Visual Basic and Visual Basic .NET for Scientists and Engineers

CHRISTOPHER FRENZ

Apress™

Visual Basic and Visual Basic .NET for Scientists and Engineers

ISBN (pbk): 1-893115-55-0

Printed and bound in the United States of America 12345678910

Technical Reviewer: MN, MCSD

Editorial Directors: Dan Appleman, Peter Blackburn, Gary Cornell, Jason Gilmore, Karen Watterson

Managing Editor: Grace Wong

Project Managers: Alexa Stuart, Erin Mulligan

Copy Editors: Jennifer Lind, Nicole LeClerc, Ami Knox

Production Editor: Kari Brooks

Compositor: Impressions Book and Journal Services, Inc.

Indexer: Carol Burbo

Cover Designer: Tom Debolski

Marketing Manager: Stephanie Rodriquez

Distributed to the book trade in the United States by Springer-Verlag New York, Inc.,175 Fifth Avenue, New York, NY, 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States, phone 1-800-SPRINGER, email orders@springer-ny.com, or visit http://www.springer-ny.com.

Outside the United States, fax +49 6221 345229, email orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 901 Grayson Street, Suite 204, Berkeley, CA 94710.

Phone 510-549-5938, fax: 510-549-5939, email info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Downloads section.

CHAPTER 9

Programming Your Own Spreadsheets

In this chapter, you'll learn how to code the spreadsheet-type interfaces that are common to so many scientific applications. These interfaces are extremely useful when dealing with matrix math or other scenarios that require a large amount of data. To develop such an advanced interface, this chapter introduces a new control along with some user events you haven't yet encountered. The examples require you to use almost everything you've seen so far, so make sure that you have the basics covered.

The MSFlexGrid Control

The MSFlexGrid control, when it's displayed on a form, appears like any spreadsheet grid. It consists of a number of intersecting rows and columns, and the points where the rows and columns intersect are known as *cells*. If you add such a control to your form and click Run, you'll see that it behaves almost like a spreadsheet program's worksheet. For example, you can move between cells with the arrow keys and mouse. The only real difference between this control and an actual spreadsheet is that you can't enter (type) data directly into a grid cell. Thus, this chapter will concentrate on coding a way to allow direct user data entry. But first, take some time to examine the important properties and events associated with this control. The MSFlexGrid control isn't a part of the standard VB toolbox. In VB 6.0, you must go to the Project menu and select the Components option. You then select the MSFlexGrid control from the Components list. In VB .NET you have to go to the Tools menu and select the Customize Toolbox option. You next select the MSFlexGrid control from the COM controls list. The control icon then appears in the toolbox, where you can drag and drop it onto a form as with any other control.

MSFlexGrid Properties

The following sections outline the properties for the MSFlexGrid control.

The AllowBigSelection Property

This property determines whether or not a whole row or column will be selected if the user clicks a row or column header. If it's set to the default of True, such whole row/column selections are allowed. If the property is set to False, only single cell selections are allowed.

The AllowUserResizing Property

This property works pretty much as you'd expect from its name. Its settings determine if neither rows nor columns can be resized (flexResizeNone) by the user, if just columns can be resized (flexResizeColumns), if just rows can be resized (flexResizeRows), or if both rows and columns can be resized (flexResizeBoth).

The Cols and Rows Properties

These properties are integer values, which determine the grid's number of columns and rows. The default for each property is 2, but you can easily adjust this through code (i.e., `MSFlexGrid1.Cols = Number of Columns`) or through the Properties window.

The Col and Row Properties

Be careful not to confuse these two properties with the Cols and Rows properties because they have a different function. These properties don't set the number of rows and columns, but rather store the numbers that represent the row and column containing the active cell. Also keep in mind that this property starts counting columns (left to right) and rows (top to bottom) from 0 and not 1. Thus, if the Cols property is set to 5, you'll find that Col property can equal any integer between 0 and 4, depending on which column contains the active cell. Please note that the Col and Row properties are available only at runtime.

The ColWidth and RowHeight Properties

You only can manipulate these properties at runtime. Nevertheless, they can be useful because they let you set a specified row's height as well as a specified column's width. You'll often find yourself needing to do this if default row or column dimensions are too large for the data to be displayed. The height and width is measured in twips.

The ColAlignment Property

This property's value determines whether or not the data displayed in a particular column is centered or justified. Setting it equal to 0 results in left justification while setting it to 1 right justifies the data. If the property is set equal to 2, the information displayed will be centered. This property isn't available at design time (except indirectly through the FormatString property).

