

DAIMLER  
EvoBus FORMADORES { IT }



# Curso WPF



Formadores IT

MADRID – BARCELONA – BILBAO – VALENCIA – SEVILLA

info@formadoresfreelance.es  
www.formadoresfreelance.es

## INTRODUCCIÓN

## ¿Qué es WPF?

- WPF es una tecnología para desarrollar la siguiente generación de aplicaciones en Windows y en la web, utilizando toda la potencia del hardware y creando las mejores experiencias de usuario (UX).

© JMA 2012

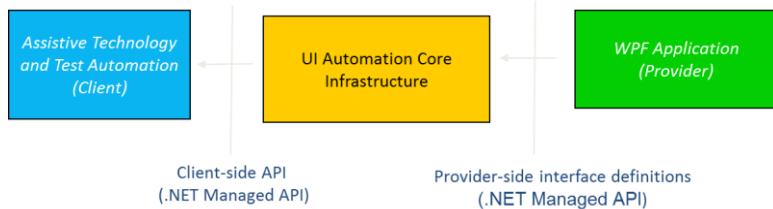
## Windows Presentation Foundation

- **Tecnología IU estratégica de Microsoft**
  - Plataforma y motor
  - Lo mejor del Web y Windows
  - Uso de GPU para alto rendimiento
- **Unificación**
  - Formularios
  - 2D / 3D
  - Video / imágenes
  - Tipografía / Documentos
  - Animaciones
  - Speech
- **Silverlight**
  - Subconjunto multiplataforma
  - Multinavegador



© JMA 2012

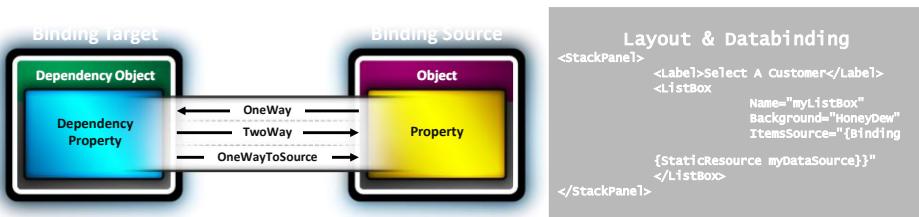
# Accesibilidad



- Mejores prácticas de accesibilidad
  - Habilita la accesibilidad mediante programación, que es apoyado por los controles comunes de WPF a través de la interfaz de usuario de API de automatización
  - Apoyo de acceso mediante teclado, tales como teclas de acceso y el orden lógico de tabulación
  - Soporte de métricas del sistema para la configuración del usuario, tales como un alto contraste y visualización de DPI.
  - Crea un interfaz multi-modal, no depender de efectos visuales para transmitir información por sí sola

© JMA 2012

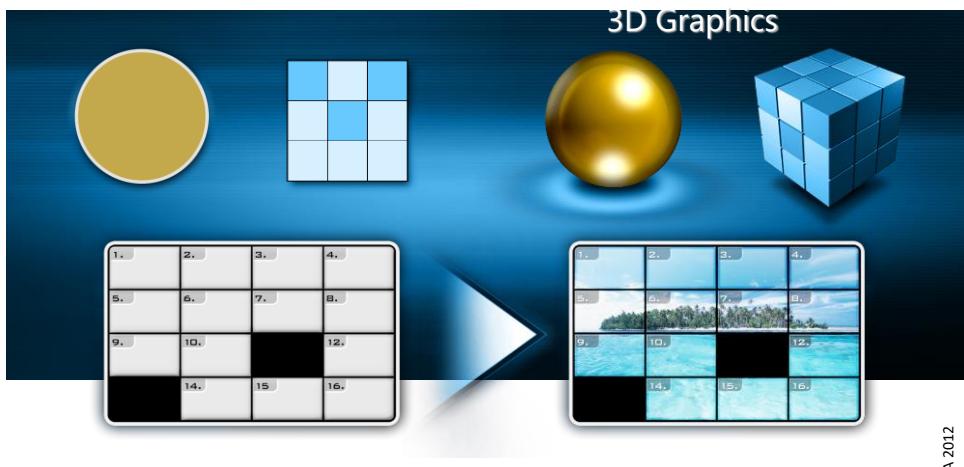
# Data Binding



- La interfaz de usuario se puede enlazar a objetos CLR y XML
- Las propiedades de dependencia también se pueden enlazar a ADO.NET y objetos asociados con los servicios web y las propiedades Web
- La ordenación, filtrado y agrupamiento de vista se pueden generar en la parte superior de los datos
- Plantillas de datos se pueden aplicar a los datos

© JMA 2012

# 2D Graphics, 3D Graphics, Imaging



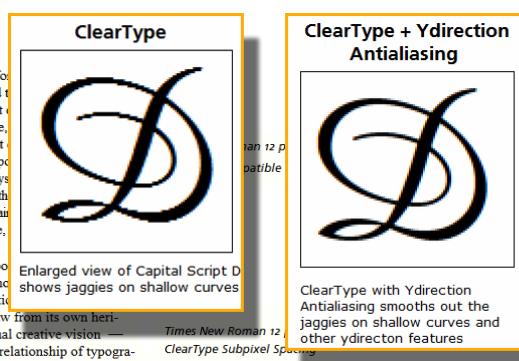
© JMA 2012

## ClearType & Antialiasing

### Sub-pixel positioning & natural widths

I want to conclude this article by suggesting why support for typography belongs among the important things, and not relegated to typography is not to prettify text, but to articulate it. That it is an aesthetic way — utilizing all the art it can draw from its own heritage, script traditions, and individual creative vision — should not disguise the expressive and organizational relationship of typography to text. A typographic culture, such as the one in which you engage as you read this article, is a system of visual indicators that helps readers navigate text and helps writers express their ideas. In the 550 years since Gutenberg developed metal type casting at Mainz, the printed Latin script has developed a particularly rich typographic culture, using romans, italics, bold

I want to conclude this article by suggesting why support for typography belongs among the important things, and not relegated to typography is not to prettify text, but to articulate it. That it is an aesthetic way — utilizing all the art it can draw from its own heritage, the heritage of manuscript tradition, and individual creative vision — should not disguise the expressive and organizational relationship of typography to text. A typographic culture, such as the one in which you engage as you read this article, is a system of visual indicators that helps readers navigate text and helps writers express their ideas. In the 550 years since Gutenberg developed metal type casting at Mainz, the printed Latin script has developed a particularly rich typographic culture, using romans, italics, bold



© JMA 2012

# Nuevas Fonts para WPF

**Calibri**  
**Candara**  
**Cambria**  
**Constantia**  
**Corbel**  
**Consolas**

Consolas

```
<TextPanel ID="root"
    xmlns="http://schemas.microsoft.com/2003/xaml
    xmlns:def="Definition"
    FontFamily="Calibri">
```

© JMA 2012

## Audio & Video



```
<Border Width="400"
    BorderBrush="Green"
    BorderThickness="9">
<StackPanel>
    <MediaElement Source="aero.wmv" />
    <Button>Hello</Button>
</StackPanel>
</Border>
```

- Formatos: WMV, MPEG y algunos AVIs
- Puede ser sincronizado con animaciones
- Windows Media Foundation utiliza para crear instancias de máquinas de reproducción gráficos DirectShow

© JMA 2012

## CPU y aceleración por Hardware



## Nuevas tecnologías de Documentos

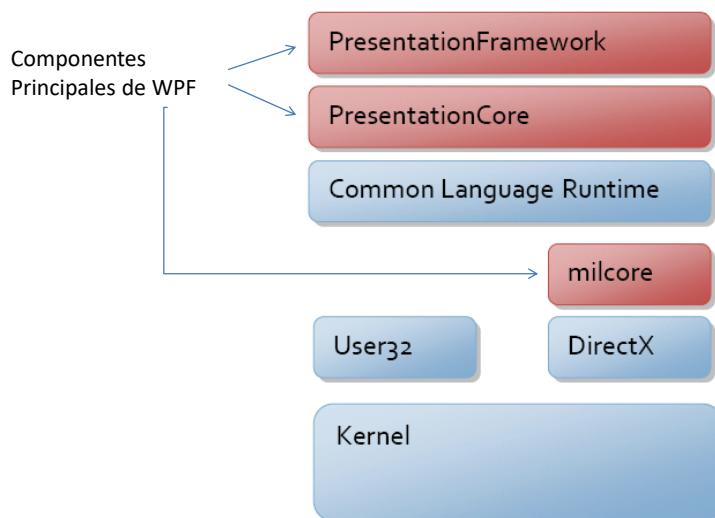


# WPF Avanzado

- Elementos componibles
- Motor de diseño flexible
- Potente arquitectura de enlace de datos
- Capacidades de impresión, fuentes y documentos
- Controles lookless
- Estilos y plantillas
- La plena integración de todas las disciplinas de la interfaz de usuario

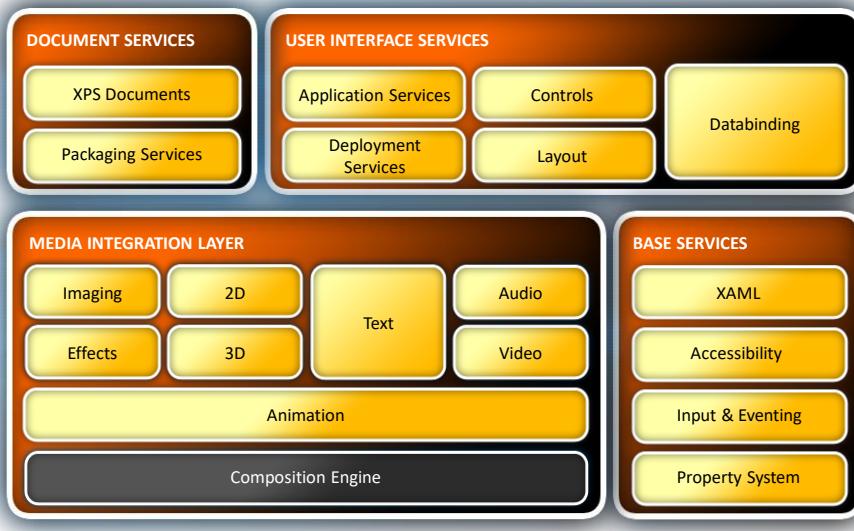
© JMA 2012

## Arquitectura WPF



© JMA 2012

# Arquitectura WPF



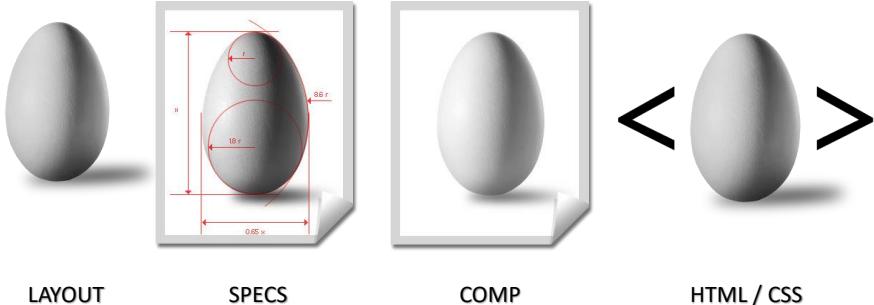
© JMA 2012

## Roles

- Developers
  - Programador
- Designer
  - Diseñador gráfico
- UX Engineer
  - Especialista en usabilidad

© JMA 2012

## Workflow Designer x Developers



© JMA 2012

## Workflow Designer x Developers

PRODUCTION



© JMA 2012

# Workflow Designer x Developers

## FINAL RESULT



© JMA 2012

## Colaboración diseñador-programador



© JMA 2012

# ¿Qué es XAML?

## XAML = Extensive Application Markup Language

- Lenguaje declarativo
- Código y diseño separados
- Fácilmente editable desde herramientas



## XAML

- XAML: Extensible Application Markup Language
- Está basado en XML
- Código mas compacto
- Jerárquico
- Soporte para listas
- Así es como se define la UI de las aplicaciones
  - WPF, Silverlight, Workflow Foundation
- Es un mapeo entre XML y los tipos del .NET Framework
  - Nodos -> Tipos
  - Atributos -> CLR Property o Dependency Property

## Sus ventajas

- Se reducen los costos de programación y mantenimiento
- La programación es más eficaz
- Se pueden usar varias herramientas de diseño para implementar y compartir el marcado XAML
- La globalización y localización de las aplicaciones WPF se ha simplificado en gran medida.

© JMA 2012

## Ejemplo

```
<Canvas  
    xmlns="http://schemas.microsoft.com  
        /client/2007">  
    <TextBlock FontSize="32"  
        Text="Hello world" />  
</Canvas>
```

Hello world

© JMA 2012

## Markup ⇔ CLR

```
<TextBlock FontSize="32"  
          Text="Hello world" />
```

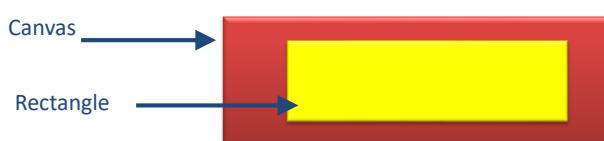
=

```
TextBlock t = new TextBlock();  
t.FontSize = 32;  
t.Text = "Hello world";
```

© JMA 2012

## Composición

```
<Canvas Width="250" Height="200">  
  <Rectangle  
    Canvas.Top="25" Canvas.Left="25"  
    Width="200" Height="150"  
    Fill="Yellow" />  
</Canvas>
```

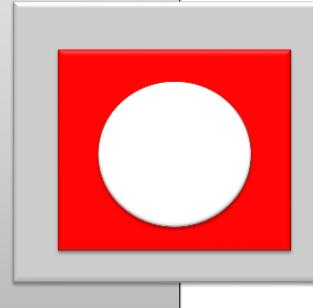


© JMA 2012

## Composición Relativa

```
<Canvas Background="Light Gray">
    <Canvas Canvas.Top="25" Canvas.Left="25"
        Width="150" Height="100"
        Background="Red">

        <Ellipse Canvas.Top="25"
            Canvas.Left="25"
            Width="150"
            Height="75"
            Fill="White" />
    </Canvas>
</Canvas>
```



© JMA 2012

## Transformaciones

- Todos los elementos las soportan
- Tipos
  - <RotateTransform />
  - <ScaleTransform />
  - <SkewTransform />
  - <TranslateTransform />
    - Moves
  - <MatrixTransform />
    - Scale, Skew and Translate Combined

© JMA 2012

## Transformaciones

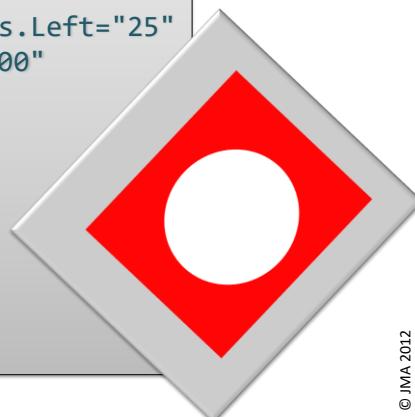
```
<TextBlock Text="Hello World">
    <TextBlock.RenderTransform>
        <RotateTransform Angle="-45" />
    </TextBlock.RenderTransform>
</TextBlock>
```



© JMA 2012

## Composición y Transformación

```
<Canvas Background="Light Gray">
    <Canvas.RenderTransform>
        <RotateTransform Angle="-45" />
    </Canvas.RenderTransform>
    <Canvas Canvas.Top="25" Canvas.Left="25"
        Width="150" Height="100"
        Background="Red">
        <Ellipse Canvas.Top="25"
            Canvas.Left="25"
            Width="150"
            Height="75"
            Fill="White" />
    </Canvas>
</Canvas>
```



© JMA 2012

# XAML

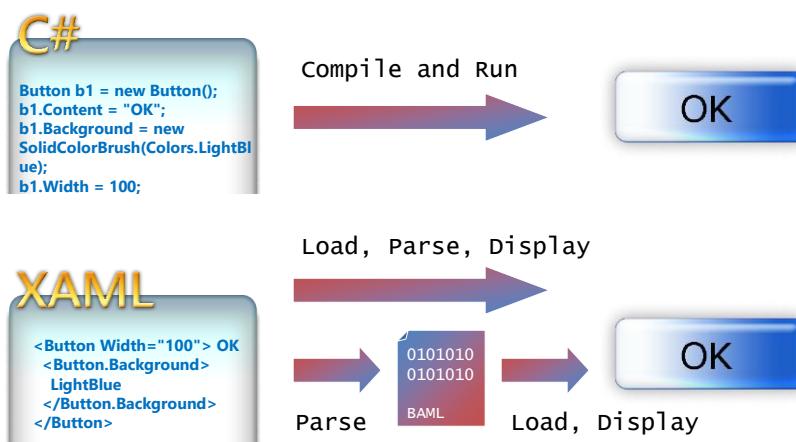
```

10   <!-- XAML basics -->
11
12   <!-- TextBox element maps to System.Windows.Controls.TextBox class
13   Text (Property Attribute) maps to Text property -->
14   <TextBox Text='WPF Demystified' />
15
16   <!-- MouseEnter (Event Attribute) maps to the MouseEnter event.
17       requires a C# or VB code file with MouseEnter procedure. -->
18   <TextBox MouseEnter='TextBox_MouseEnter' />
19
20   <!-- Property Elements are another way to specify property value -->
21   <TextBox>
22     <TextBox.Text>Another example</TextBox.Text>
23   </TextBox>
24
25   <!-- Databinding, Styles, Templates are some of the features
26       enabled with Markup Extensions -->
27
28   <TextBox Text='{Binding AuthorName}' />
29

