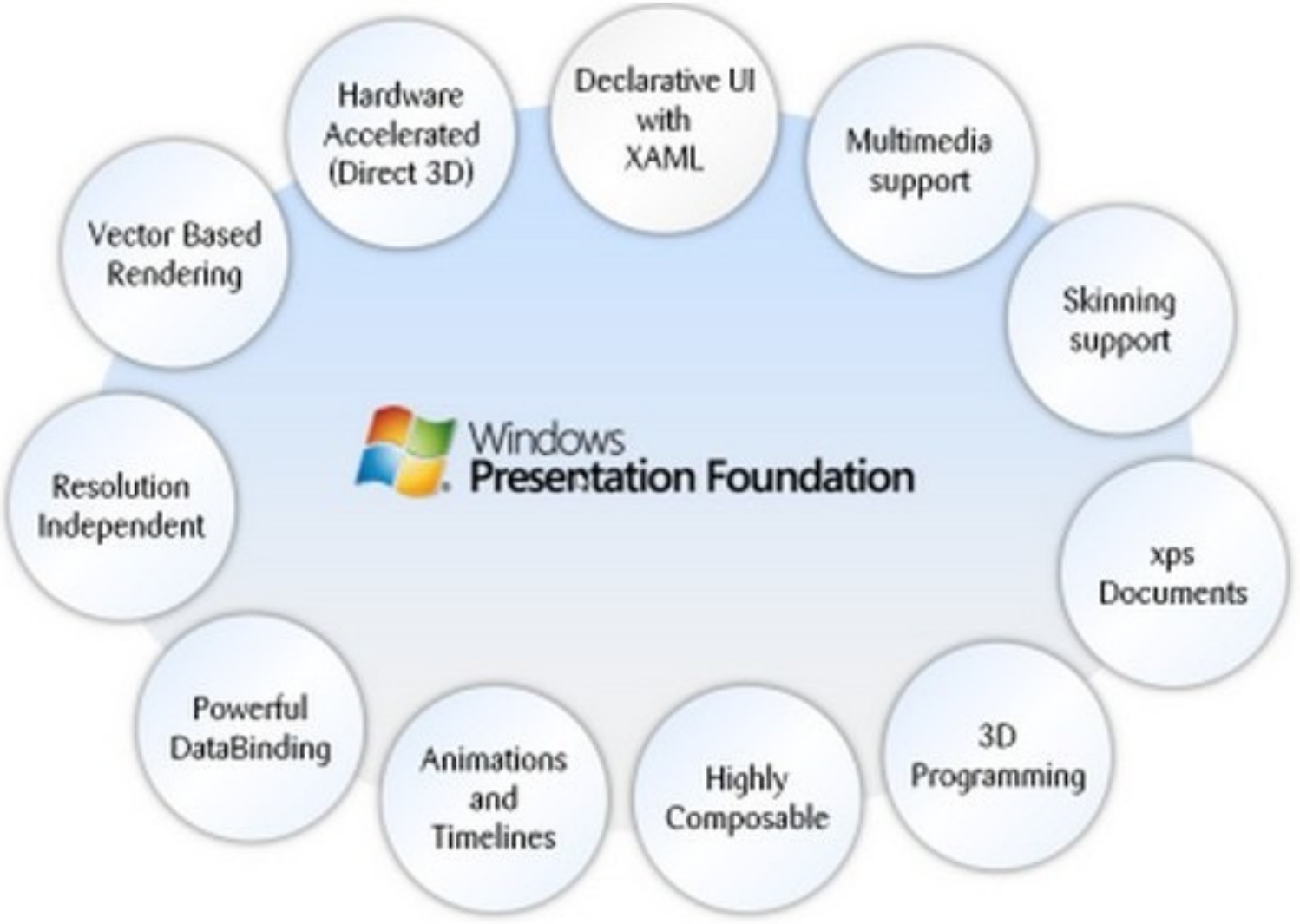
UD2.1 WPF

Ciclo: Desarrollo de Aplicaciones Multiplataforma
Módulo: Desarrollo de interfaces. Curso 2024-25