The CellLeft, CellWidth, CellTop, and CellHeight Properties

Just like with the control-level Left, Width, Top, and Height properties (now referred to as Location.X, Size.Width, Location.Y, and Size.Height in VB .NET), these properties return the size and position of the current cell in twips. The only difference is that the cell properties refer to the frame that encloses the grid and not the form. These properties aren't available at design time and are read-only at runtime.

The FixedCols and FixedRows Properties

These properties control the grid's number of fixed rows and columns. The fixed rows and columns always appear at the grid's top and left sides (non-scrollable) and in gray instead of white. These types of cells generally provide some type of header for the rows and columns.

The FixedAlignment Property

This property works exactly like the ColAlignment property. The only difference is that it controls the alignment in fixed cells and not normal cells.

The GridLines Property

This property controls whether or not grid lines are within your grid control. The default displays the grid lines, which generally makes it much easier for the end user to distinguish the different cells.

The Sort Property

This property allows you to sort the contents of selected columns according to selected criteria. For example, the user could choose to sort in ascending or descending orders, or not sort the data at all. This property isn't available at design time and is write-only at runtime. When set at the code level, the sorting takes place immediately after the line of code executes.

The Text Property

You're very familiar with the Text property by now, but keep in mind that property stores only the text of the active cell, and not the text in any other cell. Thus, this property continually changes as you switch from cell to cell. Reading the Text property returns the current cell's contents as defined by the Row and Col properties. Writing to the Text property sets the contents of the current cell or selection (a range of cells) depending on the FillStyle property.

MSFlexGrid Events

MSFlexGrid controls respond to many of the events you've already seen, such as the Click and DoubleClick events. However, several useful events are unique to this control, and are covered in this section.

The EnterCell Event

This event occurs when the program user clicks on or keys into a new cell that is different from the cell that is currently selected. The EnterCell event works like the GotFocus event, only it deals with single cells instead of a control.

The LeaveCell Event

This event is the opposite of the EnterCell event: it occurs when the focus is shifted away from the active cell to a new cell. Thus, this event is always triggered just prior to the EnterCell event. You can use this event to validate a cell's contents.

The RowColChange Event

This event is triggered when either the current row or column is changed, which is essentially whenever the user changes cells. It always follows both the LeaveCell and EnterCell events.

MSFlexGrid Methods

The two methods discussed in this section are important to MSFlexGrid controls. These methods allow you to insert or delete rows at specific locations within the grid. However, because these methods involve the syntax of AddItem and RemoveItem, don't confuse them with the ones used in conjunction with list and combo boxes.

The RemoveItem Method

You'll begin with the RemoveItem method because its syntax is slightly simpler than the AddItem method. To invoke the RemoveItem method, you only need the control's name and the row number you want to remove. Remember that the first row begins with zero and not one, however. For example, the code

```
MSFlexGrid1.RemoveItem(5)
```

would remove the sixth row and not the fifth row in the MSFlexGrid named MSFlexGrid1. In VB .NET the default name for MSFlexGrid is AxMSFlexGrid1.

The AddItem Method

This method works like the RemoveItem method; however, you gain the additional ability to specify the contents of cells within the row you're going to add. For example, say that you want to add the values contained in String1 and String2 to cells 1 and 2 of the new row. In VB 6.0, you'd accomplish this with the following code:

```
Dim Strings As String
Strings =  "String1"  & vbTab & "String2"
MSFlexGrid1.AddItem  Strings, 5
```

In VB .NET, you'd use the code:

```
Dim Strings As String
Strings = "String1" & Microsoft.VisualBasic.ControlChars.Tab & "String2"
AxMSFlexGrid1.AddItem(Strings, 5)
```

This code adds the strings found in the Strings variable into the new row you created. If you look closely, a Tab separates the two substrings within Strings. This is how VB distinguishes the information going to each cell. In essence, the Tab is the factor that tells VB that one column ended and a new column began. The number 5 that follows the String argument is an optional argument that tells VB to add the new row in the sixth position.

Entering Text into MSFlexGrid Cells

Now that you're familiar with the different MSFlexGrids events, properties, and methods, you can apply this knowledge by coding a method of allowing typed-in text to be entered into the grid cells. As mentioned before, the grid itself doesn't support direct user data entry, so you must type the text into the text box and set the Text property of the desired cell equal to the text in the text box. The trick, however, is in integrating this text box with the active grid cells so well that it seems part of the cell itself. You also need to enable the text box to move around on the grid in response to the arrow and return keys, as if it were part of the grid itself.

