El evento CellPainting del control DataGridView. Vía libre a la creatividad

Luis Miguel Blanco Ancos

CellPainting. Control total en el dibujo de celdas

Se trata de un evento que se produce para cada celda que va a dibujarse, con la particularidad de que el desarrollador dispone de pleno control sobre toda la operación de dibujo de la celda, siendo posible aplicarle efectos gráficos tales como degradados (fundidos) de colores, texturas, imágenes, etc.

CellPainting recibe como parámetro un objeto DataGridViewCellPaintingEventArgs, el cual nos servirá para obtener la información principal sobre la columna, fila y valor de celda. Pero además, entre sus propiedades encontraremos también aquellas que nos permitirán realizar todo el dibujo de la celda y su contenido al completo. La tabla 1 muestra las propiedades más destacables de esta clase.

Propiedad	**Descripción**
ColumnIndex	Índice de la colección de columnas del control, que indica la columna sobre la que se produce el evento
RowIndex	Índice de la colección de filas del control, que indica la fila sobre la que se produce el evento
CellBounds	Objeto Rectangle con la posición y dimensiones de la celda a pintar
State	Estado de la celda: seleccionada, visible, sólo lectura, etc
Graphics	Objecto Graphics conteniendo el contexto de dispositivo gráfico sobre el que realizar la operación de dibujo
Value	Valor para pintar en la celda
FormattedValue	Valor formateado para pintar en la celda
CellStyle	Estilo de la celda
Handled	Tipo boolean que permite informar al control si la operación de dibujo ha sido realizada manualmente mediante el código suministrado por el programador, o deberá ser el control quien realice este proceso

Tabla 1. Propiedades de la clase DataGridViewCellPaintingEventArgs.

Es momento, por lo tanto, de ponernos manos a la obra y crear un nuevo formulario al que añadiremos un DataGridView, que nos sirva para experimentar con esta característica. Como es habitual, todos los ejemplos de este artículo se encuentran disponibles en la dirección http://www.dotnetmania.com.

En este ejemplo aplicaremos la operación de dibujo manual a los campos ListPrice y Color, pertenecientes a la consulta SQL que mostraremos en el control de cuadrícula. Como paso previo, en el evento Load del formulario, después de asignar el DataSet al DataGridView, crearemos para los mencionados campos, sendos estilos con un tipo de letra que destaque notablemente del resto de columnas, lo que vemos en el fuente 1, donde adicionalmente, configuraremos el ajuste de tamaño automático para filas y columnas, de modo que el contenido de estos campos se muestre correctamente.

```
// obtención de datos
SqlConnection cnConexion = new SqlConnection();
cnConexion.ConnectionString = "Data Source=localhost;" +
   "Initial Catalog=AdventureWorksDW;" +
   "Integrated Security=True";
//....
string sSQL;
sSQL = "SELECT ProductKey, SpanishProductName, ListPrice, EndDate, Color ";
sSQL += "FROM DimProduct ";
sSQL += "WHERE ListPrice IS NOT NULL";

SqlCommand cmdComando = new SqlCommand(sSQL, cnConexion);
SqlDataAdapter daAdaptador = new SqlDataAdapter(cmdComando);
DataSet dsAdvWorks = new DataSet();
daAdaptador.Fill(dsAdvWorks, "DimProduct");

// establecer la fuente de datos para el DataGridView
this.dgvGrid.DataSource = dsAdvWorks;
this.dgvGrid.DataMember = "DimProduct";

// cambiar el tipo de letra a las columnas
// que vamos a usar en el evento CellPainting
DataGridViewCellStyle styListPrice = new DataGridViewCellStyle();
styListPrice.Font = new Font("Comic Sans MS", 16, FontStyle.Bold);
styListPrice.Format = "C";
this.dgvGrid.Columns["ListPrice"].DefaultCellStyle = styListPrice;

DataGridViewCellStyle styColor = new DataGridViewCellStyle();
styColor.Font = new Font("Papyrus", 16, FontStyle.Bold);
this.dgvGrid.Columns["Color"].DefaultCellStyle = styColor;

// establecer ajuste de altura automático para las filas
this.dgvGrid.AutoSizeRowsMode = DataGridViewAutoSizeRowsMode.AllCells;

// establecer ajuste de anchura automático para las columnas
this.dgvGrid.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;

// ocultar la primera columna del control
this.dgvGrid.Columns["ProductKey"].Visible = false;
```

Fuente 1.

