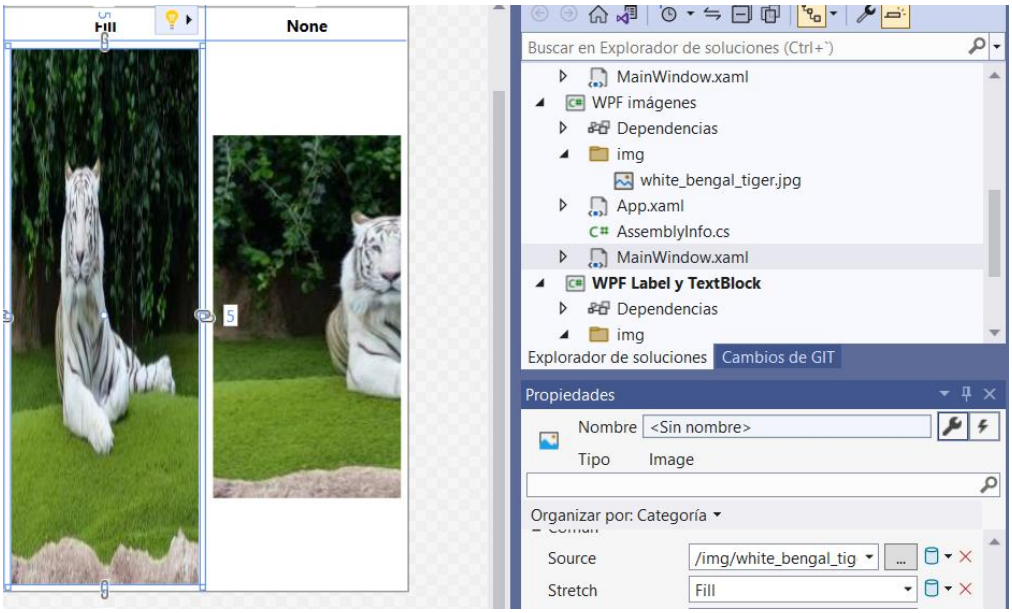
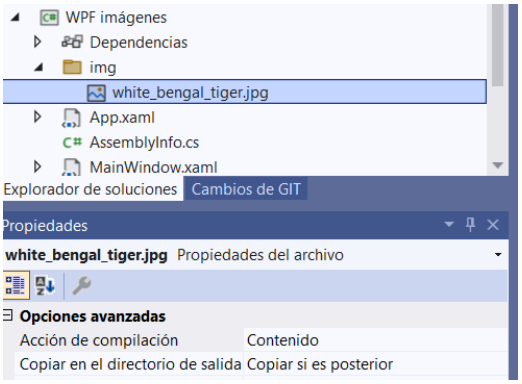
Controles sencillos de WPF

1. Imágenes

El control **Image** como tal es muy sencillo y sirve para mostrar una imagen de una ruta que se especifica en el atributo **Source.**



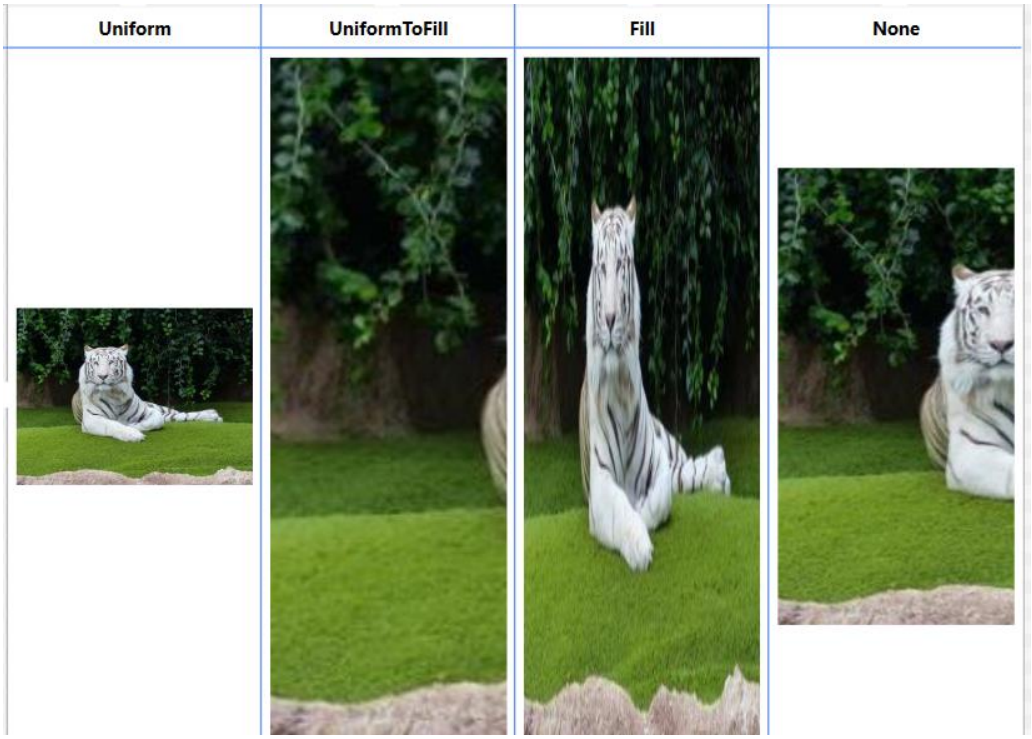
Las rutas de las imágenes tienen un comportamiento un tanto particular. Si seleccionas la imagen de la carpeta **img** desde la barra de herramientas, se copia a la carpeta raíz del proyecto y se crea la ruta /white_bengal_tiger.jpg. Si queremos emplear rutas relativas desde la carpeta img como en la captura anterior, hay que configurar la propia imagen con los siguientes valores en **Acción de compilación** y **Copiar en el directorio de salida**.