This process requires many different event procedures, as well as a custom procedure, so begin the process by coding a simple application. This application calculates a compound's number of moles based on the supplied total number of moles and mole fraction. You want to perform this calculation on a listing of different compounds typed into grid cells. To begin this application, add a MSFlexGrid control named AxMSFlexGrid1 along with three text boxes named TextBox1, TextBox2, and TextBox3 to the form. TextBox1 is the text box that floats around the grid cells, while TextBox2 reads in the total number of compounds in the solution. TextBox3 reads in the solution's total number of moles.

```
After placing your controls on the form, format the MSFlexGrid by adding the
following code to the form's Load event. Private Sub Form1_Load(ByVal
eventSender As System.Object, ByVal eventArgs _
As System.EventArgs) Handles MyBase.Load
TextBox1.Visible = False
TextBox1.Font = AxMSFlexGrid1.Font
With AxMSFlexGrid1
    .Cols = 3
    .Rows = 25
```

```
    Show()
    .Col = 0
    .Row = 0
    .Text = "Name"
    .Col = 1
    .Text = "X"
    .Col = 2
    .Text = "Moles"
    .Row = 1
    .Col = 0
End With
DimTextBox()
End Sub
```

As you can see, this procedure first sets the Visible property of TextBox1 equal to False. In order for the text box to integrate seamlessly with the grid cells, you want the text box to appear only after it's properly formatted. Next, set the text box's font equal to the grid control's font. The Set keyword ensures that the fonts stay in sync even if the MSFlexGrid font is later modified. Next, establish the grid's size by setting the number of rows equal to 25 and the number of columns equal to 3. Then, make the cell present in row zero, column zero the active cell and enter the text "Name" into the cell. Remember, the grid control's Text property stores only the text present in the active cell. Then, move the active cell over and enter "X" and "Moles" in the corresponding cells. These columns store the mole fraction (X) and the calculated number of moles. Finally, move the active cell to the grid's top leftmost cell, excluding the header titles just added. Now that your grid setup is complete, call the custom procedure DimTextBox to properly position TextBox1 in this cell.

This section presents both VB 6.0 and VB .NET versions of this procedure because of their significant differences. The code for the DimTextBox routine in VB 6.0 is as follows:

```
Private Sub DimTextBox()
With AxMSFlexGrid1
            TextBox1.Left = .CellLeft + .Left
            TextBox1.Top = .CellTop + .Top
            TextBox1.Width = .CellWidth
            TextBox1.Height = .CellHeight
            TextBox1.Visible = True
            TextBox1.SetFocus
End With
End Sub
```

Looking closely at this procedure, notice that the text box's Left, Top, Height, and Width properties adjust to correspond to the active cell's equivalent positions. This places a text box of equal size as the cell directly on top of the active cell. In fact, the text box should blend in so well that the program user won't even know it's separate from the grid control. Now that the text box is so well concealed, you can make it visible and set the focus on it. This lets users type in the text box while thinking they are entering data directly into the grid control.

To have the same effect in VB .NET, you must first address some MSFlexGrid compatibility issues. As the chapter on graphics describes, the VB .NET's default graphical display unit is pixels. However, MSFlexGrid's CellLeft, CellWidth, CellTop, and CellHeight properties are all in twips. To get the text box into the correct dimensions, access the TwipsToPixelsX and TwipsToPixelsY conversion functions by selecting Add Reference from the Project menu. Scroll down the list of choices and add the Microsoft Visual Basic .NET Compatibility dll. You should see it appear under the reference heading of the Solution Explorer window. Next, import the previously unavailable Microsoft.VisualBasic.Compatibility.VB6 namespace by adding the following line of code to the declarations section:

```
Imports Microsoft.VisualBasic.Compatibility.VB6
```

> **NOTE** *This compatibility issue exists with the current MSFlexGrid 6.0 control. If a newer version comes out, it may address this incompatibility, and the conversion functions may no longer be required.*

The VB .NET code for the DimTextBox routine is as follows:

```
Private Sub DimTextBox()
        With AxMSFlexGrid1
            TextBox1.Location = New Point(TwipsToPixelsX(.CellLeft) _
+ .Location.X, TwipsToPixelsX(.CellTop) + .Location.Y)
            TextBox1.Size = New Size(TwipsToPixelsY(.CellWidth), _
 TwipsToPixelsY(.CellHeight))
            TextBox1.Visible = True
            TextBox1.Focus()
        End With
    End Sub
```

This code functions the same way as the VB 6.0 procedure, only it makes use of the newer Location X and Y properties as well as the Size property, rather than the older syntax of Left, Top, Height, and Width.

