GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura i Esport
ceedcv CENTRE ESPECÍFIC D'EDUCACIÓ A DISTÀNCIA DE LA COMUNITAT VALENCIANA

DESARROLLO DE INTERFACES
PROPIEDADES DE DEPENDENCIA Y DATA BINDING

Departamento de Informática
Luis José Fortich Giner

Propiedades de dependencia

Los que hayan utilizado .NET antes, estaréis familiarizados con las propiedades y eventos que son parte central de todos los objetos de .NET. WPF, que es una tecnología para desarrollar interfaces, reemplaza las propiedades normales de .NET con las propiedades de dependencia. Utilizan un almacenamiento más eficiente y admiten características adicionales como la notificación de cambio y la herencia de valores de propiedad.

Las notificaciones de cambio hacen referencia a que si por ejemplo modificamos una propiedad de algún botón, WPF se encarga de avisar a otros elementos que lo necesitan, que se hizo un cambio de propiedad y entonces activará otra cosa, como por ejemplo, una animación.

La herencia de valores de propiedad ocurre cuando una de estas propiedades la definimos en una clase padre y queremos que se aplica a todos sus hijos, como por ejemplo el tamaño de letra o estilo de esta.

Creación de propiedades de dependencia

¿Para qué sirve crear propiedades de dependencia? Será necesario si deseamos agregar enlace de datos o binding, animación u otras características de WPF a una parte del código que de otro modo no lo admitiría.

Crear una propiedad de dependencia no es difícil. Solo podremos agregar propiedades de dependencia a objetos de dependencia, es decir, a clases que derivan de dependency object. Por suerte, la mayoría de las piezas claves de WPF derivan indirectamente de dependency object, siendo el ejemplo más obvio los elementos.

Ejemplo:

Tenemos un botón al cual establecemos su propiedad Margin.

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    myButton.Margin = new Thickness(20, 20, 20, 20);
}
```

Si vamos a la definición de Margin, nos va a llevar a las librerías WPF y podemos ver que es del tipo Thickness.

```
public Thickness Margin { get; set; }
```

Y si vamos más arriba en la definición, vemos que Margin es en realidad una dependency property.

```
public static readonly DependencyProperty MarginProperty;
```

Si vamos navegando hacia arriba por el árbol de herencia de nuestros elementos, podremos ver que estas propiedades son de tipo FrameworkElement, que a su vez heradan de UIElement, este de Visual y finalmente este hereda de DependencyObject.

```
public abstract class Visual : DependencyObject
```

Con esto, hemos visto que la clase FrameworkElement define una propiedad Margin que comparten todos los elementos. Por tanto, Margin es una propiedad de dependencia que se define en la clase Framework Element. Vemos que se define con el nombre de la propiedad y property al final, siendo en este caso MarginProperty. Se define como *static readonly*, lo que significa que solo se puede establecer en el constructor estático de Framework Element.

Registro de una propiedad de dependencia

Para que cualquier código utilice la propiedad, necesitamos crear una propiedad de dependencia.

¿Como usa WPF una propiedad de dependencia?

Contrariamente a lo que esperaríamos, las propiedades de dependencia no activan automáticamente eventos para informarnos cuando cambia el valor de una propiedad. En su lugar, desencadena un método protegido denominado OnPropertyChangeCallback. Este método pasa la información a dos servicios principales de WPF. Uno de ellos es el *binding* de datos y los *triggers*. En otras palabras, si deseamos reaccionar cuando una propiedad cambia, tenemos dos opciones: podemos crear un enlace o *binding* que utilice el valor de la propiedad o bien, escribir un desencadenante o *trigger* que cambie automáticamente otra propiedad.

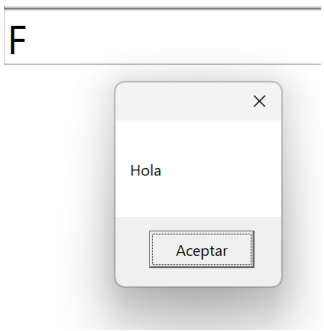
Por ejemplo, un TextBox proporciona un evento llamado TextChange, que ya viene implementado mediante el property change callback y está encapsulado en estos eventos. Para verlo más claro, veamos un ejemplo:

Creamos en nuestra ventana.xaml un TextBox y asignamos el evento TextChanged. Creamos también en el code-behind un método para crear un cuadro de diálogo que diga Hola cada vez que modifiquemos el TextBox.