A continuación pasamos a escribir el código de CellPainting, donde nuestro objetivo consistirá en pintar las mencionadas celdas, dotándolas de un efecto de fundido de colores que conseguiremos gracias al uso de un objeto PathGradientBrush. Esta clase no está accesible por defecto, así que deberemos declarar al comienzo del archivo de código el espacio de nombres System.Drawing.Drawing2D, en el cual se encuentra ubicada.

Un objeto PathGradientBrush, brevemente descrito, rellena una porción de la superficie del formulario tomando como base a una figura, por ejemplo, el rectángulo correspondiente a una celda del control, y le aplica un color central que será rodeado por otro conjunto de colores situados en un array.

La forma en que los colores se distribuirán y fundirán al aplicar este objeto se determina gracias a un array de coordenadas (objetos Point), que en este caso estará compuesto por los mismos puntos de coordenadas que el rectángulo de la celda a pintar.

Entre los diversos estados que puede tomar una celda se encuentran los correspondientes a normal y seleccionado. Esto lo sabremos preguntando a la propiedad DataGridViewCellPaintingEventArgs.State, de forma que a la hora de pintar la celda, podremos asignarle un conjunto de colores distinto según el mencionado estado.

En el momento de pintar la celda, necesitaremos un objeto de la clase Graphics, que representa el contexto de dispositivo gráfico o superficie sobre la que vamos a dibujar, es decir, el formulario. El acceso a este objeto lo conseguimos gracias a la propiedad DataGridViewCellPaintingEventArgs.Graphics, realizando el dibujo de la celda mediante a la llamada a su método FillRectangle, que recibe como parámetros el objeto PathGradientBrush para aplicar el fundido de colores, y el rectángulo de la celda que está en la propiedad DataGridViewCellPaintingEventArgs.CellBounds. El contorno o línea delimitadora de los bordes de la celda será dibujado utilizando el método Graphics.DrawRectangle, aplicando un color al borde sobre el rectángulo de la celda.

Pero esto no es suficiente, ya que solamente hemos dibujado el borde y colores de fondo de la celda. Aún nos queda el texto que representa el valor de la celda, que obtenemos de la propiedad Value o FormattedValue, pertenecientes al parámetro DataGridViewCellPaintingEventArgs del evento. Sin embargo, no podemos utilizar directamente esta propiedad; recordemos que el evento CellPainting nos obliga a realizar manualmente todas las operaciones de dibujo, por lo que deberemos convertir a modo gráfico la cadena a dibujar siguiendo estos pasos:

Tras comprobar que existe realmente un valor a dibujar, tenemos que calcular las dimensiones del texto utilizando el método estático TextRenderer.MeasureText, al que pasaremos como parámetro el valor de la celda y el tipo de letra que usaremos para

dibujarlo. Para este último, emplearemos el objeto Font situado en el estilo de la columna correspondiente a la celda. El resultado obtenido del método será un objeto Size.

A continuación, empleando el también método estático TextRenderer.DrawText, será cuando realmente dibujemos el texto, pasando como parámetro aquellas propiedades del objeto DataGridViewCellPaintingEventArgs necesarias para esta operación: el objeto Graphics; la propiedad FormattedValue, más recomendable que su homóloga Value, fundamentalmente para aquellos casos de columnas con valores numéricos o de fecha formateados; CellBounds, como superficie o rectángulo sobre la que dibujar el texto; y finalmente los valores del estilo –propiedad CellStyle-, que contienen el tipo de letra y su color.