It isn't enough, however, to just call this routine for that starting cell. Instead, you must accomplish this for every cell in the grid when it becomes the active cell. The EnterCell event discussed earlier can address this by offering a way to redimension your text box every time a new cell is entered (i.e., made active). Therefore, proceed by adding the following code:

```
Private Sub AxMSFlexGrid1_EnterCell(ByVal eventSender As System.Object, ByVal _
eventArgs As System.EventArgs) Handles AxMSFlexGrid1.EnterCell
   TextBox1.Text = AxMSFlexGrid1.Text
   DimTextBox()
End Sub
```

As you can see, this code will reposition your text box by calling the DimTextBox procedure every time a new cell is entered. Before calling on this procedure, however, set the text box's text equal to the MSFlexGrid cell's text. This ensures that users can view the text when the text box is placed on top of the grid cell.

Now that you've developed a way to reposition the text box over any active cell, you must get text entered into the text box also entered into the underlying grid cell. You accomplish this by taking advantage of the text box's Change event, so let's add the following code.

```
Private Sub TextBox1_TextChanged(ByVal eventSender As System.Object, ByVal _
eventArgs As System.EventArgs) Handles TextBox1.TextChanged
       AxMSFlexGrid1.Text = TextBox1.Text
End Sub
```

This code sets the text in the active grid cell equal to any text entered into or modified in the text box. Thus, this procedure keeps the text box in sync with its underlying cell.

However, your data entry code isn't complete. You still need to activate the arrow keys so the grid's rows and columns change even when the text box has the focus. To do this, you'll use two text box events not yet discussed, the KeyDown event and the KeyPress event.

The KeyDown Event

This event is used most often with arrow keys and function keys and is invoked when a key is pressed down. Releasing the key invokes a KeyUp event. While these events work with most keys, they can't be used in conjunction with the

Enter key, the Esc key, and the Tab key. For your purposes, you want to use the KeyDown event to activate the arrow keys by utilizing the following code:

```
Private Sub TextBox1_KeyDown(ByVal eventSender As System.Object, ByVal _
eventArgs As System.Windows.Forms.KeyEventArgs) Handles TextBox1.KeyDown
    Dim KeyCode As Short = eventArgs.KeyCode
    Dim Shift As Short = eventArgs.KeyData \ &H10000
    With AxMSFlexGrid1
        Select Case KeyCode
            Case  Keys.Down
                If .Row < .Rows - 1 Then .Row = .Row + 1
            Case Keys.Up
                If .Row > 1 Then .Row = .Row - 1
            Case Keys.Right
                If .Col < .Cols - 1 Then .Col = .Col + 1
            Case Keys.Left
                If .Col > 0 Then .Col = .Col - 1
        End Select
    End With
End Sub
```

First, notice that this event involves Text1 and not the grid control. This is because after you enter a cell, the called-upon DimTextBox routine sets the focus on the text box, and only controls with the focus can respond to key events. This procedure's arguments describe the key that was pressed (KeyCode) and the state of the Shift key. Thus, when a key is pressed, this procedure checks to see if it was an arrow key by comparing the value of KeyCode to VB constants that describe the arrow keys (i.e., vbKeyDown). If an arrow key was pressed, it adjusts the active cell of the flex grid control appropriately. Before it makes an adjustment, however, the active cell's row/column number is compared to the values of the border rows and columns to ensure these values aren't exceeded. This eliminates possible program errors by preventing users from keying off the grid.

The KeyPress Event

Although enabling arrow keys is a great feature, most spreadsheet interfaces also allow you to move down columns by hitting the Enter key. As mentioned earlier, though, the KeyDown event won't work in conjunction with the Enter key. Thus, you must use the alternate KeyPress event as follows:

```
Private Sub TextBox1_KeyPress(ByVal eventSender As System.Object, ByVal _
eventArgs As System.Windows.Forms.KeyPressEventArgs) Handles _
TextBox1.KeyPress
   Dim KeyAscii As Short = Asc(eventArgs.KeyChar)
   With AxMSFlexGrid1
      If KeyAscii = Keys.Return And .Row < .Rows - 1 Then
         .Row = .Row + 1
      End If
   End With
   If KeyAscii = 0 Then
      eventArgs.Handled = True
   End If
End Sub
```

In this procedure, KeyAscii stores the value of the ANSI character that was just typed. Thus, to detect the depression of the Enter key, you examine when the value of KeyAscii equals the ANSI value that the return (Enter) key generates when pressed. When this is the case, the value of the MSFlexGrid.Row property increases by one. As with the KeyDown events, a conditional ensures that users can't key past the grid's boundaries.

