



**Centro Integrado de Formación Profesional**  
**AVILÉS**  
Principado de Asturias

**UNIDAD 5:**  
**GENERACIÓN DE INTERFACES NATURALES**  
**DE USUARIO. INTERFACES A PARTIR DE**  
**DOCUMENTOS XML**

**DESARROLLO DE INTERFACES**

**2º CURSO**

**C.F.G.S. DESARROLLO DE APLICACIONES MULTIPLATAFORMA**

## REGISTRO DE CAMBIOS

Versión	Fecha	Estado	Resumen de cambios
1.0	19/12/2023	Aprobado	Primera versión
1.1	14/02/2024	Aprobado	Valentine's edition

## ÍNDICE

ÍNDICE .....	2
UNIDAD 5: GENERACIÓN DE INTERFACES NATURALES DE USUARIO. INTERFACES A PARTIR DE DOCUMENTOS XML .....	3
5.1. Interfaces naturales .....	3
5.1.1. Tipos .....	3
5.1.2. Voz y habla. Reconocimiento.....	4
5.1.3. Partes y movimientos del cuerpo. Detección .....	5
5.1.4. Realidad aumentada .....	7
5.1.5. Herramientas para el aprendizaje automático. Entrenamiento.....	7
5.2. Interfaces a partir de documentos XML.....	9
5.2.1. Lenguajes de descripción de interfaces basados en XML. Ámbito de aplicación.....	9
5.2.2. Herramientas libres y propietarias .....	10
5.2.3. El documento XML. Análisis y edición .....	11
5.2.4. Paletas y vistas .....	13
5.2.5. Controles y propiedades.....	15
5.2.6. Ubicación y alineamiento. Contenedores.....	26
5.2.7. Eventos y controladores .....	30
5.2.8. Depuración .....	59
5.2.9. Generación de código para distintas plataformas .....	66
ÍNDICE DE FIGURAS.....	67
BIBLIOGRAFÍA – WEBGRAFÍA .....	68

## UNIDAD 5: GENERACIÓN DE INTERFACES NATURALES DE USUARIO. INTERFACES A PARTIR DE DOCUMENTOS XML

### 5.1. INTERFACES NATURALES

En los últimos años, el mundo de la informática y las telecomunicaciones ha experimentado un aumento en presencia y popularidad de las llamadas Interfaces Naturales de Usuario (**NUI**, **Natural User Interface**). Su misión es sustituir las interfaces habituales de un sistema informático, como ratones, teclados o monitores, por otras más adaptadas al cuerpo humano: comandos por voz, detección de movimientos, reconocimiento facial, etc.

Esta tecnología es cada vez más apreciada, ya que ofrece un manejo más intuitivo de las herramientas tecnológicas, resultando sobre todo de gran ayuda para personas con diferentes limitaciones.

#### 5.1.1. TIPOS

Aunque posteriormente se detallarán algunos de estos tipos de interfaces, en este apartado se enumerará y describirá someramente la tipología de interfaces naturales de usuario:

- Interfaces de voz y habla. Se utiliza el reconocimiento de voz para interactuar con dispositivos y sistemas. Por ejemplo, se pueden usar comandos de voz que realicen acciones específicas.
- Interfaces gestuales. Se basan en la utilización de gestos y movimientos del cuerpo que son recogidos por sensores de movimiento para detectar acciones como deslizar, tocar, etc.
- Interfaces táctiles. También basadas en partes del cuerpo, en este caso se pueden incluir las pantallas táctiles las cuales responden a la interacción directa del usuario con los elementos de la pantalla.
- Interfaces hápticas. Van un paso más allá de las anteriores ya que ofrecen una realimentación en forma de sensaciones físicas al usuario. Por ejemplo, el uso de vibraciones o respuestas táctiles del dispositivo.
- Interfaces cerebrales (Brain-Computer Interface). Permiten el control de dispositivos a través de señales cerebrales. Son tecnologías que permiten una comunicación directa entre el cerebro y el dispositivo.
- Interfaces de seguimiento de mirada. Permiten el control de dispositivos mediante el movimiento de ojos de forma que se pueda entender la intención del usuario mediante un seguimiento de su mirada.
- Interfaces de reconocimiento facial. Se basan en la identificación y reconocimiento de un sujeto a través de sus rasgos faciales. No solo utilizadas en autenticación, sino también para reconocer sus emociones.

- Interfaces de escritura manual y reconocimiento de escritura. Permiten la entrada de información mediante escritura manual. Internamente usan sistemas que pueden reconocer y convertir la escritura a texto digital.
- Realidad aumentada. Se basa en la interacción con entornos digitales o realidad superpuesta en el mundo real.
- Realidad virtual. Aplicada a experiencias inmersivas en entornos digitales generados por computador.
- Interfaces sensoriales multimodales. Integran múltiples modalidades sensoriales combinando voz, gestos, tacto, etc.

### 5.1.2. VOZ Y HABLA. RECONOCIMIENTO

Como se vio anteriormente, uno de los tipos de interfaz natural tiene como elemento principal de entrada la voz y el habla del usuario buscando su reconocimiento para la interacción con el sistema. Las interacciones de voz permiten a los usuarios interactuar con los dispositivos a través de comandos de voz. Estas interacciones se basan en asistentes virtuales como Amazon Alexa o Google Assistant, que reconocen y procesan los comandos de voz del usuario.

Algunos ejemplos de dispositivos que utilizan interacciones de voz son los altavoces inteligentes, los asistentes de hogar inteligentes y los teléfonos móviles.

Otra de las aplicaciones más interesantes es lo que se conoce como tecnología de voz a texto. Se trata de un tipo de software que convierte palabras habladas en texto escrito. Las primeras experiencias se dieron en los años 50 cuando los Laboratorios Bell desarrollaron el primer sistema para reconocer palabras habladas. Principalmente con el desarrollo del aprendizaje automático y la inteligencia artificial es cuando la tecnología de voz a texto se ha convertido en una herramienta práctica y precisa para transcribir el habla.

Actualmente, esta tecnología tiene una amplia gama de aplicaciones, como transcripción, subtítulo, comandos de voz y accesibilidad para personas con discapacidad auditiva. Además de mejorar la accesibilidad de las personas, la tecnología de voz a texto puede revolucionar la forma de comunicarse y trabajar.

A pesar de las mejoras significativas en precisión y facilidad de uso, la tecnología de voz a texto sigue enfrentándose a varios retos y limitaciones. Entre ellos se encuentran:

- Acentos y dialectos. Pueden darse problemas para reconocer y transcribir acentos y dialectos no estándar o regionales.
- Ruido de fondo. Pueden surgir dificultades para separar el habla del ruido de fondo, sobre todo en entornos ruidosos.
- Ambigüedad. Puede tener dificultades para reconocer palabras o frases con múltiples interpretaciones posibles, lo que puede provocar imprecisiones en la transcripción.
- Limitaciones de vocabulario. Puede tener dificultades para reconocer y transcribir vocabulario especializado, como jerga técnica o terminología específica del sector.

Como no todo van a ser inconvenientes y limitaciones, por supuesto, la transcripción de voz a texto tiene una buena colección de ventajas algunas de las cuales ya se han citado en la descripción.

A continuación, se enumeran los aspectos ventajosos de las tecnologías de voz a texto:

- Mayor eficacia y productividad: Permite transcribir el habla en tiempo real, lo que permite a los usuarios ahorrar tiempo y centrarse en otras tareas.
- Mayor accesibilidad e inclusión. Puede ayudar a las personas con deficiencias auditivas a acceder y comprender contenidos de audio y vídeo.
- Organización y gestión de la información más sencillas. Es capaz de convertir palabras habladas en texto que se puede buscar y editar, lo que facilita la búsqueda y organización de información importante.

Algunas herramientas de reconocimiento de texto son:

- [Azure Speech Services](#). Los Servicios de voz de Microsoft Azure son un conjunto de API basadas en la nube que permiten crear aplicaciones y soluciones habilitadas para voz. Se pueden usar para convertir voz en texto, texto en voz, traducir voz e identificar hablantes
- [Google Cloud Speech-to-Text](#). Convierte voz en texto de forma precisa con una API basada en las mejores tecnologías e investigaciones de IA de Google.

Estas herramientas se integran dentro de aquellas usadas para crear asistentes virtuales. Se citan a continuación algunas de las más importantes:

- [Microsoft Bot Framework](#). Plataforma para construir y conectar bots que pueden interactuar con los usuarios a través de diferentes canales, como Skype, Microsoft Teams, Facebook Messenger, y más. Puede integrarse con Azure Speech Services para agregar funcionalidades de voz a los bots.
- [Dialogflow](#). Alojada en Google Cloud, permite construir interfaces de conversación para aplicaciones y dispositivos. Se pueden crear bots de chat y asistentes virtuales de manera fácil utilizando lenguaje natural. Admite integración con Google Cloud Speech-to-Text
- [Botpress](#): Plataforma de desarrollo de chatbots de código abierto que se pueden utilizar para crear bots con funcionalidades avanzadas. Es fácil de usar y personalizable. Se puede integrar con Google Cloud Speech-to-Text y Azure Speech Services.

### 5.1.3. PARTES Y MOVIMIENTOS DEL CUERPO. DETECCIÓN

Relacionado con el punto anterior y complementando los elementos de interacción por voz, surge un nuevo enfoque llamado Zero UI.

Zero UI está referido a interfaces de usuario donde la interacción se lleva a cabo mediante comandos de voz o gestos en lugar de con dispositivos tradicionales. Su objetivo es hacer que la interacción sea más natural y menos intrusiva, permitiendo que se interactúe con los

dispositivos de forma más fluida y libre. Es el siguiente paso en la evolución de la interacción humano-máquina y el internet de las cosas.

Entre sus ventajas se pueden destacar las siguientes:

- Permite una experiencia de usuario más natural, ya que la interacción se basa en acciones y comandos que ya están presentes en la vida cotidiana, como hablar o mover las manos. Esto puede conseguir que los dispositivos sean más fáciles de usar y más intuitivos.
- Reduce el tiempo que se pasa utilizando los ordenadores o dispositivos móviles, pero consiguiendo los mismos resultados.
- Permite una mayor accesibilidad, ya que las personas con discapacidades o limitaciones físicas pueden interactuar con los dispositivos de manera más fácil y libre.

Las interacciones gestuales permiten a los usuarios interactuar con los dispositivos mediante el uso de gestos. Estas interacciones se basan en sensores de movimiento, cámaras o una combinación de ambos, que detectan y reconocen los gestos del usuario, que tienen asignados una acción específica.

Algunos dispositivos que utilizan interacciones gestuales son los sistemas de juegos de realidad virtual, los televisores inteligentes y los dispositivos de control remoto.

Un ejemplo conocido de un dispositivo con reconocimiento gestual es Kinect para Xbox 360, que permite a los usuarios controlar e interactuar con la consola sin necesidad de utilizar un mando de videojuegos tradicional.

Otra forma de interactuar con dispositivos utilizando partes del cuerpo son las interacciones táctiles, aquellas permiten a los usuarios interactuar con los dispositivos mediante el uso de lo que se conoce como retroalimentación háptica. Esto significa que los dispositivos proporcionan una respuesta táctil al usuario a través de vibraciones, pulsaciones o texturas.

Algunos ejemplos de dispositivos que utilizan interacciones táctiles son los teléfonos móviles, los controladores de juegos y los dispositivos de realidad virtual. Otro ejemplo muy claro son los relojes inteligentes, que vibran, por ejemplo, cuando se configuran para que empiecen a contar el tiempo que vamos a hacer deporte.

En conclusión, el Zero UI es una nueva forma de interactuar con los dispositivos que busca hacer la experiencia de usuario más natural y fluida. A través de las interacciones de voz, gestuales o táctiles, entre otras, los usuarios pueden interactuar con los dispositivos de manera más intuitiva y libre. A medida que la tecnología continúa evolucionando, es probable que se vea un mayor uso del Zero UI en una variedad de dispositivos y aplicaciones, permitiendo una experiencia de usuario más natural y satisfactoria.

#### 5.1.4. REALIDAD AUMENTADA

La realidad aumentada (RA) es el término que se usa para describir al conjunto de tecnologías que permiten que un usuario visualice parte del mundo real a través de un dispositivo tecnológico con información gráfica añadida por este. El dispositivo, o conjunto de dispositivos, añaden información virtual a la información física ya existente, es decir, una parte virtual aparece en la realidad.

En la RA, la información digital, como imágenes, videos o gráficos 3D, se superpone a la vista del mundo real a través de una pantalla, cámara o proyector. Esta tecnología utiliza algoritmos de visión por computadora y técnicas de seguimiento para detectar y rastrear los objetos en el mundo real y superponer los elementos virtuales de manera que parezcan estar presentes en el mundo real.

La RA se utiliza en una amplia variedad de aplicaciones, desde juegos y entretenimiento hasta la industria manufacturera y la educación. En el ámbito empresarial, la RA se utiliza para mejorar la eficiencia y la precisión en tareas de mantenimiento y reparación de maquinarias, diseño de productos y entrenamiento de empleados. En la medicina, la RA se utiliza para guiar a los cirujanos durante procedimientos quirúrgicos y para mejorar la precisión en la identificación de estructuras anatómicas durante exámenes médicos.

La RA se distingue de la realidad virtual (RV) en que la RV crea una experiencia completamente virtual en la que el usuario está inmerso, mientras que la RA combina elementos virtuales y reales para crear una experiencia enriquecida en el mundo real.

#### 5.1.5. HERRAMIENTAS PARA EL APRENDIZAJE AUTOMÁTICO. ENTRENAMIENTO

Los pioneros de la Inteligencia Artificial soñaban con construir complejas máquinas que tuvieran las mismas características que la inteligencia humana allá por la década de los 50. En la actualidad, aunque aún sigue pareciendo lejano el hecho de programar algo tan complejo como la mente humana, se está viviendo un avance tremendo en el uso del Machine Learning. Y, desde hace unos cuantos años, concretamente, del Deep Learning. Ambos englobados en la Inteligencia Artificial, la cual fue ideada para hacer que las máquinas sean más listas, incluso que los humanos.

Los términos “machine learning”, que se traduce como “aprendizaje automático”, y “deep learning” o “aprendizaje profundo”, no son nuevos. Formaban parte de la primera era de la inteligencia artificial, en cuanto a cómo se deseaba que fuese el comportamiento de esos agentes inteligentes que imaginaban los primeros ingenieros y programadores.

Los algoritmos de machine learning están viviendo un renacimiento gracias a esta mayor disponibilidad de datos y cómputo. Estos dos elementos permiten que estos algoritmos aprendan conceptos por sí solos, sin tener que ser programados. Es decir, se trata de ese conjunto de reglas abstractas que, por sí solas, son construidas, lo que ha traído y permitido que se “autoconfiguren”.



El machine learning, o aprendizaje automático, nació como una idea ambiciosa de la IA en la década de los 60. Para ser más exactos, fue una subdisciplina de la IA, producto de las ciencias de la computación y las neurociencias. Lo que esta rama pretendía estudiar era el reconocimiento de patrones (en los procesos de ingeniería, matemáticas, computación, etc.) y el aprendizaje por parte de las computadoras. En los albores de la IA, los investigadores estaban ávidos por encontrar una forma en la cual las computadoras pudieran aprender únicamente basándose en datos. Ahora, el principal objetivo del machine learning es abordar y resolver problemas prácticos en donde se aplique cualquiera de las disciplinas numéricas antes mencionadas.

Quizás el primer gran éxito del aprendizaje automático en el mercado fue Google, que demostró que era posible encontrar información usando un algoritmo informático y este se basa en el aprendizaje automático. Desde entonces, ha habido muchos éxitos comerciales del aprendizaje automático. Compañías como Amazon y Netflix usan el aprendizaje automático para sugerir artículos que pueda interesar comprar, películas que pueda interesar ver.

El aprendizaje automático o “machine learning” es una disciplina científica del ámbito de la Inteligencia Artificial que crea sistemas que aprenden automáticamente. Aprender en este contexto quiere decir identificar patrones complejos en millones de datos. La máquina que realmente aprende es un algoritmo que revisa los datos y es capaz de predecir comportamientos futuros. Automáticamente, también en este contexto, implica que estos sistemas se mejoran de forma autónoma con el tiempo, sin intervención humana.

En el ámbito del diseño de interfaces de usuario, también se pueden aprovechar las técnicas de aprendizaje automático utilizando herramientas específicas diseñadas para tal fin. A continuación, se enumeran algunos ejemplos:

- [Framer](#): Se trata de una herramienta de diseño que incorpora funcionalidades de aprendizaje automático para crear prototipos de interfaces de usuario interactivas. Permite la creación de experiencias de usuario realistas y puede aprender de la interacción del usuario para mejorar los prototipos.
- [Adobe Sensei](#): Plataforma de inteligencia artificial y aprendizaje automático desarrollada por Adobe. Se utiliza en productos como Adobe XD para ofrecer funciones inteligentes entre las cuales están el reconocimiento de voz y la generación automática de estilos de diseño.
- [RunwayML](#): Plataforma que permite a los diseñadores integrar fácilmente modelos de aprendizaje automático en sus proyectos de diseño. Puede ser utilizado para crear interfaces interactivas que respondan a la entrada del usuario de manera inteligente.
- [Clarifai](#): Servicios de reconocimiento visual y análisis de imágenes. Puede ser empleado para desarrollar interfaces que comprendan y respondan a las imágenes cargadas por los usuarios.
- [Teachable Machine](#): Permite entrenar modelos de aprendizaje automático sin necesidad de escribir código. Se puede usar para crear experiencias interactivas basadas en la visión por computadora, como el reconocimiento de gestos o la clasificación de imágenes.

En este grupo se puede incluir la ya citada Dialogflow.



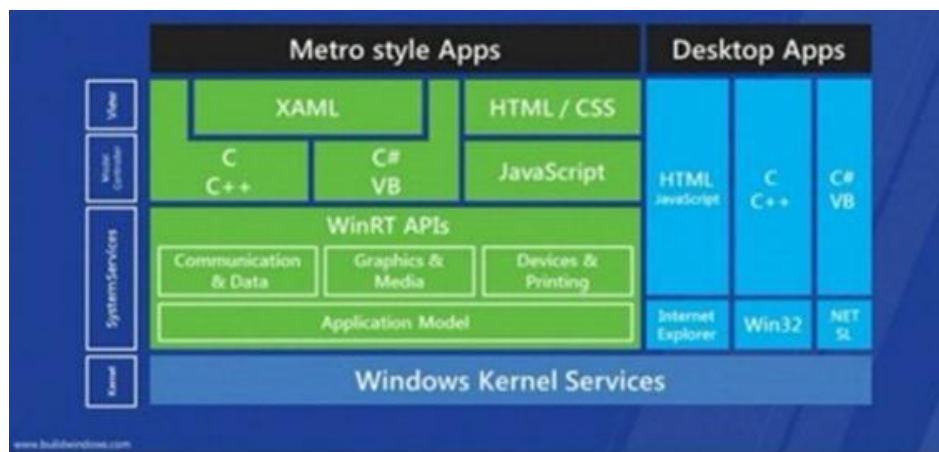
## 5.2. INTERFACES A PARTIR DE DOCUMENTOS XML

### 5.2.1. LENGUAJES DE DESCRIPCIÓN DE INTERFACES BASADOS EN XML. ÁMBITO DE APLICACIÓN

#### 5.2.1.1. XAML

Las aplicaciones Windows son aplicaciones de ventanas que presentan datos de una forma más gráfica y agradable que la ventana de línea de comandos. Para desarrollar este tipo de aplicaciones, .NET proporciona desde sus inicios la tecnología Windows Forms. La versión 3.0 del framework .NET, aparecida en 2006, llegó con una nueva tecnología de presentación: WPF (Windows Presentation Foundation). Aporta algunas mejoras notables tanto a nivel técnico como en lo relativo al desarrollo:

- La definición de la interfaz se realiza mediante el lenguaje de etiquetas XAML (eXtensible Application Markup Language) y no mediante código C#.
- Se desacopla la interfaz gráfica del código de negocio fácilmente gracias a la noción de binding.
- La visualización ya no se basa en el componente de software GDI sino en DirectX, lo que implica que algunos cálculos puede realizarlos la GPU.
- Todos los componentes de WPF utilizan el diseño vectorial.
- En un componente, el aspecto gráfico y el aspecto funcional están muy poco ligados, lo que permite modificar la parte gráfica de los componentes sin afectar a su comportamiento.



1. Arquitectura de XAML en Windows

La tecnología WPF utiliza el lenguaje XAML para la creación de interfaces gráficas. Este lenguaje es un dialecto de XML que permite instanciar objetos .NET de manera declarativa, es decir, mediante el uso de etiquetas, de manera similar a como lo hace HTML. Los distintos objetos están anidados a partir de un elemento raíz, formando un árbol lógico.

La definición de un botón mediante código XAML tiene el siguiente aspecto:

```
<Button Height="30" Width="120">Esto es un botón</Button>
```

La etiqueta <Button> instancia un objeto de tipo Button, mientras que los atributos XML asignan valor a las propiedades Height y Width del botón. Como cualquier elemento XML, una etiqueta XAML puede contener otras etiquetas o un valor textual.

## 5.2.2. HERRAMIENTAS LIBRES Y PROPIETARIAS

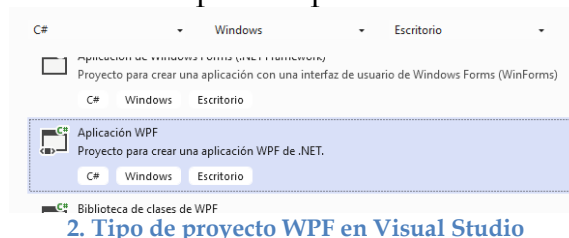
En la primera unidad se citaron algunas herramientas que permiten la edición de archivos XML como Sublime o Visual Studio Code. Además, el resto de IDEs vistos en dicha unidad (NetBeans, Eclipse, etc.) proporcionan también funcionalidades para la edición de documentos XML y sus dialectos. Algunas de las herramientas que permiten la edición y manejo de ficheros XML son:

- [XML Liquid Studio](#). Editor propietario para Windows que permite la edición de texto, la edición gráfica y la edición de tipos WYSIWYG. Tiene una versión Free Community Edition.
- [Oxygen XML Editor](#). Editor para Windows, MAC y Linux, herramienta propietaria que permite edición de texto, edición gráfica y la edición de tipos WYSIWYG.
- [XML Notepad](#). Editor para Windows con licencia pública de Microsoft, que permite la edición de texto, edición gráfica y edición de tipo WYSIWYG.

