Aplicando código y herencia en la configuración visual del control DataGrid

Luis Miguel Blanco Ancos

Creación por código, un modo alternativo de trabajo con el DataGrid

Todos estamos de acuerdo en que el diseñador de formularios es el medio más rápido y fácil para crear un control DataGrid en Windows; sin embargo, existen determinadas situaciones en las que la creación en tiempo de diseño de este control no es adecuada o posible, como por ejemplo, mostrar determinadas celdas del control con valores o configuraciones de color calculados dinámicamente.

Ante escenarios de este tipo, y por mucho que nos pese, debemos abandonar las plácidas aguas del diseñador visual, y adentrarnos en los turbulentos mares de la escritura de código para crear nuestro preciado grid.

En este artículo abordaremos precisamente esta técnica de creación de controles DataGrid y demás objetos relacionados con el mismo, de forma que podamos observar las diferencias existentes con respecto al medio tradicional, representado por el diseñador de formularios.

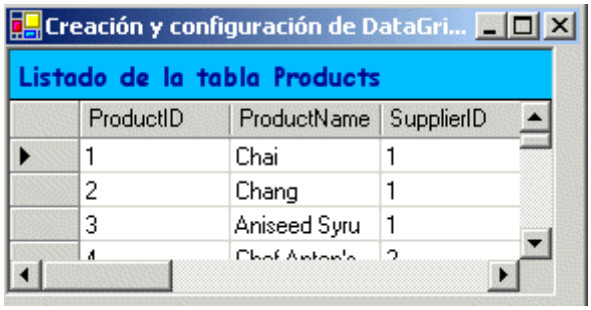
Un DataGrid con algunas sencillas mejoras de imagen

Primeramente vamos a proceder a la confección de un proyecto de tipo Windows, en el que tras crear una fuente de datos que conecte con la tabla Products, de la base de datos Northwind de SQL Server, se mostrará dicha tabla en un DataGrid sobre el que aplicaremos una ligera modificación de aspecto visual, como vemos en el siguiente fuente.

```
Imports System.Data.SqlClient

Public Class Form1
   Inherits System.Windows.Forms.Form

   Private oConnection As SqlConnection
   Private oDataAdapter As SqlDataAdapter
   Private oDataSet As DataSet
   Private oGrid As DataGrid
         '....
         '....
   Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
      ' crear conexión, adapter y dataset
```

```
        oConnection = New
SqlConnection("Server=localhost;Database=Northwind;uid=sa;pwd=")
        oDataAdapter = New SqlDataAdapter("SELECT * FROM Products", oConnection)
        oDataSet = New DataSet

        ' llenar el dataset utilizando el adaptador
        oConnection.Open()
        oDataAdapter.Fill(oDataSet, "Products")
        oConnection.Close()

        ' instanciar objeto DataGrid
        oGrid = New DataGrid

        ' añadir a la colección de controles del formulario
        Me.Controls.Add(oGrid)

        ' establecer ubicación y tamaño
        oGrid.Location = New Point(0, 0)
        oGrid.Size = New Size(Me.Width - 20, Me.Height - 30)
        oGrid.Anchor = AnchorStyles.Top Or _
           AnchorStyles.Left Or _
           AnchorStyles.Bottom Or _
           AnchorStyles.Right

        ' establecer el enlace entre el grid y la fuente de datos
        oGrid.SetDataBinding(oDataSet, "Products")

        ' modificar configuración visual base del datagrid
        oGrid.CaptionFont = New Font("Comic Sans MS", 10, FontStyle.Bold)
        oGrid.CaptionBackColor = Color.DeepSkyBlue
        oGrid.CaptionForeColor = Color.DarkBlue
        oGrid.CaptionText = "Listado de la tabla Products"
    End Sub
End Class
```

La siguiente figura muestra el aspecto del grid obtenido.



El objeto DataGrid dispone de una extensa serie de propiedades para modificar su apariencia visual, aunque por cuestiones de simplicidad en este ejemplo sólo hemos utilizado un pequeño subconjunto.

Codificación de estilos para tabla y columna

Cuando manipulamos el control DataGrid con el diseñador, podemos establecer estilos visuales para cada tabla y columna de la base de datos que necesitemos mostrar. Un estilo consiste en una configuración personalizada de colores, fuentes, formatos, etc., del elemento sobre el que se aplica, de manera que es posible disponer de combinaciones independientes para las diferentes tablas mostradas por un mismo grid, y lo que es aun más notable, distintas combinaciones de estilo para una misma tabla.

