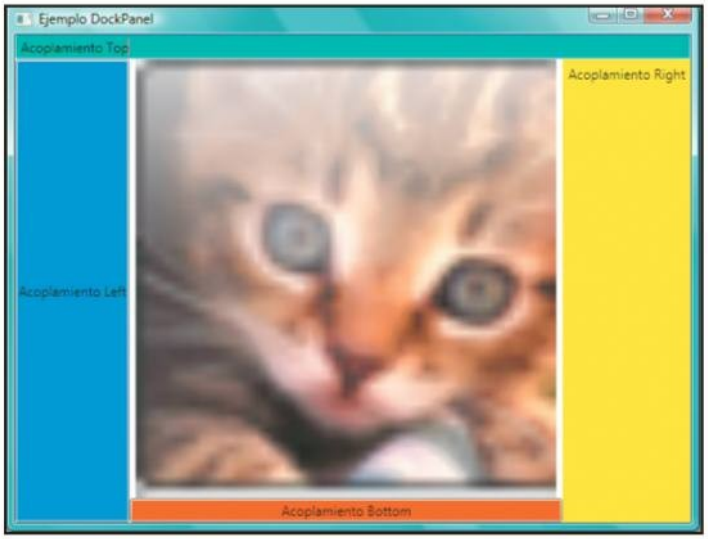


FIGURA 019 | Programa en ejecución. Hacemos clic sobre el ComboBox para desplegar la información.

Tabla 7 | Principales propiedades del control TextBox

Propiedad	Descripción
Content	Asigna u obtiene el texto mostrado dentro del control.
Background	Especifica un color de fondo.
SpellCheck.IsEnabled	Permite la comprobación de errores gramaticales dentro del control.
ContextMenu	Asigna un menú contextual personalizado.
Height	Especifica el alto de la etiqueta.
Width	Asigna el ancho de la etiqueta.

El control ComboBox permite crear sus elementos a partir de la información almacenada en la base de datos. Para mayor referencia, podemos buscar el uso de las propiedades ItemsSource, DataContext, DisplayMemberPath y SelectedValuePath.

del grupo no es establecido o es diferente, entonces los controles no quedarán asociados, y el usuario podrá seleccionar ambos a la vez:

```
<RadioButton Grid.Column="1" Grid.Row="4"
Name="rdEstadoCivilSoltero" GroupName=
"EstadoCivil">Soltero</RadioButton>
  <RadioButton Grid.Column="1" Grid.Row="5"
  Name="rdEstadoCivilCasado" GroupName=
  "EstadoCivil">Casado</RadioButton>
```

En caso de que necesitemos conocer cuál fue el control RadioButton que seleccionó el usuario, podemos asociarnos al evento Checked, el cual es ejecutado en el momento en que el usuario selecciona un valor. Para implementarlo, podemos realizar el siguiente procedimiento.

Crearemos una función que será llamada por ambos RadioButtons (rdEstadoCivilSoltero y rdEstadoCivilCasado). Ésta recibe como parámetro el sender, que es el objeto que fue seleccionado. Como es del tipo object, es necesario convertirlo al tipo RadioButton, para poder acceder a su información:

```
void EstadoCivilSeleccionado(object sender,
RoutedEventArgs e)
{
  RadioButton objetoSeleccionado = sender as
  RadioButton;
  MessageBox.Show(" El valor seleccionado fue "
  + objetoSeleccionado.Content);
}
```

El siguiente paso es asociar la función EstadoCivilSeleccionado al evento Checked de ambos controles:

```
<RadioButton Grid.Column="1" Grid.Row="4"
Name="rdEstadoCivilSoltero" GroupName=
"EstadoCivil" Checked="EstadoCivil
Seleccionado">Soltero</RadioButton>
  <RadioButton Grid.Column="1" Grid.Row="5"
  Name="rdEstadoCivilCasado" GroupName=
  "EstadoCivil" Checked="EstadoCivil
  Seleccionado">Casado</RadioButton>
```

FIGURA 020 | Cuando seleccionamos un elemento, se invoca el evento SelectionChanged.