### 5.2.2.1. Visual Studio 2022

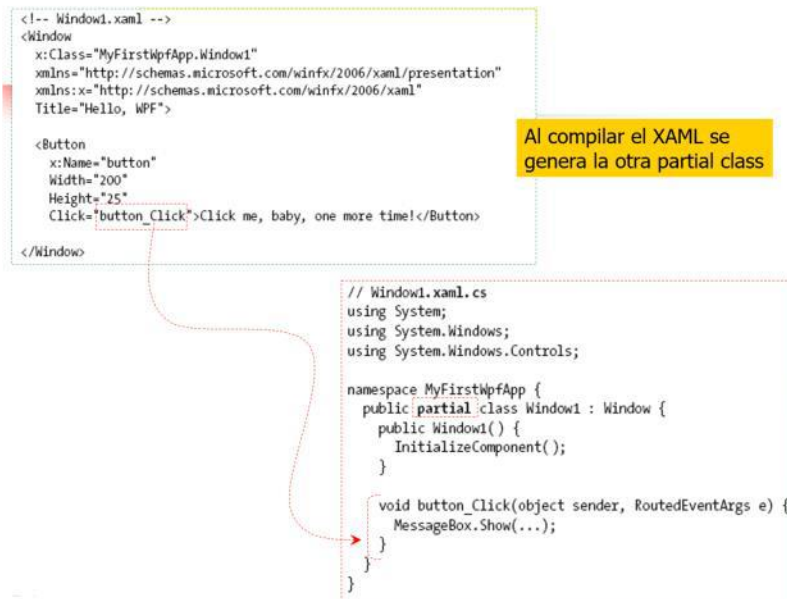
Es la herramienta por excelencia de desarrollo para aplicaciones .NET. Proporciona todo tipo de posibilidades de desarrollo, desde las clásicas aplicaciones de consola hasta desarrollo orientado a la nube pasando por aplicaciones web, de escritorio, base de datos, etc. Es multilenguaje incluyendo de base lenguajes clásicos de Microsoft como Visual Basic o C# complementados por funcionales como F#, puramente compilados como C/C++ y dando soporte a lenguajes como Python.

A continuación, se muestra cómo crear una aplicación WPF en Visual Studio 2022. En primer lugar, se acude a la opción “Crear un proyecto”. En los desplegados se elige el siguiente tipo de solución utilizando lo filtrados de la parte superior:



#### 2. Tipo de proyecto WPF en Visual Studio

En los proyectos WPF, el código asociado a los controles está en XAML y aparece en la parte inferior de la ventana en la que se ve el diseño del interfaz y se modifica para ajustar la interfaz a las necesidades requeridas. Además, también se trabaja con los ficheros que implementan la lógica, denominados **Codebehind**, y en ellos estarán entre otros la implementación de los manejadores de eventos.

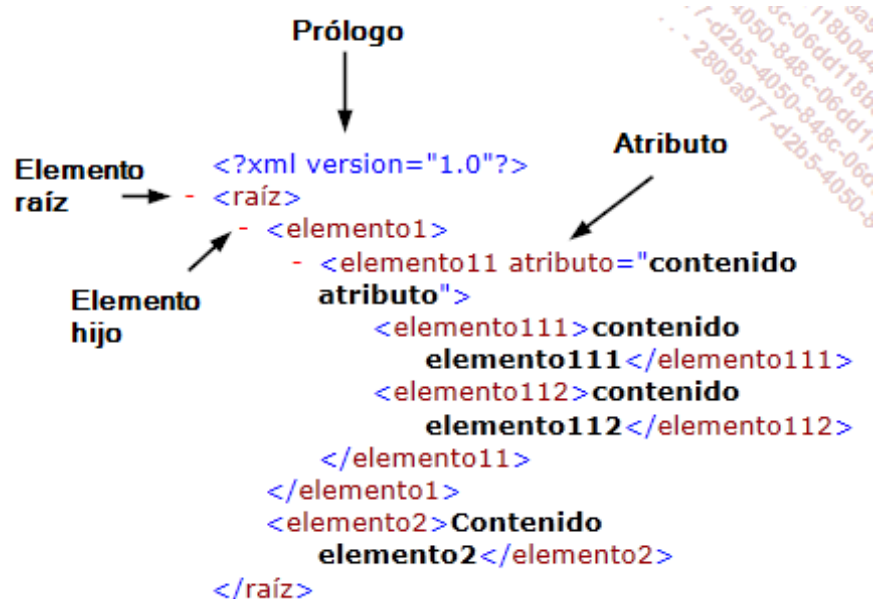


3. Código XAML vs C#

### 5.2.3. EL DOCUMENTO XML. ANÁLISIS Y EDICIÓN

Un documento XML es un archivo de texto especial compuesto por un determinado número de etiquetas en estructura de árbol. Los elementos del árbol están todos vinculados entre ellos a través de las ramas que se crean.

Un documento XML está organizado según varios componentes que pueden representarse en la forma de un árbol. Por ejemplo, el siguiente esquema enumera todos estos componentes. Las siguientes secciones presentan con más detalle las funciones de estos últimos.



4. Esquema básico de un documento XML

La primera línea de un documento XML es el prólogo y contiene, entre otras cosas, la declaración de la versión de XML utilizada para la descripción de la información. Esta declaración tiene el siguiente formato:

```
<?xml version="1.0" ?>
```

Los elementos forman la estructura en árbol de un documento XML. El nombre de los elementos se incluye dentro de las etiquetas, su valor se coloca entre una etiqueta de apertura y una etiqueta de cierre:

```
<NombreDeElemento>Valor del elemento</NombreDeElemento>
```

Para un elemento determinado, los nombres de las etiquetas de apertura y de cierre deben ser idénticos. Se deben respetar las mayúsculas y minúsculas del nombre.

Los atributos son información adicional relacionada con los elementos XML. La escritura que se utiliza para especificar un atributo de un elemento es la siguiente:

```
<elemento atributo="valor del atributo">valor del elemento</elemento>
```

En el siguiente ejemplo, el atributo sexo permite añadir información al elemento persona:

```
<persona sexo="masculino">LAVALLE</persona>
```

Algunos atributos ya están definidos por el lenguaje. Así, el atributo xml:lang permite definir el lenguaje utilizado en un archivo XML. El atributo xml:space permite excluir los espacios múltiples en un nombre.

Como en todos los lenguajes, los desarrolladores de XML pueden añadir comentarios de diseño. Estos comentarios no los procesa el compilador, pero para ello deben incluirse en el documento XML de la siguiente manera:

```
<!-- Escriba aquí su comentario -->
```

### 5.2.3.1. El documento XAML

En un documento XAML de WPF el elemento principal es la etiqueta **<Window>** que hará de elemento raíz, además de la función de contenedor del resto de elementos. Una vez alcanzada la etiqueta de cierre **</Window>** no podrá existir código o etiquetas adicionales. Dentro de la etiqueta Window se definen diferentes espacios de nombres → **xmlns** que serán utilizados posteriormente dentro del documento al representar ciertas etiquetas. Estos espacios de nombres deben incluir al menos las referencias:

- <http://schemas.microsoft.com/winfx/2006/xaml/presentation>: Este espacio de nombre es el encargado de asociar los elementos descritos en el documento con los controles de WPF contenidos en el espacio de nombres System.Windows.Controls del .NET Framework.
- <http://schemas.microsoft.com/winfx/2006/xaml>: Se encarga de definir las etiquetas utilizadas por XAML asociadas al espacio de nombres de .NET Framework System.Windows.Markup.

Los espacios de nombres que aparecerán habitualmente en un documento XAML son:

- **xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation**
- **xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml** Establece un alias de x con el espacio de nombres. Luego se usará x en el código, en lugar de todo el nombre del espacio de nombres.
- **xmlns:d="http://schemas.microsoft.com/expression/blend/2008"** Espacio de nombres destinado para el apoyo al diseñador. El espacio de nombres d: XAML habilita atributos del diseñador en elementos XAML.
- **xmlns:mc=http://schemas.openxmlformats.org/markup-compatibility/2006** Indica y soporta un modo de compatibilidad de marcado para la lectura de XAML.
- **xmlns:local="clr-namespace:WpfApp"** Se indica el nombre de la solución, en este caso WpfApp.
- **mc:Ignorable="d"** Normalmente, el prefijo d: se asocia con el atributo mc: Ignorable. Esta técnica permite a los analizadores XAML en tiempo de ejecución ignorar los atributos de diseño

```
<Window x:Class="CaribeResortPlayaBelice.Window1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:CaribeResortPlayaBelice"
        Title="Servicio de Habitaciones" Width="757" Height="484">

    <Window.Resources>
        <local:Restaurantes x:Key="Restaurantes">
            <local:Restaurante Nombre="Alhambra">
                <local:Restaurante.Platos>
                    <local:Plato Tipo="Entrada" Nombre="Ensalada Tropic" />
                    <local:Plato Tipo="Carne" Nombre="Solomillo en hoja" />
                </local:Restaurante.Platos>
            </local:Restaurante>
            <local:Restaurante Nombre="Capri">
                <local:Restaurante.Platos>
                    <local:Plato Tipo="Entrada" Nombre="Parrilla de ver" />
                    <local:Plato Tipo="Bebida" Nombre="Limoncello" Id="1" />
                </local:Restaurante.Platos>
            </local:Restaurante>
        </local:Restaurantes>
    </Window.Resources>

    <Grid DataContext="{StaticResource Restaurantes}">
        <Grid.ColumnDefinitions>
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" MinHeight="243" />
        </Grid.RowDefinitions>
    </Grid>
```

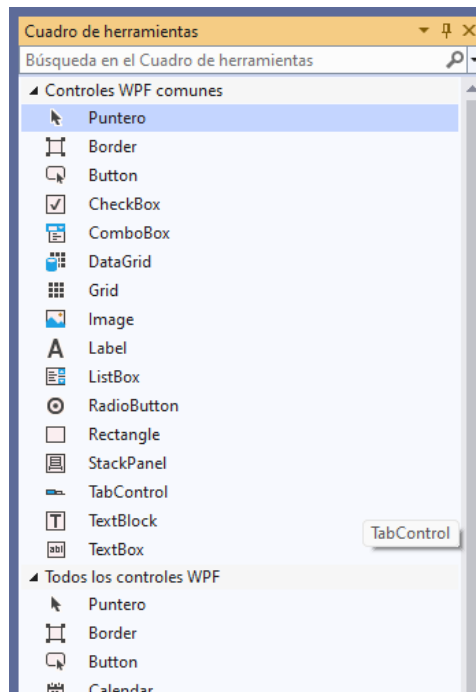
#### 5. Namespaces que pueden aparecer en un fichero XAML

### 5.2.4. PALETAS Y VISTAS

El modo de trabajo con WPF es mixto cuando se trabaja con vistas; es decir, se tiene una visualización dividida en diseño y en código pudiendo trabajar con ambas casi simultáneamente ya que, al posicionarse en una parte del código, el sistema lo hace en la de diseño y viceversa. Además, la vista de edición de XAML también permite Intellisense

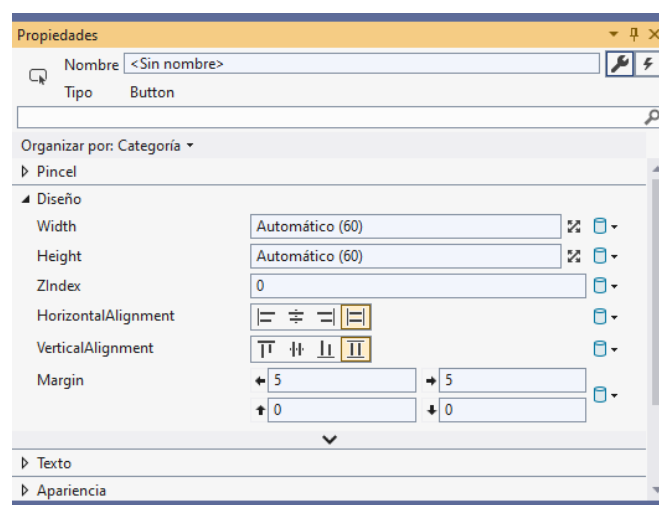
(completado de código) con lo que, aunque a priori parezca farragoso, es muy cómoda la inclusión de código de diseño.

La paleta de componentes viene especificada como Cuadro de Herramientas y puede estar flotante (desplegarse solo cuando se necesita) o fija. Permite la inclusión tanto en la vista de código como en la de diseño, lo cual hace que sea aún más potente y funcional.



6. Paleta de componentes en WPF

Otra ventana prácticamente imprescindible en cualquier IDE es el panel de propiedades. Tal como puede verse en la siguiente imagen, permite búsqueda y distintas formas de organización (por categoría, nombre y origen).



7. Panel de propiedades de un control WPF



### 5.2.5. CONTROLES Y PROPIEDADES

En XAML cada elemento gráfico se define mediante una etiqueta de apertura y otra de cierre, con una serie de atributos que determinan el aspecto y comportamiento de este. Cada una de estas etiquetas se corresponde con una clase .NET, coincidiendo el nombre de la etiqueta coincide con el del de aquella.

Las propiedades de la clase se traducen en atributos de la etiqueta, siendo en ocasiones será preferible hacer uso del anidamiento de otros elementos dentro del elemento a describir, haciendo a veces de contenedor.

Las etiquetas XAML que definen un elemento de la interfaz presentan diferentes atributos, entre los que destacan:

- **Name**, identificador único del elemento. Este identificador es especialmente útil si se desea utilizar o referenciar posteriormente en el código. Se representará como x:Name en caso de definir el espacio de nombre con el identificador x.
- **Key**, permite establecer un identificador para los elementos definido en el diccionario. Se representará como x:Key en caso de definir el espacio de nombres con el identificador x.
- **{Binding}**, elemento definido dentro del valor del atributo. Permite definir el enlace a una fuente de datos, la cual puede ser otro control, un fichero, base de datos, etc.
- **{StaticResource}**, al igual que en el caso anterior se define dentro del valor de un atributo. Permite hacer referencia a un elemento definido en el diccionario de recursos. {StaticResource} define como argumento el identificador del recurso, el cual fue previamente definido con el atributo Key en el diccionario de recursos.
- **{Null}**, valor que permite representar el valor nulo dentro de XAML

La representación de los elementos XAML se puede hacer de dos modos:

- **Basada en atributo:** los atributos presentan una sintaxis atributo = valor, el valor debe ir entre comillas.

```
<Button Width="125" Height="75">
    Aceptar
</Button>
```

- **Basada en Propiedad:** En ocasiones un atributo no será suficiente para poder describir toda la información necesaria, y se necesitará de otras propiedades que deberán ir anidadas dentro de la etiqueta asociada al componente, como se puede observar en este ejemplo con la propiedad Background. Las etiquetas anidadas tienen la sintaxis <elemento.propiedad>, siendo elemento el valor de la etiqueta que hace de contenedor.

```
<Button Width="125" Height="75">
    <Button.Background>
        <ImageBrush Opacity="0.75" />
    </Button.Background>
    Aceptar
</Button>
```



## Contenido de Texto

Para determinar el texto de un componente, existen dos modos:

- Tipo HTML, en el que el texto va entre las etiquetas del componente.
- Atributo Content encadenado al componente, del modo **<componente.Content>**

Si aparece más de una definición para el contenido de texto de un componente, el compilador dará un error indicando que hay más de una definición para la propiedad Content.

Windows proporciona una amplia biblioteca de controles para el desarrollo de interfaces de usuario. Se distinguen entre los controles que tienen una representación visual, de los que se verán los más importantes, y los que funcionan como contenedores vistos en el posteriormente.

En la siguiente URL se puede encontrar una lista ordenada alfabéticamente con todos los controles disponibles:

<https://learn.microsoft.com/es-es/windows/apps/design/controls/>

Por otra parte, en el repositorio GitHub de Microsoft se aloja una aplicación ejemplo de uso de controles XAML:

<https://github.com/microsoft/WinUI-Gallery>

### **5.2.5.1 Enlaces a datos (Data Binding)**

El enlace de datos, en su sentido más puro, es la capacidad de un control de conectarse a una fuente de datos tal que el control (a) muestre ciertos elementos de esa fuente de datos, y (b) se mantenga en sincronizar con la fuente de datos. Una vez realizada la conexión, el tiempo de ejecución maneja todos los trabajos necesarios para que esto suceda. Realmente no importa dónde o cómo estén los datos almacenados: podría ser un sistema de archivos, una colección personalizada de objetos, un objeto de base de datos, etc.

Se va a ver brevemente cómo poder establecer una conexión de enlace de datos usando WPF. La clave aquí es la clase [System.Windows.Data.Binding](#). Este es el mediador encargado de vincular un control con una fuente de datos. Para declarar un enlace, hay que saber tres cosas:

- ¿Qué propiedad del control se quiere vincular?
- ¿A qué origen de datos se quiere enlazar?
- Y, dentro del origen de datos, ¿qué elemento o propiedad específica posee el dato que interesa?

La vinculación es posible con objetos individuales (como enlazar una propiedad de cadena en un objeto a un cuadro de texto), o a colecciones de objetos (como vincular una colección List<> a un cuadro de lista). De cualquier modo, la mecánica sigue siendo la misma:

```
Binding binding = new Binding();  
binding.Source = stringList;  
listBox1.SetBinding(ListBox.ItemsSourceProperty, binding);
```

El fragmento de código anterior crea un objeto de tipo Binding y establece el origen del objeto a una colección List<String>; luego llama a SetBinding en el control (un ListBox), pasando la propiedad exacta del control a vincular con la fuente de datos, y el propio objeto binding.

También se pueden asignar fuentes de datos a un objeto especial llamado DataContext. Cada objeto [FrameworkElement](#), y aquellos que derivan de esa clase, implementan sus propios contextos de datos. Se puede pensar en esto como un área global donde los controles pueden ir para obtener sus datos para vincularse a ellos.

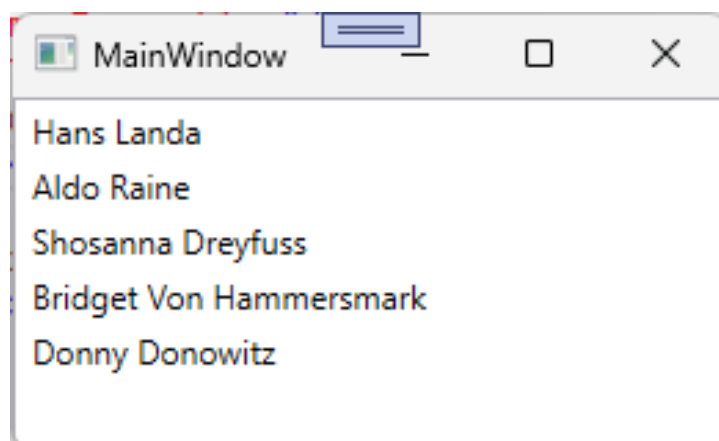
Para ilustrar esto, se va a crear un proyecto WPF con una ventana principal que contenga un ListBox. En primer lugar, se debe crear una lista de cadenas con los elementos que va a contener el control llamada stringList, cargarla en el constructor y asociarla al DataContext de la ventana:

```
private IList<string> stringList = new List<string>();  
public MainWindow()  
{  
    InitializeComponent();  
    stringList.Add("Hans Landa");  
    stringList.Add("Aldo Raine");  
    stringList.Add("Shosanna Dreyfuss");  
    stringList.Add("Bridget Von Hammersmark");  
    stringList.Add("Donny Donowitz");  
    this.DataContext = stringList;  
}
```

Ahora se inserta el ListBox, el cual incluye un ItemsSource por defecto que hay que cambiar dejando su código de la siguiente forma:

```
<ListBox ItemsSource="{Binding}"/>
```

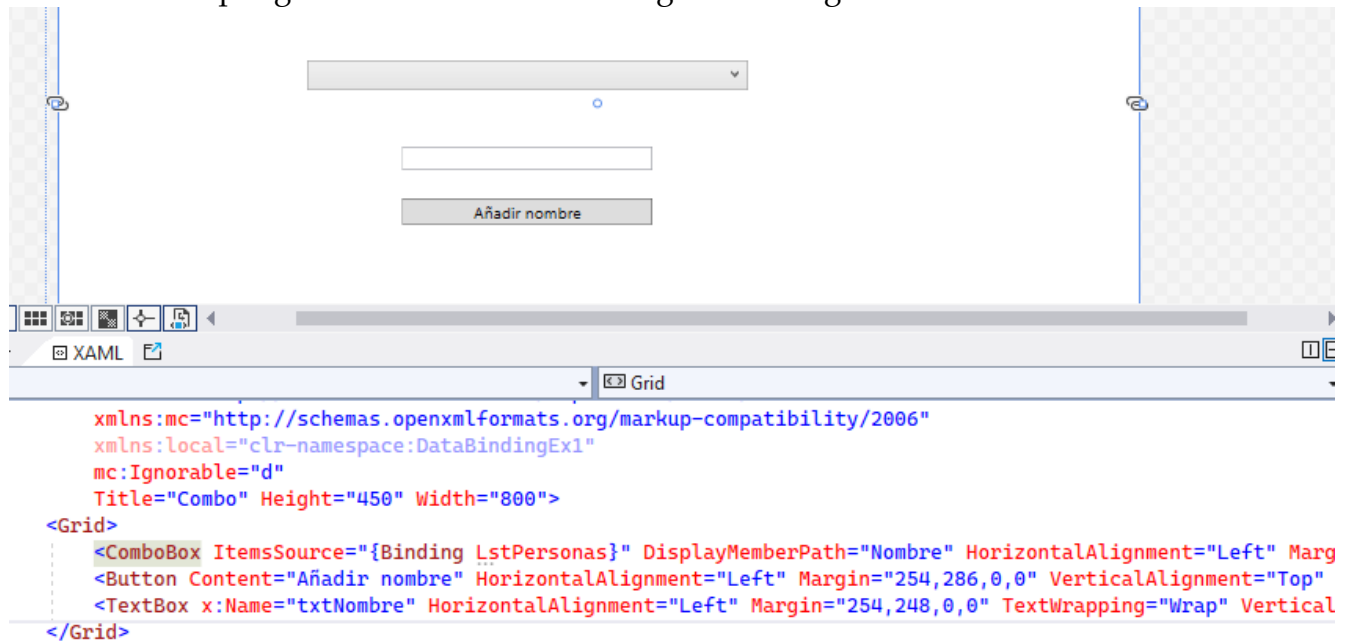
Como podrá verse en la ejecución, el sistema enlaza la lista directamente con el DataContext y muestra los elementos contenidos en él:



8. ListBox con elementos de DataContext

Si la lista fuera de un tipo distinto a los nativos (cadenas, enteros, etc.) y correspondiera a una clase, esto no funcionaría igual ya que llamaría al método ToString() de la clase. Al igual que se hizo en la unidad 3 con Java, simplemente bastaría con sobrescribir dicho método para devolver el texto deseado, aunque se pueden utilizar otras opciones.

