Titulo	**WPF - Propiedades de dependencia**
Descripción	Una Propiedad de dependencia es una propiedad de una clase, respaldada con un campo [DependencyProperty], para definir así una propiedad de dependencia. Estas propiedades admiten numerosos aspectos de la funcionalidad de Windows Presentation Foundation (WPF), incluidos los estilos, el enlace de datos, la herencia, la animación y los valores predeterminados, y , lo que quizá es más importante: no consumen memoria si no se les asigna un valor
Tabla de contenidos	WPF – Propiedades de dependencia Un ejemplo de implementación ¿Cuándo debe implementarse una propiedad de dependencia? Pasos para definir una propiedad de dependencia Registrar la propiedad con el sistema de propiedades Convenciones de nomenclatura Implementar el "contenedor" Propiedades de dependencia de sólo lectura Consideraciones de seguridad Propiedades de dependencia y constructores de clases
Type	Colección documentos sobre WPF
Autor	Joaquin Medina Serrano
Publisher	Joaquin Medina Serrano [administrador@joaquin.medina.name]
Rights	Copyright © 1997- 2014 All Rights Reserved La Güeb de Joaquín - Apuntes Tácticos - WPF Este documento tiene carácter público, puede ser copiado todo o parte siempre que se haga mención expresa de su procedencia
Fecha Creación	martes, 28 de enero de 2014
Fecha última Modificación	viernes, 31 de enero de 2014
Formato	txt/xhtml
Uri Recurso	http://joaquin.medina.name/web2008/documentos/informatica/lenguajes/puntoNET/System/Windows/WpfDocs/Docs/WPF_PropiedadesDependencia.pdf
Idioma	Es*es; Español, España

(Estándar Dublin Core [http://dublincore.org])

1. WPF – Propiedades de dependencia

1.1 Sumario

Una Propiedad de dependencia es una propiedad de una clase, respaldada con un campo [DependencyProperty], para definir así una propiedad de dependencia. Estas propiedades admiten numerosos aspectos de la funcionalidad de Windows Presentation Foundation (WPF), incluidos los estilos, el enlace de datos, la herencia, la animación y los valores predeterminados, y, lo que quizá es más importante: no consumen memoria si no se les asigna un valor

Contenido

1.2 Introducción

Una Propiedad de dependencia es una propiedad Normal (propiedad de "contenedor" de CLR) de una clase, respaldada con un campo [DependencyProperty], para definir así una propiedad de dependencia. Estas propiedades admiten numerosos aspectos de la funcionalidad de Windows Presentation Foundation (WPF), incluidos los estilos, el enlace de datos, la herencia, la animación y los valores predeterminados (o por defecto), y, lo que quizá es más importante: no consumen memoria si no se les asigna un valor

Es decir, cualquier propiedad que se enlace a cualquier control WPF, "puede" declararse como una propiedad de dependencia, y de ese modo se pueden modificar sus valores a través del código XAML de la ventana (la vista) que las usa

De hecho, las propiedades de los elementos WPF son realmente propiedades de dependencia

1.3 Un ejemplo de implementación

Ejemplo, una clase persona con propiedades de dependencia:

```
' Para INotifyPropertyChanged
Imports System.ComponentModel

''' <summary>
'''  Clase Pesona, ejemplo de propiedades de dependencia
''' </summary>
Public Class Persona
    Inherits DependencyObject

    ' ------------------------------------------------------------------
    ''' <summary>
    '''  Propiedad de dependencia Nombre
    ''' </summary>
    Public Shared ReadOnly NombreProperty As DependencyProperty = _
                  DependencyProperty.Register("Nombre", _
                          GetType(String), _
                          GetType(Persona), _
                          New PropertyMetadata(defaultValue:=String.Empty))

    ''' <summary>
    '''  El nombre de la persona
    ''' </summary>
    Public Property Nombre As String
        Get
            Return Convert.ToString(GetValue(NombreProperty))
        End Get

        Set(ByVal value As String)
                SetValue(NombreProperty, value)
        End Set
    End Property
```

```
    ' ------------------------------------------------------------------
    ''' <summary>
    '''  Propiedad de dependencia Edad
    ''' </summary>
    Public Shared ReadOnly EdadProperty As DependencyProperty = _
                  DependencyProperty.Register("Edad", _
                           GetType(Integer), _
                           GetType(Persona), _
                           New PropertyMetadata(defaultValue:=0I))

    ''' <summary>
    '''   La edad de una persona
    ''' </summary>
    Public Property Edad As Integer
        Get
            Return Convert.ToInt32(GetValue(NumeroProperty))
        End Get

        Set(ByVal value As Integer)
                SetValue(NumeroProperty, value)
        End Set
    End Property

End Class
```

