# Generalidades del lenguaje

El xaml de wpf (xaml) es el marco mas grande, de ahí se desprenden silverligth 4 y 5, aplicaciones metro de w8 y wp 7 y 8 y últimamente Windows Store. O sea que entre ellos puede existir cierta diferencia.

¡¡¡ CUIDADO, ESTOS APUNTES SON UNA MEZCLA DE WPF Y WP8 !!!

Se recomienda utilizar xaml para definir la apariencia, distribución y orden y el lenguaje de programación para determinar su comportamiento. Tambien de puede codificar la apariencia por medio de c# por ejemplo, pero mediante xaml es mejor porque es un lenguaje declarativo, se organiza mejor y se ven las jerarquías de objetos mas claramente; en general es mas intuitiva la forma de codificar.

Asi mismo, la inegracion con datos ideal debería ser mediante servicios web implementados con Entity Framework, o en su caso, bases de datos locales, ya sea con slq lite o cualquier otro.

**AL IGUAL QUE EN OTROS LENGUAJES, EN XAML SE IMPORTAN ESPACIOS DE NOMBRES MEDIANTE LA INSTRUCCIÓN XMLNS, LA CUAL VA COMPLEMENTADA CON RUTAS COMO “http:””. Asi mismo, a este espacio de nombres le podemos asignar un alias: XMLNS:MiAlias. UNO DE LOS ESPACIOS DE NOMBRES POR DEFAULT ES EL QUE USAMOS PARA EL *DISEÑO* Y TIENE EL ALIAS XMLNS:d “**[**http://...**](http://...)**” La mejor practica es ponerlo a nivel RAIZ (PhoneAplicationPage), aunque puede colocarse en cualquier otro lugar del código.**

SE PUEDE REFERENCIAR CUALQUIER ESPACIO DE NOMBRES QUE ESTE EN EL FRAMEWORK, POR EJEMPLO PARA REFERENCIAR UN VALOR DOUBLE. Primero hacemos la referencia al espacio de nombres, luego lo publicamos como recurso al tipo que le corresponde (Double) y finalmente lo instanciamos en alguna propiedad que utilice un double:

xmlns:miCore=”clr-namespace:System;Assembly=mscorelib”

<miCore:Double x:Key=”miValor”>300</core:Double>

<Button Width=”{StaticResource miValor}” …>

EN ESTE SENTIDO, TAMBIEN PODEMOS HACER INSTANCIAS DE NUESTROS PROPIOS TIPOS, O SEA CLASES. Por ejemplo de una clase persona que tiene una propiedad nombre y otra país, aqui sobrescribe tostring() para que devuelva estas propiedades concatenadas. Primero referenciamos el espacio de nombres de nuestro proyecto, luego lo publicamos como recurso poniéndole algún valor en las propiedades al estilo constructor, y finalmente utilizándolo en algún lugar donde sea prudente mostrarlas pg caja de texto.

xmlns:miEspacio=”clr-namespace:EspacioDeMiProyecto pg Proyecto1”

<miEspacio:Persona Nombre=”a” Pais=”b” x:key=”Persona1”>

<Button Content=”{StaticResource Persona1}”/> //notar que no requiere algo como Persona1.Tostring()

Cuando la propiedad margin de cualquier control esta definida con 2 o mas valores, quiere decir que cuenta con anlcaje al los lados del contenedor y si este cambia de tamaño también lo hara elcontrol, si sus valores son 0 y tiene alguna definición de alto y ancho, entonces sus dimensione son absolutas.

Siempre que se pueda se debe separar por bloque las propiedades compuestas de un control, para tener mejor legibilidad. Usar las propiedades como subelementos en vez de atributos, lo cual es muy importante entre mas compleja sea la composición de algún objeto.

Sintaxis de atributos:

<Image Margin="0,0,0,0" Name="image1" Source="spiderman.jpg" Stretch="Fill"/>

Sintaxis de subelementos:

<Image Margin="0,0,0,0" Name="image1">

<Image.Source>

<BitmapImage UriSource="spiderman.jpg" />

</Image.Source>

</Image>

Existen algunos objetos que soportan una tercera sintaxis, de contenido, por ejemplo el textblock:

<TextBlock> Hola </TextBlock>

Que seria lo mismo que la siguiente forma:

<TextBlock text=”Hola”></TextBlock>

### Xaml dinamico

Se refiere a que podemos hacer con código c# pg. Toda la declaración de objetos para que se generen las interfaces en tiempo de ejecución, el único inconveniente es que se pierde la interaccion en tiempo de diseño.

### Recursos

Los recursos “RESOURCES” son objetos que se pueden referenciar desde diversos lugares del código, accesibles mediante su propiedad key {StaticResource MiRecurso}. Los recursos son accesibles desde el objeto contendor, si queremos que lo sean para todos debemos ponerlo como recurso de la ventana. Esto es muy similar a la visibilidad y alcance de las variables en c#, y su utilidad es que al cambiar las propiedades del recurso podemos cambiar la de todos los objetos que le hacen referencia.

La máxima visibilidad seria en <Application.Resources>

Se pueden declarar en objetos que contengan la propiedad .Resources, o sea que permitan la declaración de diccionarios de recursos, por ejemplo en un Grid.Resources.

<Grid.Resources>

<SolidColorBrush x:key=”miBrocha” Color=”Red”/>

</Grid.Resources>

<TextBlock Text=”Hola” foreground=”{StaticResources miBrocha}”/>

LOS OBJETOS TOMARAN LOS RECURSOS QUE ESTEN EN EL AMBITO MAS INMEDIATO.

### Propiedades adjuntas

Es cuando en las propiedades de un objeto citamos propiedades del objeto padre para definir el comportamiento del primero. Por ejemplo para colocar una elipse utilizamos la propiedad de colocación del canvas que lo contiene:

<canvas>

<Elipse Canvas.Top=1 Canvas.Left=1>

</canvas>

Esto se planteo asi para no hacer demasiado pesado el lenguaje como w forms, en donde para todas las clases existen propiedades comunes, aunque en muchos casos no se utilicen.

En código se pueden referenciar estos valores con SetValue() y GetValue():

If((double)Elipse.GetValue(CanvasProperty)==0)

{

Elipse.SetValue(Canvas.LeftProperty,50);

}

### Comentarios

Los comentarios son al estilo <!-- código o comentario -->

**MUY UTIL PARA PROGRAMAR VOVIMIENTO DE LOS OBJETOS EN PANTALLA, VER VIDEO 1 DE LA SEMAMA 1 AL FINAL.**

### Dialogs

CAPTURAR UN CUADRO DE DIALOGO

MessageBoxResult dl = new MessageBoxResult();

dl=MessageBox.Show("Va a modificar el registro actual. ¿Desea continuar?", "Editar registro", MessageBoxButton.OKCancel,MessageBoxImage.Warning);

if (dl == MessageBoxResult.OK)

{

voiEditProducto();

voiLleListFamilia();

Capturar la tecla pulsada

private void Window\_KeyDown(object sender, KeyEventArgs e)

{

switch (e.Key)

{

case Key.Enter: this.Close();

break;

}

}

}

## Estructura de la sitaxis xaml

Apertura y cierre de la declaración de un objeto. Notar que el texto que se muestra al estilo **text** es el espacio de la propiedad content, la cual también se puede declarar explícitamente o puede contener un objeto.

<objeto1> Texto\_que\_se\_muestra\_u\_otro\_objeto\_anidado

</objeto1>

La declaración de los objetos se puede dividir en dos partes: el objeto en si y sus propiedades, y los objetos que contiene el objeto

<objeto1 propiedad1objeto1=”” propiedad2objeto1…>

<objeto2 propiedad1objeto2=”” propiedad2objeto2...>

</objeto2>

</objeto1>

Las declaraciones X:code indican que existe una porción de texto que tiene vinculación con codebehind de algún lenguaje .net

# Distribucion y orden

Como es de esperarse, cualquier contenido en la pagina o en el formulario debe estar organuizada por los diversos objetos que comprende wpf:

GENERALMENTE SE RECOMIENDA NO UTILIZAR DIMENSIONES NI COORDENADAS ABSOLUTAS, PUES LOS TIPOS DE RESOLUCIONES Y TAMAÑOS DE LAS PANTALLAS SEIEMPRE SERAN CAMBIANTES, POR ELLO LOS MEJORES CONTENEDORES SERAN EL GRID Y EL STACK, EL PRIMERO COMO EL MAS FLEXIBLE Y EL SEGUNDO COMO EL AUXILIAR.

Los controles derivan de ControlContent que solo puede contener un elemento, pero a su vez puede contener un elemento contenedor como un panel, con lo cual podemos hacer que un control contenga mas controles.Organizar por medio de panel es la forma correcta de contener controles.

<Button Name="miBoton" FontFamily="Consolas" Background ="Blue" Foreground="White" FontSize="10" Margin="213,12,12,248">

<DockPanel>

<Image Source="/images/setup.bmp"/>

<TextBlock FontSize="10">obj2</TextBlock>

</DockPanel>

</Button>

A la forma en que se distribuyen y ordenan los objetos en un panel se le conoce como layout.

La propiedad PanelX.ZIndex=”N” (X es el numero de panel en cuestión y N el numero de orden de aparicion) proporciona el orden de aparicion de los controles, un índice mayor estará por encima de los demás controles. Por ejemplo, si se quiere tener una imagen de fondo esta deberá tener el ZIndex=”0” y las demás podrían tenerlo en 1.

Panel1.ZIndex=”0”

EXISTEN 3 TIPOS DE DIMENSIONES PARA LOS CONTENEDORES: FIJO, AUTO (QUE AUTOMATICAMENTE OCUPAN TODO EL ESPACIO O TODO EL ESPACIO RESTANTE) Y EL PROPORCIONAL (\* QUE INDICA QUE SERA PG \*2 DOS VECES MAS GRANDE QUE LA ANTERIOR).

### Stackpanel

Usualmente no se utiliza como el contenedor principal, sino omo auxiliar para anidar controles. El orden de aparición es de arriba abajo o de derecha a izquierda según la propiedad orientation

<StackPanel Orientation="Vertical">

<Button Width="100" Height="50" Background ="AliceBlue" FontFamily="Arial"

FontSize="20" FontWeight="Bold">

Aceptar

</Button>

<Button Width="100" Height="50" Background ="AliceBlue" FontFamily="Arial"

FontSize="20" FontWeight="Bold">

Cancelar

</Button>

</StackPanel>

### Dockpanel

Aquí se puede indicar la posición de los objetos, derecha, izquierda, arriba o abajo; y el ultimo en agregarse ocupa todo el espacio disponible restante. La fuente es bendings porlo cual 9 y : muestran << y >>.

**<DockPanel LastChildFill="True" Background="AliceBlue">**

    <Button **DockPanel.Dock="Left"** Width="50" Height="50" Background ="CornflowerBlue" FontFamily="Webdings"

            FontSize="20" FontWeight="Bold" Foreground="White" Margin="10,0,0,0">

      9

    </Button>

    <Button **DockPanel.Dock="Right"** Width="50" Height="50" Background ="CornflowerBlue" FontFamily="Webdings"

            FontSize="20" FontWeight="Bold" Foreground="White" Margin="0,0,10,0">

      :

    </Button>

    <Image Source="Foto.jpg" Margin="10"></Image>

**</DockPanel>**

### Grid

El mas completo y flexible de todos, con este puedem realizarse todos los demás y es usualmente utilizado como contenedor principal. Se puede dividir en dos partes: la primera que es la configuración de filas y columnas y la segunda que es la colocación de objetos.

Notar como cada elemento agregado lleva el indicador de fila columna donde se ubica.

<Grid HorizontalAlignment="Center" VerticalAlignment="Center" Background="AliceBlue">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<TextBlock Grid.Column="0" Grid.Row="0" Grid.ColumnSpan="5"

Background ="Black" FontFamily="Arial" FontSize="30" FontWeight="Bold" Foreground="White" Margin="5">

0

</TextBlock>

<Button Grid.Column="0" Grid.Row="1" Height="50" Width="50"

Background ="CornflowerBlue" FontFamily="Arial" FontSize="20" FontWeight="Bold" Foreground="White">

7

</Button>

.

.

.

</Grid>

1.- \* INDICA QUE SE ESTAN UTILIZANDO SIMENSIONES RELATIVAS. ES DECIR \* SIGINFICA QUE SE ESTA DIVIDIENDO EN 2 PARTES PROPORCIONALES EL ESPACIO DISPONIBLE. 2\*, 3\*, ETC. SIGNIFICA QUE SERA EL DOBLE, TRIPLE DEL PRIMER VALOR.

2.- Si se quieren mostrar las líneas que determinan los bordes de  las filas y columnas de un Grid se debe dar valor "True" a la propiedad ShowGridLines

3.- Con la propiedad Grid.ColumnSpan, un elemento indicará sobre cuantas columnas se extenderá dentro del Grid que lo contiene.

4.- Igualmente con la propiedad Grid.RowSpan se puede indicar sobre cuántas filas se extenderá (en este caso el botón "+" se extiende entre dos filas).

### Wrappanel

Organiza los objetos según la propiedad orientation, si se agota el espacio hace un retorno de carro automáticamente, ya sea hacia abajo o hacia la derecha

<WrapPanel Orientation="Horizontal" Background="AliceBlue">

<Button Height="20" Width="50">1</Button>

<Button Height="20" Width="50">2</Button>

<Button Height="20" Width="50">3</Button>

<Button Height="20" Width="50">4</Button>

<Button Height="20" Width="50">5</Button>

<Button Height="20" Width="50">6</Button>

<Button Height="20" Width="50">7</Button>

<Button Height="20" Width="50">8</Button>

</WrapPanel>

### Canvas

Coordenadas absolutas, pero **no se recomienda en aplicaciones que cambiaran el tamaño de la ventana**. Idela por ejemplo para simular objetos sobrepuestos en otro, ejemplo un refrigerador con pegatinas.

<Canvas>

<Image Source="/images/Setup.bmp" Height="150" Width="200"/>

<TextBox Canvas.Top="10" Canvas.Left="10" Background="LightYellow" FontWeight="Bold"

Foreground="Blue" FontSize="20" FontFamily="Bradley Hand ITC" AcceptsReturn="True"/>

<TextBox Canvas.Top="100" Canvas.Left="25" Background="LightYellow" FontWeight="Bold"

Foreground="Blue" FontSize="10" FontFamily="Bradley Hand ITC" Height="35"

Width="35" AcceptsReturn="True" >mensaje de la pegatina

</TextBox>

</Canvas>

### ScrollViwer

ESTE ES UN ELEMENTO MUY UTIL PARA CUANDO LOS CONTROLES REBASEN LA CAPACIDAD DE LA PANTALLA, EL DETALLE ES QUE SOLO ADMITE UN ELEMENTO, QUE EN ESTE CASO PUEDE SER UN STACKPANEL.

### FlipView

Visto en App Win 8

Es un control como el que se usa para desplazar objetos en la web tales como fotos, contiene las fichas de desplazamiento en los extremosademas de animaciones de desplazamiento.

## Navegacion

Notar que no todos los objetos soportan un hyperlink en su content

<TextBlock>ir a :

<Hyperlink NavigateUri="page2.xaml">pagina 2</Hyperlink>

</TextBlock>

### Formularios

Cierre del formulario principal. Ojo, si ya se definió un método para el window\_closing, este se ejecutara con el this.close

This.Close();

El formulario con el que inicial la aplicación se configura en StartUpUri en el código de App.xaml

Ejecutar un procedimiento al arrancar la mostrar el formulario inicial, similar al load de un formulario

public MainWindow()

{

InitializeComponent();

this.Procedimiento();

}

Se puede definir alguna tarea para el evento closing, es decir, antes de cerrar definitivamente el formulario. Desde xaml definimos que va a ejecutar desde la propiedad closing de Window. Si seleccionamos el nuevo manejador de evento (Window\_Closing) se crea en code behind. En este evento el segundo parámetro e indica una propiedad cancel, si esta es true el formulario no se cerrara, false es su valor por defecto.

Se configura el evento para que al dar en cerrar el formulario aparezca un mensaje con Si No. messageresult.no indica que no se cierre el formulario en caso de dar un enter, para evitar perdida no intencionada.

<Window x:Class="… Closing="Window\_Closing">

private void Window\_Closing(object sender, System.ComponentModel.CancelEventArgs e)

{

MessageBoxResult k = MessageBox.Show("estas seguro de salir", "titulo", MessageBoxButton.YesNo, MessageBoxImage.Question, MessageBoxResult.No);

e.Cancel = (k == MessageBoxResult.No);

}

Llamar a un formulario existente

Window2 myW2 = new Window2();

myW2.ShowDialog();

### Procesos

Por defecto wfp es monoproceso, por lo cual una aplicación de larga duración puede hacer que el programa se pasme por un lapso. Pero el remedio es que las librerías permiten mandar un proceso de larga duración a un subproceso. Ver pp 639 libro o el ejemplo de wpf. Todo se ejecuta desde un proceso salvar que es el que instancia la clase o estructura member (1), que llama a un proceso privado que es el que realiza la escritura del archivo que tarda mucho (2), este proceso toma los datos de una clase o estructura que es como se evita el error de seguridad por usar Thread(3).

La clase Thread funciona bien para procesos sencillos, pero la clase BackgroundWorker ofrece solución para procesos mas complejos.