Image

El control Image permite visualizar imágenes que se encuentren en cualquiera de los siguientes formatos, y sus principales propiedades son descriptas en la Tabla 8:

- BMP
- JPEG

- PNG
- TIFF
- Windows Media Photo
- GIF
- ICON

En el siguiente ejemplo mostraremos una imagen GIF que tiene transparencia:

```
<Label Grid.Column="0" Grid.Row="6"  >Imagen
</Label>
        <Image  Grid.Column="1" Grid.Row="6"
        Source="logoMS.gif" Width="300"
        Height="35"/>
```

Para que la imagen se muestre de manera correcta, es necesario aplicar un color de fondo. En este caso, utilizaremos un control de tipo **Canvas**, al cual le aplicaremos un color de fondo azul y, por último, agregaremos el control **Image** como un control hijo del anterior:

```
<Label Grid.Column="0" Grid.Row="6"  >Imagen
</Label>
        <Image  Grid.Column="1" Grid.Row="6"
        Source="logoMS.gif" Width="300"
        Height="35"/>
```

En la propiedad source del control Image es posible especificar una imagen que se encuentre directamente en Internet, por ejemplo:

```
<Image source="http://dominio.com/
imagen.jpg/>
```

El corrector de ortografía es soportado en los controles TextBox y RichTextBox.

FIGURA 021 | Cuando un valor es seleccionado, se mostrará un mensaje con la información correspondiente.

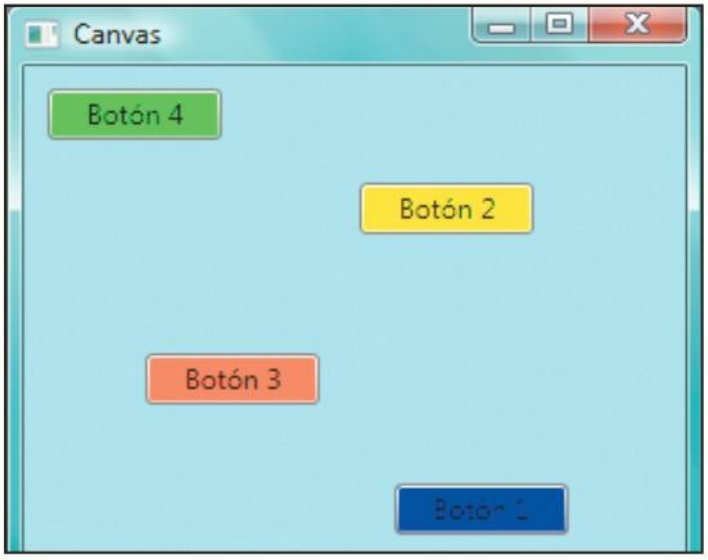


FIGURA 022 | La imagen puede verse; sin embargo, sólo los bordes se muestran bien, porque la mayoría del cuerpo de la imagen tiene aplicada transparencia.

Tabla 8 | Principales propiedades del control Image

Propiedad	Descripción
Source	Especifica la ruta en donde se encuentra la imagen por desplegar. Incluso, podría utilizarse una dirección de Internet que contenga la URL correspondiente.
Height	Especifica el alto de la imagen.
Width	Asigna el ancho de la imagen.

Objetos contenedores

A continuación, veremos cuándo y cómo utilizar estos objetos en nuestros desarrollos.

Los objetos contenedores son aquellos cuyo objetivo principal es guardar (contener) otros controles. La razón por la cual contamos con este tipo de objetos es definir la composición gráfica que tendrá nuestra ventana o nuestra página. Dentro de Windows Presentation Foundation, existen diferentes alternativas para facilitar el posicionamiento de los controles que estarán dibujados en una interfaz gráfica.

Los objetos contenedores heredan de la clase Panel y cuentan con una colección llamada **Children**, donde podemos acceder a todos los objetos que forman parte o que son contenidos por el control.
Podemos describir mediante XAML la definición y los elementos que están dentro del control, o bien agregarlos dinámicamente mediante código imperativo escrito en C# o Visual Basic, utilizando el método **Add()** de la colección **Children**.
Cada uno de los objetos contenedores organiza y dibuja los controles en la pantalla de una manera diferente y con características exclusivas. Los principales objetos contenedores con los que cuenta Windows Presentation Foundation se describen a continuación:

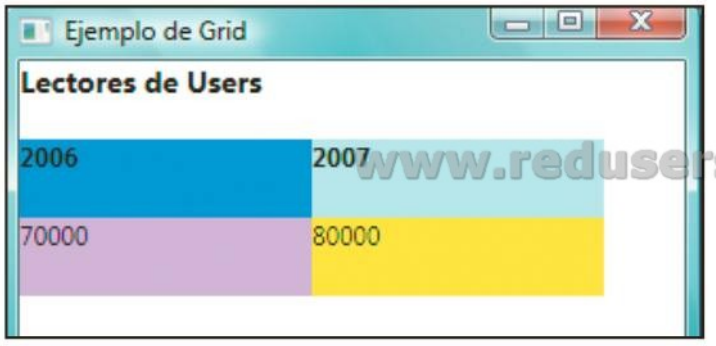


FIGURA 023 | Aplicación de ejemplo utilizando el contenedor Grid.

- Grid
- StackPanel
- DockPanel
- WrapPanel
- Canvas

Grid o grilla

El objetivo principal de **Grid** es contener a los controles hijos mediante el uso de columnas y renglones, tal como sucede en una hoja de cálculo. El objeto Grid de WPF utiliza los elementos **<ColumnDefinition/><RowDefinition/>** para ir definiendo la estructura de las columnas y los renglones que serán utilizados dentro de él, haciendo uso de la **Period Notation**, que se emplea cuando no es posible describir una propiedad mediante un simple valor escalar o no existe un convertidor de tipo predefinido. Luego es posible referirse a la columna o renglón mediante su coordenada, como se puede ver en el siguiente ejemplo:

XAML:

```
<Grid VerticalAlignment="Top" Horizontal
Alignment="Left" Width="276" Height="111">
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>
```

```
    <TextBlock   FontSize="14" FontWeight=
    "Bold" Grid.Row="0">Lectores de Users
    </TextBlock>
    <TextBlock FontSize="12"  Background=
    "DodgerBlue"  FontWeight="Bold" Grid.Row=
    "1" Grid.Column="0">2006</TextBlock>
    <TextBlock FontSize="12" Background=
    "PowderBlue"  FontWeight="Bold" Grid.Row=
    "1" Grid.Column="1">2007</TextBlock>
    <TextBlock Grid.Row="2" Background="Plum"
    Grid.Column="0">70000</TextBlock>
    <TextBlock Grid.Row="2"  Background=
    "Yellow" Grid.Column="1">80000</TextBlock>
</Grid>
```

Además, podemos definir el atributo **ShowGridLines**, que nos permite visualizar las líneas del **Grid**, mientras que los atributos **Width** y **Height** sirven para saber el alto y el ancho (también es posible utilizar el valor * para determinar que una celda o columna ocupe el espacio completo dentro del objeto que lo contiene). A continuación, se define la estructura de la tabla y, luego, insertamos objetos del tipo **Textblock** y ubicamos su posición con los atributos **Grid.Row** y **Grid.Column**. El segmento de código anterior produce el resultado que se muestra en la Figura 23.

StackPanel

Como su nombre lo indica, **StackPanel** (la traducción de stack es pila) permite apilar objetos dentro de él; es decir que cuando agregamos algún control dentro de éste (ya sea mediante XAML o mediante código .NET), StackPanel se encargará de apilar de manera horizontal o vertical el contenido que le está siendo agregado.
Esta funcionalidad nos sirve, principalmente, cuando necesitamos una manera fácil y rápida

> Los objetos contenedores son aquellos cuyo objetivo principal es guardar (contener) dentro de ellos mismos a otros controles.

de acomodar automáticamente los controles, en especial cuando estamos generando objetos en tiempo de ejecución.
La orientación del StackPanel depende del atributo que le sea especificado (horizontal o vertical).

XAML:

```
<StackPanel Orientation="Vertical">
        <TextBlock   FontSize="14"
        FontWeight="Bold">Lectores de
        Users</TextBlock>
        <TextBlock FontSize="12"
        Background="DodgerBlue"  FontWeight=
```

FIGURA 024 | Aplicación de ejemplo utilizando el contenedor StackPanel.

En el caso de WPF, tenemos la posibilidad de utilizar un objeto contenedor como DockPanel, cuyos elementos hijos adquirirán el tamaño que tenga el mismo DockPanel.

```
            "Bold">2006</TextBlock>
            <TextBlock Grid.Row="2" Background=
            "Plum">70000</TextBlock>
            <TextBlock FontSize="12" Background=
            "PowderBlue"  FontWeight="Bold"
            >2007</TextBlock>
            <TextBlock  Background="Yellow"
            >80000</TextBlock>
</StackPanel>
```

FIGURA 025 | Aplicación de ejemplo utilizando el contenedor DockPanel.