The Mole Fraction Calculation

Now that the flex grid is fully set up with the ability to allow user data entry, let's move on to actual mole fraction calculation. A mole fraction (X) simply equals the number of a solution component's moles divided by the total number of moles present within the solution. Thus, if 1 mole of compound A is in a solution that contains ten moles of material, A's mole fraction is equal to 0.1.

In your program the user enters the total number of moles along with the mole fraction of each component and calculates each component's total number of moles. To perform this calculation, add a command button to the form and place the following code in its Click event:

```
Private Sub Button1_Click(ByVal eventSender As System.Object, ByVal eventArgs As _
System.EventArgs) Handles Button1.Click
Dim Num As Integer
Dim I as Integer
Dim TotMoles As Single
Dim Moles As Single
Num = CInt(TextBox2.Text)
TotMoles = CSng(TextBox3.Text)
With AxMSFlexGrid1
For I = 1 To Num
         .Col = 1
         .Row = I
         Moles = CSng(.Text) * TotMoles
         .Col = 2
         .Text = Moles
 Next I
 .Row = 1
 .Col = 0
 DimTextBox()
 End With
End Sub
```

As you can see, this event first reads in the number of components and the total number of moles. It then iterates through the column of the grid that contains the values of X and multiplies these values by the total number of moles. This resulting value, equal to the number of moles of a particular component, is then entered into the "Moles" column.

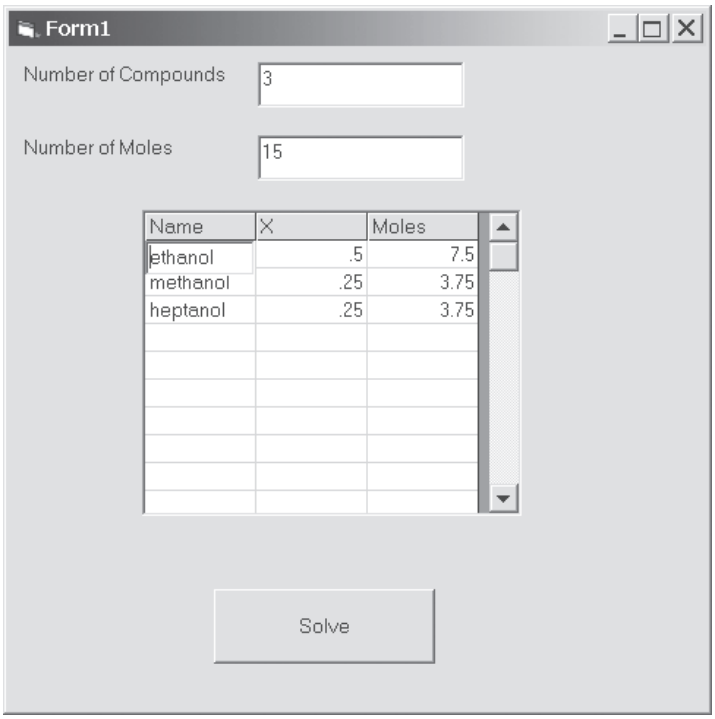


Figure 9-1. A sample output of the Mole Fraction routine

If you think carefully about this routine, one small improvement could make this routine more error-proof. A mole fraction is equal to the number of moles of a certain component divided by the total number of moles. It's impossible to have mole fractions that add up to a value greater than 1. Thus, it would be great to add up the values of X and make sure they are 1 or less before proceeding with the calculation. If they are greater than 1 an error message should alert users of their mistake. A truly well-done program takes as many precautions as possible to ensure the validity of the data generated. By this point, you should be familiar enough with the workings of the MSFlexGrid and the required VB structures. So try adding in this safeguard before moving on to the final section of this chapter.

Working with Excel Data

Microsoft Excel is a commonly used spreadsheet application, thus it is not a rarity to find that data you may want to work with in your application is stored in an Excel workbook (.xls). This is especially true since many data acquisition programs can even output Excel sheets directly. Luckily, there is a way for you to read in data from an Excel workbook and write your changes and additions back to that workbook. You can accomplish this through use of an API called Open Database Connectivity, or ODBC. What this API does is it allows you to link your program to the worksheet data, and through Structured Query Language (SQL) calls access and manipulates the data found in the worksheet. Unfortunately, at the time of this writing, ODBC .NET is not yet a completed project, and thus this book cannot delve further into this topic. ODBC .NET is due out sometime in 2002, however, and can be quite a useful tool because it can also be used for database connectivity as well as spreadsheets. To stay posted on the progress of ODBC .NET you should check with the MSDN Web site.