```

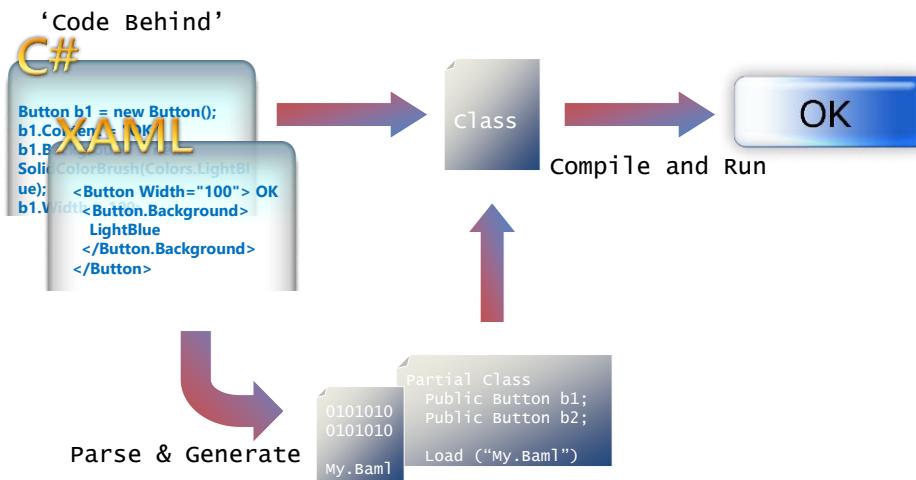
© JMA 2012

## ¿XAML o Código?



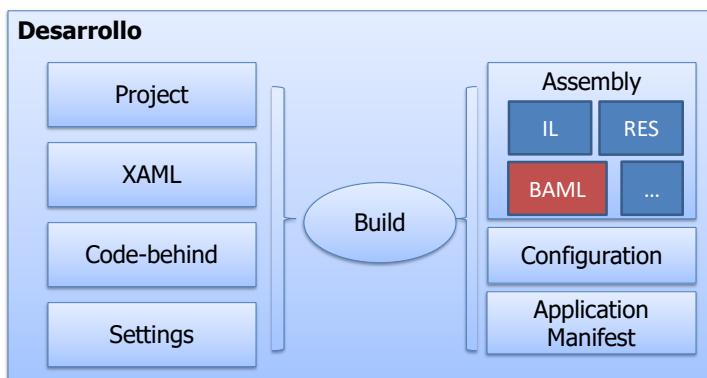
© JMA 2012

## ¿XAML o Código?



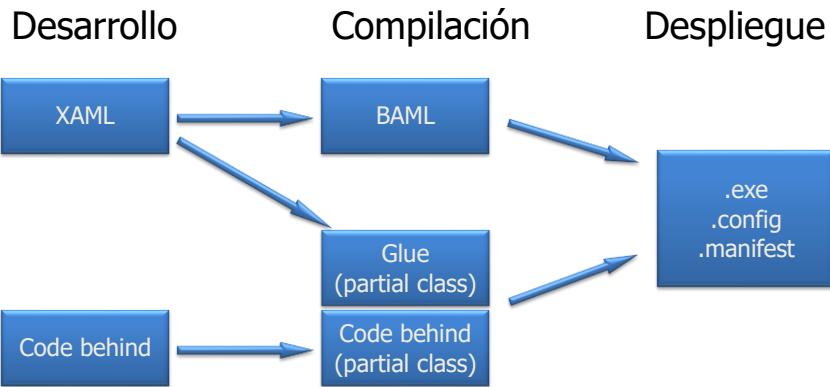
© JMA 2012

## Anatomía de una aplicación WPF



© JMA 2012

## Ciclo de vida



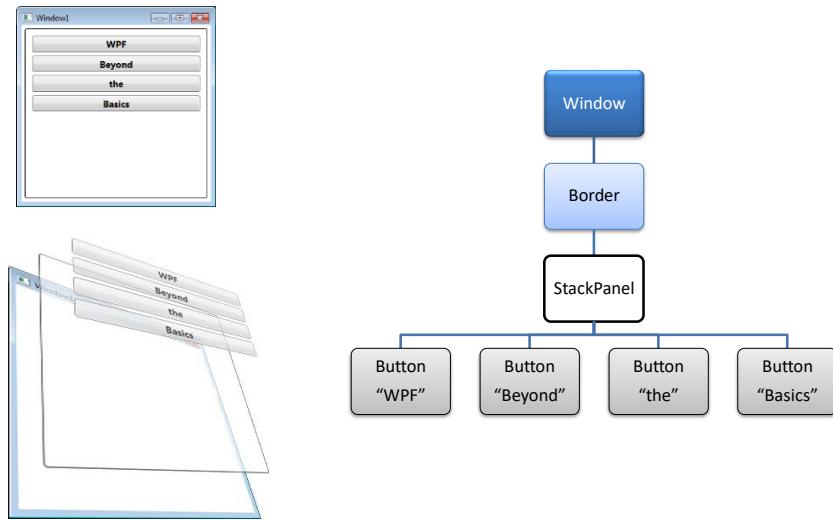
© JMA 2012

## Ejecución

- Arboles lógicos, visuales y composición
- Rendering basado en capacidades
  - ‘Hardware’ o ‘software’
  - Puede ser detectado usando RenderCapability
- Controla todo el espacio aéreo visual

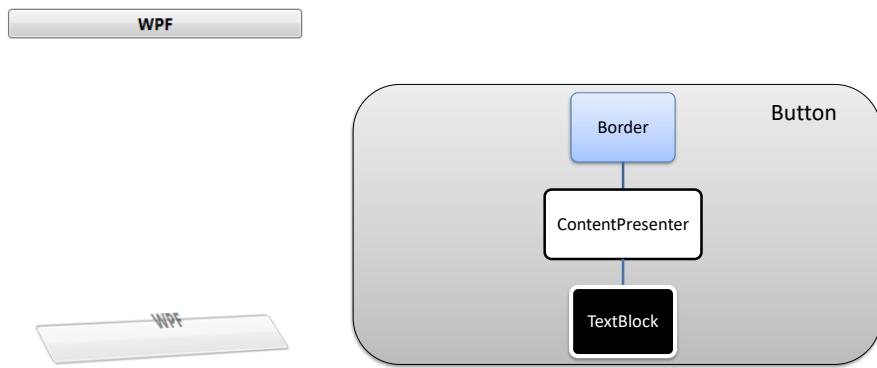
© JMA 2012

## Árbol Lógico



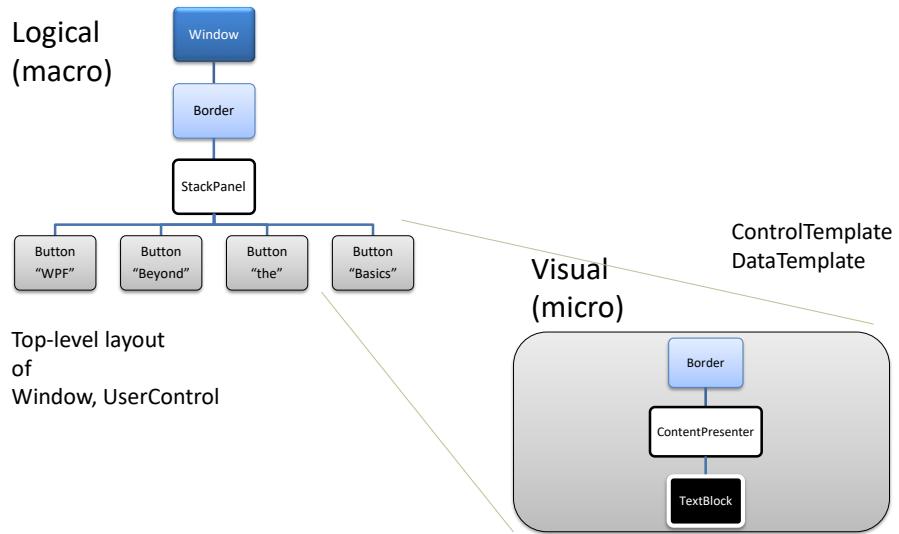
© JMA 2012

## Árbol Visual

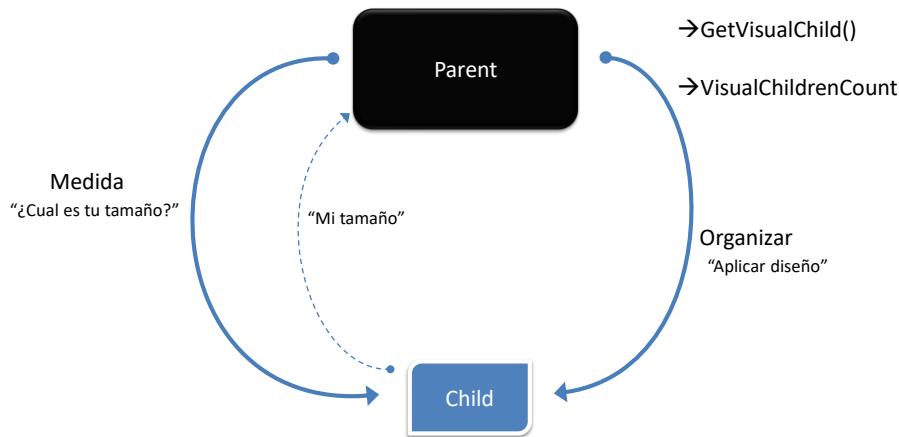


© JMA 2012

## Logical + Visual

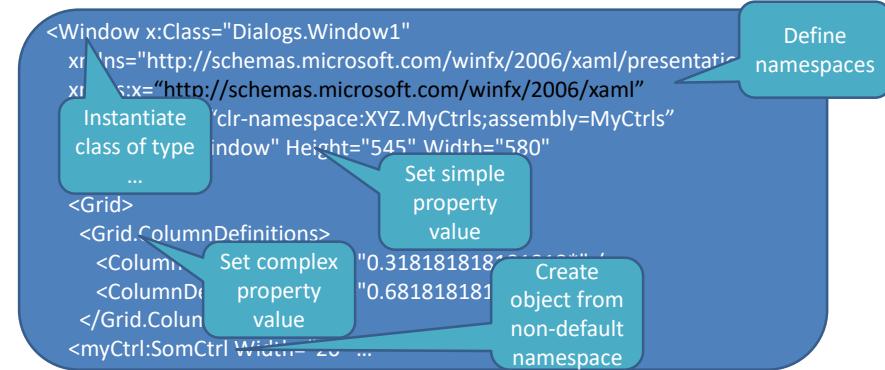


## Layout



# XAML Dinámico

- Se puede leer y guardar XAML desde código
  - XamlReader.Load, XamlReader.Parse
  - XamlWriter.Save



© JMA 2012

## Sintaxis XAML

- Documentos XML bien formados y validos
- Vocabularios propios
- Vocabulario ampliable a través de los espacios de nombres
- Soporte de extensiones de marcado
- Conversión implícita de cadena al tipo de destino:
  - Valor de propiedad
  - Nombre del controlador de eventos

© JMA 2012

# Espacios de nombres

- Básicos:

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
mc:Ignorable="d"
```

- Para utilizar otros tipos:

```
xmlns:Prefix="clr-namespace:Namespace;assembly=AssemblyName"
```

© JMA 2012

# Propiedades y eventos

- Son los atributos de la etiqueta
- Propiedades Simples  
`<nombreDeTipo ... propiedad="valor" ... />`
- Propiedades Complejas/Colecciones  
`<nombreDeTipo ... >`  
 `<nombreDeTipo.propiedad>`  
 `...`  
 `<nombreDeTipo.propiedad>`  
 `...`  
`</nombreDeTipo>`
- Propiedades Asociadas  
`<nombreDeTipo ... padre.propiedad="valor" ... />`  
`<nombreDeTipo ... hijos.propiedad="valor" ... />`
- Eventos  
`<nombreDeTipo ... evento="NombreControlador" ... />`

© JMA 2012

# Extensiones de enlazado

- Las llaves {} y } indican el uso de una extensión de marcado que se aparta del tratamiento general de valores de atributo.
  - **Binding**: proporciona un valor enlazado a datos para una propiedad, utilizando el contexto de datos que se aplica al objeto primario en tiempo de ejecución.
  - **TemplateBinding**: permite que una plantilla de control utilice valores para propiedades con plantilla procedentes de propiedades definidas por el modelo de objetos de la clase que utilizará la plantilla.
  - **RelativeSource**: proporciona información de origen para un objeto Binding que puede navegar por varias posibles relaciones en el árbol de objetos en tiempo de ejecución.
  - **StaticResource**: proporciona un valor para una propiedad sustituyendo el valor de un recurso ya definido.
  - **DynamicResource**: proporciona un valor para una propiedad aplazando ese valor para que sea una referencia a un recurso en tiempo de ejecución.
  - **ColorConvertedBitmap**: admite un escenario de creación de imágenes relativamente avanzado.
  - **ComponentResourceKey** y **ThemeDictionary**: admiten aspectos de la búsqueda de recursos y temas que se empaquetan con controles personalizados.

© JMA 2012

# Extensiones de marcado

- **x:name**
  - Nombre de referencia disponible para todas las etiquetas
  - Utilizado como nombre de las instancias CLR generadas
  - Búsquedas en el árbol: FindName("control")
- **x:key**
  - Identifica de forma exclusiva los elementos que se crean y a los que se hace referencia en un diccionario de recursos.
- **x:class**
  - Enlace al código subyacente
- **x:type**
  - Construye una referencia Type basada en un nombre de tipo.
- **{x:Static prefix:typeName.staticMemberName}**
  - Referencia a cualquier entidad de código estática por valor definida conforme a Common Language Specification (CLS).
- **{x:Null}**
  - Especifica null como valor para una propiedad.

© JMA 2012

# Extensiones de Design-Time

- d:DesignHeight and d:DesignWidth
  - d:DesignHeight="300" d:DesignWidth="400"
- d:DataContext
  - d:DataContext="{d:DesignInstance Type=local:Customer}">
- d:DesignInstance and d:IsDesignTimeCreatable
  - <Grid d:DataContext="{d:DesignInstance local:Customer, IsDesignTimeCreatable=True}">
- d:DesignData
  - d:DataContext="{d:DesignData Source=./DesignData/SampleCustomer.xaml}">
- d:DesignSource d:CreateList
  - <CollectionViewSource x:Key="CustomerViewSource"
  - d:DesignSource="{d:DesignInstance local:Customer, CreateList=True}" />
- d>Type

© JMA 2012

# XML y XAML

- Caracteres especiales
  - Less than (<) &lt;
  - Greater than (>) &gt;
  - Ampersand (&) &amp;
  - Quotation mark ("") &quot;
- No preserva el espacio en blanco (sp, br, tab, ...)
- Para preservar el espacio en blanco:
 

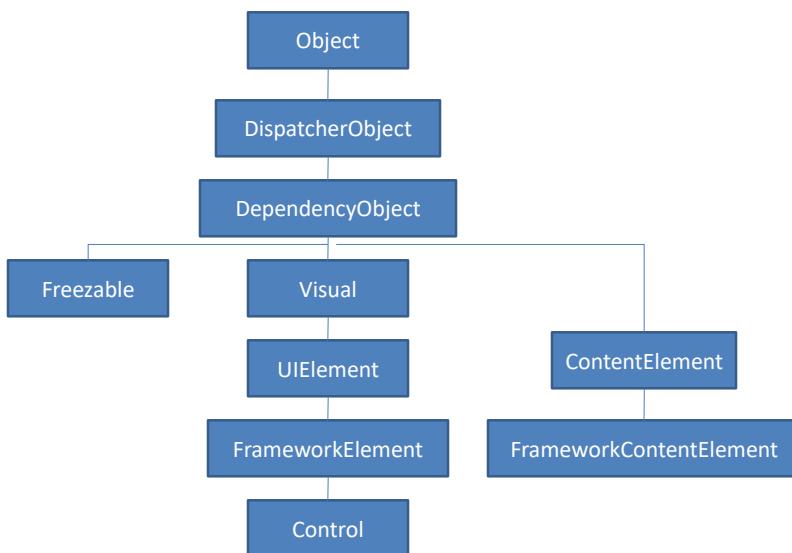
```
<nombredetipo xml:space="preserve" ... >
...
</nombredetipo>
```

© JMA 2012

# DISEÑO DE APLICACIONES Y FORMULARIOS

© JMA 2012

## Core classes



© JMA 2012

# Modelo de contenido

- Clases que incluyen contenido arbitrario
- Clases que contienen una colección de objetos UIElement
- Clases que afectan a la apariencia de un objeto UIElement
- Clases que proporcionan comentarios visuales sobre un objeto UIElement
- Clases que permiten a los usuarios introducir texto
- Clases que muestran su texto
- Clases que dan formato al texto

© JMA 2012

## Clases que incluyen contenido arbitrario

- Algunos controles pueden contener un objeto de cualquier tipo, como una cadena, un objeto DateTime o un UIElement que es un contenedor para elementos adicionales. Por ejemplo, un Button puede contener una imagen y algún texto; o CheckBox puede contener el valor de DateTime.Now.
- WPF tiene cuatro clases que pueden incluir contenido arbitrario y heredan de Control.
  - ContentControl: Un único objeto arbitrario.
  - HeaderedContentControl: Un encabezado y un único elemento; los dos son objetos arbitrarios.
  - ItemsControl: Una colección de objetos arbitrarios.
  - HeaderedItemsControl: Un encabezado y una colección de elementos; todos son objetos arbitrarios.

© JMA 2012

## Controles que contienen un único objeto arbitrario

- La clase ContentControl contiene una parte única de contenido arbitrario, la propiedad Content o el contenido de la etiqueta.

Button, ButtonBase, CheckBox, ComboBoxItem, ContentControl, Frame, GridViewColumnHeader, GroupItem, Label, ListItem, ListViewItem, NavigationWindow, RadioButton, RepeatButton, ScrollViewer, StatusBarItem, ToggleButton, ToolTip, UserControl, Window

© JMA 2012

## Controles que contienen un encabezado y contenido

- La clase HeaderedContentControl hereda de ContentControl y muestra contenido con un encabezado.
- Hereda la propiedad de contenido, Content, de ContentControl y define la propiedad Header que es del tipo Object; en consecuencia, ambos pueden ser un objeto arbitrario.
  - Expander
  - GroupBox
  - TabItem

© JMA 2012

## Controles que contienen una colección de objetos arbitrarios

- La clase `ItemsControl` hereda de `Control` y puede contener varios elementos, como cadenas, objetos u otros elementos.
- Sus propiedades de contenido son `ItemsSource` y `Items`. `ItemsSource` se suele usar para llenar `ItemsControl` con una recolección de datos.
- Si no desea emplear una colección para llenar el `ItemsControl`, puede agregar elementos mediante la propiedad `Items`.
  - `Menu`, `MenuBase`, `ContextMenu`
  - `ComboBox`
  - `ListBox`
  - `ListView`
  - `TabControl`
  - `TreeView`
  - `Selector`
  - `StatusBar`

© JMA 2012

## Clases que contienen una colección de objetos `UIElement`

- La clase `Panel` coloca y organiza los objetos `UIElement` secundarios. Su propiedad de contenido es `Children`.
  - `Canvas`
  - `DockPanel`
  - `Grid`
  - `TabPanel`
  - `ToolBarOverflowPanel`
  - `ToolBarPanel`
  - `UniformGrid`
  - `StackPanel`
  - `VirtualizingPanel`
  - `VirtualizingStackPanel`
  - `WrapPanel`

© JMA 2012

## Clases que afectan a la apariencia de un objeto UIElement

- La clase **Decorator** aplica efectos visuales a un único objeto **UIElement** secundario o alrededor del mismo. Su propiedad de contenido es **Child**.
  - **AdornerDecorator**
  - **Border**
  - **BulletDecorator**
  - **ButtonChrome**
  - **ClassicBorderDecorator**
  - **InkPresenter**
  - **ListBoxChrome**
  - **SystemDropShadowChrome**
  - **Viewbox**
- Clases que proporcionan comentarios visuales sobre un objeto **UIElement**
  - La clase **Adorner** proporciona indicaciones visuales a un usuario.

© JMA 2012

## Clases que contienen texto

- Clases que muestran su texto
  - Se pueden usar varias clases para mostrar texto sin formato o con formato.
  - Puede emplear **TextBlock** para mostrar cantidades pequeñas de texto.
  - Si desea mostrar grandes cantidades de texto, use los controles **FlowDocumentReader**, **FlowDocumentScrollView** o **FlowDocumentPageViewer**.
- Clases que permiten a los usuarios introducir texto
  - WPF proporciona tres controles primarios que permiten a los usuarios introducir texto. Cada control muestra el texto de manera diferente.
  - **TextBox**: Texto sin formato
  - **RichTextBox**: Texto con formato
  - **PasswordBox**: Texto oculto (se enmascaran los caracteres)
- Clases que dan formato al texto
  - **TextElement** y sus clases relacionadas le permiten dar formato al texto de los objetos **FlowDocument** y **TextBlock**.

© JMA 2012

# Diseño

- **Canvas:**
  - los controles secundarios proporcionan su propio diseño.
- **DockPanel:**
  - los controles secundarios se alinean con los bordes del panel.
- **Grid:**
  - los controles secundarios se sitúan por filas y columnas.
- **StackPanel:**
  - los controles secundarios se apilan vertical u horizontalmente.
- **VirtualizingStackPanel:**
  - los controles secundarios se organizan en una vista "virtual" de una sola línea en sentido horizontal o vertical.
- **WrapPanel:**
  - los controles secundarios se sitúan por orden de izquierda a derecha y se ajustan a la línea siguiente cuando hay más controles de los que caben en la línea actual.

© JMA 2012

# Propiedades asociadas

- Uno de los propósitos de una propiedad asociada es permitir que los diferentes elementos secundarios especifiquen valores únicos para una propiedad que en realidad está definida en un elemento primario.
- Una aplicación concreta de este escenario es hacer que los elementos secundarios informen al elemento primario de cómo se presentarán en la user interface (UI).
- La propiedad DockPanel.Dock se crea como una propiedad asociada porque se ha diseñado para establecerse en elementos contenidos dentro de un objeto DockPanel, en lugar de en el propio objeto DockPanel. La clase DockPanel define el campo estático DependencyProperty denominado DockProperty y, a continuación, proporciona los métodos GetDock y SetDock como descriptores de acceso públicos para la propiedad asociada.

© JMA 2012

## Canvas

- El elemento Canvas permite la colocación del contenido de acuerdo con unas coordenadas x e y absolutas.
- Los elementos se pueden dibujar en una ubicación única; o, si ocupan las mismas coordenadas, el orden en que aparecen en el marcado determina el orden en el que se dibujan.
- Canvas proporciona la compatibilidad de diseño más flexible de todos los elementos Panel.
- Las propiedades Height y Width se utilizan para definir el área del lienzo, y a los elementos que contiene se les asignan coordenadas absolutas relativas al área del elemento Canvas primario.
- Cuatro propiedades adjuntas, Canvas.Left, Canvas.Top, Canvas.Right y Canvas.Bottom, permiten obtener un control muy preciso de la posición de los objetos dentro de un elemento Canvas, y gracias a ellas el programador puede colocar y organizar con precisión los elementos en la pantalla.

© JMA 2012

## DockPanel

- El elemento DockPanel utiliza la propiedad adjunta DockPanel.Dock tal y como está establecida en los elementos de contenido secundarios para colocar el contenido a lo largo de los bordes de un contenedor.
- Cuando DockPanel.Dock se establece en Top o Bottom, coloca los elementos secundarios por encima o por debajo de los demás.
- Cuando la propiedad DockPanel.Dock está establecida en Left o en Right, coloca los elementos secundarios a la izquierda o a la derecha de los demás.
- La propiedad LastChildFill determina la posición del último elemento agregado como un elemento secundario de un elemento DockPanel.
- Puede utilizar DockPanel para colocar un grupo de controles relacionados, como un conjunto de botones. También puede utilizarlo para crear una UI "con paneles", similar a la de Microsoft Outlook.

© JMA 2012

# Grid

- Define un área de cuadrícula flexible que está compuesta de columnas y filas.
- El elemento Grid combina la funcionalidad de una posición absoluta y de un control de datos tabular.
- Grid permite colocar con facilidad elementos y aplicarles estilo.
- Con Grid podrá definir agrupaciones flexibles de filas y columnas, e incluso dispone de un mecanismo para compartir información de tamaño entre varios elementos Grid.
- Grid agrega elementos basándose en un índice de fila y columna.
- Permite la disposición en capas del contenido secundario, lo que permite que haya más de un elemento dentro de una sola "celda".
- Los elementos secundarios de Grid se pueden colocar de manera absoluta en relación con el área de los límites de la "celda". Grid.Column, Grid.ColumnSpan, Grid.Row y Grid.RowSpan permiten indicar la celda y el número de celdas ocupadas.
- RowDefinitions y ColumnDefinitions controlan el número y tamaño de las filas y columnas.

© JMA 2012

# StackPanel

- Un elemento StackPanel le permite "apilar" los elementos en una dirección asignada.
- De forma predeterminada, los elementos se apilan en dirección vertical.
- Se puede utilizar la propiedad Orientation para controlar el flujo del contenido en dirección horizontal.

© JMA 2012

## WrapPanel

- WrapPanel se utiliza para colocar de izquierda a derecha los elementos secundarios en posición secuencial, y traslada el contenido a la línea siguiente cuando alcanza el borde de su contenedor primario.
- El contenido se puede orientar en sentido horizontal o vertical. WrapPanel resulta útil para los escenarios de user interface (UI) de flujo sencillo.
- También se puede utilizar para aplicar un tamaño uniforme a todos sus elementos secundarios.

© JMA 2012

## Alineación, Márgenes y Relleno

- Los valores explícitos de las propiedades Width y Height tienen prioridad al fijar el ancho y alto de los elementos.
- Las propiedades HorizontalAlignment (Left, Right, Center o Stretch) y VerticalAlignment (Top, Center, Bottom o Stretch) describen cómo debería colocarse un elemento secundario dentro del espacio de diseño asignado del elemento primario.
- La propiedad Margin describe la distancia entre un elemento y su elemento secundario o del mismo nivel.
- La propiedad Padding describe la distancia entre un elemento y su contenido.
- Las propiedades MaxWidth, MinWidth, MaxHeight y MinHeight fijan el ancho y alto máximo y mínimo de los elementos.

© JMA 2012

# Controles de WPF por función

- **Presentación y selección de fechas:**
  - Calendar y DatePicker.
- **Presentación de datos:**
  - DataGrid, ListView y TreeView.
- **Cuadros de diálogo:**
  - OpenFileDialog, PrintDialog y SaveFileDialog.
- **Información para el usuario:**
  - AccessText, Label, Popup, ProgressBar, StatusBar, TextBlock y ToolTip.
- **Documentos:**
  - DocumentViewer, FlowDocumentPageViewer, FlowDocumentReader, FlowDocumentScrollView y StickyNoteControl.
- **Entradas de lápiz digitales:**
  - InkCanvas y InkPresenter.
- **Multimedia:**
  - Image, MediaElement y SoundPlayerAction.

© JMA 2012

# Controles de WPF por función

- **Diseño:**
  - Border, BulletDecorator, Canvas, DockPanel, Expander, Grid, GridView, GridSplitter, GroupBox, Panel, ResizeGrip, Separator, ScrollBar, ScrollViewer, StackPanel, Thumb, Viewbox, VirtualizingStackPanel, Window y WrapPanel.
- **Navegación:**
  - Frame, Hyperlink, Page, NavigationWindow y TabControl.
- **Botones:**
  - Button y RepeatButton.
- **Menús:**
  - ContextMenu, Menu yToolBar.
- **Entrada:**
  - TextBox, RichTextBox y PasswordBox.
- **Selección:**
  - CheckBox, ComboBox, ListBox, RadioButton y Slider.

© JMA 2012

## Ribbon (4.5)

- Requiere agregar manualmente la referencia System.Windows.Controls.Ribbon
- La cinta de opciones es una barra de comandos que organiza las características de una aplicación en una serie de fichas en la parte superior de la ventana de la aplicación.
- La cinta reemplaza la barra de menús y las barras de herramientas tradicionales (algunos componentes de la cinta se dibujan en la barra de título de la ventana por lo que hay que heredar de RibbonWindow en lugar de Window).
- Cada cinta tiene un menú de aplicación, una barra de herramientas de acceso rápido, fichas, grupos y controles.
- Las fichas de cinta contienen grupos y cada grupo contiene controles. Los controles relacionados se pueden combinar aún más en grupos de controles.
- Los controles de cinta incluyen controles sencillos como botones, casillas y cuadros de texto; y controles de menú como cuadros combinados, botones de expansión y botones de menú.
- Además de los componentes necesarios, una cinta también puede incluir componentes opcionales, como fichas contextuales, información sobre herramientas mejorada y galerías.

© JMA 2012

## Despliegue WPF

- Ensamblado .NET
  - Ejecutable tradicional Setup, ClickOnce
- Aplicación XBAP
  - Dentro del navegador
  - Modelo de navegación integrado con browser
- Loose XAML
  - Renderización directa en browser
  - Opciones interesantes: ASP.NET / XML + XSL
- Documento
  - Formato de documento XPS = Subset XAML

© JMA 2012

# Modelos de GUI

- SDI: interfaz de documento único
  - Ventana principal (Window):
    - Controles (UserControl)
    - Ventanas emergentes (PopUP)
- MDI: interfaz de múltiples documentos
  - Soportado por herramientas de terceros.
- Navegación: tipo web
  - Ventana de navegación (NavigationWindow).
    - Páginas de contenido (Page) e hipervínculos (Hyperlink)
    - Marcos (Frame) y motor de navegación (NavigationService)

© JMA 2012

## Ventanas secundarias

- Modeless, que no se cierran con la principal  
`Window2 win = new Window2();  
win.Show();`
- Modeless, que se cierran con la principal  
`win = new Window2();  
win.Owner = this;  
win.Show();`
- Modal (Cuadro de diálogo)  
`Window2 win = new Window2();  
win.Owner = this;  
if(win.ShowDialog() == true) // Aceptado (bool?)`

© JMA 2012

## Cuadros de diálogo

- Heredan de la clase Window
- Configurar el aspecto de la ventana:
  - Mostrar Barra de título, ícono y menú Sistema para minimizar, maximizar, restaurar y cerrar el cuadro de diálogo.
  - Mostrar botones Cerrar, Minimizar, Maximizar y Restaurar
  - ResizeMode="NoResize"  
WindowStartupLocation="CenterScreen"  
ShowInTaskbar="False"
- Mostrar un botón Aceptar (IsDefault='true') y un botón Cancelar (IsCancel='true').
- Para Aceptar (this.DialogResult = true) o Cancelar (this.DialogResult = false).

© JMA 2012

## Control de usuario

- La manera más sencilla de crear un control es crear una clase que deriva de UserControl.
- Es un modelo conveniente si desea generar el control agregando elementos existentes en él, similar a cómo crear una ventana, y si no necesita utilizar personalización compleja.
- El control no admitirá plantillas y, por tanto, no admite la personalización compleja.
- Un UserControl es un ContentControl, lo que significa que puede contener un único objeto de cualquier tipo (como una cadena, una imagen o un panel).
- Contenedor de controles:  
`<ContentControl x:Name="UCHost" />`
- Abrir control como componente:  
`UCHost.Content = new UserControl1();`
- Cerrar el control componente:  
`UCHost.Content = null;`

© JMA 2012

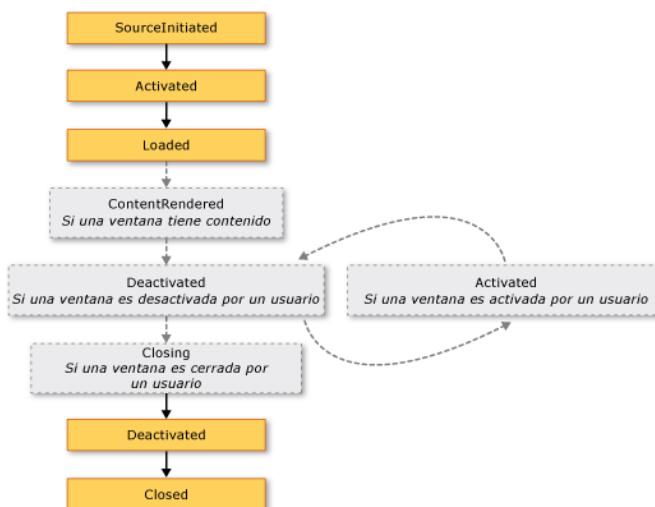
# Popup

- El control Popup proporciona una manera de mostrar contenido en una ventana independiente que flota sobre la ventana de la aplicación actual de manera relativa a un elemento designado o a una coordenada de la pantalla.
- Un control Popup muestra su contenido cuando la propiedad IsOpen está establecida en true y permanece abierto hasta que la propiedad IsOpen se establece en false.
- Sin embargo, puede cambiar el comportamiento predeterminado estableciendo la propiedad StaysOpen en false, la ventana se cierra cuando un evento del mouse se produce fuera de la ventana de Popup.