Como última acción en el código de este evento, tenemos que asignar el valor true a la propiedad DataGridViewCellPaintingEventArgs.Handled, puesto que es necesario informar al control de que nosotros hemos sido los encargados de realizar la operación de dibujo de la celda, o de lo contrario el control hará caso omiso de nuestro código, realizando él solito las operaciones de dibujo.

El fuente 2 muestra el código del evento CellPainting que acabamos de explicar en los párrafos anteriores.

```
private void dgvGrid_CellPainting(object sender, DataGridViewCellPaintingEventArgs
e)
{
   // comprobar que la fila y columna están dentro del rango
   // y que la columna corresponde a una de las que vamos
   // a pintar
   if (e.ColumnIndex > 0 && e.RowIndex >= 0 &&
      (this.dgvGrid.Columns[e.ColumnIndex].Name == "Color" |
      this.dgvGrid.Columns[e.ColumnIndex].Name == "ListPrice"))
   {
      // crear un array con las coordenadas
      // sobre las que aplicaremos los colores con fundido
      // basadas en la posición y dimensión de la celda
      Point[] aPuntos = new Point[] {new Point(e.CellBounds.X,e.CellBounds.Y),
         new Point(e.CellBounds.X, e.CellBounds.Y + e.CellBounds.Height),
         new Point(e.CellBounds.X + e.CellBounds.Width, e.CellBounds.Y +
e.CellBounds.Height),
         new Point(e.CellBounds.X + e.CellBounds.Width, e.CellBounds.Y)};

      PathGradientBrush pgbBrush;
      Color[] aColores;

      // crear el objeto brush con efecto de fundido
      // utilizando una combinación de colores diferente
      // en función de si la celda está o no seleccionada
```

```
        if ((e.State & DataGridViewElementStates.Selected) ==
            DataGridViewElementStates.Selected)
        {
            aColores = new Color[] {Color.LightYellow,
                Color.DarkTurquoise, Color.MediumSpringGreen, Color.LightSteelBlue};

            pgbBrush = new PathGradientBrush(aPuntos);
            pgbBrush.CenterColor = Color.Lavender;
            pgbBrush.SurroundColors = aColores;
        }
        else
        {
            aColores = new Color[] {Color.BlanchedAlmond,
                Color.LightSalmon, Color.LightYellow, Color.DarkOrange};

            pgbBrush = new PathGradientBrush(aPuntos);
            pgbBrush.CenterColor = Color.LawnGreen;
            pgbBrush.SurroundColors = aColores;
        }

        // aplicar el objeto brush con el efecto de colores fundidos
        // y dibujar un borde para el rectángulo de la celda
        e.Graphics.FillRectangle(pgbBrush, e.CellBounds);
        e.Graphics.DrawRectangle(new Pen(Color.DarkMagenta, 3), e.CellBounds);

        // si la celda tiene valor
        if (e.Value != null)
        {
            // calcular el tamaño del texto
            Size szTexto;
            szTexto = TextRenderer.MeasureText(e.Value.ToString(), e.CellStyle.Font);

            // dibujar el texto
            TextRenderer.DrawText(e.Graphics,
                e.FormattedValue.ToString(),
                e.CellStyle.Font,
                e.CellBounds,
                e.CellStyle.ForeColor);
        }

        e.Handled = true;
    }
}
```

Fuente 2.

La figura 1 muestra el control grid resultante tras la aplicación de este proceso de dibujo personalizado con CellPainting, donde podemos observar las celdas que hemos pintado tanto en estado normal como seleccionado.