Respecto al estilo para la tabla, es necesario instanciar un objeto de la clase DataGridTableStyle, y asignar a su propiedad MappingName una cadena con el nombre de la tabla del DataSet que va a visualizar el DataGrid cuando se utilice este estilo. A continuación asignaremos valores al resto de propiedades del estilo relacionadas con el aspecto visual, tales como BackColor, AlternatingBackColor, HeaderFont, etc.

Tras el estilo de tabla es necesario también especificar las columnas que van a ser mostradas por dicha tabla. Para esta labor disponemos de la clase abstracta DataGridColumnStyle, que define el comportamiento genérico para crear estilos de columna, y sus clases derivadas DataGridTextBoxColumn y DataGridBoolColumn.

Usaremos objetos DataGridTextBoxColumn para crear estilos de columna que visualicen datos de tipo texto, fechas y numéricos.

Por otra parte, los objetos DataGridBoolColumn serán adecuados para aplicar a columnas que muestren un valor lógico, puesto que lo que veremos con este tipo de columna será una casilla de verificación marcada o vacía, según el valor de la columna sea verdadero o falso.

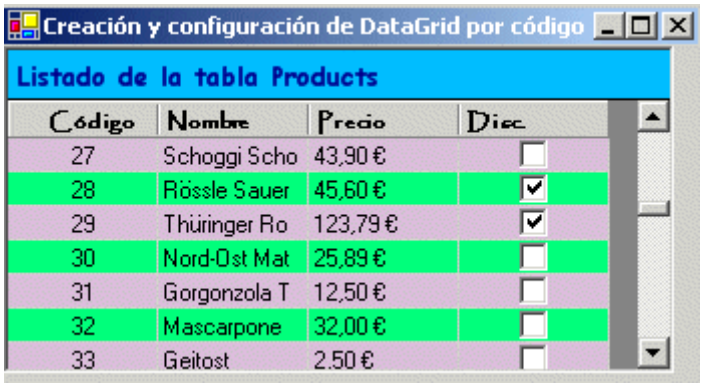
Estas clases también disponen de la propiedad MappingName, a la que asignaremos el nombre de la columna de tabla que será visualizada en el grid, siendo posible personalizar el título de columna, aplicar un formato al contenido, y otras variadas operaciones, a través del conjunto de propiedades de que disponen.

Una vez terminada la creación de los diferentes objetos de estilo de columna, los añadiremos a la colección GridColumnStyles del objeto DataGridTableStyle; e igualmente, una vez finalizada la creación del estilo de tabla, añadiremos esta a la colección TableStyles del objeto DataGrid. El fuente que vemos a continuación, partiendo del punto en el que dejamos el anterior ejemplo de código, muestra un escenario en el que intervienen las clases que acabamos de comentar.

```
' crear estilo de tabla para el datagrid
```

```
Dim oTableStyle As New DataGridTableStyle
oTableStyle.MappingName = "Products"
oTableStyle.HeaderFont = New Font("Papyrus", 8, FontStyle.Bold)
oTableStyle.BackColor = Color.Thistle
oTableStyle.AlternatingBackColor = Color.SpringGreen
oTableStyle.RowHeadersVisible = False

' crear columnas para el estilo de la tabla
Dim gcsProductID As New DataGridTextBoxColumn
gcsProductID.MappingName = "ProductID"
gcsProductID.HeaderText = "Código"
gcsProductID.Alignment = HorizontalAlignment.Center

Dim gcsProductName As New DataGridTextBoxColumn
gcsProductName.MappingName = "ProductName"
gcsProductName.HeaderText = "Nombre"

Dim gcsUnitPrice As New DataGridTextBoxColumn
gcsUnitPrice.MappingName = "UnitPrice"
gcsUnitPrice.HeaderText = "Precio"
gcsUnitPrice.Format = "C2"

Dim gcsDiscontinued As New DataGridBoolColumn
gcsDiscontinued.MappingName = "Discontinued"
gcsDiscontinued.HeaderText = "Disc."

' agregar columnas al estilo
oTableStyle.GridColumnStyles.AddRange(New DataGridColumnStyle() _
   {gcsProductID, gcsProductName, _
   gcsUnitPrice, gcsDiscontinued})

' añadir el estilo al datagrid
oGrid.TableStyles.Add(oTableStyle)
```

Tras aplicar este “lavado de cara” al DataGrid, su aspecto será muy distinto del inicial, como vemos en la siguiente figura.