```
<Popup x:Name="popDlg" StaysOpen="False" PopupAnimation="Slide" Placement="Right"
PlacementTarget="{Binding ElementName=btnAbrirPOP, Mode=OneWay}"
AllowsTransparency="True" Width="200" Height="100">
    <Border CornerRadius="10" Background="#FFFDFDD4" Padding="5">
        <Grid>
            <Button x:Name="btnCerrarPOP" Content="Cerrar"
HorizontalContentAlignment="Right" VerticalAlignment="Bottom" HorizontalAlignment="Right"
Click="btnCerrarPOP_Click"/>
        </Grid>
    </Border>
</Popup>
```

© JMA 2012

## Eventos de duración de ventana



© JMA 2012

# Modelo de Navegación

- Page: Encapsula una página de contenido a la que pueda navegar, se diseña de forma similar a las ventanas.
- Hyperlink: Permite que un usuario inicie la navegación en un objeto Page determinado.
- NavigationService: Encargado de la búsqueda y descarga de la página.
- Diario (Journal): Implementa un servicio del historial de navegación que almacena una entrada para cada fragmento de contenido al que se ha navegado previamente.
- NavigationWindow: Proporciona una ventana principal para aplicaciones independientes de navegación.
- Frame: Proporciona un contenedor de páginas para su uso en ventanas u otras páginas.

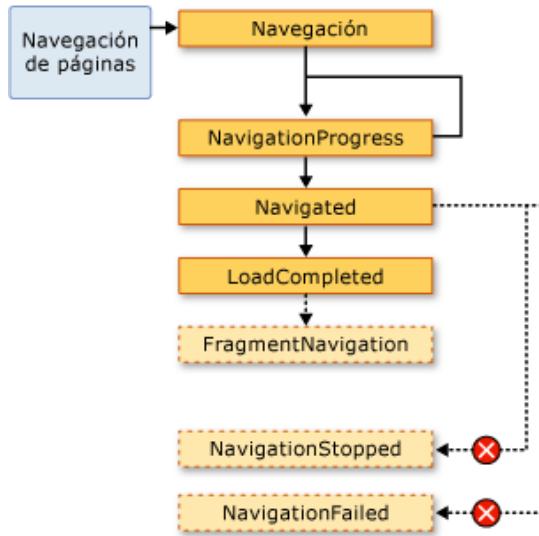
© JMA 2012

## Navegación

- Navegación por hipervínculos  
`<Hyperlink NavigateUri="pag2.xaml">...</Hyperlink>`
- Navegación por fragmentos  
`<TextBlock Name="Fragmento1">...</TextBlock>`  
`<Hyperlink  
NavigateUri="pag2.xaml#Fragmento1">...</Hyperlink>`
- Navegación por código :  
`ns = NavigationService.GetNavigationService(refPage);`  
`ns = this.NavigationService;`  
`ns.Navigate(new Page2());`  
`ns.Navigate(new Uri("Page2.xaml", UriKind.Relative));`

© JMA 2012

## Duración de la navegación



© JMA 2012

## Navegación con el diario

- Dos pilas con el historial de las páginas: una pila de retroceso y una pila de avance.
- Barra de navegación
- Declarativa por comandos (NavigationCommands):  

```
<Hyperlink Command="NavigationCommands.BrowseBack" ... >
<Hyperlink Command="NavigationCommands.BrowseForward"
... >
```
- Imperativa por código (NavigationService):
  - GoBack, GoForward
  - CanGoBack, CanGoForward

© JMA 2012

## Estado de la página

- Activación:  
`<Page ... KeepAlive="True" ...>`
- Solo se crean una vez, hay que mover la lógica de inicialización a la que hay que llamar cada vez que se navega a la página al controlador del evento Loaded.
- Controles que mantienen el estado:
  - CheckBox, ComboBox , Expander, Frame, ListBox, ListBoxItem, MenuItem, ProgressBar, RadioButton, Slider, TabControl, TabItem, TextBox
- Para que recuerde información adicional:
  - Crear propiedades de dependencia con el marcador de metadatos de FrameworkPropertyMetadata.Journal=true.
  - Implementar el interfaz IProvideCustomContentState.

© JMA 2012

## Marcos

`<Frame Source="FramePage1.xaml" ... />`

- Dispone de su propio contenido, NavigationService, diario y barra de navegación.
- JournalOwnership=
  - OwnsJournal (Diario propio)
  - UsesParentJournal (Diario del contenedor)
- NavigationUIVisibility="Hidden"

© JMA 2012

# Navegación estructurada

- Se denomina navegación estructurada cuando se necesita:
  - Pasar parámetros a la página llamada.
  - Devolver información de estado o datos a la página que llama.
  - Quitar del historial de navegación la página llamada cuando vuelve a la página que llama.
- `PageFunction<T>`, hereda de `Page` y representa un tipo especial de página que:
  - implementa los mecanismos básicos de navegación estructurada.
  - permite tratar la navegación a una página de forma similar a llamar a un método.

© JMA 2012

## PageFunction<T>

```
<PageFunction ...
    xmlns:local="clr-namespace: ..."
    x:TypeArguments="local:ParamType"
    ... >

public partial class MyPageFunction : PageFunction<ParamType> {
    public MyPageFunction(ParamType param) {
        InitializeComponent();
        ...
        p = param;
        ...
    }
    ...
}
```

© JMA 2012

# Devolución de resultado

- PageFunction<T> implementa el método OnReturn para volver al llamador, pasando un valor devuelto a través de un objeto ReturnEventArgs<T>.

```
void ok_Click(object sender, RoutedEventArgs e) {  
    OnReturn(new ReturnEventArgs<RsltType>(rslt));  
}  
void cancel_Click(object sender, RoutedEventArgs e) {  
    OnReturn(null);  
}
```

© JMA 2012

## Pasar parámetros y devolver el resultado

```
void pageFunctionHyperlink_Click(object sender, RoutedEventArgs e) {  
    var param = new ParamType(...);  
    MyPageFunction page = new MyPageFunction(param);  
    page.Return += pageFunction_Return;  
    this.NavigationService.Navigate(page);  
}  
  
void pageFunction_Return(object sender, ReturnEventArgs<string> e) {  
    estado = (e != null ? "Accepted" : "Canceled");  
    if (e != null) {  
        rslt = e.Result;  
        ...  
    }  
}
```

© JMA 2012

## Quitar la página del diario

- La navegación estructurada suele ser una actividad aislada; cuando vuelve la página llamada, la página que llama debe crear y navegar a una nueva página que llama para capturar más datos.
- De forma predeterminada, una función de página se quita automáticamente del historial cuando se llama a OnReturn, porque RemoveFromJournal está a true.
- Para mantener una función de página en el historial de navegación después de llamar a OnReturn, hay que establecer RemoveFromJournal a false.  
`<PageFunction ... RemoveFromJournal="False" ... >`

© JMA 2012

## Objeto Application

- Crear y administrar la infraestructura común de las aplicaciones.
- Realizar el seguimiento e interactuar con la duración de la aplicación.
- Recuperar y procesar los parámetros de la línea de comandos.
- Compartir propiedades y recursos del ámbito de la aplicación.
- Detectar y responder a las excepciones no controladas.
- Devolver códigos de salida.
- Administrar las ventanas en las aplicaciones independientes.
- Realizar el seguimiento y administrar la navegación.
- Se define en:

App.xaml

App.cs

```
public partial class App : Application { }
```

© JMA 2012

## Objeto inicial

- Pagina inicial:  
`<Application ... StartupUri=" MainPage.xaml" ... />`
- Ventana principal:  
`<Application ... StartupUri=" MainWindow.xaml" ... />`  
Application.MainWindow
- Evento:  
`<Application ... Startup="app_Startup" ... />`
- SplashScreen
  - Imagen WIC (BMP, GIF, JPEG, PNG o TIFF)
    - Propiedades → Acción de compilación=SplashScreen

© JMA 2012

## Cierre de la aplicación

- Para facilitar la administración del cierre de la aplicación, Application proporciona el método Shutdown, la propiedad ShutdownMode y los eventos SessionEnding y Exit.
- Modo de apagado (ShutdownMode):
  - OnLastWindowClose
  - OnMainWindowClose
  - OnExplicitShutdown

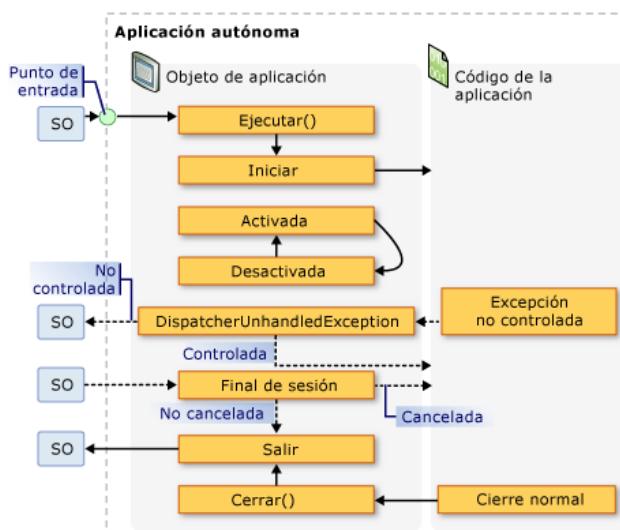
© JMA 2012

# Eventos

- LoadCompleted
  - Se produce cuando se ha cargado y analizado el contenido
- Activated
  - Se inicia y muestra un objeto Window.
  - Cuando el usuario vuelve a la aplicación.
- Deactivated
  - Un usuario cambia a otra aplicación desde la actual.
  - Cuando se cierra la aplicación.
- SessionEnding
  - Se produce cuando el usuario finaliza la sesión de Windows cerrando sesión o apagando el sistema operativo.
- Exit
  - Se produce justo antes de que se cierre una aplicación y no se puede cancelar.
- DispatcherUnhandledException
  - Cuando llega una excepción que no controlada al objeto aplicación.

© JMA 2012

## Eventos de duración de la aplicación



© JMA 2012

# Servicios de Aplicación

- App.Current: objeto Application actual (singleton).
- App.Current.Properties: Diccionario con las propiedades globales del ámbito de la aplicación.
- Recursos globales:
  - <Application.Resources>...</Application.Resources>
  - Application.Current.Resources["ApplicationScopeResource"]
- Administración de ventanas:
  - App.Current.MainWindow: ventana principal de la aplicación.
  - App.Current.Windows: Colección de ventanas instanciadas de una aplicación.
- Administración de la navegación:
  - Intercepta los eventos de Navigating, Navigated, NavigationProgress, NavigationFailed, NavigationStopped, LoadCompleted, FragmentNavigation

© JMA 2012

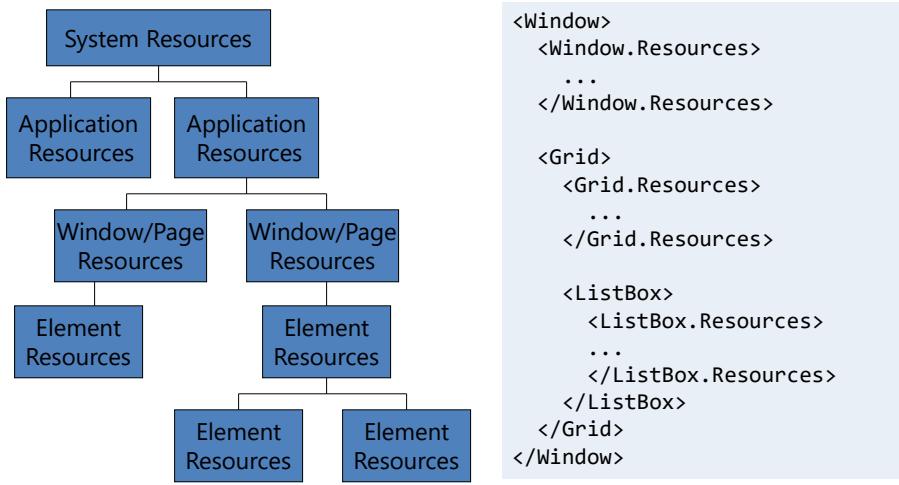
## Áreas Resources

- El área Resources expone un almacén para los recursos compartidos.
- En el área de recursos pueden definirse fuentes de datos, estilos, plantillas, ... que posteriormente podrán ser enlazados.
- Si los recursos cambian, el sistema de recursos asegura que las propiedades de elemento enlazadas a esos recursos se actualizan automáticamente para reflejar el cambio.
- Todos los UIElement puede definir su propiedad recursos.
- Todos los recursos deben estar identificados por una clave x:Key para su referencia.
- Las referencias se establecen como:
  - StaticResource: recuperar solamente
  - DynamicResource: la instancia puede cambiar

```
<Window.Resources>
  <SolidColorBrush x:Key="MyBrush" Color="Gold"/>
</Window.Resources>
...
<Rectangle Fill="{StaticResource MyBrush}" ...>
```

© JMA 2012

# Jerarquía de recursos



© JMA 2012

## SplashScreen

- La SplashScreen muestra una imagen en una ventana de inicio o pantalla, cuando se inicia una aplicación de WPF.
- La SplashScreen clase puede mostrar cualquier formato de imagen compatible con Windows Imaging Component (WIC).
- Por ejemplo, puede utilizar el formato BMP, GIF, JPEG, PNG o TIFF.
- Si la imagen es un archivo PNG e incluye un canal alfa, la imagen se representa utilizando la transparencia definida en el canal alfa.
- Para agregar una imagen existente como una pantalla de presentación
  - Cree o busque la imagen que desea utilizar para la pantalla de presentación. Puede utilizar cualquier formato de imagen compatible con Windows Imaging Component (WIC). Por ejemplo, se puede utilizar el formato BMP, GIF, JPEG, PNG o TIFF.
  - Agregue el archivo de imagen al proyecto de aplicación de WPF.
  - En el Explorador de soluciones, seleccione la imagen.
  - En la ventana Propiedades, haga clic en la flecha de la lista desplegable de la propiedad Acción de compilación.
  - Seleccione SplashScreen en la lista desplegable.

© JMA 2012

# Empaquetar URI en WPF

- En Windows Presentation Foundation (WPF), se utilizan uniform resource identifiers (URIs) para identificar y cargar archivos de muchas maneras, incluidas las que figuran a continuación:
  - Especificando la user interface (UI) que se va a mostrar cuando se inicie una aplicación por primera vez.
  - Cargando imágenes.
  - Navegando a páginas.
  - Cargando archivos de datos no ejecutables.
- Además, se pueden usar URIs para identificar y cargar archivos desde diversas ubicaciones, como las que figuran a continuación:
  - El ensamblado actual.
  - Un ensamblado al que se hace referencia.
  - Una ubicación relativa a un ensamblado.
  - El sitio de origen de la aplicación.

© JMA 2012

# Empaquetar URI en WPF

- El pack URI para un archivo de recursos compilado en un ensamblado al que se hace referencia utiliza la siguiente autoridad y ruta de acceso:
- Autoridad: application:///.
- Ruta de acceso: nombre de un archivo de recursos compilado en un ensamblado al que se hace referencia. La ruta de acceso debe tener el formato siguiente:  
nombreCortoDeEnsamblado[;Versión][;clavePública];component/rutaDeAcceso
  - nombreCortoDeEnsamblado: nombre corto del ensamblado al que se hace referencia.
  - ;Versión [opcional]: versión del ensamblado al que se hace referencia y que contiene el archivo de recursos. Se utiliza cuando hay cargados dos o más ensamblados a los que se hace referencia con el mismo nombre corto.
  - ;clavePública [opcional]: clave pública utilizada para firmar el ensamblado al que se hace referencia. Se utiliza cuando hay cargados dos o más ensamblados a los que se hace referencia con el mismo nombre corto.
  - ;component: especifica que la referencia al ensamblado se hace desde el ensamblado local.
  - /rutaDeAcceso: nombre del archivo de recursos, incluida su ruta de acceso relativa a la carpeta raíz del proyecto del ensamblado al que se hace referencia

```
// Absolute URI (default)
Uri absoluteUri = new Uri("pack://application:,,,/File.xaml", UriKind.Absolute);
// Relative URI
Uri relativeUri = new Uri("/File.xaml", UriKind.Relative);
```

© JMA 2012

# PROPIEDADES, EVENTOS Y COMANDOS

© JMA 2012

## Propiedades de dependencia

- WPF proporciona un conjunto de servicios que se pueden utilizar para extender la funcionalidad de una propiedad de CLR: las propiedades de dependencia.
- El propósito de las propiedades de dependencia es proporcionar una manera de calcular el valor de una propiedad en función del valor de otras entradas.
- Estas otras entradas pueden incluir propiedades del sistema tales como temas y preferencias del usuario, mecanismos de determinación de propiedad Just-In-Time tales como el enlace de datos y las animaciones o guiones gráficos, plantillas del uso múltiple tales como recursos y estilos, o valores conocidos a través de relaciones de elementos primarios-secundarios con otros elementos del árbol de elementos.
- Además, una propiedad de dependencia se puede implementar para que proporcione validación autónoma, valores predeterminados, devoluciones de llamada que supervisen los cambios de otras propiedades y un sistema que pueda forzar valores de la propiedad en función de información que puede estar disponible en tiempo de ejecución.
- Las clases derivadas también pueden cambiar algunas características concretas de una propiedad existente invalidando metadatos de propiedades de dependencia, en lugar de reemplazar la implementación real de propiedades existentes o crear propiedades nuevas.

© JMA 2012

## Respaldo de propiedades de dependencia

- WPF extienden la funcionalidad de propiedad CLR proporcionando un tipo que respalda una propiedad, como implementación alternativa al modelo estándar de respaldar la propiedad con un campo privado.
- El nombre de este tipo es DependencyProperty.
- El otro tipo importante que define WPF es DependencyObject.DependencyObject que define la clase base que puede registrar y poseer una propiedad de dependencia.
- Pasos:
  1. Opcionalmente, preparar metadatos.
  2. Crear y registrar la propiedad de dependencia.
  3. Crear el contenedor CLR de la propiedad.

© JMA 2012

## Propiedad de dependencia

- Una propiedad de dependencia es un atributo (campo) público, de clase, de solo lectura y de tipo DependencyProperty.
- El nombre de la propiedad de dependencia debe utilizar el notación Pascal con el sufijo Property.
 

```
public static readonly DependencyProperty
MyProperty = DependencyProperty.Register(
    "MyProperty", // Nombre sin sufijo
    typeof(int), // Tipo de datos de la propiedad
    typeof(ownerclass), // Clase a la que pertenece la clase
    new PropertyMetadata(0) // Metadatos adicionales
);
```

© JMA 2012

# Contenedor CLR

- Implementación como propiedad de instancia de la propiedad de dependencia.
- Utiliza el identificador de propiedad de dependencia sin el sufijo Property, utilizando las llamadas a GetValue y SetValue para la lectura y escritura de la propiedad, proporcionando así el respaldo para la propiedad utilizando el sistema de propiedades de WPF.
- La propiedad debe ser de tipo declarado a registrar la propiedad de dependencia.