LENGUAJES DE PROGRAMACIÓN. TIPOS. PARADIGMAS DE PROGRAMACIÓN

- ¿Por qué WPF?
- Evolución histórica tecnologías IU gráficas
  - 1985, Windows 1.0 → GDI + USER (2D)
  - 1990, Silicon Graphics → OpenGL 2D, 3D
  - 1996, DirectX 2D, DirectX3D
  - DirectX 9, versión con soporte .NET
  - GDI+, User32, Windows XP
  - 2002, .NET → WinForms sobre GDI+ y User32
  - …...
- Entre tanto muchas mejoras en hardware gráfico


Hardware Accelerated (Direct 3D)
Declarative UI with XAML
Multimedia support
Skinning support
xps Documents
3D Programming
Highly Composable
Animations and Timelines
Powerful DataBinding
Resolution Independent
Vector Based Rendering
Windows Presentation Foundation

Características de WPF

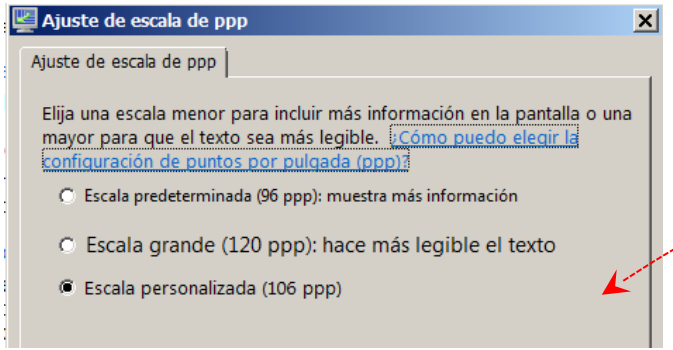
- Integración 2D, 3D, speech
- Independencia de la resolución
- 3D nativo sobre DirectX
  - Usa la aceleración hardware disponible
  - Multimedia
- Especificación declarativa del IU
  - WPF permite crear interfaces de usuario utilizando un lenguaje de marcado llamado XAML, variante XML
  - Visual Studio, así como los miembros de la familia de entornos de desarrollo de Microsoft como Blend, están preparados para generar código XAML de forma nativa.
  - Separación de comportamiento y presentación
    - Styles, Themes, Skins, Templates
- Modelo de composición muy flexible

Independencia de la resolución

- WPF usa gráficos vectoriales.
- Se ajusta mejor a las distintas resoluciones de los dispositivos.
- Tamaños en Device Independent Units (no pixels)

Device Independent Units

- Device Indendent Unit = 1/96 inch
  - A densidad normal:
    - Windows con fuentes pequeñas → 96 DPI
    - $$[\text{Physical Unit Size}] = [\text{Device-Independent Unit Size}] \times [\text{System DPI}]$$
      $$= 1/96 \text{ inch} \times 96 \text{ dpi}$$
      $$= 1 \text{ pixel}$$
  - A mayor densidad: Monitor 20” 1600x1200
    - 1/96 inch * 100 dpi = 1,042 pixel



$$[\text{Screen DPI}] = \frac{\sqrt{1600^2 + 1200^2} \text{ pixels}}{20 \text{ inches}}$$
$$= 100 \text{ dpi}$$

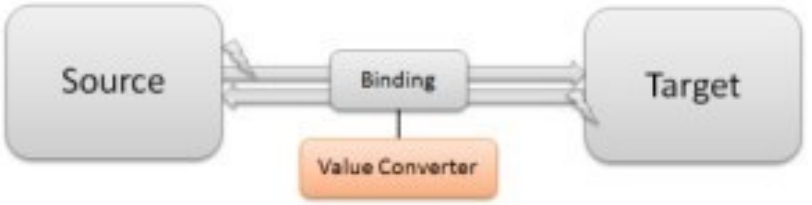
Ajustar tamaños de fuentes (ppp)

Plantillas

- En WPF podemos crear elementos reutilizables para la interfaz gráfica.
- Plantillas:
  - plantillas de control.
  - plantillas de datos.

Data Binding

- Data binding es un mecanismo mediante el cual podemos enlazar los elementos de la interfaz de usuario con los objetos que contienen la información a mostrar.
- El caso más típico de data binding es el enlazar un control de la interfaz de usuario con un valor o registro de una base de datos.



