La Güeb de Joaquín
Mi sitio de Internet

Apuntes Tácticos

Atributos .NET

Información del documento

	La Güeb de Joaquin (http://joaquin.medina.name)
Categoría	Programación .NET
Tema.:	Atributos .NET
Titulo	Atributos .NET
Autor.:	Joaquin Medina Serrano (mailto:joaquin@medina.name)
Archivo	Informatica/Lenguajes/.NET/System/ComponentModel
Sumario	Puede pensar en los atributos como anotaciones que realiza el código. Podrá aplicar atributos a casi cualquier entidad de programación que se pueda utilizar en Visual Basic NET, incluyendo clases, módulos, estructuras, propiedades, métodos, bloque de enumeración, etc .NET Framework utiliza atributos por varias razones y para tratar una serie de problemas. Los atributos describen cómo se serializan datos, se especifican características que se utilizan para exigir seguridad y se limitan optimizaciones mediante el compilador Just-in-time (JIT) para facilitar la depuración del código. También pueden grabar el nombre de un archivo o el autor del código, o controlar la visibilidad de controles y miembros durante el desarrollo de formularios.
Numero de paginas	24
Fecha de creación.	09/01/2010 18:31:00
Fecha última modificación.	22/01/2010 12:32:00

Sobre el autor

- Joaquín Medina Serrano [Web] [Mail]
- Fotógrafo erótico conceptual
- Autor de libros sobre Visual Basic, Cobol y Office
- He desarrollado diversas aplicaciones de gestión, utilizando herramientas como Clipper, Visual Basic 6 y Visual Basic .NET.
- Autor de artículos sobre Programación Orientada a Objetos, y Visual Basic .NET
- El último curso de capacitación realizado ha sido el Máster en Servicios Web, Seguridad Informática y Aplicaciones de Comercio Electrónico de la [Universidad de Zaragoza]

Contenido

1 Atributos .NET

1.1 Información general sobre atributos

Es probable que esté familiarizado con declaraciones que contienen palabras clave, como public y private, que proporcionan información adicional sobre miembros de clase. Estas palabras clave definen además el comportamiento de los miembros de clase mediante la descripción de la accesibilidad a ella desde otras clases. Los compiladores se diseñan explícitamente para reconocer palabras clave predefinidas, normalmente, no se tiene la oportunidad de crear palabras clave propias. Sin embargo, Common Language Runtime permite agregar declaraciones descriptivas a modo de palabras clave, denominadas atributos, para anotar elementos de programación como tipos, campos, métodos y propiedades.

Cuando se compila el código para el motor en tiempo de ejecución, se convierte a lenguaje intermedio de Microsoft (MSIL) y se coloca en un archivo ejecutable portable (PE) junto con los metadatos generados por el compilador. Los atributos permiten colocar información descriptiva adicional en metadatos que se pueden extraer utilizando servicios de reflexión en tiempo de ejecución. El compilador crea atributos cuando se declaran instancias de clases especiales que se derivan de System.Attribute.

.NET Framework utiliza atributos por varias razones y para tratar una serie de problemas. Los atributos describen cómo se serializan datos, se especifican características que se utilizan para exigir seguridad y se limitan optimizaciones mediante el compilador Just-in-time (JIT) para facilitar la depuración del código. También pueden grabar el nombre de un archivo o el autor del código, o controlar la visibilidad de controles y miembros durante el desarrollo de formularios.

Se pueden utilizar atributos para describir el código en cualquier forma imaginable e influir en el comportamiento en tiempo de ejecución mediante nuevas formas creativas. Los atributos permiten agregar elementos descriptivos propios a C#, Visual C++, Microsoft Visual Basic 2005 o a cualquier otro lenguaje orientado a Common Language Runtime sin tener que volver a crear el compilador.

```
Imports System.ComponentModel
```

1.2 Sintaxis de los atributos

Puede pensar en los atributos como anotaciones que realiza el código.

Podrá aplicar atributos a casi cualquier entidad de programación que se pueda utilizar en Visual Basic NET, incluyendo clases, módulos, estructuras, propiedades, métodos, bloque de enumeración, etc.

En Visual Basic .NET deberá encerrar el atributo entre paréntesis angulares (<>) y deberá introducirlo inmediatamente antes del elemento al que haga referencia. Por ejemplo, podrá aplicar System.ComponentModel.DescriptionAttribute a una clase en la forma siguiente:

```
<System.ComponentModel.DescriptionAttribute("Una persona")> _
Class Persona
    ' . . .
End Class
```

Podrá simplificar la sintaxis de los atributos de diversas maneras.

- En primer lugar, debido a que los atributos son clases de .NET, podrá acortar el nombre de un atributo utilizando una instrucción Imports adecuada.

- En segundo lugar, las directrices de .NET dictan que los nombres de todas las clases de atributos deben terminar con la palabra Attribute pero la mayoría de los compiladores de NET incluyendo Visual Basic y C# le permitirán eliminar el sufijo Attribute del nombre.
- Finalmente, podrá partir las líneas de gran longitud utilizado el carácter subrayado para mejorar la legibilidad del código Después de aplicar estas tres reglas, nuestro ejemplo inicial tendrá ahora el siguiente aspecto:

```
Imports System.ComponentModel
<DescriptionAttribute("Una persona")> Class Persona1
    ' . . .
End Class
```

```
Imports System.ComponentModel
<Description("Una persona")> _
Class Persona1
    ' . . .
End Class
```

Los atributos son clases .NET bastante peculiares, permiten el empleo de propiedades y métodos, pero no podrá hacer referencia a ellos en el código tal y como lo haría con las clases normales. De hecho, la mayor parte de las veces, sólo podrá definir una o más propiedades al crear el atributo y además, el valor de estas propiedades no pueden cambiar durante el tiempo de vida de la aplicación.

La sintaxis mostrada en el código anterior es, en realidad, una llamada al método constructor del atributo Description, que acepta como argumento el valor de la propiedad Description. Una vez que se defina esta propiedad no podrá ser modificada ni consultada, al menos por la aplicación que utiliza el atributo. Sin embargo, el compilador o el propio .NET Framework podrán consultar las propiedades de los atributos utilizando reflexión. Debido a su propia naturaleza, pocas clases de atributos, si es que existen algunas, cuentan con métodos además del método constructor.