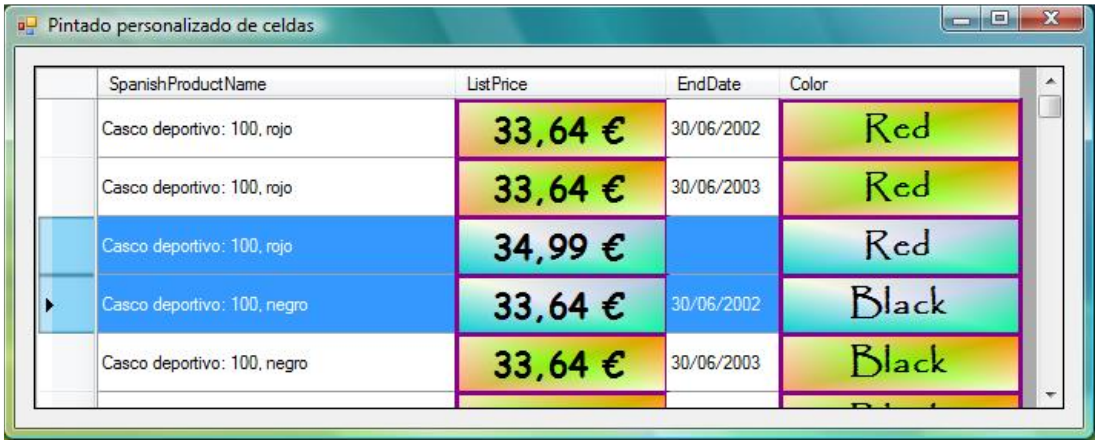

Figura 1. Celdas pintadas mediante el evento CellPainting.

Cambiando la alineación en el contenido de las celdas

El método TextRenderer.DrawText, como acabamos de comprobar, dibuja el texto centrado en la celda de manera predeterminada, pero supongamos que es necesaria una alineación diferente.

Para conseguir dicho objetivo, añadiremos a DrawText como último parámetro el tipo enumerado TextFormatFlags, cuyos valores podemos utilizar de forma combinada para crear diferentes efectos de formato, alineación, recorte de fuente, etc.

Por ejemplo, si queremos que el texto de la celda quede alineado a la parte superior de la misma, utilizaremos el código del fuente 3.

```
TextRenderer.DrawText(e.Graphics,
   e.FormattedValue.ToString(),
   e.CellStyle.Font,
   e.CellBounds,
   e.CellStyle.ForeColor,
   TextFormatFlags.Top);
```

Fuente 3.

Pero en el DataGridView que estamos desarrollando para este ejemplo estamos dibujando celdas para dos columnas diferentes mediante CellPainting. Supongamos que es necesario aplicar una alineación distinta para cada una de ellas. Esta es una situación que podemos solventar en el evento Load del formulario, asignando a la propiedad Tag de los estilos creados para estas columnas, el valor o combinación de valores de TextFormatFlags que necesitemos para establecer la alineación o efectos para el texto.

La propiedad DataGridViewCellStyle.Tag es de tipo object, lo cual la convierte en un estupendo “cajón de sastre” para almacenar valores de cualquier tipo, en función del problema a resolver en las más diversas situaciones.

Por lo tanto, como hemos dicho, si para cada estilo asignamos por separado los valores de alineación a la propiedad Tag, posteriormente, la aplicación de dichos valores en CellPainting se simplifica enormemente cuando dibujamos el texto con el método DrawText, como vemos en el fuente 4.

```
private void Form1_Load(object sender, EventArgs e)
{
   //....
   styListPrice.Tag = TextFormatFlags.Right | TextFormatFlags.VerticalCenter;
   //....
   styColor.Tag = TextFormatFlags.Left | TextFormatFlags.VerticalCenter;
   //....
}

private void dgvGrid_CellPainting(object sender, DataGridViewCellPaintingEventArgs
e)
{
   //....
   TextRenderer.DrawText(e.Graphics,
        e.FormattedValue.ToString(),
        e.CellStyle.Font,
        e.CellBounds,
        e.CellStyle.ForeColor,
        (TextFormatFlags)e.CellStyle.Tag);
   //....
}
```

Fuente 4.

En la figura 2 podemos apreciar el resultado de esta técnica para alinear los valores.