(1)

if (SD.ShowDialog().Value)

{

Member member = new Member();

member.FirstName = firstname.Text;

member.LastName = lastname.Text;

member.TowerName = towername.Text;

member.IsCaptain = iscaptain.IsChecked.Value;

member.MemberSince = membersince.SelectedDate.Value;

member.Methods = new List<string>();

foreach (CheckBox cb in methods.Items)

{

if (cb.IsChecked.Value)

{

member.Methods.Add(cb.Content.ToString());

}

}

Thread WThread = new Thread(() => this.SaveData(SD.FileName, member));

WThread.Start();

}

(2)

private void SaveData(string fname, Member member)

{

using (StreamWriter writer = new StreamWriter(fname))

{

writer.WriteLine("First Name: {0}", member.FirstName);

writer.WriteLine("Last Name: {0}", member.LastName);

writer.WriteLine("Tower: {0}", member.TowerName);

writer.WriteLine("Captain: {0}", member.IsCaptain.ToString());

writer.WriteLine("Member Since: {0}", member.MemberSince.ToString ());

writer.WriteLine("Methods: ");

foreach (string method in member.Methods )

{

writer.WriteLine(method);

}

}

Thread.Sleep(10000);

}

(3)

struct Member

{

public string FirstName { get; set; }

public string LastName { get; set; }

public string TowerName { get; set; }

public bool IsCaptain { get; set; }

public DateTime MemberSince { get; set; }

public List<string> Methods;

}

## Principales propiedades comunes de controles

### Equivalencia de sintaxis

Equivalencia entre las propiedades wpf y las de vb o c#. Esto es mientras se escribe en xaml, porque al referirnos a los mismos controles desde el código c# por ejemplo, se utilizara text en vez de content.

Name ->nombre

Content->text o caption

Aunque lo ideal seria agruparlo en el bloque <TextBox.Text>, ya que corresponde a text el vinculo binding, con lo cual se le daría mayor claridad al código

<TextBox.Text>

<Binding ElementName="numberOfTickets" Path="Value"/>

</TextBox.Text>

Ojo: si en este caso ponemos mode=”TwoWay” al cambiar el text de la caja y perder el foco el control silder también se actualizara. Es decir. No requiere necesariamente que se configure el slider para responder a un cambio en textbox

### Popups

EL EVENTO QUE CONTIENE EL MENSAJE DEBE SER ASINCRONO, POR LO QUE EL METODO SHOWASYNC DEBE SER PRECEDIDO POR AWAIT. ADEMES EL MESSAGEDIALOG DEBE USAR IMPORT PORQUE NO ES NATIVO. ESTE POPUP ES MUY INVASIVO, POR LO QUE SOLAMENTE SE ACONSEJA PARA MENSAJES CRITICOS DEL SISTEMA.

LA PROPIEDAD COMMAND.ADD TAMBIEN PERMITE RECIBIR PARAMETROS \*\*\*\*INVESTIGAR Y SE PUEDEN PONER LOS BOTONES QUE SE DESEE

private async void Button\_Click\_4(object sender, RoutedEventArgs e)

{

var msg = new MessageDialog("Mensaje","Título");

msg.Commands.Add(new UICommand("Aceptar"));

msg.Commands.Add(new UICommand("Cancelar"));

var result=await msg.ShowAsync();

switch (result.Label)

{

case "Aceptar":

//algo

break;

default:

break;

}

}

## Estilos

Se define el estilo como una propiedad compuesta Button.Resorces y para que tenga efecto en el botón se declara la propiedad Style={DinamicResources…}. Sin embargo este método es aplicable solamente al elemento que lo contiene, una opción será la de crear el estilo como recurso de la ventana, es decir, como propiedad compuesta pero del objeto <Window…>. Copiamos <Style … /Style> y ahora si será posible utilizar el estilo en distintos botones. Una ventaja adicional es que esta propiedad {Dinamic… se puede cambiar en tiempo de ejecución por ser dinámica.

EN COMMON-STANDARDSTYLES HAY MUCHOS ESTILOS PREDEFINIDOS QUE SE PUEDEN USAR, BASTA CON LLAMARLOS Style={StaticResources ESTILO}. . .

DE IGUAL FORMA, CUANDO SE AGREGA UNA PAGINA BASICA, ESTA PAGINA YA CUENTA CON ESTILOS Y COMPORTAMAIENTO PREDEFINIDO, SOLO HACE FALTA REFERENCIARLO, POR EJEMPLO AGREGA RECURSOS COMO BindableBase. CONTRARIAMENTE A LA PAGINA EN BLANCO.

<Window.Resources>

<Style x:Key="Estilo1" TargetType="Control">

<Setter Property="Background" Value="Gray"/>

<Setter Property="Foreground" Value="White"/>

<Setter Property="FontFamily" Value="comic Sans MS"/>

</Style>

<Style.Triggers>

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Background" Value="Blue"/>

</Trigger>

</Style.Triggers>

</Window.Resources>

Aplicando el estilo a los botones

<Button Style="{StaticResource Estilo1}" Content="Button". . .

Para aminorar la repetición podemos declarar TargetType=”Button” y borrar las declaraciones “Button. (…)Vemos que la declaración TargetTypo=”Button” restringe la aplicación del estilo a botones, si cambiamos a tipo Control podremos utilizar el estilo a muchos otros controles mas.

### Estilos en cascada (baseon)

Se pueden anidar estilos utilizando BaseOn=”{StaticResource St1}”

### Estilos implicitos

En vez de x:key utilizan TargetType, solo que asi se debe declarar explícitamente al tipo de objeto que aplicara, no uno genérico como Control, por ejemplo rectángulos, elipses, etc. Seria TargetType=”Elipse”

Existe una forma de crea diccionarios exportables a través de un archivo xaml, este solo se registra en el modulo Application.xaml, a través de <ResourceDiccionary.

Para poderlo utilizar en una aplicación w8 solo se agrega el archivo como vinculo (diccionario.xaml) y luego en app.xaml se agrega <ResourceDictionary Source=”MiDiccionario”/>

### Triggers

Los triggers son capacidades adicionales que ofrecen los estilos, determinan una acción que se realiza cuando cambia un valor de una propiedad, en este caso IsMouseOver, o cuando el raton se posiciona encima cambia el backgroun a azul. Tambien se pueden aplicar estilos en la ventana de propiedades, los estilos a nivel window aparecen en la ficha local.

Cuando se aplica un estilo a un control contenedor se aplica a los controles que están dentro

# Controles

### Configurar eventos desde xaml

DESDE XAML PODEMOS CONFIGURAR QUE PROCEDIMIENTO SE EJECUTA CON CLICK, PODRÍA SER EL QUE NOS DA POR DEFAULT O UNO NUEVO, LA ÚNICA CONDICIÓN ES SEA UN VOID CON PARÁMETROS OBJECT Y ROUTEEVENTARGS.

X:KEY SE UTILIZA NORMALMENTE PARA REFERENCIAR RECURSOS O DICCIONARIOS

X:NAME SE UTILIZA PARA NOMBRAR CONTROLES, POR EJEMPLO NOMBRAR UN CONTROL Y REFERENCIARLO EN CODEBEHIND

### AppBar

**FUNDAMENTALES**: YA SEA BUTTONAPPBAR O TOPAPPBAR, SON LAS BARRAS EN DONDE SE IMPLEMENTA LA FUNCIONALIDAD DE LAS APLICACIONES, AL HACER UN DESLIZ CON EL DEDO O UN CLICK DERECHO DEL MOUSE APARECEN Y PUEDEN CONTENER BOTONES Y FUNCIONALIDAD. OJO QUE VAN FUERA DEL GRID

</Grid>

<Page.BottomAppBar>

<AppBar>

<StackPanel>

<Button/>

<Button/>

</StackPanel>

</AppBar>

</Page.BottomAppBar>

<Page.TopAppBar>

<AppBar>

<StackPanel>

<Button/>

<Button/>

</StackPanel>

</AppBar>

</Page.TopAppBar>

</common:LayoutAwarePage>

### Textbox y RichTextBox

Vaciar una caja de texto, depende de lo que se quiera específicamente. Asi se puede pasarle una cadena vacia, limpiar la caja y pasarle una cadena con longitud o. un richttextbox permite mostrar texto conjuntamente con otros objetos.

TextBox1.text=String.Empty;

TextBox1.Clear();

TextBox1.text=””;

Desencadenador click

Anidar texto e imagen en un botón

<Button Width="200" Height="150" Background ="AliceBlue">

<StackPanel>

<TextBlock HorizontalAlignment="Center" FontSize="20" FontWeight="Bold" Foreground="Navy">

¿Yo, en un Botón?

</TextBlock>

<Image Source="/images/setup.bmp" Height="100" Margin="5"/>

</StackPanel>

</Button>

Una propiedad importante es la AcceptReturn para hacer que genere nueva línea

Un elemento adjunto e interesante es el PasswordBox

Poner mayúsculas automaticamente

CharacterCasing="Upper"

### Binding a controles de lista

Existen diferencias entre los ComboBox, ListBox, DataGrid…y los XXXView. Los primeros son tipo Legacy que existen por compatibilidad, útiles para tareas sencillas que no requieran mucha implementación. Los segundos más avanzados. Muchas de las propiedades no son compatibles entre unos y otros.

Se puede poblar de diversas formas, y estás aplican también a GridView, DataGrid, ListBox y CmboBox. Un ejemplo completo de Binding es el video 8 del curso de JuanK. Un ejemplo parcial en la página familia del sistema Beer.

**En general consideremos lo siguiente: Los binding mas avanzados tipo familias de la aplicación Beer se harán a traves de métodos automáticos con colecciones del ViewModel. Todo lo demás se hará directamente desde los métodos de la clase. Inclusive se podría considerar nunca utilizar una colección en VM, hasta que se logre implementar por completo las notificaciones como en el video de juank. Seguir utilizando por lo pronto DataTables.**

**Binding automatizado**: Utiliza el modelo MVVM

1. -Modelo: una clase tipada de acuerdo con la tabla correspondiente
2. -Modelo: una clase tipo colección (ObservableCollection) del tipo de la clase
3. -Vista modelo: una clase que implementara la operatividad entre vista y modelo. Heredará de BindableBase e implemenará por ejemplo **en el constructor el llenado inicial de la colección, y otros métodos que conformen colecciones para que se puedan utilizar como source de controles mediante su método ToList()**. Implementa también campos y propiedades del tipo ObservableCollection.

En el sistema Beer se utiliza en el constructor el llenado de la colección por medio del método de la clase base.

1. -Vista: en xaml es requerido el namespace al proyecto para poder referenciar las clases, por ejemplo (xmlns:fvm="clr-namespace:BeerWPF.ViewModel"). Declarar esa instancia como un recurso (<fvm:vmFamColeccion x:Key="FamColeccion"/>). Y declarar un DataContext para el objeto padre (<fvm:vmFamColeccion/>), en este caso en WindowDataContext.

En el código del control bindeado se hará referencia al recurso y a la ruta (ItemsSource="{Binding Source={StaticResource FamColeccion}, Path=FamiliaColeccion}").

**Binding semi manual con ítems tipados:** Podria ser una aproximación a la programación por capas, el control se poblará con elementos tipados de una clase base.

1. Utiliza la misma estructura de clases que la técnica anterior. Se puede valer de algún método del View Model que conforme colecciones para vaciarse en controles mediante ToList(). Esto para no utilizar directamente métodos de la clase base en donde se utilicen tablas o readers.
2. El vaciado de datos al control se hace: this.lisProdXFamilia.ItemsSource = vmProdCol.ProdXFamilia.ToList();

**Binding manual:** Se toman los objetos por separado, por ejemplo se llena una List por medio del recorrido de un DataReader o tabla. Hay que mencionar que de esta forma si responden normalmente los métodos SelectedItem, SelectedValue, etc.

En el ejemplo siguiente se llena directo del método de la clase que es de tipo datatable, mediante su método DefaultView.

claExistencia claEx = new claExistencia();

this.griProductos.ItemsSource = obtExisXIdProducto(Nom).DefaultView;

En el caso de un combo, una vez cargado el itemssource hay que definir en XAML el itemtemplate y el datatemplate

<ComboBox.ItemTemplate>

<DataTemplate>

<StackPanel Orientation="Horizontal">

<TextBlock Text="{Binding status}" Foreground="Black"></TextBlock>

<TextBlock Text="{Binding id\_status}" Foreground="White"></TextBlock>

</StackPanel>

</DataTemplate>

</ComboBox.ItemTemplate>

**Referenciar el elemento actual del grid, list o combo:**

1. Para binding automatico y semiautomático se debe establecer que la selección actual es de un tipo de dato, una vez hecho eso, ya se puede referenciar a la columna deseada:

var sel = lisProdXFamilia.SelectedItems[0] as claProducto;

voiLlenaCajas(Int16.Parse(sel.id\_producto.ToString()));

1. Para un Binding manual

Para un grid

DataRowView r =(DataRowView) griProductos.SelectedItem;

this.texId.Text = r.Row.ItemArray[0].ToString();

Para un combo o list

This.Text.Text=this.lisProdXFamilia.SelectedValue;

//de internet:

Si es un ListBox,  
  
string valor = ((System.Data.DataRowView)ListBox1.SelectedItem).Row.ItemArray[0].ToString();  
MessageBox.Show(valor);  
  
Si es un ComboBox  
  
string valor = ((System.Data.DataRowView)ComboBox1.SelectedItem).Row.ItemArray[0].ToString();  
MessageBox.Show(valor);  
  
(editado)  
Si por si caso te preguntas como recuperar la cadena que has seleccionado  
  
string valor = ComboBox1.SelectedItem.ToString();  
MessageBox.Show(valor);

**Referenciar elemento de una coleccin de VM**

//se referencia el ViewModel

vmExistenciaColeccion vmEC = new vmExistenciaColeccion();

//llamamos al metoo para que filtre por id de producto

vmEC.voiExisXIdProducto(idProd);

//mandamos el valor al control referenciando la propiedad tipo colección del vm

this.texId.Text = vmEC.ExistenciaColeccion[0].id\_existencia.ToString();

### Combox

Vaciar el combo, por ejemplo para un reset

Combo.Items.clear();

Poner algún elemento de la lista como el primero por defecto

Combo.tetxt=combo.Items[x] as string;

Llenar un combo con un amatriz

foreach (string t in matriz)

{

matriz.Items.Add(t);

}

Llenar un combo con xaml

<ComboBox Height="23" Name="comboBox1" VerticalAlignment="Top" Width="120">

<ComboBox.Items>

<ComboBoxItem>

opcion 1

</ComboBoxItem>

</ComboBox.Items>

</ComboBox>

O también

**<ComboBox Margin="10" SelectedIndex="0">**

<TextBlock FontFamily="Arial" FontWeight="Bold">

     Lección 3 Layout

  </TextBlock>

  <TextBlock FontFamily="Arial" FontWeight="Bold">

    Lección 4 Controles

</TextBlock>

**</ComboBox>**

Llenar combobox con un reader \*\*ojo porque cada ajemplo es de diferente versión. Aplica para ListBox

while (Reader.Read())

{

//primera forma vista en un VS 2010

//guarda en un string el registro del campo 1 del reader

string Nom = Reader.GetString(1);

ComboBox.Items.Add(Nom);

//segunda forma vista en un entorno de VS 2013

ComboBox.Items.Add(Reader("Nombre del campo"));

//o compuesta al parecer para ListBox

ComboBox.Items.Add("Este es el campo: " & Reader("Nombre del campo"));

}

### ListBox

ESTE CONTROL ES RUDIMENTARIO, SE COLOCA EL CONTENIDO SIN NINGUN ELEMENTO EXTRA DE DISEÑO O COMPORTAMIENTO

**<ListBox Margin="10" SelectedIndex="0">**

      <TextBlock FontFamily="Arial" FontWeight="Bold">

        Lección 3 Layout

      </TextBlock>

      <TextBlock FontFamily="Arial" FontWeight="Bold">

        Lección 4 Controles

      </TextBlock>

**</ListBox>**

### ListView

ESTE CONTROL ES CONTIENE EXTRAS DE DISEÑO, POR EJEMPLO AL CLICK SE COLOCA UN CHECK PEQUEÑO EN LA PARTE SUPERIOR DERECHA, LOS BORDES CAMBIAN, ES DECIR, ESTA ORIENTADO PLENAMENTE AL DESARROLLO TOUCH. OJO, TANTO ESTE COMO EL PRIMERO, EL TAMAÑO DE LOS ESPACIOS DE LISTA ESTA DADO POR EL TAMAÑO DE LOS PROPIOS OBJETOS ENLISTADOS, DE TAL FORMA QUE PODEMOS PONER CUALQUIER DIMENSION.

<ListView>

<Border>

<StackPanel Style="{StaticResource Stack\_H}">

<Image Source="/Assets/JESSICA.jpg"/>

<StackPanel Style="{StaticResource Stack\_V}">

<TextBlock>

<Bold FontSize="45">Jessica de Alba</Bold>

<LineBreak/>

<Italic FontSize="30">Actriz</Italic>

</TextBlock>

</StackPanel>

</StackPanel>

</Border>

</ListView>

### GridView

COMO SU NOMBRE LO INDICA ES UN CONTROL TIPO CUADRICULA, POR EJEMPLO GENERA AUTOMATICAMENTE NUEVAS COLUMNAS PARA CONTENIDOS LARGOS, SU SCROLL ES HORIZONTAL. EL TAMAÑO DE LOS ESPACIOS DEL GRID ESTA DADO POR EL PRIMER ELEMENTO, DE TAL FORMA QUE SE PUEDEN CORTAR ELEMENTOS MAS GRANDES

### Image

PARA CARGAR UNA IMAGEN AL CONTROL IMAGE SE LE DEBE PASAR UNA CADENA AL SOURCE, PERO NO UNA CADENA NORMAL, SINO UNA DE TIPO Uri.

### Check

Personalizar sus elementos al tipo check: Primero crea un objeto tipo checkbox con inicio nulo, Luego recorre los elementos de una matriz y los carga en el list y al objeto method le pasamos a content el texto que tenga el elemento de la matiz.