Hay otras propiedades respecto a las dimensiones y alineación que pueden resultar de interés.

La propiedad **Stretch** (estiramiento) controla lo que ocurre cuando las dimensiones de la imagen no encajan completamente las dimensiones del control **Image**.

En el siguiente ejemplo tenemos varios contenedores **Image** muy alargados para probar los diferentes valores.



El significado se puede intuir, aunque a continuación se indica con más detalle:

- **Uniform:** Este es el modo por defecto. La imagen va a ser escalada para que quepa en el área del control Image. La relación de aspecto de la imagen se preserva.
- **UniformToFill:** La imagen va a ser escalada para que llene completamente en el área del control Image. La relación de aspecto de la imagen se preserva.
- **Fill:** La imagen va a ser escalada para llenar el área del control Image. La relación de aspecto no se preserva, porque la altura y anchura de la imagen se escalan independientemente.
- **None:** Si la imagen es más pequeña que el control Image, no pasa nada. Si la imagen es más grande que el control Image, esta va a ser cortada para llenar el espacio del control Image, lo que quiere decir que solamente parte de la imagen será visible.

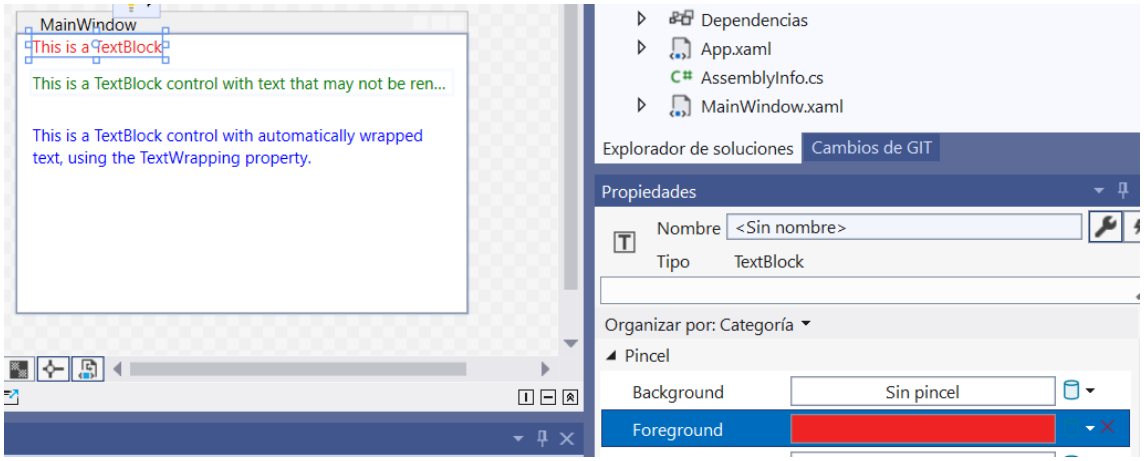
2. TextBlock y Label

Para colocar texto en la pantalla tenemos los controles **TextBlock** y **Label**.