DockPanel

Una de las características clave de WPF es la independencia de la resolución y el ajuste automático de las propiedades cuando un usuario decide efectuar un reajuste del tamaño de una ventana o de un objeto. En tecnologías previas, era necesario realizar los cálculos correspondientes para ajustar los controles al tamaño definido por el usuario. En el caso de WPF, tenemos la posibilidad de utilizar un objeto contenedor como DockPanel, cuyos elementos hijos adquirirán automáticamente el tamaño que tenga el mismo DockPanel; es decir, su objeto padre. Esto se logra a través de unas propiedades especiales llamadas **Attached Properties**, que propagan los valores del objeto padre al hijo y se cambien en forma dinámica.

Todos los controles que se le agregan a DockPanel se acoplan a sus bordes. Para lograrlo, es necesario definir en los objetos hijos la localización a la cual se acoplarán. Los valores que puede obtener la propiedad **DockPanel.Dock** de los objetos contenidos dentro de un **DockPanel** son **Top**, **Left**, **Right** y **Bottom**.

Cabe señalar, además, que es posible agrupar muchos objetos con la misma definición de la propiedad, con lo cual se seguirán agregando en el mismo orden en que hayan sido declarados o agregados.

El siguiente ejemplo muestra objetos de diferentes tipos anclados a las distintas posiciones de acoplamiento definidas por DockPanel:

XAML:


```
<DockPanel>
        <TextBox DockPanel.Dock="Top"
        Background="LightSeaGreen">
        Acoplamiento Top</TextBox>
        <Button DockPanel.Dock="Left"
        Background="DodgerBlue">Acoplamiento
```

```
        Left</Button>
        <Label DockPanel.Dock="Right"
        Background="Yellow">Acoplamiento
        Right</Label>
        <Button Background="OrangeRed"
        DockPanel.Dock="Bottom">Acoplamiento
        Bottom</Button>
        <Image Source="C:\Revista USERS\
        ObjetosContenedores\Gatito.bmp">
        </Image>
</DockPanel>
```

Al momento de ejecutar este código, podemos reajustar el tamaño de la ventana, y nos daremos cuenta de que la imagen que contiene al gato automáticamente cambia de tamaño, así como los demás controles que están anclados al DockPanel.

WrapPanel

Es muy similar al control StackPanel, pero añade la funcionalidad de que, si al momento de hacer un cambio de tamaño del objeto que contiene al WrapPanel se da el caso de que ya no existe el espacio suficiente para representar en una sola fila o columna a los objetos contenidos por él, el objeto agregará nuevas filas o columnas para hacer que todos los objetos sean visibles para el usuario al mismo tiempo, en la medida de lo posible y, de esta manera, aumentar la legibilidad de la ventana o página. Este tipo de funcionamiento resulta particularmente útil cuando estamos definiendo controles y agregándolos de manera dinámica.

XAML:

```
<WrapPanel Orientation="Horizontal">
        <TextBlock   FontSize="20"
        FontWeight="Bold">Lectores de
        Users</TextBlock>
        <TextBlock FontSize="18"
        Background="DodgerBlue"  FontWeight=
        "Bold">2006</TextBlock>
        <TextBlock FontSize="18"
        Background="Plum">70000</TextBlock>
        <TextBlock FontSize="18"
        Background="PowderBlue"  FontWeight=
        "Bold">2007</TextBlock>
        <TextBlock FontSize="18"
        Background="Yellow">80000</TextBlock>
</WrapPanel>
```