En el ejemplo anterior, la clase Persona hereda de [DependencyObject], clase que ofrece la infraestructura necesaria para soportar propiedades de dependencia. Una propiedad de dependencia es un objeto estático que se registra en un diccionario con el método [DependencyProperty.Register]. Las instancias de un [DependencyObject] pueden acceder al valor de su propiedad de dependencia con el método [GetValue], y asignarle un valor con [SetValue] (ambos heredados de [DependencyObject]).

Las propiedades de dependencia no admiten de forma nativa [INotifyPropertyChanged]. Es decir, las propiedades de dependencia no generan notificaciones de cambios de valor de la propiedad de origen ***en operaciones de enlace de datos***.

Para obtener más información sobre cómo crear propiedades para su uso en enlace de datos que puedan informar de los cambios de un destino de enlace de datos, vea los siguientes enlaces MSDN.

- **Información general sobre el enlace de datos**
  - http://msdn.microsoft.com/es-es/library/ms752347(v=vs.110).aspx
- **Cómo: Implementar la notificación de cambio de propiedad**
  - http://msdn.microsoft.com/es-es/library/ms743695(v=vs.110).aspx

1.4 Snippets

Visual Studio 2013 nos ayuda con los denominados snippets, plantillas con fragmentos de código fácilmente utilizables. Si tecleamos [**wpfdp + Tab**] (por dependency property"), el editor nos ofrece una plantilla como ésta:

```
Public Property Prop1 As String
    Get
        Return GetValue(Prop1Property)
    End Get

    Set(ByVal value As String)
        SetValue(Prop1Property, value)
    End Set
End Property

Public Shared ReadOnly Prop1Property As DependencyProperty = _
                       DependencyProperty.Register("Prop1", _
                       GetType(String), GetType(Window1), _
                       New PropertyMetadata(Nothing))
```

Con la ayuda del tabulador podemos asignar valor a los campos de la plantilla (que aparecen con fondo amarillo en el código anterior).

A continuación muestro una implementación con la interfaz [INotifyPropertyChanged]

```
' Para INotifyPropertyChanged
Imports System.ComponentModel


''' <summary>
'''  Clase Pesona, ejemplo de propiedades de dependencia
''' </summary>
Public Class Persona
    Inherits DependencyObject
    Implements INotifyPropertyChanged


    Public Event PropertyChanged(sender As Object, e As PropertyChangedEventArgs) _
           Implements INotifyPropertyChanged.PropertyChanged


    ''' <summary>
    '''  Propiedad de dependencia Nombre
    ''' </summary>
    Public Shared ReadOnly NombreProperty As DependencyProperty = _
                  DependencyProperty.Register("Nombre", _
                           GetType(String), _
                           GetType(Persona), _
                           New PropertyMetadata(defaultValue:=String.Empty))

    ''' <summary>
    '''  El nombre de la persona
    ''' </summary>
    Public Property Nombre As String
        Get
            Return Convert.ToString(GetValue(NombreProperty))
        End Get

        Set(ByVal value As String)
            Dim oldValue As String = Convert.ToString(GetValue(NombreProperty))
            If oldValue <> value Then
                SetValue(NombreProperty, value)
                RaiseEvent PropertyChanged(Me, New PropertyChangedEventArgs("Nombre"))
            End If
        End Set
    End Property

End Class
```

2 Para saber mas

2.1 ¿Cuándo debe implementarse una propiedad de dependencia?

Al implementar una propiedad en una clase, siempre y cuando la clase se derive de [DependencyObject], se tiene la opción de respaldar su propiedad con un identificador [DependencyProperty] y, de este modo, convertir la propiedad en una propiedad de dependencia.