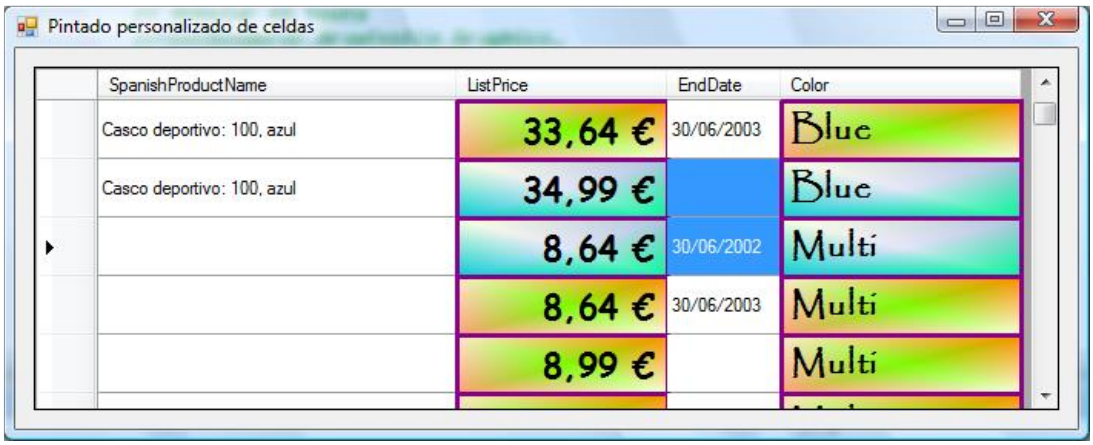

Figura 2. Pintando celdas con distinta alineación desde CellPainting.

Otros efectos sobre el contenido

Vamos a continuación, a introducir una nueva variante en este escenario de trabajo. Imaginemos que no utilizamos el ajuste automático de anchura de columnas para el control, por lo que borramos la línea de código que hace uso de la propiedad DataGridView.AutoSizeColumnsMode.

Esta acción tiene como efecto que el usuario pueda cambiar el ancho de las columnas, y al reducir su tamaño, el texto de estas quede oculto en algunos casos. El efecto conseguido por el recorte del texto puede no resultar muy agradable, por lo que si aplicamos el valor TextFormatFlags.WordEllipsis, agregando este efecto a los demás ya existentes que se utilizan en el método DrawText, el resultado será la visualización de puntos suspensivos en aquellas celdas que no puedan mostrar su contenido al completo. Este efecto lo podemos ver en la figura 3.

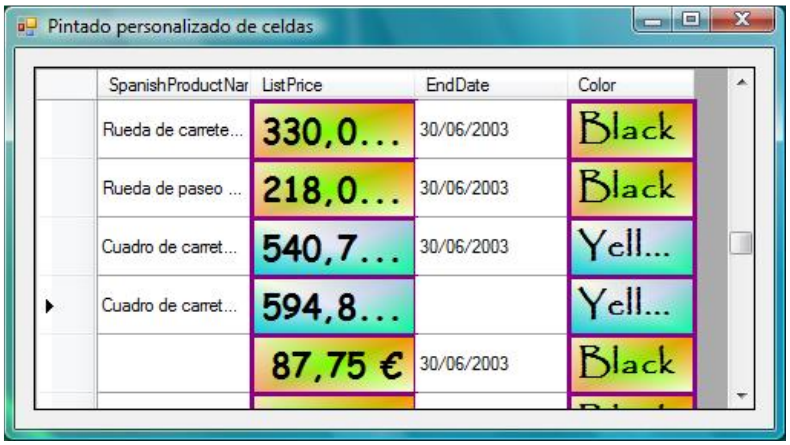

Figura 3. Puntos suspensivos en las celdas que no muestran su texto al completo.

