

# DataGridView y Estilos. Presentaciones de datos visualmente atractivas

Luis Miguel Blanco Ancos

## Configuración predeterminada. Visualización básica

Cuando empleamos un control DataGridView con su configuración de propiedades por defecto, el aspecto de los datos ofrecidos es correcto. No obstante, de la austeridad en su presentación se desprende la sensación de que con ciertos retoques estéticos, aunque los datos sigan siendo los mismos, el control ganaría enormemente en atractivo.

## Estilos. Los ornamentos del DataGridView

Para paliar esta carencia de vistosidad, es momento de que entren en el terreno de juego unos elementos que resultarán una inestimable ayuda en el trabajo cotidiano con el control DataGridView: los estilos.

Un estilo es un objeto de la clase DataGridViewCellStyle, que contiene una serie de propiedades relacionadas con los aspectos visuales de una celda, tales como colores, alineación, tipo de letra, etc. Actuando a todos los efectos como una plantilla de presentación de datos, que permite al programador aplicar uniformemente un conjunto de características visuales sobre la totalidad del control DataGridView, o sólo en determinadas regiones del mismo. En la tabla 1 podemos ver las propiedades más importantes de esta clase.

Propiedad	Descripción
BackColor	Color de fondo para la celda
ForeColor	Color de primer plano para la celda
SelectionBackColor	Color de fondo para la celda en estado seleccionado
SelectionForeColor	Color de primer plano para la celda en estado seleccionado
Format	Cadena de formato para personalizar la presentación de datos numéricos, fechas, etc.
Alignment	Valor de la enumeración DataGridViewContentAlignment, que utilizaremos para establecer la alineación del contenido de la celda
Font	Tipo de letra con el que se mostrará el texto de la celda
WrapMode	En celdas con texto extenso, esta propiedad permite dividir dicho texto en varias líneas o truncarlo
NullValue	Tipo Object que utilizaremos para mostrar el valor de la celda cuando el campo original del origen de datos sea nulo
Padding	Establece el espacio de relleno para los márgenes de la celda

Tabla 1. Propiedades de la clase DataGridViewCellStyle

### Confección de estilos. Nuestro primer estilo

Daremos comienzo a las técnicas que debemos emplear en la creación y uso de estilos con un sencillo ejemplo. El lector puede encontrar todos los ejemplos desarrollados en este artículo en la dirección <http://www.dotnetmania.com>. En primer lugar crearemos un nuevo proyecto Windows, en cuyo formulario predeterminado añadiremos un DataGridView. A continuación escribiremos en el evento Load el código del fuente 1, necesario para conectarnos a una base de datos y enviar una consulta que nos devuelva el consabido conjunto de resultados.

```
// obtención de datos
SqlConnection cnConexion;
cnConexion = new SqlConnection();
cnConexion.ConnectionString = "Data Source=localhost;" +
    "Initial Catalog=AdventureWorksDW;" +
    "Integrated Security=True";

string sSQL;
sSQL = "SELECT ProductKey, SpanishProductName, ListPrice, EndDate, ProductLine ";
sSQL += "FROM DimProduct ";
sSQL += "WHERE ListPrice IS NOT NULL";

SqlCommand cmdComando;
cmdComando = new SqlCommand(sSQL, cnConexion);

SqlDataAdapter daAdaptador;
daAdaptador = new SqlDataAdapter(cmdComando);

DataSet dsAdvWorks;
dsAdvWorks = new DataSet();

daAdaptador.Fill(dsAdvWorks, "DimProduct");
```

