Aplicación de formato y selección manual de columnas sobre el control DataGridView

Luis Miguel Blanco Ancos

Preparando el escenario de trabajo

Supongamos que nos encontramos desarrollando un formulario en el que necesitamos visualizar dentro de un DataGridView la consulta SQL del fuente 1.

```
SELECT ProductKey, SpanishProductName, ListPrice, EndDate, Color
FROM DimProduct
WHERE ListPrice IS NOT NULL
```

Fuente 1.

Pero como siempre ocurre en estos casos, los requerimientos de la aplicación dictan que ciertos campos de la tabla deberán mostrarse y formatearse de un modo distinto al estándar, con el agravante de que estas operaciones no podrán ser realizadas para todos los casos, sino en función de ciertas condiciones, determinadas por los valores de algunos campos de la tabla.

Es por ello que no podremos recurrir directamente a la solución que a priori sería la más rápida: la creación de estilos para las columnas necesarias, configurando en dichos estilos las propiedades oportunas para alterar el aspecto que los datos tienen por defecto.

Sí que emplearemos estilos para conseguir nuestros propósitos, pero será dando un rodeo que nos llevará por CellFormatting, un utilísimo y versátil evento perteneciente a DataGridView.

El lector puede encontrar todos los ejemplos desarrollados en este artículo en la dirección http://www.dotnetmania.com.

CellFormatting. Estilo y formato condicionales

Se trata de un evento que se produce para cada celda del control DataGridView que precise ser dibujada, por lo que en el método manipulador de evento que escribamos, añadiremos el código encargado de realizar las comprobaciones oportunas, para de esta manera, aplicar cuando sea necesario los cambios de estilo sobre aquellas columnas que lo requieran.

La información complementaria que necesitemos manejar, como puede ser el índice de columna o fila sobre la que se está produciendo el evento, la recibiremos en un parámetro de tipo DataGridViewCellFormattingEventArgs.

Una vez descrito brevemente este evento, pasemos a continuación a exponer las diferentes técnicas de formato que utilizaremos, en función del valor con el que debamos trabajar en cada ocasión.

Aplicar formato en función del valor de la celda actual

Esta técnica consiste en comprobar la columna a la cual pertenece la celda que se va a dibujar, así como su valor, que nos facilita la propiedad Value del objeto DataGridViewCellFormattingEventArgs. En el caso de que se cumpla una determinada condición sobre el valor, utilizaremos la propiedad CellStyle del mencionado objeto, para modificar el aspecto habitual de la celda, resaltando ciertas características de su estilo.

Este caso queda ilustrado en el fuente 2, donde aplicamos esta operación sobre los campos ListPrice y EndDate del origen de datos.

```
private void dgvGrid_CellFormatting(object sender,
DataGridViewCellFormattingEventArgs e)
{
   // si la celda a dibujar es ListPrice...
   if (this.dgvGrid.Columns[e.ColumnIndex].Name == "ListPrice")
   {
      // ...y se cumple esta condición
      if ((decimal)e.Value > 1000)
      {
         // modificar propiedades del estilo de la celda
         e.CellStyle.Alignment = DataGridViewContentAlignment.MiddleCenter;
         e.CellStyle.BackColor = Color.DarkTurquoise;
         e.CellStyle.ForeColor = Color.WhiteSmoke;
         e.CellStyle.Font = new Font("Comic Sans MS", 12);
         e.CellStyle.Format = "#,#.#0";
      }
   }

   // si la celda a dibujar es EndDate...
   if (this.dgvGrid.Columns[e.ColumnIndex].Name == "EndDate")
   {
      // ...y se cumple esta condición
      if (e.Value != System.DBNull.Value && ((DateTime)e.Value).Year == 2002)
      {
         e.CellStyle.Font = new Font("Comic Sans MS", 8 , FontStyle.Italic |
FontStyle.Bold););
         e.CellStyle.Format = "dddd, dd-MMMM-yyyy";
```

```
    }
  }
  //....
}
```

Fuente 2.

