Sustituyendo el editor de celdas del control DataGridView

Luis Miguel Blanco Ancos

El uso de controles alternativos para editar celdas

Si el valor del campo que necesitamos modificar está basado en texto simple, la funcionalidad estándar de este control es suficiente, pero ¿qué ocurre si nos enfrentamos a números o fechas?. Para estos casos, otros controles tales como NumericUpDown, MonthCalendar, etc., pueden ajustarse más adecuadamente a estas labores de edición si los incluimos en las celdas del DataGridView.

Para conseguir esta funcionalidad deberemos introducirnos en la maquinaria interna del control y realizar unos "pequeños ajustes", con los cuales lograremos editar las celdas utilizando el control de nuestra elección.

El ejemplo sobre el que centraremos nuestra atención consistirá en la creación de una columna cuyas celdas albergarán valores numéricos, los cuales editaremos a través de un control NumericUpDown ubicado en cada celda que editemos.

Punto de partida. Establecer la estructura de clases

En primer lugar crearemos un proyecto Windows al que agregaremos un archivo de clase, en el cual vamos a escribir las declaraciones de las clases que van a constituir el esqueleto básico de esta arquitectura. El lector puede encontrar el ejemplo completo desarrollado para este artículo en la dirección http://www.dotnetmania.com.

Dado que el objeto más inmediato que manipularemos será la columna del control de cuadrícula, crearemos una clase derivada de DataGridViewColumn.

A continuación procederemos igualmente con la celda, creando una clase que herede de DataGridViewTextBoxCell.

Finalmente crearemos una tercera clase, que constituirá el núcleo del proceso de edición, para lo cual deberá heredar del control que vayamos a utilizar para editar la celda –NumericUpDown-, e implementar la interfaz IDataGridViewEditingControl, cuyos miembros proporcionarán la funcionalidad necesaria que actuará de puente entre el control de edición y la celda que lo contiene. En el fuente 1 se encuentran estas declaraciones.

```
using System.Windows.Forms;
using System.Drawing;
//....
```

```
class NumArribaAbajoColumn : DataGridViewColumn
{
   //....
}

class NumArribaAbajoCell : DataGridViewTextBoxCell
{
   //....
}

class NumArribaAbajoEditingControl : NumericUpDown, IDataGridViewEditingControl
{
   //....
}
```

Fuente 1.

La clase NumArribaAbajoColumn

Esta clase, derivada de DataGridViewColumn, será la encargada de recabar toda la información necesaria para configurar el control que alojaremos en la celda.

Puesto que la columna es el medio de que dispone el programador para configurar el control contenido en la celda, crearemos en esta clase un conjunto de propiedades que se correspondan con las propiedades del control de edición. Dada la extensión del código de las clases utilizadas, en las páginas del presente artículo se ofrecen aquellas partes más importantes de las mencionadas clases, encontrándose disponibles en su totalidad en el enlace correspondiente al material de apoyo del artículo. En el fuente 2, el lector puede ver una parte del código correspondiente a la columna.

```
class NumArribaAbajoColumn : DataGridViewColumn
{
   // campos
   private decimal nMaximo;
   private decimal nIncremento;
   private bool bSeparadorMiles;
   private int nPosicionesDecimales;
   //....
   // en el constructor establecemos el tipo de celda
   // que utilizará esta columna
   public NumArribaAbajoColumn()
      : base(new NumArribaAbajoCell())
   {
      // modo de ordenación de los datos
      this.SortMode = DataGridViewColumnSortMode.Automatic;
   }
   //....
   // propiedades para configurar la celda cuando
   // editemos su valor mediante el control NumericUpDown
   public decimal Maximo
   {
      get { return nMaximo; }
      set { nMaximo = value; }
   }
```

```
  public decimal Incremento
  {
    get { return nIncremento; }
    set { nIncremento = value; }
  }

  public bool SeparadorMiles
  {
    get { return bSeparadorMiles; }
    set { bSeparadorMiles = value; }
  }

  public int PosicionesDecimales
  {
    get { return nPosicionesDecimales; }
    set { nPosicionesDecimales = value; }
  }
  //....
}
```

Fuente 2.