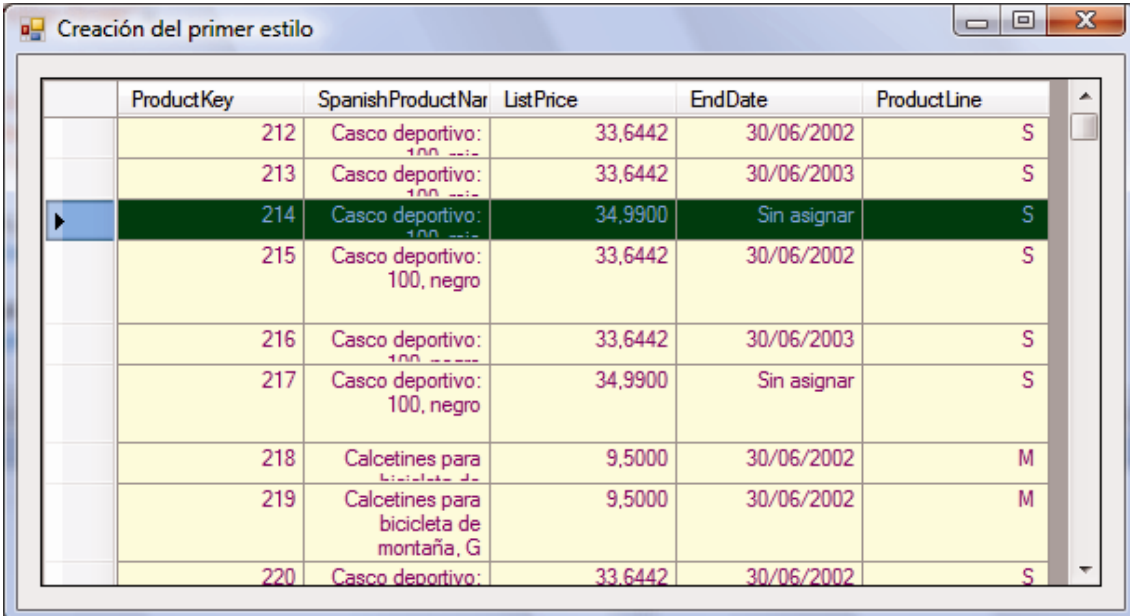
Fuente 1.

Una vez obtenido el DataSet con los registros a mostrar dentro del correspondiente DataTable, asignaremos esta fuente de datos al DataGridView utilizando sus propiedades DataSource y DataMember. A continuación instanciaremos un objeto DataGridViewCellStyle, estableciendo valores en sus principales propiedades, lo que nos permitirá modificar las características visuales predeterminadas de los datos mostrados. Finalmente asignaremos el objeto de estilo a la propiedad DefaultCellStyle del control, como vemos en el fuente 2, mientras que el resultado podemos verlo en la figura 1.

```
// establecer la fuente de datos para el DataGridView
```

```
this.dgvGrid.DataSource = dsAdvWorks;  
this.dgvGrid.DataMember = "DimProduct";  
  
// creación de estilo para el control DataGridView  
DataGridViewCellStyle styEstilo;  
styEstilo = new DataGridViewCellStyle();  
styEstilo.BackColor = Color.LightYellow;  
styEstilo.ForeColor = Color.DarkViolet;  
styEstilo.Alignment = DataGridViewContentAlignment.TopRight;  
styEstilo.NullValue = "Sin asignar";  
styEstilo.SelectionBackColor = Color.DarkGreen;  
styEstilo.SelectionForeColor = Color.LightSkyBlue;  
styEstilo.WrapMode = DataGridViewTriState.True;  
  
this.dgvGrid.DefaultCellStyle = styEstilo;
```

Fuente 2.



ProductKey	SpanishProductName	ListPrice	EndDate	ProductLine
212	Casco deportivo: 100, negro	33,6442	30/06/2002	S
213	Casco deportivo: 100, negro	33,6442	30/06/2003	S
214	Casco deportivo: 100, negro	34,9900	Sin asignar	S
215	Casco deportivo: 100, negro	33,6442	30/06/2002	S
216	Casco deportivo: 100, negro	33,6442	30/06/2003	S
217	Casco deportivo: 100, negro	34,9900	Sin asignar	S
218	Calcetines para bicileta de	9,5000	30/06/2002	M
219	Calcetines para bicicleta de montaña, G	9,5000	30/06/2002	M
220	Casco deportivo:	33,6442	30/06/2002	S

Figura 1. Creación de un primer estilo.

