Implement Advanced DataGrid Styles

Dig into some manual coding techniques for adding special effects to data displays.

by Fabio Claudio Ferracchiati and Juval Löwy

Technology Toolbox

- ☑ VB.NET
- ☑ C#
- ☐ SQL Server 2000
- ☐ ASP.NET
- ☐ XML
- ☐ VB6

Go Online!

Use these Locator+ codes at **www.visualstudiomagazine.com** to go directly to these related resources.

Download
VS0310QA Download the code for this article, which includes a VS.NET 2003 solution that creates a custom DataGrid, and an implementation of a singleton application that uses the SingletonApp object.

Discuss
VS0310QA_D Discuss this article in the VB.NET forum.

Read More
VS0310QA_T Read this article online.

VS0310GS_T Getting Started, "Manage Data Easily," by Fabio Claudio Ferracchiati

VS0302FB_T "Leverage VB.NET's Object-Oriented Features" by Francesco Balena

VS0302QA_T Q&A, "Call WinForms on Multiple Threads," by Juval Löwy and Karl E. Peterson

Q: Implement Advanced DataGrid Styles

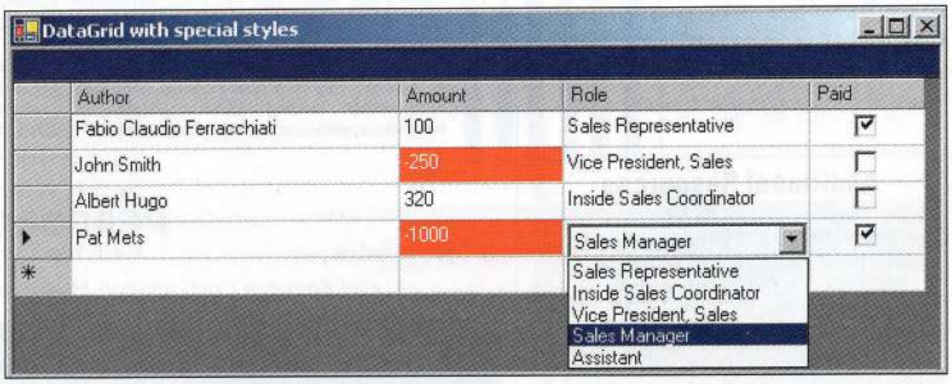
I'm developing a .NET WinForms application that uses a DataGrid to display data the app retrieves from a table in a remote database. The table has a foreign-key column, and I'd like to use a ComboBox control in the related column of the DataGrid to display data retrieved from the second table. I also want to change a cell's color when its value is less than zero. I found some information on DataGrid styles but wasn't able to implement them in my application.

A:

The DataGrid WinForms component is powerful and flexible, but implementing effects that go beyond the component's standard behavior can be difficult. You can't implement the advanced effects you want by simply setting DataGrid properties or calling DataGrid methods, and you can't delegate DataGrid style settings to the VS.NET wizard. You must specify each column manually by creating a new object from the related DataGrid column class (download the sample code from the *VSM* Web site; see the Go Online box for details). For example, if the DataGrid column must contain a string, you create a DataGridTextBoxColumn object that specifies the data-source column to bind from. Then, you add the object to the DataGridTableStyle class's GridColumnStyles collection. Finally, you must specify the reference to the DataGridTableStyle object in the DataGrid object's TableStyles collection:

```
Dim styles As New DataGridTableStyle
styles.MappingName = "Authors"
Dim colAuthor As New _
   DataGridTextBoxColumn
colAuthor.MappingName = "Author"
colAuthor.HeaderText = "Author"
colAuthor.Width = 200
   styles.GridColumnStyles.Add(colAuthor)
Dim dg As new DataGrid
dg.TableStyles.Add(styles)
```

Use the DataGridTableStyle class's MappingName property to specify the data-source table to bind from, and the DataGrid column class's MappingName property to specify the data


DataGrid with special styles

	Author	Amount	Role	Paid
	Fabio Claudio Ferracchiati	100	Sales Representative	✓
	John Smith	-250	Vice President, Sales	☐
	Albert Hugo	320	Inside Sales Coordinator	☐
▶	Pat Mets	-1000	Sales Manager	✓
*				

Figure 1 Add Special DataGrid Styles. You can use class inheritance to customize a DataGrid's standard column behavior and implement special styles. This DataGrid includes a combo box column and a cell that changes its color when its value is less than zero.