Para profundizar en este concepto, se puede visualizar otro ejemplo, pero esta vez con un ComboBox. Supóngase la vista XAML de la siguiente imagen:



#### 9. Interfaz con ComboBox

El cuadro combinado se llena con una lista de personas las cuales pertenecen a una clase Persona que tiene dos propiedades: Id y Nombre. Es muy habitual trabajar con orígenes de datos y tablas maestras que contienen datos estáticos (ciudades, provincias, países, géneros, etc.) y que siguen este formato Id, Nombre (o Descripción). La idea es que el cuadro combinado muestre la propiedad Nombre mientras que internamente guarda un objeto de este tipo.

El código de la clase Persona sería el siguiente:

```
public class Persona
{
    public Persona(int id, string nombre)
    {
        Id = id;
        Nombre = nombre;
    }
    public int Id { get; set; }
    public string Nombre { get; set; }
}
```

Por otra parte, en la vista se añadirá una lista observable de Personas:

```
public ObservableCollection<Persona> LstPersonas { get; set; }
```

Que sea observable significa que el interfaz estará pendiente de la lista para actualizar los cambios que se puedan producir en ella. Es un elemento básico para enlazar controles y datos y forma parte del patrón de diseño [Observer](#).

Ahora, al constructor de la ventana, se le añadirán las siguientes líneas de código:

```
fillList();  
this.DataContext = this;
```

El método fillList() llena la lista observable:

```
private void fillList()  
{  
    LstPersonas = new ObservableCollection<Persona>();  
    LstPersonas.Add(new Persona(1, "Hans Landa"));  
    LstPersonas.Add(new Persona(2, "Aldo Raine"));  
    LstPersonas.Add(new Persona(3, "Shosanna Dreyfuss"));  
    LstPersonas.Add(new Persona(4, "Bridget Von Hammersmark"));  
    LstPersonas.Add(new Persona(5, "Donny Donowitz"));  
}
```

En el ComboBox hay dos atributos que son clave en este enlace:

```
ItemsSource="{Binding LstPersonas}" DisplayMemberPath="Nombre"
```

El ItemsSource determina la fuente de los elementos que formarán parte del ComboBox y DisplayMemberPath se refiere a la propiedad que se mostrará (en este caso, Nombre).

Lo más interesante en este caso es poder añadir nuevos elementos al cuadro combinado y que los refleje en tiempo real. En el interfaz de ejemplo hay un cuadro de texto (txtNombre) y un botón. En este último, se puede programar su evento clic de este modo:

```
private void Button_Click(object sender, RoutedEventArgs e)  
{  
    if (!string.IsNullOrEmpty(txtNombre.Text)) {  
        // Se obtiene el máximo Id de persona y se le suma 1  
        int nuevoId = LstPersonas.Max(p => p.Id)+1;  
        LstPersonas.Add(new Persona(nuevoId, txtNombre.Text));  
    }  
}
```

De este modo, al pulsar el botón, se añade una nueva persona a la lista reflejándose automáticamente en el ComboBox. En este código se ilustra la utilización de expresiones Lambda tal como se hizo en las unidades referidas a Java, aunque esta vez con sintaxis .NET (en realidad es prácticamente idéntica, pero con => en lugar de -> como operador funcional).

Otro ejemplo interesante puede ser un enlace entre controles. Por ejemplo, puede interesar que el texto introducido en un cuadro de texto se enlace con una etiqueta y se muestre a medida que se escribe.

Supóngase esta vista XAML:



#### 10. Ejemplo de enlace entre controles

En la propiedad Content se ubica el código para enlazar el texto de la etiqueta al del cuadro de texto y eso se hace mediante `{Binding ElementName=txtBinding, Path=Text}`. El enlace es muy sencillo; por una parte, recoge con qué elemento se enlaza mediante ElementName y Path indica con qué propiedad, en este caso, Text.

#### 5.2.5.2 El control DataGrid

Este control es muy similar al JTable que se utiliza en Java Swing y sirve para visualizar una lista de datos en forma de rejilla o cuadrícula. Gracias al Binding, el uso de este componente es muy sencillo.

En su forma básica, un DataGrid enlazado a una lista de personas como la del supuesto del ComboBox sería así:

```
<DataGrid ItemsSource="{Binding LstPersonas}" AutoGenerateColumns="True"/>
```

Es decir, su contenido se enlaza a LstPersonas y se añade un atributo AutoGenerateColumns que provoca que las columnas se generen de forma automática basándose en las propiedades de la clase. Aunque indudablemente facilita mucho las cosas, no siempre se desea que se muestren todos los elementos de una clase.

Para este ejemplo, al tener solo dos propiedades, realmente es bastante absurdo ocultar el identificador, pero servirá para ilustrar de qué modo debe hacerse.

Lo primero que debe hacerse es ajustar AutoGenerateColumns a False y después, indicar expresamente la colección de columnas de que va a constar el DataGrid.

Si se desea mantener solo el nombre, el código podría ser como sigue:

```
<DataGrid ItemsSource="{Binding LstPersonas}" AutoGenerateColumns="False">
  <DataGrid.Columns>
    <DataGridTextColumn Header="Nombre completo" Binding="{Binding
Path=Nombre}"/>
  </DataGrid.Columns>
</DataGrid>
```

Es muy interesante ver el autocompletado que ofrece Visual Studio dentro de `DataGrid.Columns`, ya que se puede observar la variedad de tipos de columna que proporciona (hipervínculo, casilla de verificación, etc.).

De igual modo, al estar enlazado con una `ObservableCollection`, el `DataGrid` va a reflejar los cambios en la lista a nivel de agregado y eliminación de elementos, aunque es importante reseñar que tanto en este como en anteriores ejemplos, esto no implica que un cambio en un elemento de la lista se refleje en el componente. Para esto, hay que recurrir a la interfaz [`INotifyPropertyChanged`](#) que sirve para informar de un cambio en una propiedad. Todo esto requiere cambios importantes en la clase `Persona` que quedaría finalmente así:

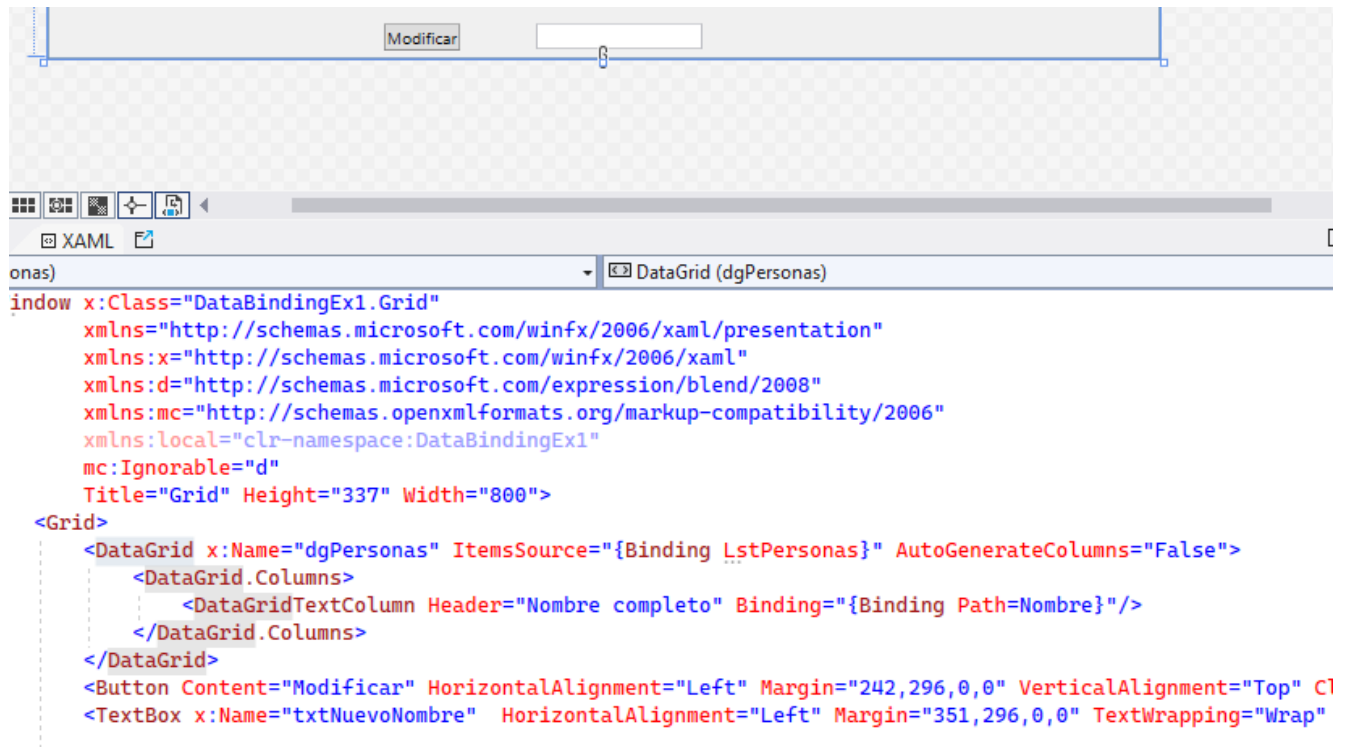
```
public class Persona : INotifyPropertyChanged
{
    private string nombre;
    public Persona(int id, string nombre)
    {
        Id = id;
        this.nombre = nombre;
    }

    public int Id { get; set; }
    public string Nombre {
        get { return nombre; }
        set {
            if (nombre != value)
            {
                nombre = value;
                OnPropertyChanged(nameof(Nombre));
            }
        }
    }
    public event PropertyChangedEventHandler? PropertyChanged;
    protected virtual void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

Como puede verse, `Persona` ahora implementa la interfaz `INotifyPropertyChanged` añadiendo un evento `PropertyChangedEventHandler` “nulable”, es decir, que admite valores nulos (se

indica por la interrogación al final del tipo). Al ajustar el nombre a la persona, se comprueba si ha cambiado y, en ese caso, se llama al evento para que se notifique el cambio de propiedad a los elementos que lo observen.

Ahora podría añadirse un botón y un cuadro de texto que permitiera modificar el nombre de un elemento del DataGrid seleccionado:



11. Interfaz de modificación de registro

En el evento clic del ratón se puede incluir este código para modificar el valor del elemento seleccionado en el DataGrid:

```

private void Button_Click(object sender, RoutedEventArgs e)
{
    if (!string.IsNullOrEmpty(txtNuevoNombre.Text) && dgPersonas.SelectedIndex != -1)
    {
        LstPersonas.ElementAt(dgPersonas.SelectedIndex).Nombre = txtNuevoNombre.Text;
    }
}

```

En este caso se comprueba que el cuadro de texto no esté vacío y que haya algún elemento seleccionado en el DataGrid. En caso de que se cumplan las condiciones, se modifica el elemento de la lista quedando reflejado automáticamente en el DataGrid.

### 5.2.5.3 Validación

En .NET hay diversas opciones para validar la información de, por ejemplo, una entidad. Por ejemplo, supóngase que se quiere añadir a la clase Persona la fecha de nacimiento y esta no puede ser anterior al 1/1/1900 ni posterior al día de ayer (la fecha actual termina a las 23:59).



Por tanto, se añadirá una nueva propiedad FechaNacimiento que también tendrá controlado su cambio de valor para notificar a los elementos que lo observen.

La clase quedaría (de momento) como sigue:

```
public class Persona : INotifyPropertyChanged
{
    public Persona() { }
    private string nombre;
    private DateTime fechaNacimiento;
    public Persona(int id, string nombre, DateTime fechaNac)
    {
        Id = id;
        this.nombre = nombre;
        this.fechaNacimiento = fechaNac;
    }
    public int Id { get; set; }
    public string Nombre
    {
        get { return nombre; }
        set
        {
            if (nombre != value)
            {
                nombre = value;
                OnPropertyChanged(nameof(Nombre));
            }
        }
    }
    public DateTime FechaNacimiento {
        get { return fechaNacimiento; }
        set
        {
            if (fechaNacimiento != value)
            {
                fechaNacimiento = value;
                OnPropertyChanged(nameof(fechaNacimiento));
            }
        }
    }
    public event PropertyChangedEventHandler? PropertyChanged;
    protected virtual void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

En esta clase, se añadirá un interfaz llamado [IDataErrorInfo](#) el cual permitirá añadir funcionalidades para errores personalizados. El encabezado de la clase quedará así junto con los nuevos métodos que requiere el interfaz:

```
public class Persona : INotifyPropertyChanged, IDataErrorInfo
{
    public string Error { get { return string.Empty; } }
    public string this[string columnName] => throw new NotImplementedException();
}
```

La propiedad **Error** contendrá un mensaje indicando qué es lo que no está bien en el objeto. De momento no devolverá más que una cadena vacía.

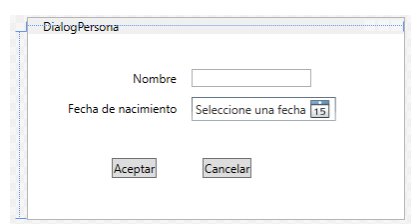
La propiedad que interesa en este caso es la siguiente, que sería lo que se conoce como una propiedad indexada. En C#, un indexador permite a la instancia de una clase actuar como si fuera un array, permitiendo el acceso a los elementos mediante un índice. Aquí se van a introducir las reglas de negocio de la clase de una forma similar a la siguiente:

```
public string this[string columnName]
{
    get
    {
        string result = string.Empty;
        if (columnName.Equals("Nombre") && string.IsNullOrEmpty(nombre))
        {
            result = "Debe introducir un nombre";
        }
        if (columnName.Equals("FechaNacimiento") && fechaNacimiento >= new
DateTime(2010, 1, 1))
        {
            result = "La fecha de nacimiento debe ser anterior al 1/1/2010";
        }
        return result;
    }
}
```

Supóngase que se quiere añadir una funcionalidad de añadido a la lista de nuevas personas. En la ventana donde está el grid se le añade un botón para agregar personas.

```
<Button Content="Añadir persona" HorizontalAlignment="Left" VerticalAlignment="Top"
Click="btnNuevo_Click"/>
```

Ahora se creará un diálogo para introducir los datos de la persona incluyendo su fecha de nacimiento.



12. Diálogo de agregado de personas

Este diálogo utilizará como DataContext un objeto de tipo Persona que se enlazará a los controles correspondientes. Por tanto, en su codebehind se incluirán líneas como las siguientes:

```
public Persona Persona { get; set; }  
public DialogPersona(Persona p) {  
    InitializeComponent();  
    Persona = p;  
    this.DataContext = Persona;  
}
```

Es decir, se creará una propiedad Persona propia del diálogo y se asignará el DataContext a ella. La persona se puede proporcionar en el constructor para que, al instanciarlo en el formulario anterior, se pueda proporcionar el objeto que será finalmente añadido. Los botones de aceptar y cancelar se quedarían así:

```
private void btnCancelar_Click(object sender, RoutedEventArgs e) {  
    Persona.Id = -1;  
    this.Close();  
}  
private void btnAceptar_Click(object sender, RoutedEventArgs e) {  
    this.Close();  
}
```

Con esta codificación, en ambos botones se cierra el diálogo, pero en la de cancelar, se le da a la persona un identificador -1. En cuanto al cuadro de texto que contiene el nombre y al DatePicker de la fecha, para que enlacen con el objeto Persona hay que hacerles unos retoques que incluyen también los atributos NotifyOnValidationError y ValidatesOnDataErrors para tener en cuenta las validaciones a nivel de clase creadas anteriormente.

```
<TextBox HorizontalAlignment="Left" Text="{Binding Path=Nombre,  
NotifyOnValidationError=True, ValidatesOnDataErrors=True}" TextWrapping="Wrap"  
VerticalAlignment="Top" Width="120"/>  
<DatePicker HorizontalAlignment="Left" DisplayDate="{Binding Path=FechaNacimiento,  
NotifyOnValidationError=True, ValidatesOnDataErrors=True}" VerticalAlignment="Top"/>
```

Volviendo al formulario original, el botón de Nueva persona tendría una codificación como esta:

```
private void btnNuevo_Click(object sender, RoutedEventArgs e)  
{  
    Persona p = new Persona();  
    p.Id = LstPersonas.Max(p => p.Id) + 1;  
    p.FechaNacimiento = DateTime.Now;  
    DialogPersona dlg = new DialogPersona(p);  
    dlg.ShowDialog();  
    if (dlg.Persona != null && dlg.Persona.Id > 0)  
    {  
        LstPersonas.Add(dlg.Persona);  
    }  
}
```

Lo interesante de esta parte de validación es que también se pueden programar manejadores de eventos para controlar si se ha producido un error. En este caso, se programará un manejador que controle si hay uno o más errores y, en ese caso, deshabilitará el botón aceptar. En caso contrario, lo habilitará. El manejador de evento se declara en el atributo `Validation.Error` dentro del control (en este supuesto, en el `TextBox` y en el `DatePicker`).

`Validation.Error="Validation_Error"`

Y el código del evento puede ser el siguiente:

```
private void Validation_Error(object sender, ValidationErrorEventArgs e){
    if (e.Action == ValidationErrorEventAction.Added)
        errores++;
    else
        errores--;
    if (errores == 0)
        btnAceptar.IsEnabled = true;
    else
        btnAceptar.IsEnabled = false;
}
```

Además, hay que añadir la variable errores a la clase diálogo:

```
private int errores = 0;
```

## 5.2.6. UBICACIÓN Y ALINEAMIENTO. CONTENEDORES

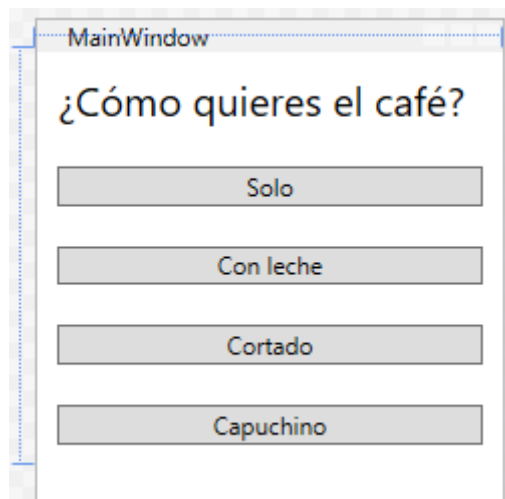
La ordenación de controles siguiendo un posicionamiento fijo basado en coordenadas absolutas determinadas por una dirección expresada en píxeles es muy poco fiable al existir varias formas de representar la interfaz de acuerdo con la resolución de pantalla, sistema operativo, etc. Cuando se busca crear una interfaz válida para todo tipo de entornos, con independencia de los factores ya mencionados, es necesario utilizar estructuras denominadas contenedores, los cuales van a permitir organizar la información. A continuación, se ven diferentes tipos de contenedores que es posible usar.

### **StackPanel**

Aplicado principalmente para el diseño de listas, permite apilar los elementos bien uno junto a otro (orientación horizontal) o encima uno de otro (orientación vertical). Estos elementos presentan una gran utilidad a la hora de usar botones de aceptar y/o cancelar situados en la parte inferior o superior de la interfaz. El contenedor permite ignorar los cambios de fuente y/o tamaño que se puedan realizar al manejar entre ellos la distancia y alineación jugando con los márgenes asociados a cada elemento a través de la propiedad `margin`.

### Ejemplo 1:

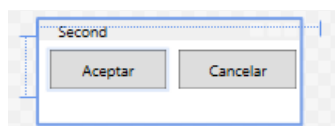
```
<StackPanel>
  <TextBlock Text="¿Cómo quieres el café?" Margin="10" FontSize="20"/>
  <Button Content="Solo" Margin="10"/>
  <Button Content="Con leche" Margin="10"/>
  <Button Content="Cortado" Margin="10"/>
  <Button Content="Capuchino" Margin="10"/>
</StackPanel>
```



13. Vista de diseño del ejemplo 1 de StackPanel

### Ejemplo 2:

```
<StackPanel Margin="8,8,8,8" Orientation="Horizontal">
  <Button Content="Aceptar" MinWidth="93"/>
  <Button Content="Cancelar" Margin="10,0,0,0" MinWidth="93"/>
</StackPanel>
```



14. Vista de diseño del ejemplo 2 de StackPanel

## DockPanel

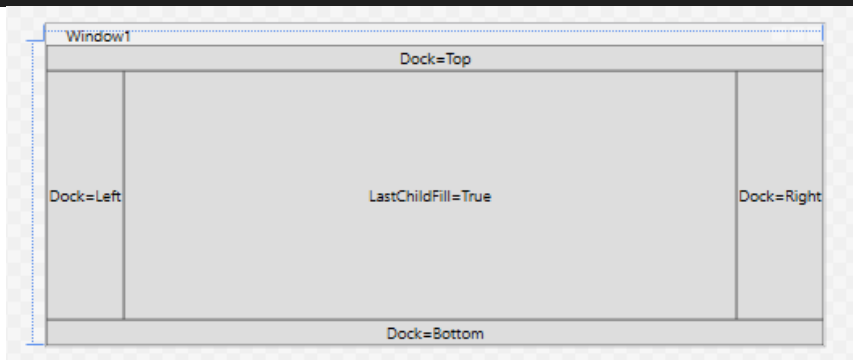
Permite anclar los elementos en los márgenes izquierdo, derecho, superior o inferior, de tal forma que permanecen en una determinada ubicación con independencia de si se redimensiona la ventana o el contenedor.

Una opción muy habitual es establecer un elemento que ocupe todo el contenido restante, principalmente en la gestión de las ventanas de Windows en el explorador de archivos o gestores de correo, una parte izquierda con las opciones del menú y el resto para el contenido. Esta opción se configura a través de la propiedad `LastChildFill` a `true`.

Otro aspecto muy importante a la hora de manejar este contenedor es el orden en el que son definidos los elementos, ya que condicionan el resto. El primer elemento definido puede determinar sus propiedades de ancho o alineación y el resto ocupará el espacio sobrante.