La figura 1 muestra el control resultado de esta operación.

Figura 1. Formato aplicado a números y fechas.

Formato indirecto

La siguiente técnica que mostraremos se basa en aplicar un formato de forma indirecta, o lo que es lo mismo, cuando se vaya a dibujar la celda de la columna ProductKey, comprobamos el valor de otra celda –la que se halla en la columna EndDate-, y según el año que tenga esa fecha, cambiamos o no el estilo de la celda de ProductKey. En este caso, además, no accedemos directamente a los miembros de CellStyle, sino que creamos previamente un objeto de estilo que finalmente asignamos a esta propiedad, como vemos en el fuente 3.

```
private void dgvGrid_CellFormatting(object sender,
DataGridViewCellFormattingEventArgs e)
{
  //....
  // si la celda a dibujar es ProductKey...
  if (this.dgvGrid.Columns[e.ColumnIndex].Name == "ProductKey")
  {
    DateTime dtEndDate;

    // comprobar el valor del campo EndDate,
    if
(DateTime.TryParse(this.dgvGrid.Rows[e.RowIndex].Cells["EndDate"].Value.ToString(),
      out dtEndDate))
    {
```

```
            // si se cumple esta condición
            if (dtEndDate.Year == 2003)
            {
               // crear un estilo...
               DataGridViewCellStyle styCelda = new DataGridViewCellStyle();
               styCelda.BackColor = Color.PaleVioletRed;
               styCelda.ForeColor = Color.LightYellow;
               styCelda.Font = new Font("Castellar", 12, FontStyle.Bold);
               // ...y asignarlo a la celda
               e.CellStyle = styCelda;
            }
         }
      }
      //....
}
```

Fuente 3.

Podemos comprobar el efecto en la figura 2.

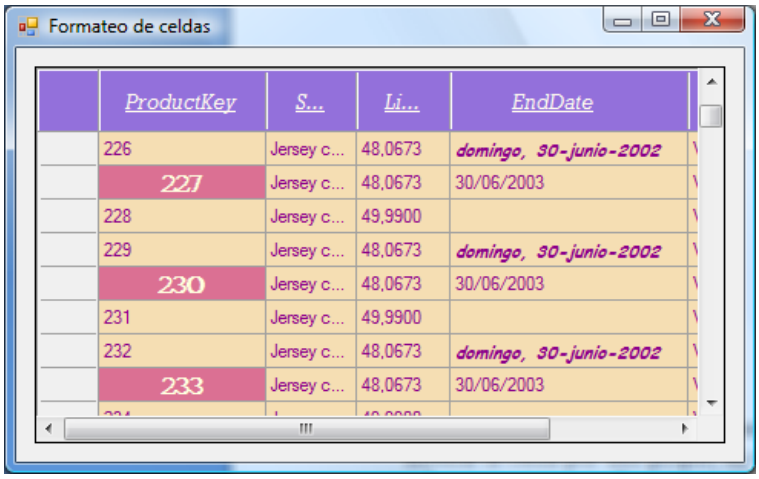

Figura 2. Modificar el aspecto de una celda en base al valor de otra.

Cambiando el valor original de la celda

Seguidamente ilustramos la capacidad de cambiar el valor que originalmente debería mostrar la celda por uno propio; en este caso sin alterar para nada los estilos de la columna –aunque perfectamente podríamos haberlo hecho-. La columna a manipular corresponderá al campo Color de la fuente de datos. Dado que la propiedad DataGridViewCellFormattingEventArgs.Value es de lectura y escritura, vamos a tomar su valor original, volviéndolo a asignar ya traducido, como vemos en el fuente 4.

```
private void dgvGrid_CellFormatting(object sender,
DataGridViewCellFormattingEventArgs e)
{
```

```
    //....
    // si la celda a dibujar es Color...
    if (this.dgvGrid.Columns[e.ColumnIndex].Name == "Color")
    {
      // cambiar el texto de la celda
      switch (e.Value.ToString())
      {
        case "Black":
          e.Value = "Negro";
          break;
        case "Blue":
          e.Value = "Azul";
          break;
        case "Grey":
          e.Value = "Gris";
          break;
        //....
      }
    }
    //....
}
```

Fuente 4.