Fijémonos en el constructor de esta clase. Una columna necesita conocer el tipo que tendrá que usar a la hora de construir sus celdas, por lo tanto invocamos aquí al constructor base, que recibe como parámetro una instancia de DataGridViewCell –más concretamente de nuestra clase NumArribaAbajoCell, derivada de DataGridViewTextBoxCell-, que utilizará como plantilla en la creación del celdas.

Por otra parte, para permitir la ordenación de la columna, asignamos a su propiedad SortMode el valor Automatic de la enumeración DataGridViewColumnSortMode, mediante el que dicho orden se realizará automáticamente al pulsar sobre su cabecera.

La clase NumArribaAbajoCell

Tras la columna, el paso lógico será la creación de la clase que represente a las celdas, heredando de DataGridViewTextBoxCell.

El funcionamiento a desempeñar por esta clase va a consistir en devolver información acerca de los tipos utilizados para editar la celda y su valor, para lo cual, reemplazaremos respectivamente las propiedades EditType y ValueType de la clase base.

La propiedad DefaultNewRowValue la usaremos para devolver el valor de la celda cuando sea creada una nueva fila en la cuadrícula.

Finalmente escribiremos el método InitializeEditingControl, que se ocupará, como su nombre indica, de obtener una instancia del control a usar para editar la celda, así como de inicializar sus propiedades a partir de los valores que hemos pasado al objeto columna, al que accedemos mediante la propiedad OwningColumn.

También controlaremos posibles problemas tales como la inexistencia de valores para la celda, o que se halle en el intervalo numérico permitido. Si no se cumple alguna de estas condiciones, generaremos una excepción, asignando explícitamente un valor para la celda. Ver el fuente 3.

```
class NumArribaAbajoCell : DataGridViewTextBoxCell
{
   // obtener el tipo de control utilizado para editar la celda
   public override Type EditType
   {
      get { return typeof(NumArribaAbajoEditingControl); }
   }

   // obtener el tipo de dato que es editado en la celda
   public override Type ValueType
   {
      get { return typeof(Decimal); }
   }

   // establecer el valor por defecto para una nueva celda
   public override object DefaultNewRowValue
   {
      get { return 0; }
   }

   // crear y alojar en la celda una instancia del control
   // utilizado para editar el valor de la celda
   public override void InitializeEditingControl(int rowIndex, object initialFormattedValue,
DataGridViewCellStyle dataGridViewCellStyle)
   {
      base.InitializeEditingControl(rowIndex, initialFormattedValue, dataGridViewCellStyle);

      // obtener el control que usaremos para editar el valor de la celda
      NumArribaAbajoEditingControl ctlEditorCelda =
(NumArribaAbajoEditingControl)DataGridView.EditingControl;

      // obtener el objeto columna propietario de la celda que vamos a editar
      NumArribaAbajoColumn colNumArribaAbajo = (NumArribaAbajoColumn)this.OwningColumn;

      // asignar valor a las propiedades del control editor
      ctlEditorCelda.Minimum = colNumArribaAbajo.Minimo;
      ctlEditorCelda.Maximum = colNumArribaAbajo.Maximo == 0 ? 100 : colNumArribaAbajo.Maximo;
      ctlEditorCelda.Increment = colNumArribaAbajo.Incremento == 0 ? 1 :
colNumArribaAbajo.Incremento;
      ctlEditorCelda.ThousandsSeparator = colNumArribaAbajo.SeparadorMiles;
      ctlEditorCelda.DecimalPlaces = colNumArribaAbajo.PosicionesDecimales;
      ctlEditorCelda.UpDownAlign = colNumArribaAbajo.AlineacionFlechas;
      ctlEditorCelda.TextAlign = colNumArribaAbajo.AlineacionTexto;
      ctlEditorCelda.BackColor = colNumArribaAbajo.ColorFondo;

      // comprobar si el valor de la celda que debemos editar
      // está en el rango admisible por el control NumericUpDown
      if (this.Value == DBNull.Value)
      {
         ctlEditorCelda.Value = 0;
```

```
      }
      else
      {
        if (((decimal)this.Value >= ctlEditorCelda.Minimum) &&
          ((decimal)this.Value <= ctlEditorCelda.Maximum))
        {
          ctlEditorCelda.Value = (decimal)this.Value;
        }
        else
        {
          this.RaiseDataError(new DataGridViewDataErrorEventArgs(
            new Exception(string.Format("Valor {0} incorrecto. Rango admisible de valores de {1} a {2}",
              this.Value, ctlEditorCelda.Minimum, ctlEditorCelda.Maximum)),
            this.ColumnIndex, this.RowIndex, DataGridViewDataErrorContexts.Display));
          this.Value = ctlEditorCelda.Minimum;
        }
      }
    }
}
```