Estilos

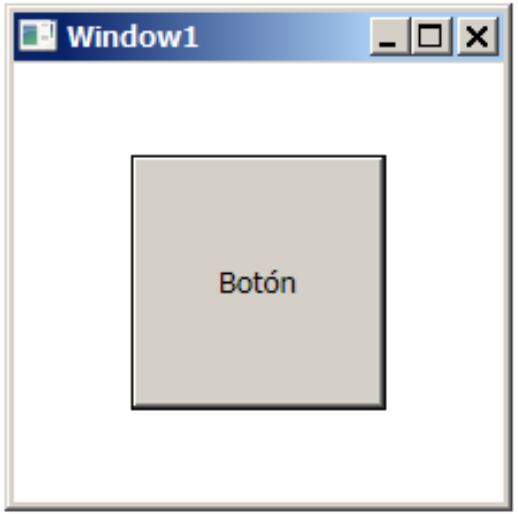
- Los estilos en WPF son muy parecidos a las hojas de estilos en cascada, CSS, para HTML. (más potentes que el CSS).
- Se puede manejar cualquier característica gráfica que te imagines, márgenes, espacios, colores, dimensiones y muchos más.

Especificación declarativa

- Separación de roles diseñador y programador
  - Los diseñadores manejan herramientas específicas
    - Resultados exportables a XAML y herramientas de conversión
    - Estilos separados del comportamiento del componente
  - Los programadores añaden el comportamiento funcional a la aplicación

Modelo de composición

- Prácticamente todos los controles pueden albergar otros elementos.



- En muchos casos es posible componer nuevos controles sin necesidad de programar

Conceptos principales en WPF

- XAML
- Jerarquía de clases
- Árbol lógico y árbol visual
- Dependency Properties
- Eventos
- Layouts
- Controles
- DataBinding

Avanzadas

- Styles
- Control templates
- Animación y Multimedia
- Gráficos 3D
- Text flow y gráficos
- Printing y XPS

XAML = eXtensible Application Markup Language

- Marcado declarativo y fácilmente utilizable
- El código y el contenido están separados
- Se puede renderizar en el navegador / aplicación independiente

XAML

```
<Button Width="100"> OK
  <Button.Background>
    LightBlue
  </Button.Background>
</Button>
```

C#

```
Button b1 = new Button();
b1.Content = "OK";
b1.Background = new
SolidColorBrush(Colors.LightBlue);
b1.Width = 100;
```

XAML introducción

- Al interpretarlo se crean árboles de objetos .NET
  - Es una forma de serializar objetos en XML
  - Cada XML-Element es una clase .NET
  - Cada XML-Property es una propiedad .NET
- Se puede programar WPF sin XAML, todo en C#; como en WindowsForms
- Crea la estructura de objetos y asigna valores a sus propiedades pero se complementa con código
  - Code behind → Manejadores de eventos

```xaml
<!-- Window1.xaml -->
<Window
  x:Class="MyFirstWpfApp.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Hello, WPF">

  <Button
    x:Name="button"
    Width="200"
    Height="25"
    Click="button_Click">Click me, baby, one more time!</Button>

</Window>
```

XAML y code behind

```xaml
// Window1.xaml.cs
using System;
using System.Windows;
using System.Windows.Controls;

namespace MyFirstWpfApp {
  public partial class Window1 : Window {
    public Window1( ) {
      InitializeComponent( );
    }

    void button_Click(object sender, RoutedEventArgs e) {
      MessageBox.Show(...);
    }
  }
}
```

XAML y equivalente C#

```
namespace MyFirstWpfApp {
  class Window1 : Window {
    public Window1() {
      this.Title = "Hello, WPF";

      // Do something interesting (sorta...)
      Button button = new Button();
      button.Content = "Click me, baby, one more time!";
      button.Width = 200;
      button.Height = 25;
      button.Click += button_Click;

      this.Content = button;
    }

    void button_Click(object sender, RoutedEventArgs e) {
      MessageBox.Show(
        "You've done that before, haven't you...",
        "Nice!");
    }
  }
}
```

```
<!-- Window1.xaml -->
<Window
  x:Class="MyFirstWpfApp.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Hello, WPF">

  <Button
    x:Name="button"
    Width="200"
    Height="25"
    Click="button_Click">Click me, baby, one more time!</Button>

</Window>
```

XAML Namespaces

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

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

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

Namespace de WPF

Namespace de propio de XAML

Namespace del proyecto

Sintaxis elementos propiedad

- La asignación de valores a propiedades se puede hacer con atributos XML

```
<TextBlock Width="50" FontWeight="Bold" Text="{Binding Path=Total}"/>
<TextBlock Text="{Binding Path=Total}">
    <TextBlock.Width>
        50
    </TextBlock.Width>
</TextBlock>
```