Aparte del evidente cambio en la combinación de colores para los estados normal y seleccionado de las celdas, existen algunos aspectos adicionales dignos de mención, como sería el tratamiento del texto mostrado. Por defecto, el contenido de las celdas queda alineado a la izquierda, pero en esta ocasión, utilizando la enumeración `DataGridViewContentAlignment`, logramos que se sitúe en la parte superior derecha.

Respecto a la columna `SpanishProductName` habremos notado que es un campo con valores de longitud superior al resto. Gracias a la propiedad `WrapMode` del estilo, conseguimos que su texto se divida en varias líneas, de forma que encaje

adecuadamente en la celda. Esto podemos comprobarlo aumentando manualmente la altura de algunas filas en la cuadrícula.

Finalmente nos encontramos con el caso de registros conteniendo campos con valores nulos, como ocurre en la columna EndDate, para la que utilizamos una cadena con un literal informativo de tal situación en la propiedad NullValue del estilo.

Como puede comprobar el lector, establecer un estilo básico en un control DataGridView resulta una operación muy sencilla de realizar.

### Aplicación de estilos por regiones

Acabamos de ver que el área de actuación de un estilo, cuando es asignado a la propiedadDefaultCellStyle de un DataGridView, abarca todas las celdas de datos del control. Pero podemos ser más selectivos en el uso de un estilo, haciendo que este afecte a una zona más concreta, las cabeceras de columna por ejemplo.

Esta funcionalidad la conseguiremos asignando el estilo a la propiedad ColumnHeadersDefaultCellStyle del control, como vemos en el fuente 3, donde empleamos un tipo de letra distinto del resto de celdas y establecemos una alineación a la derecha usando la propiedad Alignment del estilo, de forma que el texto de las cabeceras y el del resto de celdas queden alineados en extremos opuestos.

```
// creación de estilo para las cabeceras del DataGridView
```

```
DataGridViewCellStyle styEstilo;  
styEstilo = new DataGridViewCellStyle();  
styEstilo.BackColor = Color.LightBlue;  
styEstilo.ForeColor = Color.DarkOliveGreen;  
styEstilo.Font = new Font("Bradley Hand ITC", 10, FontStyle.Bold);  
styEstilo.Alignment = DataGridViewContentAlignment.BottomRight;
```

```
// asignar estilo a las cabeceras del control
```

```
this.dgvGrid.ColumnHeadersDefaultCellStyle = styEstilo;
```

Fuente 3.

Sin embargo, al ejecutar la aplicación, los colores definidos para la cabecera del control con toda seguridad no se mostrarán, permaneciendo la combinación predeterminada original.

La respuesta a este aparentemente anómalo comportamiento la encontramos en la propiedad DataGridView.EnableHeadersVisualStyles, la cual tiene por defecto el valor true, significando esto que las cabeceras del control utilizarán la combinación de colores del tema actualmente seleccionado en el sistema operativo.

Asignando false a esta propiedad lograremos que las cabeceras se muestren con los colores establecidos en nuestro estilo. Adicionalmente, para comprobar que la

alineación del texto en la cabecera se realiza en la parte inferior derecha de sus celdas, configuraremos el control para establecer por código la altura de las mismas, utilizando para ello las propiedades `ColumnHeadersHeightSizeMode` y `ColumnHeadersHeight`, como vemos en el fuente 4.

```
//....
// establecer altura para las cabeceras de columna
this.dgvGrid.ColumnHeadersHeightSizeMode =
DataGridViewColumnHeadersHeightSizeMode.EnableResizing;
this.dgvGrid.ColumnHeadersHeight = 75;

// deshabilitar el uso de estilos del tema actual
this.dgvGrid.EnableHeadersVisualStyles = false;
```

Fuente 4.