Fuente 3.

La clase NumArribaAbajoEditingControl

Y llegamos por fin a esta clase, que como hemos mencionado anteriormente, al heredar del control que especifiquemos, conseguiremos que una instancia del mismo sea alojada en la celda para editar su valor.

Adicionalmente, mediante la implementación de los miembros pertenecientes a la interfaz IDataGridViewEditingControl, podremos desarrollar varias acciones, tales como el acceso al control DataGridView que actúa como contenedor de la celda; la fila sobre la cual se va a realizar la edición; las operaciones de teclado que deberán ser manejadas por el propio control de edición en lugar de la celda contenedora, y un largo etcétera. Veamos una muestra en el fuente 4.

```
class NumArribaAbajoEditingControl : NumericUpDown, IDataGridViewEditingControl
{
  // campos
  private DataGridView oDGV;
  private int nIndiceFila;
  private bool bValorCambiado = false;

  public DataGridView EditingControlDataGridView
  {
    get { return oDGV; }
    set { oDGV = value; }
  }

  public int EditingControlRowIndex
  {
    get { return nIndiceFila; }
    set { nIndiceFila = value; }
```

```
  }

  public bool EditingControlWantsInputKey(Keys keyData, bool dataGridViewWantsInputKey)
  {
    // establecer las pulsaciones de teclado
    // que el control alojado en la celda podrá utilizar
    switch (keyData & Keys.KeyCode)
    {
      case Keys.Up:
      case Keys.Down:
      case Keys.Left:
      case Keys.Right:
      case Keys.Home:
      case Keys.End:
        return true;

      default:
        return false;
    }
  }

  protected override void OnValueChanged(EventArgs e)
  {
    bValorCambiado = true;
    this.EditingControlDataGridView.NotifyCurrentCellDirty(true);
    base.OnValueChanged(e);
  }
  //....
}
```

Fuente 4.

Cabe destacar, en el apartado visual de este control, que si asignamos un estilo a la columna que lo contiene, las propiedades de dicho estilo no se aplicarán directamente al control ubicado en la celda.

La solución que a priori podríamos pensar en desarrollar, consistiría en la creación, dentro de la clase columna, de las propiedades necesarias que posteriormente pasaríamos a la clase de la celda; esta última se encargaría de asignar las mencionadas propiedades al control en su método InitializeEditingControl. El fuente 5 muestra esta situación aplicada a la propiedad BackColor.

```
class NumArribaAbajoColumn : DataGridViewColumn
{
  //....
  private Color clrColorFondo;

  public Color ColorFondo
  {
    get { return clrColorFondo; }
    set { clrColorFondo = value; }
  }
  //....
}
```

```
class NumArribaAbajoCell : DataGridViewTextBoxCell
{
   //....
   public override void InitializeEditingControl(int rowIndex, object initialFormattedValue,
DataGridViewCellStyle dataGridViewCellStyle)
   {
      //....
      ctlEditorCelda.BackColor = colNumArribaAbajo.ColorFondo;
   }
   //....
}
```

Fuente 5.