Ejemplo 1:

```
<DockPanel LastChildFill="True">
  <Button Content="Dock=Top" DockPanel.Dock="Top"/>
  <Button Content="Dock=Bottom" DockPanel.Dock="Bottom"/>
  <Button Content="Dock=Left" />
  <Button Content="Dock=Right" DockPanel.Dock="Right" />
  <Button Content="LastChildFill=True"/>
</DockPanel>
```



15. Vista de diseño del ejemplo 1 de DockPanel

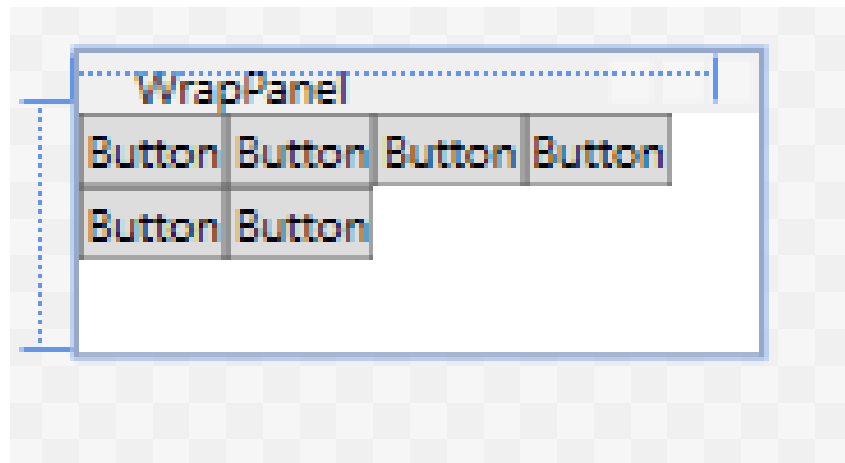
Si no se ajusta la propiedad Dock en un control, se asume que es Left.

## WrapPanel

Agrupar los elementos siguiendo una orientación vertical u horizontal, pero a diferencia de StackPanel, este contenedor sólo permite los elementos que caben en una fila y entonces los ordena en la siguiente.

La salida producida por este contenedor es la misma que la generada en el explorador de archivos de un sistema operativo Windows a la hora de escoger una vista basada en iconos.

```
<WrapPanel Orientation="Horizontal">
  <Button Content="Button"/>
  <Button Content="Button"/>
  <Button Content="Button"/>
  <Button Content="Button"/>
  <Button Content="Button"/>
  <Button Content="Button"/>
</WrapPanel>
```

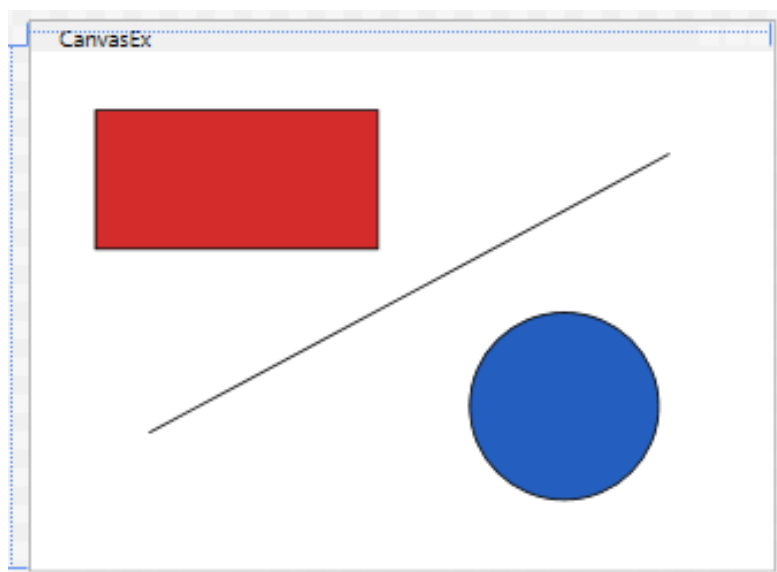


16. Vista de diseño del ejemplo de WrapPanel

## Canvas

El elemento con mayor funcionalidad y que más opciones brinda, ya que permite ubicar los elementos dentro de unas coordenadas específicas. De todas formas, las coordenadas pueden ser especificadas de acuerdo a coordenadas relativas utilizando las referencias Canvas.Left, Canvas.Right, etc.

```
<Canvas>
  <Rectangle Height="75" Canvas.Left="34" Stroke="Black" Canvas.Top="31"
Width="152" Fill="#FFD42B2B"/>
  <Ellipse Height="101" Canvas.Left="234" Stroke="Black" Canvas.Top="139"
Width="102" HorizontalAlignment="Left" VerticalAlignment="Center" Fill="#FF245FBF"/>
  <Path Data="M62.556054,202.98619 C341.03139,54.667803 340.53144,54.167993
340.53144,54.167993" Fill="#FFF4F4F5" Height="150" Canvas.Left="62.556"
Stroke="Black" Stretch="Fill" Canvas.Top="54.168" Width="279"/>
</Canvas>
```

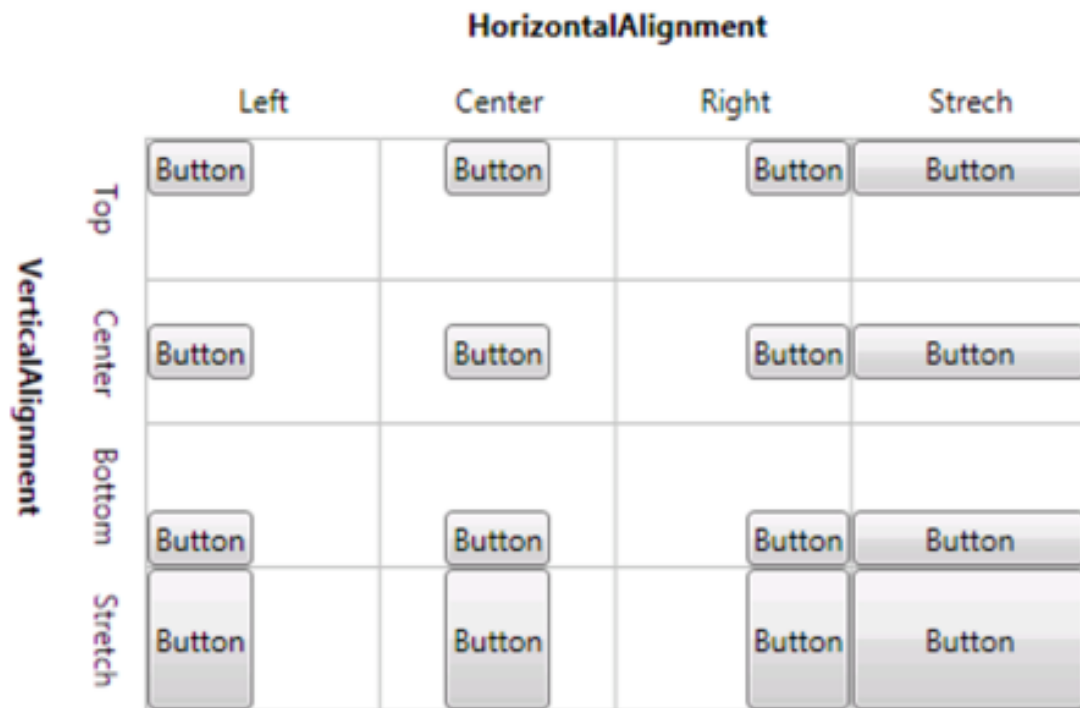


17. Ejemplo de figuras con Canvas



Nota: Para incluir la recta en el diseño anterior, es preciso abrir la herramienta Blend y usar la pluma.

En cuanto a la alineación de elementos, en la siguiente figura se muestran las distintas disposiciones que puede tener un botón en un Panel.



18. Alineamiento de componentes

## 5.2.7. EVENTOS Y CONTROLADORES

### 5.2.7.1 Eventos enrutados

WPF utiliza una mejora importante a la clásica concepción del manejo de eventos en .NET, los eventos enrutados. Por ejemplo, se puede imaginar la situación en la que se tiene un botón que consta de una imagen de fondo y un texto. Esto implica que el control tiene una serie de elementos discretos que lo forman: un bloque de texto, una imagen y la forma básica y fondo del botón.

Se trata entonces de una colección de elementos separados entre ellos, con lo que el manejo de eventos a priori parece ser algo complicado. No solo se trata de reaccionar al clic del fondo del botón sino también al de su texto y al de la imagen. Aquí es donde entran en juego los eventos enrutados, ya que son capaces de llamar a manejadores de eventos en todo el árbol visual. Esto quiere decir que se puede implementar un manejador de eventos al nivel Button y se puede confiar en que un clic en cualquiera de sus elementos acabará encontrando el manejador correspondiente a dicha acción.

Existen tres categorías de eventos enrutados en WPF:

- Eventos de burbuja. Estos eventos viajan por el árbol visual comenzando en el elemento receptor inicial.
- Eventos de túnel. Comienzan en la parte superior del árbol visual del control y se mueven hacia abajo hasta que alcanzan el elemento receptor.
- Eventos directos. Son los equivalentes a los eventos estándar en .NET: solo se llamará al manejador de eventos para el elemento receptor.

Los eventos por sí mismos se pueden declarar en XAML o en código. En esta porción de código se puede ver un botón con un evento `MouseEnter` definido:

```
<Button MouseEnter="button1_MouseEnter" Height="23" Name="button1" Width="75">  
OK</Button>
```

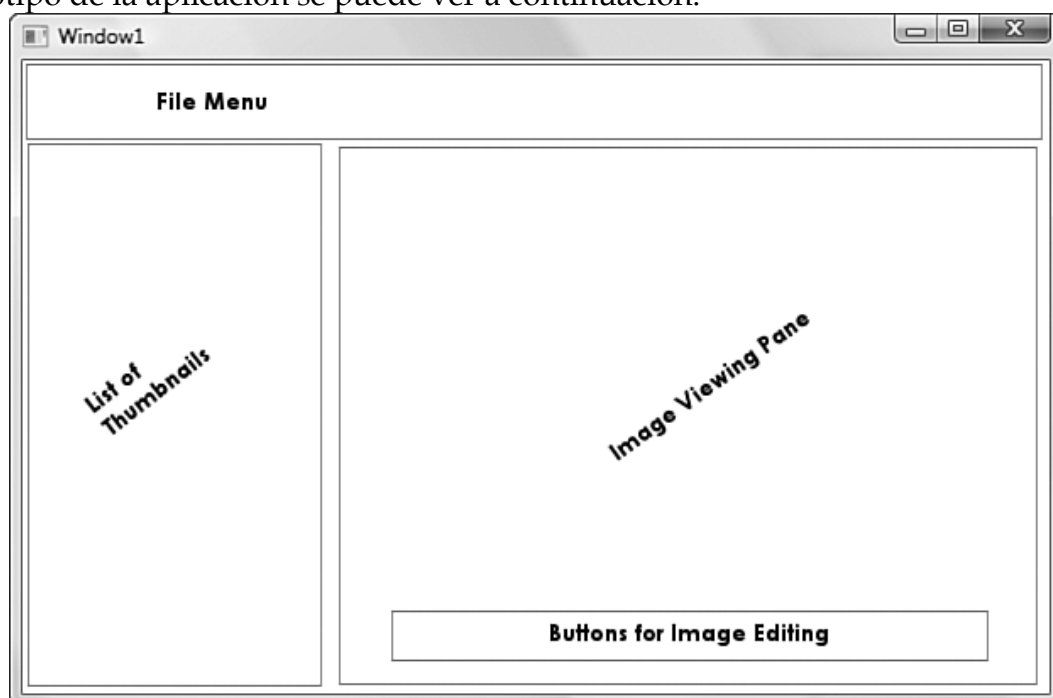
El manejador de eventos en C# es igual que cualquier otro manejador .NET:

```
private void button1_MouseEnter(object sender, MouseEventArgs e)  
{  
    MessageBox.Show("MouseEnter on button1");  
}
```

### 5.2.7.2 Ejemplo: Visor de imágenes sencillo

Para ilustrar cómo se puede diseñar una aplicación WPF nada mejor que un ejemplo. En este caso, se trata de una aplicación para visualizar imágenes. En ella se va a mostrar una lista de vistas previas de imágenes y, una vez seleccionada una de ellas, se podrá ver la imagen en sí misma e incluso realizar cambios sobre ella.

Un prototipo de la aplicación se puede ver a continuación:



19. Prototipo de visor de imágenes

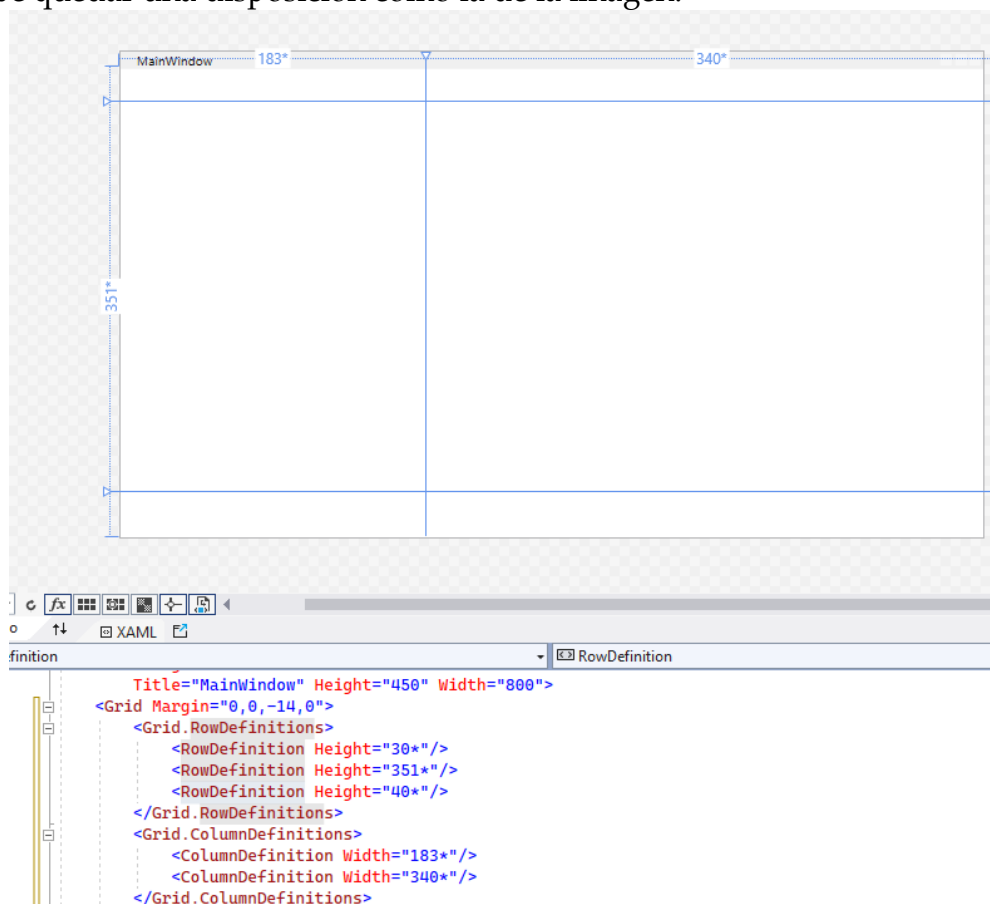
Los requisitos base son:

- Cuando se carga la aplicación, esta obtendrá las imágenes contenidas en el directorio especificado.
- Cada imagen se mostrará en un cuadro de lista que contendrá miniaturas de las imágenes sin texto
- Cuando el usuario haga clic sobre cualquiera de los elementos en el cuadro de lista, el área de visualización mostrará la imagen seleccionada.
- El usuario podrá entonces manipular la imagen: se puede aplicar un efecto de blanco y negro, rotar a favor o en contra de las agujas del reloj, voltear o reflejar.

Por tanto, para comenzar, primeramente, se crea una aplicación WPF que tendrá como nombre **ImageViewer**.

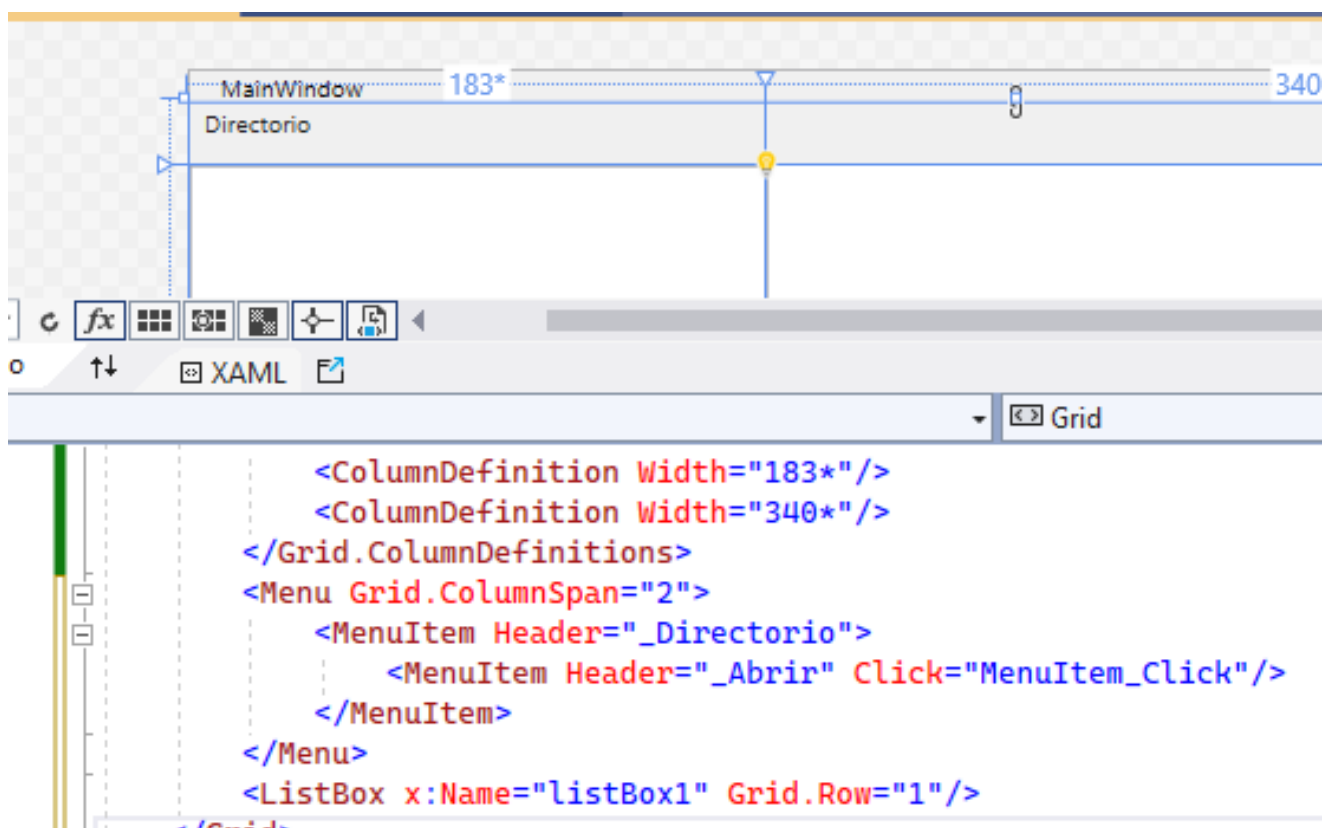
La ventana principal contendrá un Grid (aparece por defecto) y algunos StackPanel o WrapPanel. En primer lugar, se selecciona el Grid (en diseño o código) y se hace clic sobre el borde del encabezado para crear un borde de columna. Como puede verse, el Grid ahora tiene dos columnas y se debe ajustar el ancho de la primera para contener las miniaturas. Del mismo modo se pueden crear filas, pero esta vez haciendo clic sobre el borde izquierdo del Grid.

Al final debe quedar una disposición como la de la imagen:



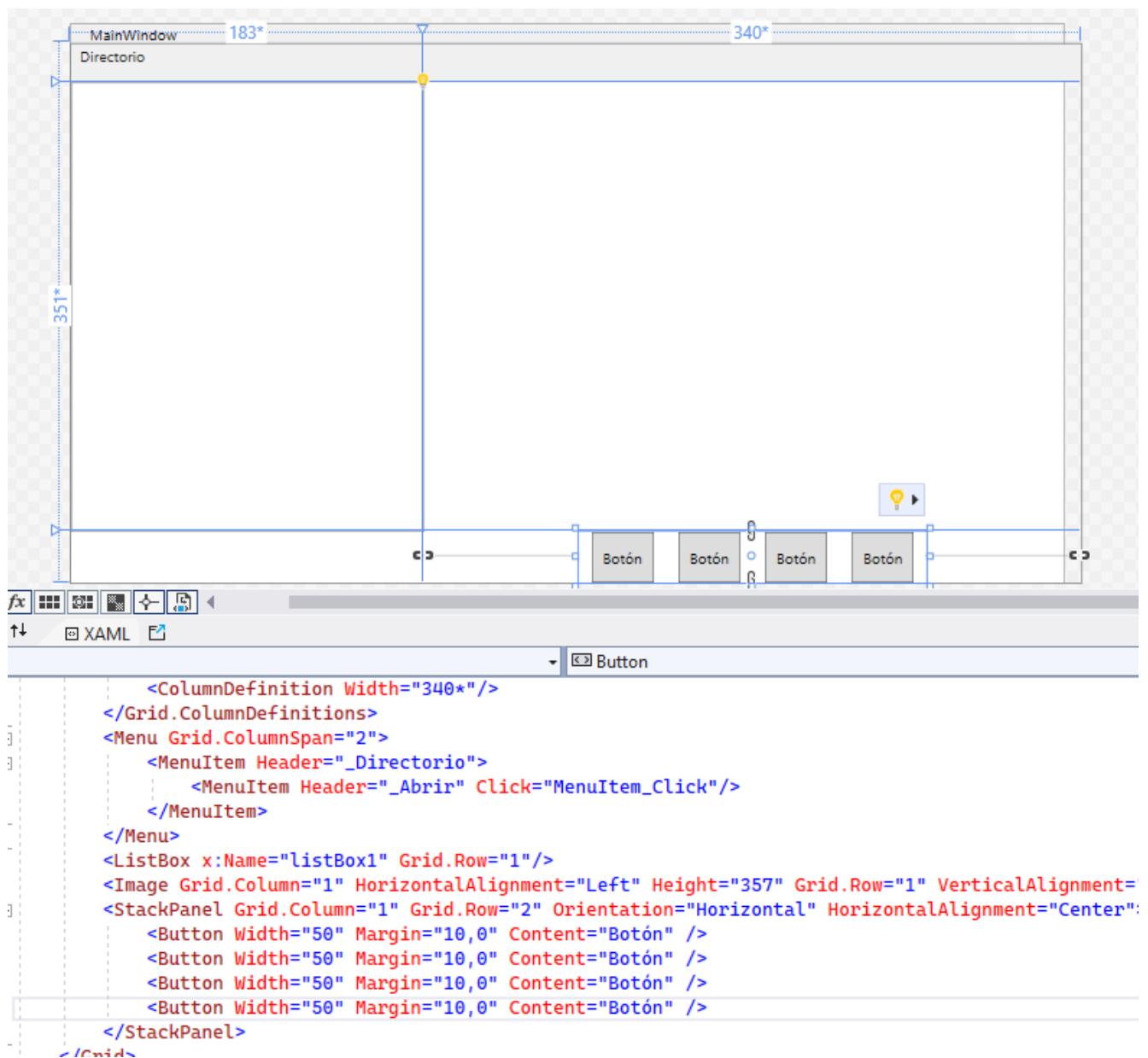
20. Disposición inicial de ImageViewer

A continuación, arrastrar un ListBox a la fila del medio en la columna de la izquierda y un control de menú en la fila superior, columna izquierda. Ya que se quiere que el menú ocupe el ancho entero del Grid, se necesitará cambiar manualmente el XAML en este punto especificando **Grid.ColumnSpan="2"** dentro del elemento Menu. Hay que asegurarse de que no hay ningún margen ni en la lista ni en el menú para permitir que este último se expanda y contraiga a la vez que lo hace el Grid dejando además que el menú ocupe todo el espacio de la primera fila. Dado que se necesitará seleccionar un directorio del sistema de archivos, se puede crear una entrada de menú llamada **Directorio** y una subentrada **Abrir**. Se incluye un elemento Menu y dentro de él un MenuItem con la propiedad Header como **\_Directorio** (se antepone un guion bajo). El MenuItem se añade en la colección Items del propio menú y del mismo modo se agrega el **\_Abrir** como MenuItem con la misma operativa. En este último Item se añade un evento Click.