Aprovechando la herencia para modificar los estilos de columna

Si bien la creación por código de columnas para un estilo de tabla nos otorga una mayor flexibilidad, realmente no es una ventaja muy superior con respecto al uso del diseñador, ya que seguimos sin poder manipular plenamente a nuestro capricho algunos aspectos de este tipo de objetos; para muestra un botón:

Supongamos que necesitamos aplicar un color de fondo a una de las columnas del estilo de tabla del DataGrid que estamos desarrollando, por ejemplo ProductName, pero desafortunadamente ninguna de las clases hijas de DataGridColumnStyle nos proporciona una propiedad para establecer tal característica.

Actualmente podemos asignar el color pero de forma global al estilo de la tabla, con lo que todas las columnas mostrarán la misma combinación de colores.

Mas no debemos desfallecer en nuestro intento, ya que sortearemos este escollo recurriendo a la herencia, y creando una clase derivada de DataGridTextBoxColumn; en ella reemplazaremos una de las sobrecargas del evento Paint, el cual se produce cada vez que el objeto necesita pintar el valor de un campo correspondiente a la columna de la base de datos con la que está conectado. Así pues, escribiremos la clase DGTxtColColor, y a continuación, modificaremos en el código del formulario la parte correspondiente a la columna ProductName, como se muestra en el siguiente código fuente

```
Public Class DGTxtColColor
   Inherits DataGridTextBoxColumn

   ' escribir miembros de clase
   ' para guardar el color personalizado
   Private mColorFondo As Color

   Public Property ColorFondo() As Color
      Get
         Return mColorFondo
      End Get
      Set(ByVal Value As Color)
         mColorFondo = Value
      End Set
   End Property

   ' cada vez que haya que pintar
   ' una celda en el grid...
   Protected Overloads Overrides Sub Paint(ByVal g As System.Drawing.Graphics, _
      ByVal bounds As System.Drawing.Rectangle, _
      ByVal source As System.Windows.Forms.CurrencyManager, _
      ByVal rowNum As Integer, _
      ByVal backBrush As System.Drawing.Brush, _
```

```
        ByVal foreBrush As System.Drawing.Brush, _
        ByVal alignToRight As Boolean)

        ' crear un nuevo objeto Brush con
        ' el color personalizado y pasarlo
        ' al método base
        Dim oBrush As New SolidBrush(mColorFondo)

        MyBase.Paint(g, bounds, source, rowNum, _
           oBrush, _
           foreBrush, _
           alignToRight)
    End Sub
End Class

'------------------------
'....
' código del formulario
Dim gcsProductName As New DGTxtColColor
gcsProductName.MappingName = "ProductName"
gcsProductName.HeaderText = "Nombre"
gcsProductName.ColorFondo = Color.Gold
```

La siguiente figura muestra el modo en cómo afecta esta clase a la columna del grid sobre la que es aplicada. Nótese que, independientemente de la configuración de colores del estilo de tabla, la columna correspondiente al objeto DGTxtColColor obliga a la presentación de su propio color.

Si quisiéramos aplicar este mismo efecto sobre una columna con valores lógicos, tendríamos que crear una clase con el código igual que la anterior, pero derivada de DataGridBoolColumn.

Sustituyendo la casilla por un literal

Llegados a este punto ya sabemos, tras los ejercicios realizados, que un campo lógico dentro de un DataGrid es mostrado como una casilla de verificación. Cuando el grid es editable este comportamiento nos favorece, ya que así podemos modificar el valor del

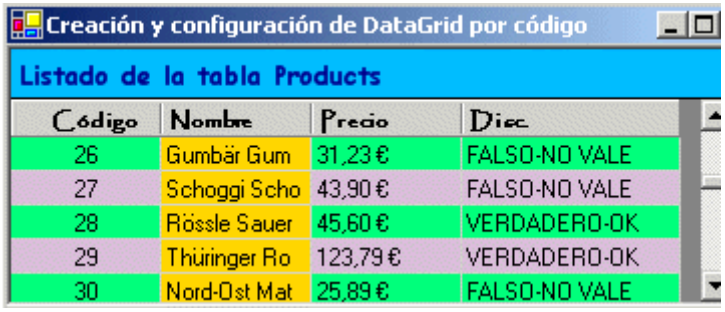
campo, pero si necesitamos presentar los datos como de sólo lectura, sería más útil mostrar un literal del tipo Verdadero-Falso, Si-No, Correcto-Incorrecto, etc.