Sin embargo, existe un medio mucho más adecuado, creado ex profeso para que el control editor de la celda pueda obtener las propiedades que necesita del estilo asignado a la columna. Consiste en implementar el método ApplyCellStyleToEditingControl de la interfaz, que recibe un parámetro conteniendo el estilo perteneciente a la columna, del cual tomaremos aquellos valores que necesitemos, para asignarlos a las propiedades en este caso del NumericUpDown alojado en la celda. Ver el fuente 6.

```
class NumArribaAbajoEditingControl : NumericUpDown, IDataGridViewEditingControl
{
   //....
   public void ApplyCellStyleToEditingControl(DataGridViewCellStyle dataGridViewCellStyle)
   {
      // aplicar al control de edición de la celda
      // las propiedades del estilo de la celda
      this.Font = dataGridViewCellStyle.Font;
      this.BackColor = dataGridViewCellStyle.ForeColor;
      this.ForeColor = dataGridViewCellStyle.BackColor;
   }

   //....
}
```

Fuente 6.

Empleando la nueva columna en el DataGridView

Es momento de hacer uso práctico a todo el código que hemos desarrollado. Para ello, dentro del formulario del proyecto sobre el que estamos trabajando, añadiremos un control DataGridView, que rellenaremos con algunos campos de la tabla DimProduct, perteneciente a la base de datos AdventureWorksDW. En concreto será el campo ListPrice el que editaremos mediante nuestra columna personalizada.

Debido a que la consulta SQL que pasamos al DataGridView ya contiene el campo ListPrice, en primer lugar eliminaremos de la cuadrícula la columna que por defecto crea el control para este campo. A partir de dicho punto instanciaremos un objeto de nuestra jerarquía de clases, configurando las propiedades intrínsecas de la columna,

pero también las correspondientes al control NumericUpDown de edición. Por otra parte crearemos un estilo para la columna, cuyos valores veremos igualmente reflejados al editarla.

Como paso final, asignaremos a una de las celdas un valor superior al rango permitido por el control de edición, lo que nos permitirá que nuestra clase desencadene el evento DataError de la cuadrícula de datos, en el caso de que editemos dicha celda. Todo ello lo vemos en el fuente 7.

```
private void Form1_Load(object sender, EventArgs e)
{
   //....
   // eliminar columna original
   this.dataGridView1.Columns.Remove("ListPrice");

   // crear columna para edición personalizada
   NumArribaAbajoColumn colListPrice = new NumArribaAbajoColumn();
   colListPrice.Name = "colListPrice";
   colListPrice.DataPropertyName = "ListPrice";
   colListPrice.HeaderText = "Precio producto";
   colListPrice.Maximo = 3000;
   colListPrice.Incremento = 2;
   colListPrice.SeparadorMiles = true;
   colListPrice.PosicionesDecimales = 3;
   colListPrice.AlineacionTexto = HorizontalAlignment.Right;

   // crear estilo para la nueva columna
   DataGridViewCellStyle styEstilo = new DataGridViewCellStyle();
   styEstilo.Alignment = DataGridViewContentAlignment.MiddleCenter;
   styEstilo.BackColor = Color.LightBlue;
   styEstilo.SelectionBackColor = Color.GreenYellow;
   styEstilo.SelectionForeColor = Color.Firebrick;
   styEstilo.Font = new Font("Tahoma", 10, FontStyle.Bold);

   // aplicar estilo a columna
   colListPrice.DefaultCellStyle = styEstilo;

   // agregar nueva columna al grid
   this.dataGridView1.Columns.Add(colListPrice);

   // modificar una celda, introducir un valor
   // mayor del rango soportado por el control de edición
   this.dataGridView1.Rows[2].Cells["colListPrice"].Value = 3500;
}

private void dataGridView1_DataError(object sender, DataGridViewDataErrorEventArgs e)
{
   MessageBox.Show("Columna: " + this.dataGridView1.Columns[e.ColumnIndex].HeaderText + "\n" +
      "Fila: " + (e.RowIndex + 1) + "\n" +
      "Mensaje de error: " + e.Exception.Message,
      "Error de edición de celda");
}
```

Fuente 7.

La figura 1 muestra el control en pleno proceso de edición.