2 Atributos generales

2.1 AttributeProviderAttribute

Habilita el redireccionamiento de atributos. En el modelo de objetos de .NET Framework, hay algún caso en el que una propiedad se especifica de forma imprecisa intencionadamente. Por ejemplo, la propiedad DataGridView.DataSource se especifica como object. La razón de que esto ocurra es que esta propiedad puede aceptar varios tipos de entrada. Pero, lamentablemente, por este mismo motivo, no se proporciona un lugar común para agregar metadatos que describan las características de la propiedad. Es necesario que todas las propiedades DataSource de .NET Framework tengan metadatos idénticos para los convertidores de tipos, editores de tipos de la interfaz de usuario y otros servicios que requieren metadatos. El objeto AttributeProviderAttribute soluciona esta situación.

Cuando este atributo se coloca en una propiedad, las reglas de obtención de atributos de la colección de MemberDescriptor.Attributes del descriptor de propiedades difieren. Normalmente, el descriptor de propiedades recopila los atributos locales y, posteriormente, los combina con los atributos del tipo de propiedad. En este caso, los atributos se toman del tipo devuelto por el objeto AttributeProviderAttribute, no del tipo de propiedad real. Este atributo se utiliza en la propiedad DataGridView.DataSource para señalar el tipo específico del objeto DataGridView.DataSource para la interfaz IListSource y los metadatos correspondientes se colocan en la interfaz IListSource para habilitar el enlace de datos. De este modo, otras partes externas pueden agregar metadatos con facilidad a todos los orígenes de datos.

La prioridad de los atributos obtenidos a partir de un tipo declarado en el objeto AttributeProviderAttribute se encuentra a medio camino entre la prioridad de los atributos del tipo de propiedad y la prioridad de los atributos de la propiedad. En la siguiente lista se muestra el conjunto completo de atributos combinados disponibles, en orden de prioridad:

- Atributos de propiedad
- Atributos del proveedor de atributos
- Atributos del tipo de propiedad

En el siguiente ejemplo de código se muestra la forma de utilizar el objeto AttributeProviderAttribute para marcar una propiedad DataSource con un tipo específico de interfaz IListSource.

```
<Category("Data"), _
Description("Indicates the source of data for the control."), _
RefreshProperties(RefreshProperties.Repaint), _
AttributeProvider(GetType(IListSource))> _
Public Property DataSource() As Object
    Get
        Return Me.dataGridView1.DataSource
    End Get
    Set(ByVal value As Object)
        Me.dataGridView1.DataSource = value
    End Set
End Property
```

2.2 BindableAttribute

Marca una propiedad como apropiada para enlazarle datos

Se utiliza en tiempo de diseño para indicar si un diseñador debe ofrecer una propiedad que está disponible para el enlace de datos en un diseñador visual. Esta clase no afecta al hecho de si la propiedad puede enlazarse a datos en tiempo de ejecución.

- Bindable(BindableSupport.No) La propiedad no es enlazable en tiempo de diseño.
- Bindable(BindableSupport.Yes) La propiedad es enlazable en tiempo de diseño.
- Bindable(BindableSupport.Default) La propiedad está establecida en el valor predeterminado.

2.3 BrowsableAttribute

De forma predeterminada todas las propiedades se muestran en la ventana Propiedades. Este procedimiento no resulta muy adecuado para las propiedades de sólo lectura (que, de forma predeterminada, se mostrarán atenuadas) o para las propiedades que sólo se pueden asignar en tiempo de ejecución utilizando código. Podrá controlar la visibilidad de los elementos contenidos en la ventana Propiedades utilizando el atributo. Browsable: valor predeterminado de este atributo es True, por lo que deberá configurarlo como False para ocultar el elemento:

```
    'Esta propiedad tiene la misma semántica que el método Validate.
    '(se trata de un ejemplo que muestra cómo se
    'puede ocultar una propiedad)
    <Browsable(False)>
    ReadOnly Property IsValid() As Boolean
        Get
            Return Validate()
        End Get
    End Property
```

2.4 CategoryAttribute

Las propiedades se pueden agrupar por categorías dentro de la ventana Propiedades utilizando el atributo Category. Podrá especificar una de las categorías existentes (Diseño, Comportamiento, Apariencia) o definir una nueva. Si la 'categoría' que escribimos no existe se creara una nueva, si no utilizamos este atributo, la propiedad aparecerá dentro de la categoría Varios: ('Misc' en ingles)

```
  <Description("El control que muestra el mensaje de error"), _
   Category("Validacion")> _
   Property DisplayControl() As Control
     ' . . .
   End Property
```

2.5 ConditionalAttribute

Indica a los compiladores que se debería omitir una llamada al método o atributo a menos que se defina un símbolo de compilación condicional especificado.

Visual Basic siempre ha ofrecido una forma de incluir o de excluir trozos de código en el código compilado, para ello se utilizan las directivas #If. Todas las técnicas basadas en directivas del compilador adolecen de un punto débil que se hace evidente en el siguiente código:

```
#If LOG Then
    Sub LogMsg(ByVal MsgText As String)
       Console.WriteLine (MsgText)
    End Sub
#End If

Sub Main()
    LogMsg("Se ha iniciado el programa")
    ' . . .
    LogMsg("Se ha finalizado el programa")
End Sub
```

Como resulta evidente, podrá excluir el procedimiento LogMsg del código compilado sin más que asignar el valor cero a la constante LOG, pero obtendría numerosos errores de compilación si así lo hiciera porque todas las llamadas a dicho procedimiento permanecerían sin resolver. La única forma de enfrentarse a este problema (aparte de añadir una instrucción #If a cada una de las llamadas que se hicieran a LogMsg) es excluir el cuerpo del propio procedimiento:

```
    Sub LogMsg(ByVal MsgText As String)
#If LOG Then
    Console.WriteLine (MsgText)
#End If
    End Sub
```

Sin embargo, esta solución no es buena porque no evita la sobrecarga de todas las llamadas al procedimiento LogMsg. Visual Basic .NET (y otros lenguajes .NET, tales como C#) ofrecen una solución mucho más limpia, basada en el empleo del atributo Conditional:

```
<Conditional("LOG")> _
Sub LogMsg(ByVal MsgText As String)
    Console.WriteLine(MsgText)
End Sub

Sub ProbarAtributoCondicional()
    LogMsg("Se ha iniciado el programa")
    ' . . .
    LogMsg("Se ha finalizado el programa")
End Sub
```

El procedimiento que haya sido marcado con el atributo Conditional 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. Esta práctica produce el código más eficaz sin forzarle a agregar demasiadas directivas en su listado.