El control resultante lo vemos en la figura 2.

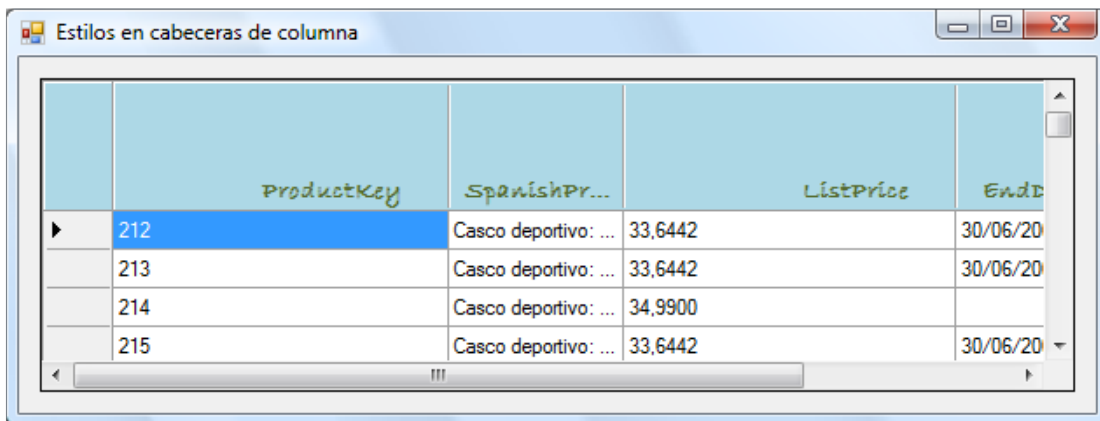


Figura 2. Estilo aplicado a las cabeceras del control.

Para establecer estilos en el resto de regiones del `DataGridView` tendremos que asignar el objeto de estilo a la propiedad adecuada del control. La tabla 2 relaciona las propiedades con las que tendremos que trabajar para aplicar estilos a las diferentes regiones de la cuadrícula.

Propiedad	Región de <code>DataGridView</code> afectada
<code>DefaultCellStyle</code>	Todas las celdas del control excepto las cabeceras de columna y fila
<code>Columns</code> . Colección de objetos <code>DataGridViewColumn</code>	Una o varias columnas del control
<code>ColumnHeadersDefaultCellStyle</code>	Cabeceras de columna
<code>Rows</code> . Colección de objetos <code>DataGridViewRow</code>	Una o varias filas del control
<code>RowsDefaultCellStyle</code>	Todas las filas del control
<code>AlternatingRowsDefaultCellStyle</code>	Filas alternas del control

RowHeadersDefaultCellStyle	Cabeceras de fila
----------------------------	-------------------

Tabla 2. Propiedades para aplicar estilos por regiones del DataGridView.

En el caso de las colecciones Columns y Rows, el estilo se aplica por separado a la propiedad DefaultCellStyle de cada uno de los objetos integrantes de la colección que necesiten el estilo.

Si por ejemplo, además de los estilos que ya hemos establecido previamente sobre las celdas de datos y columnas, necesitamos facilitar la legibilidad de las filas, aplicaremos un nuevo estilo mediante la propiedad AlternatingRowsDefaultCellStyle, lo que se demuestra en el fuente 5, donde además, también creamos y asignamos un estilo para la columna ListPrice.