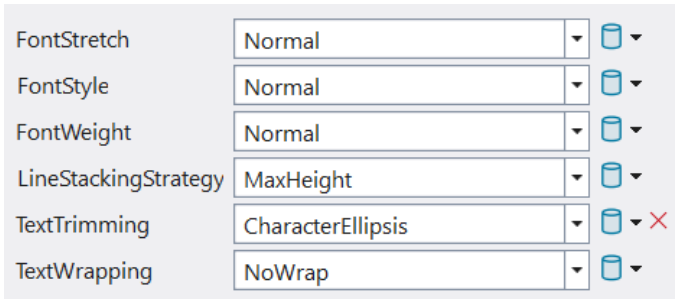
Normalmente se usa **Label** para textos cortos de una línea (que pueden contener, por ejemplo, una imagen).

Por otra parte, **TextBlock** trabaja muy bien con cadenas de texto de varias líneas, pero solo puede contener texto. Veamos un ejemplo y las propiedades necesarias para gestionar cuando un texto tiene varias líneas.

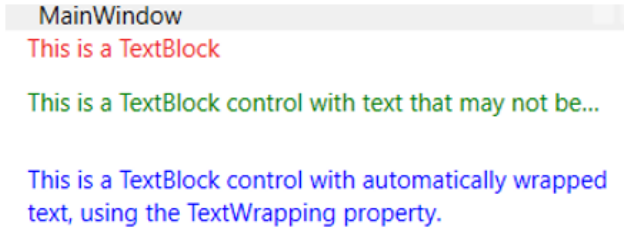
Como se puede observar es posible cambiar el color de texto desde el menú Pincel



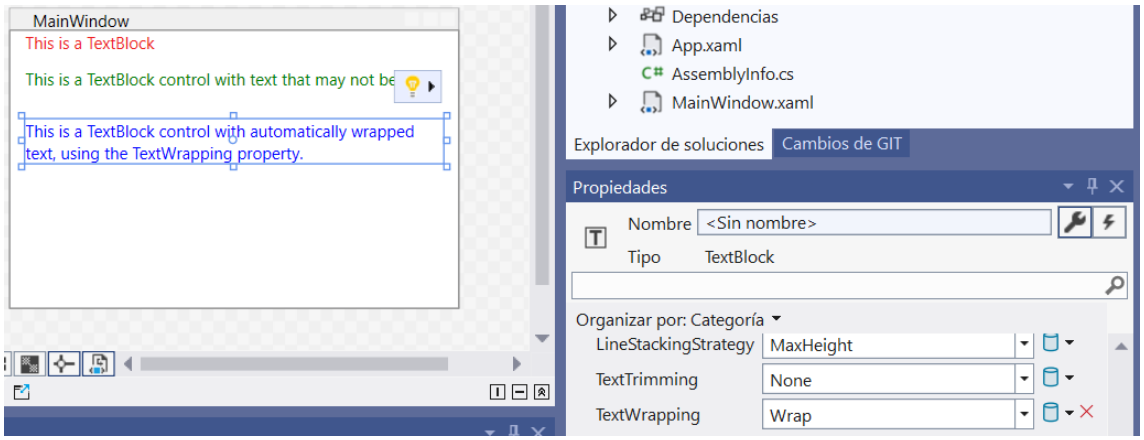
El resto de textos del ejemplo sirven para probar las propiedades **TextTrimming** y **TextWrapping.** El texto verde de la imagen de arriba tiene la propiedad **TextTrimming** con el valor **CharacterEllipsis** que añade puntos suspensivos cuando el texto desborda el espacio disponible.



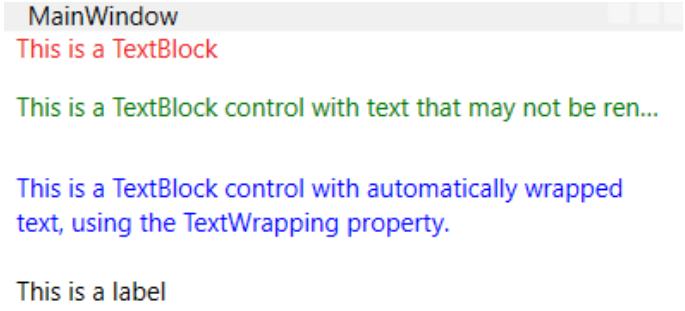
Otra opción de **TextTrimming** sería **WordEllipsis** que añade puntos suspensivos al final de la última palabra que tiene espacio.



Por otro lado, **TextWrap** con el valor **Wrap** añade un salto de línea si el texto no tiene suficiente espacio.