```
public int MyProperty {  
    get { return (int)GetValue(MyPropertyProperty); }  
    set { SetValue(MyPropertyProperty, value); }  
}
```

© JMA 2012

# Metadatos

- Se crean como una instancia de PropertyMetadata o de la más especializada FrameworkPropertyMetadata.
- Permiten establecer:
  - valor predeterminado de la propiedad de dependencia.
  - que afectan al diseño del elemento (AffectsArrange, AffectsMeasure, AffectsRender).
  - que afectan al diseño del elemento contenedor (AffectsParentArrange, AffectsParentMeasure).
  - modo de enlace de datos ( IsNotDataBindable, BindsTwoWayByDefault)
  - métodos de devolución de llamada para propagar cambios o convertir el valor.

© JMA 2012

# Métodos de devolución de llamada

- **PropertyChangedCallback:** La devolución de llamada de cambio de propiedad para el valor Current se utiliza para reenviar el cambio a otras propiedades dependientes, invocando explícitamente las devoluciones de llamada de forzado de valores registradas para esas otras propiedades:

```
private static void OnCurrentReadingChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e) {
    d.CoerceValue(MinReadingProperty);
    d.CoerceValue(MaxReadingProperty);
}
```

- **CoerceValueCallback:** La devolución de llamada de forzado de valores comprueba los valores de las propiedades de las que depende potencialmente la propiedad actual y fuerza el valor actual si es necesario:

```
private static object CoerceCurrentReading(DependencyObject d, object value) {
    Gauge g = (Gauge)d;
    double current = (double)value;
    if (current < g.MinReading) current = g.MinReading;
    if (current > g.MaxReading) current = g.MaxReading;
    return current;
}
```

© JMA 2012

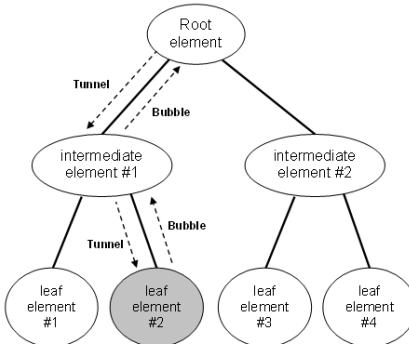
# Eventos enrutados

- **Definición funcional:** un evento enrulado es un tipo de evento que puede invocar controladores o varios agentes de escucha en un árbol de elementos, en lugar de simplemente en el objeto que lo desencadenó.
- **Definición de implementación:** Un evento enrulado es un evento CLR que está respaldado por una instancia de la clase RoutedEvent y que es procesado por el sistema de eventos de Windows Presentation Foundation (WPF).
- Una aplicación típica de WPF contiene muchos elementos, estos elementos se relacionan entre sí a través de un árbol de elementos. En función de la definición del evento, la ruta de eventos puede viajar en cualquiera de las dos direcciones, pero generalmente viaja partiendo del elemento de origen y, a continuación, "se propaga" en sentido ascendente por el árbol de elementos hasta que llega a la raíz (normalmente una página o una ventana).

© JMA 2012

# Enrutamiento de eventos

- Directo (Direct): sólo el propio elemento de origen tiene la oportunidad de invocar controladores como respuesta. Esto es análogo al "enrutamiento" utilizado por Windows Forms para los eventos.
- Propagación (Bubbled): se invocan los controladores de eventos en el origen del evento. A continuación, el evento enrulado va pasando por los elementos primarios sucesivos hasta alcanzar la raíz del árbol de elementos.
- Túnel (Tunneled): inicialmente, se invocan los controladores de eventos en la raíz de árbol de elementos. A continuación, el evento enrulado viaja a través de los elementos secundarios sucesivos a lo largo de la ruta, hacia el elemento del nodo que es el origen del evento enrulado (el elemento que desencadenó el evento enrulado).



© JMA 2012

## Definir Eventos enrutados

- Al igual que en las propiedades de dependencia hay seguir los pasos:
  1. Opcionalmente, preparar metadatos.
  2. Crear y registrar el evento enrulado.
  3. Crear el contenedor CLR del evento.

```
public static readonly RoutedEvent TapEvent =
EventManager.RegisterRoutedEvent(
    "Tap", RoutingStrategy.Bubble, typeof(RoutedEventHandler),
    typeof(MyButtonSimple));
```

```
public event RoutedEventHandler Tap {
    add { AddHandler(TapEvent, value); }
    remove { RemoveHandler(TapEvent, value); }
}
```

© JMA 2012

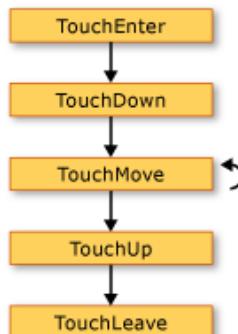
# Entrada táctil y manipulación

- El nuevo hardware y API del sistema operativo Windows 7 proporcionan a las aplicaciones la capacidad de recibir varias entradas táctiles simultáneamente.
- WPF habilita las aplicaciones para detectar y responder a la entrada táctil de una manera similar a otra entrada, como el mouse o teclado, y generar los eventos cuando se produce la entrada táctil.
- WPF expone dos tipos de eventos cuando se produce un toque o entrada táctil: los eventos de toque y eventos de manipulación.
- Los eventos de toque proporcionan los datos sin procesar de cada dedo en una pantalla táctil y su movimiento.
- Los eventos de manipulación interpretan la entrada como acciones.

© JMA 2012

## Eventos Touch

- Las clases base, UIElement, UIElement3D y ContentElement, definen los eventos a los que se puede suscribir para que la aplicación responda a la entrada táctil.
- Los eventos Touch son útiles cuando la aplicación interpreta la entrada táctil como algo distinto de la manipulación de un objeto.
  - TouchDown
  - TouchMove
  - TouchUp
  - TouchEnter
  - TouchLeave
  - PreviewTouchDown
  - PreviewTouchMove
  - PreviewTouchUp
  - GotTouchCapture
  - LostTouchCapture



© JMA 2012

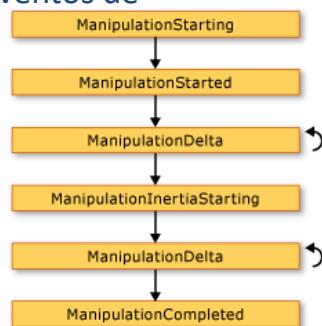
# Eventos de manipulación

- A diferencia de los eventos Touch que simplemente notifican la posición de la entrada táctil, los eventos de manipulación notifican cómo se puede interpretar la entrada.
- Hay tres tipos de manipulaciones:
  - Traducción: Coloque un dedo en un objeto y mueva el dedo por la pantalla táctil para invocar una manipulación de traducción. Esto normalmente mueve el objeto.
  - Expansión: Coloque dos dedos en un objeto y acerque y aleje los dedos uno de otro para invocar una manipulación de expansión. Esto normalmente cambia el tamaño del objeto.
  - Rotación: Coloque dos dedos en un objeto y rote los dedos para invocar una manipulación de rotación. Esto normalmente gira el objeto.
- Se puede producir más de un tipo de manipulación simultáneamente.

© JMA 2012

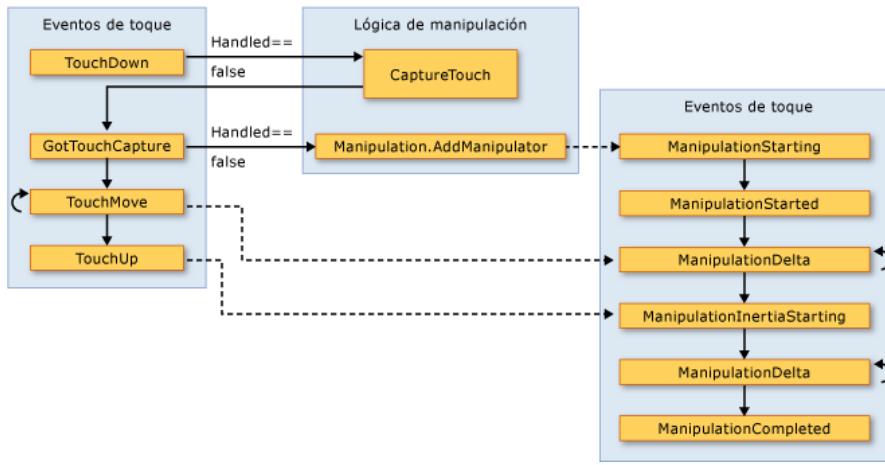
# Eventos de manipulación

- `UIElement` define los siguientes eventos de manipulación.
  - `ManipulationStarting`
  - `ManipulationStarted`
  - `ManipulationDelta`
  - `ManipulationInertiaStarting`
  - `ManipulationCompleted`
  - `ManipulationBoundaryFeedback`
- La propiedad `ManipulationContainer` establece el contenedor al que son relativos todos los cálculos y eventos de manipulación.



© JMA 2012

# Relación entre los eventos de manipulación y toque



© JMA 2012

## Comandos

- Los comandos habilitan el control de entrada en un nivel más semántico que la entrada del dispositivo por eventos.
- Los comandos son directivas simples, como Cut, Copy, Paste u Open.
- Los comandos son útiles para centralizar la lógica de comandos, se podría obtener acceso al mismo comando desde un control Menu, unToolBar, a través de un método abreviado de teclado, ...
- Los comandos también proporcionan un mecanismo para deshabilitar los controles cuando el comando deja de estar disponible.
- WPF proporciona una biblioteca de comandos comunes que consta de ApplicationCommands, MediaCommands, ComponentCommands, NavigationCommands y EditingCommands; o también puede definir la suya propia.

© JMA 2012

## RoutedCommand

- RoutedCommand es la implementación WPF de la interfaz ICommand.
- Cuando se ejecuta RoutedCommand, se desencadenan los eventos PreviewExecuted y Executed en el destino del comando, que se tunelizan y se propagan por el árbol de elementos al igual que cualquier otra entrada.
- Si no se establece ningún destino de comando, el elemento que tenga el foco de teclado será el destino del comando.
- La lógica que ejecuta el comando está asociada a un elemento CommandBinding. Cuando un evento Executed alcanza un elemento CommandBinding para ese comando en concreto, se llama a ExecutedRoutedEventHandler en CommandBinding. Este controlador ejecuta la acción del comando.

© JMA 2012

## Comando enrulado

- El modelo de comando enrulado de WPF se puede descomponer en cuatro conceptos básicos:
  - El comando: es la acción que se va a ejecutar.
  - El origen del comando: es el objeto que invoca el comando.
  - El destino del comando: es el objeto en el que se ejecuta el comando.
  - El enlace del comando: es el objeto que asigna la lógica de comando al comando.

© JMA 2012

# El comando

- Los comandos de WPF se crean mediante la implementación de la interfaz `ICommand`.
- `ICommand` expone dos métodos, `Execute` y `CanExecute`, y un evento, `CanExecuteChanged`.
  - `Execute` realiza las acciones que están asociadas al comando.
  - `CanExecute` determina si el comando se puede ejecutar en el destino del comando actual.
  - El evento `CanExecuteChanged` se produce si el administrador de comandos que centraliza las operaciones de comandos detecta un cambio en el origen del comando que podría invalidar un comando que se ha iniciado pero que el enlace de comando aún no ha ejecutado.
- La implementación de WPF de `ICommand` es la clase `RoutedCommand` y el centro de interés de esta información general.

```
public interface ICommand {
    void Execute(object parameter);
    bool CanExecute(object parameter);
    event EventHandler CanExecuteChanged;
}
```

© JMA 2012

# Orígenes de comando

- Un origen de comando es el objeto que invoca el comando.
- Los orígenes de comando en WPF implementan generalmente la interfaz `ICommandSource`.
- `ICommandSource` expone tres propiedades:
  - `Command` es el comando a ejecutar cuando se invoca el origen de comando.
  - `CommandTarget` es el objeto en el que se ejecuta el comando. Hay que destacar que en WPF la propiedad `CommandTarget` de `ICommandSource` solamente es aplicable cuando la interfaz `ICommand` es un objeto `RoutedCommand`. Si se establece la propiedad `CommandTarget` en un objeto `ICommandSource` y el comando correspondiente no es de tipo `RoutedCommand`, se omite el destino del comando. Si no se establece `CommandTarget`, el destino del comando será el elemento que tenga el foco del teclado.
  - `CommandParameter` es un tipo de datos definido por el usuario utilizado para pasar información a los controladores que implementan el comando.

© JMA 2012

## Enlace del comando

- Un objeto `CommandBinding` asocia un comando con los controladores de eventos que implementan el comando.
- La clase `CommandBinding` contiene una propiedad `Command`, y eventos `PreviewExecuted`, `Executed`, `PreviewCanExecute` y `CanExecute`.

```
<Window.CommandBindings>
    <CommandBinding Command="ApplicationCommands.Open"
                    Executed="OpenCmdExecuted"
                    CanExecute="OpenCmdCanExecute"/>
</Window.CommandBindings>
```

```
void OpenCmdExecuted(object target, ExecutedRoutedEventArgs e) { ... }
void OpenCmdCanExecute(object sender, CanExecuteRoutedEventArgs e)
{
    e.CanExecute = true;
}
```

© JMA 2012

## Destino de comando

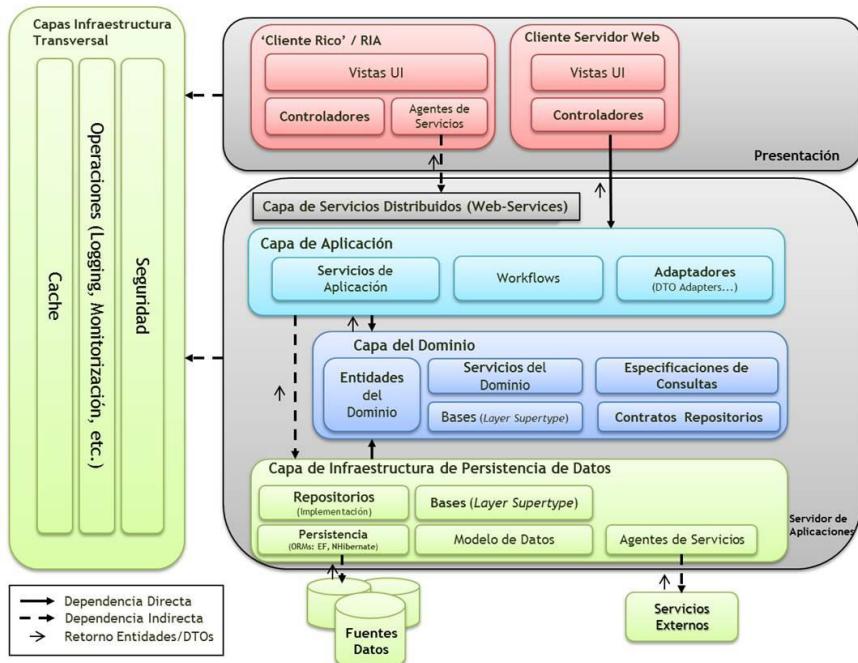
- El destino de comando es el elemento en el que se ejecuta el comando.
- En lo referente a un objeto `RoutedCommand`, el destino de comando es el elemento en el que se inicia el enrutado de `CanExecute` y `Executed`.
- La propiedad `CommandTarget` de `ICommandSource` solamente se aplica cuando `ICommand` es `RoutedCommand`.
- Si se establece la propiedad `CommandTarget` en un objeto `ICommandSource` y el comando correspondiente no es de tipo `RoutedCommand`, se omite el destino del comando.

© JMA 2012

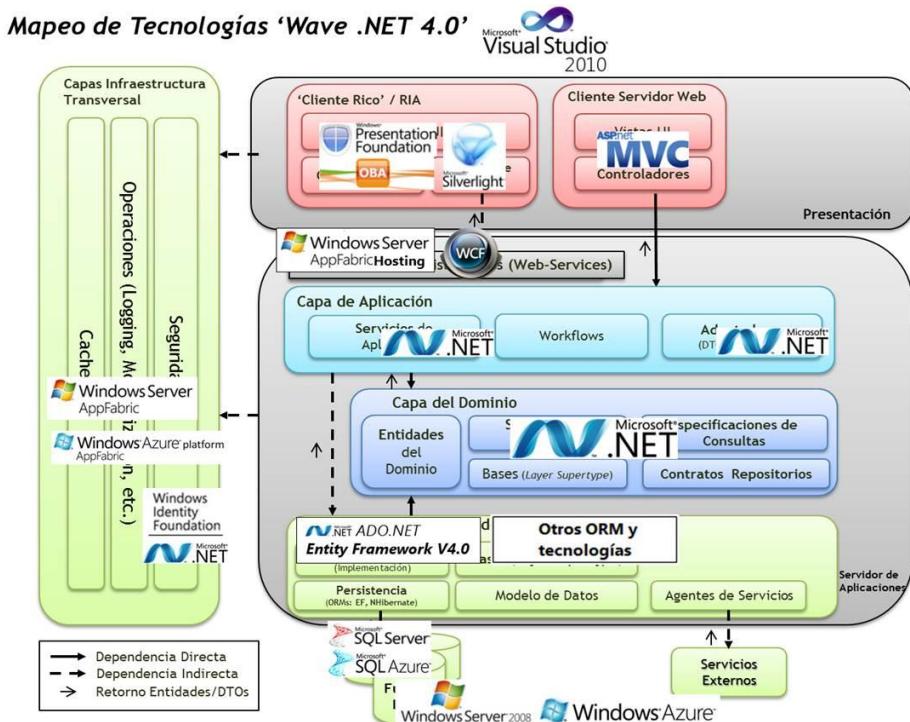
## ENLACE A DATOS

© IMA 2012

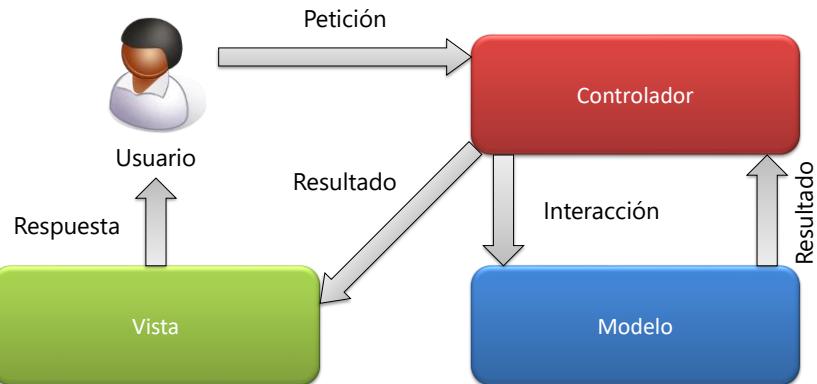
*Arquitectura N-Capas con Orientación al Dominio*



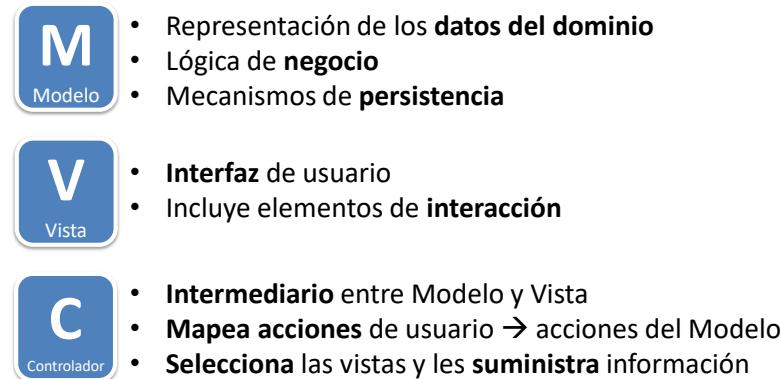
© IMA 2012



## El patrón MVC en ASP.NET



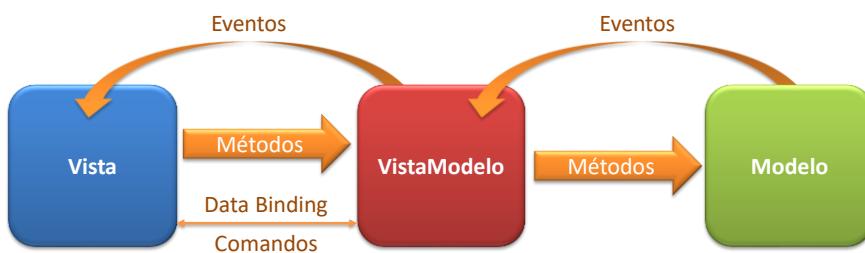
# El patrón MVC



© JMA 2012

## Patrón de Diseño Model View ViewModel (MVVM)

- El **Modelo** es la entidad que representa el concepto de negocio.
- La **Vista** es la representación gráfica del control o un conjunto de controles que muestran el Modelo de datos en pantalla.
- La **VistaViewModel** es la que une todo. Contiene la lógica del interfaz de usuario, los comandos, los eventos y una referencia al Modelo.



© JMA 2012

## Características MVVM

- La vista y el modelo de vista se comunican mediante enlaces de datos, métodos, propiedades, eventos y mensajes.
- El modelo de vista expone propiedades y comandos además de modelos.
- La vista se encarga de sus propios eventos relacionados con la interface al usuario y los pasa al modelo de vista mediante comandos.
- Los modelos y propiedades en el modelo de vista son actualizados desde la vista usando enlaces de datos bidireccionales.

© JMA 2012

## ¿Cuáles son los beneficios del patrón MVVM?

- *Separación de vista / presentación.*
- *Permite las pruebas unitarias:* como la lógica de presentación está separada de la vista, podemos realizar pruebas unitarias sobre la VistaModelo.
- *Mejora la reutilización de código.*
- *Soporte para manejar datos en tiempo de diseño.*
- *Múltiples vistas:* la VistaModelo puede ser presentada en múltiples vistas, dependiendo del rol del usuario por ejemplo.

© JMA 2012

# Patrones

- Observable
  - IObservable
  - IObservableCollection
  - INotifyPropertyChanged
  - ObservableCollection
- Command
  - ICommand
- Editable
  - IEditableObject

© JMA 2012

## Observable Class