La figura 3 muestra el resultado de esta traducción.

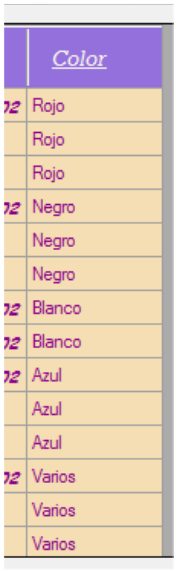

Figura 3. Cambiando (traduciendo) el valor de las celdas.

Observando el resultado al ejecutar el formulario, nos percataremos de que las cabeceras muestran el texto “a sus anchas”. Quiero con esto decir que no ocurre como en otras ocasiones, donde el espacio dedicado a la cabecera de columna era extremadamente justo, mostrándose incluso algunos títulos recortados. ¿Cómo

logramos esta característica?, pues mediante la utilización de dos líneas de código en el evento Load del formulario: la primera para indicarle al control que ajuste automáticamente el tamaño de las celdas con la propiedad AutoSizeColumnsMode; y la segunda en la propiedad Padding del estilo de la cabecera, donde usando un objeto de dicho tipo, conseguimos establecer un espacio alrededor del título y los límites de la celda de cabecera. Veámoslo en el fuente 5.

```
// establecer el tamaño automático para las celdas
this.dgvGrid.AutoSizeColumnsMode =
DataGridViewAutoSizeColumnsMode.DisplayedCells;
//....
// estilo para la cabecera
DataGridViewCellStyle styCabecera = new DataGridViewCellStyle();
//....
styCabecera.Padding = new Padding(10);
this.dgvGrid.ColumnHeadersDefaultCellStyle = styCabecera;
```

Fuente 5.

Cuando utilizamos un estilo aplicado a una columna que contiene valores nulos, podemos conseguir, a través de la propiedad NullValue del estilo, que estos se muestren con un texto informativo. Sin embargo, en la consulta SQL que estamos usando para estos ejemplos, el campo SpanishProductName contiene registros con valores que no son nulos, sino vacíos, por lo cual no tendría efecto el uso de la propiedad NullValue para el estilo de dicha columna.

Nos encontramos pues ante un candidato ideal para tratar mediante el evento CellFormatting. Todo lo que tenemos que hacer, es añadir el código del fuente 6 al manipulador de este evento, para que muestre un literal en el caso de que la celda de esta columna no vaya a mostrar valores, y de paso cambiamos su color de fondo.

```
// si la celda a dibujar es SpanishProductName...
if (this.dgvGrid.Columns[e.ColumnIndex].Name == "SpanishProductName")
{
   if (e.Value.ToString() == string.Empty)
   {
      e.CellStyle.BackColor = Color.Aqua;
      e.Value = "Producto sin descripción";
   }
}
```

Fuente 6.

Asignar una imagen a la celda

Si queremos otorgar mayor vistosidad a la información mostrada, podría interesarnos visualizar diferentes imágenes en las celdas de una columna, en función de una condición dada por el valor de uno de los campos, por ejemplo ListPrice.