CheckBox method = null;

foreach (string m in matriz)

{

method = new CheckBox();

method.Margin = new Thickness(0, 0, 0, 10);

method.Content = m;

lista.Items.Add(method);

}

Mandar una lista a un campo de una clase o estructura. Primero se crea la clase o estructura y su campo tipo list (1), luego se crea en algún evento o procedimiento la descarga al campo, en este ejemplo manda valores de unos check, pero podría ser de otra lista (2), por ultimo para descargar desde la clase o estructura en un list o similar (3)

(1)

public List<string> Methods;

(2)

Clase member = new Clase();

member.Methods = new List<string>();

foreach (CheckBox cb in algunCheckBox.Items)

{

if (cb.IsChecked.Value)

{

member.Methods.Add(cb.Content.ToString());

}

}

(3)

foreach (string met in clase.Methods )

{

…implementacion;

}

Configurar el contenido a modo de grid (esta configuración se basa en un ejemplo de entity framework donde loscampos están vinculados a una clase entidad)

<ListView Height="243" HorizontalAlignment="Left" Margin="12,39,0,0" Name="saveChanges" VerticalAlignment="Top" Width="568" IsSynchronizedWithCurrentItem="True" ItemsSource="{Binding}">

<ListView.View>

<GridView>

<GridView.Columns>

<GridViewColumn Width="75" Header="Product ID" DisplayMemberBinding="{Binding Path=ProductId}"/>

<GridViewColumn Width="225" Header="Name" DisplayMemberBinding="{Binding Path=ProductName}"/>

<GridViewColumn Width="135" Header="Quantity per unit" DisplayMemberBinding="{Binding Path=QuantityPerUnit}"/>

<GridViewColumn Width="75" Header="Unit price" DisplayMemberBinding="{Binding Path=UnitPrice}"/>

</GridView.Columns>

</GridView>

</ListView.View>

</ListView>

### Label

No tiene por que incluir solo texto, también puede desencadenar un comando a través de su propiedad AccessText y con un binding mandar a cierto control enlazado. En el ejemplo, cuando se le sal alt+n pone el cursor en el control enlazado.

**<Label Name="Etiqueta1"**

**Target="{Binding ElementName=Texto1}"**

**VerticalAlignment="Center">**

**<AccessText>\_Nombre:</AccessText>**

**</Label>**

### TabControl

Útil para organizar contenido por pestañas

<TabControl Margin="10">

<TabItem>

<TabItem.Header>

<TextBlock FontFamily="Arial" FontWeight="Bold">

Lección 4 Controles

</TextBlock>

</TabItem.Header>

<StackPanel>

<Image Height="200" Source="/images/setup.bmp"></Image>

<Button>

b1

</Button>

<TextBox>

este es el texto

</TextBox>

</StackPanel>

</TabItem>

</TabControl>

### Check y radiobutton

poner el check en no seleccionado

check.IsChecked=False;

Desencadenador checked

Ámbito: se modifica su propiedad groupname

Mandar el valor checked del check solo en caso de estar seleccionado. EVITA poner un if para evaluar si esta seleccionado o no

iscaptain.IsChecked.ToString()

### DatePicker

poner el picker en una fecha actual. También se puede utilizar Now en vez de Today, ver opciones distintas y resultados

Picker.Text=DateTime.Today.Tostring();

EN LA BASE EL TIPO DE DATOS DATETIME O DATE

-SE TOMA LA FECHA CON EL DatePicker.Text Y DE AHI SE MANDA COMO CUALQUIER OTRA CADENA DE TEXTO PERO CUIDADO CON EL FORMATO QUE DEBE SER YYYY-MM-DD PARA SQL SERVER

SIGUIENDO CON SQL SERVER Y LOS CAMPOS DATETIME, ESTA ES LA FORMA EN QUE SE GUARDA EN LA BASE. EN CASE DE QUERER GUARDAR LA FECHA/HORA ACTUAL

DECLARACION UNIVERSAL PARA MUCHAS DBA SIN GIONES NI DIAGONALES:

DateTime d = new DateTime();

d = DateTime.Now;

string format = "yyyyMMdd HH:mm:ss";

ASIGNACION EN LA CADENA DEL INSERT ENTRE CIMILLAS SIMPLES “ ‘ “:

d.ToString(format)

EVITAR QUE SE GUARDE 01/01/1900 CUANDO HAY CAMPOS DATETIME NULL QUE POR ACTUALIZACION manda esa fecha. PARA EVITAR ESO SE CREA UNA CADENA QUE LES MANDA UN NULL.

string ft = "";

ft = strFechaTermReal;

if (ft == null)

{ ft = "Null"; }

else { ft ="'" + strFechaTermReal + "'"; }

DE AHÍ SE PASA ft AL SQL SIN “ ’ ” COMILLAS SIMPLES

### DataGrid

EN WPF ESTE CONTROL YA NO TIENE LAS MISMAS CARACTERISTICAS. PARA LLENAR UN GRID LA PROPIEDAD ES ITEMSOURCE Y DEBE SER UN OBJETO QUE IMPLEMENTE LA INTERFACE IENUMERABLE. ES ESTE EJEMPLO UNA COLECCIÓN LIST.

PRIMERO CREAMOS UNA CLASE AUTOR

class Author

{

public int ID { get; set; }

public string Name { get; set; }

. . .

}

LUEGO UN PROCEDIMIENTO QUE GENERE UNA COLECCIÓN LIST DEL TIPO AUTOR Y CREE UNA FILA

private List<Author> LoadCollectionData()

{

List<Author> authors = new List<Author>();

authors.Add(new Author()

{ID = 101,Name = "Mahesh Chand",BookTitle = "Graphics",DOB = new DateTime(1975, 2, 23),

IsMVP = false});

}

LUEGO DESDE EL MAINWINDOW LLAMAMOS AL PROCEDIMIENTO PARA QUE CARGUE EL DATAGRID

dataGrid1.ItemsSource = LoadCollectionData();

ALGO IMPORTANTE, EN LA DECLARACION XAML EL GRID DEBE TENER TRUE EL AUTOGENERAR COLUMNAS

<Grid>

<DataGrid AutoGenerateColumns="True" Height="267" . . .

PROPIEDADES DEL GRID

Background="LightGray" RowBackground="LightYellow"

AlternatingRowBackground="LightBlue"

IsReadOnly="True"

AreRowDetailsFrozen="True" REDIMENSIONAR

HorizontalScrollBarVisibility="Visible"

VerticalScrollBarVisibility="Visible"

CanUserResizeColumns="False"

FrozenColumnCount="2"

Desde code behind

//se instancia una nueva DataTable con el nombre de la tabla de la base

DataTable dt = new DataTable("nombre real de la tabla deseada");

//se llena la DataTable con un adapter

DAdapter.Fill(dt);

DataGrid.ItemsSource = dt.DefaultView;

DAdapter.Update();

**Llenar un DataGrid con una DataTable manual**

DataTable tabV = new DataTable();

DataColumn producto = new DataColumn();

DataColumn cantidad = new DataColumn();

DataColumn total = new DataColumn();

.

.

.

producto.ColumnName = "Producto";

producto.DataType = System.Type.GetType("System.String");

cantidad.ColumnName = "Canitdad";

cantidad.DataType = System.Type.GetType("System.Int16");

total.ColumnName = "Total";

total.DataType = System.Type.GetType("System.Decimal");

tabV.Columns.Add("Producto");

tabV.Columns.Add("Cantidad");

tabV.Columns.Add("Total");

var f=tabV.NewRow();

f["Producto"] = "p1";

f["Cantidad"] = "2";

f["Total"] = "10";

tabV.Rows.Add(f);

griExistencias.ItemsSource = tabV.DefaultView;

**Formato de celdas tabla binding manual**

Grid llenado con una datatable manual, aun asi se define bimdimg. Importante quitar el autogenerado de columnas.

<DataGrid x:Name="griVenta" Grid.Column="1" Grid.Row="6" Grid.ColumnSpan="9"

Grid.RowSpan="4" IsReadOnly="True"

AutoGenerateColumns="False"

CanUserAddRows="True">

<DataGridTextColumn Header="Total" Binding="{Binding Total}">

<DataGridTextColumn.CellStyle>

<Style>

<Setter Property="TextBlock.Foreground" Value="Red"/>

<Setter Property="TextBlock.FontSize" Value="20"/>

<Setter Property="TextBlock.TextAlignment" Value="Right"/>

<Setter Property="TextBlock.Width" Value="120"/>

</Style>

</DataGridTextColumn.CellStyle>

</DataGridTextColumn>

</DataGrid.Columns>

**Recorrer un grid y acumular un valor**

var f=tabV.NewRow();

f["Producto"] = "p1";

f["Cantidad"] = "2";

f["Total"] = "10";

tabV.Rows.Add(f);

griVenta.ItemsSource = tabV.DefaultView;

decimal decSubtot = 0;

foreach(DataRow fil in tabV.Rows)

{

decSubtot += decimal.Parse(fil[2].ToString());

}

this.texSubtotal.Text = decSubtot.ToString();

**Eliminar manualmente una fila del datagrid bindeado manualmente**

Fue cargada manualmente una datatable, al eliminar la fila actual del grid automáticamente se elimina también de la datatable. En este caso no don instancias separadas. Luego de eso se puede recorrer la datatable que ya estará actualizada

DataRowView r = (DataRowView)griVenta.SelectedItem;

r.Delete();

### Slider y scroll

Trabajan sobre un rango de valores, su propiedad de vinculación es binding. Notar en el ejemplo como el slider se vincula a la propiedad height para la imagen y al value para el textbox; además de que se declara distinto.

<StackPanel>

<Slider Name="imageSizeSlider" Margin="10, 0, 0, 0"

Width="100" Orientation="Horizontal" HorizontalAlignment="Left"

Value="50" Minimum="0" Maximum="200" TickPlacement="BottomRight" TickFrequency="10"/>

<Image Source="/images/setup.bmp" HorizontalAlignment="Left"

Margin="50,20,0,0" Height="{Binding ElementName=imageSizeSlider,Path=Value}"/>

<TextBox Height="23" Name="textBox1" Width="19">

<Binding ElementName="imageSizeSlider" Path="Value"/>

</TextBox>

</StackPanel>

Si la imagen es muy grande se activan las barras de desplazamiento

<ScrollViewer HorizontalScrollBarVisibility="Auto"

VerticalScrollBarVisibility="Auto" Height="152" Width="197">

<Image Source="/images/setup.bmp" Height="390" Width="278" />

</ScrollViewer>

### Dock panel

Es un elemento contenedor que facilita la edición en de grupos de controles, siempre que se pueda debe utilizarse como elemento de orden y organización en los controles. Las propiedades interesantes son:

Width=”Auto” -> el ancho se ajusta al tamaño del formulario

HorizontalAlignment=”Stretch” ->

VerticalAlignment=”Top” -> que se mantenga arriba

Margin=”0” -> que la esquina superior izquierda comience en 0

### Barra de estado

Se puede usar para mandar mensajes de forma discreta, sin que la aplicación invada el primer plano, por ejemplo con mensajes en pantalla. El problema es que para manipularla se debe crar un subproceso Dispatcher , también se utiliza la clase Thread y la importación System.Windows.Threading. al final del proceso que queremos manejar en segundo plano por ser largo ponemos:

Action action=new Action(()=>{statusBar1.Items.Add("salvado");});

this.Dispatcher.Invoke(action, DispatcherPriority.ApplicationIdle);

### Menu

Tipo MenuItem (el mas común de los estilos de menu): Configurar un menú desde código xaml, también se puede configurar desde propiedades, en la pestaña Items (collection). Observar que con la etiqueta menuitem.icon se agrega una imagen a cada categoría del menú, y que el menú también puede recibir un estilo. Una de las propiedades mas importantes es el click de cada menúitem, con el cual se le asigna algun procedimiento al evento click.

<DockPanel Height="100" HorizontalAlignment="Stretch" Name="dockPanel1" VerticalAlignment="Top" Width="Auto">

<Menu Height="23" Name="menu1" Width="Auto" DockPanel.Dock="Top" HorizontalAlignment="Stretch" VerticalAlignment="Top" Style="{StaticResource Estilo1}>

<MenuItem Header="\_File">

<MenuItem Header="\_New member" Name="NewMember">

<MenuItem.Icon>

<Image Source="PH02736U.BMP"/>

</MenuItem.Icon>

</MenuItem>

<MenuItem Header="\_Save member" Name="saveMember">

<MenuItem.Icon>

<Image Source="PH02746U.BMP"/>

</MenuItem.Icon>

</MenuItem>

<Separator/>

<MenuItem Header="\_Exit" Name="exit"/>

</MenuItem>

<MenuItem Header="\_Ayuda" />

</Menu>

</DockPanel>

Tipo Lista: presenta ejemplos de un botón y un combo en un menú, siguiendo el ejemplo anterior y desarrollando el menu ítem de ayuda. Claro que el ejemplo es a modo de ilustración de la flexibilidad de wpf, pero siempre se debe procurar simplicidad en el diseño para que sea intuitivo

<MenuItem Header="\_Help">

<Button Content="Agregar"/>

<ComboBox>

<ComboBox.Items>

<ComboBoxItem>

op 1

</ComboBoxItem>

<ComboBoxItem>

op 2

</ComboBoxItem>

</ComboBox.Items>

</ComboBox>

</MenuItem>

### Menu contextual

se agrega al final de la etiqueta window.resouces. en el menú ítem se liga a su evento click algún método en particular. Ver que la mayoria de los controles tienen por default el menú contextual copiar, pegar y cortar, cuando uno define el propio de eliminan estos (1), y también se agrega en las propiedades del objeto que va a mostrar el menú contextual (2).

(1)

…

<ContextMenu x:Key="textBoxMenu" Style="{StaticResource Estilo1}"/>

<MenuItem Header="clear name" Name="clearname" Click="clearname\_Click"/>

</Window.Resources>

(2)

ContextMenu="{StaticResource textBoxMenu}"

Menu contextual desde código C# y con delegados: primero definimos una variable tipo menu contextual a nivel formulario (1), luego en el constructor definimos un elemento de menú y configuramos su header y su click, el cual lo mandamos a un evento existente que guarda (2), otro elemento de menú que limpia los campos (3), luego agregamos los elementos de menú al menú contextual (4), por ultimo en cada uno de los métodos llamados saveMember y clear los ligamos al menú contextual (5) \*\*ojo, en el ejercicio también se ligo este ultimo paso al método en que HABILITAN los campos (newMember), porque si no se hace jamás aparece el menú contextual del formulario\*\* pp 627 libro

(1)

private ContextMenu WCMenu = null;

(2)

MenuItem SMMenu = new MenuItem();

SMMenu.Header = "Save Member details";

SMMenu.Click += new RoutedEventHandler(saveMember\_Click);

(3)

MenuItem CFMenu = new MenuItem();

CFMenu.Header = "Clear form";

CFMenu.Click += new RoutedEventHandler(clear\_Click);

(4)

WCMenu = new ContextMenu();

WCMenu.Items.Add(SMMenu);

WCMenu.Items.Add(CFMenu);

(5)

this.ContextMenu = WCMenu;

### TextBlock

Sirve para mostrar texto de solo lectura y darle formato. Aparte permite dos subelementos: Run y LineBreack. Por ejemplo en la oración “Diplomado de Windows phone 8” podríamos utilizar el run para estilizar solo la palabra diplomado, manteniendo la integridad dela oración, y mandando el 8 a un salto de línea con otro estilo.

< TextBlock >

<Run Foreground=”Green”>Diplomado </Run>

de Windows phone

<LineBreack>

<Run FontSize=”10”>8</Run>

</LineBreack >

</TextBlock>

### TextBox

Recibir el valor de una resgistro filtrado de DataTable

this.texExistencias.Text = tabExis.Rows[0]["existencia"].ToString();

Mascara de solo nuneros. Acepta números y el enter

private void texCantidad\_KeyDown(object sender, KeyEventArgs e)

{

if(e.Key>=Key.D0 && e.Key <=Key.D9 || e.Key == Key.Enter)

{

e.Handled = false;

}

else

{

e.Handled = true;

}

}

### MediaElement

Soporta la reproducción de audio y video. Ejemplo de un control que reproduce automáticamente y con volumen 0 un video del proyecto, cosa que no se recomienda por al amaño del xap.

<MediaElement Source=”xx.mp4” Autoplay=”True” Volume=”0”/>

### ToolBar

Similar a las que miestra office u otro programa para agrupar accesos directos a commandos, obviamente por medio del evento click.

**<ToolBar>**

   <Button Click="Button\_Click"><Image Source="new.bmp"/></Button>

   <Button><Image Source="open.bmp"/></Button>

   <Button><Image Source="save.bmp"/></Button>

   <Separator/>

   <Button><Image Source="cut.bmp"/></Button>

   <Button><Image Source="copy.bmp"/></Button>

   <Button><Image Source="paste.bmp"/></Button>

   <Separator/>

   <Button><Image Source="print.bmp"/></Button>

   <Button><Image Source="preview.bmp"/></Button>

**</ToolBar>**

### ViewBox

Muy útil porque hace que todo lo que contiene se ajuste de tamaño sin perder definición. Por ejemplo la palotima de un checkbox que de otra forma no se puede mofificar su tamaño

### Expander

Excelente control que se contrae y sirve para mostrar opciones de controles

<Expander Background="AliceBlue"

FontSize="20" Foreground="DarkBlue"

HorizontalAlignment="Left"

Header="Lección Controles Básicos"

ExpandDirection="Down" IsExpanded="True"

Margin="10">

<StackPanel>

<RadioButton IsChecked="True" FontSize="16" Margin="10,0,10,0">

Excelente

</RadioButton>

<RadioButton FontSize="16" Margin="10,0,10,0">

Muy Bien

</RadioButton>

<RadioButton FontSize="16" Margin="10,0,10,0">

Bien

</RadioButton>

<Button>otro control en el expander</Button>

</StackPanel>

</Expander>

### ProgressRing

CONTROL SUMAMENTE VISTOSO ENFOCADO A MOSTRAR ACTIVIDAD.