O con elementos anidados que sigan la sintaxis <Elemento.Propiedad>

Attached properties

```
<Window ...>
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition />
      <RowDefinition />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition />
      <ColumnDefinition />
      <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Button Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="2">A</Button>
    <Button Grid.Row="0" Grid.Column="2">C</Button>
    <Button Grid.Row="1" Grid.Column="0" Grid.RowSpan="2">D</Button>
    <Button Grid.Row="1" Grid.Column="1">E</Button>
    <Button Grid.Row="1" Grid.Column="2">F</Button>
    <Button Grid.Row="2" Grid.Column="1">H</Button>
    <Button Grid.Row="2" Grid.Column="2">I</Button>
  </Grid>
</Window>
```

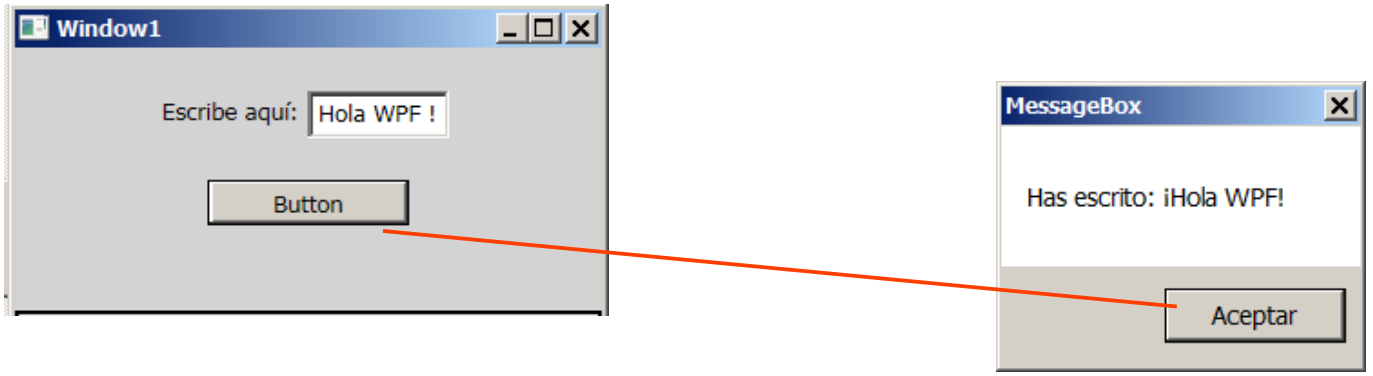
- Propiedades que se establecen para un elemento (Grid) desde otro contenido (Button) que no las posee

Markup extensions

- Permiten la asignación de valores en tiempo de ejecución
- Entre llaves, expresión que se evalúa

```
...
<Style TargetType="{x:Type Button}">
...
```

Type	Usage
x:NullExtension	Used to indicate null (evaluated at compile time)
x:TypeExtension	Retrieves type object (evaluated at compile time)
x:ArrayExtension	Creates an array
x:StaticExtension	Retrieves static property value
StaticResourceExtension	Performs one-shot resource lookup
DynamicResourceExtension	Sets up resource binding
ComponentResourceKey	Creates a resource key for cross-component system resource references
Binding	Creates a data binding
RelativeSource	Creates a RelativeSource for use in a data binding
TemplateBinding	Connects a property in a control template to a property of the templated control

