Titulo	**WPF – Implementación interfaz [INotifyPropertyChanged]**
Descripción	La interfaz [INotifyPropertyChanged] permite que una propiedad de un objeto notifique que el valor de la propiedad ha cambiado, de forma que se pueda mostrar información actualizada.
Tabla de contenidos	Implementación básica Definir la clase. Las variables de la clase El evento [PropertyChanged]. La función [OnPropertyChanged] El código completo de la implementación básica Implementación Avanzada Comprobación de la existencia de la propiedad Otras formas de incluir el nombre de la propiedad Mediante reflexión Usando el atributo [CallerMemberNameAttribute] La nueva función [OnPropertyChanged] mejorada
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	miércoles, 29 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_ImplementacionINotifyPropertyChanged.pdf
Idioma	Es-es; Español, España

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

1 WPF – Implementación interfaz [INotifyPropertyChanged]

1.1 Sumario

La interfaz [INotifyPropertyChanged] permite que una propiedad de un objeto notifique que el valor de la propiedad ha cambiado, de forma que se pueda mostrar información actualizada.

Contenido

1 WPF – Implementación interfaz [INotifyPropertyChanged] 3
1.1 Sumario 3
1.2 Introducción 4
1.3 Implementación básica 4
1.3.1 Definir la clase. 4
1.3.2 Las variables de la clase 4
1.3.3 El evento [PropertyChanged]. 5
1.3.4 La función [OnPropertyChanged] 5
1.3.5 El código completo de la implementación básica 6
1.3.6 ¿Cómo se usa? 7
1.4 Implementación Avanzada 8
1.5 Comprobación de la existencia de la propiedad 8
1.6 Otras formas de incluir el nombre de la propiedad 9
1.6.1 Mediante reflexión 9
1.6.2 Usando el atributo [CallerMemberNameAttribute] 12
1.7 La nueva función [OnPropertyChanged] mejorada 13
1.8 Código completo de la clase 14
1.9 Referencia bibliográfica 14

1.2 Introducción

En el patrón MVVM, todos los objetos enlazados a una vista implementan la interfaz [INotifyPropertyChanged]. Para evitar repetir código (DRY: Don't Repeat Yourself (¡No Te Repitas!), he escrito una clase (diseñada para ser usada mediante herencia), que contiene toda la implementación de esa interfaz.

1.3 Implementación básica

1.3.1 Definir la clase.

```
''' <summary>
'''  Clase que encapsula la interfaz [INotifyPropertyChanged]
''' </summary>
<Serializable()> _
Public MustInherit Class ImplementacionINotifyPropertyChanged
    Implements System.ComponentModel.INotifyPropertyChanged

    Protected Sub New()
        ' no hacer nada
    End Sub
```

1.3.2 Las variables de la clase

En realidad no hace falta ninguna, pero a efectos informativos guardamos el nombre de la última propiedad que ha cambiado. Su valor se cambia en la función [OnPropertyChanged] que es la encargada de disparar el evento [PropertyChanged]

```
    ''' <summary>
    '''  Variable de clase para el nombre de la propiedad que ha cambiado
    ''' </summary>
    Private _notifyNombrePropiedadQueCambia As String = String.Empty

    ''' <summary>
    '''  El nombre de la propiedad que ha cambiado
    ''' </summary>
    Public ReadOnly Property NotifyNombrePropiedadQueCambia As String
        ' Se carga en la función [OnPropertyChanged]
        Get
            Return _notifyNombrePropiedadQueCambia
        End Get
    End Property
```

1.3.3 El evento [PropertyChanged].

```
'-----------------------------------------------------------------------
''' <summary>
'''  Evento [PropertyChanged] de la interfaz [INotifyPropertyChanged]
''' </summary>
<NonSerializedAttribute()> _
Public Event PropertyChanged As  _
        System.ComponentModel.PropertyChangedEventHandler _
        Implements System.ComponentModel.INotifyPropertyChanged.PropertyChanged
```

Observaciones sobre esta declaración.