No siempre será necesario o adecuado convertir una propiedad en una propiedad de dependencia, sino que dependerá de las necesidades de su escenario. Sin embargo, debe implementar la propiedad como una propiedad de dependencia siempre que desee que la propiedad admita una o varias de las siguientes funciones de WPF:

- Desea que su propiedad pueda modificarse con un estilo.
- Desea que su propiedad admita el enlace de datos.
- Desea que su propiedad pueda usarse con una referencia de recurso dinámico.
- Desea heredar automáticamente el valor de una propiedad de un elemento primario del árbol de elementos.
- Desea que su propiedad soporte una animación.
- Desea que el sistema de propiedades notifique el momento en el que cambie el valor anterior de la propiedad mediante acciones realizadas por el sistema de propiedades, el entorno o el usuario, o mediante la lectura y el uso de estilos. El uso de metadatos de propiedad permite que su propiedad especifique un método de devolución de llamada que se invocará cada vez que el sistema de propiedades determine que el valor de la propiedad ha cambiado definitivamente. Un concepto relacionado esto último es la conversión del valor de las propiedades.
- Desea utilizar convenciones de metadatos establecidas que también se utilizan en los procesos de WPF, como notificar si el cambio de valor de una propiedad debe exigir que el sistema de diseño recomponga la representación visual de un elemento. O bien, desea poder utilizar invalidaciones para los metadatos de forma que las clases derivadas puedan modificar las características basadas en metadatos, como el valor predeterminado.
- Desea que las propiedades de un control personalizado sean compatibles con características de Visual Studio 2013 WPF Designer, como la edición en la ventana Propiedades.
- Más información en
    - **Propiedades de dependencia personalizadas**
    - http://msdn.microsoft.com/es-es/library/ms753358(v=vs.110).aspx

2.2 Pasos para definir una propiedad de dependencia

La definición de una propiedad de dependencia consta de cuatro pasos distintos. Estos pasos no representan cuatro líneas de código diferentes, porque algunos de ellos acaban combinándose en una sola línea de código en la implementación:

- (Opcional) Cree metadatos de propiedad para la propiedad de dependencia.
- Registre el nombre de la propiedad con el sistema de propiedades, especificando un tipo de propietario y el tipo de valor de propiedad. Especifique también los metadatos de la propiedad, si se utilizan.
- Defina un identificador [DependencyProperty] en el objeto que contendrá la propiedad:
  - [`Public Shared ReadOnly EdadProperty As DependencyProperty`]
- Defina una propiedad de "contenedor" de CLR (es decir, una propiedad estándar) cuyo nombre coincida con el nombre de la propiedad de dependencia. Y a continuación, implemente los descriptores de acceso [Get] y [Set] de la propiedad de "contenedor" de CLR para conectarla con la propiedad de dependencia que la respalda.

2.2.1 Registrar la propiedad con el sistema de propiedades

A continuación, deberá registrar la propiedad de dependencia en una tabla que mantiene el sistema de propiedades y asignarle un identificador único que se utilice como calificador para las operaciones posteriores del sistema de propiedades.

Para registrar la propiedad, llame al método [Register] dentro del cuerpo de su clase (dentro de la clase, pero fuera de las definiciones de miembros) La llamada al método [Register] también proporciona el campo de identificador como valor devuelto. La razón por la que la llamada a [Register] se realiza fuera de las demás definiciones de miembros es porque este valor devuelto se utiliza para asignar y crear un campo [`Public Shared ReadOnly`] de tipo [DependencyProperty] como parte de la clase. Este campo se convierte en el identificador para la propiedad de dependencia.

```
''' <summary>
'''  Propiedad de dependencia Edad
''' </summary>
Public Shared ReadOnly EdadProperty As DependencyProperty = _
             DependencyProperty.Register("Edad", _
                      GetType(Integer), _
                      GetType(Persona), _
                      New PropertyMetadata(defaultValue:=0I))
```

2.2.2 Implementar el "contenedor"

El contenedor de la propiedad de dependencia es en realidad una propiedad Normal (propiedad de “contenedor” de CLR), en la que se llama a [GetValue] en la implementación de [Get] y a [SetValue] en la implementación de [Set] Salvo en circunstancias excepcionales, sus implementaciones de contenedores sólo deberían realizar las acciones [GetValue] y [SetValue], respectivamente.