VB.NET • Create a Combo Box Column

```
Public Class DataGridComboBoxColumn
   Inherits DataGridTextBoxColumn
   Public MyCombo As ComboBox
   Public Sub New()
      MyBase.New()

      MyCombo = New ComboBox
      AddHandler MyCombo.Leave, New EventHandler( _
         AddressOf LeaveComboBox)
   End Sub
   Protected Overloads Overrides Sub Edit(ByVal _
      source As CurrencyManager, ByVal rowNum _
      As Integer, ByVal bounds As Rectangle, _
      ByVal readOnly1 As Boolean, ByVal _
      instantText As String, ByVal cellIsVisible As Boolean)
      MyBase.Edit(source, rowNum, bounds, _
         readOnly1, instantText, cellIsVisible)
      MyCombo.Parent = Me.TextBox.Parent
      MyCombo.Location = Me.TextBox.Location
      MyCombo.Size = New Size(Me.TextBox.Size.Width, _
         MyCombo.Size.Height)
      MyCombo.Text = Me.TextBox.Text
      Me.TextBox.Visible = False
      MyCombo.Visible = True
      MyCombo.BringToFront()
      MyCombo.Focus()
   End Sub
   Private Sub LeaveComboBox(ByVal sender _
      As Object, ByVal e As EventArgs)
      MyCombo.Hide()
   End Sub
End Class
```

Listing 1 You can make a DataGrid column show and hide a combo box, instead of the classic textbox, by deriving from the DataGridTextBoxColumn class to manage events such as Edit and Leave.

source's column name to bind from.

The DataGrid component doesn't offer a column class for displaying a combo box containing data you retrieve from a different table. However, .NET's object-oriented features let you derive new classes from a generic class and add features. You can use the DataGridTextBoxColumn class as the base class, then add combo box support and color changes.

Follow these steps to implement a combo box column (see Listing 1). First, derive a new class from the DataGridTextBoxColumn class. Then, add a public ComboBox component for specifying data from a different table. Next, add a class constructor where you initialize objects and specify event handlers. Now, manage the Edit event, which is raised when the user sets the focus to the cell containing the combo box (by clicking on the cell or using the Tab key to go through the cells). In the body of the Edit event handler, give the combo box the same dimensions and text as the selected cell. Finally, hide the combo box in the body of the Leave event, which is raised when the user leaves the cell.

You can add a new DataGridComboBoxColumn object to the DataTableStyles object now, specifying the combo box's width and the data source's mapping column name, and adding combo box items you retrieve from the related table (I've added some values manually for code readability):

```
Dim colCombo As DataGridComboBoxColumn
colCombo = New DataGridComboBoxColumn
colCombo.MappingName = "Role"
colCombo.HeaderText = "Role"
colCombo.Width = 150
colCombo.MyCombo.Items.Clear()
colCombo.MyCombo.Items.Add(...)
colCombo.MyCombo.DropDownStyle = _
   ComboBoxStyle.DropDownList
styles.GridColumnStyles.Add(colCombo)
```

The DataGrid still has two issues at this point. When you navigate through the cells with the Tab key, you can't leave the focus over the cell that contains the combo box. Also, if you change a value in the combo box and leave the cell, the new value replaces the original value but doesn't update the corresponding value in the data source.