- Microsoft recomienda declarar los eventos como eventos genéricos, sin embargo, no se pueden serializar los datos genéricos, por esa razón, recurro a la declaración tradicional del evento. Más información en:
  - http://msdn.microsoft.com/es-es/db0etb8x.aspx
  - http://msdn.microsoft.com/es-es/ms182178.aspx
- En segundo lugar, en realidad si se serializa la clase no me interesa para nada serializar este evento, por eso se marca como [NonSerializedAttribute]

1.3.4 La función [OnPropertyChanged]

La función [OnPropertyChanged] que dispara el evento. Esta es la función que debe ser llamada por las propiedades para disparar el evento [PropertyChanged]

```
'-----------------------------------------------------------------------
''' <summary>Función que dispara el evento [PropertyChanged]</summary>
''' <param name="nombreDeLaPropiedadChanged">
'''         Se espera una cadena de texto con el nombre
'''         de la propiedad que cambia
''' </param>
Protected Overloads Sub OnPropertyChanged(
          ByVal nombreDeLaPropiedadChanged As String)
    ' Evitar problemas tontos de cadenas vacias
    If String.IsNullOrWhiteSpace(nombreDeLaPropiedadChanged) = False Then
        ' Actualizar el nombre de la propiedad que cambia
        _nombrePropiedadQueCambia = nombreDeLaPropiedadChanged
        ' Disparar el evento
        RaiseEvent PropertyChanged( _
           Me,
           New System.ComponentModel.PropertyChangedEventArgs(nombreDeLaPropiedadChanged))
    End If
End Sub
```

1.3.5 El código completo de la implementación básica

```
''' <summary>
'''  Clase que encapsula la interfaz [INotifyPropertyChanged]
''' </summary>
<Serializable()> _
Public MustInherit Class ImplementacionINotifyPropertyChanged
    Implements System.ComponentModel.INotifyPropertyChanged

   ' Un constructor
    Protected Sub New()
        ' no hacer nada
    End Sub

    '----------------------------------------------------------------------
    ''' <summary>
    '''  Variable de clase para el nombre de la propiedad que ha cambiado
    ''' </summary>
    Private _notifyNombrePropiedadQueCambia As String = String.Empty

    ''' <summary>
    '''  El nombre de la propiedad que ha cambiado
    ''' </summary>
    Public ReadOnly Property NotifyNombrePropiedadQueCambia As String
        ' Se carga en la función [OnPropertyChanged]
        Get
            Return _notifyNombrePropiedadQueCambia
        End Get
    End Property

    '----------------------------------------------------------------------
    <NonSerializedAttribute()> _
    Public Event PropertyChanged As  _
              System.ComponentModel.PropertyChangedEventHandler _
              Implements System.ComponentModel.INotifyPropertyChanged.PropertyChanged


'----------------------------------------------------------------------
 ''' <summary>Función que dispara el evento [PropertyChanged]</summary>
 ''' <param name="nombreDeLaPropiedadChanged">
 '''  Se espera una cadena de texto con el nombre
 '''  de la propiedad que cambia
 ''' </param>
 Protected Overloads Sub OnPropertyChanged(
           ByVal nombreDeLaPropiedadChanged As String)
     ' Evitar problemas tontos de cadenas vacias
     If String.IsNullOrWhiteSpace(nombreDeLaPropiedadChanged) = False Then
         ' Actualizar el nombre de la propiedad que cambia
         _nombrePropiedadQueCambia = nombreDeLaPropiedadChanged
         ' Disparar el evento
         RaiseEvent PropertyChanged( _
            Me,
            New System.ComponentModel.PropertyChangedEventArgs(nombreDeLaPropiedadChanged))
     End If
 End Sub


End Class
```

1.3.6 ¿Cómo se usa?