> **TIP** *Although not nearly as powerful or flexible, the techniques covered in Chapter 8 allow you to access your Excel data with VB code. All you need to do is save your spreadsheet as a .txt file.*

Getting Your Grids to Dynamically Respond

Now that you understand the basics of coding with MSFlexGrids, you can move past a simple mole fraction calculation and develop a more powerful routine. Think back to Chapter 5, where you looked at an iterative technique called Gauss-Siedel iteration. Engineers often use this technique to calculate 2-D temperature distributions at discrete points along a surface. To use this technique, each point's temperature is described by an energy balance equation and the equation coefficients entered into a matrix. Consecutive iterations perform on the elements of this matrix until the resultant values differ by no more than the set convergence criterion. To make this routine especially powerful, you'll get your grid and program to behave dynamically to handle any size coefficient matrix.

Begin by adding three text boxes named Text1, Text2, and Text3 and an MSFlexGrid named MSFlexGrid1 to your form. Text1 acts as the floating text box, which facilitates data entry into the grid. Text2 allows the user to input the number of nodes (points) and, hence, the number of equations to be entered into your matrix. Last, Text3 allows the application users to specify their required convergence criterion. You'll now add two command buttons named Command1

and Command2 to the form. The first command button (Command1) reformats the grid layout to enable the user to enter the correct number of matrix elements. The second command button initiates the actual iteration once all of the required data is entered.

Now that all of the required elements are on the form, add the code needed to accomplish your goals:

```
Public NumNodes As Short
Public MaxRow As Short

Private Sub Command1_Click(ByVal eventSender As System.Object, ByVal _
eventArgs As System.EventArgs) Handles Command1.Click
    Dim y As Short
    Dim x As Short
    With MSFlexGrid1
        For x = 0 To NumNodes + 3
            For y = 0 To MaxRow + 1
                .Col = x
                .Row = y
                .Text = ""
           Next y
        Next x
        .Row = 0
        NumNodes = CShort(Text2.Text)
        For x = 1 To NumNodes
            .Col = x
            .Text = "ai" & x
        Next x
        .Col = NumNodes + 1
        .Text = "Ci"
        .Col = NumNodes + 2
        .Text = "Estimations"
        .Col = 0
        For x = 1 To NumNodes
            .Row = x
            .Text = CStr(x)
        Next x
        MaxRow = .Row
        .Row = 1
        .Col = 1
    End With
End Sub
```

```
Private Sub Command2_Click(ByVal eventSender As System.Object, ByVal _
eventArgs As System.EventArgs) Handles Command2.Click
    Dim K As Short
    Dim J As Short
    Dim I As Short
    Dim z As Short
    Dim y As Short
    Dim x As Short
    Dim a(,) As Double
    Dim C() As Double
    Dim T1() As Double
    Dim T2() As Double
    Dim T3() As Double
    Dim N As Integer
    N = CShort(Text2.Text)
    ReDim a(N, N)
    ReDim C(N)
    ReDim T1(N)
    ReDim T2(N)
    ReDim T3(N)
    Dim Conv As Boolean
    Dim E As Double 'convergence criterion
    Dim Count As Short
    Dim S1 As Double
    Dim S2 As Double
    E = CDbl(Text3.Text)
    For x = 1 To N
        For y = 1 To N
            MSFlexGrid1.Row = x
            MSFlexGrid1.Col = y
            a(x, y) = CDbl(MSFlexGrid1.Text)
        Next y
    Next x
    For z = 1 To N
        MSFlexGrid1.Col = NumNodes + 1
        MSFlexGrid1.Row = z
        C(z) = CDbl(MSFlexGrid1.Text)
        MSFlexGrid1.Col = NumNodes + 2
        T2(z) = CDbl(MSFlexGrid1.Text)
    Next z
    MSFlexGrid1.Row = NumNodes + 2
    MSFlexGrid1.Col = 0
    MSFlexGrid1.Text = "Iteration"
```

```
    For I = 1 To NumNodes
        MSFlexGrid1.Col = I
        MSFlexGrid1.Text = "T" & I
    Next I
    Count = 1
    Do Until Conv = True
        Conv = True
        MSFlexGrid1.Row = MSFlexGrid1.Row + 1
        MSFlexGrid1.Col = 0
        MSFlexGrid1.Text = CStr(Count)
        For I = 1 To N
            S1 = 0
            S2 = 0
            T1(I) = (C(I) / a(I, I))
            If I - 1 > 0 Then
                For J = 1 To I - 1
                    S1 = S1 + (a(I, J) / a(I, I)) * T2(J)
                Next J
                T1(I) = T1(I) - S1
            End If
            If I + 1 <= N Then
                For K = I + 1 To N
                    S2 = S2 + (a(I, K) / a(I, I)) * T2(K)
                Next K
                    T1(I) = T1(I) - S2
            End If
            MSFlexGrid1.Col = MSFlexGrid1.Col + 1
            MSFlexGrid1.Text = (System.Math.Round(T1(I)*10000)/10000)
'The Round Function trims extra decimal places and allows easier viewing in cell
            If T1(I) - T2(I) > E Then
                Conv = False
            End If
            T3(I) = T2(I)
            T2(I) = T1(I)
        Next I
        Count = Count + 1
    Loop
    MaxRow = MSFlexGrid1.Row
End Sub
```

```
Private Sub Form1_Load(ByVal eventSender As System.Object, ByVal eventArgs As _
System.EventArgs) Handles MyBase.Load
    Text1.Visible = False
    Text1.Font = MSFlexGrid1.Font
    Dim x As Short
    With MSFlexGrid1
        .Cols = 100
        .Rows = 2000
        Show()
        .Col = 0
        .Row = 0
        .Text = "Equation #"
        For x = 1 To 4
            .Col = x
            .Text = "ai" & x
        Next x
        .Col = 5
        .Text = "Ci"
        .Col = 6
        .Text = "Estimations"
        .Col = 0
        For x = 1 To 4
            .Row = x
            .Text = CStr(x)
        Next x
            .Row = 1
            .Col = 1
    End With
    NumNodes = 4
    MaxRow = 4
    DimTextBox()
End Sub

Private Sub DimTextBox()
    With MSFlexGrid1
        Text1.Location = New Point(TwipsToPixelsX(.CellLeft) _
+ .Location.X, TwipsToPixelsX(.CellTop) + .Location.Y)
        Text1.Size = New Size(TwipsToPixelsY(.CellWidth), _
TwipsToPixelsY(.CellHeight))
        Text1.Visible = True
        Text1.Focus()
    End With
End Sub
```