# Eventos enrutado

POR EJEMPLO: UN BOTON QUE INCLUYE UN TEXTO Y UNA IMAGEN Y SI QUEREMOS QUE RESPONDA AL EVENTO CLICK, DEBERIAMOS HACER QUE TANTO EL TEXTO COMO LA IMAGEN IMPLEMENTEN EL MANEJADOR DEL MISMO, LO CUAL ES TEDIOSO Y MAS SI SE PIENSA QUE EL MISMO BOTON PUEDE INCLUIR MULTIPLES ELEMENTOS ANIDADOS.

EL ELEMENTO CONTENEDOR CREA UNA ESPECIE DE ARBOL DE LOS ELEMENTOS QUE CONTIENE, Y EL EVENTO ENRUTADO ES LA POSIBILIDAD DE PODER LLAMAR A LOS EVENTOS DE LOS MISMOS, LOS QUE ESTAN EN RUTA

1. HACIA FUERA COMIENZA EN EL ELEMENTO QUE LO DESENCADENA Y VA HACIA EL ELEMENTO RAIZ
2. HACIA DENTRO COMIENZA EN EL RAIZ Y VA HACIA EL QUE DESENCADENO EL EVENTO
3. DIRECTOS SOLO SE BUSCA EN EL ELEMENTO QUE LO ORIGINO

EN EL EJEMPLO EXISTE UN GRID CON VARIOS ELEMENTOS ANIDADOS, QUEREMOS QUE AL MOVER EL MOUSE INDIQUE QUE ELEMENTO ES Y LA POSICION. DE TAL MODO QUE SOLO ES NECESARIO INDICAR A NIVEL VENTANA O PAGINA (ELEMENTOS PADRE) QUE SE MANEJARA EL EVENTO MOUSEMOVE Y CON ESO TODOS LOS ELEMENTOS CONTENIDOS SE REMITIRAN A ESE EVENTO. SERIA EL CASO 1 BUBBLING

MouseMove="WindowMouseMove"

>

<Grid>

<StackPanel>

<TextBlock Margin="10" Name="TB"

FontSize="16" FontWeight="Bold" Foreground="Red">

</TextBlock>

<StackPanel HorizontalAlignment="Left">

<TextBlock Margin="10" FontFamily="Arial" FontSize="20"

FontWeight="Bold" Foreground="Orange">

Desarrolla con

</TextBlock>

<Image Source="/images/image001\_01c.gif" Height="50"/>

</StackPanel>

<StackPanel Orientation="Horizontal">

<Image Source="/images/image001\_01c.gif" Height="100"/>

<TextBlock Margin="10" VerticalAlignment="Center" FontFamily="Arial"

FontSize="20" FontWeight="Bold" Foreground="Navy">

y aprende WPF con nosotros

</TextBlock>

</StackPanel>

<StackPanel Orientation="Horizontal" HorizontalAlignment="Center">

<Button FontFamily="Arial" FontSize="20" FontWeight="Bold">

Aceptar

</Button>

<Button FontFamily="Arial" FontSize="20" FontWeight="Bold">

Cancelar

</Button>

</StackPanel>

</StackPanel>

</Grid>

</Page>

EN CODE BEHIND

private void WindowMouseMove(object sender, MouseEventArgs args)

{

//TB ES LA ETIQUETA DONDE SE MOSTRARA LA INFORMACION

TB.Text = "(" + args.GetPosition(this).X + "," +args.GetPosition(this).X + ") : " + args.Source.GetType().Name;

}

OJO, PARA HACER QUE LA RUTA SE QUEDE EN UN ELEMENTO DETERMINADO SE LE DEBE DAR TRUE AL VALOR HANDLED a la propiedad Handled del parámetro de tipo RoutedEventArgs.

EN RESUMEN, SI PONEMOS UN MANEJADOR DE EVENTO EN EL ELEMENTO CONTENEDOR, ESTE SERVIRA PARA QUE LOS ELEMENTOS CONTENIDOS BUSQUEN DICHO MANEJADOR, Y NO SEA NECESARIO CREAR O VINVULAR UNO A UNO.

### Comandos

EN WPF ALGUNOS CONTROLES YA VIENEN CON COMANDOS PREDETERMINADOS PARA HACER MAS FACIL LA IMPLEMENTACION, ASI POR EJEMPLO EL TEXTBOX YA TIENE COPIAR, PEGAR Y CORTAR A TRAVES DE SU PROPIEDAD COMMAND.

ASOCIAR UN COMANDO Cut A UN BOTON, TAMBIEN SE USA EL COMMANDTARGET CON BINDING PARA INDICAR SOBRE QUE ELEMENTO ACTUARA EL COMANDO. SOLO QUE ESTE PUNTO REMITE LA ACCION UNICAMENTE AL ELEMENTO TEXTBOX1.

<Button Command="Cut" FontSize="18" **CommandTarget="{Binding ElementName=TextBox1}"**>

              Cortar

</Button>

PARA CORREGIR ESTE PROBLEMA, SE ELIMINA COMMANDTARGET PARA QUE LA EL COMANDO CUT SE ACTIVE SOBRE EL ELEMENTO QUE TENGA EL FOCO, AHORA SOLO HAY QUE DEFINIRLO EN EL ELEMENTO PADRE EN DONDE SE ENCUENTREN LOS CONTROLES DESEADOS. AQUÍ DEFINIMOS EN EL STACK ISFOCUSSCOPE PARA QUE BUSQUE QUIEN TIENE EL FOCO, OMITIMOS COMMANDTARGET EN LOS CONTROLES Y ENTONCES LOS COMANDOS CUT Y COPY SE HABILITARAN PARA TODOS LOS CONTROLES ANIDADOS EN EL STACK QUE ASI LO SOPORTEN

<StackPanel **FocusManager.IsFocusScope="True"** Orientation="Horizontal" HorizontalAlignment="Center"

              Background="Red" Margin="10">

      <Button Command="Cut" FontSize="18">Cortar</Button>

      <Button Command="Copy" FontSize="18">Copiar</Button>

FALTA VER LA IMPLEMENTACION PARA OBJETOS QUE NO SOPORTEN COMANDOS

# Enlace a datos (Binding)

**Consideraciones sobre e paradigma MVVM**

EL BINDING ES ESENCIALMENTE LA VINCULACION DE CONTROLES CON FUENTES DE DATOS, YA SEA OTROS CONTROLES Y BASES DE DATOS.

* EN LA VISTA VAN LAS PAGINAS, LAS CUALES DEBEN TENER LA DEFINICION Y COMPORTAMIENTO DE LOS CONTROLES, EL MENOR CODIGO POSIBLE, ASI COMO LA IMPLEMENTACION DE LA INTERACCION CON EL USUARIO
* EN MODELO VAN LAS CLASES QUE SIRVEN PRECISAMENTE COMO MODELOS DE DATOS, SOLAMENTE LAS PROPIEDADES QUE DEFINEN LOS TIPOS.
* EN LA VISTAMODELO VAN LAS CLASES PUENTE ENTRE VISTA Y MODELO, ES DECIR LAS CLASES CON LAS INICIALIZACIONES, LA INSTANCIACION DEL MODELO, ETC.

## Binding entre controles

SE ENLAZA LA PROPIEDAD QUE QUEREMOS, CON DOS “PARAMETROS”: EL PRIMERO ES BINDING Y EL NOMBRE DEL ELEMENTO AL CON EL QUE QUEREMOS ENLAZAR, Y EL SEGUNDO PATH Y LA PROPIEDAD DE ESE ELEMENTO CONTRA LA QUE SE ENLAZA:

1. TEXTBOX: EL TEXT DE T2 CONTRA EL TEXT DE T1

<TextBox Name="T2” Text="{Binding ElementName=T1, Path=Text}"/>

1. TEXT DE T2 CONTRA EL WIDTH DE T1

<TextBox Name="T2" Text="{Binding ElementName=T1, Path=Width}"/>

1. EL WIDTH DE T1 CONTRA EL VALUE DE UN SLIDER

<TextBox Name="T2” Width="{Binding ElementName=SL, Path=Value}"/>

1. EL TEXT DE T1 CONTRA SU PROPIO TAG \*\*VISTO EN APP W8\*\*

<TextBox Name="T2" Text="{Binding RelativeSource={RelativeSource Self}, Path=Tag}"

Tag="MI TAG"/>

1. EL ISACTIVATE DEL PROGRESSRING CONTRA EL ISCHECKED DE UN CHECK LLAMADO ch1

<ProgressRing IsActive="{Binding ElementName=ch1, Path=IsChecked}"/>

1. EL TEXT Del textbox se enlace al listbox y cada que se seleccione un elemento del list se refleje en el text.

<StackPanel DataContext="{Binding selecteditem, ElementName= l1}">

<TextBox Text="{Binding Path=Nombre}" VerticalAlignment="Top" Width="120"/>

<ListBox x:Name="l1" ItemsSource="{Binding Path=Personas, Source={StaticResource RecDatos}}"/>

1. AHORA CREAMOS EL RECURSO Y EL CODIGO DE UN LISTBOX, QUE ES UN CONTROL QUE SOPORTA LA CONEXIÓN CON COLECCIONES DE DATOS.

<NamSpaEnlaceDatos:claDatos x:Key="RecDatos"/>

<ListBox ItemsSource="{Binding Path=Personas, Source={StaticResource RecDatos}}">

En wp8 binding también cuenta con otras propiedades importantes, por ejemoplo stringformat=’c’ que dara formato de números a una cantidad, o targetnullvalue=’n/d’ que mostrara esa cadena en valores null. Fallbackvalue=’error’ si la expresión de enlace causa error

AQUÍ SE UTILIZA LA CLASE BINDING DEL ESPACIO System.Windows.Data.Binding, Y COMO ES SABIDO EVITA ESCRIBIR CODIGO DEL LADO DEL CLIENTE COMO DEL LADO DE LOS DATOS. ESTA CLASE BINDING SE PUEDE UTILIZAR ENTRE CONTROLES O ENTRE CONTROLES Y FUENTES DE DATOS ADO.NET

CHECAR LOS MODIFICADORES DE BINDING

Mode=OneTime SOLO SE ACTUALIZA UNA VEZ, DE LA PROPIEDAD AL CONTROL

ONEWAY, SE ACTUALIZA SIEMPRE DE LA PROPIEDAD AL CONTROL

TwoWat SE ACTUALIZA SIEMPRE EN AMBOS SENTIDOS

## Enlace a datos mediante un objeto de negocio

ADEMAS DEL BINDING ENTRE CONTROLES, EN ESTE PARADIGMA DE PROGRAMACION TAMBIEN EXISTE EL BINDING CONTRA UN OBJETO DE NEGOCIO, QUE ES EL AMBIENTE EN EL QUE NORMALMENTE SE UTILIZARA. ESTE TIPO DE VINCULACION SE PUEDE DIVIDIR EN BINDIGN A PROPIEDADES INDIVIDUALES Y A COLECCIONES, Y AMBOS NECESITAN DE UN CONTEXTO DE DATOS EN EL CUAL BUSCAR LAS DIRECTIVAS DE COMPORTAMIENTO.

## El Contexto De Datos

CUANDO CONTAMOS CON CONTROLES ORIENTADOS A DATOS ES NECESARIO IMPLEMENTAR UN CONTEXTO DE DATOS, ASI LOS CONTROLES BUSCARAN EN EL CONTENEDOR PADRE LAS DIRECTIVAS PARA HACER BINDING, APARTE CON DATA CONTEXT SE ESTABLECE UNA FUENTE DE DATOS EN UN CONTENEDOR PADRE Y YA NO ES NECESARIO ESTABLECERLO EN LOS OBJETOD HIJO, EVITA REPETICION DE CODIGO. En el ejercicio siguiente ya no será necesario poner un source en cada control enlazado.

Ver datacontext en tiempo de diseño. Útil para ver en tiempo de diseño el comportamiento del enlace. Al parecer no funciona en aplicaciones wpf.

<Grid x:Name="MiGrid">

<StackPanel DataContext="{Binding Source={StaticResource RecPersona}}">

## Binding a propiedades individuales (Inotifiedpropertychanged)

EL PROBLEMA ES EL SIGUIENTE: SI CON CODIGO TRADICIONAL C# CREAMOS UNA CLASE E INTENTAMOS TENER SINCRONIZADO LOS VALORES DE LA CLASE CON ALGUNA REPRESENTACION VISUAL EN ALGUN CONTROL DE UN FORMULARIO, DEBEMOS CREAR UN NUMERO GRANDE DE LINEAS DE CODIGO EN DIFERENTES LUGARES Y OBJETOS, EM XAML EN REALIDAD SE NECESITA MUCHO MENOS PARA MANTENER ESTA SINCRONIZACION. UNA DE LAS VENTAJAS MAS IMPORTANTES ES QUE INCLUSO EN TIEMPO DE DISEÑO SE PUEDEN VER LOS VALORES DE ANLACE.

PARA ELLO DEBEMOS TENER UN **DATACONTEXT,** SOBRE EL CUAL BUSQUEN LOS CONTROLES LAS DIRECTIVAS DE LA VINCULACION, LUEGO ESTE DATCONTEXT DEBE IMPLEMENTAR LA INTERFACE INotifiedPropertyChanged, AL HACERLO, SE DEBE GENERAR UN CONTROLADOR DE EVENTO PropertyChangeEventHandler QUE SERA EL QUE MANTENDRA ACTUALIZADOS LOS CAMBIOS Y FINALMENTE SE DEBE LLAMAR A ESTE EVENTO DESDE EL CODIGO QUE MODIFICA EL VALOR: USUALMENTE EL SET DE LA PROPIEDAD QUE MODIFICA EL CAMPO, ESTO MEDIANTE EL CODIGO PropertyChange(EL-CONTEXTO, PropertyChangeEventArgs(“ LA-PROP-QUE-LLAMA”))

**EJEMPLO:**

COMENCEMOS CON UN EJEMPLO DE DOS TEXT QUE USAN BINDING A DOS PROPIEDADES DE UNA CLASE QUE UTILIZAMOS COMO DATACONTEXT.

LA CLASE ES claPersona

<LA CAPA VISTA O PAGINA>

<TextBox Text="{Binding Nombre}" Grid.Column="0" Grid.Row="0"></TextBox>

<TextBox Text="{Binding Apellido}" Grid.Column="1" Grid.Row="0"></TextBox>

EN EL CODEBEHIND CREAMOS LA INSTANCIA DE LA CLASE COMO OBJETO A NIVEL PAGINA, ADEMAS DEFINIMOS COMO DATACONTEXT A LA CLASE YA NO ES NECESARIO PONERLES UN PATH, SOLO REFRENCIAR LAS PROPIEDADES.

<EN CODEBEHIND DE LA PAGINA>

//este es el objeto creado a nivel pagina para ser usado en diversos momentos

claPersona myModelo = new claPersona();

protected override void OnNavigatedTo(NavigationEventArgs e)

{

//esto es como el load del win forms. tambien podria ir codigo en el initializecomponent

this.DataContext = myModelo;

}

//este es un botón que cambia las propiedades de la clase y se refleja en los text

private void butAgregar\_Click\_1(object sender, RoutedEventArgs e)

{

myModelo.Nombre = "adan";

myModelo.Apellido = "perez";

}

<EL MODELO DE DATOS O DATACONTEXT>

public class claPersona: INotifyPropertyChanged

{

//implementa INotifyPropertyChange e implementa al mismo tiempo un manejador de evento PropertyChange

//al implementar bindablebase en ves de INotifyPropertyChange, ya no es necesario un void en los set, ya que bindable proporciona un método setproperty que solo hace set cuando realmente cambia el valor de la propiedad, tampoco se necesita un manejador de eventos “public event PropertyChangedEventHandler PropertyChanged;”. Bindable es una interface mas sofisticada que INotify…

public string strNombre;

public string Nombre

{

get { return strNombre; }

set

{

strNombre = value;

RaisePropertyChanged(); }

}

public string strApellido;

public string Apellido

{

get { return strApellido; }

set

{

strApellido = value;

RaisePropertyChanged(); }

}

//el constructor que pone valores de inicio

public claPersona()

{

this.Nombre = "";

this.strApellido = "";

}

//el procedimiento para simplificar el set de las propiedades

private void RaisePropertyChanged([CallerMemberName] string Name = "")

{

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs(Name));

}

//este es el avento publico

public event PropertyChangedEventHandler PropertyChanged;

**OTRO EJEMPLO DE BINDING DECLARANDO EL NAMESPACE Y EL RECURSO.**

OBSERVAR QUE TAMBIEN SE PUEDE DECLARAR EL NAME SPACE Y DATACONTEXT EN XAML. LO PRIMERO QUE HAY QUE HACER CREAR UNA ENTRADA AL NAMESPACE XMLNS. LUEGO CREAMOS EL RECURSO Y DAMOS VALORES PARA EL CONSTRUCTOR DE LA CLASE QUE YA HEMOS DEFINIDO PARA QUE DE VALORES POR DEFAULT AL FORMULARIO. NOTAR QUE AL GRID LE DAMOS UN NOMBRE PARA REFERENCIARLO EN CODE BEHIND (MiGrid).

<Window x:Class="EnlaceDatos.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:NamSpaEnlaceDatos="clr-namespace:EnlaceDatos">

<Grid x:Name="MiGrid">

<Grid.Resources>

<NamSpaEnlaceDatos:claPersona x:Key="RecPersona" Nombre="adan" Pais="mexico"/>

</Grid.Resources>

EN LA INTERFACE DE USUARIO ENLAZAMOS MEDIANTE BINDING. LAS TRES PROPIEDADES PRINCIPALES SON SOURCE, PATH Y MODE.