21. Configuración de menú en ImageViewer

Para visualizar las imágenes, se usará un control de tipo Image. Hay que arrastrarlo dentro de la segunda fila columna derecha y asegurarse de que no hay márgenes en propiedades. Finalmente, se arrastra un StackPanel en la tercera fila, columna a la derecha y sin márgenes con orientación horizontal y alineación horizontal en Centro. Este panel es donde se ubicarán los botones de manipulación de imagen. Para ello, se arrastran cuatro botones en el StackPanel y se ajustan sus márgenes y altura/anchura hasta que tenga un aspecto operativo.



22. Configuración de botones e imagen

## Almacenando las imágenes

Una vez ubicados los elementos, ahora toca ajustar los elementos para el enlace de datos. La primera tarea consiste en almacenar los ficheros en una colección. Para este supuesto viene bien una clase en **System.Windows.Media.Imaging** que se ajusta a los requisitos de la aplicación: [BitmapSource](#). Como colección una `List<BitmapSource>` debería servir de momento, pero se necesita una forma de publicar la lista. De esta forma se creará una clase envoltorio que cargue la lista y la expone como propiedad.

Se añade una clase al proyecto con el siguiente código:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Windows.Media.Imaging;
namespace ImageViewer
{
    public class DirectoryImageList{
        private string path;
        private List<BitmapSource> images = new List<BitmapSource>();
        public DirectoryImageList(string path){
            this.path = path;
            loadImages();
        }
        public List<BitmapSource> Images {
            get{ return images; }
            set{ images = value; }
        }
        public string Path {
            get { return path; }
            set { path = value;
                loadImages();
            }
        }
        private void loadImages() {
            images.Clear();
            BitmapImage img;
            Array.ForEach(Directory.GetFiles(path), f =>{
                try {
                    img = new BitmapImage(new Uri(f));
                    images.Add(img);
                }
                catch { // Se deja vacío; ignorando cualquier fichero que no se pueda
cargar como imagen
                }
            });
        }
    }
}
```

El método loadImages es donde se encuentra la lógica más importante ya que se encarga de enumerar los ficheros dentro de un directorio dado e intenta cargarlos en un objeto BitmapImage. Si lo hace con éxito, se sabe que es un fichero de imagen. En caso contrario, se ignora la excepción resultante y se continúa normalmente.

De vuelta a la clase MainWindow, se necesitan crear algunos campos privados para mantener una instancia de esta clase recién creada y para mantener la ruta que esté seleccionada en ese momento por el usuario. Esto es algo que se hace mediante un diálogo común lanzado desde

la opción de menú Directorio -> Abrir. A continuación, se muestran las variables miembro a introducir en la clase dentro del fichero MainWindow.xaml.cs:

```
private DirectoryImageList imgList;  
private string path =  
Environment.GetFolderPath(Environment.SpecialFolder.MyPictures);
```

Nótese que se apunta como directorio por defecto a “Mis Imágenes”. Para cargar el objeto lista, se escribirá un método **resetList** en esta misma clase:

```
private void resetList()  
{  
    if (IsValidPath(path))  
    {  
        imgList = new DirectoryImageList(path);  
    }  
    this.DataContext = imgList.Images;  
}  
private bool IsValidPath(string path)  
{  
    try  
    {  
        string folder = System.IO.Path.GetFullPath(path);  
        return true;  
    }  
    catch {  
        return false;  
    }  
}
```

Ahora hay que realizar una serie de ajustes al constructor para llamar a este método recién creado y asociar el DataContext a la lista de imágenes:

```
public MainWindow()  
{  
    InitializeComponent();  
    resetList();  
}
```

Para que el ListBox enlace con el DataContext, hay que añadir el valor {Binding} a su propiedad ItemsSource:

```
<ListBox x:Name="listBox1" Grid.Row="1" ItemsSource="{Binding}"/>
```

Si se ejecuta ahora la aplicación, se verá un contenido familiar pero no en el formato que se desearía a priori ya que muestra las rutas en forma de texto y lo que se necesita es que sean imágenes. Hay que recurrir entonces a crear un elemento Resources en la ventana y crear una DataTemplate que encaje perfectamente con el formato de miniatura deseado.

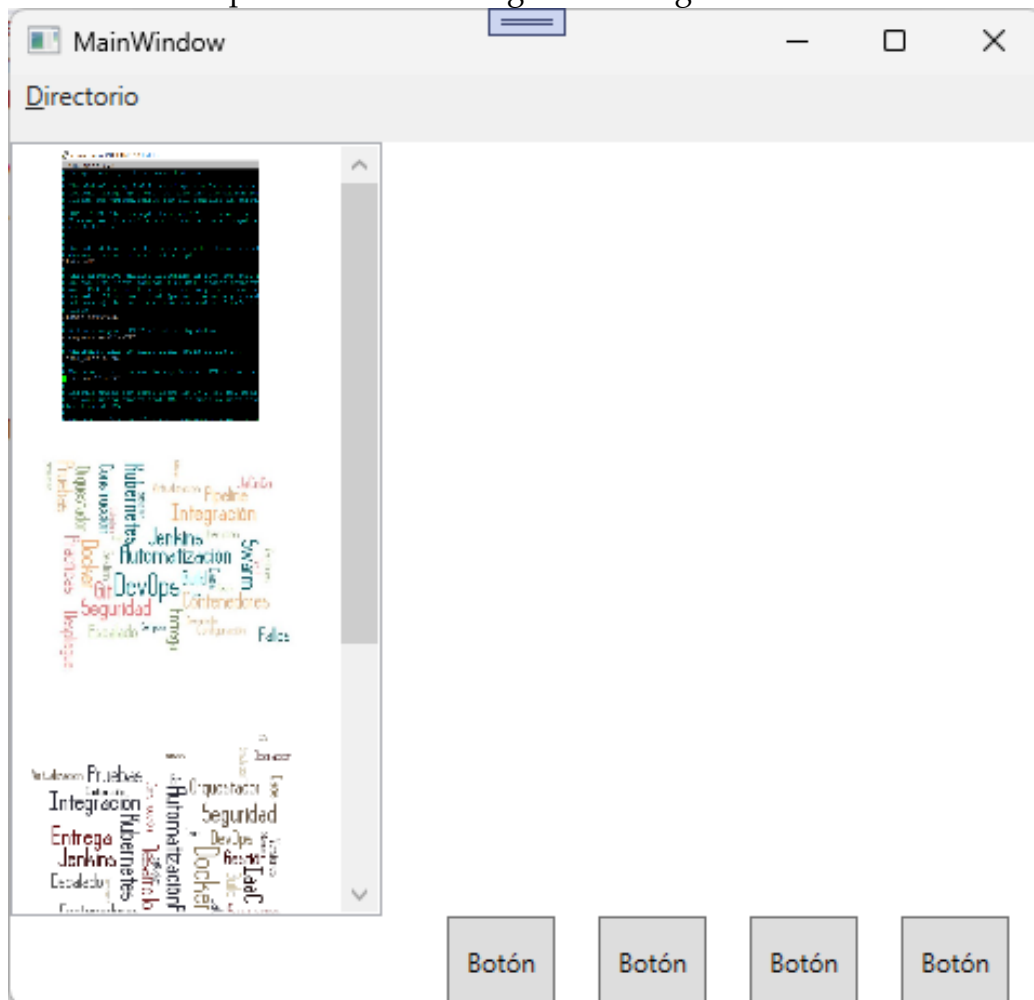


```
<Window.Resources>
    <DataTemplate x:Key="ImageDataTemplate">
        <Image Source="{Binding UriSource.LocalPath}" Width="125" Height="125"/>
    </DataTemplate>
</Window.Resources>
```

Este código debe introducirse justo después del elemento Window de MainWindow.xaml. Ahora vuelve a modificarse el enlace del ListBox haciendo referencia al DataTemplate:

```
<ListBox x:Name="listBox1" Grid.Row="1" ItemsSource="{Binding}"
ItemTemplate="{StaticResource ImageDataTemplate}"/>
```

En la plantilla de datos, el elemento imagen espera una URI para cada imagen, con lo que se usa el UriSource.LocalPath que proporciona el objeto BitmapSource. Ahora el interfaz ya parece otra cosa tal como puede verse en la siguiente imagen:



23. ImageViewer con miniaturas en la lista

Hacer clic sobre una miniatura en la lista debería cargar la imagen en el control central. Si se crea un manejador de evento SelectionChanged y se enlaza al ListBox, se puede actualizar la propiedad Imagen.Source correspondiente. El evento se declara dentro del elemento ListBox:

```
<ListBox x:Name="listBox1" SelectionChanged="listBox1_SelectionChanged" Grid.Row="1"
ItemsSource="{Binding}" ItemTemplate="{StaticResource ImageDataTemplate}"/>
```

Ahora se ajusta el manejador de evento en el código C# de la vista (se da por hecho que la imagen está nombrada como image1):

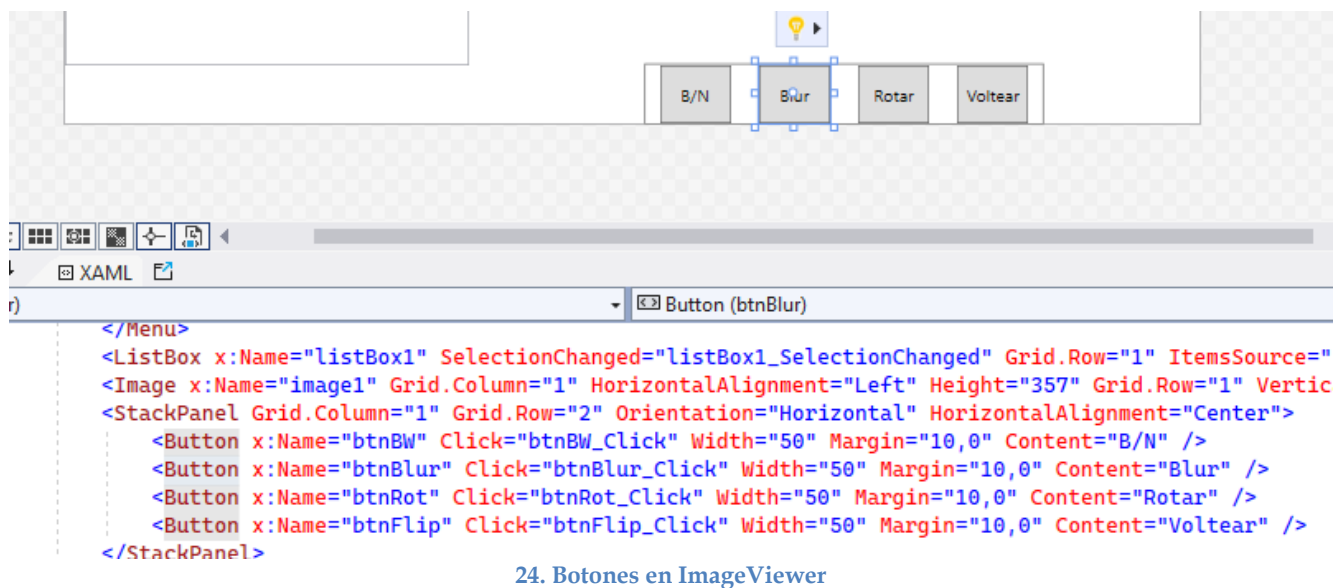
```
private void listBox1_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    image1.Source = (BitmapSource)((sender as ListBox).SelectedItem);
}
```

## Manejadores de evento de botones y efectos de imagen

Una vez cargadas las imágenes y mostradas en el control correspondiente, ahora hay que centrarse en los cuatro botones de edición y efectos los cuales incluirán:

- Un filtro de blanco y negro
- Difuminación de imagen
- Rotación
- Volteo

Por tanto, se etiquetarán y nombrarán los botones además de ajustar sus eventos clic (operación sencilla con autocompletado dentro del código XAML).



Para manipular las imágenes, se emplea lo que se conoce como transformación, que viene a ser una manipulación de una superficie 2D para rotar, inclinar o cualquier otro cambio que afecte a la apariencia de la superficie.

La rotación se puede hacer directamente mediante [RotateTransform](#) tal como se muestra en el código siguiente:

```
private void btnRot_Click(object sender, RoutedEventArgs e){
    CachedBitmap cache = new CachedBitmap((BitmapSource)image1.Source,
    BitmapCreateOptions.None, BitmapCacheOption.OnLoad);
    image1.Source = new TransformedBitmap(cache, new RotateTransform(90));
}
```

El volteo también es fácil, pero usando [ScaleTransform](#):

```
private void btnFlip_Click(object sender, RoutedEventArgs e){
    CachedBitmap cache = new CachedBitmap((BitmapSource)image1.Source,
    BitmapCreateOptions.None, BitmapCacheOption.OnLoad);
    ScaleTransform scale = new ScaleTransform(-1, -1, image1.Source.Width / 2,
    image1.Source.Height / 2);
    image1.Source = new TransformedBitmap(cache, scale);
}
```

El difuminado se proporciona mediante un mecanismo llamado efecto. Creando una instancia. Creando un objeto de tipo [BlurEffect](#) y asignándolo a la imagen, WPF aplicará el algoritmo apropiado para difuminar la imagen:

```
private void btnBlur_Click(object sender, RoutedEventArgs e)
{
    if (image1.Effect != null)
    {
        // Si hay un efecto, lo elimina
        image1.Effect = null;
    }
    else
    {
        // En caso contrario, añade el efecto a la imagen
        image1.Effect = new BlurEffect();
    }
}
```

El código para el botón de B/N quedaría así:

```
private void btnBW_Click(object sender, RoutedEventArgs e)
{
    BitmapSource img = (BitmapSource)image1.Source;
    image1.Source = new FormatConvertedBitmap(img, PixelFormats.Gray16,
    BitmapPalettes.Gray256, 1.0);
}
```

## Selección de ruta mediante CommonDialog

La última funcionalidad que queda de la aplicación debe permitir cambiar la ruta de los ficheros de imagen. WPF por sí mismo no tiene clases integradas para diálogos que permitan manejar esto, pero se pueden utilizar las de **System.Windows.Forms** (el equivalente a Swing en .NET). Concretamente, la clase **FolderBrowserDialog** lanzada desde el menú Abrir. Para ello, hay que modificar las propiedades del proyecto en su apartado General y marcar **Windows Forms / Habilitar Windows Forms** para este proyecto.

El código de la entrada del menú llamará a un método privado creado para la ocasión:

```
private void MenuItem_Click(object sender, RoutedEventArgs e)
{
    setPath();
}
private void setPath()
{
    FolderBrowserDialog dlg = new FolderBrowserDialog();
    dlg.ShowDialog();
    path = dlg.SelectedPath;
    resetList();
}
```

### 5.2.7.3 Ejemplo: Acceso a datos desde un archivo CSV mediante Entity Framework

En este apartado se va a desarrollar una aplicación que accederá a datos contenidos en un archivo .CSV. Vaya por delante que este ejemplo está aplicado a este tipo de ficheros pero que podría migrarse sin problema a otro origen de datos que bien podría consistir en la conexión a bases de datos como SQL Server, Oracle, PostgreSQL o MariaDB. Ello se debe a que se utiliza el ORM desarrollado por Microsoft llamado [Entity Framework](#). Para trabajar con archivos CSV se utilizará [CSVHelper](#), una biblioteca para .NET que permite la lectura y escritura sobre archivos de valores separados por comas (CSV). Se creará, por tanto, un proyecto de aplicación WPF llamado **VacasWPF**.

El ejemplo consistirá en realizar operaciones CRUD sobre un archivo que contiene información sobre vacas en el Principado de Asturias. Su estructura se puede ver en la siguiente imagen:

```
id;nomMunicipio;f_nacim;f_destete;alzada;peso;sexo;tipo
1;Navia;08/04/2004;12/08/2004;94;175;H;C
2;Grado;09/06/2007;28/11/2007;104;253;H;C
3;Somiedo;10/03/2004;07/07/2004;0;190;H;C
4;Lena;02/06/2009;16/12/2009;106;218;H;N
5;Salas;12/06/2006;02/01/2007;106;262;M;C
6;Llamas de la Ribera;12/09/2005;27/01/2006;102;213;H;AC
7;Belmonte de Miranda;28/03/2006;16/08/2006;0;160;H;C
8;Cangas del Narcea;14/02/2012;27/06/2012;100;161;H;C
9;Regueras (Las);30/03/2010;22/10/2010;100;199;H;C
10;Salas;12/06/2009;24/02/2010;108;315;M;C
11;Lena;05/04/2006;01/12/2006;105;222;H;N
12;Tineo;18/08/2006;20/11/2006;89;131;M;C
13;Somiedo;10/10/2003;10/04/2004;0;190;H;C
14;Cangas del Narcea;25/02/2010;06/09/2010;102;187;H;C
15;Regueras (Las);08/03/2008;21/10/2008;107;224;M;N
...
```

25. Estructura de vacas.csv

Un ejemplo de este archivo se puede descargar en la siguiente dirección:

<https://github.com/juandbp2dam/VacasWPF/blob/main/VacasWPF/Data/vacas.csv>

Para instalar los paquetes requeridos, hay que acudir a la opción **Administrar paquete NuGet** en el menú contextual del proyecto e instalar los paquetes que muestra la imagen.

Paquetes de nivel superior (3)



**CsvHelper** por Josh Close

A library for reading and writing CSV files. Extremely fast, flexible, and easy to use. Supports reading and writing of custom class objects.



**Microsoft.EntityFrameworkCore** por Microsoft

Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL, and other databases through a provider plugin API.



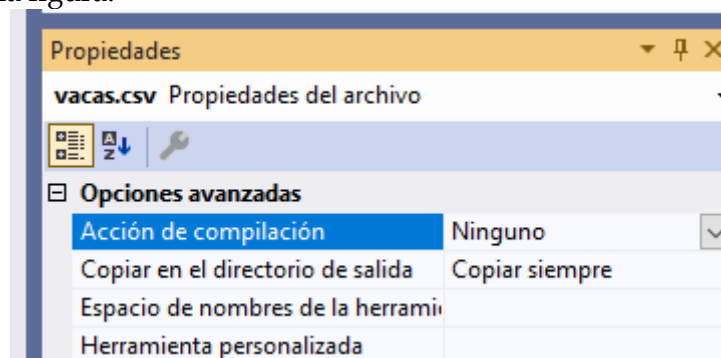
**Microsoft.EntityFrameworkCore.InMemory** por Microsoft

In-memory database provider for Entity Framework Core (to be used for testing purposes).

## 26. Paquetes para CSV y Entity Framework

De Entity Framework se requiere el núcleo y la parte encaminada a trabajar con entidades en memoria, ya que es el modelo elegido en este caso. La idea es manipular los datos en memoria y hacerlos persistir al CSV cuando así se requiera.

En primer lugar, se creará un directorio **Data** dentro del proyecto en el que alojar el fichero CSV, de forma que esté lo suficientemente separado de la lógica de programación. En las propiedades del fichero, hay que realizar una modificación para que siempre se copie en el directorio resultado de la compilación. En la propiedad **Copiar en el directorio de salida**, se ajustará el valor **Copiar siempre** tal como se puede ver en la figura.



## 27. Copiar siempre en fichero CSV

A continuación, se va a crear un nuevo directorio llamado **Models** donde se ubicarán aquellas clases para manejo de los modelos de datos del sistema.

Para trabajar con los registros del fichero, se va a necesitar una clase que ejerza como **DataObject** donde se ubicarán, no solo las propiedades requeridas, sino la lógica necesaria para manejar otras operaciones. A esta clase se la denominará **Vaca**.

El código de la clase Vaca será el siguiente:

```
using CsvHelper.Configuration.Attributes;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

namespace VacasWPF.Models
{
    public class Vaca : IEditableObject
    {
        #region Constructores

        public Vaca()
        {
        }

        public Vaca(int id, string nomMunicipio, DateOnly f_nacim,
                    DateOnly f_destete, int alzada, int peso, string sexo, string tipo)
        {
            this.id = id;
            this.nomMunicipio = nomMunicipio;
            this.f_nacim = f_nacim;
            this.f_destete = f_destete;
            this.alzada = alzada;
            this.peso = peso;
            this.sexo = sexo;
            this.tipo = tipo;
        }

        #endregion

        #region Propiedades

        [Key]
        public int id { get; set; }
        public string nomMunicipio
        { get; set; }
        [Format("dd/MM/yyyy")]
        public DateOnly f_nacim { get; set; }
        [Format("dd/MM/yyyy")]
        public DateOnly f_destete { get; set; }
        public int alzada { get; set; }
        public int peso { get; set; }
        public string sexo { get; set; }
        public string tipo { get; set; }

        #endregion
    }
}
```

```
#region Implementación de interfaces

private string oldNomMunicipio;
private DateOnly oldFNacim;
private DateOnly oldFDestete;
private int oldAlzada;
private int oldPeso;
private string oldSexo;
private string oldTipo;

public void BeginEdit()
{
    oldNomMunicipio = this.nomMunicipio;
    oldFDestete = this.f_destete;
    oldAlzada = this.alzada;
    oldFNacim = this.f_nacim;
    oldPeso = this.peso;
    oldSexo = this.sexo;
    oldTipo = this.tipo;
}

public void CancelEdit()
{
    this.nomMunicipio = oldNomMunicipio;
    this.alzada = oldAlzada;
    this.f_destete = oldFDestete;
    this.f_nacim = oldFNacim;
    this.peso = oldPeso;
    this.sexo = oldSexo;
    this.tipo = oldTipo;
}

public void EndEdit()
{
}

}
#endregion
}
```

En el código hay elementos reconocibles y otros no tanto. Contiene dos constructores, uno sin argumentos y otro con todas las propiedades. Por otra parte, **es imprescindible que los nombres de las propiedades sean exactamente igual a los de las columnas del CSV** que están en el encabezado. Esto es vital para que el mapeo con el fichero se automático.



Por otra parte, se puede ver que algunas de las propiedades contienen lo que se llaman atributos personalizados, los cuales se utilizan para añadir funcionalidades extra al código. Por ejemplo, [Key] sirve para indicar que Id es la clave principal en el fichero mientras que Format determina el formato de fecha a utilizar en el fichero.

Antes de afrontar los siguientes pasos, es importante disponer de una clase que aglutine algunas de las variables globales del proyecto. Como en anteriores ocasiones, un buen nombre para ella puede ser LogicaNegocio. Su código podría ser el siguiente:

```
using CsvHelper.Configuration;
using System.Globalization;
using System.Text;

namespace VacasWPF
{
    public static class LogicaNegocio
    {
        private static string nombreArchivo = "vacas.csv";
        private static string ruta = AppDomain.CurrentDomain.BaseDirectory;
        public static string RutaArchivo =
            System.IO.Path.Combine(ruta, "Data", nombreArchivo);
        /// <summary>
        /// Configuración del CSV
        /// </summary>
        public static CsvConfiguration CsvConfig =
            new CsvConfiguration(CultureInfo.CurrentCulture)
            { Delimiter = ";", Encoding = Encoding.UTF8 };
    }
}
```

En primer lugar, la clase se define como estática, lo cual quiere decir que no necesita ser instanciada (y, de hecho, no se puede) ya que va a contener solo variables propias de la clase las cuales también van a ser estática. Esta es una de las diferencias de C# con Java; .NET permite definir no solo variables y/o métodos estáticos, sino también clases, mientras que en Java solo pueden utilizarse variables y/o métodos estáticos.

En esta lógica de negocio se almacenan datos como el nombre del archivo, la ruta del directorio base y se conforma la ruta completa del archivo. Además, se ajusta la configuración global para el manejo de archivos CSV indicando que se usará la cultura actual y el punto y coma como delimitador, cerrando con la codificación en UTF-8.

Para persistir la información se utilizará, como ya se ha comentado, el ORM desarrollado por Microsoft conocido como Entity Framework. Este ORM se basa en la existencia de lo que se conoce como contexto de base de datos representado por la clase [DbContext](#), aunque en este caso se generará un contexto personalizado para el modelo de datos de la aplicación cuyo nombre será AppVacasDbContext.

Un ejemplo de esta clase se muestra a continuación:

```
using CsvHelper;
using Microsoft.EntityFrameworkCore;
using System.Collections.ObjectModel;
using System.IO;

namespace VacasWPF.Models
{
    public class AppVacasDbContext : DbContext
    {
        public ObservableCollection<Vaca> Vacas { get; set; }

        public AppVacasDbContext() {
            Vacas = new ObservableCollection<Vaca>();
        }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            // Usa InMemory como el proveedor de base de datos
            optionsBuilder.UseInMemoryDatabase("InMemoryDatabase");
        }

        public override int SaveChanges()
        {
            int retValue = base.SaveChanges();
            saveToCsv();
            return retValue;
        }

        public override int SaveChanges(bool acceptAllChangesOnSuccess)
        {
            int retValue = base.SaveChanges(acceptAllChangesOnSuccess);
            saveToCsv();
            return retValue;
        }

        private void saveToCsv()
        {
            using (var writer = new StreamWriter(LogicaNegocio.RutaArchivo))
            using (var csvWriter = new CsvWriter(writer, LogicaNegocio.CsvConfig))
            {
                csvWriter.WriteRecords(this.Vacas.ToList());
            }
        }
    }
}
```

En este código hay un buen puñado de aspectos interesantes. En primer lugar, como es de rigor, se debe heredar de DbContext para aprovechar todas las funcionalidades que incorpora

EntityFramework de base. En esta clase, se incluye una colección observable de objetos de tipo vaca que es la que, a priori, conformará la colección de registros que podrán usarse sobre este contexto. Al crear un nuevo contexto, se inicializa esta colección sin elementos.

A continuación, se sobrescribe el método `OnConfiguring`, el cual va a determinar qué modelo de Entity Framework se utilizará; en este caso, se realizarán las operaciones en memoria, aunque esto no es del todo cierto.

Los contextos de Entity Framework tienen un método `SaveChanges()` cuya única misión es guardar los cambios en el modelo subyacente, en este caso, la memoria. Pero, dado que en este caso se pretende que se persista también en ficheros CSV, se han sobrescrito los métodos de la clase base añadiendo una llamada a `saveToCsv()` para registrar también los cambios en los ficheros CSV. Interesante es ver en este método cómo se produce esta persistencia: se crea un `StreamWriter` que luego se pasará como parámetro al `CsvWriter` que trasladará la lista completa de vacas al fichero.

Con estos pasos ya se está en condiciones de poder visualizar la colección de vacas en un control, por ejemplo, en un `DataGrid`. Supóngase que la ventana de visualización se llama `MainWindow` y que contiene un elemento de este tipo con una codificación similar a la siguiente:

```
<DataGrid AutoGenerateColumns="False" IsReadOnly="False" Name="dgVacas"
ItemsSource="{Binding Vacas}" Margin="0,0,0,74"
CellEditEnding="dgVacas_CellEditEnding" >
  <DataGrid.Columns>
    <DataGridTextColumn Header="Id" Binding="{Binding id}" />
    <DataGridTextColumn Header="Municipio" Binding="{Binding nomMunicipio}" />
    <DataGridTextColumn Header="FechaNac" Binding="{Binding f_nacim,
Converter={StaticResource FechaConverter}, Mode=TwoWay}"/>
    <DataGridTextColumn Header="FechaDestete" Binding="{Binding f_destete,
Converter={StaticResource FechaConverter}, Mode=TwoWay}"/>
    <DataGridTextColumn Header="Alzada" Binding="{Binding alzada}"/>
    <DataGridTextColumn Header="Peso" Binding="{Binding peso}"/>
    <DataGridTextColumn Header="Sexo" Binding="{Binding sexo}"/>
    <DataGridTextColumn Header="Tipo" Binding="{Binding tipo}"/>
  </DataGrid.Columns>
</DataGrid>
```

En este código hay varias partes que requieren una explicación, la cual se expondrá a medida que avance el ejemplo.

En primer lugar, tal como se ha hecho en otras ocasiones, hay que ajustar cuál será el `DataContext` sobre el que trabajará la ventana. Para ello, se debe modificar el constructor de la ventana XAML y ajustar dicho `DataContext` además de llamar a un método `fillDataGrid()` que poblará el `DataGrid` con los datos de las vacas contenidas en el contexto. Este código se introduce en el `CodeBehind` de la ventana.

Por ejemplo, si se llama MainWindow.xaml, su CodeBehind será MainWindow.xaml.cs. Hay que asegurarse de que se tienen los siguientes namespaces importados:

```
using CsvHelper;  
using System.IO;  
using System.Windows;  
using System.Windows.Controls;  
using VacasWPF.Models;
```

En primer lugar, en este fichero se declararán una serie de variables que se necesitarán para su funcionamiento. Por una parte, una lista de vacas que se poblará con los datos recogidos del CSV y un objeto de tipo AppVacasDBContext con el contexto recién creado:

```
private List<Vaca> lstVacas { get; set; }  
private AppVacasDBContext context = new AppVacasDBContext();
```

A continuación, se muestra el código del constructor en el que se asocia el context como DataContext y se llena el DataGrid con el contenido del fichero CSV. Eso se hace de forma indirecta, ya que se llena primero la lista de vacas y luego se asocian al DataContext que sirve de origen al Grid.

```
public MainWindow()  
{  
    InitializeComponent();  
    this.DataContext = context;  
    fillDataGrid();  
}  
private void fillDataGrid()  
{  
    using (var reader = new StreamReader(LogicaNegocio.RutaArchivo))  
    using (var csv = new CsvReader(reader, LogicaNegocio.CsvConfig))  
    {  
        lstVacas = new List<Vaca>(csv.GetRecords<Vaca>().ToList());  
        lstVacas.ForEach(v => context.Vacas.Add(v));  
    }  
}
```

Dado que dicho DataContext ya contiene un DBSet de Vacas, si se observa el código del DataGrid se puede encontrar la línea **ItemsSource="{Binding Vacas}"** que enlaza dicha colección con el control. Pero el enlace con el control no cesa ahí, sino que se propaga en cada columna del DataGrid que está enlazada a una propiedad del objeto Vaca y, por ende, a uno de los campos del CSV como se ve en el ejemplo.

```
<DataGridTextColumn Header="Alzada" Binding="{Binding alzada}"/>
```

En la parte correspondiente a las columnas destacan las relativas a fechas, las cuales siguen esta sintaxis:

```
<DataGridTextColumn Header="FechaNac" Binding="{Binding f_nacim,  
Converter={StaticResource FechaConverter}, Mode=TwoWay}"/>
```

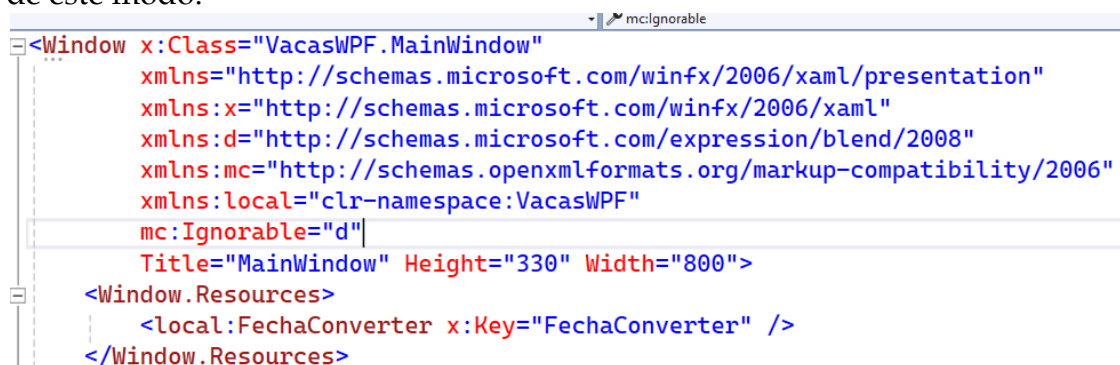
En este caso, es preciso hacer transformaciones ya que, aunque se usa un tipo `DateTime`, la visualización de fechas es en formato MM-DD-AAAA. La idea es hacer que el `DataGrid` sea editable, con lo que interesa que visualice en este formato, pero también que grabe los datos con él. Para solventar esta incidencia, hay que utilizar un artefacto llamado `FechaConverter` que se va a incluir como una nueva clase:

```
public class FechaConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
CultureInfo culture)
    {
        if (value is DateTime fecha)
            return fecha.ToString("dd/MM/yyyy");
        return value;
    }

    public object ConvertBack(object value, Type targetType, object parameter,
CultureInfo culture)
    {
        if (DateTime.TryParseExact(value.ToString(), "dd/MM/yyyy",
CultureInfo.InvariantCulture, DateTimeStyles.None, out DateTime result))
            return result;
        return value;
    }
}
```

Este tipo de convertidor implementa el interfaz [IValueConverter](#) que contiene dos métodos, `Convert` y `ConvertBack`. Se trata de implementar un método que convierta a cadena con el formato correspondiente (`Convert`) y otro que convierta de cadena con ese formato a fecha correcta (`ConvertBack`).

Para que la ventana reconozca este convertidor, hay que agregarlo como recurso local, lo que se haría justo debajo del código de `Window`, quedando la parte superior del XAML más o menos de este modo:



```
<Window x:Class="VacasWPF.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:VacasWPF"
    mc:Ignorable="d"
    Title="MainWindow" Height="330" Width="800">
    <Window.Resources>
        <local:FechaConverter x:Key="FechaConverter" />
    </Window.Resources>
```

28. Inclusión de recurso local `FechaConverter`

Con el código introducido hasta ahora, ya se podría visualizar la información de las vacas en el `DataGrid` con el formato dado.

En la siguiente imagen se puede ver cómo queda el listado de vacas del fichero.

Id	Municipio	FechaNac	FechaDestete	Alzada	Peso	Sexo	Tipo	
1	Navia	08/04/2004	12/08/2004	94	175	H	C	
2	Grado	09/06/2007	28/11/2007	104	253	H	C	
3	Somiedo	10/03/2004	07/07/2004	0	190	H	C	
4	Lena	02/06/2009	16/12/2009	106	218	H	N	
5	Salas	12/06/2006	02/01/2007	106	262	M	C	
6	Llamas de la Ribera	12/09/2005	27/01/2006	102	213	H	AC	
7	Belmonte de Miranda	28/03/2006	16/08/2006	0	160	H	C	
8	Cangas del Narcea	14/02/2012	27/06/2012	100	161	H	C	
9	Regueras (Las)	30/03/2010	22/10/2010	100	199	H	C	
10	Salas	12/06/2009	24/02/2010	108	315	M	C	

29. DataGrid con las vacas del CSV

En este ejemplo se va a dar un paso más allá, más que nada, para probar las posibilidades que ofrece Entity Framework. Por ejemplo, se pueden añadir dos botones: uno que permita agregar una nueva vaca a la colección y otro que sirva para borrar la que esté seleccionada en la cuadrícula. Por otra parte, ya que el DataGrid es editable, se va a permitir que se modifiquen los datos de una vaca con solo cambiarlos y validar la información directamente desde el control.

En primer lugar, es preciso añadir dos botones que se van a nombrar como btnNueva y btnEliminar. En cada uno de ellos se programará un manejador Click de forma que en su código XAML se haga referencia a cada evento, tal como puede verse en la imagen a continuación.

```
Click="btnNueva_Click"/>  
' Click="btnEliminar_Click"/>
```

30. Eventos de añadido y borrado de vacas

Añadir dichos manejadores es tan sencillo como escribir el código XAML de Click y dejar que el autocompletado haga el resto. También se puede recurrir a la ventana de propiedades donde aparece un grupo identificado como un rayo el cual recoge todos los eventos que soporta ese control.

Primeramente, se verá el código de añadido de una nueva vaca. Para reducir la complejidad del ejemplo, se creará una vaca programáticamente.

```
private void btnNueva_Click(object sender, RoutedEventArgs e)  
{  
    int newKey = context.Vacas.Max(v => v.id) + 1;  
    Vaca newVaca = new Vaca(newKey, "Oviedo", new DateOnly(2008, 4, 2),  
        new DateOnly(2008, 9, 21), 123, 87, "H", "C");  
    context.Vacas.Add(newVaca);  
    context.SaveChanges();  
}
```

El código resulta interesante por varias cuestiones. En primer lugar, obtiene el id de la vaca nueva extrayendo el máximo de los identificadores ya existentes (mediante una expresión Lambda) y sumándole 1. Al igual que en Java, el uso de expresiones Lambda facilita muchas tareas en .NET y especialmente si se trabaja con datos. A partir de ahí, crea una vaca con ese id y una serie de datos y la añade al contexto. Por último, llama al método SaveChanges del contexto para que quede actualizado. Anteriormente se sobrescribió este método para que reflejara tanto cambios en memoria como en el CSV. Para comprobar que se ha persistido también en el archivo CSV, hay que acudir al fichero que se encuentra en el directorio donde está el ejecutable compilado (/bin/Debug/netxxx/). Es importante observar que el fichero contenido en el proyecto de desarrollo nunca se modifica, solo se lee, y que los cambios realizados van todos a parar al directorio donde se compila la aplicación.

En cuanto al borrado, el código es el siguiente:

```
private void btnEliminar_Click(object sender, RoutedEventArgs e)
{
    context.Vacas.Remove(dgVacas.SelectedItem as Vaca);
    context.SaveChanges();
}
```

Como puede verse, muy sencillo, se obtiene el elemento seleccionado en el grid, se le convierte al tipo Vaca y se le elimina de la colección del contexto. Como esta está mapeada al grid, el cambio es inmediato y en el SaveChanges dicho cambio se persiste en el fichero también.