You can avoid the first issue by using the overridable WndProc method's Message parameter, which helps you trap Windows messages:

```
Public Class DataGridComboBox
  Inherits ComboBox
  Private WM_KEYUP As Integer = &H101
  Protected Overrides Sub WndProc(ByRef m _
     As System.Windows.Forms.Message)
     If m.Msg = WM_KEYUP Then
        Return
     End If
     MyBase.WndProc(m)
  End Sub
End Class
```

The preceding code derives a DataGridComboBox class from the ComboBox class and uses it to trap and ignore the WM_KEYUP Windows message. The Return instruction breaks the message that the .NET Framework sends to the base class. This enables the Tab key to maintain focus over the cell containing the combo box.

The second issue requires some extra work, because you must track when the user starts editing the value in the combo box, then commit the changes to the data source (see Listing 2). Add the Boolean m_isEditing variable to track users' changes to the combo box. Then, add a new event handler that responds to the SelectionChangeCommitted event, which is raised when the user changes a value in the combo box and the change is committed. Mark the m_isEditing variable in the body of the OnSelectionChangeCommitted event handler function as True, because the user has edited the combo box and you must commit the changes to the data source as well. The DataGridColumnStyles class's overridable Commit method lets you commit changes to the data source. You check the m_isEditing variable in this method's body to commit the changes only when the user has altered the combo box.

You can use the same base classes to create a DataGrid column whose background and foreground colors change depending on the value in a cell (see Figure 1). The DataGridTextBoxColumn class's overridable Paint method lets you change this aspect of a DataGrid column. One of the Paint method's three prototypes offers two Brush objects you can use to color a cell's background and foreground (download Listing 3). You use this method's GetColumnValueAtRow method to retrieve the cell's value. GetColumnValueAtRow is a generic Object type, because the method doesn't know if the cell contains an Integer, a String, and so on. The code casts the Object type to an Integer type, then checks for its value. When the value is less than zero, the background color changes to red and the foreground color to white. Finally, you call the base class's Paint method to provide these new values. *—F.C.F.*

Q: Build a Singleton WinForms App

I want to allow only a single instance of my application to run at any point in time (like Outlook). .NET should ignore the user's request to launch the application if it's already running. How can I achieve this?

A:

What you describe is called a *singleton application.* Rich-client apps often need to be singletons. Unfortunately, the Application class doesn't provide singleton support. I'll show you how to use my own version of a singleton application object—called SingletonApp—which I define in the WinFormsEx assembly (download the sample code). You can use SingletonApp the same way you use a normal, multi-instance Application class, except it ensures the application is a singleton:

```
using WinFormsEx;
public class MyForm : Form
{
   static void Main()
   {
      SingletonApp.Run(new MyForm());
   }
   /* Rest of MyForm implementation */
}
```

The preceding code displays the MyForm form if no other instance of it is running. Otherwise, the application exits. The SingletonApp class provides the same Run methods as the Application class.

You need some sort of cross-process communications mechanism to implement a singleton, because if the user launches a new instance of the application, it's in a new process, yet it must detect if another process is already running. You can use the Mutex synchronization object for this purpose:

```
public sealed class Mutex : WaitHandle
{
   public Mutex();
   public Mutex(bool initiallyOwned);
   public Mutex(bool initiallyOwned,
     string name);
   public Mutex(bool initiallyOwned,
      string name,
      out bool createdNew);
   public void ReleaseMutex();
}
```

You normally use the Mutex to ensure mutual exclusion of threads from a resource or code section. Only one thread at a time can own the Mutex. A thread owns the Mutex by calling one of the wait methods of the Mutex's base class—WaitHandle. If the Mutex is unowned, then .NET doesn't block the thread, and the thread gets ownership. If another thread owns the Mutex, .NET blocks the new thread until the Mutex is released, or until a specific timeout has elapsed. You release the Mutex by calling the ReleaseMutex() method, which sets the Mutex state to unowned, allowing other threads access to it. Mutex's default constructor creates it in the unowned state. The constructor's other parameterized versions accept the initiallyOwned flag, which lets you set the Mutex's initial state explicitly.