Al igual que en el apartado anterior, la herencia será nuestra gran aliada para "desfacer este entuerto". En esta ocasión escribiremos una clase hija de DataGridTextBoxColumn, que reemplace el método GetColumnValueAtRow. Tal y como su nombre indica, este método devuelve el valor correspondiente a la celda de la fila en la que estamos posicionados. Cuando se visualiza un DataGrid, automáticamente se recorren todas las filas y columnas de la tabla de datos, por consiguiente, este método es llamado por el propio control al realizar su proceso de muestra de datos. En el siguiente código fuente vemos la implementación de esta nueva clase y su aplicación contra la columna Discontinued.

```
Public Class DGTxtColLiteral
   Inherits DataGridTextBoxColumn

   ' escribir miembros de clase para contener
   ' los valores de los literales
   Private msLitVerdadero As String
   Private msLitFalso As String

   Public Property LiteralVerdadero() As String
      Get
         Return msLitVerdadero
      End Get
      Set(ByVal Value As String)
         msLitVerdadero = Value
      End Set
   End Property

   Public Property LiteralFalso() As String
      Get
         Return msLitFalso
      End Get
      Set(ByVal Value As String)
         msLitFalso = Value
      End Set
   End Property

   Protected Overrides Function GetColumnValueAtRow(ByVal source As
System.Windows.Forms.CurrencyManager, _
      ByVal rowNum As Integer) As Object
      ' obtener el valor de la celda y
      ' sustituir por el literal
      Dim oValorCelda As Object
      oValorCelda = MyBase.GetColumnValueAtRow(source, rowNum)
```

```
        Dim bytValor As Byte
        bytValor = Convert.ToByte(oValorCelda)

        If bytValor = 0 Then
           Return msLitFalso
        Else
           Return msLitVerdadero
        End If
    End Function
End Class

'-----------------------
'....
' código del formulario
Dim gcsDiscontinued As New DGTxtColLiteral
gcsDiscontinued.MappingName = "Discontinued"
gcsDiscontinued.HeaderText = "Disc."
gcsDiscontinued.LiteralVerdadero = "VERDADERO-OK"
gcsDiscontinued.LiteralFalso = "FALSO-NO VALE"
```

En la siguiente figura observamos que ahora, la columna Discontinued ya no muestra la casilla de marcado, y en su lugar vemos el literal que hemos asignado al objeto DGTxtColLiteral.



Si quisiéramos en estos momentos dar una vuelta más de tuerca a este ejercicio, podríamos ampliar la clase que acabamos de crear y añadirle la funcionalidad de la clase DGTxtColColor, de modo que al mostrarse el literal tuviera un color de fondo para el literal verdadero, y otro para el falso.

Estilos de columna con expresiones

En el escenario anterior, el valor reflejado en las celdas de la columna se obtenía tras evaluar el contenido del campo, y en el caso de que se cumpliera una condición, se mostraba un literal predeterminado; esta sería la forma sencilla de abordar el proceso.

Pero variemos el enfoque de la situación y supongamos que el escenario a resolver es el siguiente: necesitamos mostrar una columna en la que sus celdas tomen un color de fondo en base al cumplimiento de una condición dada en otra columna de la tabla, y además, la condición a comprobar deberá pasarse dinámicamente en tiempo de

ejecución. ¿Complicado?, no se preocupe estimado lector, a continuación demostraremos que “no es tan fiera la celda como la pintan”.

Ilustremos el anterior planteamiento con un ejemplo: al visualizar la tabla Products, cuando el valor del campo UnitPrice sea menor de 25, pondremos un color en la celda correspondiente a la columna QuantityPerUnit; en el caso de que no se cumpla esta condición, utilizaremos un color distinto.

Analizando este asunto con detenimiento nos percataremos de que la pieza más complicada de construir en este mecanismo es aquella encargada de evaluar una expresión en una celda distinta de la que estamos manipulando, y si aquella condición es o no cierta, cambiar el color de fondo en la celda en la que nos encontramos actualmente.

La solución que proponemos es la siguiente: obtener la fuente de datos del control DataGrid y añadirle una columna con la expresión a evaluar. Posteriormente, en el evento de pintado de la columna, comprobar si se cumple la condición especificada por la expresión, y en función del resultado obtenido, aplicar el color de fondo pertinente a la celda.