Por último, el control **Label**, en su forma más simple se parece mucho al **TextBlock** que usamos anteriormente.

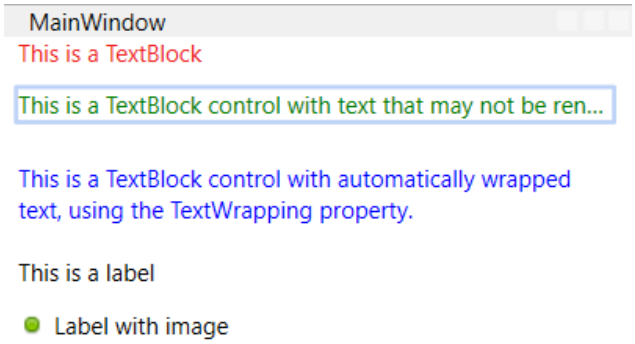


Sin embargo, rápidamente notarás que en lugar de una propiedad de **Text**, tiene una propiedad **Content**. La razón de esto es que **Label** puede alojar cualquier otro tipo de control directamente dentro de él, en lugar de solo texto.

Esto reemplazaría la propiedad **Content** por otro elemento o contenedor. A tener en cuenta que solo puede haber un ítem, por lo que si queremos más de uno tenemos que agruparlo en el mismo panel como en el ejemplo de debajo, que tenemos una imagen y un TextBlock en el mismo StackPane.

```
<Label>
    <StackPanel HorizontalAlignment="Center" Orientation="Horizontal">
        <Image Source="/icon.png" Width="22" Stretch="None"/>
        <TextBlock>Label with image</TextBlock>
    </StackPanel>
</Label>
```

El resultado sería el siguiente

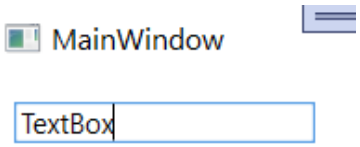


Tanto el **Label** como el **TextBlock** ofrecen cada uno ventajas únicas, por lo que su uso depende mucho de la situación.

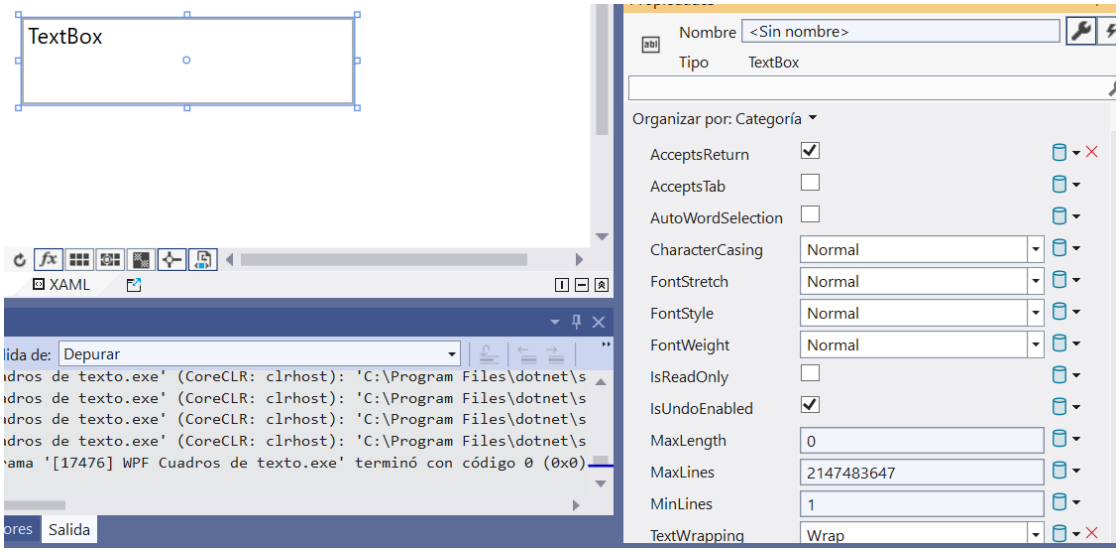
3. TextBox y PasswordBox

El control **TextBox** es el control más básico para introducir texto en WPF, permitiendo al usuario final escribir texto plano en una sola línea para un formulario o como múltiples líneas como en un editor de texto.

En su versión más simple es un texto que tiene un valor de muestra según la propiedad Text.