Y ya para terminar con el tipo TextFormatFlags, vamos a comentar otro efecto realmente curioso. El valor NoClipping de esta enumeración tiene como finalidad no recortar los extremos del texto cuando reducimos el tamaño de la celda. Si combinamos este valor con HorizontalCenter, como muestra el fuente 5, obtendremos el interesante efecto de la figura 4, donde los extremos de texto de la celda "invaden" las celdas adyacentes.

```
TextRenderer.DrawText(e.Graphics,
   e.FormattedValue.ToString(),
   e.CellStyle.Font,
   e.CellBounds,
   e.CellStyle.ForeColor,
   TextFormatFlags.HorizontalCenter | TextFormatFlags.NoClipping);
```

Fuente 5.

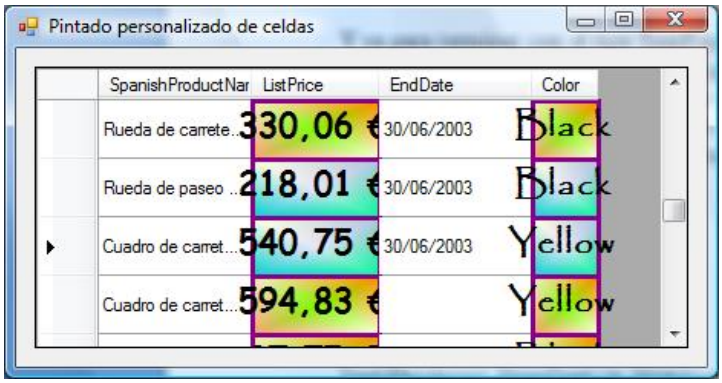

Figura 4. Los extremos del texto en las celdas no se recortan.

CellPainting en las cabeceras de columna. Indicando el orden de los registros con una imagen propia

Para aplicar el evento CellPainting en las cabeceras del control, emplearemos la misma lógica que en las celdas de datos, pero debemos tener en cuenta que el comportamiento de una cabecera, cuando el usuario hace clic sobre ella, consiste en ordenar sus celdas mostrando además un pequeño icono que indica el orden establecido.

Si aplicamos una operación de dibujo personalizado a la celda de cabecera con CellPainting, al hacer clic en ella para ordenar sus datos, estos efectivamente se ordenarán, pero el icono informativo del tipo de orden no se mostrará, ya que cuando realizamos el dibujo de una celda con este evento, como ya sabemos, la responsabilidad de pintar hasta el último elemento necesario recae sobre el programador.

Dado que el código a utilizar para pintar una celda de cabecera es muy parecido al de una celda normal de datos, dentro del fuente 6, en lo que se refiere al código del evento CellPainting, nos centraremos sólo en aquellos aspectos aplicables para este caso concreto.

Podemos ver, en primer lugar, la necesidad de comprobar que el índice de la columna sea mayor o igual a cero, el de fila menor de cero, y el nombre de la columna a dibujar corresponda a la que necesitamos pintar de forma personalizada.

A continuación debemos averiguar el nombre de la columna pulsada, disponible en la variable sCabeceraPulsada, que hemos declarado con ámbito a nivel de la clase del formulario. Dicha variable la asignamos en otro evento: ColumnHeaderMouseClick, provocado cuando el usuario hace clic en una cabecera de columna.

Después de dibujar el título de la celda alineado a la izquierda, comprobaremos si la columna va a ordenarse en alguna dirección. En caso afirmativo, disponemos de dos imágenes agregadas al proyecto como recursos incrustados (este valor lo establecemos en la ventana de propiedades de la imagen, dentro de la propiedad

"Acción de generación"). Según el orden, obtendremos la imagen correspondiente, asignándola a una variable de tipo Bitmap.