Podrá definir constantes de compilación a nivel aplicación utilizando la página Generar del cuadro de dialogo Páginas de propiedades del proyecto. Este cuadro dé diálogo le permite definir un par de constantes de compilación utilizadas con frecuencia, denominadas DEBUG y TRACE, sin más que hacer clic con el ratón

Como el compilador puede enviar todas las llamadas al método destino (LogMsg, en el ejemplo anterior) el atributo Conditional sólo funcionará con procedimientos que no devuelvan un valor y se ignorará cuando se aplique a procedimientos Function Si desea utilizar el atributo Conditional con un procedimiento que devuelva un valor al llamador, deberá utilizar un procedimiento Sub con argumentos ByRef.

Un último comentario sobre el atributo Conditional: este atributo permite definiciones múltiples, lo que significa que podrá especificar varios atributos Conditional para el mismo método. En este caso, las llamadas al método se incluirán en la aplicación compilada si cualquiera de las constantes de compilación mencionadas tiene un valor que no sea cero:

```
<Conditional("LOG"), Conditional("TRACE")> _
Sub LogMsg(ByVal MsgText As String)
    Console.Writetine(MsgText)
End Sub
```

2.6 DebuggerStepThroughAttribute

Podrá utilizar el atributo DebuggerStepThrough para marcar una funcion que el depurador de Visual Basic .NET deba ignorar porque deba ejecutarse como un todo o, simplemente, porque ya ha sido probada y depurada.

```
<DebuggerStepThroughAttribute()> _
 Private Sub InitializeComponent()
 End Sub
```

2.7 DefaultEventAttribute

Especifica el evento predeterminado de un componente.

En el ejemplo siguiente se define una clase de colección denominada MyCollection. La clase está marcada con un atributo DefaultEventAttribute que especifica CollectionChanged como evento predeterminado.

```
<DefaultEvent("CollectionChanged")> _
Public Class MyCollection
    Inherits BaseCollection
          ' . . .
          ' . . .
    Public Event CollectionChanged (, _
           ByVal sender As Object, _
           ByVal e As CollectionChangeEventArgs)
          ' . . .
          ' . . .
End Class 'MyCollection
```

2.8 DefaultPropertyAttribute

Especifica la propiedad predeterminada de un componente.

En el ejemplo siguiente se define un control denominado MyControl. La clase está marcada con un atributo DefaultPropertyAttribute que especifica MyProperty como propiedad predeterminada.

```
<DefaultProperty("MyProperty")> _
Public Class MyControl
    Inherits Control
     ' . . .
     ' . . .
    Public Property MyProperty() As Integer
      ' . . .
    End Property
      ' . . .
      ' . . .
End Class 'MyControl
```

2.9 DefaultValueAttribute

La ventana Propiedades de un control, muestra en negrita los valores que son distintos a los valores predeterminados de las propiedades, por lo que tal vez se esté preguntando en qué lugar se encuentran definidos los valores predeterminados. Como puede imaginarse, los valores predeterminados se definen utilizando otro atributo, denominado DefaultValue.

```
    Dim m_BeepOnError As Boolean = True
    <Description("Si es True, se emitirá un sonido cuando " & _
                 "falle la validación"), DefaultValue(True)> _
    Property BeepOnError() As Boolean
        Get
            Return mBeepOnError
        End Get
        Set(ByVal Value As Boolean)
            mBeepOnError = Value
        End Set
    End Property
```

Se puede crear DefaultValueAttribute con cualquier valor constante. El valor predeterminado de un miembro debe ser su valor inicial. Un diseñador visual puede utilizar el valor predeterminado para restablecer el valor del miembro. Los generadores de código pueden utilizar también los valores predeterminados para determinar si debe generarse código para el miembro.

En el siguiente ejemplo se establece el valor predeterminado de MyProperty en false.

```
Private MyVar as Boolean = False
<DefaultValue(False)> _
Public Property MyProperty() As Boolean
    Get
        Return MyVar
    End Get
    Set
        MyVar = Value
    End Set
End Property
```

Observe que el valor predeterminado en realidad, no inicializa la propiedad. Para ello, deberá utilizar un inicializador o emplear el código necesario dentro de un procedimiento Sub New.

Por desgracia, el atributo DefaultValue tiene un problema: sólo acepta valores constantes y, en determinados casos, el valor pedeterminado no es una constante. Por ejemplo, el valor SystemColors.ControlText, que es el valor inicial de la propiedad ErrorForeColor, no es una constante, por lo que no podrá pasarlo al constructor del atributo DefaultValue. Para resolver esta dificultad, el autor del control personalizado puede crear un procedimiento Resetxxxx especial, siendo xxxx el nombre de la propiedad. Veamos el código contenido en la clase TextBoxEx que llama implícitamente el Diseñador de formularios para inicializar las dos propiedades de color:

```
    Sub ResetErrorForeColor()
        ErrorForeColor = SystemColors.ControlText
    End Sub
```

Si una propiedad está asociada con un método Resetxxxx, podrá redefinir la propiedad sin más que hacer clic en la opción Restablecer que aparece en el menú contextual que se mostrará sin más que hacer clic en el nombre de la propiedad dentro de la ventana Propiedades.