Todas estas operaciones las codificaremos igual que en los anteriores casos, dentro de una clase hija de DataGridTextBoxColumn, cuyo código podemos ver en el siguiente fuente, al igual que su aplicación dentro del formulario.

```
Public Class DGTxtColExpresion
   Inherits DataGridTextBoxColumn

   Private mColumnaExpresion As DataColumn
   Private mColorExpVerdadera As Color
   Private mColorExpFalsa As Color

   ' propiedades para el color de fondo
   ' según se evalue el contenido de la celda
   Public Property ColorExpVerdadera() As Color
      Get
         Return mColorExpVerdadera
      End Get
      Set(ByVal Value As Color)
         mColorExpVerdadera = Value
      End Set
   End Property

   Public Property ColorExpFalsa() As Color
      Get
         Return mColorExpFalsa
      End Get
      Set(ByVal Value As Color)
```

```
      mColorExpFalsa = Value
   End Set
End Property

Public Sub AgregarColumnaExpresion(ByVal sNombreColumna As String, _
   ByVal sExpresion As String)

   Dim oDataGrid As DataGrid
   Dim oDataView As DataView

   ' obtener el DataGrid en el que esta
   ' columna está contenida
   oDataGrid = Me.DataGridTableStyle.DataGrid
   ' obtener la fuente de datos asociada al DataGrid
   oDataView = CType(oDataGrid.BindingContext(oDataGrid.DataSource, _
      oDataGrid.DataMember), CurrencyManager).List

   ' crear una columna y asignarle la expresión a evaluar
   mColumnaExpresion = New DataColumn(sNombreColumna)
   mColumnaExpresion.Expression = sExpresion
   ' añadir la columna a la fuente de datos del DataGrid
   oDataView.Table.Columns.Add(mColumnaExpresion)
End Sub

Protected Overloads Overrides Sub Paint(ByVal g As System.Drawing.Graphics, _
   ByVal bounds As System.Drawing.Rectangle, _
   ByVal source As System.Windows.Forms.CurrencyManager, _
   ByVal rowNum As Integer, _
   ByVal backBrush As System.Drawing.Brush, _
   ByVal foreBrush As System.Drawing.Brush, _
   ByVal alignToRight As Boolean)

   Dim oDataRowView As DataRowView
   Dim oDataRow As DataRow
   Dim bValorCelda As Boolean
   Dim oBrushFondo As Brush

   ' obtener la fila a pintar de la fuente de datos
   oDataRowView = source.List(rowNum)
   oDataRow = oDataRowView.Row
   ' evaluar el contenido de la expresión
   ' asociada a la columna que hemos añadido
   ' dinámicamente a este objeto
   bValorCelda = CType(oDataRow(mColumnaExpresion.ColumnName), Boolean)

   ' dependiendo del valor de la columna
   ' de expresión, seleccionar el color
```

```
        ' que debemos aplicar
        If bValorCelda Then
            oBrushFondo = New SolidBrush(mColorExpVerdadera)
        Else
            oBrushFondo = New SolidBrush(mColorExpFalsa)
        End If

        ' pintar la celda con el color de fondo apropiado
        MyBase.Paint(g, bounds, source, rowNum, _
            oBrushFondo, foreBrush, alignToRight)
    End Sub

End Class

'---------------------------
' código del formulario
'....
Dim gcsUnitPrice As New DataGridTextBoxColumn
gcsUnitPrice.MappingName = "UnitPrice"
gcsUnitPrice.HeaderText = "Precio"
gcsUnitPrice.Format = "C2"

Dim gcsQPerUnit As New DGTxtColExpresion
gcsQPerUnit.MappingName = "QuantityPerUnit"
gcsQPerUnit.HeaderText = "Cant. por unidad"
gcsQPerUnit.ColorExpVerdadera = Color.LightSkyBlue
gcsQPerUnit.ColorExpFalsa = Color.Violet

' agregar columnas al estilo
oTableStyle.GridColumnStyles.AddRange(New DataGridColumnStyle() _
    {gcsProductID, gcsProductName, gcsDiscontinued, _
gcsUnitPrice, gcsQPerUnit})

' añadir el estilo al datagrid
oGrid.TableStyles.Add(oTableStyle)

' añadir la expresión de columna a evaluar
gcsQPerUnit.AgregarColumnaExpresion("ChkPrecio", "UnitPrice < 25")
```

La siguiente figura muestra el DataGrid con las dos columnas involucradas en este proceso.

Celdas de cierre

Y tras este ejemplo damos por concluido este artículo, a lo largo del cual nos hemos dedicado a programar el control DataGrid desde una perspectiva orientada exclusivamente al código, prescindiendo de los diseñadores, y aplicando la herencia para conseguir una mayor personalización en la presentación de los datos. Esperamos que todo lo aquí comentado sea de utilidad para nuestros lectores.