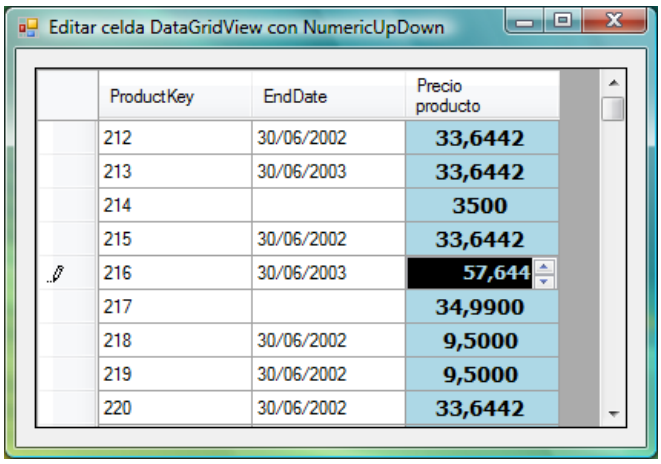

Figura 1. Celda del DataGridView editada mediante un NumericUpDown.

Alojando controles no directamente editables

En la columna que acabamos de crear, el usuario puede introducir los valores en la celda utilizando las flechas del NumericUpDown, o bien tecleándolos directamente.

Habitualmente, el comportamiento del control ubicado en la celda permite que su contenido sea editado, pero también es posible utilizar controles basados la selección de un valor, sin posibilidad de editarlo. Como ejemplo tenemos la clase DataGridViewComboBoxColumn, perteneciente a la propia plataforma .NET Framework.

Pero el objetivo del presente artículo es la personalización de este aspecto del control DataGridView, por lo que podemos recurrir a desarrollar una columna cuyas celdas contengan un control con estas características, como puede ser MonthCalendar, el cual ofrece un calendario donde el usuario sólo puede seleccionar fechas, no existiendo la posibilidad de escribir su valor.

Por cuestiones de espacio no es posible ofrecer el código al completo para esta nueva columna personalizada, aunque como ya hemos mencionado, todo el código tratado en este artículo se encuentra entre los ejemplos descargables del sitio web de la revista.

Dado que MonthCalendar es un control que al ser alojado en las celdas de una columna ocupará un mayor espacio cuando sea editado, tendremos que realizar los cálculos oportunos que nos permitan aumentar las dimensiones de la celda sobre la que se produzca la edición del valor, restaurando la celda al tamaño inicial una vez concluida la operación, como vemos en el fuente 8.

```
public class CalendarioMensualColumn : DataGridViewColumn
{
```

```
  //....
}

public class CalendarioMensualCell : DataGridViewTextBoxCell
{
  //....
  public override void InitializeEditingControl(int rowIndex, object initialFormattedValue,
System.Windows.Forms.DataGridViewCellStyle dataGridViewCellStyle)
  {
    base.InitializeEditingControl(rowIndex, initialFormattedValue, dataGridViewCellStyle);

    CalendarioMensualEditingControl ctlEditorCelda =
(CalendarioMensualEditingControl)DataGridView.EditingControl;
    CalendarioMensualColumn colCalendarioMensual =
(CalendarioMensualColumn)this.OwningColumn;

    // guardar las coordenadas de la celda (fila y columna)
    ctlEditorCelda.EditingControlRowIndex = rowIndex;
    ctlEditorCelda.IndiceColumna = this.OwningColumn.Index;

    // guardar las dimensiones originales de la celda
    ctlEditorCelda.AlturaCelda = this.DataGridView.Rows[rowIndex].Height;
    ctlEditorCelda.AnchuraCelda = this.DataGridView.Columns[this.OwningColumn.Index].Width;

    // establecer nueva dimensión para la celda,
    // de forma que visualicemos el control editor al completo
    this.DataGridView.Rows[rowIndex].Height = ctlEditorCelda.Size.Height;
    this.DataGridView.Columns[this.OwningColumn.Index].Width = ctlEditorCelda.Size.Width;

    // comprobar la fecha a pasar al control de edición
    if (this.Value == DBNull.Value)
    {
      ctlEditorCelda.SelectionEnd = DateTime.Today;
    }
    else
    {
      ctlEditorCelda.SelectionEnd = (DateTime)this.Value;
    }
  }
}

public class CalendarioMensualEditingControl : MonthCalendar, IDataGridViewEditingControl
{
  // campos
  //....
  private int nIndiceFila;
  private int nIndiceColumna;
  private int nAlturaCelda;
  private int nAnchuraCelda;
  //....
  // propiedades particulares
  public int IndiceColumna
  {
    get { return nIndiceColumna; }
    set { nIndiceColumna = value; }
  }
```

```
  public int AlturaCelda
  {
    get { return nAlturaCelda; }
    set { nAlturaCelda = value; }
  }

  public int AnchuraCelda
  {
    get { return nAnchuraCelda; }
    set { nAnchuraCelda = value; }
  }

  // métodos reemplazados
  //....
  protected override void OnLostFocus(System.EventArgs e)
  {
    base.OnLostFocus(e);

    // restaurar celda a su tamaño original
    oDGV.Rows[nIndiceFila].Height = nAlturaCelda;
    oDGV.Columns[nIndiceColumna].Width = nAnchuraCelda;
  }
  //....
}
```