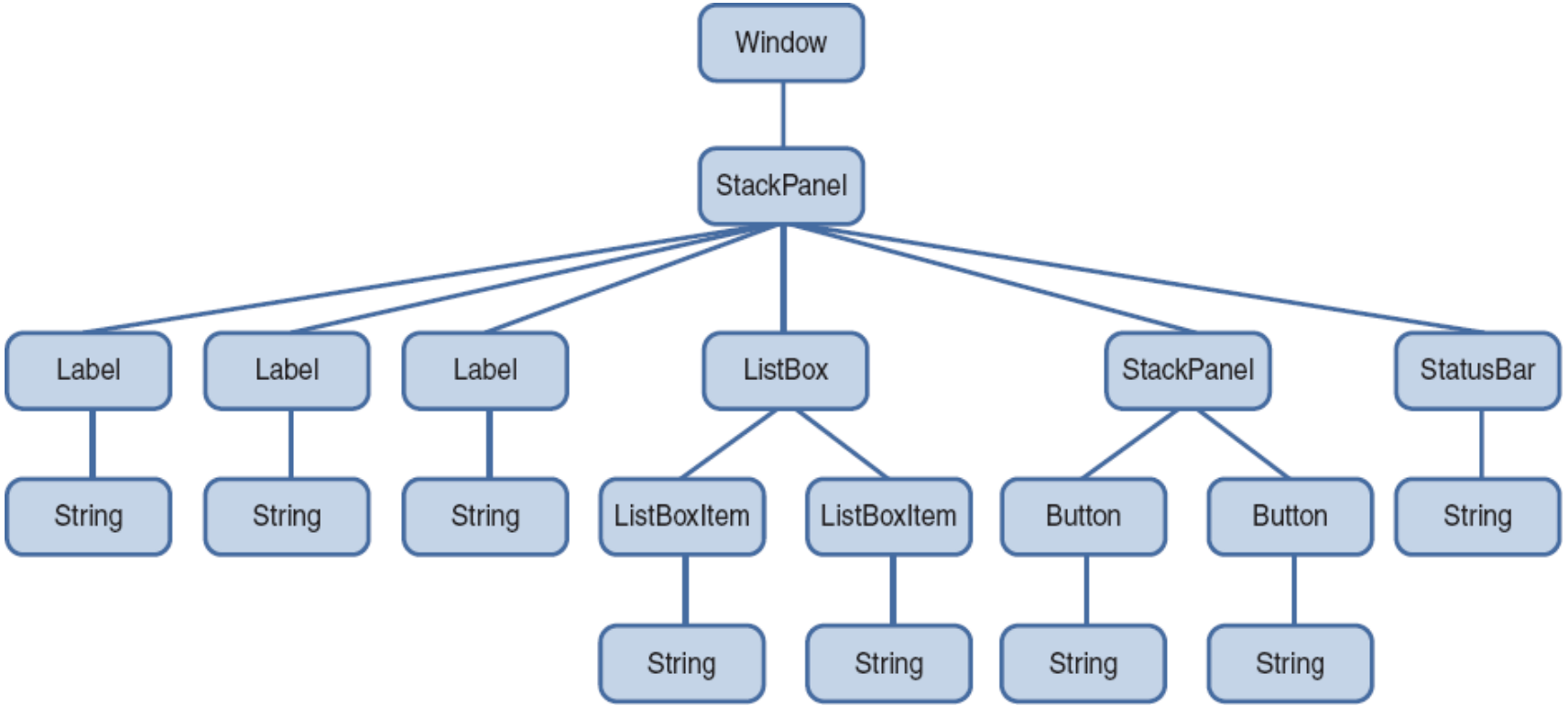


- Ejercicio:
  - Hacer un programa que reproduzca la ventana de la imagen y el comportamiento del botón
  - Ver XAML y code-behind C#
    - Editar XAML con el visor y editor de texto
    - Usar el visor de propiedades

