Explore your Enterprise

Create a Windows 95 Explorer-style database explorer.

by Chris Barlow

You can write some neat applications using the same "explorer" paradigm that Microsoft used in Windows 95 (and now Windows NT 4.0). At my company, SunOpTech, we use this new paradigm in our ObjectJob Explorer, which is part of our ObjectJob manufacturing decision-support system. The ObjectJob Explorer allows the user to explore his or her enterprise, examining the available capacity of plants, departments, work cells, and machines based on the current jobs in the system (see Figure 1). We have found that users can easily explore their enterprise with the ObjectJob Explorer because it provides them with a familiar interface. The familiarity shortens training time and increases the productivity of new users. You can write a similar application using two of the new controls built into Visual Basic 4.0: the TreeView and the ListView.

In the last several columns, I've written about the new controls available in Visual Basic 4.0 Professional Edition with Windows 95, including the RichTextBox, CommonDialog, ToolBar, and StatusBar controls. Last month I showed how to develop your own Visual Basic class to create splitter windows, which are an integral part of the explorer interface. This month we'll look at the TreeView and ListView controls that allow you to create the main windows of an explorer such as the ObjectJob Explorer.

If you are a confirmed Windows 95 user like me, you have probably been using the Windows Explorer to examine the folders and documents located on your computer and on other computers around the network. If you haven't used it yet, right-click on the Start button and select Explore.

The left window displays a hierarchy of data: My Computer, Network Neighborhood, Recycle Bin, and My Briefcase. This hierarchy begins with your Desktop and displays the nodes that are the next level down in the hierarchy, also known as the child nodes.

As you click on the plus symbol next to a node, that branch of the hierarchy expands to display the underlying child nodes. Expanding the My Computer node displays Floppy Drive A, Hard Drive C, CD-ROM Drive D, Control Panel, Printers, and Dial-up Networking. Expanding the Hard Drive C node displays the folders (subdirectories for you Windows 3.x readers) on the C drive.

Quite a bit of functionality is built into this window. A single click on the folder name changes the display to a text box where you can type in a new name for the folder. You can drag and drop a folder to a new location. Right-clicking on a node gives you a popup menu for actions related to this node. The neat thing is that you can implement this same functionality in your Windows 95, 32-bit Visual Basic programs using the TreeView control that comes with the Professional and Enterprise editions.

Similarly, you can use the ListView control to duplicate the functionality of the right window of the Windows Explorer. Notice that as you click on a node in the left window, the right window displays information about the child nodes. The information in the right window can be displayed in four different ways: large icons, small icons, list view, and detail view. You can single-click on one of the items in the right window to change the display to a text box where you can type in a new name for that item. You can also drag and drop between the right and left windows.

Finally, the vertical line between the left and right windows allows the user to change the vertical split to widen or narrow these windows. I've included my code for the CSplitter class with the source code for this column. Take a look at my May 1996 column, "Split Your Windows," for details on how the CSplitter class works.

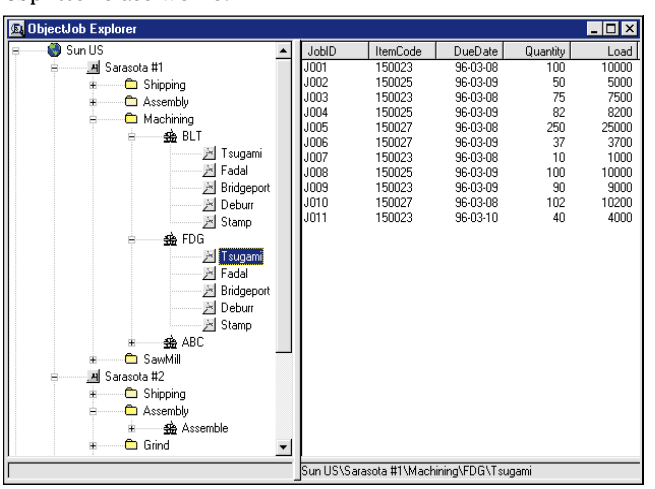

FIGURE 1 ***Let's Go Exploring.*** *Notice how easy it is to explore all the plants, departments, and machines in the enterprise and see the job data. The Sun US node represents the Organization; Sarasota #1 is one of the Plants; Shipping, Assembly, and Machining are Departments; BLT is a Cell; and Tsugami, Fadal, and so on, are Machines.*

Chris Barlow is president and CEO of SunOpTech, a developer of manufacturing decision-support applications including the ObjectBank and the ObjectJob Systems, where he and Ken Henderson hold a software patent related to decentralized distributed asynchronous object-oriented systems. Chris holds degrees from Harvard Business School and Dartmouth College where he worked with Drs. Kemeny and Kurtz on the BASIC language. Reach Chris on the Internet at ChrisB@SunOpTech.com or through SunOpTech's World Wide Web server at www.SunOpTech.com.