El atributo DefaultValue tiene otro empleo menos evidente: si una propiedad tiene asignada su valor predeterminado (como especifica este mismo atributo) esta propiedad no se almacenará y el diseñador no generará código para ella. Este comportamiento resulta adecuado porque evita la generación de una buena cantidad de código inútil que ralentizaría la representación del formulario padre. Además, ya sabe que no puede especificar un valor no constante, un color, una fuente o cualquier otro ejemplo complejo en el atributo DefaultValue. En este caso, deberá implementar un método denominado ShouldSerializexxxx (donde xxxx es el nombre de la propiedad) que devolverá True si se debe serializar la propiedad y False si su valor actual es igual a su valor predeterminado:

```
    Function ShouldSerializeErrorForeColor)) As Boolean
        ' No podemos utilizar los operadores sobre objetos,
        ' por lo que utilizamos el método Equals.
        Return Not Me.ErrorForeColor.Equals(SystemColors.ControlText)
    End Function
```

Un ejemplo de una propiedad que cambia el color de un TextBox determinado en un formulario o un control

```
    ' ===========================================================
    Private _TextBoxBackColorValorDefecto As Color = _
                    System.Drawing.SystemColors.Window
    '-----------------------------------------
    ''' <summary>
    '''  Color de fondo del Cuadro de texto
    ''' </summary>
    <Description("Color de fondo del Cuadro de texto")> _
    Public Property ZTextBoxBackColor() As Color
        Get
            Return Me.TextBoxDatos.BackColor
        End Get
        Set(ByVal value As Color)
            If Not (Me.TextBoxDatos.BackColor = value) Then
                Me.TextBoxDatos.BackColor = value
                ' Disparar el evento
                RaiseEvent PropertyChanged( _
                   Me, _
                   New System.ComponentModel.PropertyChangedEventArgs( _
                       System.Reflection.MethodBase.GetCurrentMethod.Name))
            End If
        End Set
    End Property
    '-----------------------------------------
    Private Sub ResetZTextBoxBackColor()
        ZTextBoxBackColor = _TextBoxBackColorValorDefecto
    End Sub
    '-----------------------------------------
    Private Function ShouldSerializeZTextBoxBackColor() As Boolean
        If Me.ZTextBoxBackColor = _TextBoxBackColorValorDefecto Then
            Return False
        Else
            Return True
        End If
    End Function
```

2.10 DescriptionAttribute

Define la cadena mostrada cerca del borde inferior de la ventana Propiedades:

```
    <Description("El control que muestra el mensaje de error")> _
     Property DisplayControl() As Control
       ' . . .
     End Property
```

2.11 DesignerSerializationVisibilityAttribute

Especifica la visibilidad de una propiedad para el serializador en tiempo de diseño.

Especifica el tipo de persistencia que se va a utilizar al serializar una propiedad en un componente en tiempo de diseño.

- DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden) El generador de código no crea código para el objeto.
- DesignerSerializationVisibility(DesignerSerializationVisibility.Visible) El generador de código crea código para el objeto.
- DesignerSerializationVisibility(DesignerSerializationVisibility.Content) El generador de código crea código para el contenido del objeto en lugar del objeto en sí.

Cuando un serializador conserva el estado con persistencia de un documento en modo de diseño, suele agregar código al método de inicialización de los componentes para conservar los valores de las propiedades establecidas en tiempo de diseño. Este comportamiento es el predeterminado para la mayoría de los tipos básicos si no se establece ningún atributo que indique lo contrario.

Con el objeto DesignerSerializationVisibilityAttribute se puede indicar si el valor de una propiedad es Visible, y debe conservarse en el código de inicialización, si es Hidden, y no debe conservarse en el código de inicialización, o si está compuesto por Content, y debe disponer de código de inicialización generado para cada propiedad pública que no esté oculta del objeto asignado a la propiedad.

Los miembros que no tengan DesignerSerializationVisibilityAttribute se tratarán como si tuvieran DesignerSerializationVisibilityAttribute con el valor Visible. Si es posible, un serializador del tipo serializará los valores de una propiedad marcada como Visible. Para especificar la serialización personalizada de un determinado tipo o propiedad, utilice DesignerSerializerAttribute.

Para obtener más información, vea Información general sobre atributos y Extender metadatos mediante atributos.

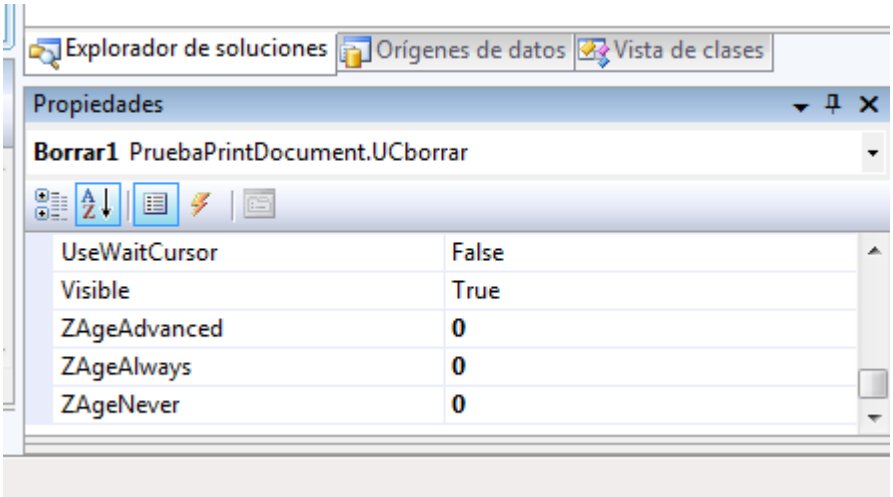
Conserva los valores de una propiedad pública de un control de usuario, que pueden configurarse en tiempo de diseño

```
<DesignerSerializationVisibility( _
            DesignerSerializationVisibility.Content)> _
Public ReadOnly Property Dimensiones() As DimensionData
    Get
        Return New DimensionData(Me)
    End Get
End Property
```

2.12 EditorBrowsableAttribute

Es una forma de mostrar u ocultar una propiedad en el editor de texto. Modifica la forma en la que el "intelisense" muestra la lista de miembros de nuestra clase.