Para manipular imágenes en un DataGridView disponemos de la clase DataGridViewImageColumn, que como su nombre indica, permite crear columnas cuyo contenido sea dicho tipo de dato. Por lo que en el evento Load procederemos a instanciar un objeto de esta clase, insertándolo en la colección de columnas de la cuadrícula, al lado de ListPrice. Posteriormente, en el evento CellFormatting, cuando detectemos que la celda de esta nueva columna va a ser dibujada, le asignaremos una imagen creando un objeto Bitmap a partir de un archivo gráfico, como vemos en el fuente 7.

```
private void Form1_Load(object sender, EventArgs e)
{
   //....
   // columna para mostrar imágenes
   DataGridViewImageColumn colDinero = new DataGridViewImageColumn();
   colDinero.Name = "colDinero";
   this.dgvGrid.Columns.Insert(3, colDinero);
}

private void dgvGrid_CellFormatting(object sender,
DataGridViewCellFormattingEventArgs e)
{
   //....
   //....
   // si la celda a dibujar corresponde a la columna de imágenes
   // añadir a la celda la imagen en función del valor del campo ListPrice
   if (this.dgvGrid.Columns[e.ColumnIndex].Name == "colDinero")
   {
      string sImagen;

      if ((decimal)this.dgvGrid.Rows[e.RowIndex].Cells["ListPrice"].Value > 1000)
      {
         sImagen = @"\euro.jpg";
      }
      else
      {
         sImagen = @"\dolar.jpg";
      }

      e.Value = new Bitmap(new Bitmap(Application.StartupPath + sImagen),
         this.dgvGrid.Rows[e.RowIndex].Cells["colDinero"].Size);
   }
}
```

Fuente 7.

La ejecución del control con esta nueva columna podemos verla en la figura.

Figura 4. DataGridView con columna de imagen.

Selección de celdas por columna

A poco que hayamos utilizado el control DataGridView, nos habremos percatado de que, por defecto, permite realizar una selección múltiple de todas las celdas de una fila con un solo clic en la cabecera de dicha fila. Sin embargo, en algunas situaciones sería deseable disponer de la capacidad de seleccionar todas las celdas de una columna al hacer clic en la cabecera de las mismas.

El comportamiento de este control, en lo que a selección múltiple de celdas se refiere, se encuentra gobernado por su propiedad SelectionMode, que corresponde al tipo enumerado DataGridViewSelectionMode, cuyos valores determinarán el modo en cómo las celdas son seleccionadas al hacer clic el usuario sobre ellas. La tabla 1 muestra una descripción de los miembros de esta enumeración.

Propiedad	**Descripción**
CellSelect	Selección por celdas independientes. Permite seleccionar una o varias.
ColumnHeaderSelect	Se seleccionará la columna al completo al hacer clic en su cabecera. También podemos seguir seleccionando celdas independientes.
FullColumnSelect	Se seleccionará la columna al completo al hacer clic en su cabecera, o bien en cualquier celda de esa columna.
FullRowSelect	Se seleccionará la fila al completo al hacer clic en la cabecera de fila, o bien en cualquier celda de esa fila.
RowHeaderSelect	Se seleccionará la fila al completo al hacer clic en su cabecera de fila. También podemos seguir seleccionando celdas independientes.

Tabla 1. Valores de la enumeración DataGridViewSelectionMode.

Como reza el título de este apartado, si queremos seleccionar todas las celdas de una columna al hacer clic en su cabecera, asignaremos el valor DataGridViewSelectionMode.ColumnHeaderSelect a la propiedad SelectionMode del control.

El efecto más inmediato que probablemente obtendremos, será un tremendo error al intentar ejecutar la aplicación. Esto es debido al hecho de que los objetos columna del control tienen por defecto, en su propiedad SortMode, el valor Automatic, lo cual es incompatible con la capacidad de selección por columna.

Para solucionar este entuerto bastará con recorrer la colección de columnas de la cuadrícula y cambiar el valor de esta propiedad a Programmatic, lo cual implicará que sea el código de la aplicación quien se encargue de las operaciones de ordenación. Ver el fuente 8.

```
// cambiar el modo de ordenación de cada
// columna a manual
foreach (DataGridViewColumn oColumna in this.dgvGrid.Columns)
{
   oColumna.SortMode = DataGridViewColumnSortMode.Programmatic;
}

// ahora ya se puede establecer la selección por columna
this.dgvGrid.SelectionMode = DataGridViewSelectionMode.ColumnHeaderSelect;
```

Fuente 8.