Para usar esta clase tenemos que usarla como clase base de la clase que queramos que notifique los cambios de las propiedades, por ejemplo en una clase [Persona]

```
Public Class Colega
    Inherits ImplementacionINotifyPropertyChanged

    '  Variable interna para la propiedad [Nombre], El nombre del colega
    Private _nombre As String = String.Empty


    ''' <summary>
    '''  El nombre del colega
    ''' </summary>
    Public Property Nombre As String
        Get
            Return _nombre
        End Get

        Set(ByVal value As String)
            If _nombre <> value Then
                _nombre = value
                Call Me.OnPropertyChanged("Nombre")
            End If
        End Set
    End Property

End Class
```

Observa que en la propiedad solo tenemos que llamar a la función [OnPropertyChanged] de la clase base [ImplementacionINotifyPropertyChanged] que es la que se encarga de disparar el evento que notifica que la propiedad [Nombre] ha cambiado.

1.4 Implementación Avanzada

1.5 Comprobación de la existencia de la propiedad

Una vez mostrada la implementación básica de esta clase, podemos mejorarla de varias formas, la primera que se nos ocurre es que no se comprueba que exista el nombre de la propiedad en la clase, es decir, existe la posibilidad de que nos equivoquemos al escribir el nombre de la propiedad.

Para ello vamos a escribir una función que se encargue de comprobar si el nombre de la propiedad recibido corresponde o no con una propiedad de la clase derivada. Por otra parte, esta comprobación de seguridad solo debe hacerse en el proceso de depuración, pero no en el proceso de explotación, porque varias razones, la primera es que no hace falta comprobar permanentemente que la propiedad pertenece a la clase o no, una vez que se ha comprobado la primera vez las demás veces sobran. Y la segunda es que por poco tiempo que se use en realizar esta tarea, se está usando en realizar una comprobación innecesaria por lo que la aplicación se ralentizara innecesariamente.

```
''' <summary>
'''  Verifica que el nombre de la propiedad exista en la clase
'''  La propiedad debe estar calificada como [Public].
''' </summary>
<ConditionalAttribute("DEBUG"),
 DebuggerStepThrough()> _
Private Shared Sub VerificarExistenciaNombrePropiedad(
                   ByVal unaClase As Object,
                   ByVal nombreDeLaPropiedadChanged As String)

    '------------------------------------------------------
    'Colección con las propiedades de la clase derivada
    Dim properties As System.ComponentModel.PropertyDescriptorCollection =
                    System.ComponentModel.TypeDescriptor.GetProperties(unaClase)
    '------------------------------------------------------
    ' Evitar problemas con valores Null
    If Not (properties Is Nothing) Then
        ' Comprobar si existe la propiedad
        If (properties(nombreDeLaPropiedadChanged) Is Nothing) Then
            '-------------------------------
            ' La propiedad no existe
            ' Disparar el error personalizado de propiedad no existe
            ' en la clase derivada y montar un mensaje explicativo extenso
            '-------------------------------
            ' el nombre de la clase base es el nombre de esta clase
            ' [ImplementacionINotifyPropertyChanged]
            Dim nombreClaseBase As String =
                System.Reflection.MethodBase.GetCurrentMethod.DeclaringType.Name
            Throw New InvalidPropertyNameException(
                      nombreClaseBase,
                      unaClase,
                     nombreDeLaPropiedadChanged)
        End If
        properties = Nothing
    End If
End Sub
```

1.5.1.1 Observaciones:

Esta función primero guarda en una colección todas las propiedades de la clase y a continuación comprueba si el nombre recibido coincide con el de alguna propiedad. Si no coincide con ninguna dispara un error

La función está marcada con dos atributos que no son muy frecuentes:

- El atributo [ConditionalAttribute("DEBUG")] indica que la función se incluirá siempre en la aplicación compilada; sin embargo, las llamadas al procedimiento sólo se incluirán si se ha definido la constante de compilación y si su valor es distinto de cero. En caso contrario, se descartarán estas llamadas. En este caso concreto quiere decir que la función solo se ejecutara cuando se compile en modo [Debug]. Cuando se compile en modo [Release] no se ejecutara aunque se la llame en el código. Observa que si ha habido un error tipográfico al escribir el nombre de la función, la primera vez que se ejecute el programa se detectara y se disparara el error personalizado. Cuando vayamos a compilar la aplicación en modo [Release], seguro que todo esos errores tipográficos habrán sido corregidos
- El atributo [**DebuggerStepThrough**] significa que cuando se ejecuta paso a paso pulsando la tecla [F8], paso a paso por instrucciones este código se comporta como [Mayus + F8] Paso a paso por procedimientos] es decir no se puede establecer un punto de interrupción en el interior de la función
- Una de copyright.
  - La idea de esta función la tuve leyendo este artículo en la que se describe la forma de hacer esta misma tarea, aunque en otro contexto de trabajo.
  - WPF Apps With The Model-View-ViewModel Design Pattern
    - http://msdn.microsoft.com/es-es/magazine/dd419663.aspx