```
private void dgVacas_CellEditEnding(object sender, DataGridCellEditEndingEventArgs e)
{
    if (e.EditAction == DataGridEditAction.Commit)
    {
        Vaca vacaGrid = e.Row.Item as Vaca;
        if (vacaGrid != null)
        {
            Dispatcher.BeginInvoke(new Action(() =>
            {
                context.SaveChanges();
            })), System.Windows.Threading.DispatcherPriority.Background);
        }
    }
}
```

El hecho de tener que utilizar este artefacto con un Dispatcher se debe a que la actualización en el DataGrid no se hace automáticamente, sino de forma diferida. Mediante este código, se puede hacer de forma directa en el sistema.

Y con esto se podría concluir este ejemplo que ha ilustrado cómo se puede enlazar con un DataContext de EntityFramework y archivos CSV. Por supuesto, se pueden crear sistemas más complejos con más de un fichero CSV dentro de un mismo contexto y es ahí donde radica la



potencia de EntityFramework ya que mediante expresiones LinQ y Lambda, podrían realizarse consultas que enlazaran los registros de más de un fichero.

#### 5.2.7.4 El patrón Model View ViewModel

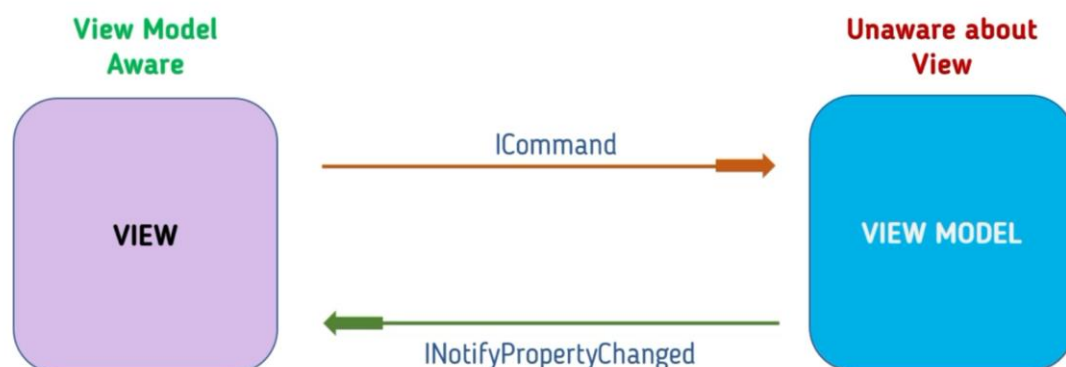
A continuación, se va a ver cómo crear una aplicación de navegación por páginas usando el patrón MVVM (Model View View-Model). Esta arquitectura ya se mencionó en la unidad 1 y se basa en la existencia de los elementos clásicos en este tipo de arquitectura como son la vista y el modelo, aunque añade un elemento más que es lo que se conoce como ViewModel y que contendrá la lógica de negocio. Básicamente, la idea es que nada del código esté contenido en el archivo C# del codebehind que tiene cada fichero XAML.

Pero ¿cómo funciona este patrón? Entre otras cosas, el ViewModel no sabe nada de la vista a la que se asocia, lo que además permite que luego se puedan pasar las pruebas unitarias sobre él. Aunque esto parece poco operativo, tiene sentido cuando se piensa en la vista, ya que esta sí que conoce al ViewModel porque es el elemento del que va a obtener su información. Dado que en WPF hay un elemento importante en la vista llamado DataContext, este será el que enlace directamente con el ViewModel.

Un elemento importante en esta arquitectura es el interfaz [ICommand](#), el cual permite la comunicación de la vista con el ViewModel. Por tanto, este comando en sí mismo será un objeto o una clase dentro del ViewModel.

Aunque se ha mencionado que el ViewModel no sabe nada de la vista, este elemento utiliza una interfaz, [INotifyPropertyChanged](#) para informar a la vista de que ha cambiado una propiedad, algo ideal para crear enlaces de datos con ella.

En definitiva, y tal como puede verse en la siguiente figura, básicamente se necesitan cuatro elementos para implementar un MVVM: Vista, ViewModel, ICommand e INotifyPropertyChanged.



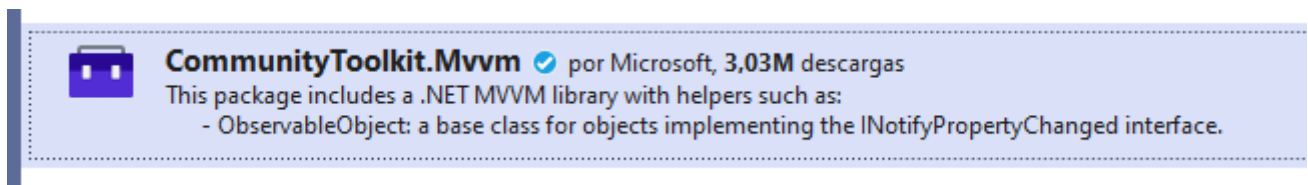
31. Arquitectura MVVM en WPF



Existen infinidad de implementaciones de MVVM para .NET y WPF. Como ejemplo, se podrían enumerar las siguientes:

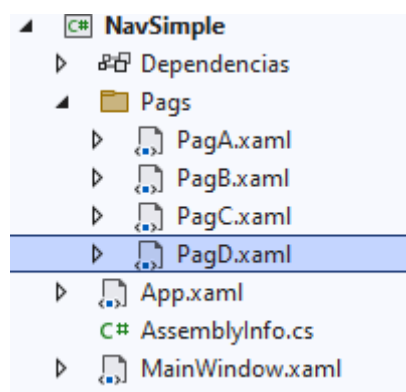
- [Haley.MVVM](#). Implementación simple de MVVM que cubre los requisitos básicos de la arquitectura tales como comandos delegados, colecciones notificables, servicio de diálogo, inyección de dependencias, etc.
- [CommunityToolkit.MVVM](#). Biblioteca para implementación de MVVM propia de Microsoft.

Para ilustrar el uso de esta arquitectura, se mostrará un ejemplo de proyecto MVVM en el que se creará un sistema de navegación entre apartados de una aplicación. En primer lugar, se crea una solución WPF con el nombre NavSimple. En el apartado de dependencias, se accede a Administrar paquetes NuGet y se busca el etiquetado como CommunityToolkit.Mvvm.



32. Community Toolkit MVVM en NuGet

En este caso, para ver el funcionamiento de MVVM se añadirán cuatro páginas XAML, que son ideales para realizar aplicaciones de navegación al estilo de un navegador. Para organizar mejor esta información en el proyecto, se puede crear una carpeta llamada **Pags** y dentro ubicar las cuatro UserControls etiquetados como PagA, PagB, PagC y PagD. De momento, la estructura de la solución quedaría como en la figura.



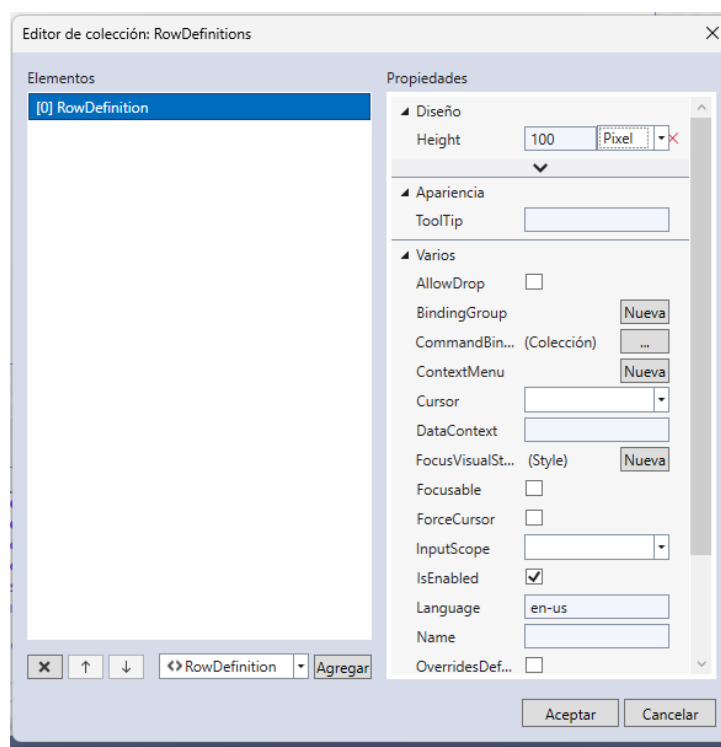
33. Estructura de solución de navegación simple

Para distinguir bien cada página, se puede cambiar su color de fondo e incluir un TextBlock en el centro que indique en qué página se está. En la siguiente figura puede verse una muestra de una de las cuatro páginas.



34. Ejemplo de página para navegación

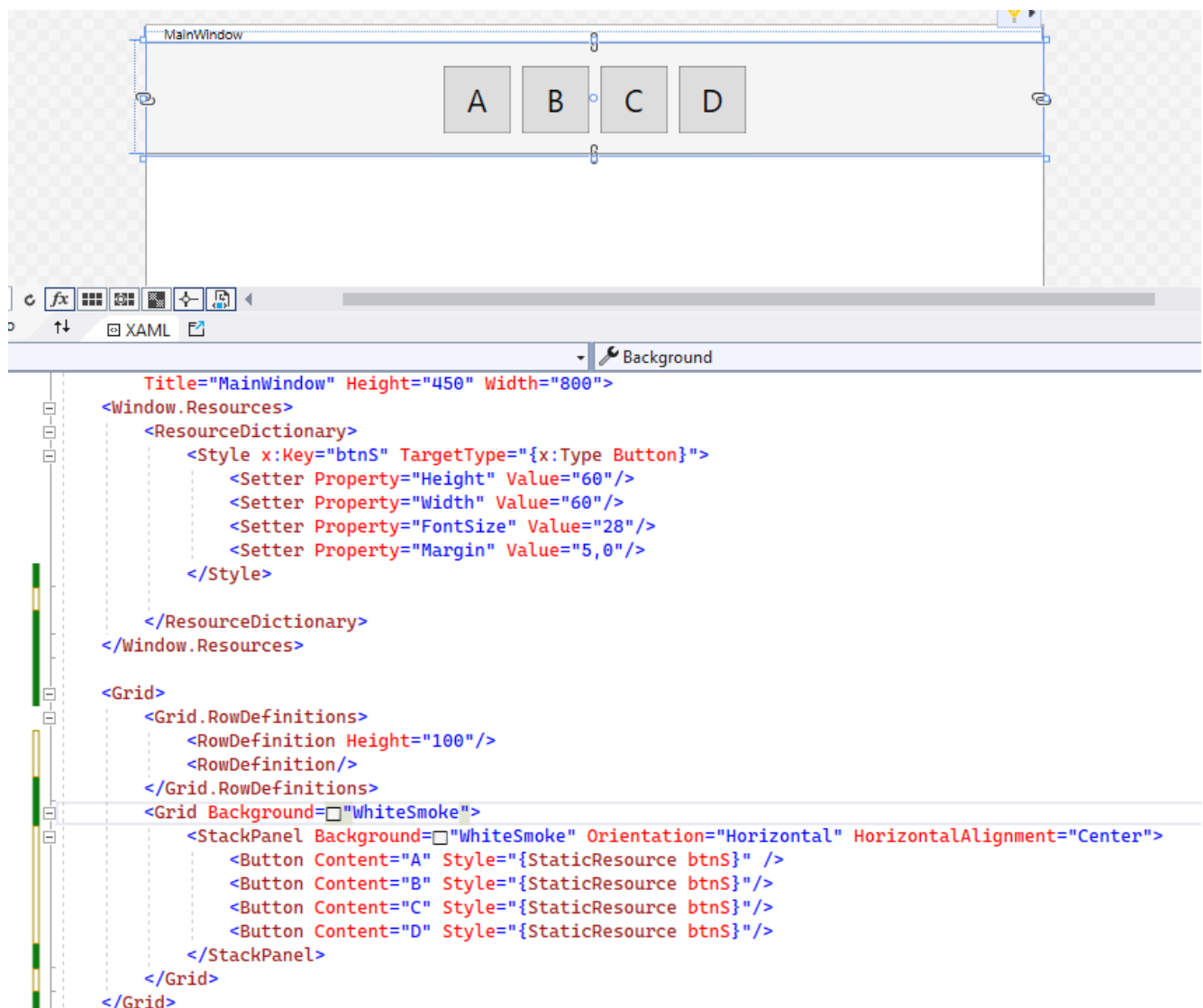
Bien, la idea es crear una barra de navegación en la parte superior mientras que, en el resto del formulario aparezca cada una de las páginas cuando se haga clic sobre el botón de acceso correspondiente. Todo ello se hará en el MainWindow.xaml. En primer lugar, se accede al Grid y dentro de sus propiedades, en Diseño, se entra en la opción **RowDefinitions**. Esto sirve para determinar cómo va a ser esa primera fila en la que se introducirán los botones de navegación. Como se ve en la figura, se puede tener una colección de definiciones de fila, con lo que pulsando al “+” se va a añadir una nueva en la que simplemente se cambiará la altura de la fila a 100 pixels.



35. Definición de fila de Grid

Esto sirve para indicar que los elementos sucesivos que se inserten en este tendrán una altura de 100 píxeles con lo que se procede a insertar uno nuevo (anidado al principal) para incluir los cuatro botones de navegación. Para ello, se va a utilizar un StackPanel con orientación Horizontal, el cual va a apilar los botones sin necesidad de estar preocupándose por su ubicación.

Dado que los cuatro botones van a tener el mismo tamaño, aspecto y margen entre ellos, resulta muy interesante introducir el uso de estilos en WPF. Para ello, en la propia ventana XAML (elemento Window) se dispone de un apartado llamado Resources donde se alojarán los recursos necesarios, entre ellos, los estilos que se vayan a usar en el formulario. Se muestra cómo quedará en la siguiente figura procediendo a explicar el código.



36. Configuración de panel de navegación con estilos

Como puede verse, dentro del Grid se introduce el StackPanel con una orientación y alineación horizontal determinados. Dentro de la ventana, se añade un elemento Window.Resources en el que se incluirá un diccionario de recursos (ResourceDictionary) que contendrá un nuevo

estilo llamado btnS que se aplica a todos los elementos de tipo Button. En el estilo se dan los valores de las propiedades que se quieren cambiar del control, en este caso altura, anchura, tamaño de fuente y margen. Por último, se introducen en el panel cuatro botones (A, B, C y D) con ese estilo. Como se ve en el código, dicho estilo se toma de un recurso estático llamado btnS. StaticResource se usa para eso precisamente, para obtener algo concreto que se pueda parametrizar como recurso y enlazarlo directamente al componente.

A continuación, se debe crear el ViewModel que dará soporte a esta barra de navegación. Esta clase implementará la clase abstracta ObservableObject la cual a su vez implementa INotifyPropertyChanged. Para ello, primero hay que importar el espacio de nombres ComponentModel:

```
using CommunityToolkit.Mvvm.ComponentModel;
```

La clase, tendrá como mínimo el siguiente código:

```
public class MainViewModel : ObservableObject
{
    public MainViewModel()
    {
    }
}
```

Para poder identificar de forma única las páginas y no tener que estar pasando parámetros en forma de cadena, se puede crear una enumeración que contenga las cuatro requeridas (A, B, C y D):

```
public enum PageId
{
    A, B, C, D
}
```

Se incluye, por tanto, una propiedad en el ViewModel que recoja dicha página haciéndolo de esta forma:

```
private PageId pageID;
public PageId PageID
{
    get { return pageID; }
    set { SetProperty(ref pageID, value); }
}
```

Como puede verse, para el setter utiliza [SetProperty](#) con el fin de poder notificar el cambio de la propiedad a la vista o donde se requiera. Este método comprueba si el valor de la propiedad ha cambiado y en caso afirmativo lanza un evento [PropertyChanging](#)

En el constructor del ViewModel se seleccionará la página por defecto, que en este caso es la A:

```
public MainViewModel()  
{  
    PageID = PageId.A;  
}
```

Ahora se creará un método para el cambio de página en cual será necesario para poder notificar dicha modificación tal como se verá posteriormente:

```
private void changePage(PageId newPage) { this.PageID = newPage; }
```

Se necesita un comando a través del cual la vista le indicará al ViewModel cómo actualizarlo. Este comando debe implementar la interfaz [ICommand](#). El código a introducir es el siguiente:

```
public ICommand CMDChangePage => new RelayCommand<PageId>(changePage);
```

Aquí lo que se hace es crear lo que se denomina un Relay Command o Comando de Retransmisión. Dado que el comando en sí requerirá un PageId al que cambiar en su llamada, se crea el RelayCommand como método genérico sobre el tipo PageId recogiendo como argumento el método de cambio de página. La forma de construir esta propiedad se apoya en una expresión Lambda.

Bien, lo que se necesita ahora es que la vista sea capaz de reconocer al ViewModel. Hay muchas formas de hacer esto, pero la más simple es asociarlo en el propio constructor de la vista. El elemento DataContext es clave a la hora de enlazar datos con controles en WPF por lo que es ideal para realizar la asociación con el ViewModel.

```
public MainWindow()  
{  
    InitializeComponent();  
    this.DataContext = new MainViewModel();  
}
```

Asociar cada botón al comando recién creado y darle el parámetro requerido para cada página es una operación muy sencilla que se debe hacer en la vista y se basa en modificar el código de cada botón para incluir estos datos:

```
<Button Content="A" Style="{StaticResource btnS}" Command="{Binding CMDChangePage}"  
CommandParameter="{x:Static local:PageId.A}"/>  
<Button Content="B" Style="{StaticResource btnS}" Command="{Binding CMDChangePage}"  
CommandParameter="{x:Static local:PageId.B}"/>  
<Button Content="C" Style="{StaticResource btnS}" Command="{Binding CMDChangePage}"  
CommandParameter="{x:Static local:PageId.C}"/>  
<Button Content="D" Style="{StaticResource btnS}" Command="{Binding CMDChangePage}"  
CommandParameter="{x:Static local:PageId.D}"/>
```

De este modo, se le dice a la vista que el comando que debe lanzar es CMDChangePage y que el parámetro que debe lanzar en la llamada al comando es la página correspondiente a cada

botón. Es interesante ver cómo asocia el botón mediante Binding (es preciso recordar que el DataContext está vinculado al ViewModel) y la representación del parámetro en forma de recurso local estático. Para probar que el comando se ejecuta correctamente, se podría poner un punto de interrupción dentro del método changePage y comprobar que se llega a él con la pulsación de cada botón.

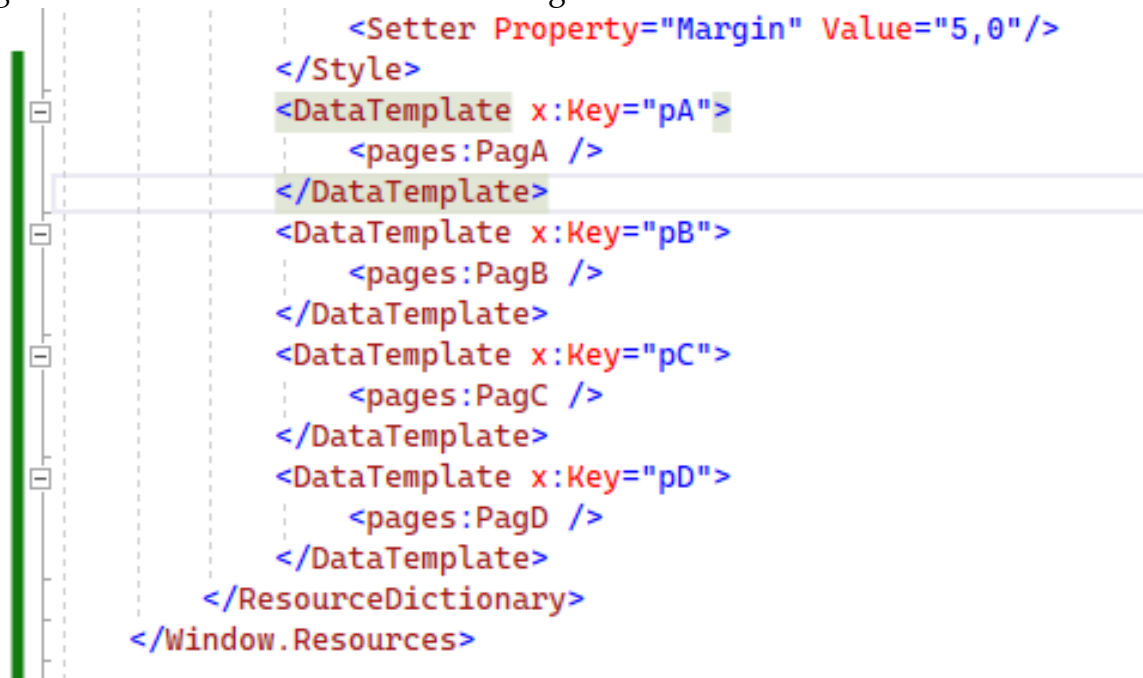
El siguiente paso requiere otra vez modificaciones en la vista. La primera, la importación del espacio de nombres XML que contiene las páginas (en el elemento Window como atributo):

```
xmlns:pages="clr-namespace:NavSimple.Pags"
```

Una vez importado, se etiqueta cada una de las páginas una vez incrustadas como recursos. Se añadiría el siguiente código dentro del <ResourceDictionary> detrás de <Style>:

```
<DataTemplate x:Key="pA">
    <pages:PagA />
</DataTemplate>
<DataTemplate x:Key="pB">
    <pages:PagB />
</DataTemplate>
<DataTemplate x:Key="pC">
    <pages:PagC />
</DataTemplate>
<DataTemplate x:Key="pD">
    <pages:PagD />
</DataTemplate>
```

El código debe verse de forma similar a la imagen:



```
<Setter Property="Margin" Value="5,0"/>
</Style>
<DataTemplate x:Key="pA">
    <pages:PagA />
</DataTemplate>
<DataTemplate x:Key="pB">
    <pages:PagB />
</DataTemplate>
<DataTemplate x:Key="pC">
    <pages:PagC />
</DataTemplate>
<DataTemplate x:Key="pD">
    <pages:PagD />
</DataTemplate>
</ResourceDictionary>
</Window.Resources>
```

37. Páginas como recursos

Ahora solo resta crear un ContentControl que contenga las páginas y que reciba la orden de la botonera superior seleccionando cada página según su pulsación.

Se adjunta el código a incluir (después del Grid que contiene el StackPanel) y se explica su funcionamiento.