De este modo, como vemos en la figura 5, ya será posible la selección por columna.

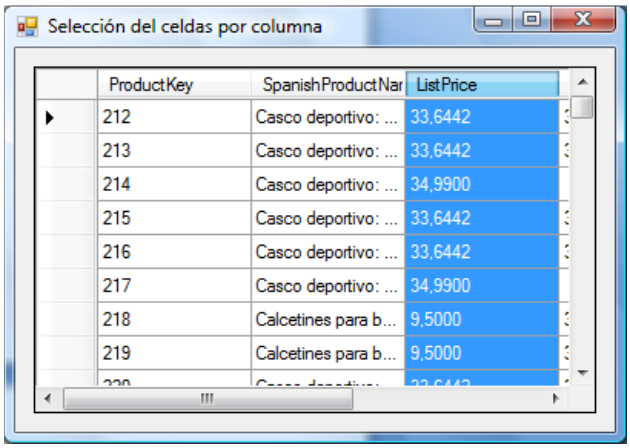

Figura 5. Selección por columna en el control.

Hemos conseguido pues, que al hacer clic en la celda de cabecera de una columna, esta quede seleccionada. Pero hemos ganado una funcionalidad y perdido otra: la

ordenación automática de columnas, lo que quiere decir que nos veremos obligados a implementar este aspecto por código.

El punto más idóneo para realizar estas operaciones será el evento ColumnHeaderMouseClick, producido al hacer clic sobre la cabecera de una columna; donde en primer lugar obtendremos, gracias al parámetro DataGridViewMouseCellEventArgs, y su propiedad ColumnIndex, la columna seleccionada para ordenar.

La columna que estaba previamente ordenada, si es que había alguna, se encontrará en la propiedad SortedColumn del control. Como siguiente paso, determinaremos la dirección en la que se van a ordenar los datos, pero sin ordenarlos realmente aún. Lo que haremos será guardar en una variable de tipo ListSortDirection esta dirección para el orden.

A continuación ordenaremos la columna correspondiente a la cabecera pulsada utilizando el método DataGridView.Sort, al que pasaremos como parámetro la columna a ordenar y su dirección.

Por último, en el caso de que existiese una columna anteriormente ordenada, y por lo tanto seleccionada, quitaremos dicho estado de selección asignando false a su propiedad Selected. Curiosamente, esta operación provocará una excepción, pero controlándola mediante un bloque try…catch, el estado de selección de la columna en cuestión finalmente quedará quitado. Veamos estas operaciones en el fuente 9.

```
private void dgvGrid_ColumnHeaderMouseClick(object sender,
DataGridViewCellMouseEventArgs e)
{
   // obtener las columnas pulsada y actualmente ordenada
   DataGridViewColumn colPulsada = this.dgvGrid.Columns[e.ColumnIndex];
   DataGridViewColumn colOrdenActual = this.dgvGrid.SortedColumn;

   // establecer un orden por defecto
   ListSortDirection xDireccionOrden = ListSortDirection.Ascending;

   // calcular qué orden vamos a aplicar:
   // ----------------------------------
   // si no hay columna ordenada actualmente se ordenará en ascendente
   if (colOrdenActual == null)
   {
      xDireccionOrden = ListSortDirection.Ascending;
   }
   else
   {
      // si hay columna ordenada actualmente:
      // ------------------------------------
      // si se ha pulsado la misma columna que ya estaba ordenada
```

```
        // se invierte el orden actual
        if (colPulsada == colOrdenActual &&
            this.dgvGrid.SortOrder == SortOrder.Ascending)
        {
            xDireccionOrden = ListSortDirection.Descending;
        }
        else
        {
            // en cualquier otro caso se establece orden ascendente
            xDireccionOrden = ListSortDirection.Ascending;
        }
    }

    // ordenar la columna
    this.dgvGrid.Sort(colPulsada, xDireccionOrden);

    // si había una columna previamente seleccionada
    // y es distinta de la que acabamos de ordenar
    if (colOrdenActual != null && colOrdenActual.Name != colPulsada.Name)
    {
        try
        {
            // quitar el estado de selección
            // de la columna anterior
            colOrdenActual.Selected = false;
        }
        catch
        { }
    }
}
```

Fuente 9.