```
public class Entidad : INotifyPropertyChanged {
    public event PropertyChangedEventHandler PropertyChanged;
    public void OnPropertyChanged(PropertyChangedEventArgs e) {
        if (PropertyChanged != null)
            PropertyChanged(this, e);
    }
    private string propiedad;
    public string Propiedad {
        get { return propiedad; }
        set {
            if (propiedad != value) {
                propiedad = value;
                OnPropertyChanged("Propiedad");
            }
        }
    }
}
```

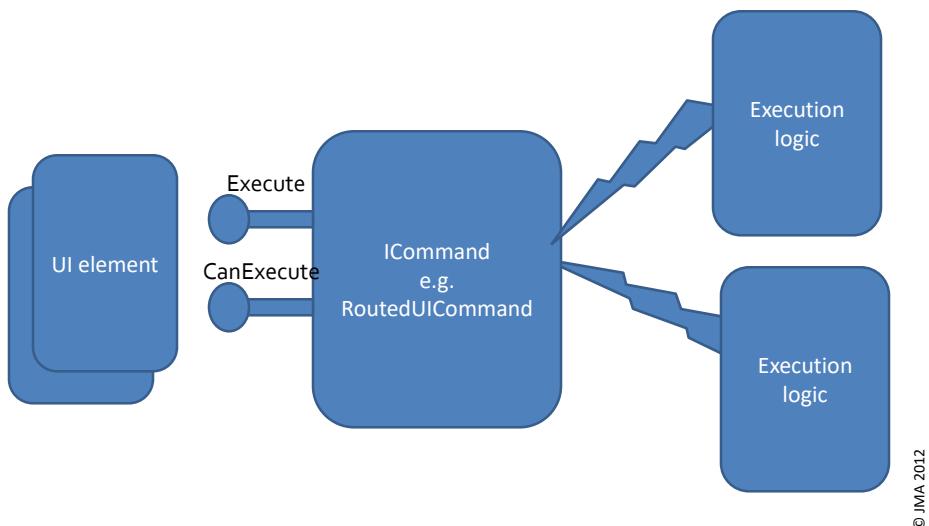
© JMA 2012

# Observable Class (v 4.5)

```
public abstract class ObservableBase : INotifyPropertyChanged {
    public event PropertyChangedEventHandler PropertyChanged;
    protected virtual void NotifyPropertyChanged([CallerMemberName] string propertyName = null) {
        Debug.Assert(
            string.IsNullOrEmpty(propertyName) ||
            (GetType().GetProperty(propertyName) != null),
            "Check that the property name exists for this instance.");
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
public class Entidad : ObservableBase {
    private string propiedad;
    public string Propiedad {
        get { return propiedad; }
        set {
            if (propiedad != value) {
                propiedad = value;
                NotifyPropertyChanged();
            }
        }
    }
}
```

© JMA 2012

# Patrón Command



© JMA 2012

## DelegateCommand

```
public class DelegateCommand : ICommand {  
    public DelegateCommand(Action<object> ExecuteHandler,  
        Func<object,bool> CanExecuteHandler) { ... }  
    public DelegateCommand(Action<object> ExecuteHandler) { }  
}  
  
public DelegateCommand Comando {  
    get {  
        return new DelegateCommand(  
            cmd => { ... },  
            cmd => { return true; }  
        );  
    }  
}
```

<https://www.wpftutorial.net/DelegateCommand.html>

© JMA 2012

## Frameworks de MVVM

- Prism – Microsoft
- MVVM Light
- Simple MVVM
- Caliburn

© JMA 2012

## Enlace de datos (DataBinding)

- *El enlace de datos es el proceso que establece una conexión entre la UI de la aplicación y la lógica del negocio*

Los objetos controles tienen funciones integradas que permiten definir de forma flexible elementos de datos individuales o colecciones de elementos de datos.

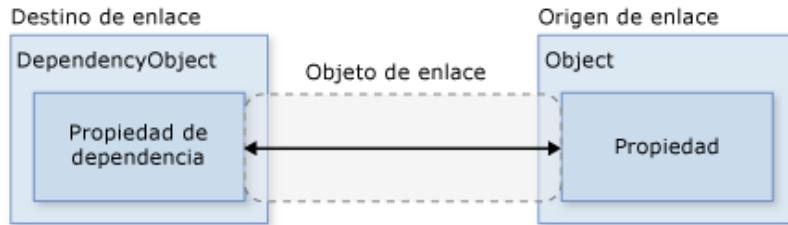
© JMA 2012

## Ventajas del DataBinding

- Un mayor número de propiedades que admiten de forma inherente el enlace de datos.
- Una representación flexible de los datos en la UI.
- La separación bien definida de la lógica del negocio de la UI.

© JMA 2012

## ... Enlace de Datos

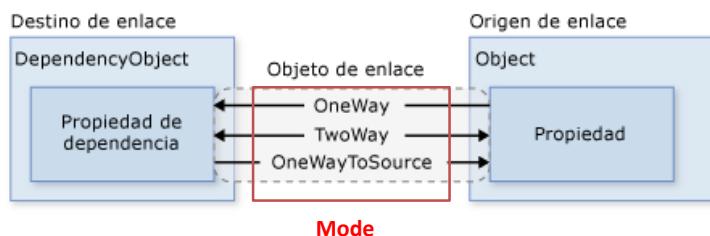


**El enlace de datos es esencialmente el puente entre el destino del enlace y el origen del enlace.**

© JMA 2012

### Dirección del flujo de datos

Se controla este comportamiento estableciendo la propiedad Mode del objeto Binding



© JMA 2012

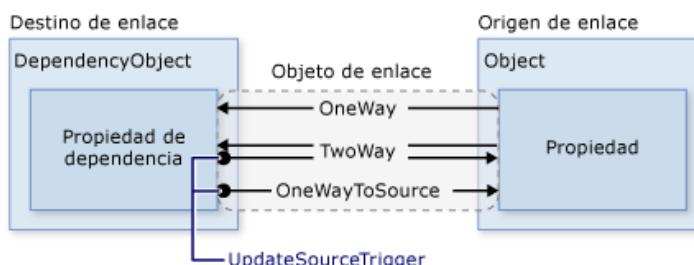
## Dirección del flujo de datos

- El enlace **OneWay** permite que los cambios en la propiedad de origen actualicen automáticamente la propiedad de destino, pero los cambios en la propiedad de destino no se propagan de nuevo a la propiedad de origen.
- El enlace **TwoWay** permite que los cambios realizados en la propiedad de origen o en la de destino se actualicen automáticamente en el otro.
- **OneWayToSource** es el enlace inverso de OneWay; actualiza la propiedad de origen cuando cambia la propiedad de destino.
- El enlace **OneTime** permite que la propiedad de origen inicialice la propiedad de destino, pero los demás cambios no se propagan.

© JMA 2012

### ¿Qué desencadena la actualización del origen?

La propiedad [UpdateSourceTrigger](#) del enlace determina qué desencadena la actualización del origen.



- Si el valor de [UpdateSourceTrigger](#) es `PropertyChanged`, el valor se actualizará en cuanto cambie la propiedad de destino y viceversa.
- Si el valor de [UpdateSourceTrigger](#) es `LostFocus` (valor por defecto), ese valor sólo se actualizará con el nuevo valor cuando la propiedad de destino pierda el foco.

© JMA 2012

# Orígenes de enlaces

- Recursos:
  - Objetos del CLR
  - ObjectDataProvider
  - XmlDataProvider
  - CollectionViewSource
- Propiedad DataContext del contenedor.

© JMA 2012

## ObjectDataProvider

- Ajusta y crea un objeto que puede utilizar como origen de enlace.
- Permite pasar parámetros al constructor de la clase.
- Permite ejecutar un método con parámetros del objeto como fuente de datos.
- La propiedad Data expone el objeto de datos subyacente.

```
<Window.Resources>
    <ObjectDataProvider ObjectType="{x:Type local:TemperatureScale}"
        MethodName="ConvertTemp" x:Key="convertTemp">
        <ObjectDataProvider.MethodParameters>
            <system:Double>0</system:Double>
            <local:TempType>Celsius</local:TempType>
        </ObjectDataProvider.MethodParameters>
    </ObjectDataProvider>
```

© JMA 2012

## XmlDataProvider

- Los datos subyacentes a los que se puede tener acceso mediante el enlace de datos en la aplicación pueden ser cualquier árbol de nodos XML.

- El referenciado de los nodos se realiza mediante consultas Xpath.

```
<XmlDataProvider x:Key="InventoryData" XPath="Inventory/Books">
    <x:XData>
        <Inventory xmlns="">
            ...
        </Inventory>
    </x:XData>
</XmlDataProvider>
<XmlDataProvider x:Key="BookData" Source="data\bookdata.xml"
    XPath="Books"/>
<XmlDataProvider x:Key="BookData" Source="http://MyUrl"
    XPath="Books"/>
```

© JMA 2012

## CollectionViewSource

- Representa una vista para agrupar, ordenar, filtrar y navegar por una colección de datos.

- Requiere los xmlns:

```
– xmlns:scm="clr-
    namespace:System.ComponentModel;assembly=WindowsBase"
– xmlns:dat="clr-
    namespace:System.Windows.Data;assembly=PresentationFramework"
<CollectionViewSource Source="{StaticResource places}" x:Key="cvs">
    <CollectionViewSource.SortDescriptions>
        <scm:SortDescription PropertyName="CityName"/>
    </CollectionViewSource.SortDescriptions>
    <CollectionViewSource.GroupDescriptions>
        <dat:PropertyGroupDescription PropertyName="State"/>
    </CollectionViewSource.GroupDescriptions>
</CollectionViewSource>
```

© JMA 2012

# Extensiones de enlazado

- Las llaves ({ y }) indican el uso de una extensión de marcado que se aparta del tratamiento general de valores de atributo.
  - **Binding**: proporciona un valor enlazado a datos para una propiedad, utilizando el contexto de datos que se aplica al objeto primario en tiempo de ejecución.
  - **RelativeSource**: proporciona información de origen para un objeto Binding que puede navegar por varias posibles relaciones en el árbol de objetos en tiempo de ejecución.
  - **TemplateBinding**: permite que una plantilla de control utilice valores para propiedades con plantilla procedentes de propiedades definidas por el modelo de objetos de la clase que utilizará la plantilla.
  - **StaticResource**: proporciona un valor para una propiedad sustituyendo el valor de un recurso ya definido.
  - **DynamicResource**: proporciona un valor para una propiedad aplazando ese valor para que sea una referencia a un recurso en tiempo de ejecución.
  - **ColorConvertedBitmap**: admite un escenario de creación de imágenes relativamente avanzado.
  - **ComponentResourceKey** y **ThemeDictionary**: admiten aspectos de la búsqueda de recursos y temas que se empaquetan con controles personalizados.

© JMA 2012

## {Binding}

- Origen:
  - Source, RelativeSource
- Ruta de acceso:
  - Path, XPath, ElementName
- Dirección:
  - Mode, BindsDirectlyToSource
- Conversiones:
  - StringFormat, Converter, ConverterParameter, ConverterCulture
- Notificaciones:
  - NotifyOnSourceUpdated, NotifyOnTargetUpdated, NotifyOnValidationError, UpdateSourceTrigger
- Validaciones:
  - ValidationRules, ValidatesOnExceptions, ValidatesOnDataErrors

© JMA 2012

# Nivel de vinculación

- Contextos de datos: DataContext
- Vinculación de propiedades
  - Propiedades de dependencia
- Vinculación de colecciones
  - ItemsControl
    - ItemsSource, DisplayMemberPath, SelectedValuePath
    - SelectedValue, SelectedItem

© JMA 2012

# Convertidores

```
/// <summary>
/// Convierte un numero a cadena
/// </summary>
public class StringToNullConverter : IValueConverter {
    #region Miembros de IValueConverter

    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture) {
        return value;
    }

    public object ConvertBack(object value, Type targetType, object
        parameter, System.Globalization.CultureInfo culture) {
        if (value == null)
            return null;
        else if(value.ToString().EstaVacio())
            return null;
        return value;
    }
}
```

© JMA 2012

# Reglas de validación

```
public class RequiredValidator : ValidationRule {
    public override ValidationResult Validate(object value,
        System.Globalization.CultureInfo cultureInfo) {
        if (value == null || String.IsNullOrEmpty(value.ToString()))
            return new ValidationResult(false, "Requerido");
        else
            return ValidationResult.ValidResult;
    }
}
<TextBox.Text>
<Binding Path="Nombre" UpdateSourceTrigger="PropertyChanged" >
    <Binding.ValidationRules>
        <c:RequiredValidator />
    </Binding.ValidationRules>
</Binding>
</TextBox.Text>
```

© JMA 2012

## Validación

- **IDataErrorInfo**
  - Proporciona la funcionalidad para facilitar información de error personalizada a la que puede enlazar una interfaz de usuario.
  - **Error** : Obtiene un mensaje de error que indica lo que le pasa a este objeto.
  - **Item[String]**: Obtiene el mensaje de error correspondiente a la propiedad con el nombre especificado.

© JMA 2012

# Validación

- **INotifyDataErrorInfo (v 4.5)**
  - Permite que las clases de entidad de datos implementar reglas de validación personalizadas y exponer los resultados de validación de forma asíncrona, admitiendo objetos de error personalizados, varios errores por propiedad, errores entre propiedades (que afectan a varias propiedades, se pueden asociar con una o todas las propiedades afectadas) y errores de nivel de entidad (que afectan a varias propiedades o afectar a la entidad completa sin que afecte a una propiedad determinada).
  - **HasErrors:** Obtiene un valor que indica si la entidad tiene errores de validación.
  - **GetErrors(String):** Obtiene los errores de validación para una propiedad específica o para toda la entidad.
  - **ErrorsChanged:** Evento que se produce cuando los errores de validación han cambiado para una propiedad o para la toda la entidad.

© JMA 2012

## Plantilla de Validación

