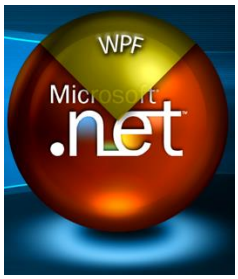


Windows **P**resentation **F**oundation **(WPF)** es una tecnología de Microsoft que potencia las capacidades de desarrollo de interfaces de interacción integrando y ampliando características de las aplicaciones windows y de las aplicaciones web. WPF ofrece una amplia infraestructura y potencialidad gráfica con la que se pueden desarrollar aplicaciones de excelente y atractiva apariencia, con mayores y más funcionales facilidades de interacción que incluyen animación, video, audio, documentos, navegación, gráfica 3D.

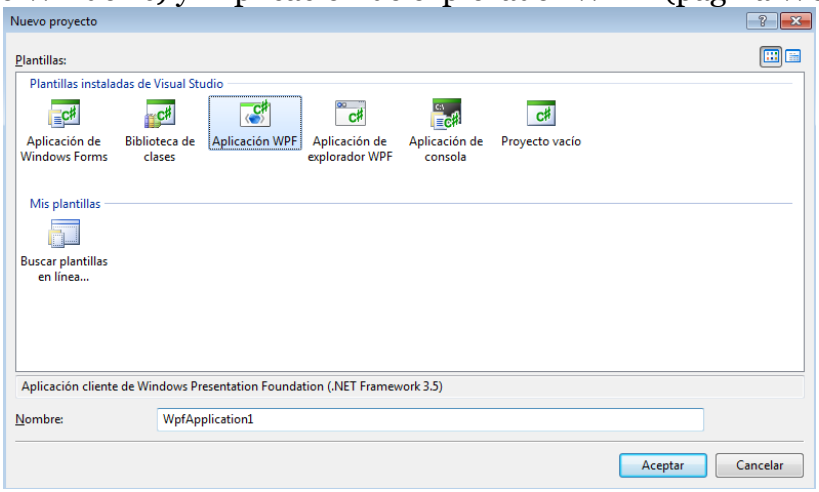
WPF separa, con el *lenguaje declarativo XAML* y los *lenguajes de programación de .NET*, la interfaz de interacción de la lógica del negocio, propiciando una arquitectura *Modelo Vista Controlador* para el desarrollo de las aplicaciones.

XAML

XAML (siglas de e**X**tensible **A**pplication **M**arkup **L**anguage) es el lenguaje declarativo propuesto por Microsoft para definir las interfaces de usuario de las aplicaciones. XAML se basa en una sintaxis XML. XAML propicia separar la definición de las interfaces de usuario de la lógica propia de la aplicación. En este sentido ofrece soporte para expresar el desarrollo de aplicaciones sobre la arquitectura conocida como **MVC** (Model View Controler) Modelo Vista Controlador. La intención con XAML es que en un lenguaje declarativo se puedan definir los elementos que compondrán una interfaz de usuario para que estos puedan ser desplegados y conectados con la lógica de la aplicación mediante un motor de presentación que ejecuta sobre .NET Framework conocido como WPF. XAML sigue las reglas sintácticas de XML. **Cada elemento XAML tiene por tanto un nombre y puede tener uno o más atributos.** XAML ha sido diseñado para establecer una correspondencia lo más directa posible con el CLR de .NET. Por lo general todo elemento en XAML se corresponde con una clase del CLR de .NET, en particular de WPF, y todo atributo XAML se corresponde con el nombre de una propiedad o de un evento de dicha clase. Todos los nombres de elementos y de atributos XAML son sensibles a mayúscula y minúscula. Los valores de los atributos, con independencia de su tipo deben escribirse entre comillas dobles ("). La mayoría de los elementos XAML tienen la característica de que se despliegan visualmente (***render***), pueden recibir entrada de teclado y *mouse*, disparan eventos y saben visualizar el tamaño y posición de sus elementos contenidos (hijos).

Aplicaciones WPF

Al crear una aplicación WPF, se puede elegir entre "Aplicación WPF" (Formulario Windows) y "Aplicación de explorador WPF" (página Web).