// estilo para las filas alternas

```
DataGridViewCellStyle styAlterno;  
styAlterno = new DataGridViewCellStyle();  
styAlterno.BackColor = Color.DarkViolet;  
styAlterno.ForeColor = Color.LightYellow;  
styAlterno.NullValue = "SIN VALOR";  
this.dgvGrid.AlternatingRowsDefaultCellStyle = styAlterno;
```

// estilo para una columna independiente

```
DataGridViewCellStyle styColumna;  
styColumna = new DataGridViewCellStyle();  
styColumna.BackColor = Color.LightSeaGreen;  
styColumna.ForeColor = Color.FloralWhite;  
styColumna.Font = new Font("Tahoma", 12, FontStyle.Bold);  
styColumna.Alignment = DataGridViewContentAlignment.MiddleCenter;  
this.dgvGrid.Columns["ListPrice"].DefaultCellStyle = styColumna;
```

Fuente 5.

El resultado queda palpable en la figura 3.

ProductKey	ProductNAME	ListPrice	EndDate
212	Casco deportivo:	33,6442	30/06/2002
213	Casco deportivo:	33,6442	30/06/2003
214	Casco deportivo:	34,9900	Sin asignar
215	Casco deportivo:	33,6442	30/06/2002
216	Casco deportivo:	33,6442	30/06/2003
217	Casco deportivo:	34,9900	SIN VALOR
218	Calcetines para	9,5000	30/06/2002
219	Calcetines para	9,5000	30/06/2002
220	Casco deportivo:	33,6442	30/06/2002

Figura 3. Estilos aplicados a diversas regiones del control DataGridView.

El mecanismo que controla el orden de asignación de los estilos a las regiones de la cuadrícula se encuentra gobernado por el propio control. De esta forma, independientemente del orden que en nuestro código utilicemos para establecer los estilos, estos serán aplicados según el orden mostrado por la tabla 3.

Región del control	Propiedad
Conjunto general de todas las celdas de datos excepto las cabeceras de columna y filas	DefaultCellStyle
Columna (se asignan de forma independiente)	DataGridView.Columns[IndiceColumna].DefaultCellStyle
Filas	RowsDefaultCellStyle
Filas alternas	AlternatingRowsDefaultCellStyle

Tabla 3. Orden de asignación de estilos a las regiones del DataGridView.

### Un estilo y un formato para cada tipo de dato

Gracias a la propiedad DataGridViewColumn.ValueType, podemos conocer el tipo de dato que internamente manipula una columna determinada del control, lo cual resulta de gran ayuda si nuestra intención se basa en utilizar un estilo distinto para las columnas del DataGridView, en función del tipo de dato con el que deban trabajar.

Este es el objetivo que perseguimos en el fuente 6, donde en primer lugar creamos un estilo para cada tipo de dato. Recorriendo a continuación la colección Columns del control, comprobaremos el tipo de dato de cada columna y asignaremos el estilo según sea dicho tipo.

Dado que estamos manejando datos de fecha y numéricos, los cuales se prestan amablemente a ser formateados, en esta ocasión haremos uso de la propiedad