Nota, se oculta en el editor de texto, pero en la ventana de propiedades se muestra siempre



El valor EditorBrowsableState.Always, muestra la propiedad siempre

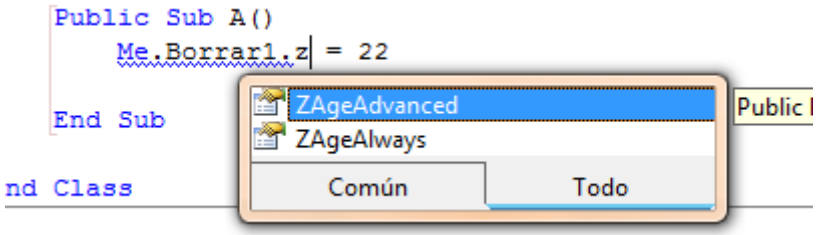
El valor EditorBrowsableState.Advanced, muestra la propiedad siempre y cuando la pestaña 'todos esta pulsada'

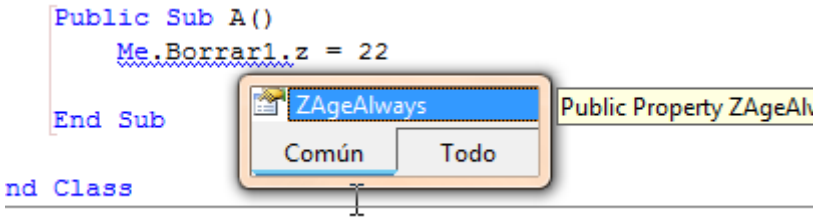
El valor EditorBrowsableState.Never, no muestra la propiedad en ningún caso, se oculta

```
<EditorBrowsable( _
       System.ComponentModel.EditorBrowsableState.Never)> _
Public Property ZAgeNever() As Integer
End Property
```

```
<EditorBrowsable( _
       System.ComponentModel.EditorBrowsableState.Always)> _
Public Property ZAgeAlways() As Integer
End Property

<EditorBrowsable( _
       System.ComponentModel.EditorBrowsableState.Advanced)> _
Public Property ZAgeAdvanced() As Integer
End Property
```





2.13 LocalizableAttribute

De forma predeterminada, las propiedades no son localizables (traducibles) y el Diseñador de formularios no almacenará sus valores en un archivo de recursos independiente cuando el usuario seleccione un lenguaje distinto al predeterminado. Podrá modificar este comportamiento predeterminado utilizando el atributo Localizable:

Cuando se genera código para un componente, los miembros que están marcados con el objeto LocalizableAttribute establecido en true tienen sus valores de propiedad guardados en archivos de recursos. Se pueden traducir estos archivos de recursos sin modificar el código.

```
<Localizable(True)> _
Property HeaderCaption() As String
  ' . . .
End Property
```

2.14 MergablePropertyAttribute

El atributo indica si una propiedad será visible en la ventana Propiedades cuando se seleccionen varios controles. El valor predeterminado de este atributo es True, por lo que deberá incluirlo de forma explícita sólo si no desea permitir que el usuario modifique esta propiedad para todos los controles seleccionados. En la práctica, utilizará este atributo cuando una propiedad no pueda tener el mismo valor para varios controles contenidos en un formulario (como sucede con las propiedades TabIndex o Name):

Las propiedades que están marcadas con el objeto MergablePropertyAttribute establecido en true se pueden combinar con propiedades que pertenecen a otros objetos en una ventana Propiedades. Las propiedades que

están marcadas con el objeto MergablePropertyAttribute establecido en false deben mostrarse por separado. El valor predeterminado es true.

```
<MergableProperty(True)> _
Public Property MyProperty() As Integer
          ' . . .
          ' . . .
End Property
```

2.15 ObsoleteAttribute

Marca los elementos del programa que ya no se utilizan.

Obsolete es aplicable a todos los elementos de programa con excepción de ensamblados, módulos, parámetros o valores devueltos. Al marcar un elemento como obsoleto, se informa a los usuarios de que el elemento se quitará en versiones futuras del producto.

Supongamos ahora que le han pasado un proyecto nada trivial y que le encargan la dura tarea de mejorar su rendimiento rescribiendo partes del mismo. Piense en el siguiente procedimiento:

```
    Sub BubbleSort(ByVal arr() As String)
    End Sub
```

BubbleSort no resulta muy eficiente por lo que la ha mejorado creando una nueva funcion de ordenación basada en un algoritmo de ordenación más eficaz (o, simplemente, decide utilizar el método Array.Sort). A continuación, comenzará a sustituir todas las llamadas que se realicen en su programa a BubbleSort. Sin embargo, no desea realizar una operación directa de búsqueda y sustitución porque desea comprobar personalmente cada una de las llamadas realizadas. Finalmente, eliminará la funcion BubbleSort pero no podrá hacerlo ahora porque ciertas partes de la aplicación no se compilarían. .NET FrameWork ofrece una solución sencilla para esta situación: el atributo Obsolete. El método constructor de este atributo puede: no aceptar argumentos, aceptar un argumento (el mensaje de advertencia) o aceptar dos argumentos (el mensaje y un valor Boolean que indique si el mensaje se va a considerar como un error de compilación):

En el siguiente ejemplo se define una clase que contiene un método marcado con ObsoleteAttribute. Cualquier código que intente llamar al método hará que el compilador emita una advertencia.

```
    'Marcar como obsoleto
    <Obsolete("Sustituir BubbleSort por ShellSort")> _
    Sub BubbleSort(ByVal arr() As String)
    End Sub
```

La variante siguiente hace que la funcion BubbleSort aparezca como un error de compilación:

```
    'Marcar BubbleSort corno obsoleta
    <Obsolete("Sustituir BubbleSort por ShellSort", True)> _
    Sub BubbleSort(ByVal arr() As String)
    End Sub
```

2.16 RefreshPropertiesAttribute

Indica que debe actualizarse la cuadrícula de propiedades cuando cambia el valor de la propiedad asociada.