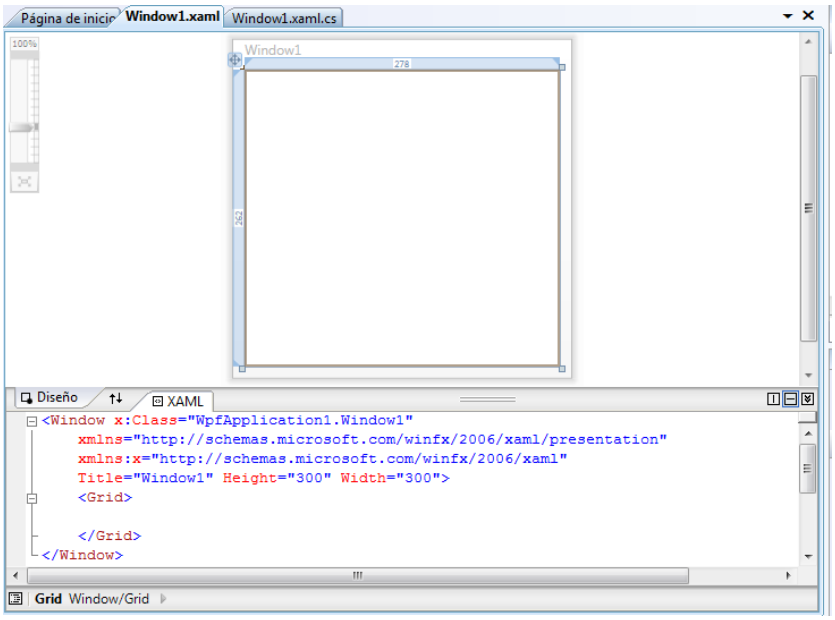


Al crear el proyecto se muestran dos pestañas:

1. **xxx.xaml**

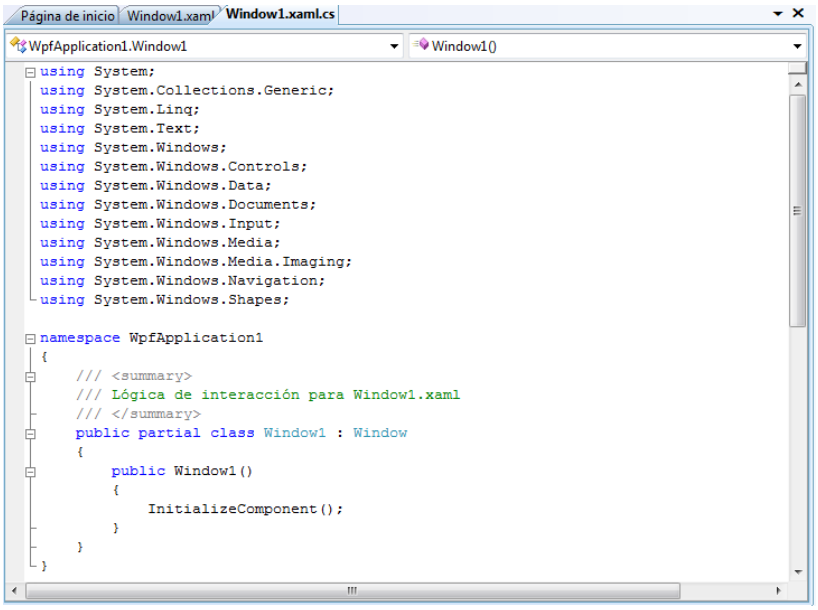
Contiene dos partes:

- El código XAML.
- La vista gráfica o *renderización (*Diseño*)*.



2. **xxx.xaml.cs**

Aquí .NET muestra el código en C# que se necesita para realizar la parte procedimental del proyecto. Este código se llama ***code-behind*** (literalmente "el código que está atrás").



En la pantalla de XAML se crea una instancia de un objeto de la clase *Window1*, que por herencia es también de la clase *Window*. Se definen algunos atributos: el título (*Title*), la altura (*Height*) y el ancho (*Width*). Las unidades en que se indican las medidas son independientes de la pantalla. Una pulgada corresponde siempre a 96 unidades
Dentro de la ventana hay un *Grid*. Es un elemento de *layout* (disposición de distintas áreas dentro de la ventana).