No obstante, es posible que la celda necesite dibujarse, pero no porque se haya pulsado la cabecera, en cuyo caso la propiedad DataGridView.SortOrder contendrá el valor None, y la variable Bitmap estará a null. Por dicho motivo, tenemos que comprobar que exista realmente una imagen a dibujar, y en ese caso, dibujarla en la parte derecha de la celda utilizando el método Graphics.DrawImage, calculando la posición en la cual dibujar, a partir del tamaño de la propia imagen y las propiedades del rectángulo de la cabecera.

```
string sCabeceraPulsada = string.Empty;
//....
private void Form1_Load(object sender, EventArgs e)
{
   //....
   this.dgvGrid.EnableHeadersVisualStyles = false;

   // crear un estilo para las cabeceras del control
   DataGridViewCellStyle styCabecera = new DataGridViewCellStyle();
   styCabecera.BackColor = Color.BlanchedAlmond;
   styCabecera.ForeColor = Color.DarkBlue;
   styCabecera.Alignment = DataGridViewContentAlignment.MiddleLeft;
   styCabecera.Font = new Font("Century Schoolbook", 10, FontStyle.Bold);
   styCabecera.Padding = new Padding(10);

   this.dgvGrid.ColumnHeadersDefaultCellStyle = styCabecera;

   // ampliar la anchura de la columna
   // sobre la que vamos a dibujar la cabecera
   this.dgvGrid.Columns["ProductKey"].Width += 40;
}

// cada vez que hacemos clic en la cabecera
// guardamos el nombre de la columna pulsada
private void dgvGrid_ColumnHeaderMouseClick(object sender,
DataGridViewCellMouseEventArgs e)
{
   sCabeceraPulsada = this.dgvGrid.Columns[e.ColumnIndex].Name;
}

private void dgvGrid_CellPainting(object sender, DataGridViewCellPaintingEventArgs
e)
{
   // comprobar si la cabecera a pintar es la que necesitamos
   if (e.ColumnIndex >= 0 &&
      e.RowIndex < 0 &&
```

```
    this.dgvGrid.Columns[e.ColumnIndex].Name == "ProductKey")
  {
    //....
    // dibujar el título de la cabecera
    // alineado a la izquierda
    TextRenderer.DrawText(e.Graphics,
      e.FormattedValue.ToString(),
      e.CellStyle.Font,
      e.CellBounds,
      e.CellStyle.ForeColor,
      TextFormatFlags.Left | TextFormatFlags.VerticalCenter);

    // si la cabecera pulsada es la que necesitamos,
    // en función del tipo de ordenación,
    // obtener la imagen del recurso incrustado en el proyecto
    Bitmap bmpImagen = null;

    if (sCabeceraPulsada == "ProductKey")
    {
      switch (this.dgvGrid.SortOrder)
      {
        case SortOrder.Ascending:
          bmpImagen = new Bitmap(GetType(), "FlechaArriba.png");
          break;

        case SortOrder.Descending:
          bmpImagen = new Bitmap(GetType(), "FlechaAbajo.png");
          break;
      }
    }

    // pintar la imagen en la celda de la cabecera
    // si realmente se ha establecido un orden
    if (bmpImagen != null)
    {
      e.Graphics.DrawImage(bmpImagen,
        e.CellBounds.Right - (bmpImagen.Width + e.CellStyle.Padding.Left),
        e.CellBounds.Y + e.CellStyle.Padding.Top);
    }

    e.Handled = true;
  }
}
```

Fuente 6.

La figura 5 muestra el control en ejecución, donde podemos observar la celda de cabecera con las operaciones que acabamos de describir.

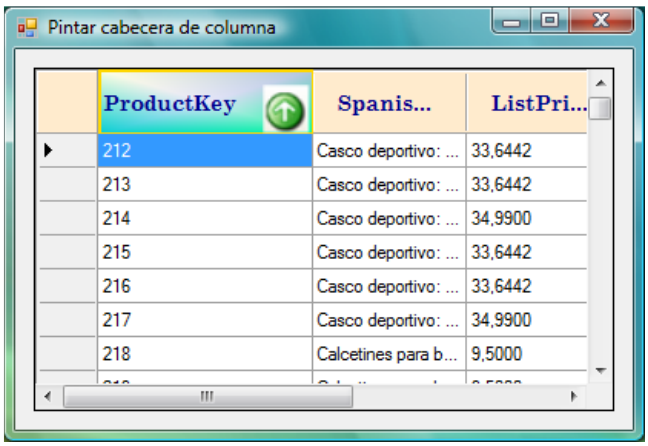

Figura 5. Cabecera de columna con dibujo personalizado de icono de orden y fundido de colores.