Todas las propiedades de dependencia públicas existentes que se incluyen en las clases de WPF utilizan este modelo de implementación de contenedor simple; la mayor parte de la complejidad del

funcionamiento de las propiedades de dependencia es inherentemente un comportamiento del sistema de propiedades o se implementa a través de otros conceptos, como la conversión o las devoluciones de llamada de cambio de propiedad a través de los metadatos de las propiedades.

```
''' <summary>
'''   La edad de una persona
''' </summary>
Public Property Edad As Integer
    Get
        Return Convert.ToInt32(GetValue(NumeroProperty))
    End Get

    Set(ByVal value As Integer)
            SetValue(NumeroProperty, value)
    End Set
End Property
```

De nuevo, por convención, el nombre de la propiedad de contenedor [Edad] debe coincidir con el nombre elegido y usado como primer parámetro de la llamada a [Register] que registró la propiedad. Si su propiedad no cumple esta convención, no necesariamente se deshabilitan todos los usos posibles, pero encontrará varios problemas importantes:

- No funcionarán determinados aspectos de los estilos y las plantillas.
- La mayoría de las herramientas y los diseñadores deben basarse en las convenciones de nomenclatura para serializar correctamente el XAML o para proporcionar ayuda sobre cada propiedad en el entorno de diseñador.

2.2.3 A modo de resumen

La forma completa de implementar una propiedad de dependencia es la siguiente:

```
''' <summary>
'''  Propiedad de dependencia Edad
''' </summary>
Public Shared ReadOnly EdadProperty As DependencyProperty = _
             DependencyProperty.Register("Edad", _
                     GetType(Integer), _
                     GetType(Persona), _
                     New PropertyMetadata(defaultValue:=0I))

''' <summary>
'''   La edad de una persona
''' </summary>
Public Property Edad As Integer
    Get
        Return Convert.ToInt32(GetValue(NumeroProperty))
    End Get

    Set(ByVal value As Integer)
            SetValue(NumeroProperty, value)
    End Set
End Property
```

2.3 Convenciones de nomenclatura de las propiedades de dependencia

Existen convenciones de nomenclatura establecidas para las propiedades de dependencia que siempre debe cumplir siempre salvo en circunstancias excepcionales.

La propiedad de dependencia propiamente dicha tendrá un nombre básico, (como " Edad ") y se proporciona como primer parámetro de [Register]. Este nombre debe ser único para cada "tipo" de registro (es decir, por cada objeto, en este ejemplo "Persona").

¡Atención!: Las propiedades de dependencia heredadas a través de los tipos base ya forman parte del tipo de registro; los nombres de las propiedades heredadas no pueden volver a registrarse.

Al crear el campo de identificador, asígnele el nombre de la propiedad tal y como lo registró, más el sufijo Property. Este campo es el identificador de la propiedad de dependencia y se utilizará después como entrada para las llamadas a [SetValue] y [GetValue] que se realizarán en los contenedores, por parte de cualquier otro acceso de código a la propiedad efectuado por el código propio, por parte de cualquier acceso de código externo permitido, por parte del sistema de propiedades y, posiblemente, por parte de los procesadores de XAML.

> Nota
>
> La implementación normal consiste en definir la propiedad de dependencia en el cuerpo de la clase, pero también es posible definir una propiedad de dependencia en el constructor estático de la clase. Este enfoque podría tener sentido es necesaria más de una línea de código para inicializar la propiedad de dependencia.

2.4 Propiedades de dependencia de sólo lectura

Puede definir una propiedad de dependencia que sea de sólo lectura. No obstante, los escenarios para los que debe definirse una propiedad de sólo lectura son algo distintos, así como el procedimiento utilizado para registrarlos con el sistema de propiedades y exponer el identificador.