1.6 Otras formas de incluir el nombre de la propiedad

1.6.1 Mediante reflexión

Ya he comentado que uno de los problemas principales de esta implementación es que existe la posibilidad de que exista un error tipográfico a la hora de escribir el nombre de la propiedad. Este problema, también puede estar originado por la refactorización de los nombres de las propiedades, ya que el proceso de refactorización no cambia los literales.

Por ejemplo

```
'' <summary>
'''  El nombre del colega
''' </summary>
Public Property Nombre As String
    Get
        Return _nombre
    End Get
```

```
    Set(ByVal value As String)
        If _nombre <> value Then
            _nombre = value
             Call Me.OnPropertyChanged("NombrePropiedadErroneo")
        End If
    End Set
End Property
```

Podemos resolverlo usando la reflexión para obtener el nombre de la propiedad que cambia, por ejemplo.

```
    Set(ByVal value As String)
        If _nombre <> value Then
            _nombre = value
             Call Me.OnPropertyChanged(System.Reflection.MethodBase.GetCurrentMethod.Name)
        End If
    End Set
End Property
```

Pero este enfoque, aunque evita todos los problemas de tipografía y los originados por las refactorizaciones presenta otro problema, y es que el nombre de la propiedad que devuelve está precedida por la cadena [Set_] es decir, el nombre que devuelve la función que usa reflexión es [Set_Nombre] que evidentemente tampoco se corresponde con el autentico nombre de la propiedad.

Esta diferencia es importante cuando se está trabajando con ventanas WPF y enlaces a datos, porque esa diferencia de nombre es suficiente para que los enlaces a datos no funcionen.

La forma de resolverlo es escribir una función que elimine esas cadenas de los nombres de las propiedades

```
    ''' <summary>
    '''    Quita el prefijo [Get_] y/o [Set_] del nombre de la propiedad si existe
    ''' </summary>
    ''' <param name="nombreDeLaPropiedadChanged">
    '''         El nombre de la propiedad que cambia</param>
    ''' <returns>La propiedad corregida</returns>
    ''' <remarks>
    '''    Si obtengo el nombre de la propiedad que cambia mediante reflexión,
    '''    Por ejemplo con: [System.Reflection.MethodBase.GetCurrentMethod.Name]
    '''    el nombre de la propiedad que se obtiene esta precedido
    '''    por el prefijo [get_], o [set_]
    '''</remarks>
    ''' <exception cref="InvalidPropertyNameException">
    '''  Hay un error de de logica. No se puede lanzar un evento [Changed]
    '''  desde un constructor, desde un constructor estatico, o desde un desctructor
    '''</exception>
```

```
Private Shared Function CorregirNombrePropiedad(
                        ByVal nombreDeLaPropiedadChanged As String) As String

    ' El valor de retorno (por defecto)
    ' por si es null, o cadenas de espacios
    Dim auxCadena As String = String.Empty

    If String.IsNullOrWhiteSpace(nombreDeLaPropiedadChanged) = False Then
        ' este es el valor recibido que se devuelve si esta correcto
        auxCadena = nombreDeLaPropiedadChanged

        ' buscar [Get] y/o [Set]
        Dim nombreMayus As String =
            nombreDeLaPropiedadChanged.ToUpper(
                    System.Globalization.CultureInfo.CurrentCulture)

        'Corregir los nombres de miembro que se devuelven cuando se utiliza
        ' reflexión para obtener el nombre de la propiedad.

        If nombreMayus.StartsWith("SET_",
                    StringComparison.CurrentCultureIgnoreCase) = True OrElse
           nombreMayus.StartsWith("GET_",
                    StringComparison.CurrentCultureIgnoreCase) = True Then
            auxCadena = auxCadena.Remove(0, 4)
        End If

        '--------------------------------------------------------------------------
        'Observación
        '   hay que declarar un error porque es un error de lógica del programa
        '   no se puede lanzar un evento Changed desde un constructor o un destructor
        '--------------------------------------------------------------------------
        ' Constructor   La cadena ".ctor"
        If nombreMayus.StartsWith(".CTOR",
                           StringComparison.CurrentCultureIgnoreCase) = True Then
            Throw New InvalidPropertyNameException(
                    "Error de lógica. No se puede lanzar " & _
                    "un evento [Changed] desde un constructor")
        End If
        'Constructor estático  La cadena ".cctor"
        If nombreMayus.StartsWith(".CCTOR",
                            StringComparison.CurrentCultureIgnoreCase) = True Then
            Throw New InvalidPropertyNameException(
                    "Error de lógica. No se puede lanzar  " & _
                    "un evento [Changed] desde un constructor estático")
        End If
        'Un destructor  La cadena "Finalize"
        If nombreMayus.StartsWith("FINALIZE",
                             StringComparison.CurrentCultureIgnoreCase) = True Then
            Throw New InvalidPropertyNameException(
                    "Error de lógica. No se puede lanzar  " & _
                    "un evento [Changed] desde un destructor")
        End If
    End If
    Return auxCadena
End Function
```