En la figura 6 volvemos a observar el control en ejecución, permitiéndonos ya tanto seleccionar la columna como ordenarla.

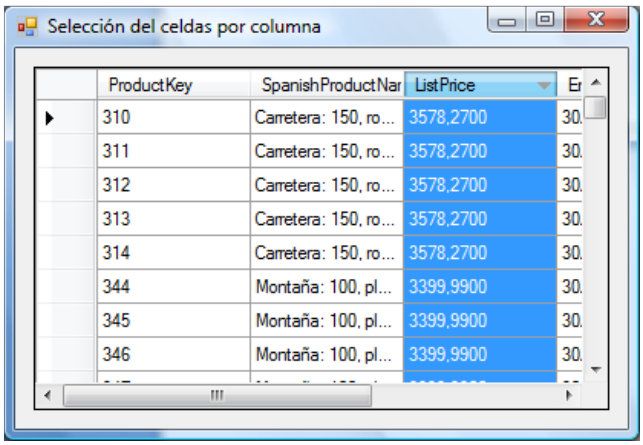

Figura 6. Seleccionar una columna y ordenarla.

Posicionamiento y bloqueo de columnas

Las clases DataGridView y DataGridViewColumn ofrecen al desarrollador un conjunto de propiedades destinadas a la manipulación de las posiciones de las columnas dentro del control.

Comenzaremos por la propiedad DataGridView.AllowUserToOrderColumns, de tipo boolean. Asignándole el valor true, conseguiremos que el usuario al hacer clic y arrastrar en la cabecera de una columna, cambie la posición de la misma con respecto a las demás. Cuando añadimos un nuevo control de este tipo al formulario, por defecto no podremos cambiar las posiciones de sus columnas, ya que el valor de esta propiedad es false.

Puede, sin embargo, que en lugar de dejar al usuario el movimiento de las posiciones de columnas, queramos establecerlo nosotros por código. Para ello contamos con la propiedad DataGridViewColumn.DisplayIndex, que permite asignar a la columna la posición de visualización, independientemente de la posición real que dicha columna tenga en la colección de columnas del control.

Por otro lado, también disponemos de la capacidad de bloquear o congelar una columna mediante su propiedad Frozen, consiguiendo, al asignarle el valor true, que dicha columna y las que se encuentren a su izquierda se mantengan fijas cuando el usuario utilice la barra de desplazamiento horizontal del DataGridView.

Como ejemplo ilustrativo de estas características, vamos a crear un formulario que contenga un TextBox por cada una de las columnas del grid, y un botón que al ser pulsado, asignará nuevas posiciones a las columnas a través de su propiedad DisplayIndex, en base a los números que hayamos introducido en los mencionados controles TextBox.