```
<ControlTemplate x:Key="ValidationTemplate">
    <Grid DataContext="{Binding}" Name="CntrlContainer">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="Auto" />
        </Grid.ColumnDefinitions>
        <AdornedElementPlaceholder Name="AdornedCntrl" />
        <Image x:Name="iconoErr" Stretch="None"
            Source="/Demos;component/Imagenes/16x16/exclamation.png"
            ToolTip="{Binding ElementName=AdornedCntrl,
Path=AdornedElement.(Validation.Errors)[0].ErrorContent}" Grid.Column="1" />
    </Grid>
</ControlTemplate>
<Style x:Key="CntrValidado" TargetType="{x:Type Control}">
    <Setter Property="Validation.ErrorTemplate" Value="{StaticResource ValidationTemplate}" />
    <Style.Triggers>
        <Trigger Property="Validation.HasError" Value="true">
            <Setter Property="ToolTip" Value="{Binding
Path=(Validation.Errors)[0].ErrorContent, RelativeSource={x:Static RelativeSource.Self}}"/>
            <Setter Property="Background" Value="{StaticResource
FondoErrorValidacion}" />
        </Trigger>
    </Style.Triggers>
</Style>
```

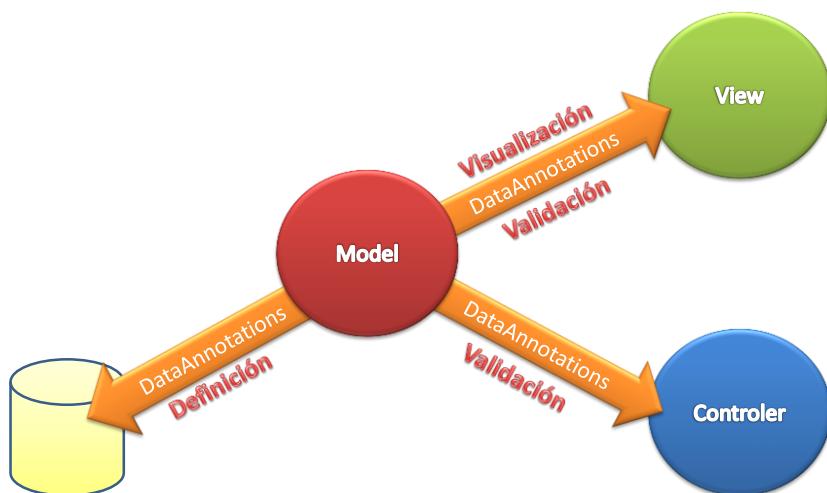
© JMA 2012

## DataAnnotations

- System.ComponentModel.DataAnnotations;
- Permiten definir los metadatos de los modelos de datos para su uso por entidades externas.
- Cuenta con decoradores para:
  - Definición del modelo de datos: claves (PK), asociaciones (FK), simultaneidad, ...
  - Interfaz de usuario y localización: Título, descripción, formato, solo lectura, ...
  - Validaciones y errores personalizados: Obligatoriedad, longitudes, formatos, ...

© JMA 2012

## Contexto Declarativo



© JMA 2012

# Visualización

Decorador	Descripción
DataType	Especifica el nombre de un tipo adicional que debe asociarse a un campo de datos.
Display	Permite especificar las cadenas traducibles de los tipos y miembros de las clases parciales de entidad.
DisplayFormat	Especifica el modo en que los datos dinámicos de ASP.NET muestran y dan formato a los campos de datos.
ScaffoldColumn	Especifica si una clase o columna de datos usa la técnica scaffolding.
ScaffoldTable	Especifica si una clase o tabla de datos usa la técnica scaffolding.
UIHint	Especifica la plantilla o el control de usuario que los datos dinámicos usan para mostrar un campo de datos.
<b>System.Web.Mvc</b>	
HiddenInput	Indica que una propiedad se debería presentar como un elemento input oculto.

© JMA 2012

## [Display]

- **Name:** valor que se usa para mostrarlo en la interfaz de usuario (Título, Etiqueta, ...).
- **Description:** valor que se usa para mostrar una descripción en la interfaz de usuario.
- **Prompt:** valor que se usará para establecer la marca de agua para los avisos en la interfaz de usuario.
- **ShortName:** valor que se usa para la etiqueta de columna de la cuadrícula.
- **GroupName:** valor que se usa para agrupar campos en la interfaz de usuario.
- **Order:** número del orden de la columna.

© JMA 2012

# Tipos asociados

Asociado	Descripción
DateTime	Representa un instante de tiempo, expresado en forma de fecha y hora del día.
Date	Representa un valor de fecha.
Time	Representa un valor de hora.
Duration	Representa una cantidad de tiempo continua durante la que existe un objeto.
PhoneNumber	Representa un valor de número de teléfono.
Currency	Representa un valor de divisa.
Text	Representa texto que se muestra.
Html	Representa un archivo HTML.
MultilineText	Representa texto multilínea.
EmailAddress	Representa una dirección de correo electrónico.
Password	Representa un valor de contraseña.
Url	Representa un valor de dirección URL.
ImageUrl	Representa una URL en una imagen.
CreditCard	Representa un número de tarjeta de crédito.
PostalCode	Representa un código postal.
Upload	Representa el tipo de datos de la carga de archivos.

© JMA 2012

# Validación

Decorador	Descripción
Required	Especifica que un campo de datos necesita un valor.
DataType	Especifica el nombre de un tipo adicional que debe asociarse a un campo de datos. Decoradores especializados: CreditCard, EmailAddress, EnumDataType, FileExtensions, Phone, Url
Compare	Compara dos propiedades.
Range	Especifica las restricciones de intervalo numérico para el valor de un campo de datos.
StringLength	Especifica la longitud mínima y máxima de caracteres que se permiten en un campo de datos. Decoradores especializados: MinLength, MaxLength
RegularExpression	Especifica que un valor de campo de datos debe coincidir con la expresión regular especificada.
CustomValidation	Especifica un método de validación personalizado que se utiliza para validar una propiedad o una instancia de clase.

© JMA 2012

# Validación manual de objetos

- En `System.ComponentModel.DataAnnotations`, la clase estática `Validator` ofrece métodos que permiten realizar las comprobaciones de forma directa sobre objetos o propiedades concretas.

```
IEnumerable<ValidationResult> getValidationErrors(object obj) {  
    var validationResults = new List<ValidationResult>();  
    var context = new ValidationContext(obj, null, null);  
    Validator.TryValidateObject(obj,  
        context,  
        validationResults,  
        true);  
    return validationResults;  
}
```

© JMA 2012

## IValidableObject

- Obliga a implementar un único método, llamado `Validate()`, que determina si el objeto especificado es válido.
- Será invocado automáticamente por `TryValidateObject()` siempre que no encuentre errores al comprobar las restricciones especificadas mediante anotaciones.
- Devolverá una lista de objetos `ValidationResult` con los resultados de las comprobaciones.
- En las clases prescriptivas (EF) se aplica con una partial class.
- Interfaces relacionados: `IDataErrorInfo`, `INotifyDataErrorInfo`

© JMA 2012

## Anotar clases ya existentes

- Se requiere una clase auxiliar con las propiedades a decorar decoradas.
- Declarativa: [MetadataType(typeof(tipo))]
  - Se aplica con una partial class (en el mismo ensamblado que la clase a anotar).
- Imperativa: System.ComponentModel.TypeDescriptor y AssociatedMetadataTypeTypeDescriptionProvider:

```
var descriptionProvider = new
    AssociatedMetadataTypeTypeDescriptionProvider(
        typeof(Friend), typeof(FriendMetadata));
TypeDescriptor.AddProviderTransparent(
    descriptionProvider, typeof(Friend));
```

© JMA 2012

## Repositorio

- Un repositorio separa la lógica empresarial de las interacciones con la base de datos subyacente y centra el acceso a datos en un área, lo que facilita su creación y mantenimiento.
- El repositorio pertenecen a la capa de infraestructura y devuelve los objetos del modelo de dominio.
- Deberían implementar el patrón de doble herencia.
- Forma parte de los Domain Driven Design patterns: Domain Entity, Value-Object, Aggregates, Repository, Unit of Work, Specification, Dependency Injection, Inversion of Control (IoC).

© JMA 2012

# MultiBinding y PriorityBinding

- Asociar una colección de objetos Binding a una sola propiedad.

```
<TextBlock>
    <TextBlock.Text>
        <MultiBinding StringFormat="{1}, {0}">
            <Binding Path="FirstName"></Binding>
            <Binding Path="LastName"></Binding>
        </MultiBinding>
    </TextBlock.Text>
</TextBlock>
```

- Asociar el primer enlace de la colección que genera un valor correctamente.

```
<TextBlock Background="Honeydew" Width="100" HorizontalAlignment="Center">
    <TextBlock.Text>
        <PriorityBinding FallbackValue="defaultvalue">
            <Binding Path="SlowestDP" IsAsync="True"/>
            <Binding Path="SlowerDP" IsAsync="True"/>
            <Binding Path="FastDP" />
        </PriorityBinding>
    </TextBlock.Text>
</TextBlock>
```

© JMA 2012

## PLANTILLAS DE DATOS

© JMA 2012

# Plantillas de datos

- El modelo de plantillas de datos de WPF proporciona gran flexibilidad para definir la presentación de los datos.
- Los controles WPF tienen funcionalidades integradas para admitir la personalización de la presentación de los datos.
- Para mostrar un objeto utiliza el método `ToString()` si no dispone de una plantilla de datos.
- Las plantillas se definen con la etiqueta `DataTemplate` y reciben como `DataContext` el objeto a dar formato.

```
<DataTemplate>
    <StackPanel>
        <TextBlock Text="{Binding Path=TaskName}" />
        <TextBlock Text="{Binding Path=Description}" />
        <TextBlock Text="{Binding Path=Priority}" />
    </StackPanel>
</DataTemplate>
```

© JMA 2012

# Ámbito de las plantillas

- Local:

```
<ListBox ItemsSource="{Binding Source={StaticResource myList}}">
    <ListBox.ItemTemplate>
        <DataTemplate>
            ...
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```
- Recurso:

```
<Window.Resources>
    ...
    <DataTemplate x:Key="myTaskTemplate">
        ...
    </DataTemplate>
    ...
</Window.Resources>
...
<ListBox ... ItemTemplate="{StaticResource myTaskTemplate}" />
```

© JMA 2012

## La propiedad DataType

- La propiedad **DataType** permite asociar una plantilla a un determinado tipo de datos (clase).
- Si se ha establecido el **DataType** no es obligatorio asignar la **x:Key** cuando se definen en el área de recursos, pero en la misma área no puede haber dos plantillas diferentes asociadas al mismo **DataType**.
- En caso de no indicarse explícitamente mediante la **x:Key**, WPF recorre ascendenteamente el árbol de recursos buscando la primera plantilla que satisfaga el tipo.
- Especialmente indicado para colecciones y elementos multi-tipos.

```
<DataTemplate DataType="{x:Type local:Task}">
    <StackPanel>
        ...
    </StackPanel>
</DataTemplate>
```

© JMA 2012

## DataTriggers

- Un desencadenador de datos establece propiedades o inicia acciones, como una animación, cuando cambia un valor de propiedad vinculada.

```
<DataTemplate x:Key="myTaskTemplate">
    ...
    <DataTemplate.Triggers>
        <DataTrigger Binding="{Binding Path=TaskType}" Value="Home">
            <Setter TargetName="border" Property="BorderBrush" Value="Yellow"/>
            <Setter TargetName="border" Property="Background" Value="White" />
        </DataTrigger>
    </DataTemplate.Triggers>
</DataTemplate>
```

- Se puede disparar por la concurrencia de varios valores:

```
<MultiDataTrigger>
    <MultiDataTrigger.Conditions>
        <Condition Binding="{Binding Path=Name}" Value="Portland" />
        <Condition Binding="{Binding Path=State}" Value="OR" />
    </MultiDataTrigger.Conditions>
    <Setter Property="Background" Value="Cyan" />
</MultiDataTrigger>
```

© JMA 2012

# ESTILOS, PLANTILLAS Y CREACIÓN DE CONTROLES

© JMA 2012

## Diccionarios de recursos

- El diccionario contiene recursos de WPF utilizados por los componentes y otros elementos de una aplicación WPF.
- Se crean como ficheros XAML sin código subyacente.

```
<ResourceDictionary  
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
    xmlns:local="clr-namespace:MVVM" >
```

```
</ResourceDictionary>
```

- Esta compuesto de recursos que debes estar identificados por x:Key o x>Type.
- No puede haber dos recursos con el mismo x:Key.
- No puede haber dos recursos con el mismo x>Type si no tienen asociado un x:Key dado que el tipo actúa de clave en estos casos.
- Todos los recursos reutilizables deberían pertenecer a un diccionario, de tal forma que los recursos se pueden compartir entre las aplicaciones y también quedan aislados de un modo más cómodo para su localización.

© JMA 2012

# Diccionarios de recursos

- Los ficheros con diccionarios de recursos deben combinarse con recursos locales del elementos donde quieren utilizarse.
- Un elemento tiene disponibles todos los recursos de su contenedor.
- El app.xaml contiene los recursos globales para toda la aplicación.

```
<Application.Resources>
    <ResourceDictionary>
        <ResourceDictionary.MergedDictionaries>
            <ResourceDictionary Source="Theme/MiTema.xaml" />
            <ResourceDictionary Source="myresourcedictionary.xaml"/>
            <ResourceDictionary Source="myresourcedictionary2.xaml"/>
        </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
</Application.Resources>
```

© JMA 2012

# Estilos

- Un Style es un conjunto de valores por defecto de las propiedades de cualquier elemento que derive de FrameworkElement o FrameworkContentElement.
- Los valores locales de las propiedades priman sobre los valores establecidos en el Style.
- El Style se puede aplicar a múltiples elementos por lo que se unifica el diseño y facilita la mantenibilidad manteniendo la coherencia estética de la aplicación.
- El estilo se establece con las propiedades Style y las propiedades con sufijo Style.

```
<Style TargetType="{x:Type TextBlock}">
    <Setter Property="HorizontalAlignment" Value="Center" />
    <Setter Property="FontFamily" Value="Comic Sans MS"/>
    <Setter Property="FontSize" Value="14"/>
</Style>
```

© JMA 2012

## Elementos del estilo

- **TargetType:** tipo para el que está previsto este estilo, sin x:Key se aplica a todos los objetos de este tipo.
- **BasedOn:** estilo definido que es la base del estilo actual.
- **Resources:** colección de recursos que se pueden usar en el ámbito del estilo.
- **Setters:** colección de objetos Setter y EventSetter.
- **Triggers:** colección de objetos TriggerBase que aplican valores de propiedad basados en condiciones especificadas.

© JMA 2012

## Setter

- Representa un establecedor que aplica un valor de propiedad.
  - **Property:** establece la propiedad a la que se va a aplicar el valor de la propiedad Value.
  - **Value:** establece el valor que se va aplicar a la propiedad especificada por Setter.
  - **TargetName:** establece el nombre del objeto al que va destinado el Setter cuando se define en un Trigger de plantilla.
- ```
<Setter Property="Background"
       Value="{DynamicResource WindowBackgroundBrush}" />
```

© JMA 2012

## Desencadenadores

- Un desencadenador establece propiedades o inicia acciones, como una animación, cuando cambia un valor de propiedad o cuando se genera un evento.
- Style , ControlTemplate y DataTemplate tienen una propiedad Triggers que puede contener un conjunto de desencadenadores.
- Hay varios tipos de desencadenadores:
  - Desencadenadores de propiedades
  - Desencadenadores de eventos
  - Desencadenadores de enlazados a datos

© JMA 2012

## Desencadenadores de propiedades

- Un objeto Trigger que establece valores de propiedad o inicia acciones según el valor de una propiedad se denomina desencadenador de propiedad.

```
<Style.Triggers>
  <Trigger Property="IsSelected" Value="True">
    <Setter Property="Opacity" Value="1.0" />
    <Setter Property="MaxHeight" Value="75" />
  </Trigger>
  ...
</Style.Triggers>
```

© JMA 2012

## Desencadenadores de eventos

- Un desencadenador EventTrigger inicia un conjunto de acciones sobre un guion gráfico (Storyboard) en función de la aparición de un evento.
- RoutedEvent: establece el objeto RoutedEvent que activará este desencadenador.
- SourceName: establece el nombre del objeto con el evento que activa este desencadenador. Sólo los desencadenadores de elemento o de plantilla usan esto.

```
<EventTrigger RoutedEvent="Mouse.MouseEnter">
  <EventTrigger.Actions>
    <BeginStoryboard>
      <Storyboard>
        <DoubleAnimation Duration="0:0:0.2" Storyboard.TargetProperty="MaxHeight" To="90" />
      </Storyboard>
    </BeginStoryboard>
  </EventTrigger.Actions>
</EventTrigger>
<EventTrigger RoutedEvent="Mouse.MouseLeave">
  <EventTrigger.Actions>
    <BeginStoryboard>
      <Storyboard>
        <DoubleAnimation Duration="0:0:1" Storyboard.TargetProperty="MaxHeight" />
      </Storyboard>
    </BeginStoryboard>
  </EventTrigger.Actions>
</EventTrigger>
```

© JMA 2012

## Combinación condiciones

- MultiTrigger permite establecer valores de propiedad según varias condiciones.
 

```
<MultiTrigger>
  <MultiTrigger.Conditions>
    <Condition Property="HasItems" Value="false" />
    <Condition Property="Width" Value="Auto" />
  </MultiTrigger.Conditions>
  <Setter Property="MinWidth" Value="120"/>
</MultiTrigger>
```
- MultiDataTrigger se utilizan cuando la propiedad de la condición está enlazada a datos.