The parameterized constructors also allow you to specify a Mutex name in the name parameter. The Mutex name is any identifying string, such as "My Mutex." By default, a Mutex has no name. However, if you do provide a name, then any thread on the machine—including threads in other processes—can try to access this Mutex. When

Continued on page 53.

VB.NET • Commit User Changes

```
Public Class DataGridComboBoxColumn
  Inherits DataGridTextBoxColumn

  Public MyCombo As DataGridComboBox
  Private m_isEditing As Boolean
  Public Sub New()
    MyBase.New()
    MyCombo = New DataGridComboBox
    m_isEditing = False
    AddHandler MyCombo.Leave, New _
      EventHandler(AddressOf LeaveComboBox)
    AddHandler MyCombo.SelectionChangeCommitted, New _
      EventHandler(AddressOf _
      OnSelectionChangeCommitted)
  End Sub
  Protected Overloads Overrides Sub Edit(ByVal source As _
    CurrencyManager, ByVal rowNum As Integer, ByVal _
    bounds As Rectangle, ByVal readOnly1 As Boolean, _
    ByVal instantText As String, ByVal cellIsVisible As _
    Boolean)
    MyBase.Edit(source, rowNum, bounds, _
      readOnly1, instantText, cellIsVisible)
    MyCombo.Parent = Me.TextBox.Parent
    MyCombo.Location = Me.TextBox.Location
    MyCombo.Size = New Size(Me.TextBox.Size.Width, _
      MyCombo.Size.Height)
    MyCombo.Text = Me.TextBox.Text
    Me.TextBox.Visible = False
    MyCombo.Visible = True
    MyCombo.BringToFront()
    MyCombo.Focus()
  End Sub
  Private Sub LeaveComboBox(ByVal sender As _
    Object, ByVal e As EventArgs)
    MyCombo.Hide()
  End Sub
  Protected Overloads Overrides Function Commit(ByVal _
    dataSource As CurrencyManager, ByVal rowNum As _
    Integer) As Boolean
    If m_isEditing Then
      m_isEditing = False
      SetColumnValueAtRow(dataSource, rowNum, _
        MyCombo.Text)
    End If
    Return True
  End Function
  Private Sub OnSelectionChangeCommitted(ByVal sender As _
    Object, ByVal e As EventArgs)
    m_isEditing = True
  MyBase.ColumnStartedEditing(sender)
  End Sub
End Class
```

Listing 2 This DataGrid combo box column implementation commits changes the user enters through the combo box to the data source.

Continued from page 48.

you specify a name, the OS checks whether another application or a thread in the same application has created a Mutex with that name already, and if so, gives the creating thread a local object that represents the global Mutex. If you try to create a named Mutex, you should pass false for initiallyOwned, because otherwise .NET won't block your thread, even if another thread in a different process owns the Mutex already.

Now, here is the trick for using a Mutex for a singleton WinForms app. The SingletonApp class uses the IsFirstInstance helper method:

```
static bool IsFirstInstance()
{
   m_Mutex = new
      Mutex(false,"SingletonApp Mutex");
   bool owned = false;
   owned = m_Mutex.WaitOne
      (TimeSpan.Zero,false);
   return owned ;
}
```

IsFirstInstance creates a named Mutex, then waits on it for a timeout of zero. If this is the only instance of the app running, the Mutex is unowned, and the wait operation returns immediately, indicating ownership. If another instance is running, then the wait returns false. The SingletonApp encapsulates a normal Application object and delegates the actual implementation of Run to it, if it's the only instance running. Otherwise, it simply ignores the request to run a new form.