Creación de un saludo con Wpf

Dentro del bloque grid se puede escribir:

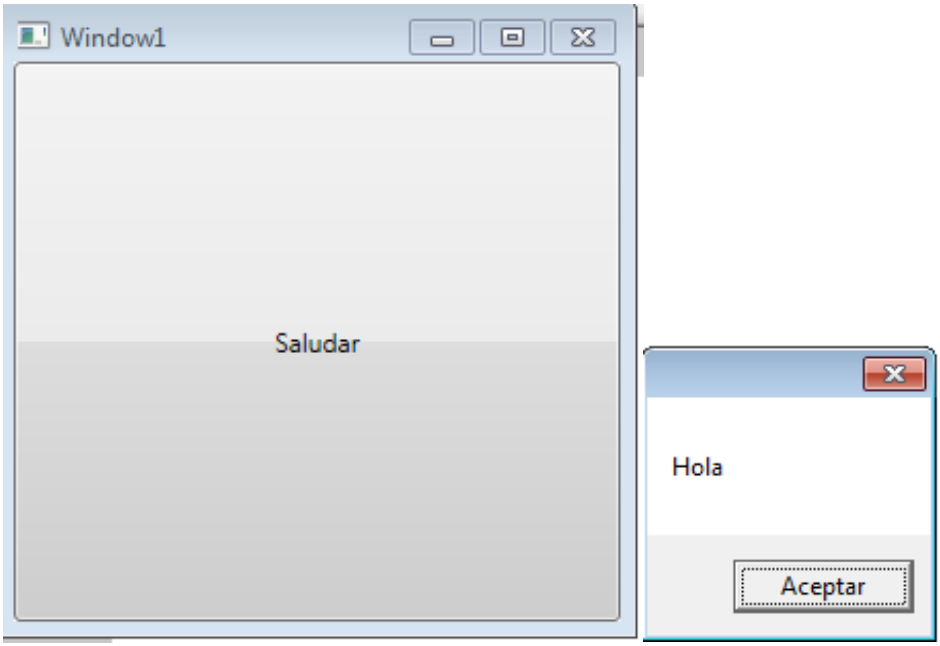
```
<Button Click="Button_Click">
    Saludar
</Button>
```

Si se va a la ventana de Diseño, haciendo doble click sobre "Saludar", se pude ver, el code-behind generado:

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

}
```

Dentro de este evento (click en un botón) se puede escribir un código C#, como por ejemplo:

```
MessageBox.Show("Hola");
```



Se puede generar el mismo proyecto utilizando las herramientas del “Cuadro de herramientas”.

Por ejemplo generar un botón “Saludar” y una caja de texto para ver el saludo:


```
<Window x:Class="WpfApplication1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300">
    <Grid>
```

```
        <Button Height="23" Margin="99,50,104,0" Name="button1"
VerticalAlignment="Top" Click="button1_Click">Saludar</Button>
        <TextBox Height="23" Margin="119,0,39,60" Name="textBox1"
VerticalAlignment="Bottom" TextChanged="textBox1_TextChanged" />
    </Grid>
</Window>
```

```
        private void button1_Click(object sender, RoutedEventArgs e)
        {
            textBox1.Text="Hola";
        }

        private void textBox1_TextChanged(object sender,
TextChangedEventArgs e)
        {

        }
```

Los elementos XAML tienen atributos. Puede ser conveniente tener a la vista el panel de edición de atributos. Con el cursor en el elemento XAML cuyos atributos se desea editar, aparece un panel de propiedades al presionar F4.

PANELES

Son una herramienta principal para el diseño de la interfaz gráfica.
Los paneles principales son:

- **Grid**

Distribución de elementos en una tabla, con la posibilidad de fundir filas y columnas.

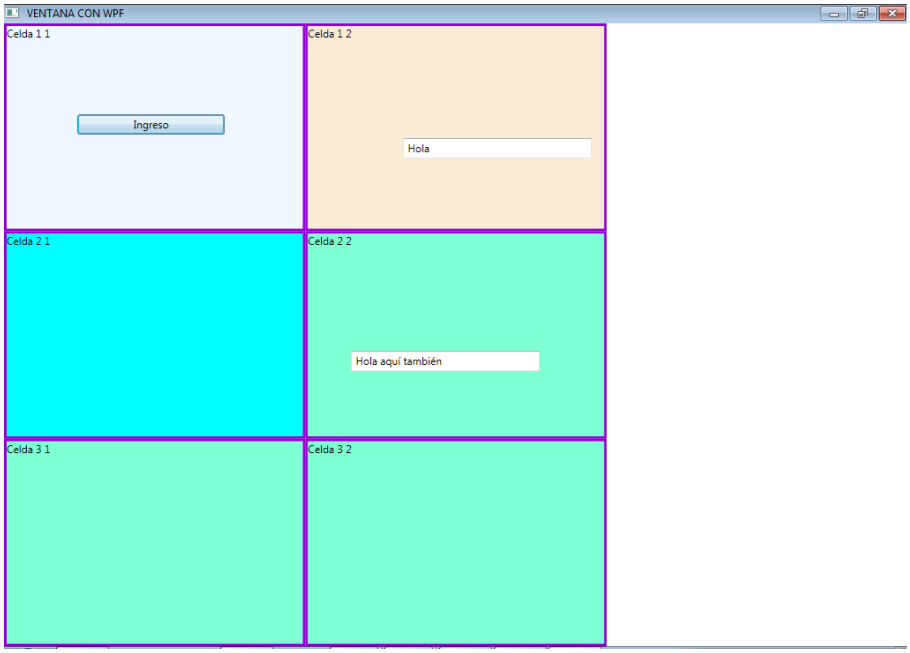
Es el panel más sencillo. Funciona como matriz con sus celdas. Una vez que se definen la cantidad de filas y columnas, se indica el contenido de cada celda y se ensambla así la interfaz de usuario.

Ejemplo:

```
Window x:Class="WpfApplication1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="VENTANA CON WPF" Height="1024" Width="768">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition/>
            <RowDefinition/>
        </Grid.RowDefinitions>

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

        <!--Colocamos un elemento en cada celda-->
        <Border Background="AliceBlue" Grid.Row="0" Grid.Column="0"
                BorderBrush="DarkViolet" BorderThickness="3">
            <TextBlock Text="Celda 1 1"></TextBlock>
        </Border>
        <Border Background="AntiqueWhite" Grid.Row="0" Grid.Column="1"
                BorderBrush="DarkViolet" BorderThickness="3">
            <TextBlock Text="Celda 1 2"></TextBlock>
        </Border>

        <Border Background="Aqua" Grid.Row="1" Grid.Column="0"
                BorderBrush="DarkViolet" BorderThickness="3">
            <TextBlock Text="Celda 2 1"></TextBlock>
        </Border>
        <Border Background="Aquamarine"  Grid.Row="1" Grid.Column="1"
                BorderBrush="DarkViolet" BorderThickness="3">
            <TextBlock Text="Celda 2 2"></TextBlock>
        </Border>
        <Border Background="Aquamarine"  Grid.Row="2" Grid.Column="0"
                BorderBrush="DarkViolet" BorderThickness="3">
            <TextBlock Text="Celda 3 1"></TextBlock>
        </Border>
        <Border Background="Aquamarine"  Grid.Row="2" Grid.Column="1"
```

```
                BorderBrush="DarkViolet" BorderThickness="3">
            <TextBlock Text="Celda 3 2"></TextBlock>
        </Border>
        <Button Height="23" Margin="83,103,91,0" Name="button1"
VerticalAlignment="Top" Click="button1_Click">Ingreso</Button>
        <TextBox Grid.Column="1" Height="23" Margin="111,130,17,0"
Name="textBox1" VerticalAlignment="Top"
TextChanged="textBox1_TextChanged" />
        <TextBox Grid.Row="1" Height="23" Margin="52,136,76,0"
Name="textBox2" VerticalAlignment="Top"
TextChanged="textBox2_TextChanged" Grid.Column="1" />
    </Grid>
</Window>
```

```
        private void button1_Click(object sender, RoutedEventArgs e)
        {
            textBox1.Text = "Hola";
            textBox2.Text = "Hola aquí también";

        }

        private void textBox1_TextChanged(object sender,
TextChangedEventArgs e)
        {

        }

        private void textBox2_TextChanged(object sender,
TextChangedEventArgs e)
        {

        }
```



- **StackPanel**

Distribución de elementos en sucesión vertical u horizontal.

Funciona realmente como un *stack* o pila. Se pueden agregar cualquier cantidad de elementos, cada uno de los cuales se adiciona al stack después del último. Dentro de su interior es posible colocar cualquier elemento válido de WPF. Cuando se usa StackPanel se pueden seleccionar dos orientaciones: Horizontal y Vertical. Si se selecciona Horizontal, los elementos se van agregando de izquierda a derecha, mientras que con el Vertical lo harán de arriba hacia abajo.

Ejemplo:

```
<Window x:Class="WpfApplication1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300">
    <StackPanel Orientation="Horizontal">
        <Border Background="Beige" BorderBrush="White" Width="50">
            <Button Height="29" Name="button1" Width="27"
Click="button1_Click">A</Button>
        </Border>
        <Border Background="Bisque" BorderBrush="BlueViolet"
Width="150">
            <TextBox Height="23" Name="textBox1" Width="100"
TextChanged="textBox1_TextChanged" HorizontalContentAlignment="Center"
/>
        </Border>
        <Border Background="CadetBlue" BorderBrush="Brown" Width="50">
            <Button Height="25" Name="button2" Width="28"
Click="button2_Click">B</Button>
        </Border>
    </StackPanel>
</Window>
```