Modo1: onetime se hace el anlace y no se vuelven a comunicar mas

Modo2: oneway cuando fuente cambia notifica automáticamente a destino, sincronizando. Es la default

Modo3: twoway cuando cambia fuente y destino se notifican automáticamente

<TextBox Text="{Binding Path=Nombre, Source={StaticResource RecPersona}}"/>

<TextBox Text="{Binding Path=Pais, Source={StaticResource RecPersona}}"/>

<Button Content="Cambiar"/>

EL CODIGO DEL BOTON QUE CAMBIA LAS PROPIEDADES DE LA CLASE:

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

var p=MiGrid.Resources["RecPersona"] as claPersona;

p.Nombre = "otro nombre";

p.Pais = "otro pais";

}

EN ESTE EJEMPLO, **LA CLASE claNotificador SE IMPLEMENTO A MANERA DE CLASE BASE PARA DISPARAR MAS FACILMENTE EL EVENTO PROPERTYCHANGED, que es el único evento que hereda INotifyPropertyChanged. ESTO ULTIMO EN REALIDAD ES UNA IMPLEMENTACION DEL MODELO MVVM EN LA CUAL SE MAXIMIZA LA REUTILIZACION Y MINIMIZA EL CODIGO.**

class claPersona:claNotificador

{

string strNombre;

public string Nombre

{

get { return strNombre; }

set { strNombre = value;

OnPropertyChanged("Nombre");

}

}

string strPais;

public string Pais

{

get { return strPais; }

set { strPais = value;

OnPropertyChanged("Pais");

}

}

public override string ToString()

{

return string.Format("{0} de {1} " + Nombre, Pais );

}

}

**Novedad:** al llamar a OnPropertyChanged("Nombre"); debemos pasarle como parámetro la cadena con el nombre de la propiedad. Pero que pasaría si cambiaramos ese nombre? Que la llamada al método no se modificaria automáticamente. Por ello en wp8 existe [CallerMemberName] string propertyname =null que se pone en el método que recibe la llamada, con lo cual ya sabe que método lo esta llamando y solo bastaría con OnPropertyChanged();

.

public class claNotificador:INotifyPropertyChanged

{

protected virtual void OnPropertyChanged(string propertyname)

//aquí seria ([CallerMemberName] string propertyname =null)

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(propertyname));

}

}

public event PropertyChangedEventHandler PropertyChanged;

}

Una novedad en VS2012 fue la interface BindableBase que aventajaba a INotify en que solo actualizaba si la propiedad realmente hubiera sido cambiada, pero esta interface ya no apareció para VS 2013 y posteriores. En el proyecto de estudio básico de XAML se realizo la clase BindableBase y su propiedad SetProperty de manera manual porque es muy útil.

(actualización de sep 2015)

## Binding a listas o colecciones (Inotifiedcollectionchanged)

**EL EJEMPLO TRATA DE UNA LISTA DE ACTRICES, EL CONSTRUCTOR DE LA VISTAMODELO CARGA TRES Y CON LOS BOTONES SE PUEDEN AGREGAR Y BORRAR MAS ACTRICES, PERO SOLO EN MEMORIA, NUNCA SE ACCESA A UN ALMACEN DE DATOS TAL COMO SQLSERVER. SE DICE QUE EN EL PARADIGMA DE APPS EN WINDOWS STORE LA INTERACCION CON BASES DE DATOS ES MEDIANTE SERVICIOS WEB O A LO MAXIMO CON SQL-LITE.**



**LA CAPA VISTA (PAGE.XAML)**

**HAY QUE ADVERTIR QUE EN ASSETS SE ENCUENTRAN LAS FOTOS, QUE SE AGREGO UNA PAGINA BASICA PARA QUE SE AGREGARAN LAS CLASES BIBDABLEBASE A COMMON.**

<Page

x:Class="BindingColecciones.vista.ListasEstaticas"

. . .

xmlns:local="using:BindingColecciones.vista"

xmlns:vm="using:BindingColecciones.vistamodelo"

. . .>

//RECURSOS DE STYLE PARA DAR FORMATO A LOS CONTROLES

<Page.Resources>

<Style x:Key="Stack\_H" TargetType="StackPanel">

<Setter Property="Orientation" Value="Horizontal"/>

</Style>

<Style x:Key="Stack\_V" TargetType="StackPanel">

<Setter Property="Width" Value="500"/>

<Setter Property="Margin" Value="5"/>

</Style>

<Style x:Key="img" TargetType="Image">

<Setter Property="Width" Value="150"/>

<Setter Property="Height" Value="150"/>

<Setter Property="Margin" Value="5"/>

<Setter Property="Stretch" Value="UniformToFill"/>

</Style>

<Style x:Key="txt" TargetType="TextBlock">

<Setter Property="MaxHeight" Value="150"/>

<Setter Property="TextTrimming" Value="WordEllipsis"/>

<Setter Property="TextWrapping" Value="Wrap"/>

</Style>

<Style x:Key="brd" TargetType="Border">

<Setter Property="BorderBrush" Value="DarkGray"/>

<Setter Property="BorderThickness" Value="2"/>

</Style>

</Page.Resources>

<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">

//LA COLECCIÓN VISTAMODELO COMO RECURSO DEL GRID, PUEDE SER RECURSO DE LA PAGE

<Grid.Resources>

<vm:CollectionViewModel x:Name="CollectionVM" x:Key="CollectionVM"/>

</Grid.Resources>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

//SE LE PASA LA COLECCIÓN VISTAMODELO COMO ITEMSOURCE AL LISTVIEW

<ListView ItemsSource="{Binding Source={StaticResource CollectionVM},

Path=Peoplelist}" Grid.Column="0">

//LOS ITEMTEMPLATE Y DATATEMPLATE, INDISPENSABLES PARA COLECCIONES BINDEABLES A CONTROLES

<ListView.ItemTemplate>

<DataTemplate>

<Border Style="{StaticResource brd}">

<StackPanel Style="{StaticResource Stack\_H}">

<Image Style="{StaticResource img}" Source="{Binding Imagen}"/>

<StackPanel Style="{StaticResource Stack\_V}">

<TextBlock Style="{StaticResource txt}" Text="{Binding Nombre}" FontSize="45" FontWeight="Bold"/>

<TextBlock Style="{StaticResource txt}" Text="{Binding Descripcion}" FontSize="30"/>

</StackPanel>

</StackPanel>

</Border>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

<StackPanel Grid.Column="1">

<Button Content="Add" Width="200" Click="Button\_Click\_2"></Button>

<Button Content="Del" Width="200" Click="Button\_Click\_3"></Button>

</StackPanel>

</Grid>

</Page>

**LA CAPA MODELO**

//EL MODELO SOBRE EL QUE SE BASA LA COLECCION

namespace BindingColecciones.modelo

{

public class claPersona:BindableBase

{

private string strNombre;

public string Nombre

{

get { return strNombre; }

set

{

strNombre = value;

SetProperty(ref strNombre, value);

}

}

private string strDescripcion;

public string Descripcion

{

get { return strDescripcion; }

set

{

strDescripcion = value;

SetProperty(ref strDescripcion, value);

}

}

private Uri uriImagen;

public Uri Imagen

{

get { return uriImagen; }

set

{

uriImagen = value;

SetProperty(ref uriImagen, value);

}

}

}

}

//LA COLECCIÓN QUE IMPLEMENTA OBSERVABLECOLLECTION DEL TIPO CLAPERSONA

namespace BindingColecciones.modelo

{

public class PersonCollection:ObservableCollection<claPersona>

{

}

}

**LA CAPA VISTAMODELO**

//ESTA CAPA FUNCIONA COMO PUENTE ENTRE VISTA Y MODELO, AQUÍ SE IMPLEMENTAN INICIALIZACIONES Y EVENTOS. NOTAR QUE EN EL CONSTRUCTOR SE AGREGAN TRES ELEMENTOS A LA COLECCION

namespace BindingColecciones.vistamodelo

{

public class CollectionViewModel:BindableBase

{

private PersonCollection \_peoplelist = new PersonCollection();

public PersonCollection Peoplelist

{

get

{

return \_peoplelist;

}

set

{

SetProperty(ref \_peoplelist, value);

}

}

public CollectionViewModel()

{

\_peoplelist.Add(new claPersona()

{

Nombre = "Scarlett Johansson",

Descripcion = "Actriz estadounidense",

Imagen = new Uri("ms-appx:///Assets/SCARLETT.jpg")

});

\_peoplelist.Add(new claPersona()

{

Nombre = "Charlize Theron",

Descripcion = "Actriz Sudafricana",

Imagen = new Uri("ms-appx:///Assets/CHARLIZE.jpg")

});

\_peoplelist.Add(new claPersona()

{

Nombre = "Jessica Alba",

Descripcion = "Actriz estadounidense",

Imagen = new Uri("ms-appx:///Assets/JESSICA.jpg")

});

}

}

}

**PLANTILLA DE DATOS**

CUANDO HACEMOS BINDING A UNA COLECCIÓN ES NECESARIO CONTAR CON UNA PLANTILLA DE DATOS, PUESTO QUE POR LO REGULAR NO EXISTE UN NUMERO FIJO DE ELEMENTOS A FORMATEAR, ADEMAS SE DEBE TENER LA FUNCIONALIDAD DE QUE NOTIFIQUE CUANDO LA COLECCIÓN SUFRA UN CAMBIO EN SU NUMERO O FORMA DE SUS ELEMENTOS. Se puede declarar como recurso y utilizarlos en diferentes lugares.

<Grid.Resources>

<DataTemplate x:Key="DTDatos">

<StackPanel>

<TextBlock Text="{Binding Nombre}" Background="Azure"></TextBlock>

<TextBlock Text="{Binding Pais}" Background="Bisque"></TextBlock>

</StackPanel>

</DataTemplate>

AHORA PARA APLICARLA A ALGUN CONTROL ACORDE

<ListBox x:Name="l1" ItemsSource="{Binding Path=Personas,

Source={StaticResource RecDatos}}" Margin="0,150,0,0"

ItemTemplate="{StaticResource DTDatos}"/>

**OTRO EJEMPLO DE BINDING A COLECCIONES**

PARA NO TENER QUE HACER UNA IMPLEMENTACIONC COMPLETAL COMO LA ANTERIOR, UTILIZAMOS UNA COLECCIÓN GENERICA QUE SE LLAMA ObservableCollection<T> LA CUAL ES NATIVA DE LAS APLICACIONES W8 STORE Y que es una forma adecuada para enlazar colecciones. Primero creamos una nueva clase llamada claDatos. OJO, EN ESTE CASO EL NOTIFICADOR DEBE AVISAR CUANDO SE AGREGA U QUITA UN ELEMENTO DE LA COLECCIÓN, QUE PUEDE SER UNA FUENTE DE DATOS CUALQUIERA.

NOTA: ASI POR EJEMPLO, PODEMOS HACER UNA CLASE QUE IMPLEMENTE ObservableCollection<”DE-UNA-CLASE-PERSONA-P.G.”>

public class claDatos

{

//declaramos una coleccion observablecollection de tipo claPersonas

ObservableCollection<claPersona> ocPersonas;

//luego hacemos que tenga una propiedad de tipo OC y que al invocar su get tenga algun valor.

public ObservableCollection<claPersona> Personas

{

get

{

if (ocPersonas == null)

{

ocPersonas = new ObservableCollection<claPersona>();

for (int i = 0; i < 10; i++)

{

//con este código podemos agregar nuevos valores a la colección que se mostraran de manera inmediata, por ejemplo con un botón donde se agreguen nuevos valores.

ocPersonas.Add(new claPersona() {Nombre="nombre " + i.ToString(),

Pais="Pais" + i});

}

}

return ocPersonas;

}

set

{

}

}

}

## Validación

Siguiendo el caso anterior, <ExceptionValidationRule> es una regla que destacara el control responsable, para que se sepa que se ha infringido. Algo parecido a error provider.

Hay que tomar varias consideraciones:

1. La validación ocurre cuando el control a validar pierde el foco, pero por ejemplo el menú no se consider aparte del formulario, por lo cual al moverse no se perderá el foco.
2. Cuando los controles inician están vacios, por lo cual solo validara cuando se haya alterado su estado inicial.

<Window x:Class="OrderTickets.TicketForm"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="Ticket Orders" Height="250" Width="480" ResizeMode="NoResize"

xmlns:myTO="clr-namespace:OrderTickets">

<!--myTO es como using que permite utilizar los recursos que estan en OrderTickets.

a partir de este ahora podemos declarar recursos en windiw.resources a traves de myTO

por ejemplo: myTO:ClaseX x:key="NOM" PROPIEDAD1="XXX" con lo

cual estamos accesando a la claseX con key NOM y poniendole XXX a su propiedad1, esto

en caso de que dicha clase tenga un constructor para definir valores de inicio-->

<Window.Resources>

<!--al darle x:key="" estamos definiendo un nombre con el que se referenciara esta

instancia de la clase TicketOrder-->

<myTO:TicketOrder x:Key="OrderData"/>

<!--añadimos el converter como recurso-->

<myTO:PrivilegeLevelConverter x:Key="privilegeLevelConverter"/>

<!--para mostrar los mensajes de error se declara un estilo que detecta como true validation error

y hace que el error se muestre como tooltip del control. despues se pone la referencia

al estilo en cada control-->

<Style x:Key="errorStyle" TargetType="Control">

<Style.Triggers>

<Trigger Property="Validation.HasError" Value="True">

<Setter Property="ToolTip" Value="{Binding RelativeSource={x:Static

RelativeSource.Self}, Path=(Validation.Errors)[0].ErrorContent}">

</Setter>

</Trigger>

</Style.Triggers>

</Style>

</Window.Resources>

<Grid>

<DockPanel Name="dockPanel1" Height="34" VerticalAlignment="Top" Margin="0">

<Menu Height="22" Name="menu1" Width="Auto" VerticalAlignment="Top" >

<MenuItem Header="\_File" >

<MenuItem Header="\_Purchase" Name="purchaseTickets" Click="purchaseTickets\_Click" />

<MenuItem Header="E\_xit" Name="exit" Click="exit\_Click" />

</MenuItem>

</Menu>

</DockPanel>

<Label Content="Event:" Height="28" HorizontalAlignment="Left" Margin="21,24,0,0" Name="label1" VerticalAlignment="Top" Width="120" />

<ComboBox Height="23" HorizontalAlignment="Left" Margin="156,29,0,0" Name="eventList"

VerticalAlignment="Top" Width="205" Style="{StaticResource errorStyle}">

<!--conexion al error style-->

<ComboBox.Items>

<ComboBoxItem>

Party in Great Shevington Park

</ComboBoxItem>

<ComboBoxItem>

Daytrip to Upper Gumtree

</ComboBoxItem>

<ComboBoxItem>

Little Mudford Festival

</ComboBoxItem>

<ComboBoxItem>

Downley Hatch Summer Banquet

</ComboBoxItem>

</ComboBox.Items>

<ComboBox.Text>

<!--binding a la instancia de clase-->

<Binding Source="{StaticResource OrderData}" Path="EventName" UpdateSourceTrigger="Explicit">

<Binding.ValidationRules>

<ExceptionValidationRule></ExceptionValidationRule>

</Binding.ValidationRules>

</Binding>

</ComboBox.Text>

</ComboBox>

<Label Content="Customer Reference:" Height="28" HorizontalAlignment="Left" Margin="21,73,0,0" Name="label2" VerticalAlignment="Top" Width="120" />

<!--aplicamos el estilo de error en cada control-->

<TextBox Height="23" HorizontalAlignment="Left" Margin="156,78,0,0"

Name="customerReference" VerticalAlignment="Top" Width="205"

Style="{StaticResource errorStyle}">

<!--estamos vinculando a la instancia OrderData a la propiedad CustomerReference-->

<Binding Source="{StaticResource OrderData}" Path="CustomerReference" UpdateSourceTrigger="Explicit">

<Binding.ValidationRules>

<ExceptionValidationRule></ExceptionValidationRule>

</Binding.ValidationRules>

</Binding>

</TextBox>

<Label Content="Privilege Level:" Height="28" HorizontalAlignment="Left" Margin="21,120,0,0" Name="label3" VerticalAlignment="Top" Width="120" />

<ComboBox Height="23" HorizontalAlignment="Left" Margin="156,125,0,0" Name="privilegeLevel"

VerticalAlignment="Top" Width="205" Style="{StaticResource errorStyle}">

<!--conexion al error style-->

<ComboBox.Items>

<ComboBoxItem IsSelected="True">

Standard

</ComboBoxItem>

<ComboBoxItem>

Premium

</ComboBoxItem>

<ComboBoxItem>

Executive

</ComboBoxItem>

<ComboBoxItem>

Premium Executive

</ComboBoxItem>

</ComboBox.Items>

<ComboBox.Text>

<Binding Source="{StaticResource OrderData}" Path="PrivilegeLevel" Converter="{StaticResource privilegeLevelConverter}"

UpdateSourceTrigger="Explicit">

<!--ojo, aqui tambien se concluye que la validacion es mas efectiva cuando se

aplica en un momento definido para todos los controles, lo mismo que con el error

provider y demas enfoques. esto se hace con UpdateSourceTrigger="Explicit" en todos los

controles y el momento seria en el purchase del menu-->

<Binding.ValidationRules>

<ExceptionValidationRule></ExceptionValidationRule>

</Binding.ValidationRules>

</Binding>

</ComboBox.Text>

</ComboBox>

<Label Content="Tickets:" Height="28" HorizontalAlignment="Left" Margin="21,165,0,0" Name="label4" VerticalAlignment="Top" Width="120" />

<Slider Height="22" HorizontalAlignment="Left" Margin="156,171,0,0" Name="numberOfTickets" VerticalAlignment="Top" Width="205" SmallChange="1"

TickPlacement="BottomRight" Maximum="10" IsSnapToTickEnabled="True"