## DataGridView y Estilos. Presentaciones de datos visualmente atractivas

Format, existente en la clase DataGridViewCellStyle, para adaptar la presentación de las columnas correspondientes a estos valores.

```
// crear un estilo para cada tipo de dato
// de las columnas del grid
DataGridViewCellStyle styTexto = new DataGridViewCellStyle();
styTexto.BackColor = Color.SeaShell;
styTexto.ForeColor = Color.DarkMagenta;
styTexto.SelectionBackColor = Color.Lime;
styTexto.SelectionForeColor = Color.MediumBlue;
styTexto.Alignment = DataGridViewContentAlignment.TopCenter;
styTexto.Font = new Font("Century Gothic", 10, FontStyle.Italic | FontStyle.Bold);
styTexto.WrapMode = DataGridViewTriState.True;
```

```
DataGridViewCellStyle styFecha = new DataGridViewCellStyle();
styFecha.BackColor = Color.LightYellow;
styFecha.ForeColor = Color.Olive;
styFecha.SelectionBackColor = Color.Lime;
styFecha.SelectionForeColor = Color.MediumBlue;
styFecha.Format = "dddd, dd \\de MMMM \\de yyyy";
styFecha.NullValue = "SIN FECHA";
```

```
DataGridViewCellStyle styInt = new DataGridViewCellStyle();
styInt.BackColor = Color.PowderBlue;
styInt.ForeColor = Color.Navy;
styInt.SelectionBackColor = Color.Lime;
styInt.SelectionForeColor = Color.MediumBlue;
styInt.Alignment = DataGridViewContentAlignment.MiddleRight;
```

```
DataGridViewCellStyle styDec = new DataGridViewCellStyle();
styDec.BackColor = Color.SandyBrown;
styDec.ForeColor = Color.DarkBlue;
styDec.SelectionBackColor = Color.Lime;
styDec.SelectionForeColor = Color.MediumBlue;
styDec.Alignment = DataGridViewContentAlignment.MiddleRight;
styDec.Format = "#,#.##0";
```

```
// recorrer la colección de columnas,
// en función del tipo de dato, asignar un estilo
foreach (DataGridViewColumn dgvcColumna in this.dgvGrid.Columns)
{
    if (dgvcColumna.ValueType == typeof(string))
    {
        dgvcColumna.DefaultCellStyle = styTexto;
    }

    if (dgvcColumna.ValueType == typeof(DateTime))
```



```
{
    dgvcColumna.DefaultCellStyle = styFecha;

    // ajustar automáticamente el tamaño de la columna de fecha
    dgvcColumna.AutoSizeMode =
DataGridViewAutoSizeColumnMode.DisplayedCells;
}

if (dgvcColumna.ValueType == typeof(int))
{
    dgvcColumna.DefaultCellStyle = styInt;
}

if (dgvcColumna.ValueType == typeof(decimal))
{
    dgvcColumna.DefaultCellStyle = styDec;
}
}

// ajustar automáticamente el tamaño de las filas
this.dgvGrid.AutoSizeRowsMode =
DataGridViewAutoSizeRowsMode.AllCellsExceptHeaders;
```

Fuente 6.

Observe el lector que al manipular la columna correspondiente al valor de fecha, establecemos el ajuste de tamaño de la misma mediante la propiedad `AutoSizeMode`, para que su anchura se adapte al valor que debe mostrar y no quede este truncado. Por otro lado, al final de este fuente también establecemos el ajuste de tamaño, en este caso para la altura de las filas del control, empleando su propiedad `AutoSizeRowsMode`. Con ello conseguimos que el texto del campo `SpanishProductName` se muestre en su totalidad, en aquellas celdas cuyo contenido sea más extenso.

La figura 4 muestra este ejemplo en ejecución, donde podemos apreciar las diferentes configuraciones de colores, formato y alineación, dependiendo del tipo de dato de la columna. La excepción la representa la combinación de colores para las columnas en estado seleccionado, donde hemos utilizado los mismos valores en las propiedades `SelectionBackColor` y `SelectionForeColor` para todos los estilos.



ProductKey	SpanishProductNar	ListPrice	EndDate	ProductLine
285	Cuadro de carretera GB: negro, 52	297,63	domingo, 30 de junio de 2002	R
286	Cuadro de carretera GB: negro, 52	306,56	lunes, 30 de junio de 2003	R
287	Cuadro de carretera GB: negro, 52	337,22	SIN FECHA	R
288		1.204,32	domingo, 30 de junio de 2002	M
289		1.240,45	lunes, 30 de junio de 2003	M
290		1.364,50	SIN FECHA	M
291		1.364,50	domingo, 30 de junio de 2002	M
292		1.364,50	domingo, 30 de junio de 2002	M

Figura 4. Aplicando diferentes estilos según el tipo de dato de cada columna.

### Modificación dinámica de estilos durante la navegación

Cuando manipulamos un control DataGridView, existen situaciones en las que resultaría deseable que al efectuar una interacción determinada sobre el control, este respondiera mediante algún cambio en su estado visual.