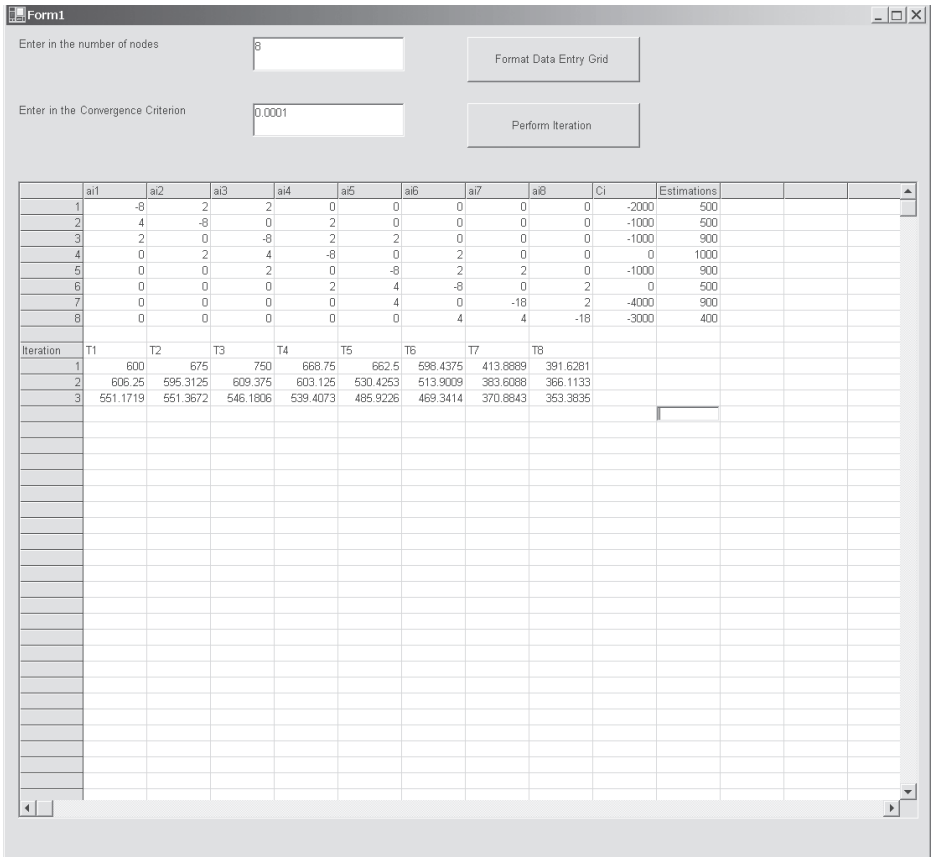
```
Private Sub MSFlexGrid1_EnterCell(ByVal eventSender As System.Object, ByVal _
eventArgs As System.EventArgs) Handles MSFlexGrid1.EnterCell
    Text1.Text = MSFlexGrid1.Text
    DimTextBox()
End Sub
Private Sub Text1_TextChanged(ByVal eventSender As System.Object, ByVal _
eventArgs As System.EventArgs) Handles Text1.TextChanged
    MSFlexGrid1.Text = Text1.Text
End Sub

Private Sub Text1_KeyDown(ByVal eventSender As System.Object, ByVal _
eventArgs As System.Windows.Forms.KeyEventArgs) Handles Text1.KeyDown
    Dim KeyCode As Short = eventArgs.KeyCode
    Dim Shift As Short = eventArgs.KeyData \ &H10000
    With MSFlexGrid1
        Select Case KeyCode
            Case  Keys.Down
                If .Row < .Rows - 1 Then .Row = .Row + 1
            Case Keys.Up
                If .Row > 1 Then .Row = .Row - 1
            Case Keys.Right
                If .Col < .Cols - 1 Then .Col = .Col + 1
            Case Keys.Left
                If .Col > 1 Then .Col = .Col - 1
        End Select
    End With
End Sub

Private Sub Text1_KeyPress(ByVal eventSender As System.Object, ByVal _
eventArgs As System.Windows.Forms.KeyPressEventArgs) Handles _
Text1.KeyPress
    Dim KeyAscii As Short = Asc(eventArgs.KeyChar)
    With MSFlexGrid1
        If KeyAscii = Keys.Return And .Row < .Rows - 1 Then
      .Row = .Row + 1
        End If
    End With
    If KeyAscii = 0 Then
     eventArgs.Handled = True
End If
End Sub
```