Es útil cuando el hecho de asignar un nuevo valor a una determinada propiedad afecta a otras propiedades contenidas en la ventana Propiedades. El comportamiento predeterminado de la ventana Propiedades es que sólo se actualizará el valor de la propiedad que se está modificando, pero podrá especificar que se modifique el valor de todas las propiedades relacionadas utilizando este atributo:

Ejemplo, suponga que tengo una clase que transforma temperaturas de grados Celsius a grados Fahrenheit, y muestra en esas propiedades el valor para la misma temperatura, si cambia una de las propiedades, la otra tiene que cambiar también porque están mostrando la misma temperatura, aunque la escala es diferente

```
    <RefreshProperties(RefreshProperties.All)> _
    Property MaxValue() As Long
        Get
            Return m_MaxValue
        End Get
        Set(ByVal Value As Long)
            m_MaxValue = Value
            'Esta propiedad puede afectar a la propiedad Value.
            If Value > m_MaxValue Then Value = m_MaxValue
        End Set
    End Property
```

En el ejemplo de código siguiente se muestra el uso de la clase RefreshPropertiesAttribute para especificar el modo de actualización de una propiedad DataSource.

```
<Category("Data"), _
Description("Indicates the source of data for the control."), _
RefreshProperties(RefreshProperties.Repaint), _
AttributeProvider(GetType(IListSource))> _
Public Property DataSource() As Object
    Get
        Return Me.dataGridView1.DataSource
    End Get

    Set(ByVal value As Object)
        Me.dataGridView1.DataSource = value
    End Set
End Property
```

2.16.1 RefreshProperties (Enumeración)

Define los identificadores que indican el tipo de actualización de la ventana Propiedades.

- RefreshProperties(RefreshProperties.None) No es necesario actualizar.
- RefreshProperties(RefreshProperties. All) Las propiedades deben ser obligatorias y se ha de actualizar la vista.
- RefreshProperties(RefreshProperties.Repaint) Se ha de actualizar la vista.

2.17 ToolboxBitmapAttribute

Le permite especificar un icono para representar un control en un contenedor, como el Diseñador de formularios de Microsoft Visual Studio.

Se aplica a nivel clase. En su formato más sencillo, este atributo aceptará la ruta de acceso de un archivo que contenga un icono o mapa de bits de 16 por 16 píxeles:

Se puede aplicar un objeto ToolboxBitmapAttribute a un control de manera que los contenedores, como el Diseñador de formularios de Microsoft Visual Studio, puedan recuperar un icono que represente el control. El mapa de bits del icono puede encontrarse en un archivo por sí solo o incrustado en el ensamblado que contiene el control. El tamaño del mapa de bits que se incrusta en el ensamblado del control (o que se almacena en un archivo independiente) debe ser de 16 por 16. El método GetImage de un objeto ToolboxBitmapAttribute puede devolver la imagen pequeña de 16 por 16 o una imagen grande de 32 por 32 creada mediante un ajuste de escala de la imagen pequeña.

```
<ToolboxBitmap("C:\CustomControlsDemo\FileTextBox.Ico")> _
Public Class FileTextBox

End Class
```

3 Atributos de Serialización

3.1 SerializableAttribute

.NET Framework puede inspeccionar cualquier objeto en tiempo de ejecución para descubrir, leer y asignar todos los campos y propiedades del objeto. Este mecanismo es posible gracias a una parte de .NET Framework denominado reflexión y es la base para la persistencia automática de cualquier clase que escriba con un mínimo de esfuerzo de su parte.

De manera predeterminada, las clases no son serializables a menos que se encuentren marcadas con SerializableAttribute. En la práctica, lo único que tendrá que hacer para convertir una clase en serializable es marcarla con el atributo Serializable, cuyo constructor no necesita argumentos:

```
<Serializable()> Class Persona
  ' . . .
End Class
```

El atributo debe emplearse aunque la clase implemente la interfaz ISerializable para controlar el proceso de serialización.

Todos los campos públicos y privados de una clase marcados por el atributo SerializableAttribute se serializan de forma predeterminada, a menos que la clase implemente la interfaz ISerializable para reemplazar el proceso de serialización. El proceso de serialización predeterminado excluye los campos marcados con el atributo NonSerializedAttribute.

3.2 NonSerializedAttribute

Con el atributo NonSerialized, podrá marcar aquellos campos o propiedades que no desee serializar cuando serialice todo el objeto. Como regla general, no deberá serializar variables que almacenen valores que pueda calcular con facilidad a partir de otras propiedades o propiedades que no vayan a seguir siendo válidas cuando se deserialice el objeto. Tampoco deben serializarse aquellos campos que almacenen los datos confidenciales.

Durante el proceso de serialización, todos los campos públicos y privados de una clase se serializan de forma predeterminada. Los campos marcados con NonSerializedAttribute se excluyen de la serialización

¡¡¡ Importante !!! Este atributo solo funciona con las clases BinaryFormatter y/o SoapFormatter. Es decir en los procesos de serialización en formato binario y en formato Soap. Las clases XmlSerializer que manejan la serialización XML ignoran totalmente este atributo. Puede usar el atributo XmlIgnoreAttribute(clase) para obtener la misma funcionalidad

Veamos una versión de la clase Persona que vamos a utilizar como ejemplo para serializar

Observa que hay un campo privado [FechaNac] que se carga desde el constructor y un campo calculado [_Edad] que al ser calculado lo marcamos con el atributo “NonSerialized”

```
<Serializable()> _
Public Class Persona
    Public _Nombre As String
    Public _Apellidos As String
    Private _FechaNac As Date
    <NonSerialized()> Private _Edad As Integer
```

```
    Public Sub New()
        ' no hacer nada
    End Sub
    ' observe que fecha de nacimiento solo puede
    ' tomar valores utilizando el constructor
    Public Sub New(ByVal nombre As String, _
                   ByVal apellidos As String, _
                   ByVal fechaNacimiento As Date)

        Me._Nombre = nombre
        Me._Apellidos = apellidos
        Me._FechaNac = fechaNacimiento
    End Sub

    Public ReadOnly Property Edad() As Integer
        Get
            ' calcular la edad ( mas o menos)
            If _Edad = 0 Then _Edad = Date.Now.Year - _FechaNac.Year
            Return _Edad
        End Get
    End Property

End Class '/Persona
```

La presencia del atributo Serializable es todo lo que necesita .NET Framework para hacer que la clase sea persistente.