Pongamos como ejemplo un requerimiento bastante típico: cuando el usuario desplaza el cursor del ratón sobre las celdas del control, debemos cambiar las propiedades de presentación de la celda sobre la que se encuentre el cursor, o bien la fila entera.

Tal funcionalidad la lograremos utilizando los eventos CellMouseEnter y CellMouseLeave del control DataGridView, que como indican sus respectivos nombres, se producen cuando el cursor del ratón entra y sale de la superficie ocupada por una celda.

Estos eventos reciben como parámetro un objeto DataGridViewCellEventArgs, cuyas propiedades ColumnIndex y RowIndex, utilizadas en combinación con la colección DataGridView.Cells, nos servirán para averiguar la celda sobre la que deberemos aplicar el cambio de estilo.

El fuente 7 muestra el caso descrito, donde primeramente declaramos una variable con ámbito para la clase del formulario, que contendrá el nombre del campo/columna sobre el que vamos a actuar, ListPrice. A continuación, en el evento Load del formulario, creamos un estilo para el control y deshabilitamos la visualización de las etiquetas flotantes en las celdas, para poder apreciar mejor el resultado en ejecución.

Seguidamente escribimos los métodos manipuladores de los eventos antes mencionados. En CellMouseEnter, en lugar de instanciar un objeto DataGridViewCellStyle, manipulamos directamente las propiedades del estilo ya

existente en la celda para obtener el resultado deseado; mientras que CellMouseLeave es mucho más simple: asignamos a la celda el estilo predeterminado –original- del control.

```
public partial class Form1 : Form
{
    // asignar el nombre de columna
    string sNombreColumna = "ListPrice";

    private void Form1_Load(object sender, EventArgs e)
    {
        //....
        // desactivar la visualización de tooltips en las celdas
        this.dgvGrid.ShowCellToolTips = false;

        // crear y aplicar un estilo al grid
        DataGridViewCellStyle styTexto = new DataGridViewCellStyle();
        styTexto.BackColor = Color.Wheat;
        styTexto.ForeColor = Color.DarkMagenta;
        this.dgvGrid.DefaultCellStyle = styTexto;
    }

    // al entrar el cursor del ratón en la celda
    private void dgvGrid_CellMouseEnter(object sender, DataGridViewCellEventArgs e)
    {
        // si el cursor está en una celda de datos
        if (e.RowIndex >= 0)
        {
            // comprobar el nombre de la columna
            if (this.dgvGrid.Columns[e.ColumnIndex].Name == sNombreColumna)
            {
                // modificar propiedades de estilo de la celda en la que estamos situados
                this.dgvGrid.Rows[e.RowIndex].Cells[sNombreColumna].Style.BackColor =
                Color.MediumPurple;
                this.dgvGrid.Rows[e.RowIndex].Cells[sNombreColumna].Style.ForeColor =
                Color.Lavender;
                this.dgvGrid.Rows[e.RowIndex].Cells[sNombreColumna].Style.Format =
                "#,#.##0";
                this.dgvGrid.Rows[e.RowIndex].Cells[sNombreColumna].Style.Font = new
                Font("Sylfaen", 14, FontStyle.Bold);
                this.dgvGrid.Rows[e.RowIndex].Cells[sNombreColumna].Style.Alignment =
                DataGridViewContentAlignment.MiddleCenter;
            }
        }
    }

    // al salir el cursor del ratón en la celda
```

## DataGridView y Estilos. Presentaciones de datos visualmente atractivas

```
private void dgvGrid_CellMouseLeave(object sender, DataGridViewCellEventArgs e)
{
    // si el cursor está en una celda de datos
    if (e.RowIndex >= 0)
    {
        // devolver el estilo de la celda a su valor original
        this.dgvGrid.Rows[e.RowIndex].Cells[sNombreColumna].Style =
this.dgvGrid.DefaultCellStyle;
    }
}
}
```

Fuente 7.

La figura 5 muestra este ejemplo en ejecución, donde podemos apreciar la celda sobre la que está posicionado actualmente el cursor.