Style="{StaticResource errorStyle}">

<!--conexion al error style-->

<Slider.Value>

<Binding Source="{StaticResource OrderData}" Path="NumberOfTickets" UpdateSourceTrigger="Explicit">

<Binding.ValidationRules>

<ExceptionValidationRule></ExceptionValidationRule>

</Binding.ValidationRules>

</Binding>

</Slider.Value>

</Slider>

<TextBox Height="23" HorizontalAlignment="Left" Margin="377,167,0,0"

Name="Tickets" VerticalAlignment="Top" Width="19"

IsReadOnly="False">

<Binding ElementName="numberOfTickets" Path="Value" Mode="TwoWay"/>

</TextBox>

</Grid>

</Window>

EN CODE BEHIND DE LA CLASE MAIN, EN ESTE CASO LA TicketForm SERIA DE LA SIGUIENTE MANERA:

namespace OrderTickets

{

public partial class TicketForm : Window

{

public TicketForm()

{

InitializeComponent();

}

private void purchaseTickets\_Click(object sender, RoutedEventArgs e)

{

//aqui es donde se definio seria el momento de desencadenar la validacion

BindingExpression eventBe = eventList.GetBindingExpression(ComboBox.TextProperty);

BindingExpression customerReferenceBe = customerReference.GetBindingExpression(TextBox.TextProperty);

BindingExpression privilegeLevelBe = privilegeLevel.GetBindingExpression(ComboBox.TextProperty);

BindingExpression numberOfTicketsBe = numberOfTickets.GetBindingExpression(Slider.ValueProperty);

eventBe.UpdateSource();

customerReferenceBe.UpdateSource();

privilegeLevelBe.UpdateSource();

numberOfTicketsBe.UpdateSource();

if (eventBe.HasError || customerReferenceBe.HasError || privilegeLevelBe.HasError

|| numberOfTicketsBe.HasError)

{

}

else {

Binding ticketOrderinding = BindingOperations.GetBinding(privilegeLevel, ComboBox.TextProperty);

TicketOrder ticketOrder = ticketOrderinding.Source as TicketOrder;

MessageBox.Show(ticketOrder.ToString(), "Purchase");

}

}

private void exit\_Click(object sender, RoutedEventArgs e)

{

this.Close();

}

}

}

EL CODE BEHIND DE LAS CLASES AUXILIARES SERIA:

using System.Windows.Data;

namespace OrderTickets

{

enum PrivilegeLevel { Standard, Premium, Excecutive, PremiumExcecutive}

class TicketOrder

{

private string eventName;

public string EventName

{

get { return this.eventName; }

set{

if (string.IsNullOrEmpty(value))

{

throw new ApplicationException("especifica evento");

}

else

{

this.eventName = value;

}

}

}

private string customerRefrence;

public string CustomerReference

{

get { return customerRefrence; }

set {

if (string.IsNullOrEmpty(value))

{

throw new ApplicationException("especifica cliente");

}

else

{

this.customerRefrence = value;

}

}

}

private PrivilegeLevel privilegeLevel;

public PrivilegeLevel PrivilegeLevel

{

get { return privilegeLevel; }

set {

this.privilegeLevel = value;

if(!this.checkPrivNumTickets(value,this.numberOfTickets))

{

throw new ApplicationException("privilegio menos para este numero de tick");

}

}

}

private bool checkPrivNumTickets(PrivilegeLevel pl, short nt)

{

bool retVal = false;

switch (pl)

{

case PrivilegeLevel.Standard:

retVal = (nt <= 2);

break;

case PrivilegeLevel.Premium:

retVal=(nt<=4);

break;

case PrivilegeLevel.Excecutive:

retVal=(nt<=8);

break;

case PrivilegeLevel.PremiumExcecutive:

retVal=(nt<=10);

break;

}

return retVal;

}

private short numberOfTickets;

public short NumberOfTickets

{

get { return numberOfTickets; }

set {

this.numberOfTickets = value;

if (!this.checkPrivNumTickets(this.privilegeLevel, value))

{

throw new ApplicationException("muchos tick para lso privilegios");

}

if (this.numberOfTickets <= 0)

{

throw new ApplicationException("debes comprar un ticket");

}

}

}

public override string ToString()

{

string formatString = string.Format("Evento: {0}\tCustomer: {1}\tPrivilege: {2}\tTickets: {3}",

this.eventName,this.customerRefrence,this.privilegeLevel.ToString(),this.numberOfTickets.ToString()

);

return formatString;

}

}

//ojo, para poder vincular el combo privilegelevel a la propiedad privilegelevel

//de la clase, solo que el primero es una text y el segundo una enumeracion, por lo

//cual se implementa esta clase conversora. que usa using System.Windows.Data;

[ValueConversion(typeof(string), typeof(PrivilegeLevel))]

public class PrivilegeLevelConverter: IValueConverter

{

public object Convert(object value, Type targetType, object parameter,

System.Globalization.CultureInfo culture)

{

PrivilegeLevel privilegeLevel = (PrivilegeLevel)value;

string convertedPrivilegeLevel = string.Empty;

switch (privilegeLevel)

{

case PrivilegeLevel.Standard:

convertedPrivilegeLevel = "Standard";

break;

case PrivilegeLevel.Premium:

convertedPrivilegeLevel = "Premium";

break;

case PrivilegeLevel.Excecutive:

convertedPrivilegeLevel = "Excecutive";

break;

case PrivilegeLevel.PremiumExcecutive:

convertedPrivilegeLevel = "Premium Excecutive";

break;

}

return convertedPrivilegeLevel;

}

public object ConvertBack(object value, Type targetType, object parameter,

System.Globalization.CultureInfo culture)

{

PrivilegeLevel privilegeLevel = PrivilegeLevel.Standard;

string convertedPrivilegeLevel = string.Empty;

switch ((string)value)

{

case "Standard":

privilegeLevel=PrivilegeLevel.Standard;

break;

case "Premium":

privilegeLevel = PrivilegeLevel.Premium;

break;

case "Excecutive":

privilegeLevel = PrivilegeLevel.Excecutive;

break;

case "PremiumExcecutive":

privilegeLevel = PrivilegeLevel.PremiumExcecutive;

break;

}

return privilegeLevel;

}

}

}

## Almacenamiento local de datos

Uso en el ambiente WP8

### API de .NET para windows phone

Es el espacio de nombres system.io.isolatedstorage y es un sistema de archivos para almacenamiento local, cuenta con clases especializadas. Un ejemplo sencillo en el que se guarda nombres en un archivo txt local. PRIMERO SOBRECARGAMOS EL LOAD Y EN EL CLICK DE UN BOTON. OJO QUE ESTAMOS UTILIZANDO LA CLASE IsolatedStorageFile PARA EL MANEJO DE ARCHIVOS

public partial class MainPage : UserControl

{

public MainPage()

{

InitializeComponent();

//primero sobrecargamos el loaded de mainpage

this.Loaded += MainPage\_Loaded;

}

//al iniciar verificamos si existe el archivo y si es asi lo muestra en el text

void MainPage\_Loaded(object sender, RoutedEventArgs e)

{

using(var isf=IsolatedStorageFile.GetUserStoreForApplication())

{

if (isf.FileExists(filename))

{

using(var stream =isf.OpenFile(filename, FileMode.Open))

{

using(var reader= new StreamReader(stream))

{

var content = reader.ReadToEnd();

txtName.Text = content;

reader.Close();

}

}

}

}

}

const string filename = "nombre.txt";

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

//se observa que la variable tiene muchos metodos para el manejo de datos

using (var isf = IsolatedStorageFile.GetUserStoreForApplication())

{

if (isf.FileExists(filename))

{

isf.DeleteFile(filename);

}

using (var stream = isf.CreateFile(filename))

{

using (StreamWriter writer = new StreamWriter(stream))

{

writer.WriteLine(txtName.Text);

writer.Close();

}

}

}

}

}

EN CODIGO XAML NO TIENE MAYOR COMPLEJIDAD. AHORA CADA VEZ QUE INICIE LA APLICACIÓN BUSCARA SI EXISTE EL ARCHIVO Y SI ES ASI MOSTRARA LO QUE TIENE, Y CON EL BOTON GUARDARA LA CADENA QUE SE LE PONGA AL TEXT.

OTRA FORMA MAS FACIL, TRATANDOSE DE ALMACENAMIENTO SIMPLE DE CADENAS DE TEXTO PLANO ES LA CLASE IsolatedStorageSettings, YA QUE NO CREA ARCHIVOS SONO ESPACIO EN UN DICCIONARIO LOCAL.

void MainPage\_Loaded(object sender, RoutedEventArgs e)

{

if(IsolatedStorageSettings.ApplicationSettings.Contains(filename))

{

txtName.Text = IsolatedStorageSettings.ApplicationSettings[filename].ToString();

}

const string filename = "nombre.txt";

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

if (IsolatedStorageSettings.ApplicationSettings.Contains(filename))

{

IsolatedStorageSettings.ApplicationSettings.Remove(filename);

}

IsolatedStorageSettings.ApplicationSettings.Add(filename, txtName.Text);

### Windows phone runtime

Ventaja es que permite la reutilización con w8 metro, además sus métodos son asíncronos.

Ver video 2 de la semana 2

### Acceso a tarjetas SD

La restricción es que es de solo lectura, por seguridad. Solo lee extensiones registradas en el manifiesto, en extensions de capabilities. Wp solo cuenta con celulares con una ranura

Ver video 2 de la semana 2

### Datos complejos: serailizacion y deserializacion

Seralizacion xml \*\*\* hacer el ejercicio, se ve muy util

JSon

### Bases de datos locales Necesario el proyecto de mvvm para hacer pruebas.

**SQL CE**: recordemos que es una base de datos privada por aplicación, y la herramienta de programación es a través de linq to sql. No se permite transact sql directamente. Las ventajas es que es el motor de sql server por lo que hay integración.

Primero se crea el contexto de datos: lo cual representara la BD. Dentro se crean las clases que serán tablas y sus propiedades las columnas. Requiere using System.Data.Linq.Mapping; y using System.Data.Linq; ojo que en Silverlight no funcionan estos ensamblados, además de instalar el sp2 de sqlCE el cual solo se instala y se agregan automáticamente ensamblados y referencias.

El contexto de datos EN DONDE SE DEFINE QUE EXISTIRA UNA TABLA Personas de tipo Persona:

public class PersonaDataContext : DataContext

{

public Table<Persona> Personas;

public PersonaDataContext(string connectionString) : base(connectionString) { }

}

La tabla Persona y sus columnas:

[Table]

public class Persona

{

[Column(IsDbGenerated = true, IsPrimaryKey = true)]

public int ID { get; set; }

[Column]

public string Nombre { get; set; }

[Column(CanBeNull = true)]

public string Empresa { get; set; }

public override string ToString()

{

return Nombre + " " + Empresa ;

}

}

Clase auxiliar que encapsula la creación de la base de datos \*\*\*\* podría hacerse en el constructor de la clase Persona ¿???

class PersonaDataBase

{

public static PersonaDataContext GetDatabase()

{

var context = new PersonaDataContext(@"C:\Users\adan\Desktop\personas.sdf");

if (!context.DatabaseExists())

{

context.CreateDatabase();

}

return context;

}

}

Inserción de datos:

var db = PersonaDataBase.GetDatabase();

for (int i = 0; i < 1000; i++)

{

db.Personas.InsertOnSubmit(new Persona()

{

Nombre = "Persona " + i.ToString(),

Empresa = "MVA"

});

}

db.SubmitChanges();

consulta de datos: para no hacer una plantilla de datos para el list se sobrescribe el tostring(), lo cual se dice es mas recomendable

private void Button\_Click\_2(object sender, RoutedEventArgs e)

{

var query = from persona in PersonaDataBase.GetDatabase().Personas

//where persona.Nombre.Contains("5") //contiene

//where persona.Nombre.Equals("Persona 5 MVA") //igual

orderby persona.ID

select persona;

L1.ItemsSource = query.ToList();

}

Modificación de datos: al click en el list manda a otra pagina auxiliar. Primero debemos programar el doble click de la lista, al hacerlo nos manda a una nueva ventana y llena los campos con los valores

private void L1\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

var Fil=L1.SelectedItem as Persona;

Window1 pf = new Window1();

pf.T2.Text = Fil.ID.ToString();

pf.T1.Text = Fil.Nombre;

if (pf.ShowDialog().Value)

{

Fil.ID = Convert.ToInt32(pf.T2.Text);

Fil.Nombre = pf.T1.Text;

pf.T1.IsEnabled = true;

}

Luego en el formulario de edición mediante un botón de modificar

private void B1\_Click(object sender, RoutedEventArgs e)

{

PersonaDataContext DB = new PersonaDataContext(@"C:\Users\adan\Desktop\personas.sdf");

Persona persona = DB.Personas.Single(p=>p.ID== Convert.ToInt32(this.T2.Text));

persona.Nombre = T1.Text;

DB.SubmitChanges();

**SQL LITE:** se puede trabajar con liq to sql y con transact sql directo

* Primero se debe bajar la actualización desde el menú tools-extensiones y actualizaciones, el paquete se llama SQLite for Windows Phone.
* Luego en el mismo tools-manejo de librería de paquetes-consola de manejo, ejecutamos la sentencia PM> Install-Package sqlite-net. Este comando agrega un par de clases al proyecto.
* Después descargamos del sitio de “github.com” el proyecto de sqlite net (sqlite.vcxproj) y una vez descargado lo agregamos a la solución.
* LUEGO en las propiedades del proyecto en build ponetos todas la configuraciones y todas las plataformas, además de la cadena “;USE\_WP8\_NATIVE\_SQLITE.
* POR ULTIMO AGREGAMOS LA REFERENCIA AL PROYECTO AGREGADO, QUE SE LLAMA SQLITE Y compilamos.

## Acceso a datos mediante ado.net entity model

Primero se crea un nuevo proyecto al cual se le agrega un nuevo elemento tipo ado. Net entity data model.

Se utiliza o se crea una conexión a la base de datos y con el asistente se agregan las tablas que se desee. Con esto se estará mapeando la base y se crearan clases especializadas para el manejo de datos. **Aquí se guarda por defecto el nombre de la conexión que se utilizara después, por ejemplo XXXEntities,** también se define el nombre del espacion de nombres de la aplicacion

Un elemento interesante es que las cadenas de conexión se guardan en un archivo aparte, por lo cual se puede realizar migración sin mucho trabajo. Solo se debe compilar el archivo app.config junto con la aplicación, para luego modificarlo con un bloc de notas.

LOS CONTROLES SE VINCULAN A LA FUENTE MEDIANTE BINDING QUE ES UNA ETIQUETA QUE VA EN LA PROPIEDAD QUE SE QUIERE VINCULAR. Y DESDE EL CODEBEHIND SE CARGAN LOS DATOS EN SU PROPIEDAD DATACONTEXT

### Combobox

SE CARGA EL CONTROL COMBO CON LOS ELEMENTOS DE UNA TABLA SIN NINGUN CRITERIO O FILTRO, TAL CUAL

EN LA CONFIGURACION DE WINDOW

<Window x:Class="Suppliers.SupplierInfo"

LA DECLARACION DEL RECURSO EN WINDIW RESOURCES

<!--definimos la plantilla SuppliersTemplate que utilizara el combo, notar como se genera

una fila con tres campos consecutivos que a su vez estan vinculados-->

<DataTemplate x:Key="SuppliersTemplate">

<StackPanel Orientation="Horizontal">

<TextBlock Text="{Binding Path=SupplierID}"></TextBlock>

<TextBlock Text=" : "/>

<TextBlock Text="{Binding Path=CompanyName}"/>

<TextBlock Text=" : "/>

<TextBlock Text="{Binding Path=ContactName}"/>

</StackPanel>

</DataTemplate>

</Window.Resources>

LA CONFIGURACION DEL CONTROL

<ComboBox Height="23" HorizontalAlignment="Left" Margin="10,10,0,0" Name="supplierList"

VerticalAlignment="Top" Width="520" IsSynchronizedWithCurrentItem="True"

ItemsSource="{Binding}" ItemTemplate="{StaticResource SuppliersTemplate}"

SelectionChanged="supplierList\_SelectionChanged"/>

<!--aqui es donde se implementa la plantilla en el control mediante itemtemplate y el com

portamiento con selectionchanged y issynchronized...para que el combo muestre el primer ele

mento y para que el control este sincronizado con el elemento actual de la clase de coleccion

ObjectSet<T>. Algo que es importante: por ahora itemsource tiene un enlace vacio, por lo

cual wpf enlazara con un objeto especificado en la propiedad datacontext, en code behind-->

EN CODE BEHIND

//se agregan estas referencias

using System.ComponentModel;

using System.Collections;

namespace Suppliers

{

public partial class SupplierInfo : Window

{

//se crean las variables que contendran la conexion y los datos de proveedores

//asi se guardo la configuracion de la conexion con el asistente, de hecho es el nombre

//que da por defecto NorthwindEntities

private NorthwindEntities northwindContext = null;

private Supplier supplier = null;

A LA HORA DE CARGAR EL FORMULARIO DE INICIO SE LLENA EL CONTROL

private void Window\_Loaded(object sender, RoutedEventArgs e)

{

//crea una nueva instancia de la conexion y liga en combo pro medio de su

//data context a la propiedad suppliers de la conexion

this.northwindContext = new NorthwindEntities();

supplierList.DataContext = this.northwindContext.Suppliers;

}

CARGAR UN CONTROL LIST AL SELECCIONAR UN ELEMENTO DEL COMBO

private void supplierList\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

//TOMA EL ELEMENTO ACTUAL DEL PROVEEDOR Y CARGA LA VARIABLE CON LOS PRODUCTOS DEL MIMSO

//LUEGO CARGA ESA LISTA EN EL CONTROL A TRAVES DE SU DATA CONTEXT

this.supplier = supplierList.SelectedItem as Supplier;

this.northwindContext.LoadProperty<Supplier>(this.supplier, s => s.Products);

this.productsInfo = ((IListSource)supplier.Products).GetList();

productList.DataContext = this.productsInfo;