```
<ContentControl Grid.Row="1" Content="{Binding }">
  <ContentControl.Style>
    <Style TargetType="{x:Type ContentControl}">
      <Style.Triggers>
        <DataTrigger Binding="{Binding
PageID,NotifyOnSourceUpdated=True,UpdateSourceTrigger=PropertyChanged}"
Value="{x:Static local:PageId.A}">
          <Setter Property="ContentTemplate" Value="{StaticResource pA}"/>
        </DataTrigger>
        <DataTrigger Binding="{Binding
PageID,NotifyOnSourceUpdated=True,UpdateSourceTrigger=PropertyChanged}"
Value="{x:Static local:PageId.B}">
          <Setter Property="ContentTemplate" Value="{StaticResource pB}"/>
        </DataTrigger>
        <DataTrigger Binding="{Binding
PageID,NotifyOnSourceUpdated=True,UpdateSourceTrigger=PropertyChanged}"
Value="{x:Static local:PageId.C}">
          <Setter Property="ContentTemplate" Value="{StaticResource pC}"/>
        </DataTrigger>
        <DataTrigger Binding="{Binding
PageID,NotifyOnSourceUpdated=True,UpdateSourceTrigger=PropertyChanged}"
Value="{x:Static local:PageId.D}">
          <Setter Property="ContentTemplate" Value="{StaticResource pD}"/>
        </DataTrigger>
      </Style.Triggers>
    </Style>
  </ContentControl.Style>
</ContentControl>
```

El ContentControl es un control para alojar contenido, en este caso cada página en forma de control de usuario. Se ubica en la fila 1 del Grid, de ahí su atributo Grid.Row. El cambio de páginas se debe hacer usando lo que se conoce como un [DataTrigger](#), elemento que representa un trigger que modifica una propiedad o ejecuta una acción cuando un dato enlazado alcanza una condición concreta. El DataTrigger se asocia a PageId y además añade que se notifique cuando se ha modificado el origen (NotifyOnSourceUpdated). El desencadenante de la actualización del origen es el cambio de la citada propiedad. En el Setter, se indica la propiedad a modificar (ContentTemplate) con el valor, que en este caso es el recurso estático correspondiente a cada página (pA, pB, etc.).

Si ahora se prueba la aplicación, se verá cómo se modifica la página consultada según el botón pulsado, lo que sirve como panel de navegación.



A continuación, se va a tratar de resumir el proceso realizado con los siguientes pasos:

1. Se ha creado una vista principal y una serie de páginas diferentes en forma de controles de usuario. Estas páginas no tienen ni idea de la página principal ni tampoco de su ViewModel.
2. Se crea un ViewModel y se agrega un objeto observable propio del framework MVVM. Principalmente, se necesitan dos interfaces, uno de notificación de cambio de propiedad y otro el propio comando, que es el mecanismo con el que la vista le envía un mensaje al ViewModel notificando los cambios de propiedad. Este proceso es rotativo y cada vez que la vista envía un mensaje al ViewModel se llama al RelayCommand el cual ejecuta el método de página.
3. Según la información de la vista, se cambia la propiedad en ella y a su vez llama a otra que notifica que se ha cambiado la propiedad volviendo a informar a la vista del cambio de propiedad (una especie de feedback).
4. Para el envío del mensaje se usa una enumeración la cual facilita el etiquetado de las páginas.
5. En base a todo el proceso anterior, se cambia la plantilla del ContentControl asociando el ViewModel con la vista. No hay que olvidar que se ha utilizado como DataContext de la vista el propio ViewModel.

### 5.2.8. DEPURACIÓN

Las causas de errores lógicos pueden ser particularmente delicadas de controlar. Visual Studio proporciona, por este motivo, una serie de herramientas que permiten controlar el estado de la ejecución de la aplicación, situar puntos de interrupción o visualizar los datos manipulados por la aplicación en tiempo de ejecución.

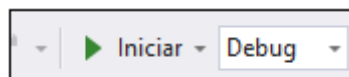
#### 5.2.8.1 Control de la ejecución

Para poder explorar el código de cara a buscar la causa de un error de ejecución o un error lógico es importante poder controlar la ejecución de una aplicación en condiciones que permitan realizar esta exploración. Visual Studio integra esta posibilidad mediante el uso de un depurador. En Visual Studio, un proyecto se puede encontrar en tres estados distintos:

- ❖ En modo de diseño: el desarrollo está en curso.
- ❖ En ejecución: el código se compila y la aplicación se ejecuta y se asocia al depurador integrado.
- ❖ En pausa: el depurador ha detenido la ejecución de la aplicación entre dos instrucciones.

#### a. Arranque

La ejecución de la aplicación en modo de depuración (Debug en Visual Studio) se realiza haciendo clic en el botón Iniciar de la barra de herramientas:

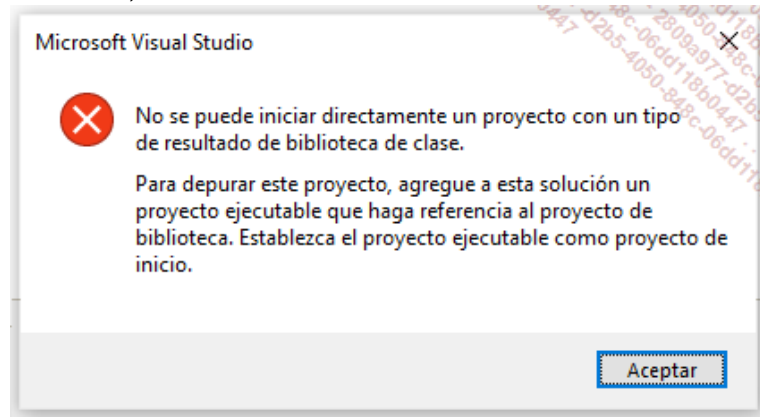


38. Botón Iniciar en Visual Studio



También es posible ejecutar la depuración pulsando la tecla F5 del teclado. La combinación de las teclas Ctrl + F5 ejecuta la aplicación en modo normal sin asociarla al depurador de Visual Studio. El proyecto sigue en modo de diseño, pero es imposible compilarlo o ejecutarlo mientras no se cierre la aplicación.

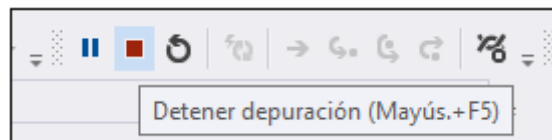
Cuando la solución contiene varios proyectos, el proyecto generado y ejecutado es el que se ha definido como proyecto de inicio. Es necesario, también, que este proyecto genere un archivo ejecutable. En caso contrario, se muestra un error:



39. Error en proyectos no ejecutables

#### b. Detención

Cuando el proyecto está en modo de ejecución o en pausa es imposible detener completamente su ejecución para pasar de nuevo al modo de diseño. Esta acción se realiza haciendo clic sobre el botón de detención que aparece en la barra de herramientas en modo de ejecución o en pausa.

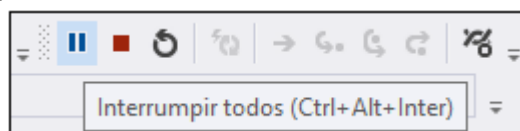


40. Botón de detección

La información que aparece sobre este botón indica que la combinación de teclas Shift + F5 detiene también la ejecución de la aplicación en modo de depuración.

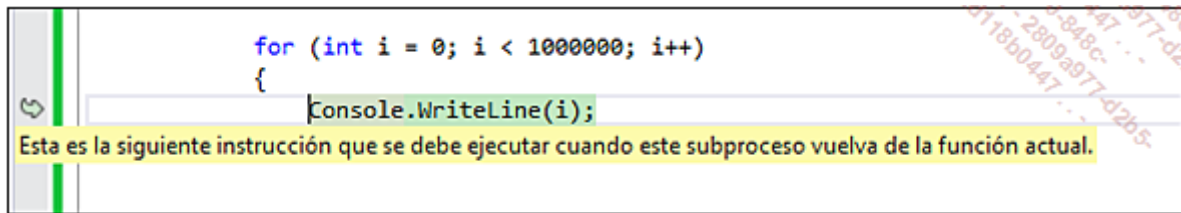
#### c. Pausa

Visual Studio permite poner en pausa la aplicación que está en curso de depuración para permitir inspeccionar el código ejecutado. Esta acción está disponible mediante el botón Pausa en la barra de herramientas.



41. Botón de pausa

Una vez se pone en pausa la aplicación, se sitúa una marca sobre la siguiente línea que se ejecutará.



42. Siguiente instrucción a ejecutar

Este método es relativamente azaroso puesto que hace falta mucha suerte para detener la aplicación en la línea precisa. Como se verá más adelante, el uso de puntos de interrupción es mucho más eficaz para explorar el código en ejecución.

#### d. Reanudar

Una vez interrumpida la ejecución de la aplicación, existen varias opciones disponibles. En primer lugar, es posible retomar el curso normal de la ejecución haciendo clic sobre el botón Continuar o presionando la tecla F5. Es posible, también, seguir en modo paso a paso para seguir los pasos de la ejecución.

Existen tres modos de ejecución paso a paso:

- ❖ El paso a paso principal (F10)
- ❖ El paso a paso detallado (F11)
- ❖ El paso a paso que sale (Shift + F11).

El paso a paso principal y el paso a paso detallado son casi idénticos. La diferencia entre ambos modos consiste en la manera de gestionar las llamadas a procedimientos o funciones. Cuando la aplicación se detiene sobre una línea que contiene una ejecución de un método, el paso a paso detallado va a permitir entrar en esta llamada y explorar su código. El modo de paso a paso principal ejecutará esta función en un único bloque.

El paso a paso que sale permite ejecutar un método en curso de un bloque y retoma el proyecto en modo pausa en la línea siguiente a la llamada al método.

Una última solución consiste en ejecutar el código en curso hasta la posición definida por el cursor de edición. Para ello, se hace clic en la línea y, a continuación, se utiliza la combinación de teclas Ctrl + F10. Esta solución puede resultar muy práctica para salir de la ejecución de un bucle, por ejemplo.

#### **5.2.8.2 Puntos de interrupción**













Los puntos de interrupción son herramientas indispensables para depurar una aplicación en C#. Ofrecen la posibilidad al desarrollador de definir la ubicación de una o varias interrupciones en la ejecución del código, lo que permite trabajar sobre escenarios de

depuración complejos, que requieren una serie de manipulaciones, pero sin tener que ejecutar la totalidad del código en modo paso a paso.

Los puntos de interrupción pueden ser condicionales, es decir, es posible especificar una condición que debe cumplirse para que el depurador interrumpa la ejecución.

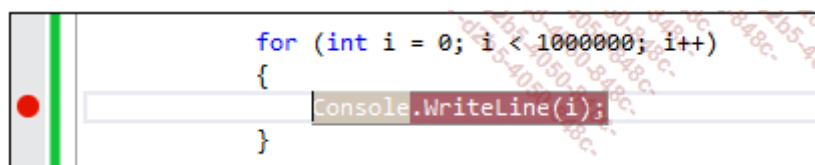
Los TracePoints son muy parecidos a los puntos de interrupción, pero permiten seleccionar una acción a realizar cuando se alcanzan. Esta acción puede ser el paso al modo en pausa y/o mostrar un mensaje que permita trazar la ejecución.

Los puntos de interrupción y TracePoints se representan en Visual Studio mediante iconos que indican su naturaleza y su estado. La siguiente tabla agrupa los distintos iconos. Para cada uno de los cuatro primeros elementos, el icono se divide en dos versiones: la primera se corresponde con el estado activo y la segunda con el estado deshabilitado.

		Representa un punto de interrupción normal.
		Representa un punto de interrupción condicional.
		Representa un TracePoint normal.
		Representa un TracePoint avanzado (con una condición, un contador de pasos o un filtro).
		Punto de interrupción o TracePoint deshabilitado a causa de un error en una condición.
		Punto de interrupción o TracePoint deshabilitado a causa de un problema temporal.

### Uso de un punto de interrupción

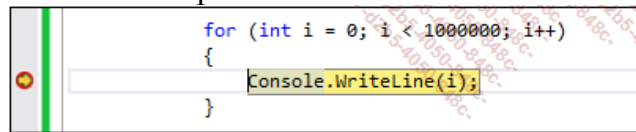
Para situar un punto de interrupción, basta con situar el cursor de edición sobre la línea de código correspondiente y presionar la tecla F9. La línea correspondiente se subraya en rojo y se muestra el icono correspondiente al punto de interrupción en el margen de la ventana de edición de código. Es posible, también, hacer clic directamente en el propio margen para definir un punto de interrupción.



43. Punto de interrupción

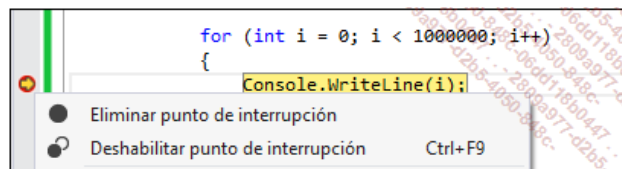
Cuando en la depuración se alcanza un punto de interrupción, Visual Studio pasa a primer plano y la sección de código que contiene el punto de interrupción se muestra en la ventana de edición de código. El color de subrayado de la línea pasa de rojo a amarillo. En este instante,

la instrucción todavía no se ha ejecutado y es posible utilizar el modo paso a paso o el paso a paso detallado para visualizar el comportamiento de esta instrucción.



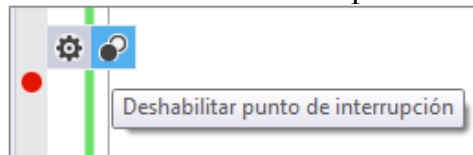
44. Punto de interrupción en ejecución

Para deshabilitar el punto de interrupción, se hace clic con el botón derecho sobre el icono correspondiente al punto de interrupción para abrir el menú contextual y seleccione, a continuación, Deshabilitar punto de interrupción, o bien utilizar la combinación de teclas Ctrl + F9 cuando el cursor de edición esté situado sobre una línea sobre la que se haya definido un punto de interrupción.



45. Opciones sobre punto de interrupción

También es posible deshabilitarlo utilizando el menú rápido que se muestra cuando se pasa el cursor del ratón por encima del icono que representa el punto de interrupción. El segundo botón de este menú permite habilitar o deshabilitar el punto de interrupción.

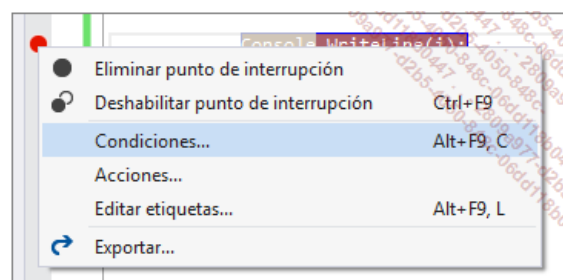


46. Deshabilitado de punto de interrupción

Los puntos de interrupción se eliminan cuando se hace clic en el icono que tienen asociado, o bien mediante la opción Eliminar punto de interrupción del menú contextual.

## Puntos de interrupción condicionales

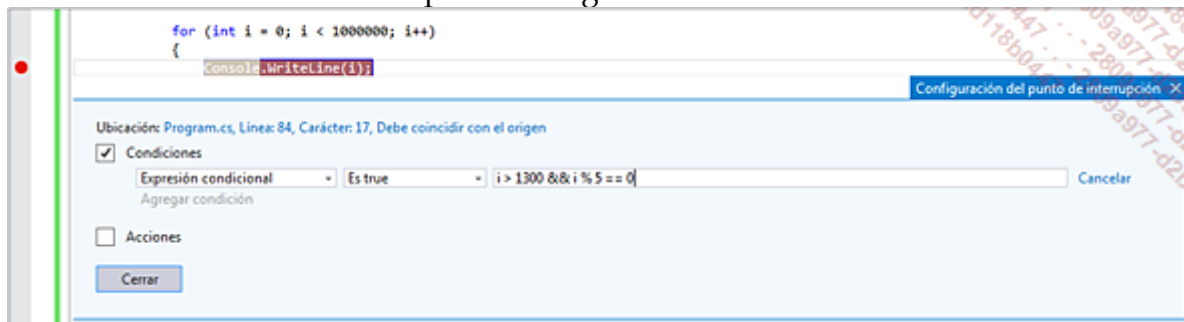
Es posible definir condiciones sobre los puntos de interrupción de manera que solamente se interrumpa el flujo de ejecución en aquellos casos particulares en los que, por ejemplo, un contador de bucle alcanza un valor particular. Para agregar una condición a un punto de interrupción, se despliega el menú contextual haciendo clic sobre el icono y se selecciona **Condiciones**.



47. Punto de interrupción condicional

Se abre a continuación la ventana Configuración del punto de interrupción, integrada en el editor que permite modificar las condiciones. Es posible introducir una o varias expresiones en C# evaluadas con cada ejecución de la línea de código. Estas se definen mediante tres elementos: un tipo, un modo de evaluación y una expresión.

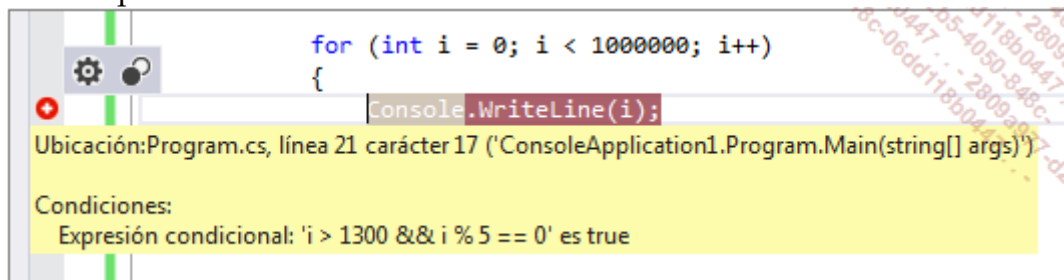
Por defecto, la expresión esperada es de tipo Expresión condicional: se trata de una expresión en C# que devuelve un valor booleano, como una comparación. El modo de evaluación inicial es, por defecto, Es true. Esta configuración indica que la ejecución se interrumpe cuando el resultado de la evaluación de la expresión valga true.



48. Expresión condicional en depuración

Aquí, la condición indica que el código debe interrumpirse únicamente cuando la variable *i* tenga un valor superior a 1300 y este valor sea múltiplo de 5.

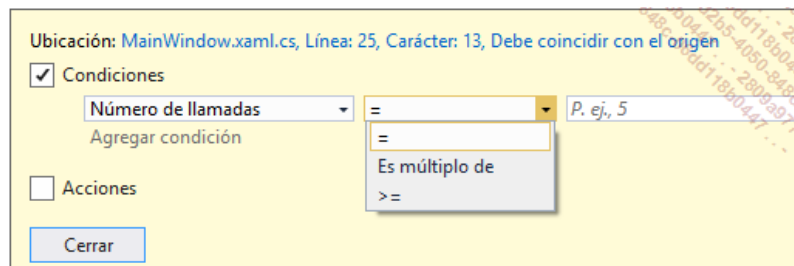
El icono que representa el punto de interrupción cambia de aspecto y la información correspondiente presenta la condición definida.



49. Punto de interrupción condicional en ejecución

El cambio del modo de evaluación por el valor cuando cambie provoca una modificación en el comportamiento del punto de interrupción: la ejecución se interrumpe únicamente cuando el valor de retorno de la evaluación de una condición sea diferente a la evaluación anterior. En el caso descrito anteriormente, la evaluación devuelve false hasta que el valor de *i* vale 1304. Cuando pasa de 1305, la evaluación de la condición devuelve true: el código se interrumpe. Pero en la siguiente iteración, *i* vale 1306, de modo que la expresión vale false: el punto de interrupción no se produce.

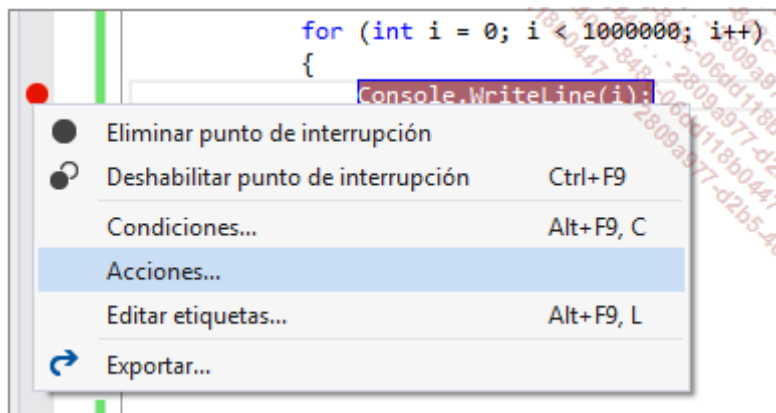
Las condiciones pueden, también, evaluar expresiones relativas al número de accesos realizados a un punto de interrupción. Este se activa solamente cuando se alcanza un cierto número de veces.



50. Depuración después de un número de llamadas

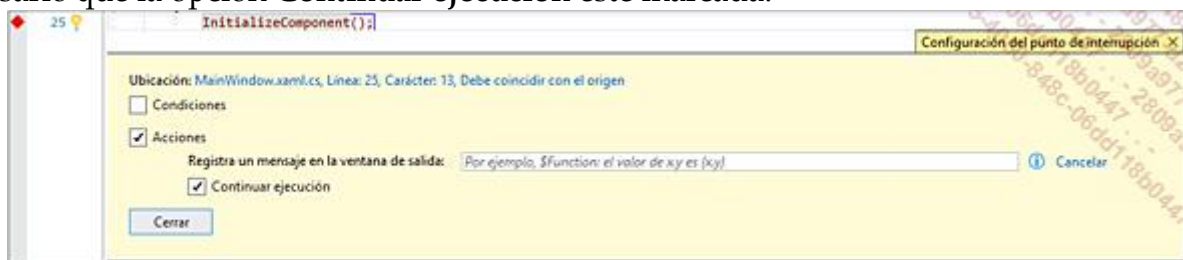
## TracePoints

Los TracePoints pueden crearse mediante la opción Acciones del menú contextual de un punto de interrupción.



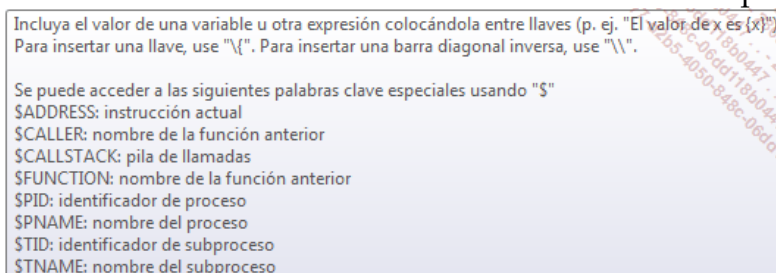
51. Tracepoint

El uso de esta opción abre también la ventana **Configuración del punto de interrupción**. Esta configuración permite definir mediante una macro un mensaje que se mostrará en la ventana de salida de Visual Studio. Para que el punto de interrupción sea realmente un TracePoint es necesario que la opción **Continuar ejecución** esté marcada.



52. Configuración del punto de interrupción

Es posible visualizar las distintas palabras clave que pueden utilizarse en las macros pasando el ratón por encima del icono Información situado a la derecha del campo editable.



53. Palabras clave en depuración

Si se aplican varias condiciones a un TracePoint, este se activa únicamente si se cumplen dichas condiciones. Conservando la condición creada más arriba, el inicio del resultado en la ventana de salida es el siguiente:

```
Function: ConsoleApplication2.Program.Main(string[]), El valor  
de i es 1305  
Function: ConsoleApplication2.Program.Main(string[]), El valor  
de i es 1310  
Function: ConsoleApplication2.Program.Main(string[]), El valor  
de i es 1315  
Function: ConsoleApplication2.Program.Main(string[]), El valor  
de i es 1320
```

#### 54. Salida de Tracepoint

### 5.2.9. GENERACIÓN DE CÓDIGO PARA DISTINTAS PLATAFORMAS

Uno de los retos más importantes con los que se enfrentan los desarrolladores es el intercambio de datos entre sistemas incompatibles. Si se realiza con XML se reduce en gran medida la dificultad ante la que enfrentarse en cuanto al intercambio de datos entre sistemas incompatibles.

Gracias a su portabilidad, XML se ha convertido en una de las tecnologías más utilizadas como base para el almacenaje de contenidos, como modelo de representación de metadatos y como medio de intercambio de contenidos. Esto se debe a que es un lenguaje independiente de plataforma, lo que implica que cualquier software desarrollado para XML puede leer y procesar estos datos independientemente del hardware o sistema operativo. La tecnología XML se basa en tres pilares fundamentales que permiten generar código para diferentes plataformas:

1. Representación de metadatos: lo más importante para esta tarea es el sistema de indexación y recuperación que permite discriminar dentro de un contenido tanto elementos como atributos a recuperar.
2. Medio de intercambio de contenidos: la integración que permite XML en distintas plataformas se basa en la facilidad de intercambio de contenidos dado que se puede utilizar para múltiples fines como integración en una base de datos, visualización en un sitio web o mensajes entre sistemas de información.
3. Almacenamiento de contenidos: en los últimos años está creciendo la demanda de bases de datos nativas XML, las cuales almacenan y gestionan documentos en este formato sin ningún tipo de transformación previa.



## ÍNDICE DE FIGURAS

1. Arquitectura de XAML en Windows .....	9
2. Tipo de proyecto WPF en Visual Studio .....	10
3. Código XAML vs C# .....	11
4. Esquema básico de un documento XML.....	11
5. Namespaces que pueden aparecer en un fichero XAML .....	13
6. Paleta de componentes en WPF .....	14
7. Panel de propiedades de un control WPF .....	14
8. ListBox con elementos de DataContext.....	17
9. Interfaz con ComboBox .....	18
10. Ejemplo de enlace entre controles.....	20
11. Interfaz de modificación de registro.....	22
12. Diálogo de agregado de personas.....	24
13. Vista de diseño del ejemplo 1 de StackPanel.....	27
14. Vista de diseño del ejemplo 2 de StackPanel.....	27
15. Vista de diseño del ejemplo 1 de DockPanel.....	28
16. Vista de diseño del ejemplo de WrapPanel .....	29
17. Ejemplo de figuras con Canvas .....	29
18. Alineamiento de componentes.....	30
19. Prototipo de visor de imágenes.....	31
20. Disposición inicial de ImageViewer .....	32
21. Configuración de menú en ImageViewer.....	33
22. Configuración de botones e imagen .....	34
23. ImageViewer con miniaturas en la lista .....	37
24. Botones en ImageViewer .....	38
25. Estructura de vacas.csv.....	40
26. Paquetes para CSV y Entity Framework.....	41
27. Copiar siempre en fichero CSV .....	41
28. Inclusión de recurso local FechaConverter.....	48
29. DataGrid con las vacas del CSV .....	49
30. Eventos de añadido y borrado de vacas .....	49
31. Arquitectura MVVM en WPF.....	51
32. Community Toolkit MVVM en NuGet .....	52
33. Estructura de solución de navegación simple.....	52
34. Ejemplo de página para navegación.....	53
35. Definición de fila de Grid .....	53
36. Configuración de panel de navegación con estilos .....	54
37. Páginas como recursos.....	57
38. Botón Iniciar en Visual Studio .....	59
39. Error en proyectos no ejecutables .....	60
40. Botón de detección .....	60
41. Botón de pausa.....	60



42. Siguiendo instrucción a ejecutar .....	61
43. Punto de interrupción.....	62
44. Punto de interrupción en ejecución .....	63
45. Opciones sobre punto de interrupción.....	63
46. Deshabilitado de punto de interrupción.....	63
47. Punto de interrupción condicional .....	63
48. Expresión condicional en depuración .....	64
49. Punto de interrupción condicional en ejecución.....	64
50. Depuración después de un número de llamadas .....	65
51. Tracepoint.....	65
52. Configuración del punto de interrupción.....	65
53. Palabras clave en depuración .....	65
54. Salida de Tracepoint.....	66

## **BIBLIOGRAFÍA - WEBGRAFÍA**

Powers et al. (2008). *Microsoft Visual Studio Unleashed*. Editorial Pearson.

Randolph et al. (2010). *Professional Visual Studio 2020*. Editorial Wrox.

González P. (2016). *Desarrollo de interfaces. Elaboración de interfaces mediante documentos XML*.  
[https://www.youtube.com/watch?v=zv4eOM\\_dZ6s&list=PLIfP1vJ2qaknhfmG5Ce8O-paV001qfnUS](https://www.youtube.com/watch?v=zv4eOM_dZ6s&list=PLIfP1vJ2qaknhfmG5Ce8O-paV001qfnUS)

Putier, S. (2021). *C# 8 y Visual Studio 2019 - Los fundamentos del lenguaje*. Editorial Eni.

García Miguel, D. (2021). *Desarrollo de interfaces*. Editorial Síntesis

Amberscript (2023). *Las mejores herramientas de conversión de voz a texto 2023*  
<https://www.amberscript.com/es/blog/conversion-de-voz-a-texto-2023/>

Torresburriel Estudio (2023). *Zero UI: una nueva forma de interactuar con los dispositivos*  
<https://torresburriel.com/weblog/zero-ui-una-nueva-forma-de-interactuar-con-los-dispositivos/>

Gamco (2023) ¿Qué es Realidad Aumentada? <https://gamco.es/glosario/realidad-aumentada/>

Mouse Events (2023). *Learn to setup Page Navigation in WPF - MVVM*  
<https://www.udemy.com/course/learn-to-setup-page-navigation-in-wpf-mvvm/>