ProductKey	SpanishProductNar	ListPrice	EndDate
235	Jersey con logoti...	48,0673	30/06/
236	Jersey con logoti...	48,0673	30/06/
237	Jersey con logoti...	49,9900	
238	Cuadro de carret...	1.263,46	30/06/
239	Cuadro de carret...	1301,3636	30/06/
240	Cuadro de carret...	1431,5000	

Figura 5. Modificación dinámica de estilo en celda al pasar el cursor del ratón.

Si queremos hacer extensiva esta funcionalidad a la totalidad de la fila, recorreremos todas sus celdas empleando la colección Cells del objeto DataGridViewRow actual. Nótese que para las celdas en las que debemos aplicar formato hacemos uso de la propiedad DataGridViewCell.OwningColumn, que nos devuelve el objeto DataGridViewColumn, mediante el que averiguamos la columna sobre la que estamos iterando, aplicando el formato cuando sea oportuno. Todo ello lo podemos ver en el fuente 8.

```
private void dgvGrid_CellMouseEnter(object sender, DataGridViewCellEventArgs e)
{
    if (e.RowIndex >= 0)
    {
        foreach (DataGridViewCell oCelda in this.dgvGrid.Rows[e.RowIndex].Cells)
        {
            oCelda.Style.BackColor = Color.MediumPurple;
            oCelda.Style.ForeColor = Color.Lavender;
            oCelda.Style.Font = new Font("Sylfaen", 14, FontStyle.Bold);
        }
    }
}
```

```
oCelda.Style.Alignment = DataGridViewContentAlignment.MiddleCenter;

switch (oCelda.OwningColumn.Name)
{
    case "ListPrice":
        oCelda.Style.Format = "#,##.#0";
        break;

    case "EndDate":
        oCelda.Style.Format = "ddd-MMMM-yy";
        break;
}
}
}
}

private void dgvGrid_CellMouseLeave(object sender, DataGridViewCellEventArgs e)
{
    if (e.RowIndex >= 0)
    {
        foreach (DataGridViewCell oCelda in this.dgvGrid.Rows[e.RowIndex].Cells)
        {
            oCelda.Style = this.dgvGrid.DefaultCellStyle;
        }
    }
}
```

Fuente 8.

La figura 6 muestra este efecto aplicado a la fila al completo.



The screenshot shows a window titled "Modificación dinámica de estilo" containing a DataGridView table. The table has six columns: ProductKey, SpanishProductNar, ListPrice, EndDate, and ProductLine. Row 251 is highlighted in a dark purple color, while other rows have a light yellow background. The data for row 251 is: ProductKey: 251, SpanishProductNar: Cuadro..., ListPrice: 1.301,36, EndDate: lun-junio-03, ProductLine: R.

ProductKey	SpanishProductNar	ListPrice	EndDate	ProductLine
248	Cuadro de carret...	1301,3636	30/06/2003	R
249	Cuadro de carret...	1431,5000		R
250	Cuadro de carret...	1263,4598	30/06/2002	R
251	Cuadro...	1.301,36	lun-junio-03	R
252	Cuadro de carret...	1431,5000		R
253	Cuadro de carret...	297,6346	30/06/2002	R
254	Cuadro de carret...	306,5636	30/06/2003	R
255	Cuadro de carret...	337,2200		R

Figura 6. Modificación dinámica de estilo en fila al pasar el cursor del ratón.

**Articulo.Style.Estado = Terminando;**

## DataGridView y Estilos. Presentaciones de datos visualmente atractivas

A lo largo de este artículo, hemos abordado los estilos del DataGridView, como medio para realzar los datos expuestos mediante este control, de manera que la información exhibida resulte al usuario útil a la par que atractiva. Esperamos que todo este conjunto de características sirvan para allanar el camino a los lectores que necesiten desarrollar aplicaciones utilizando este control.