}

### ListBox

SE CARGA EL CONTROL LIST CON LOS ELEMENTOS DE UNA TABLA SIN NINGUN CRITERIO O FILTRO, TAL CUAL

<Window x:Class="Suppliers.SupplierInfo"

<ListView Height="187" HorizontalAlignment="Left" Margin="10,43,0,0" Name="productList"

VerticalAlignment="Top" Width="520" IsSynchronizedWithCurrentItem="True"

ItemsSource="{Binding}" >

<!--aqui tambien definimos el comportamiento con el en combo, ademas de que definimos

la grilla que llevara el list, tipo datagridview. esto se hace configurando la propiedad

view

OJO: qui tambien utilizaremos una clase conversora porque los datos delprecio los manda

en formato general, pero se pretende mandarlos como moneda en euros-->

<ListView.View>

<GridView>

<GridView.Columns>

<GridViewColumn Width="75" Header="Product id" DisplayMemberBinding="{Binding Path=ProductID}"/>

<GridViewColumn Width="225" Header="Name" DisplayMemberBinding="{Binding Path=ProductName}"/>

<GridViewColumn Width="135" Header="Q P Unit" DisplayMemberBinding="{Binding Path=QuantityPerUnit}"/>

<GridViewColumn Width="75" Header="Unit Price" DisplayMemberBinding="{Binding Path= UnitPrice}"/>

</GridView.Columns>

</GridView>

</ListView.View>

</ListView>

EN CODE BEHIND

//se agregan estas referencias

using System.ComponentModel;

using System.Collections;

namespace Suppliers

{

public partial class SupplierInfo : Window

{

//asi se guardo la configuracion de la conexion con el asistente, de hecho es el nombre

//que da por defecto NorthwindEntities

private NorthwindEntities northwindContext = null;

private Product product=null;

private void Window\_Loaded(object sender, RoutedEventArgs e)

{

this.northwindContext = new NorthwindEntities();

productList.DataContext = this.northwindContext.Products;

}

NOTA: SI SE QUISIERA MOSTRAR UNA LISTA FILTRADA SEGÚN EL CRITERIO DE PROVEEDOR, SE TENDRIA QUE CAMBIAR LA DECLARACION DE LA VARIABLE private Product product=null; POR private IList productsInfo = null; QUE ES UNA LISTA ACCESIBLE POR SU INDICE. ADEMAS SE OMITIRIA CARGAR DESDE EL INICIO LA LISTA productList.DataContext = this.northwindContext.Products; Y SE AGREGARIA UN CODIGO COMO EL DE ARRIBA EN EL CUAL SE FILTRARIAN DATOS POR CADA CAMBIO DE SELECCIÓN EN EL CONTROL PADRE

### Modificar el origen

Entity framework realiza una concurrencia optimista, o sea que supone que la probabilidad de que dos o mas personas realicen un cambio al mismo tiempo es remota. Pero en realidad dos usuarios si pueden realizar una consulta y tener la misma copia de datos, realizar una modificación y una de ellas se perderá.

Como la simultaneidad pesimista es considerada muy restrictiva, entity implementa una intermedia, o sea que cada elemento de la clase entityobject cuent con MODO DE SIMULTANEIDAD. Por ejemplo, se puede hacer click en algún campo de la vista diseño de las clases generadas por entity model (.edmx) y ponerle fixed a su modo de simultaneidad.

De esta forma entity mantiene una copia en cache para comprobar que nadie ha modificado ese campo desde que se capturo, si asi es realiza el cambio, de lo contrario lo rechaza.

OJO: ESTO SE REALIZA MEDIANTE UNA TRANSACCION, ES DECIR, PUEDE UNO REALIZAR VARIAS MODIFICACIONES Y LLAMAR UNA VEZ A SAVECHENGES, SI ALGUNA DE ELLAS FALLA SE ELIMINAN TODAS

### Editar datos

**Primero realizaremos una edicion de datos de forma estatica, es decir, mandando los datos editados desde el mismo código**

En el ejemplo siguiente se hace un cambio manual en la tabla productos en el campo productname utilizando los objetos de entity model. La principal propiedad es la savechanges del objeto northwindcontext.

this.northwindContext = new NorthwindEntities();

Product product = northwindContext.Products.Single(p=>p.ProductID==14);

product.ProductName = "cambiado";

northwindContext.SaveChanges();

cargando los datos en el list para comprobar los ambios

productList.DataContext = this.northwindContext.Products;

**AHORA DE FORMA DINAMICA UTILIZANDO UN FORMULARIO AUXILIAR Y UN BOTON PARA CONFIRMAR CAMBIOS**

1.-PRIMERO SE CAPTURA EL DOBLE CLICK EN EL LISTBOX

private void productList\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

editProduct(this.productList.SelectedItem as Product);

}

2.- SE CREA UN NUEVO FORMULARIO EN DONDE SE CARGARAN LOS DATOS SELECCIONADOS DEL LIST, ESTE TENDRA AULGUN MECANISMO DE VALIDACION Y DE CONFIRMACION

private void button2\_Click(object sender, RoutedEventArgs e)

{

if (string.IsNullOrEmpty(this.productName.Text))

{

MessageBox.Show("error en nombre");

//se supone que return regresa al estado anterior sin pasar por

//this.dialogresult=true

return;

}

decimal result;

if (!decimal.TryParse(this.unitPrice.Text, out result))

{

MessageBox.Show("el precio debe ser numero");

return;

}

if (result < 0)

{

MessageBox.Show("el precio debe ser mayor a cero");

return;

}

//cantidad por unidad permite null

//el boton cancelar tiene la propiedad iscancel configurada a true pro default

this.DialogResult = true;

}

3- SE LLAMA DE ALGUNA PARTE AL PROCEDIMIENTO QUE GESTIONA LA EDICION

private void editProduct(Product product)

{

//INSTANCIA AL FORMULARIO Y CARGA LOS DATOS QUE TRAE EL PARAMETRO

//product EN SUS CONTROLES TEXT

ProductForm pf = new ProductForm();

pf.Title = "Edit Product Details";

pf.productName.Text = product.ProductName;

pf.quantityPerUnit.Text = product.QuantityPerUnit;

pf.unitPrice.Text = product.UnitPrice.ToString();

//si del formulario de edicion regresa un valor true la

//instruccion de edicion se realizara

if (pf.ShowDialog().Value)

{

product.ProductName = pf.productName.Text;

product.QuantityPerUnit = pf.quantityPerUnit.Text;

product.UnitPrice = Decimal.Parse(pf.unitPrice.Text);

//habilita el boton salvar

saveChanges.IsEnabled = true;

}

}

4.- SE CONFORMAN LOS CAMBIOS

private void saveChanges\_Click(object sender, RoutedEventArgs e)

{

try

{

this.northwindContext.SaveChanges();

saveChanges.IsEnabled = false;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error saving changes");

this.northwindContext.Refresh(RefreshMode.ClientWins, northwindContext.Products);

}

}

### Refrescar la coleccion entity

Por ejemplo si falla una transaccion no se realizan los cambiso en la base, pero la copia sigue sin actualizarse, por lo cual es conveniente realizer un refresco en los datos:

### Agregar datos

**PRIMERO DE FORMA ESTATICA DIRECTA DESDE EL CODIGO**

Se utiliza using System.Data.Objects;

//agregar datos

northwindContext=new NorthwindEntities();

//le mandamos cero porque es un campo id y sqlserver crea su propio valor

Product newProduct = Product.CreateProduct(0, "nuevo p", false);

//En el ejemplo solo se agrega un registro con 4 capos ???, ya que el 0 no lo

//toma en cuenta sqlserver

newProduct.UnitPrice = 55;

newProduct.QuantityPerUnit = "10 cajas";

ObjectSet<Product> products = northwindContext.Products;

products.AddObject(newProduct);

//aqui se agrega la clave externa a suplier

Supplier supplier = northwindContext.Suppliers.Single(s=>s.SupplierID==1);

supplier.Products.Add(newProduct);

northwindContext.SaveChanges();

**AHORA DE FORMA DINAMICA MEDIANTE EL FORMULARIO AUXILIAR**

SE LLAMA ALPROCEDIMIENTO

private void button3\_Click(object sender, RoutedEventArgs e)

{

////agregar datos

addnewProduct();

}

EL PROCEDIMIENTO PASA LOS VALORES DEL FORMULARIO

private void addnewProduct()

{

ProductForm pf = new ProductForm();

pf.Title = "New Product for " + supplier.CompanyName;

if (pf.ShowDialog().Value)

{

//de igual manera crea una nueva instancia del formulario para agregar

//si el resultado del dialog es true se agregara un nuevo registro

//es confuso lo de pasar un supplierid, se supone que toma por default el id del

//supplier que esta activo a la hora del doble click

Product newProd = new Product();

newProd.ProductName = pf.productName.Text;

newProd.QuantityPerUnit = pf.quantityPerUnit.Text;

newProd.UnitPrice = Decimal.Parse(pf.unitPrice.Text);

this.supplier.Products.Add(newProd);

this.productsInfo.Add(newProd);

saveChanges.IsEnabled = true;

}

}

SE DEBEN CONFIRMAR LOS CAMBIOS

\*\*\*\*\*\*\*\*UN EJERCICIO DIFERENTE: EN UN ENTITY DE UNA SOLA TABLA\*\*\*\*\*\*

AGREGAR DATOS.

agendaEntities1 ent = new agendaEntities1() ;

cita newCita =new cita();

newCita.dia = claAgendar.Fecha;

newCita.hora = claAgendar.Hora;

newCita.asunto = claAgendar.Asunto;

ent.citas.Add(newCita);

ent.SaveChanges();

### Eliminar datos

**PRIMERO DE FORMA ESTATICA DIRECTA DESDE EL CODIGO**

northwindContext = new NorthwindEntities();

//creamos la instruccion liq To sql en donde contendra los datos objetivo

var prodctList = from p in northwindContext.Products

where p.ProductID == 78

select p;

ObjectSet<Product> products = northwindContext.Products;

foreach (var product in prodctList)

{

products.DeleteObject(product);

}

northwindContext.SaveChanges();

**UTILIZA ES ESPACIO DE NOMBRES using System.Data.Objects;**

**AHORA EN FORMA DINAMICA**

PRIMERO SE LLAMA AL PROCEDIMIENTO

private void button5\_Click(object sender, RoutedEventArgs e)

{

deleteProduct(this.productList.SelectedItem as Product);

}

LUEGO EN EL PROCEDIMIENTO SE CONFIRMA LA ELIMINACION

private void deleteProduct(Product product)

{

MessageBoxResult response = MessageBox.Show(

String.Format("Delete {0}", product.ProductName),

"Confirm", MessageBoxButton.YesNo, MessageBoxImage.Question,

MessageBoxResult.No);

if (response == MessageBoxResult.Yes)

{

this.northwindContext.Products.DeleteObject(product);

saveChanges.IsEnabled = true;

}

}

FINALMENTE SE CONFIRMAN LOS CAMBIOS

## Consumo de datos

SE REFIERE A UNA SERIE DE CLASES ESPECIALIZADAS EN ACCESAR A DATOS DISPONIBLES EN DIFERENTES FUENTES Y FORMATOS, POR EJEMPLO XML. POR LO CUAL ESTAS CLASES DEBEN TENER METODOS ESPECIALIZADOS EN EL TRATAMIENTOS DE ESTOS DATOS.

### Webclient

Es una clase conmunmente utilizada en Silverlight, pero su uso es sencillo

### httpclient

Es una clase mas moderna, acorde con la infraestructura de programación de wp8, metro, etc. ESTA SE PUEDE IMPLEMENTAR MEDIANTE UNA LIBRERÍA DE CLASES PORTABLE, CON LO CUAL ES DISEÑADOR PRESENTARA UNA SERIE DE OPCIONES CON LAS QUE UNO DESEA “PORTABILIZAR” LA LIBRERÍA.

UN DETALLE ES QUE ESTA LIBRERÍA SE DEBE DESCARGAR DE NUGET A TRAVES DEL CONTEXTUAL DEL PROYECTO Y SOBRE LA OPCION MANAGE NUGET PACKAGES

### httpWebRequest y httpWebResponse

PERMITEN MAYOR CONTROL SOBRE LA COMUNICACIÓN

### Servicios WCF

Se agrega una referencia al servicio desde el contextual del proyecto.

### Mobile services

También se descarga de nuget, enfocada para la nube a través de w azure.

VIDEO 1 DE SEGUNDA SEMANA

# Graficos

GENERALMENTE SI QUISIERAMOS PONER TEXTO EN DETERMINADO COLOR LO PODRIAMOS HACER CON UN SIMPLE LABEL Y COLOR, SIN NECESIDAD DE CREAR UNA FIGURA GEOMETRICA. EL CASO ES QUE CUANDO SE INTENTE CREAR UN ELEMENTO DE DISEÑO ES MEJOR HACERLO CON ELEMENTOS GRAFICOS Y FIGURAS GEOMETRICAS, Y CUANDO SE INTENTE HACER INTERACCION CON EL USUARIO SE HAGA MEDIANTE CONTROLES DE USUARIO.

### Figuras geometricas

LOS GRAFICOS HEREDAN DE LA CLASE SHAPE. OJO, LOS ELEMENTOS COMO LOS POLIGONOS, POLIGONALES Y LINEAS AL ESPECIFICAR SUS COORDENADAS NO SE AJUSTAN AUTOMATICAMENTE A SU CONTENEDOR, EXCEPTO USANDO UN VIEWBOX.

EJEMPLO DE UN RECTANGULO UBICADO DENTRO DE UN CANVAS Y DE UN DOCKPANEL, EL SEGUNDO CON ESQUINAS REDONDEADAS POR RADIUSX y Y.

<Rectangle Canvas.Left="20" Canvas.Top="20" Width="100" Height="60" Stroke="Green" Fill="LightGreen"/>

<Rectangle DockPanel.Dock="Left" Width="100" Stroke="Green" Fill="LightGreen" **RadiusX="10" RadiusY="10"**/>

ELIPSE

**<Ellipse Width="100" Height="60" Stroke="Maroon" Fill="Tomato"/>**

LINEA

**<Line X1="20" Y1="20" X2="100" Y2="80" Stroke="Blue"/>**

POLIGONOS: FIGURAS CERRADAS

<Polygon Points="10,40 32,40 40,10 48,40 70,40 53,53 60,80 40,65 20,80 27,53"

Fill="Gold" Stroke="DarkGoldenrod"/>

POLIGONALES: FIGURAS ABIERTAS

### Decoradores

SON ELEMENTOS CONTENEDORES DE UN SOLO OBJETO Y APLICAN FORMATO AL MISMO. GENERALMENTE NO SON VISIBLES PORQUE NO REQUIEREN LAS MISMAS PROPIEDADES QUE SUS SIMILARES, EN ESTE CASO UN RECTANGULO

BORDER. NOTAR LAS DIFERENCIAS BACKGROUND VS FILL Y BORDERBRUSH VS STROKE. CHECAR SI LA PROPIEDAD ES **CornerRadius="10,10,0,0**". NOTAR ESTAS DIFERENCIA AL TRATARSE DE CONTROLES VS ELEMENTOS GRAFICOS.

<Border Width="100" Height="40" CornerRadius="10" Margin="20" BorderBrush="RoyalBlue" Background="AliceBlue" BorderThickness="1">

<Label VerticalAlignment="Center" HorizontalAlignment="Center">

Texto centrado

</Label>

</Border>

VIEWBOX: PARA AJUSTAR EL ELEMENTO AL TAMAÑO DEL CONTENEDOR, NO AÑADE NINGUN ELEMENTO DE FORMATO, SOLO DE DIMENSION. EN EL EJEMPLO por defecto HorizontalAlignment y VerticalAlignment tienen valor "Stretch" por lo cual el ViewBox toma las dimensiones de la ventana, A su vez, el valor por defecto de la propiedad Stretch del ViewBox es "Uniform", PARA QUE LA IMAGEN CREZCA PROPORCIONALMENTE.

OJO, LA PROPIEDAD MAS IMPORTANTE ES STRETCH:

FILL -> TODA EL AREA DEL VIEW; NONE-> SE MUESTRA TAL CUAL; UNIFORMTOFILL->SE AJUSTA A LA DIMENSION MAYOR SIN PEDER PROPORCION POR LO QUE PUEDE NO MOSTRARSE POR COMPLETO EN LA VENTANA; UNIFORM-> SE AJUSTA A LA MANOR DE MANERA PROPORCIONAL

**<Viewbox>** <Image Source="Images/Windito.png"/> **</Viewbox>**

### Figuras combinadas

Path es un elemento mediante el cual se puede dibujar una figura como combinación de otras y combine geometry nos indica la forma de cobinarlas o en que puntos de combinan a través de su propiedad geometrycombinemode=unión, xor, exclude e intersection.

Las figuras geométricas que utiliza path son primitivas, o sea que no son elementos de la interfaz de usuario, sino que son utilizadas por otros controles para dibujar sus formas.

<Path Stroke="Black">

<Path.Data>

<CombinedGeometry>

<CombinedGeometry.Geometry1>

<EllipseGeometry Center="50,50" RadiusX="20" RadiusY="20"/>

</CombinedGeometry.Geometry1>

<CombinedGeometry.Geometry2>

<EllipseGeometry Center="50,90" RadiusX="30" RadiusY="30"/>

</CombinedGeometry.Geometry2>

</CombinedGeometry>

</Path.Data>

</Path

Un **PathGeometry** define una figura plana formada a partir de la unión de segmentos (líneas, arcos, cuadráticos y curvas de bezier). La utilidad principal del **PathFigure** es su propiedad StartPoint donde debe indicarse el punto inicial a partir del cual se construirá la figura.