Gran parte del proceso de creación de una propiedad de dependencia de sólo lectura es igual al de Creación de una propiedad de dependencia normal. Hay tres diferencias importantes:

- Al registrar la propiedad, llame al método [RegisterReadOnly] en lugar de al método [Register] normal para el registro de propiedades.
- Al implementar la propiedad "contenedor" de CLR, asegúrese de que tiene el calificador [ReadOnly], de tal forma que no haya ninguna incoherencia en el estado de sólo lectura para el contenedor público que se expone.
- El objeto devuelto por el registro de sólo lectura es [DependencyPropertyKey] en lugar de [DependencyProperty]. Este campo debe almacenarse igualmente como miembro, pero no es habitual convertirlo en un miembro público del tipo.

Dado que [DependencyPropertyKey] es privada, y que el sistema de propiedades no la propaga fuera del código, una propiedad de dependencia de sólo lectura aporta mayor seguridad de establecimiento que una propiedad de dependencia de lectura y escritura. Para una propiedad de dependencia de lectura y escritura, el campo de identificación es público de manera explícita o implícita, con lo que la propiedad se puede establecer ampliamente.

```
Public ReadOnly Property Apellidos As String
    Get
        Return CStr(GetValue(Persona.ApellidosProperty))
    End Get
End Property

Private Shared ReadOnly ApellidosPropertyKey As DependencyPropertyKey = _
                        DependencyProperty.RegisterReadOnly("Apellidos", _
                        GetType(String), _
                        GetType(Persona), _
                        New PropertyMetadata(Nothing))

Public Shared ReadOnly ApellidosProperty As DependencyProperty = _
                       ApellidosPropertyKey.DependencyProperty
```

2.4.1 Snippets

Si tecleamos [**wpfdpro + Tab**] (por dependency property read only"), el editor nos ofrece una plantilla como ésta:

```
Public ReadOnly Property Prop1 As String
        Get
            Return GetValue(Window1.Prop1Property)
        End Get
    End Property

    Private Shared ReadOnly Prop1PropertyKey As DependencyPropertyKey = _
                            DependencyProperty.RegisterReadOnly("Prop1", _
                            GetType(String), _
                            GetType(Window1), _
                            New PropertyMetadata(Nothing))

    Public Shared ReadOnly Prop1Property As DependencyProperty = _
                           Prop1PropertyKey.DependencyProperty
```

Con la ayuda del tabulador podemos asignar valor a los campos de la plantilla (que aparecen con fondo amarillo en el código anterior).

2.5 Consideraciones de seguridad de las propiedades de dependencia

Las propiedades de dependencia deben declararse como propiedades públicas. Los campos de identificador de las propiedades de dependencia deben declararse como campos estáticos públicos.

2.6 Propiedades de dependencia y constructores de clases

Para evitar posibles problemas con la inicialización en tiempo de ejecución, no debe establecer los valores de propiedad de dependencia dentro de los constructores de clases

Existe un principio general en programación de código administrado (que suelen aplicar las herramientas de análisis del código como FxCop) según el cual los constructores de clases no deben llamar a métodos virtuales. Esto se debe a que es posible llamar a constructores como inicialización base de un constructor de clases derivadas, y podría entrarse en el método virtual a través del constructor en un estado de inicialización incompleto de la instancia del objeto que se está construyendo.

2.7 Referencia Bibliográfica

Desarrollo de aplicaciones con .NET y WPF

- Andrés Marzal
  - Departamento de Lenguajes y Sistemas Informáticos
  - Universidad Jaume I
  - De charlas, 24 de mayo de 2010
  - http://decharlas.uji.es/resources/dotnet/wpf.pdf

- **Modelos de constructores seguros para objetos DependencyObject**
  - http://msdn.microsoft.com/es-es/library/ms754149(v=vs.110).aspx
- **Información general sobre el enlace de datos**
  - http://msdn.microsoft.com/es-es/library/ms752347(v=vs.110).aspx
- **Cómo: Implementar la notificación de cambio de propiedad**
  - http://msdn.microsoft.com/es-es/library/ms743695(v=vs.110).aspx
- **Propiedades de dependencia personalizadas**
  - http://msdn.microsoft.com/es-es/library/ms753358(v=vs.110).aspx

(Página dejada intencionalmente en blanco)