En primer lugar vamos a realizar una serialización del tipo SOAP, observa el código que empleamos para serializar en formato SOAP

```
   Public Sub PruebaSerializaSoap()
        ' secuencia de archivo para grabar el resultado
        Dim fs As IO.FileStream = New IO.FileStream( _
                  "d:\borrar\clase.Soap", IO.FileMode.Create)
        ' formateador soap
        Dim sf As New SoapFormatter( _
                   Nothing, _
                   New StreamingContext(StreamingContextStates.File))
        ' El objeto a serializar
        Dim per As New Persona ( _
                   "Joaquin", "Medina Serrano", _
                   CType("1955/9/9", Date))
        ' serializar el objeto
        sf.Serialize(fs, per)
        fs.Close()
    End Sub
```

Y a continuación puedes ver el resultado de la serialización:

```
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
                   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                   xmlns:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0"
                   SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <a1:Persona id="ref-1"
xmlns:a1="http://schemas.microsoft.com/clr/nsassem/HerramientaToHtml/HerramientaTo
Html%2C%20Version%3D2010.1.6.920%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull
">
```

```
            <_Nombre id="ref-3">Joaquin</_Nombre>
            <_Apellidos id="ref-4">Medina Serrano</_Apellidos>
            <_FechaNac>1955-09-09T00:00:00.0000000+02:00</_FechaNac>
        </a1:Persona>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

El detalle más interesante en este caso es que aunque el campo ._FechaNac es privado, el mecanismo de deserialización es capaz de recuperarlo correctamente desde la secuencia de entrada (la prueba está en que la propiedad Edad se evalúa correctamente). En otras palabras, el mecanismo de deserialización es capaz de ignorar las reglas de alcance.

Sin embargo si usamos una serialización de tipo XML el resultado varía porque este tipo de serializaciones solo guarda los campos públicos. Al igual que antes primero el código empleado para serializar el objeto y después el resultado de la serialización

```
    Public Sub PruebaSerializaXml()
        ' El objeto a serializar
        Dim per As New Persona0( _
                    "Joaquin", "Medina Serrano", _
                    CType("1955/9/9", Date))
        ' el serializador xml
        Dim ser As New XmlSerializer(GetType(Persona))
        ' secuencia de archivo para grabar el resultado
        Dim fs As IO.FileStream = New IO.FileStream( _
                   "d:\borrar\clase.xml", IO.FileMode.Create)
        ' serializar el objeto
        ser.Serialize(fs, per)
        fs.Close()
    End Sub
```

El resultado de la serialización XML, Observa que solo parecen los campos públicos, es decir, estos atributos solo funcionan con la serialización Binaria o con la serialización SOAP, la serialización XML tiene sus atributos propios

```
<?xml version="1.0"?>
<Persona xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <_Nombre>Joaquin</_Nombre>
    <_Apellidos>Medina Serrano</_Apellidos>
</Persona>
```

4 Serialización XML

Un ejemplo de serialización XML

El espacio de nombres que se va a emplear

```
Imports System.Xml.Serialization
```

La clase que vamos a serializar

```
<Serializable()> _
Public Class Cliente0
    Public ID As Integer
    Public Nombre As String
    Public Apellidos As String
    Public Ciudad As String
    Public Notas As String
    <XmlIgnoreAttribute()> Public FechaNac As Date
    Private _Edad As Integer

    Public Sub New()
        ' no hacer nada
    End Sub
    Public Sub New(ByVal nombre As String, _
                   ByVal apellidos As String, _
                   ByVal fechaNacimiento As Date)
        Me.Nombre = nombre
        Me.Apellidos = apellidos
        Me._FechaNac = fechaNacimiento
    End Sub

    Public ReadOnly Property Edad() As Integer
        Get
            ' calcular la edad ( mas o menos)
            If _Edad = 0 Then _Edad = Date.Now.Year - _FechaNac.Year
            Return _Edad
        End Get
    End Property

End Class
```

Después el código que usa la clase y la serializa

```
   Public Sub PruebaSerializaXmlCliente()
        ' El objeto a serializar
        Dim cli0 As New Cliente0( _
                   "Joaquin", "Medina Serrano", _
                   CType("1955/9/9", Date))
        cli0.Ciudad = "Zaragoza"
        cli0.Notas = "comentario amplio y sabroso sobre este cliente"
        cli0.ID = 1
        ' el serializador xml
        Dim ser As New XmlSerializer(GetType(Cliente0))
        ' secuencia de archivo para grabar el resultado
        Dim fs As IO.FileStream = New IO.FileStream( _
                  "d:\borrar\cliente0.xml", IO.FileMode.Create)
        ' serializar el objeto
        ser.Serialize(fs, cli0)
        fs.Close()
    End Sub
```

Y por último el resultado obtenido