1.6.2 Usando el atributo [CallerMemberNameAttribute]

En la documentación MSDN de Microsoft se dice:

Se puede utilizar el atributo [CallerMemberName] para evitar especificar el nombre de miembro como argumento String. Mediante esta técnica, se evita el problema de que un proceso de Refactorización de cambio de nombre no cambie los valores String. Esta ventaja es especialmente útil para las siguientes tareas:

- Usar seguimiento y rutinas de diagnóstico.
- Implementar la interfaz INotifyPropertyChanged al enlazar datos. Esta interfaz permite que la propiedad de un objeto notifique a un control enlazado que la propiedad ha cambiado, de forma que se pueda mostrar información actualizada. Sin el atributo CallerMemberName, se debe especificar el nombre de propiedad como un literal.
- En esta dirección de MSDM se amplía este concepto
    - Información del llamador (C# y Visual Basic)
    - http://msdn.microsoft.com/es-es/library/hh534540.aspx

Para poder usar el atributo [CallerMemberNameAttribute], hay que modificar la función [OnPropertyChanged] que es la que dispara el evento [PropertyChanged] de la forma que se muestra a continuación, declarando el atributo en el parámetro de la función. Además, hay que declararlo como [Optional] y darle un valor por defecto (en este caso [Nothing])

Una observación: un parámetro opcional requiere una constante para inicializarse y el valor [String.Empty] no es una constante

```
'-----------------------------------------------------------------------
''' <summary>Función que dispara el evento [PropertyChanged]</summary>
''' <param name="nombreDeLaPropiedadChanged">
'''  Se espera una cadena de texto con el nombre
'''  de la propiedad que cambia
''' </param>
Protected Overloads Sub OnPropertyChanged(
         <System.Runtime.CompilerServices.CallerMemberNameAttribute> _
                 Optional ByVal nombreDeLaPropiedadChanged As String = Nothing)
    '---------------------------------------
    ' Evitar problemas tontos (valores Nothing)
    If String.IsNullOrWhiteSpace(nombreDeLaPropiedadChanged) = False Then
        ' actualizar el nombre de la propiedad que cambia
        _nombrePropiedadQueCambia = nombreDeLaPropiedadChanged
        ' Disparar el evento
        RaiseEvent PropertyChanged( _
          Me,
          New System.ComponentModel.PropertyChangedEventArgs(nombreDeLaPropiedadChanged))
    End If
End Sub
```