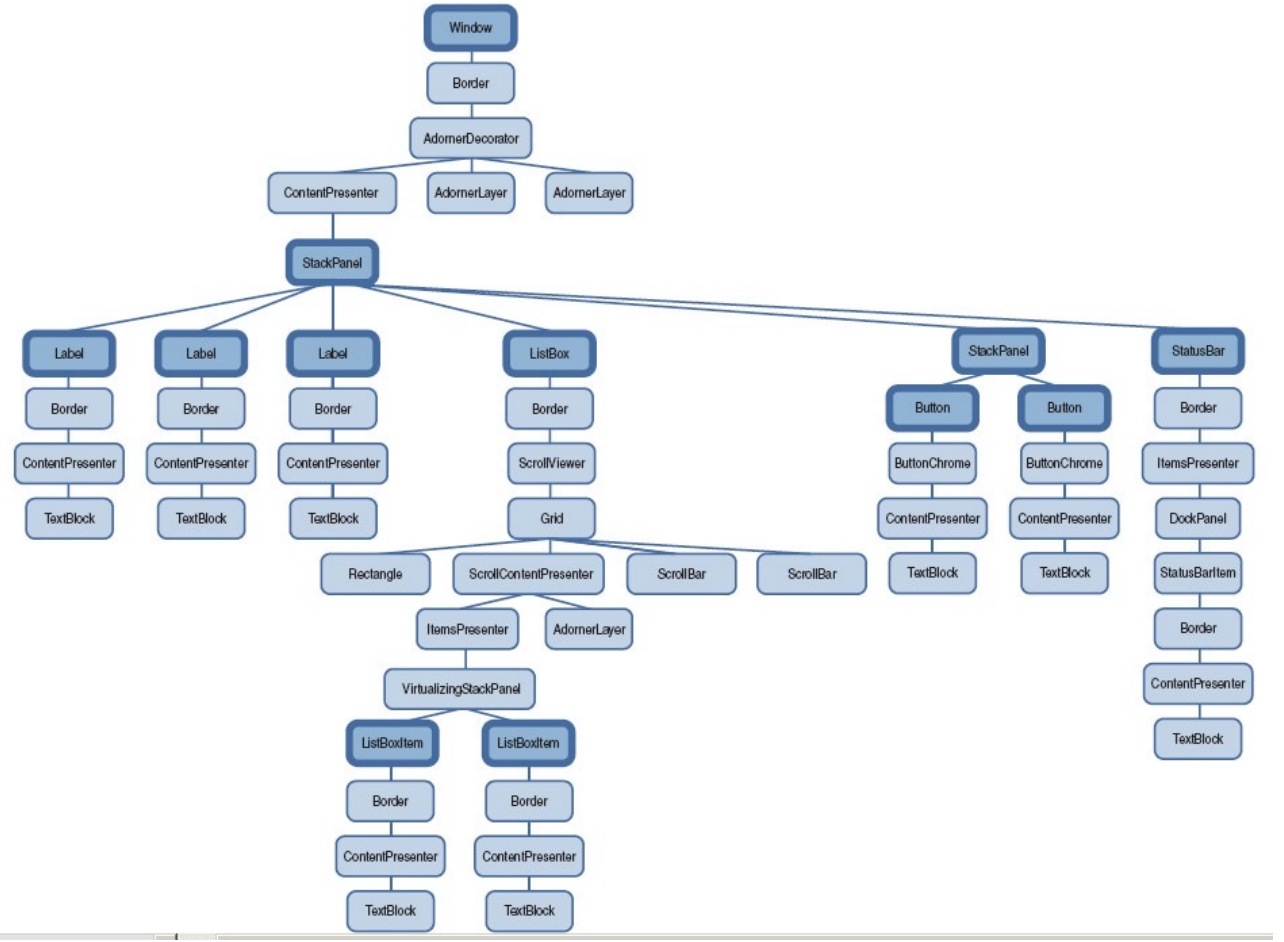
Árbol lógico y árbol visual

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  Title="About WPF Unleashed" SizeToContent='WidthAndHeight"
  Background="OrangeRed">
  <StackPanel>
    <Label FontWeight="Bold" FontSize='20' Foreground='White">
      WPF Unleashed (Version 3.0)
    </Label>
    <Label>© 2006 SAMS Publishing</Label>
    <Label>Installed Chapters:</Label>
    <ListBox>
      <ListBoxItem>Chapter 1</ListBoxItem>
      <ListBoxItem>Chapter 2</ListBoxItem>
    </ListBox>
    <StackPanel Orientation="Horizontal' HorizontalAlignment="Center">
      <Button MinWidth="75" Margin='10'>Help</Button>
      <Button MinWidth="75" Margin='10'>OK</Button>
    </StackPanel>
    <StatusBar>You have successfully registered this product.</StatusBar>
  </StackPanel>
</Window>
```

Árbol lógico



Árbol visual



Árbol lógico y Árbol visual

- El árbol lógico es el que ve el programador
  - Contiene controles y objetos (strings, etc)
- El árbol visual puede ser cambiado (sin afectar al lógico) cambiando la plantilla del Control o el Tema.
  - Lo puede definir el diseñador
  - Contiene los elementos visuales
  - Todos los derivados de FrameworkElement

Árbol lógico y Árbol visual

- ¿Por qué es importante la distinción?
  - El programador sólo debe hacer referencia a los elementos del árbol lógico
    - Así, la apariencia puede ser cambiada libremente
  - Varias características WPF ocurren en el árbol lógico
    - Propagación de eventos
    - Ubicación de Recursos
    - Dependency properties

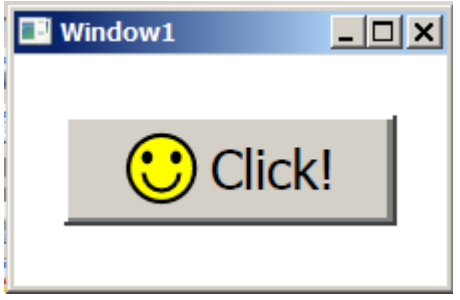
Dependency Properties

- Propiedades de un objeto no almacenadas en el objeto (en una tabla externa)
- Ventajas:
  - Compartición de valores por defecto
  - Menor consumo de memoria
  - Herencia de propiedades por contención
  - Notificación de cambios

Routed events (eventos enrutados)

- WPF utiliza eventos enrutados para superar las limitaciones que tienen los eventos normales (eventos del CLR).
- Un evento enrutado es un evento del CLR que está respaldado por una instancia de la clase RoutedEvent y registrado en el sistema de eventos de WPF. Además, al propagarse entre el elemento que lo desencadena y la raíz del árbol de elementos, puede ser respondido por cualquier elemento que esté en esa ruta, en rugar de por el elemento que lo desencadenó.

- Botón compuesto:




```
<Grid>
    <Button PreviewMouseDown="PreviewMouseDownButton" MouseDow
        <Grid PreviewMouseDown="PreviewMouseDownGrid" MouseDow
            <Grid.ColumnDefinitions>
                <ColumnDefinition />
                <ColumnDefinition />
            </Grid.ColumnDefinitions>
            <Canvas PreviewMouseDown="PreviewMouseDownCanvas"
                <Ellipse PreviewMouseDown="PreviewMouseDownEll
                <Ellipse Canvas.Left="4.5" Canvas.Top="5" Widt
                <Ellipse Canvas.Left="11" Canvas.Top="5" Width
                <Path Data="M 5,10 A 3,3 0 0 0 13,10" Stroke="
            </Canvas>
            <TextBlock Grid.Column="1">Click!</TextBlock>
        </Grid>
    </Button>
</Grid>
```

Si se hace click en el TextBlock se debe tomar como click en el ratón

Se puede considerar los eventos enrutados desde una perspectiva funcional o de implementación:

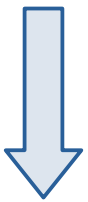
- Desde una perspectiva funcional , un evento enrutado es un tipo de evento que puede invocar controladores en varios agentes de escucha de un árbol de elementos, no solo en el origen del evento. Un agente de escucha de eventos es el elemento donde se adjunta e invoca un controlador de eventos. Un origen de eventos es el elemento o objeto que generó originalmente un evento.
- Desde una perspectiva de implementación , un evento enrutado es un evento registrado con el sistema de eventos de WPF, respaldado por una instancia de la RoutedEvent clase y procesado por el sistema de eventos de WPF. Normalmente, un evento enrutado se implementa con un "contenedor" de evento CLR para habilitar la asociación de controladores en XAML y en código subyacente como lo haría con un evento CLR.

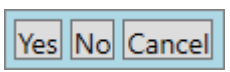
Tipos de enrutamiento

- Propagación
  - Va subiendo por el árbol desde el elemento que lo desencadena hasta la raíz.
- Túnel
  - Si se inicia en la raíz y viaja a un destino situado más abajo en el árbol, teniendo su meta en el elemento que lo desencadenó, pudiendo ser respondido por cualquier elemento en la ruta que tenga un controlador para dicho evento.
- Directo
  - Sólo puede ser respondido por el elemento que lo desencadenó.

Ejemplo 2:

```
<Border Height="30" Width="200" BorderBrush="Gray" BorderThickness="1">
    <StackPanel Background="LightBlue" Orientation="Horizontal" Button.Click="YesNoCancelButton_Click">
        <Button Name="YesButton">Yes</Button>
        <Button Name="NoButton">No</Button>
        <Button Name="CancelButton">Cancel</Button>
    </StackPanel>
</Border>
```





```
private void YesNoCancelButton_Click(object sender, RoutedEventArgs e)
{
    FrameworkElement sourceFrameworkElement = e.Source as
FrameworkElement;
    switch (sourceFrameworkElement.Name)
    {
        case "YesButton":
            // YesButton logic.
            break;
        case "NoButton":
            // NoButton logic.
            break;
        case "CancelButton":
            // CancelButton logic.
            break;
    }
    e.Handled = true;
}
```