```
<?xml version="1.0"?>
<Cliente0 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <ID>0</ID>
    <Nombre>Joaquin</Nombre>
    <Apellidos>Medina Serrano</Apellidos>
    <Ciudad>Zaragoza</Ciudad>
    <Notas>comentario amplio y sabroso sobre este cliente</Notas>
</Cliente0>
```

Observa que los campos privados no se han serializado y que el campo [FechaNac] tampoco porque esta marcado con el atributo XmlIgnoreAttribute que impide la serializacino de ese campo, es el atributo equevalente a [NonSerializedAttribute] en la serialización binaria o Soap

4.1 Atributos que controlan la serialización XML

Se pueden aplicar atributos a clases y a miembros de clase para controlar la manera en que XmlSerializer serializa o deserializa una instancia de la clase.

Tabla con los atributos para controlar la serialización XML (extraída de la documentación .NET)

http://msdn.microsoft.com/es-es/library/83y7df3e.aspx

Atributo	Se aplica a	Descripción
XmlAnyAttributeAttribute	Campos Public que devuelven una matriz de objetos XmlAttribute.	Durante la deserialización, la matriz se rellenará con objetos XmlAttribute que representan a todos los atributos XML desconocidos para el esquema.
XmlAnyElementAttribute	Campos Public que devuelven una matriz de objetos XmlElement.	Durante la deserialización, la matriz se rellenará con objetos XmlElement que representan a todos los elementos XML desconocidos para el esquema.
XmlArrayAttribute	Campos y propiedades Public que devuelven matrices de objetos complejos.	Los miembros de la matriz se generarán como miembros de una matriz XML.
XmlArrayItemAttribute	Campos y propiedades Public que devuelven matrices de objetos complejos.	Tipos derivados que se podrán insertar en una matriz.
XmlAttributeAttribute	Campos y propiedades Public.	La clase se serializará como un atributo XML.
XmlElementAttribute	Campos y propiedades Public.	El campo o propiedad se señalizará como un elemento XML.
XmlEnumAttribute	Identificadores de enumeración.	El nombre elemento de un miembro de enumeración.
XmlIgnoreAttribute	Campos y propiedades Public.	Se deberá ignorar la propiedad o el campo cuando se señalice la clase contenedora.
XmlIncludeAttribute	Declaraciones de clases derivadas Public y métodos públicos (para documentos WSDL)	Se deberá incluir la clase cuando se generen los esquemas (y, de esta forma, se reconocerán cuando se señalicen).
XmlRootAttribute	Declaraciones de clase Public.	La clase representa el elemento raíz del documento XML (utilice el atributo para especificar el espacio de nombres y el nombre del elemento).
XmlTextAttribute	Campos y propiedades Public.	La propiedad o el campo se deberá señalizar como texto XML.
XmlTypeAttribute	Declaraciones de clase Public.	La clase se deberá señalizar como un tipo XML; se ignorará si se utiliza conjuntamente con XmlRootAttribute (utilice el atributo para especificar el espacio de nombres y el nombre del elemento).

Entre estos atributos cabe destacar:

- El atributo XmlRoot le permitirá definir el nombre utilizado por el elemento raíz XML más externo, el espacio de nombres del elemento y si va a aparecer o no el atributo xsi:null cuando el objeto a serializar sea Nothing.
- El atributo XmlElement le permitirá especificar un nombre distinto para el elemento XML correspondiente a un determinado campo o propiedad. También le permitirá decidir si el elemento va a aparecer por completo aunque sea Nothing.
- El atributo XmlAttributeAttribute le permitirá marcar los campos y propiedades que se vayan a serializar como atributos en lugar de cómo elementos, decidir el nombre que se va a utilizar para el atributo y decidir si se va a omitir o no cuando el valor del campo sea Nothing (observe que podrá abreviar el nombre de este atributo en el código utilizando XmlAttribute).
- El atributo_XmlText le permitirá especificar que desea representar un campo o propiedad como texto XML sin que aparezca encerrado entre etiquetas XML.
- El atributo XmlIgnore le permitirá marcar una propiedad o un campo que no desee serializar.

Vamos a crear una nueva clase Cliente1 que utilice alguno de los atributos mencionados anteriormente:

```
<Serializable(), _
XmlRootAttribute("Cliente", _
                 Namespace:="http://www.jms32.com", _
                 IsNullable:=False)> _
Friend Class Cliente1
    <XmlAttributeAttribute("IdCliente")> Public ID As Integer
    <XmlElement("nombre")> Public Nombre As String
    <XmlIgnore()> Public Apellidos As String
    <XmlElement("ciudad", IsNullable:=False)> Public Ciudad As String
    <XmlText()> Public Notas As String
    <XmlIgnoreAttribute()> Public FechaNac As Date
    Private _Edad As Integer

    'El resto igual que la clase cliente0

End Class
```

El atributo XmlElement puede adoptar otros valores. Por ejemplo, el valor IsNullable especifica el comportamiento cuando el campo o propiedad sea Nothing: si True, el elemento XML se serializará y se agregará un atributo XML xsi:nil; si es falso o se omite, se descartará el elemento XML cuando el valor del campo sea Nothing.

Al serializar la clase Cliente1 se obtiene este texto XML .

```
<?xml version="1.0"?>
<Cliente xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         IdCliente="1"
         xmlns="http://www.jms32.com">
    <nombre>Joaquin</nombre>
    <ciudad>Zaragoza</ciudad>
    comentario amplio y sabroso sobre este cliente
</Cliente>
```

Como puede ver, el campo ID se ha representado en el atributo IdCliente, el campo Dirección no ha sido serializado (porque su valor es Nothing), el elemento raíz ha sido calificado con el espacio de nombres http://www.jms32.com y los restantes elementos XML se muestran en minúsculas. El elemento XML raíz sigue siendo Cliente, incluso aunque el nombre de la clase haya cambiado a Cliente1, gracias al atributo XmlRoot (también podríamos haber utilizado el atributo XmlType).

El constructor de los dos atributos XmlAttributeAttribute y XmlElement también puede tomar un valor Namespace, que especifica el espacio de nombres para el atributo o elemento generado. Para ver la forma en que el valor Namespace afecta al resultado, modifique la clase Cliente2 tal y como se indica a continuación:

```
<Serializable()> _
Public Class Cliente2
    <XmlAttributeAttribute("IdCliente", _
                           Namespace:="http://www.jms32.com")> _
                           Public ID As Integer
    <XmlElement("nombre", Namespace:="http://www.jms32.com")> _
                          Public Nombre As String
    <XmlIgnore()> Public Apellidos As String
    <XmlElement("ciudad", IsNullable:=False)> Public Ciudad As String
    <XmlText()> Public Notas As String
    <XmlIgnoreAttribute()> Public FechaNac As Date
    Private _Edad As Integer

    '(el resto de la clase como antes)
End Class
```

Esta nueva versión de la clase produce el siguiente texto XML

```
<?xml version="1.0"?>
<Cliente2 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          d1p1:IdCliente="1"
          xmlns:d1p1="http://www.jms32.com">
    <d1p1:nombre>Joaquin</d1p1:nombre>
    <ciudad>Zaragoza</ciudad>
    comentario amplio y sabroso sobre este cliente
</Cliente2>
```

5 Referencia Bibliográfica

Para obtener más información, vea

- Información general sobre atributos
  - [http://msdn.microsoft.com/es-es/library/xtwkdas5.aspx]
- Extender metadatos mediante atributos.
  - [http://msdn.microsoft.com/es-es/library/5x6cd29c.aspx]