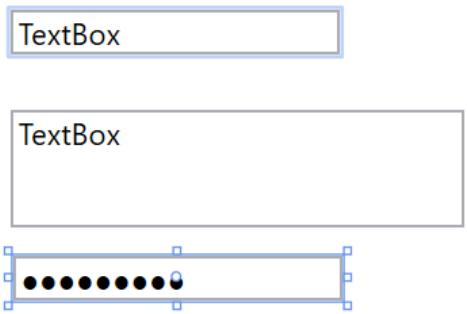
Una de las propiedades más empleadas es **Wrap** que permite que la información introducida ocupe más de una línea, aunque para poder añadir saltos de línea tenemos que activar la propiedad **AcceptsReturn**.



Una desventaja de **TextBox** es que no tiene ninguna propiedad para añadir directamente un texto de muestra para dar una pista al usuario sobre el campo del formulario.

La forma de conseguirlo sería con estilos que estudiaremos más adelante o modificando la opacidad en función de si el usuario tiene el foco, que ya sería necesario con eventos.

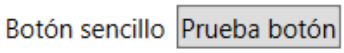
Por otro lado, el control **PasswordBox** funciona de manera similar, aunque ocultando el texto como se puede observar en el ítem de debajo del todo.



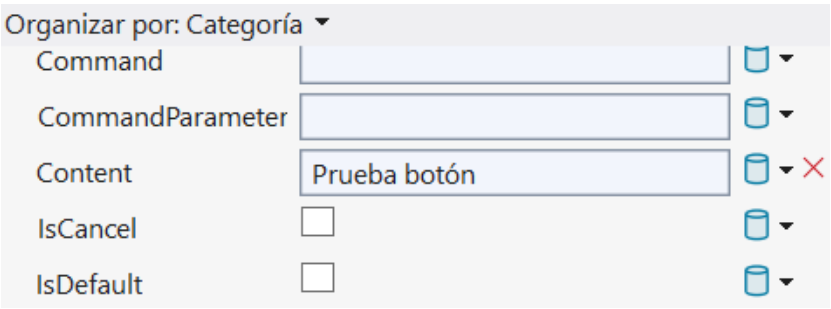
Entre las propiedades más utilizadas, es posible cambiar los caracteres que ocultan el texto y establecer un tamaño máximo de contraseña.

4. Button

Un botón por defecto tiene un funcionamiento bastante sencillo para enviar datos o llevar a cabo alguna acción.

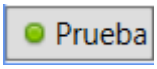


Algunas propiedades empleadas aparte de las básicas de alineación y fuente son **IsCancel** e **IsDefault**.



IsCancel permite que un botón se active al presionar ESC, mientras que **IsDefault** permite que se active al presionar ENTER.

Un botón no solo puede contener texto. Es posible añadir imágenes o contenido más avanzado. Por ejemplo, vamos a probar con una imagen.



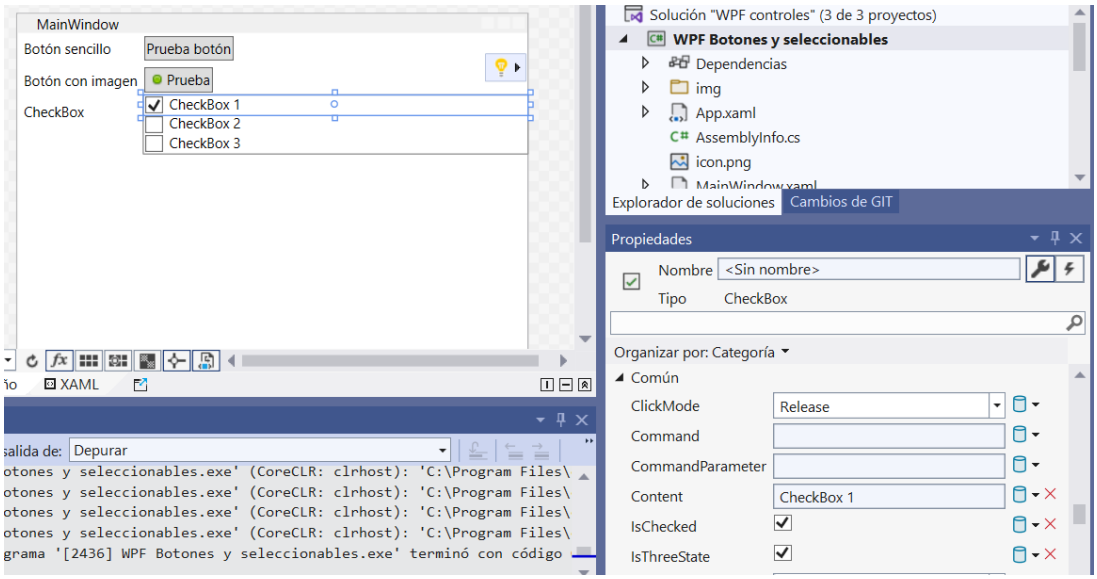
Para ello, es tan sencillo como modificar el contenido textual de un botón por un contenedor (StackPanel en este caso) que contiene la imagen y un TextBlock con el resto.

```
<Button Grid.Column="1" HorizontalAlignment="Left" Grid.Row="1" VerticalAlignment="Center">
    <StackPanel Orientation="Horizontal">
        <Image Source="/icon.png" Stretch="None"/>
        <TextBlock Text="Prueba"></TextBlock>
    </StackPanel>
</Button>
```