```
        private void button2_Click(object sender, RoutedEventArgs e)
        {
            textBox1.Text = "-- B --";
        }

        private void textBox1_TextChanged(object sender,
TextChangedEventArgs e)
        {

        }

        private void button1_Click(object sender, RoutedEventArgs e)
        {
            textBox1.Text = "-- A --";
        }
```

- **WrapPanel**

Distribución de elementos en sucesión vertical u horizontal en "líneas" (como el texto, que fluye de una línea a la siguiente).

Es similar al StackPanel. Los elementos se van adicionando en forma indefinida, pero en el WrapPanel, lo hacen cuando y no hay espacio en la ventana; de esta forma el elemento continúa en el siguiente renglón o columna.

- **DockPanel**

Distribución de elementos con anclaje a punto cardinal y posible expansión del último al área sobrante.

Es algo más complicado que los anteriores. Cuando se utiliza es necesario planificar correctamente la interfaz,
Al colocar un elemento en el DockPanel, se debe considerar el espacio que está libre. En él es posible colocar el elemento en la parte superior, inferior, derecha o izquierda. Una vez hecho esto, quedará un nuevo espacio libre, en el que se podrá ubicar el siguiente elemento.

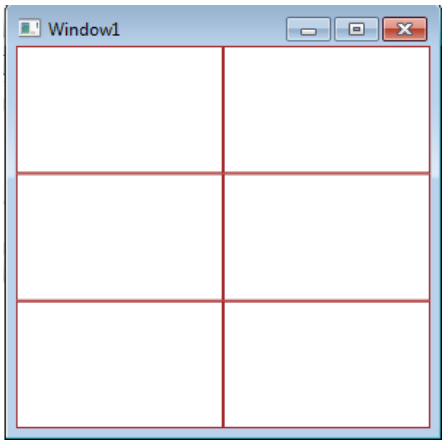
- **UniformGrid**

Distribución de elementos en una matriz cuadrada.

Es similar a Grid, ya que contiene una serie de celdas. La ventana se dividirá de modo uniforme para contener esas celdas.
Hay que establecer de antemano la cantidad de columnas y filas de la matriz.

Ejemplo:

```
<Window x:Class="WpfApplication2.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300">
    <UniformGrid Columns="2" Rows="3">
        <Border BorderBrush="Brown"  BorderThickness="1"></Border>
        <Border BorderBrush="Brown"  BorderThickness="1"></Border>
        <Border BorderBrush="Brown"  BorderThickness="1"></Border>
        <Border BorderBrush="Brown"  BorderThickness="1"></Border>
        <Border BorderBrush="Brown"  BorderThickness="1"></Border>
        <Border BorderBrush="Brown"  BorderThickness="1"></Border>

    </UniformGrid>
</Window>
```



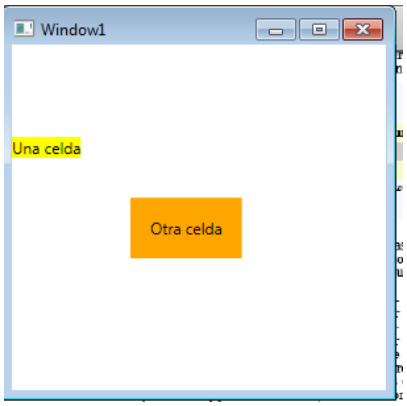
- **Canvas**

Ubicación precisa de elementos.

Es muy sencillo de manejar. Se puede imaginar un lienzo sobre el cual se van a dibujar los diferentes controles o elementos de interfaz. Es el panel más parecido a la forma tradicional de desarrollar las interfaces. Se colocan los elementos dando sus coordenadas dentro de Canvas.

Ejemplo:

```
Window x:Class="WpfApplication1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300">
    <Canvas>
        <Border Background="Yellow" Canvas.Bottom="50"
Canvas.Top="70">
            <TextBlock Text="Una celda"/>
        </Border>
        <Border Background="Orange"  Padding="15" Canvas.Left="90"
Canvas.Bottom="100">
                <TextBlock Text="Otra celda"/>
            </Border>
    </Canvas>
</Window>
```



ACCESO A BASES DE DATOS

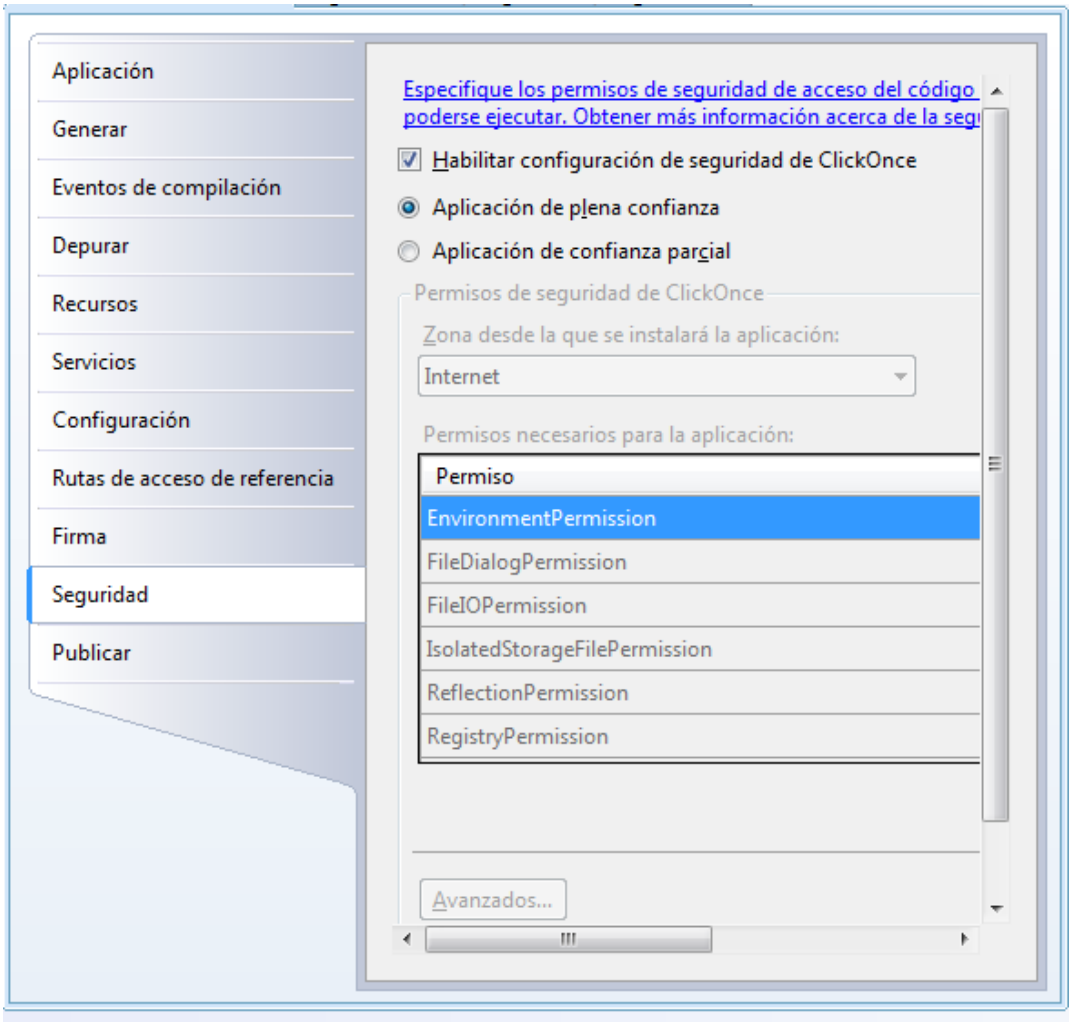
Para acceder a BD con una aplicación WPF de formulario, se tienen que tener en cuenta las mismas consideraciones que para los proyectos de formularios o de consola (Referencia al DBMS y espacio de direcciones para las clases que usan el sistema de BD elegido).

Sin embargo para acceder desde proyectos Browser WPF se deben considerar temas de ***seguridad***.

- **En una aplicación:**

Una manera de poder acceder a una base de datos con suficientes permisos es yendo a:
Proyecto ➔ Propiedades de (aplicación) ➔ Seguridad ➔ Aplicación de plena confianza

Si no se dan los permisos la aplicación no podrá "ver" el servidor al que queremos acceder (por ej. localhost).



- **En el sistema operativo (*como default para todas la aplicaciones*):**

Panel de control ➔ Opciones de Internet ➔ Seguridad ➔ Nivel personalizado ➔ Configuración de seguridad