1.7 La nueva función [OnPropertyChanged] mejorada

Queda un último problemilla, a pesar de todo, las Leyes de Murphy dicen, entre otras cosas [Si algo puede fallar, fallará], por eso al final ha decidido integrar todo lo que hemos visto en este documento en la función [OnPropertyChanged] para asegurarme de que el código de la clase es robusto y que se use como se use se evitan todos los errores posibles. Por lo menos los que a mí se me han ocurrido. Aunque hay que tener en cuenta que las leyes de Murphy siempre están trabajando

- http://www2.uah.es/vivatacademia/anteriores/quince/leyesde.htm

```
 '-----------------------------------------------------------------------
''' <summary>Función que dispara el evento [PropertyChanged]</summary>
''' <param name="nombreDeLaPropiedadChanged">
'''  <para>Se espera una cadena de texto con uno de los siguientes valores:</para>
'''  <para> a) Una cadena vacía, en este caso la función averiguara el
'''            nombre de la propiedad a través de su
'''            atributo [CallerMemberName]]</para>
'''  <para> b) Una cadena de texto con el nombre de
'''            la propiedad (Ejemplo, "Apellidos") </para>
'''  <para> c) Una cadena proporcionada por reflexión
'''            [System.Reflection.MethodBase.GetCurrentMethod.Name]</para>
''' </param>
''' <exception cref="InvalidPropertyNameException">
'''    <para> Si el nombre de la propiedad no existe en la clase</para>
'''    <para> o bien si</para>
'''    <para> Hay un error de lógica. No se puede lanzar un evento [Changed]
'''           desde un constructor, desde un constructor estático,
'''           o desde un destructor</para>
''' </exception>
Protected Overloads Sub OnPropertyChanged(
    <System.Runtime.CompilerServices.CallerMemberNameAttribute>
              Optional ByVal nombreDeLaPropiedadChanged As String = Nothing)
    ' --------------------------------
    ' Evitar problemas tontos
    If String.IsNullOrWhiteSpace(nombreDeLaPropiedadChanged) = False Then
        ' --------------------------------
        ' Quitar los  prefijos [Get_], o [Set_]
        nombreDeLaPropiedadChanged = CorregirNombrePropiedad(nombreDeLaPropiedadChanged)
        ' --------------------------------
        ' Vuelvo a preguntar porque después de las correcciones
        ' puede venir una cadena vacía y así evito un error
        If String.IsNullOrWhiteSpace(nombreDeLaPropiedadChanged) = False Then
            ' --------------------------------
            ' Comprobar que el nombre de la propiedad existe en la clase
            Call VerificarExistenciaNombrePropiedad(Me, nombreDeLaPropiedadChanged)
            ' --------------------------------
            ' Actualizar el nombre de la propiedad que cambia
            _nombrePropiedadQueCambia = nombreDeLaPropiedadChanged
            ' Disparar el evento
          RaiseEvent PropertyChanged( _
           Me,
           New System.ComponentModel.PropertyChangedEventArgs(nombreDeLaPropiedadChanged))
        End If
    End If
End Sub
```

1.8 Código completo de la clase

Puedes descargarte el código completo de esta clase en formato [ZIP] desde este enlace

```
-----------------------------------
http://joaquin.medina.name/web2008/documentos/informatica/lenguajes/puntoNET/System/Windows
/WpfDocs/Docs/Wpf_ImplementacionINotifyPropertyChanged.zip

Nombre Fichero    [Wpf_ImplementacionINotifyPropertyChanged.zip]
Tamaño en bytes   [6721]
Md5Hash           [C2CF06378B272593484BC1E67C8CEB9F]
-----------------------------------
```

1.9 Referencia bibliográfica

- **Información del llamador (C# y Visual Basic)**
- http://msdn.microsoft.com/es-es/library/hh534540.aspx
- http://msdn.microsoft.com/es-es/library/system.runtime.compilerservices.callermembernameattribute(v=vs.110).aspx

- **INotifyPropertyChanged (Interfaz)**
- http://msdn.microsoft.com/es-es/library/system.componentmodel.inotifypropertychanged.aspx

- **Leyes de Murphy**
- http://www2.uah.es/vivatacademia/anteriores/quince/leyesde.htm