También añadiremos un control CheckBox al formulario, que al ser marcado, bloqueará las dos primeras columnas del DataGridView. Para hacer este efecto más palpable visualmente, ampliaremos la anchura de la segunda columna utilizando su propiedad DividerWidth. Finalmente, como mero efecto estético, cambiaremos el color de las líneas de la cuadrícula usando la propiedad GridColor del control. Todo ello lo vemos en el fuente 10, donde debemos aclarar que los nombres que hemos dado a los controles TextBox del formulario están compuestos por el prefijo txt más el nombre del campo; de ahí que al recorrer la colección de controles, cada vez que encontremos un TextBox, extraigamos el nombre del campo aplicando el método Substring a la propiedad Name del TextBox.

```
// cambiar las posiciones de visualización de las columnas
// utilizando su propiedad DisplayIndex
private void btnPosicionar_Click(object sender, EventArgs e)
```

```
{
   // si hay una columna bloqueada, no se puede
   // aplicar el cambio de posición con DisplayIndex
   if (this.dgvGrid.Columns[1].Frozen)
   {
      MessageBox.Show("Desbloquear columnas");
   }
   else
   {
      // recorrer los controles, obtener el valor de los textbox
      // y asignarlo como nueva posición de columna en el grid
      foreach (Control oControl in this.Controls)
      {
         if (oControl.GetType() == typeof(TextBox))
         {
            TextBox txtTexto = oControl as TextBox;
            this.dgvGrid.Columns[txtTexto.Name.Substring(3)].DisplayIndex =
               int.Parse(txtTexto.Text) - 1;
         }
      }
   }
}

// bloquear-desbloquear las dos primeras columnas,
// establecer un grosor para el borde de una columna
// y color para las líneas del control
private void chkBloquear_CheckedChanged(object sender, EventArgs e)
{
   if (this.chkBloquear.Checked)
   {
      this.dgvGrid.Columns[1].Frozen = true;
      this.dgvGrid.Columns[1].DividerWidth = 10;
      this.dgvGrid.GridColor = Color.DarkOrchid;
   }
   else
   {
      this.dgvGrid.Columns[0].Frozen = false;
      this.dgvGrid.Columns[1].Frozen = false;
      this.dgvGrid.Columns[1].DividerWidth = 0;
      this.dgvGrid.GridColor = SystemColors.ControlDark;
   }
}
```

Fuente 10.

Resulta interesante, una vez que hemos bloqueado las columnas, hacer un desplazamiento de las mismas, observando cómo la primera de ellas que no está

bloqueada se va ocultando al ser desplazada a la izquierda. Efecto que vemos en la figura 7.

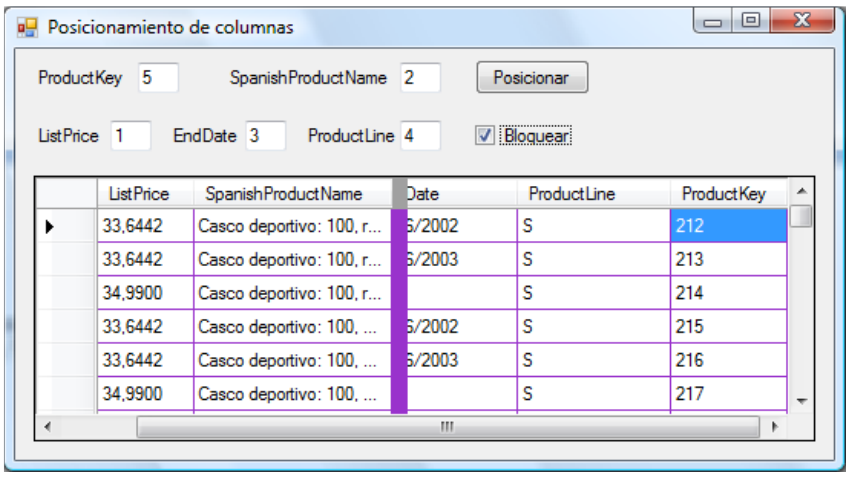

Figura 7. Cambio de posición visual y bloqueo de columnas.

Conclusiones

En el presente artículo hemos abordado diversas características del control DataGridView relacionadas con su capacidad de presentación de datos. Estas características nos permiten realizar operaciones tales como la aplicación de formato sobre los valores de las celdas, selección de columnas y cambio en el orden predeterminado en que las mismas se muestran al usuario, tanto por código como debido a la acción del propio usuario sobre los elementos del control en tiempo de ejecución. Confiamos en que todas estas posibilidades para obtener un mayor beneficio visual en nuestra cuadrícula de datos, resulten interesantes a los lectores que necesiten implementar este tipo de mejoras en sus aplicaciones.