```
<TextBox TextChanged="TextBox_TextChanged"/>
```

```
private void TextBox_TextChanged(object sender, TextChangedEventArgs e)
{
    MessageBox.Show("Hola");
}
```

El resultado es que, al introducir cualquier texto en este elemento, aparecerá un cuadro de diálogo con un Hola.



Realmente lo que está ocurriendo es que está exponiendo esta funcionalidad a través de un evento que en este caso es TextChanged, pero internamente llama a este property change callback.

Data Binding

Las propiedades de dependencia utilizan un almacenamiento más eficiente y admiten funciones adicionales como sería la notificación de cambios y la herencia de valores de la propiedad, es decir, tienen la capacidad de propagar valores predeterminados en el árbol de elementos. Por ejemplo, un botón, tiene una serie de controles o subelementos que lo conforman. Todos esos elementos de los que hereda son su árbol de elementos.

Estas propiedades son clave en el enlace de datos, estilos y animaciones. La mayoría de las propiedades que exponen los elementos de WPF son propiedades de dependencia. Posiblemente hemos estado trabajando con este tipo de propiedades sin darnos cuenta, ya que estas están diseñadas para consumirse de la misma manera que las propiedades normales. Sin embargo, estas propiedades de dependencia no son propiedades normales, si no que son necesarias para una variedad de características como acabamos de mencionar.

Al contrario de lo que esperamos, las propiedades de dependencia no activan eventos automáticamente para avisarnos cuando cambia el valor de una propiedad. En su lugar, activan un método protegido que se llama on property change callback. Este método pasa la información a 2 servicios de WPF: bindings y triggers.

Estos 2 servicios están activos siempre mientras estamos ejecutando una aplicación WPF. Si deseamos reaccionar cuando cambia una propiedad, tenemos dos opciones: podemos crear un enlace que use el valor de la propiedad, es decir, un *data binding* o podemos escribir un activador o *trigger* que cambie automáticamente otra propiedad e inicie una animación.

Por ejemplo, si estamos creando un control, podemos usar este mecanismo de devolución de llamada para reaccionar a los cambios de la propiedad e incluso generar un evento con todo esto. Muchos controles comunes usan esta técnica para las propiedades que corresponden a la información proporcionada por el usuario. Por ejemeplo, en los TextBox proporcionan un evento TextChanged. Estos eventos utilizan este mecanismo que acabamos de mencionar para reaccionar a los cambios de propiedad. Lo importante es entender que estamos utilizándolas para reaccionar a los cambios en dichas propiedades y así poder utilizar otras características como los *bindings o triggers*.

Enlace entre elementos

Para crear un enlace o *binding* tenemos que utilizar {Binding ElementName=”nombreControl”, Path=”NombrePropiedad”, Mode=TwoWay}, siendo:

- **ElementName:** el nombre del control al que hace referencia el enlace.
- **Path:** el nombre de la propiedad para establecer el enlace.
- **Mode:** este valor puede tomar diversos valores como OneWay, TwoWay, OneWayToSource o OneTime.

El valor OneWay es el valor por defecto, por lo que si no ponemos nada se ejecuta en este modo y hace referencia a que solo modifica en una dirección, un cambio desde el origen afecta al destino.

El valor TwoWays funciona en ambas direcciones, es decir, los cambios producidos tanto en origen como en destino afectan al otro.

El valor OneTime no es muy utilizado y sería propagar los cambios una única vez.

El valor OneWayToSource es parecido al OneWay pero a la inversa. Es decir, en OneWay la propiedad enlazada cambiaba la propiedad en la interfaz de usuario. En este caso, es al contrario, la propiedad en la interfaz de usuario cambia la propiedad de origen que tenemos enlazada.

- **UpdateSourceTrigger** es una propiedad clave en el data binding de WPF que controla cuándo se actualiza el valor de la propiedad de origen en el enlace. En otras palabras, especifica en qué momento se deben transferir los datos desde el destino (por ejemplo, un control en la interfaz de usuario) hacia el origen.

Los posibles valores para UpdateSourceTrigger son:

- **Default:** Este valor utiliza el comportamiento predeterminado, que depende del tipo de propiedad de destino. Por ejemplo, para TextBox.Text, el valor predeterminado es LostFocus. Para otros tipos de propiedades, puede variar.
- **PropertyChanged:** la propiedad de origen se actualiza tan pronto como cambia el valor del destino. Es decir, cada vez que el usuario escribe algo en un TextBox, la propiedad de origen se actualiza inmediatamente.
- **LostFocus:** la propiedad de origen se actualiza cuando el control pierde el foco. Es decir, la propiedad de origen se actualiza después de que el usuario haya terminado de editar el valor y ha hecho clic en otro lugar o ha presionado "Tab".
- **Explicit:** la actualización de la propiedad de origen debe ser desencadenada explícitamente, generalmente desde el código. Esto te da control completo sobre cuándo ocurre la actualización.
  Ejemplo:

```
<TextBox Text="{Binding Nombre, UpdateSourceTrigger=Explicit}" />
```

```
/* Actualizar explícitamente la propiedad de origen */
BindingExpression bindingExpression = textBox.GetBindingExpression(TextBox.TextProperty);
bindingExpression?.UpdateSource();
```

Enlace entre elementos mediante código

Podemos realizar *bindings* mediante código, de la siguiente forma:

```
Binding binding = new Binding();
binding.Source = sliderFontSize;
binding.Path = new PropertyPath("Value");
binding.Mode = BindingMode.TwoWay;
lblSampleText.SetBinding(TextBlock.FontSizeProperty, binding);
```

Para ver este ejemplo en acción, mirad la solución DataBindingEjemplos2.

Eliminar enlaces mediante código

Para eliminar los enlaces mediante código, podemos utilizar la propiedad:

```
BindingOperations.ClearAllBindings(lblSampleText);
```

Siendo lblSampleText el nombre de nuestro elemento que hemos enlazado previamente. Para más información, mirad el ejemplo en el proyecto “BindingCodigo” en la solución DataBindingEjemplos2.

Recuperar un enlace mediante código

Mediante el método GetBinding podemos obtener el enlace asignado a una propiedad. Este método recibe dos argumentos: el elemento enlazado y la propiedad que tiene la expresión de enlace.

Un ejemplo, que tenéis disponible en la solución DataBindingEjemplos2, es el siguiente:

```
Binding binding2 = BindingOperations.GetBinding(lblSampleText,
TextBlock.FontSizeProperty);
```