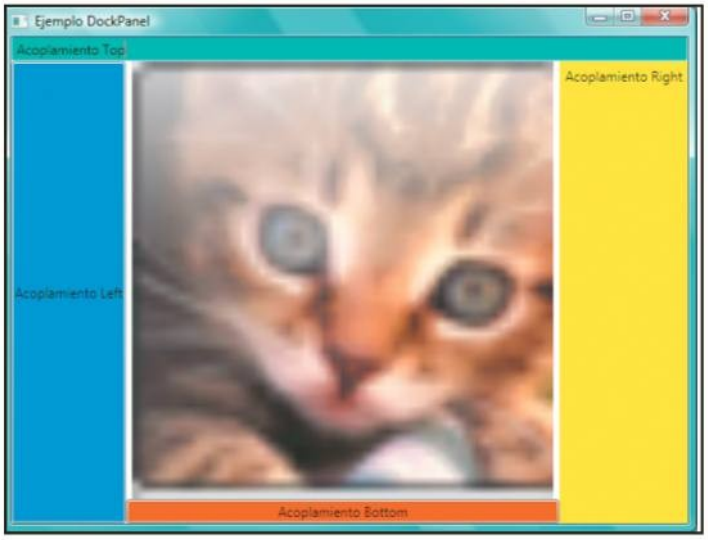


FIGURA 026 | Aplicación de ejemplo utilizando el contenedor DockPanel.

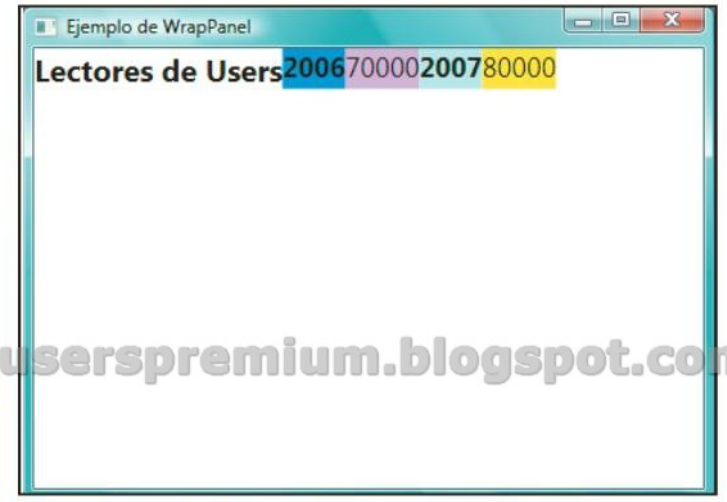


FIGURA 027 | Aplicación de ejemplo utilizando el contenedor WrapPanel.

Si decidimos reajustar el tamaño de la ventana, podemos notar cómo, automáticamente, el objeto WrapPanel se encarga de reacomodar los elementos definidos.

FIGURA 028 | Aplicación de ejemplo utilizando el contenedor WrapPanel, después de cambiar el tamaño de la ventana.

FIGURA 029 | Aplicación de ejemplo utilizando el contenedor Canvas.

El canvas

Puede suceder que nos encontremos con un escenario en el cual no necesitemos que los controles se ajusten automáticamente conforme se van añadiendo, sino que queramos especificar, de manera explícita, la posición relativa que tomará cada uno de ellos, para que queden en el mismo lugar en el que fueron diseñados al momento de definirlos con XAML.
Cuando tenemos este escenario, lo más conveniente es utilizar un objeto del tipo Canvas. El uso de este control es adecuado si deseamos que, a pesar de que el usuario haga modificaciones en el tamaño de la ventana, los controles conserven su posición prefedinida.
Cuando usamos **Canvas**, es posible utilizar las propiedades **Canvas.Left**, **Canvas.Top**, **Canvas.Bottom** y **Canvas.Top** de los controles que contiene.

XAML:

```
<Canvas Background="LightBlue">
        <Button Canvas.Left="159"
        Canvas.Top="186" Height="23" Name=
        "button1" Width="75" Background=
        "Blue">Botón 1</Button>
        <Button Background="Yellow"
        Canvas.Left="144" Canvas.Top="52"
        Height="23" Name="button2" Width=
        "75">Botón 2</Button>
        <Button Background="Tomato"
        Canvas.Left="52" Canvas.Top="128"
        Height="23" Name="button3" Width=
        "75">Botón 3</Button>
        <Button Background="Lime"
        Canvas.Left="10" Canvas.Top="10"
        Height="23" Name="button4" Width=
        "75">Botón 4</Button>
</Canvas>
```

Argentina $8,90 (recargo al interior $0,20) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

20

XAML

Características generales de XAML
Ejemplos / Controles simples

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Windows Presentation Foundation

Principales Namespaces / Herencia de clases / Objetos "freezables"

ISBN 978-987-1347-43-8
00020
9 789871 347438