© JMA 2012

## Plantillas de controles

- Una plantilla ControlTemplate especifica la estructura y el comportamiento visuales de un control.
- Puede personalizar la apariencia de un control proporcionándole una nueva plantilla ControlTemplate.
- Cuando se crea una plantilla ControlTemplate, se reemplaza la apariencia de un control existente sin cambiar su funcionalidad.
- Las plantillas ControlTemplate se crean en XAML, por lo que se puede cambiar la apariencia de un control sin escribir código.

© JMA 2012

## Diseño de la plantilla

- Una plantilla ControlTemplate es la combinación de objetos FrameworkElement para compilar un único control.
- Una plantilla ControlTemplate debe tener un solo objeto FrameworkElement como su elemento raíz, que normalmente contiene otros objetos FrameworkElement.
- La combinación de objetos constituye la estructura visual del control.
- La extensión de marcado TemplateBinding enlaza una propiedad de un elemento que se encuentra en la plantilla ControlTemplate a una propiedad pública definida por el control.
- El control ContentPresenter que muestra el contenido de cualquier tipo de objeto.
- Dispone de la Triggers para asociar desencadenadores.

© JMA 2012

# Estados visuales

- Un comportamiento visual describe la apariencia del control cuando se encuentra en un estado determinado.
- Para especificar la apariencia de un control cuando se encuentra en un estado determinado se emplean objetos VisualState.
- Un objeto VisualState contiene un objeto Storyboard que cambia la apariencia de los elementos que se encuentran en la plantilla ControlTemplate.
- No es necesario escribir código para que esto suceda, ya que la lógica del control cambia de estado mediante VisualStateManager.
- Cuando el control entra en el estado especificado por la propiedad VisualState.Name, se inicia Storyboard.
- Cuando el control sale del estado, Storyboard se detiene.

© JMA 2012

## VisualState

```
<VisualStateManager.VisualStateGroups>
    <VisualStateGroup Name="CommonStates">
        <VisualState Name="Normal" />
        <VisualState Name="MouseOver">
            <Storyboard>
                <ColorAnimation Storyboard.TargetName="BorderBrush"
                    Storyboard.TargetProperty="Color" To="Red" />
            </Storyboard>
        </VisualState>
        <VisualState Name="Pressed">
            <Storyboard>
                <ColorAnimation Storyboard.TargetName="BorderBrush"
                    Storyboard.TargetProperty="Color" To="Transparent"/>
            </Storyboard>
        </VisualState>
    </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
```

© JMA 2012

# Contrato de control

- Un control que usa una plantilla ControlTemplate para especificar su estructura visual (mediante objetos FrameworkElement) y su comportamiento visual (mediante objetos VisualState) emplea el modelo de control de elementos.
- Los elementos que un autor de plantillas ControlTemplate debe conocer se comunican mediante el contrato de control.
- Si se entienden los elementos de un contrato de control, es posible personalizar la apariencia de cualquier control que use el modelo de control de elementos.

```
[TemplatePartAttribute(Name = "PART_EditableTextBox", Type = typeof(TextBox))]
[TemplatePartAttribute(Name = "PART_Popup", Type = typeof(Popup))]
public class ComboBox : ItemsControl

<ControlTemplate TargetType="ComboBox">
...
<TextBox x:Name="PART_EditableTextBox" ... />
...
<Popup x:Name="PART_Popup" ...
</ControlTemplate>
```

© JMA 2012

# Temas

- Los temas de Windows Presentation Foundation (WPF) se definen mediante el mecanismo de estilos y plantillas que Windows Presentation Foundation (WPF) expone para personalizar los efectos visuales de cualquier elemento.
- Los recursos de tema de Windows Presentation Foundation (WPF) están almacenados en diccionarios de recursos incrustados.
- Estos diccionarios de recursos deben incrustarse en un ensamblado firmado y pueden incrustarse en el mismo ensamblado que el propio código o en un ensamblado paralelo.
- En el caso de PresentationFramework.dll, el ensamblado que contiene los controles de Windows Presentation Foundation (WPF), los recursos de tema están en una serie de ensamblados paralelos.

© JMA 2012

# Temas

- Combinar para una aplicación.  

```
<ResourceDictionary.MergedDictionaries>
    <!--
        Windows Vista theme      themes/Aero.NormalColor.xaml  Aero
        Windows Classic theme   themes/Classic.xaml    Classic
        Default Windows XP theme themes/Luna.NormalColor.xaml LunaBlue
        Olive green Windows XP theme themes/Luna.Homestead.xaml LunaOliveGreen
        Silver Windows XP theme  themes/Luna.Metallic.xaml  LunaSilver
        Windows XP Media Center Edition theme  themes/Royale.NormalColor.xaml  Royale
        Zune Windows XP theme    themes/Zune.NormalColor.xaml  Zune
        Expression Blend theme   themes/BlendDictionary.xaml  Blend
    -->
    <ResourceDictionary Source="/PresentationFramework.Aero, Version=4.0.0.0,
        Culture=neutral, PublicKeyToken=31bf3856ad364e35,
        ProcessorArchitecture=MSIL;component/themes/aero.normalcolor.xaml" />
    <ResourceDictionary Source="Theme/MiTema.xaml" />
</ResourceDictionary.MergedDictionaries>
```
- Para acceder a los recurso de un tema en el ResourceDictionary :  

```
xmlns:Themes="clr-
namespace:Microsoft.Windows.Themes;assembly=PresentationFramework.Aero"
```

© JMA 2012

# Creación de controles

- Alternativas a la escritura de un nuevo control
  - Contenido enriquecido.
  - Estilos.
  - Plantillas de datos.
  - Plantillas de control.
  - Desencadenadores.
- Modelos para la creación de controles
  - Derivar de UserControl
  - Derivar de Control
  - Derivar de FrameworkElement

© JMA 2012

# Principios de diseño

- Utilizar propiedades de dependencia
- Utilizar eventos enrutados
- Utilizar el Binding
- Diseñar para diseñadores (Visual Studio)
- Definir y usar recursos compartidos
- Definir la estructura y el comportamiento visuales en un ControlTemplate
- Proporcionar el contrato de control

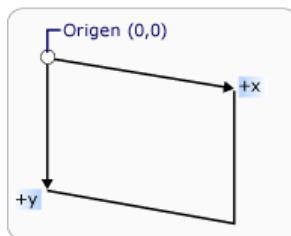
© JMA 2012

# Gráficos

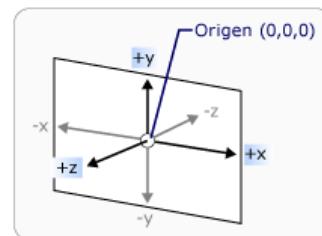
- Gráficos 2D
- Gráficos 3D → ViewPort3D

## Coordinadas

2 Dimensiones



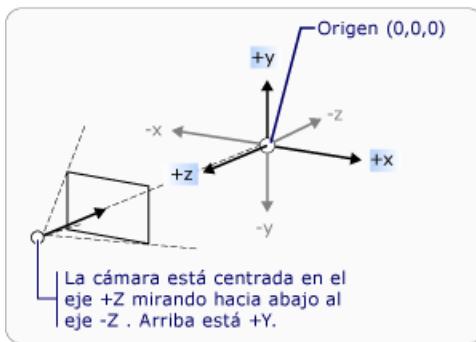
3 Dimensiones



© JMA 2012

## Gráficos 3D

Al crear una escena 3-D, es importante recordar que, en realidad, se está creando una representación 2-D de los objetos 3-D. Dado que una escena 3-D tiene un aspecto diferente dependiendo del punto de vista del espectador, debe especificar ese punto de vista. La clase [Camera](#) permite especificar este punto de vista para una escena 3-D.



© JMA 2012

## Animaciones

**Una animación es una ilusión que se crea mediante el cambio rápido entre una serie de imágenes, cada una de las cuales es ligeramente diferente de la anterior.**

WPF incluye un sistema de control de tiempo eficaz que se expone a través del código administrado y del Extensible Application Markup Language (XAML) que se integra perfectamente en el marco de WPF. La animación WPF facilita la animación de controles y otros objetos gráficos.

© JMA 2012

# Transformaciones

**Se desplazan todos los puntos de un modelo u objeto según un valor o modo especificado.**

## En 2-D:

- *RotateTransform.*
- *ScaleTransform.*
- *SkewTransform.*
- *TranslateTransform.*

## En 3-D:

- *RotateTransform3D.*
- *ScaleTransform3D.*
- *TranslateTransform3D.*
- *MatrixTransform3D.*

© JMA 2012

## Entonces para animar los gráficos...

- Es posible animar directamente las propiedades de los elementos primitivos, pero suele ser más fácil animar las transformaciones que cambian la posición o el aspecto de los modelos.
- Para animar un objeto en WPF, se crea una escala de tiempo, se define una animación (que, en realidad, es un cambio de algún valor de propiedad a lo largo del tiempo) y se especifica la propiedad a la que aplicar la animación.

© JMA 2012

# Animaciones From/To/By

- Se considera la animación básica.
- Es una animación que ocurre entre dos valores inicio y final y va incrementando con un valor de incremento el valor de inicio.
- Tiene una propiedad From con la que se especifica el valor inicio y una propiedad To para especificar el valor final.
- En lugar de la propiedad To se puede usar una propiedad By.
- Para comenzar lo primero que tenemos que hacer es inicializar un StoryBoard (conjunto de imágenes o animaciones en secuencia) para ello utilizamos BeginStoryboard, este lo podemos usar tanto con un Trigger o un EventTrigger.

© JMA 2012

## Animaciones From/To/By

- Los EventTrigger definen tres propiedades fundamentales:
- La propiedad **SourceName** de tipo string que se refiere al nombre del elemento (que se le asocia al elemento con la propiedad Name o x:Name) sobre el que se trabaja.
- La propiedad **RoutedEvent** contiene el nombre del evento que provocará el “desencadenado” de las acciones definidas en el EventTrigger.
- La propiedad **Actions** es la propiedad que define el conjunto de acciones a desencadenar cuando ocurra el evento especificado como condición del EventTrigger.
- A diferencia de los Trigger y DataTrigger, los EventTrigger no tienen concepto de terminación.
- La propiedad TargetProperty indica que propiedad del objeto se va a controlar.

© JMA 2012

# DOCUMENTOS

© JMA 2012

## Introducción

- Windows Presentation Foundation (WPF) proporciona un conjunto versátil de componentes que permiten a los programadores generar aplicaciones con características de documento avanzadas y una experiencia de lectura mejorada.
- Además de las mejoras en las funciones y en la calidad, Windows Presentation Foundation (WPF) proporciona servicios de administración simplificados para el empaquetado, la seguridad y el almacenamiento de los documentos.
- Tipos de documentos
  - Los documentos fijos están diseñados para las aplicaciones que requieren una presentación "what you see is what you get" (WYSIWYG) precisa, independiente del hardware de pantalla o de impresión utilizado.
  - Los documentos dinámicos están diseñados para optimizar su presentación y legibilidad y son óptimos para su uso cuando la facilidad de lectura constituye el principal escenario de consumo del documento.

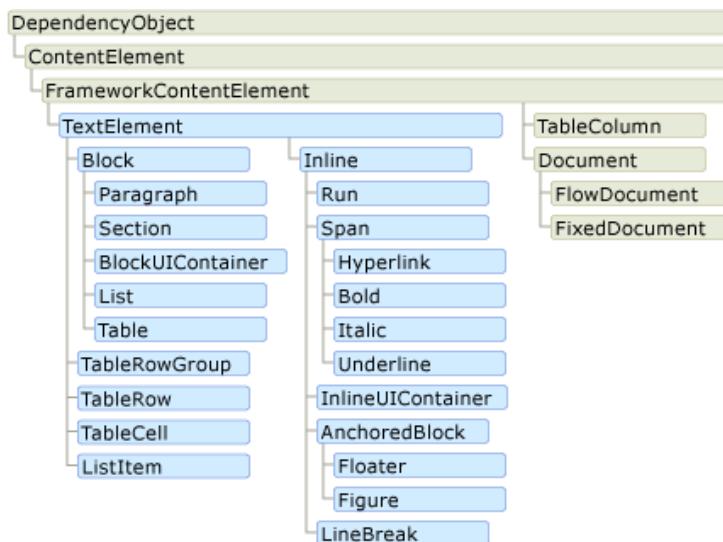
© JMA 2012

# Controles de documentos

- Control de documentos fijos: DocumentViewer
  - DocumentViewer proporciona una interfaz de usuario intuitiva que ofrece compatibilidad integrada para las operaciones comunes, tales como las operaciones de salida impresa, copia en el portapapeles, aplicación de zoom o búsqueda de texto. DocumentViewer se ha diseñado para mostrar el contenido en modo de sólo lectura; la edición o modificación de contenido no está disponible y no se admite.
- Controles de documentos dinámicos
  - FlowDocumentReader dispone de características que permiten al usuario elegir dinámicamente distintos modos de visualización, incluido el modo de visualización de una sola página (una página a la vez), dos páginas a la vez (formato de lectura de libro) y desplazamiento continuo (sin límite).
  - FlowDocumentPageViewer muestra el contenido en el modo de visualización de una sola página.
  - FlowDocumentScrollView muestra el contenido en modo de desplazamiento continuo.

© JMA 2012

# Elementos de documentos



© JMA 2012

## FlowDocument: XAML

```
<FlowDocumentReader xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <FlowDocument>
        <Paragraph>
            <Bold>Some bold text in the paragraph.</Bold>
            Some text that is not bold.
        </Paragraph>
        <List>
            <ListItem>
                <Paragraph>ListItem 1</Paragraph>
            </ListItem>
            <ListItem>
                <Paragraph>ListItem 2</Paragraph>
            </ListItem>
            <ListItem>
                <Paragraph>ListItem 3</Paragraph>
            </ListItem>
        </List>
    </FlowDocument>
</FlowDocumentReader>
```

© JMA 2012

## FlowDocument: C#

```
Paragraph myParagraph1 = new Paragraph(new Run("Paragraph 1"));
Paragraph myParagraph2 = new Paragraph(new Run("Paragraph 2"));
Paragraph myParagraph3 = new Paragraph(new Run("Paragraph 3"));
```

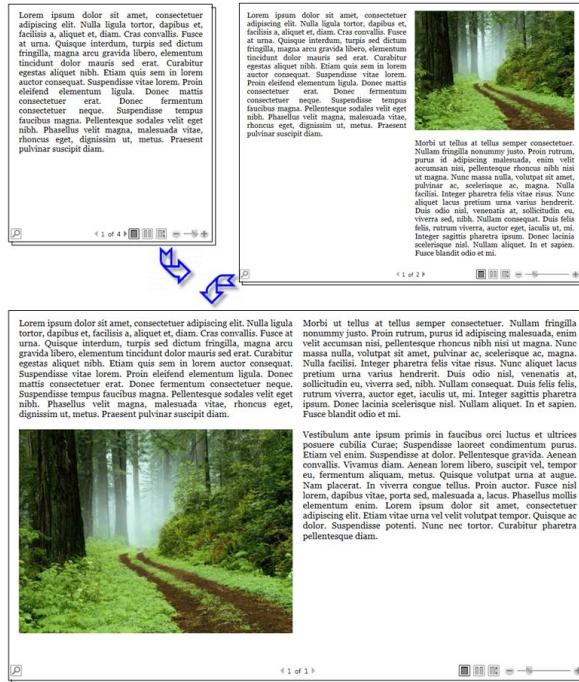
```
Section mySection = new Section();
mySection.Background = Brushes.Red;
```

```
mySection.Blocks.Add(myParagraph1);
mySection.Blocks.Add(myParagraph2);
mySection.Blocks.Add(myParagraph3);
```

```
FlowDocument myFlowDocument = new FlowDocument();
myFlowDocument.Blocks.Add(mySection);
```

```
this.Content = myFlowDocument;
```

© JMA 2012



© JMA 2012

## XAML Dinámico

```

FlowDocument doc = new FlowDocument();
doc.Padding = new Thickness(60, 100, 30, 30);
// Set PageHeight and PageWidth to "Auto".
doc.PageHeight = Double.NaN;
doc.PageWidth = Double.NaN;
// Specify minimum page sizes.
doc.MinPageWidth = 680.0;
doc.MinPageHeight = 480.0;
//Specify maximum page sizes.
doc.MaxPageWidth = 1024.0;
doc.MaxPageHeight = 768.0;

string sDoc = null;
if (t.Documentacion != null) {
    sDoc = @"<Section xmlns='http://schemas.microsoft.com/winf/2006/xaml/presentation'
Style='{StaticResource Detalle_Section}'>";
    sDoc+= (t.Documentacion.toXPS() + "</Section>";
doc.Blocks.Add((System.Windows.Markup.XamlReader.Parse(sDoc) as Section));
}
Visor.Document = doc;

```

© JMA 2012

DAIMLER  
EvoBus FORMADORES { IT }

## Encuesta de Satisfacción Online

<https://docs.google.com/forms/d/e/1FAIpQLSfq9vERNj16CkRaayvuFxK3-HDv2WuG9YZHbyRb4tos2jHEOg/viewform>

Formadores IT

MADRID – BARCELONA – BILBAO – VALENCIA – SEVILLA

info@formadoresfreelance.es  
www.formadoresfreelance.es

DAIMLER  
EvoBus

# FORMADORES { IT }

[www.formadoresfreelance.es](http://www.formadoresfreelance.es)

Consultores y formadores freelance