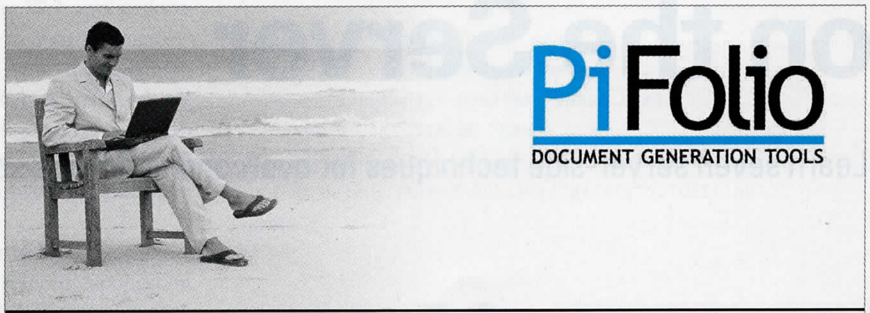
One additional problem remains: How does the single instance release the Mutex when the application shuts down? The Application class's ApplicationExit event lets you know the application is about to shut down. SingletonApp subscribes to this event and releases and closes the Mutex in its handling of the exit event (download Listing 4). *—J.L.*

Fabio Claudio Ferracchiati has 10 years of experience using Microsoft technologies. He's been focusing attention recently on the new .NET Framework architecture and languages and has written books for Wrox Press about this technology. He works in Rome for the CPI Progetti SpA company (www.cpiprogetti.it). Contact him at ferracchiati@rocketmail.com.

Juval Löwy is a software architect and the principal of IDesign, a consulting and training company focused on .NET design and .NET migration. Juval is Microsoft's regional director for the Silicon Valley, working with Microsoft on helping the industry adopt .NET. His latest book is *Programming .NET Components* (O'Reilly & Associates). Juval speaks frequently at software-development conferences. Contact him at www.idesign.net.

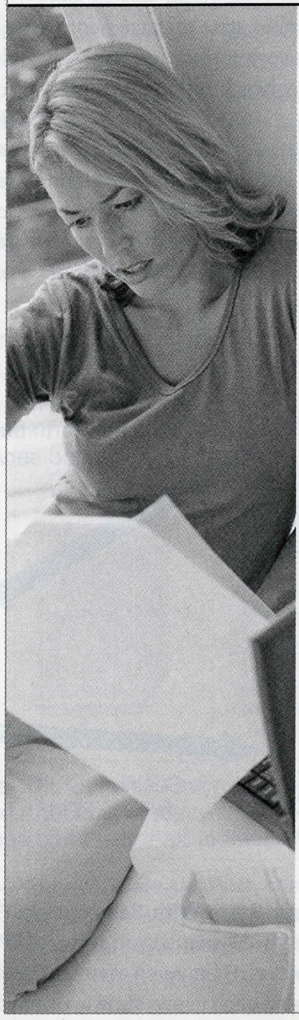
Additional Resources

- *Programming Microsoft Visual Basic .NET (Core Reference)* by Francesco Balena [Microsoft Press, 2002, ISBN: 0735613753]
- *Programming .NET Components* by Juval Löwy [O'Reilly & Associates, 2003, ISBN: 0596003471]



PiFolio
DOCUMENT GENERATION TOOLS

Now as easy as a walk to the beach:
Report and layout any data in MS Word.



Give your clients the best of two worlds: PiFolio, a simple yet powerful tool that lets anybody report and layout any data from any source in MS Word.

- PiFolio closes the gap between your programming platform and MS Word. Just integrate it in any specific application for a one time fee. No further royalties to be paid.
- Generate forms, reports and more in different styles and formats – without tedious hours of source code programming. You will give your client exactly what he needs. And all documents can be easily restructured later on – whenever there is a need for it.
- PiFolio enhances the value of your work as your product will be MS Office™ integrated: A simple mouse click is enough to connect your application with the versatility of MS Word.

Ease of use. Ease of mind. Fast results: Discover the power of PiFolio now.

Visit us at

www.pifolio.com