CellPainting en las cabeceras de fila. Indicando el número de fila a todo color

Y si hemos podido pintar con alegres y gráciles colorines una cabecera del control, las cabeceras de las filas no van a ser menos, ni se van a librar de nuestro arrebato artístico, así que tomando nuestros botes de pintura y pinceles digitales, pongámonos a trabajar.

La técnica que vamos a emplear, como puede intuir el lector, será muy similar a los otros casos anteriores, con la salvedad de que ahora vamos a dibujar el número de cada fila, por lo que tendremos que calcular dicho valor basándonos en la propiedad DataGridViewCellPaintingEventArgs.RowIndex, y comprobar que la celda sobre la que vamos a realizar las operaciones pertenece a una fila, o lo que es lo mismo, al tomar el parámetro DataGridViewCellPaintingEventArgs del evento CellPainting, la propiedad ColumnIndex deberá ser menor de cero, y la propiedad RowIndex mayor o igual que cero.

Los efectos de colores fundidos y cálculo de coordenadas para pintar el objeto PathGradientBrush serán los mismos que en los anteriores ejemplos, por lo que el cambio en este caso concreto, consistirá en crear un tipo de letra explícitamente para dibujar el número en la celda. Por este motivo, en el fuente 7 se exponen aquellas instrucciones que representan cambios con respecto a los anteriores ejemplos.

```
private void dgvGrid_CellPainting(object sender, DataGridViewCellPaintingEventArgs
e)
{
   // si la celda a pintar es cabecera de fila
   if (e.ColumnIndex < 0 && e.RowIndex >= 0)
   {
      // calcular el número a pintar en la celda
```

```
        int nNumeroFila = e.RowIndex + 1;
        //....
        // crear la fuente a usar para dibujar el número
        Font oFont = new Font("Comic Sans MS", 12, FontStyle.Italic);
        // calcular el tamaño del texto
        Size szTexto = TextRenderer.MeasureText(nNumeroFila.ToString(), oFont);

        // dibujar el número
        TextRenderer.DrawText(e.Graphics,
           nNumeroFila.ToString(),
           oFont,
           e.CellBounds,
           e.CellStyle.ForeColor);

        e.Handled = true;
    }
}
```

Fuente 7.

Para comprobar la aplicación de este efecto sobre esta parte del control, veamos la figura 6.

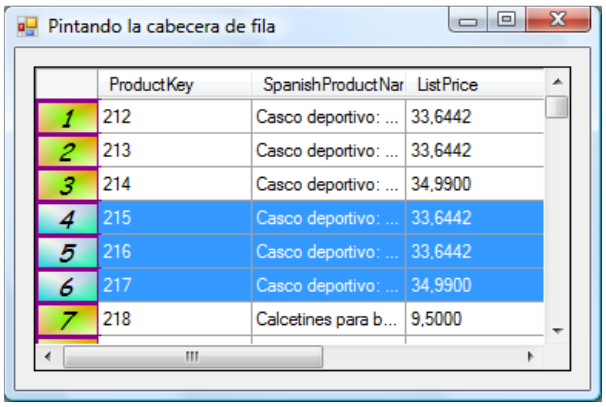

Figura 6. Personalizando el dibujo de las cabeceras de fila.

Acabando la pintura

Llegamos al punto final de este artículo, en el que hemos realizado un repaso del evento CellPainting, perteneciente al control DataGridView, como medio para aplicar ciertos efectos avanzados de presentación sobre las celdas del control, a fin de lograr un resultado más agradable a la vista de nuestros usuarios. Que ustedes lo disfruten.