Fuente 8.

En la clase CalendarioMensualEditingControl declaramos un conjunto de propiedades para almacenar la posición y tamaño de la celda. Estas propiedades las emplearemos en el método CalendarioMensualCell.InitializeEditingControl, cuando el control de edición sea creado. Para detectar el momento en el que el usuario termina de editar el control, reemplazaremos el método OnLostFocus en la clase derivada de MonthCalendar, dentro del cual devolveremos la celda a su tamaño original.

El fuente 9 muestra la creación en el formulario de un objeto correspondiente a esta nueva columna personalizada.

```
// eliminar la columna previa
this.dataGridView1.Columns.Remove("EndDate");

// crear columna personalizada para el campo EndDate
CalendarioMensualColumn colEndDate = new CalendarioMensualColumn();
colEndDate.Name = "colEndDate";
colEndDate.DataPropertyName = "EndDate";
colEndDate.HeaderText = "Fecha final";
colEndDate.MostrarFechaActual = false;
colEndDate.MostrarNumerosSemana = true;
this.dataGridView1.Columns.Add(colEndDate);
```

Fuente 9.

La figura 2 muestra el control DataGridView editando este nuevo tipo de celda que acabamos de crear.

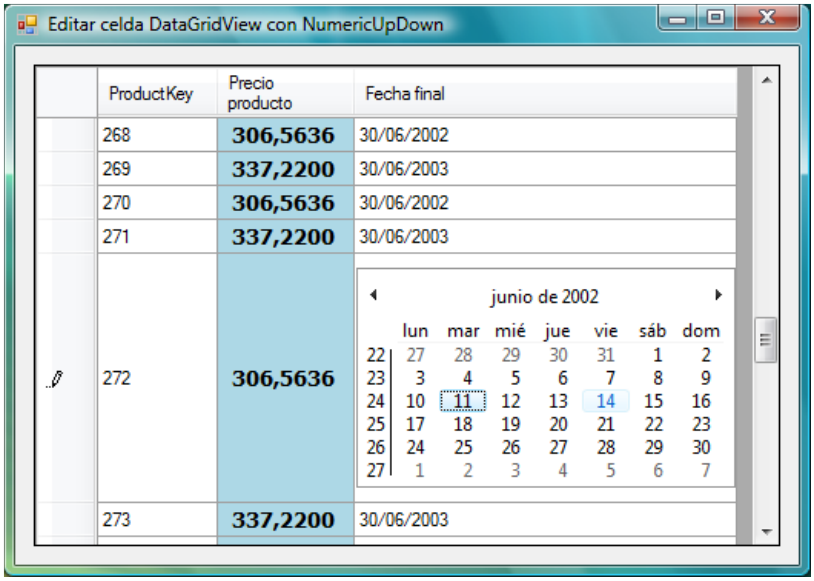

Figura 2. Columna para editar fechas mediante un control MonthCalendar.

Vamos concluyendo

Esperamos que las capacidades de extensión del control DataGridView, en los aspectos de creación de columnas personalizadas expuestos en este artículo, hayan despertado el interés del lector, animándole a desarrollar sus propias columnas para resolver aquellos aspectos concretos no disponibles “de serie” en este control. Un saludo a todos.