LINEA. Solo se debe definir el segundo punto

<Path Stroke="Black">

        <Path.Data>

          <PathGeometry>

            <PathFigure StartPoint="5,5">

              <LineSegment Point="50,50"/>

            </PathFigure>

          </PathGeometry>

        </Path.Data>

      </Path>

ARCO

<Path Stroke="Black" Grid.Column="1">

        <Path.Data>

          <PathGeometry>

            <PathFigure StartPoint="5,5">

              <ArcSegment Point="50,50" Size="45,25"/>

            </PathFigure>

          </PathGeometry>

        </Path.Data>

      </Path>

SEGUNDO EJEMPLO DE ARCO QUE DIBUJA UN CLIP

<Path Stroke="#444" StrokeThickness="3" Margin="20,20">

        <Path.Data>

          <PathGeometry>

**<PathFigure StartPoint="5,0">**

**<ArcSegment Size="10,18" Point="20,0"**

**SweepDirection="Clockwise"/>**

**<LineSegment Point="25,60"/>**

**<ArcSegment Size="7.5,9" Point="10,60"**

**SweepDirection="Clockwise"/>**

**<LineSegment Point="10,26"/>**

**</PathFigure>**

          </PathGeometry>

      </Path.Data>

    </Path>

QUADRATIC. Se define a través de 3 puntos

<Path Stroke="Black" Grid.Column="1">

        <Path.Data>

          <PathGeometry>

            <PathFigure StartPoint="5,5">

              <ArcSegment Point="50,50" Size="45,25"/>

            </PathFigure>

          </PathGeometry>

        </Path.Data>

      </Path>

BEZIER. Se define a través de 4 puntos

<Path Stroke="Black" Grid.Column="3">

        <Path.Data>

          <PathGeometry>

            <PathFigure StartPoint="5,5">

              <BezierSegment Point1="50,15" Point2="15,50" Point3="50,50"/>

            </PathFigure>

          </PathGeometry>

        </Path.Data>

      </Path>

ADEMAS DE ESTAS EXISTEN TRES OBJETOS OPTIMIZADOS DE OS MISMOS: PolyLineSegment, PolyBezierSegment y PolyQuadraticBezierSegment. VER REFERENCIA.

### Brochas

BROCHA SOLIDA:

Normalmente se utiliza mediante el método abreviado, es decir, al asignar un color al background de un elemento en realidad se le esta asignando una brocha solida:

<Button Background="Tomato"> Pulsame </Button>

Que seria el equivalente al método extendido:

<Button>

**<Button.Background>**

**<SolidColorBrush Color="Tomato"/>**

**</Button.Background>**

</Button>

EXISTEN 5 TIPOS DE BROCHAS, LA BROCHA SOLID ES LA PREDEFINIDA QUE SE VE MAS ARRIBA, Y SE PUEDE APLICAR EN SINTAXIS DE PROPIEDADES o en sintaxis de subelemento.

LA BROCHA DE GRADIENTE LINEAR SE DEBE APLICAR COMO SUBELEMENTO. Aqui mostrara el texto degradado de azul a amarillo:

<TextBlock FontSize="50" >hola desde wpf

<TextBlock.Foreground>

<LinearGradientBrush>

<GradientStop Color="Blue"></GradientStop>

<GradientStop Color="Yellow" Offset="1"></GradientStop>

</LinearGradientBrush>

</TextBlock.Foreground>

</TextBlock>

LA BROCHA RADIAL SE APLICA DE LA MISMA MANERA QUE LA LINEAR.

LA BROCHA DE IMAGEN

<TextBlock FontSize="50" >hola desde wpf

<TextBlock.Foreground>

<ImageBrush ImageSource="19062010071.jpg">

</ImageBrush>

</TextBlock.Foreground>

</TextBlock>

FINALMENTE LA BROCHA DE VIDEO. ACEPTA UN MEDIA ELEMENT COMO FUENTE, ademas de un objeto **CaptureSource y un CaptureDeviceConfiguration,** cuya fuente es la camara del dispositivo y el Segundo metodo es para utilizer la camara defult, por si existen dos. UN PUNTO IMPORTANTE ES QUE SE DEBEN HABILITAR LAS CAPABILITIES DEL DISPOSITIVO PARA PODER TILIZAR. Ver unicio del video 3.

Las capabilities se hablitan en WMAppManifest.xml

EXISTEN 5 FORMAS DE ASIGNAR COLOR

1.- LA PRIMERA YA VISTA A TRAVES DE UN ALIAS COMO “Tomato” PARA COLORES FRECUENTES QUE SE PUEDEN NOMBRAR.

2.- LA SEGUNDA CON UNA COMBINACION RGB DE 6 DIGITOS “#RRGGBB”

3.- LA TERCERA MEDIANTE UNA COMBINACION RGB DE 3 DIGITOS “#RGB” QUE ES LA FORMA ABREVIADA DEL METODO ANTERIOR.

4 Y LA CUARTA MEDIANTE UNA COMBINACION DE 8 DIGITOS “#AARRGGBB”, EN LA CUAL LOS DOS PRIMEROS INDICAN EL GRADO DE OPACIDAD. sI FUERA POR DECIR “#00555555” SERIA BLANCO A PESAR DE LOS DEMAS NUMEROS PORQUE 00 INDICA OPACIDAD NULA

5.- COMBINACION DE 4 DIGITOS “#ARGB” QUE ES LA FORMA ABREVIADA DEL METODO ANTERIOR. ES A SU VEZ LA FORMA MAS USUAL

### Gradientes

LINEALGRADIENTBRUSH

SON LOS EFECTOS DE DEGRADACION DE COLOR QUE DAN LA APARIENCIA, POR EJEMPLO CRISTALINA. EN EL EJEMPLO STARTPOINT Y ENDPOINT INDICAN DESDE DONDE COMIENZA LA COMPOSICION DEL COLOR Y CADA GRADIENTSTOP LAS CAPAS DE COLOR QUE VA A TENER EL OBJETO.

<Button.Background>

<LinearGradientBrush StartPoint="1,1" EndPoint="0,0">

<GradientStop Color="LightYellow" Offset="0"/>

<GradientStop Color="Yellow" Offset="0.4"/>

<GradientStop Color="Gold" Offset="0.6"/>

<GradientStop Color="Goldenrod" Offset="1"/>

</LinearGradientBrush>

</Button.Background>

DE ESTA FORMA PODEMOS DAR LA APARIENDIA DE VOLUMEN A LOS OBJETOS AL VARIAR LOS TONOS DEL GRADIENTSTOP. EN ESTE EJEMPLO SE UTILIZA UN BORDER QUE ES EL ELEMENTO PURAMENTE DECORATIVO, SI QUISIERAMOS PONER UN BOTON DENTRO HABRIA QUE DARLE OPACIDAD CERO Y SIN BORDER (BORDERBRUSH TAMBIEN CON OPACIDAD CERO) PARA PODER OBSERVAR EL FORMATO DEL BORDER, YA QUE EL BOTON EN SI YA CUENTA CON COLORES PROPIOS.

RADIALGRADIENTBRUSH

TAMBIEN CUENTA CON LA COLECCIÓN DE GRADIENTSTOP QUE CONFORMAN LOS ANILLOS DE COLORES DEL GRADIENT RADIAL. EL CENTRO SE UBICA POR DEFECTO EN EL CENTRO (0.5,0.5). JUNTO CON EL GRADIENTORIGIN PARA QUE EL CENTRO DEL GRADIENTE COINCIDA CON EL CENTRO INICIAL DEL COLOR. RADIUS MODIFICA LA SIMENSION DE LA ELIPSE, SIENDO EL 0.5 EL DE LA DIMENSION QUE CONTIENE EL GRADIENT EL TAMAÑO POR DEFAULT.

<Grid.Background>

**<RadialGradientBrush Center="0.3,0.4" GradientOrigin="0.3,0.4" RadiusX="1.1" RadiusY="0.9"** >

**<GradientStop Color="Red" Offset="0"/>**

**<GradientStop Color="Yellow" Offset="0.5"/>**

**<GradientStop Color="Blue" Offset="1"/>**

**</RadialGradientBrush>**

      </Grid.Background>

</Grid>

TAPIZ DE FONDO (TILEBRUSH): ESTA CLASE CUENTA CON TRES SUBCLASES, TODAS HEREDAN DE TILEBRUSH:

IMAGEBRUSH.

OJO, ES DISTINTO QUE IMAGE. PÓRQUE IMAGE ES UN ELEMENTO DE INTERFAZ DE USUARIO QUE PUEDE RECIBIR EVENTOS DEL RATON Y OTROS. IMAGEBRUSH se puede rellenar el fondo completo con una imagen sin que esto signifique que se ha puesto una  imagen como un elemento de interacción, ES PURAMENTE DECORATIVO COMO LAS DEMAS CLASES EN EL AMBITO GRAFICO.

HAY QUE NOTAR QUE LA IMAGEN SE REDIMENSIONA CONFORME EL OBJETO CONTENEDOR, EN ESTE CASO EL WINDOW. ESTO SE DEFINE EN LA PROPIEDAD STRETCH que por DEFAULT ES FILL. ASI MISMO VER LA ALINEACION X y Y. SUS PRINCIPALES FUNCIONES SON ALINEACION, DISTRIBUCION Y RELLENO.

<Window.Background>

<ImageBrush ImageSource="YELLOW.GIF" Stretch="Uniform" AlignmentX="Left" AlignmentY="Top"/>

</Window.Background>

<Grid>

</Grid>

DISTRIBUCION:

Viewbox y ViewboxUnits: Son las propiedades que permiten seleccionar una porción del contenido para aplicar la extensión alineación y relleno dentro del área de dibujo. Viewbox describe un rectángulo que representa el área de dibujo a tomar para extender y alinear la imagen, mientras que ViewboxUnits indica cómo deben interpretarse las coordenadas que se empleen para definir el Viewbox.

<Border Margin="5" BorderBrush="Black" BorderThickness="1.5" Grid.Row="3">

<Border.Background>

<ImageBrush ImageSource="YELLOW.GIF" Stretch="Uniform"

Viewbox="0.1,0.1,0.1,0.9" ViewboxUnits="RelativeToBoundingBox"/>

</Border.Background>

</Border>

Y RELLENO:

### Transformaciones

TODO ELEMENTO VISUAL ADMITE LA PROPIEDAD RENDERTRANSFORM. OJO: EN WPF NO ADMITE LA DECLARACION DE COMPOSITETRANSFORM, aparentemente solo en wp8.

<Button> hola

<Button.RenderTransform>

<CompositeTransform ScaleX="0.5"/>

</Button.RenderTransform>

</Button>

Transformaciones de escala

Transformaciones de inclinacion

Transformaciones de rotacion

Transformacin de traslacion

Transformacion de composicion, que es una mezcla de las 4.

UNA PROPIEDAD MUY UTIL ES ManipulationDelta, la cual se dispara con la interaccion tactil. Ver video 3 a la mitad.

### Animaciones

**ANIMACIONES DE LINEA DE TIEMPO**. En el ejemplo se crea una animacion que dura 10 segundos del recurso TransRecta en su propiedad TraslateX. Es importante que las animaciones deben estar en un diccionari de recursos y deben contener un storyboard. Checar otras propiedades del storyboard como AutoReverse y RepeateBehavior.

doubleAnimation

<Grid.Resources>

<Storyboard x:Name="st1">

<DoubleAnimation From="10" To="20" Duration="00:00:10"

Storyboard.TargetName="TransRecta" Storyboard.TargetProperty="TraslateX"/>

</Storyboard>

</Grid.Resources>

<Rectangle Width="300" Height="auto" Fill="Red"></Rectangle>

<Rectangle.RenderTransform>

<CompositeTransform x:Name="TransRecta" TraslateX="1200"/>

</Rectangle.RenderTransform>

Ahora en el MainPageLoaded, carga uno la pagina This.Loaded+=MainPage\_Loaded; y luego en el avento MainPage\_Loaded ponemos st1.Begin();

colorAnimation. Para esta animacion deberiamos declarar un fill al rectangulo y el coloranimation, dentro del mismo storyboard, que puede ir al mismo tiempo que la otra animacion.

<ColorAnimation From="Aqua" To="Azure"

Storyboard.TargetName="col1" Storyboard.TargetProperty="Color"/>

</Storyboard>

</Grid.Resources>

<Rectangle.fill>

<SolidColorBrush x:Name="Col" Color="Aqua"/>

</Rectangle.fill>

pointAnimation

**ANIMACIONES DE RECUADRO.** Aqui hay mas control porque podemos poner cierto comportamiento en x momento. Ojo, este tipo de animaciones se crean con el entorno de blend, ya que a mano seria complejo. En blend las herramientas son como las de flash, en donde en cada momento decidimos el comportamiento de los objetos. Ver video 3 final.

Ambas tecnicas de animacion tienen un comportamiento en el tiempo: la ya vista es lineal, el otro donde puede uno curvar el movimiento es en la pestaña Easing en KeySpline. Sin embargo ya existen 11 funciones de suavizacion, y se encuentran en la pestaña EasingFunction tambien de Blend.

Tambien se puede con codigo:

<DoubleAnimation.EasingFunction>

<QuinticEasy . . . />

</DoubleAnimation.EasingFunction >

</DoubleAnimation>

# Convertidores

Existe una clase que convierte valores y es muy útil porque se puede implementar en diferentes entornos. En una clase cualquiera implementamos la interface IValueConverter del espacio de nombres system.windows.data. Por ejemplo: si queremos controlar la visibilidad de algún objeto mediante la propiedad boleana del toogleswitch no funcionara, pues una es boleana y la otra visibility.

1. TENEMOS LA CLASE QUE IMPLEMENTE IValueConverter

namespace BindingListas.converter

{

public class ConBool2Visib:IValueConverter

{

public object Convert(object value, Type targetType, object parameter, string language)

{

var tmp = (bool)value;

return tmp ? Visibility.Visible : Visibility.Collapsed;

}

public object ConvertBack(object value, Type targetType, object parameter, string language)

{

var tmp = (Visibility)value;

return tmp==Visibility.Collapsed ? false: true;

}

}

}

1. EL ESPACIO DE NOMBRES

xmlns:cnv="using:BindingListas.converter"

1. EL RECURSO

<Page.Resources>

<cnv:ConBool2Visib x:Key="b2v"/>

1. Y LA DECLARACION DE LOS CONTROLES

<ToggleSwitch Name="tg" IsOn="True" OffContent="apagado" OnContent="encendido" />

<Rectangle Fill="Orange" Visibility="{Binding ElementName=tg, Path=IsOn,

Converter={StaticResource b2v}}"/>

**OTRO EJEMPLO DE CONVERTIDOR**

ESTE EJEMPLO ES SOBRE ?? UN STRING A SOLID COLOR BRUSH

public class claConvertidor:IValueConverter

{

public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)

{

if (value == null)

return null;

var country = value.ToString().ToLower();

SolidColorBrush result = new SolidColorBrush(Colors.White);

switch (country)

{

case "0":

result=new SolidColorBrush(Colors.Cyan);

break;

}

return result;

}

public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)

{

throw new NotImplementedException();

}

Lo declaramos en el diccionario de recursos, si debe ir declarado antes de la línea en donde se utiliza en la plantilla de datos

<NamSpaEnlaceDatos:claConvertidor x:Key="Converter"/>

Y lo llamamos desde el datatemplate del control que queremos que muestre los datos. Notese que el converter se aplica al foreground, ya que queremos que ciertos países se muestren en x color, aun siendo foreground de tipo solidbrush y pais de tipo string, el trabajo de conversión lo hace la clase claConverter.

<TextBlock Text="{Binding Pais}" Background="Bisque"

Foreground="{Binding Pais, Converter={StaticResource RecConv}}"></TextBlock>

# Commandos

Por ejemplo, cuando se escribe code behind se esta perdiendo interaccion en tiempo de diseño, además dereutilizacion de código.

Ver video 6 primera semana

# Componentes

CONTROLES DE USUARIO CAP 8 VIDEO AVANZADO, TAMBIEN VER DISEÑO DE ROTACION, ETC.

# MVVM

Este modelo tiene varias ventajas

VIEW: va todo lo que es interface de usuario, contexto de datos, minimo código, comportamiento.

VIEWMODEL: La implementación de la vista.

MODEL: Los objetos de negocio.

### String builder

Es excelente para construir complejas composiciones de texto con excelente rendimiento. Por ejemplo: crea un string con parámetros que toma de tres controles, y luego crea un string builder que agrega al primero a través de su método AppendLine (también existe el Insert y Remove, entre otros). Por ultimo agrega a la cadena cada content del elemento seleccionado dentro de un list y muestra al builder completo en un messagebox.show.

string nameTower = string.Format("member name {0} {1} from the tower at {2} rings the following methods:",

firstname.Text, lastname.Text, towername.Text);

StringBuilder details = new StringBuilder();

details.AppendLine(nameTower);

foreach (CheckBox cb in methods.Items)

{

if (cb.IsChecked.Value)

{

details.AppendLine(cb.Content.ToString());

}

}

MessageBox.Show(details.ToString(), "TITULO");

## Validación \*\* Complicaciones\*\*

Aplica para todos los lenguajes

El modelo de validación a través del LostFocus ofrece problemas porque se complica la lógica cuando se requiere vincular la validación entre controles. En el caso de wpf permite definir reglas de validación como parte del modelo de negocio, definiendo clases plenamente accesibles desde la vista xaml. Algo similar sucede en C# puro y en vb, ya que una opción es crear la validación en la implementación de las propiedades y campos de una clase.

# Generalidades del diseño Metro

MARGENES: EL ROW DE ENCABEZADOS ES DE 140 Y LA PRIMERA COLUMNA 100