5. CheckBox

El control **CheckBox** permite al usuario final alternar una opción entre activo o inactivo.

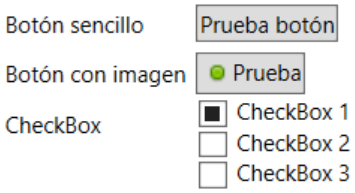
La propiedad más empleada es **IsChecked** que nos permite indicar si la casilla ha sido seleccionada o no.



Un **CheckBox** puede tener un contenido diferente a texto y se añade de manera similar a lo que hemos visto anteriormente con los botones o Label.

Los CheckBox internamente pueden tener dos estados: true y false. No obstante, un tercer valor intrínseco es **null.** Podemos hacer que el valor null también sea representable mediante la propiedad **IsThreeState** que podemos ver también arriba.

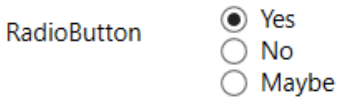
Este sería el resultado si se permite alternar con este estado.



6. RadioButton

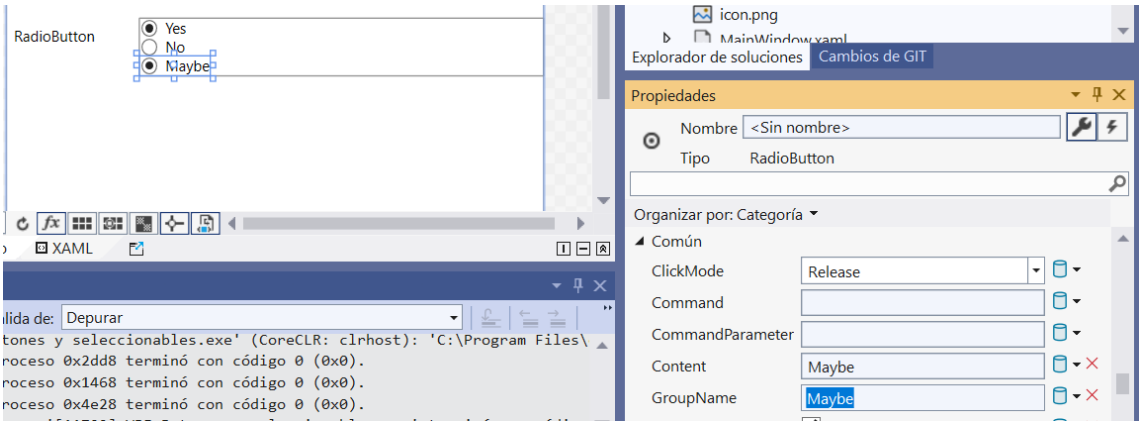
Un **RadioButton** en WPF está más orientado a que el usuario solo pueda seleccionar un ítem al mismo tiempo.

Supongamos el siguiente diseño:



Por defecto, solo se puede seleccionar un **RadioButton** al mismo tiempo. Para agrupar los diferentes **RadioButton** tenemos la propiedad GroupName.

Por ejemplo, el **RadioButton** que se muestra a continuación tiene un valor de **GroupName** diferente al resto y lo seleccionamos por defecto.



Entonces, “Yes” y “No” son excluyentes, mientras que el campo con el texto “Maybe” se puede seleccionar de manera independiente.

Al igual que en otros controles, podríamos modificar el contenido de un **RadioButton** y añadir algo diferente a texto. El método sería similar, es decir, arrastrando un panel sobre el **RadioButton** y añadiendo las imágenes y texto necesarias.