Taking a good look at the code, notice that you use the same KeyDown, KeyPress, EnterCell, and Change events as in the previous section, as these events are fairly standard to all applications with this sort of interface. You also use the DimTextBox procedure and the form's Load event to set up the grid's default size. This Load event sets up a grid to hold all of the information needed to perform this iteration on a surface that consists only of four nodes. At the end of the procedure the Load event also stores this number of nodes in the Public variable NumNodes and stores the value of the highest row number that data was entered into in the Public variable MaxRow.

The purposes of these two Public variables is apparent when you look at the Click event for Command1, the reformatting command button. This Click event uses information in these variables to erase all of the grid's information. The procedure then reformats the grid to handle another matrix of the size specified in Text2. As you can see, all of the row and column labels change to correspond to this newly sized grid. Once the grid is established at the correct size, the user enters the correct data and proceeds to clicking Command2. This initiates the Click event of Command2, which contains the code for the Gauss-Siedel iteration discussed in Chapter 5. The procedure reads the matrix information into the required arrays and performs the required iteration. As the procedure iterates, it formats a portion of the grid (below the data entry region) to display the iteration's calculated values. Thus, if the system of equations is a solvable set of equations (as they should be for a real system), you can watch the calculated values converge to within the specified criterion (see Figure 9-2 for an example set). If the equations aren't solvable, the Gauss-Siedel iteration diverges over time and leads to an overflow error. Thus, to challenge yourself, modify the routine so it only runs for X number of iterations and outputs an error message box saying that the results didn't converge in X number of iterations. It can't be stressed enough that good scientific and engineering applications take measures to ensure both the application's stability and the data's integrity.



	ai1	ai2	ai3	ai4	ai5	ai6	ai7	ai8	Ci	Estimations
1	-8	2	2	0	0	0	0	0	-2000	500
2	4	-8	0	2	0	0	0	0	-1000	500
3	2	0	-8	2	2	0	0	0	-1000	900
4	0	2	4	-8	0	2	0	0	0	1000
5	0	0	2	0	-8	2	2	0	-1000	900
6	0	0	0	2	4	-8	0	2	0	500
7	0	0	0	0	4	0	-18	2	-4000	900
8	0	0	0	0	0	4	4	-18	-3000	400

Iteration	T1	T2	T3	T4	T5	T6	T7	T8
1	600	675	750	668.75	662.5	598.4375	413.8889	391.6281
2	606.25	595.3125	609.375	603.125	530.4253	513.9009	383.6088	366.1133
3	551.1719	551.3672	546.1806	539.4073	485.9226	469.3414	370.8843	353.3835

Figure 9-2. A sample Gauss-Siedel iteration

The preceding issue aside, though, this can be a powerful and highly useful iterative technique when used properly. Hopefully, this application also gave you some insights into the types of powerful routines you can develop using spreadsheet type interfaces.