EXPLORING A DATABASE

The Windows Explorer is used to explore a drive. Its "data" consists of the folders and documents on the drive. The ObjectJob Explorer, however, explores a database containing information about the jobs being produced in a manufacturing plant. Each time the user clicks on a node, the app will read data from the database and display it in the explorer windows.

I've included with the source code a greatly simplified access database, OBJobExp.MDB, that contains the hierarchy relationship shown in Figure 2. Each Machine is located within a Cell, and each Machine has a daily Supply (how long the machine will run each day) in seconds. Cells are located within a Department, Departments are located within a Plant, and Plants are located within an Organization. Finally, Jobs are assigned to a particular machine and each Job is for production of a certain ItemCode, quantity, and load.

Each table in this simplified database has a field that links its records to a single record in its "parent" table in a traditional one-to-many relationship. For example, because the parent of the Plants table is the Organization table, each record in the Plants table has an OrgID field that contains one of the OrgIDs in the Organization table to indicate the Organization that this Plant is part of. Similarly, because the Departments table is the child of the Plants table, you will find a PlantID in every record in the Departments table to link the Department to its Plant.

In the left window, using the TreeView control, you'll want to display this hierarchy of Organizations, Plants, Departments, and so on. When the user clicks on a Machine node, you'll want to fill the ListView control in the right window with data from the Jobs table for that Machine (see Figure 1). When the user clicks on a higher node in the hierarchy, you'll want to display summary information about the children of that node. For example, if the user clicks on a Cell (see Figure 3), you would display information about the percent utilization of each Machine in the Cell. Because you want your program to be fast no matter how many Jobs are in the database, you'll want to read the database for Job information only after the user has clicked on a Machine node.

CREATING THE INTERFACE

Enough background—let's start developing the application. Start a new project in Visual Basic. Select the Custom Controls menu item from the Tools menu, and make sure the Microsoft Windows Common Controls box is checked so that the TreeView and ListView controls are in your toolbox.

Draw a TreeView control on the left side of the form and draw a ListView control on the right side of the form. Add label controls under each of these controls. Then put a long, narrow CommandButton control to divide the left and right windows—this will be your splitter control. Finally, add an ImageList control to your form to contain the images you'll display for each node. Because this control will be invisible at run time, it doesn't matter where you locate it. Your form should look something like the form in Figure 4.

If you look again at Figure 3, you'll see that the nodes of the

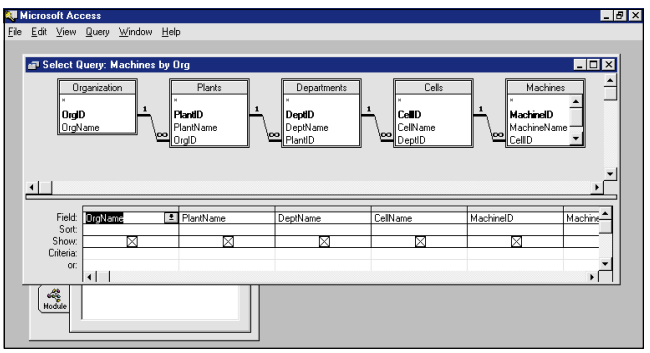

FIGURE 2 ***The Database Table Relationship.*** *Access allows you to set the table relationships and support cascading updates and deletes.*

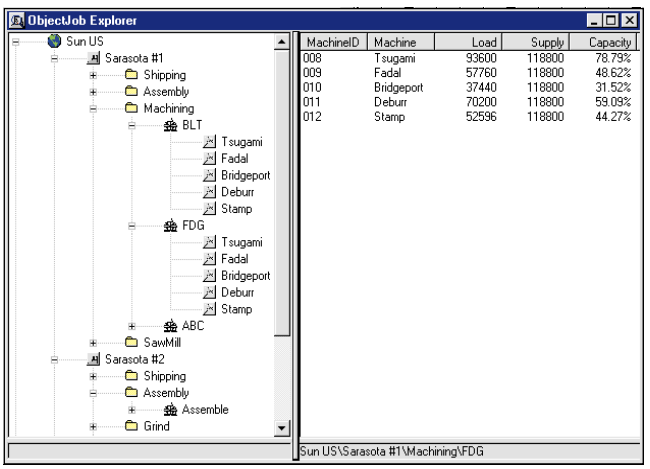

FIGURE 3 ***The Cell Node.*** *It is easy to display summary information, such as overall machine utilization, when the user clicks on parent nodes.*

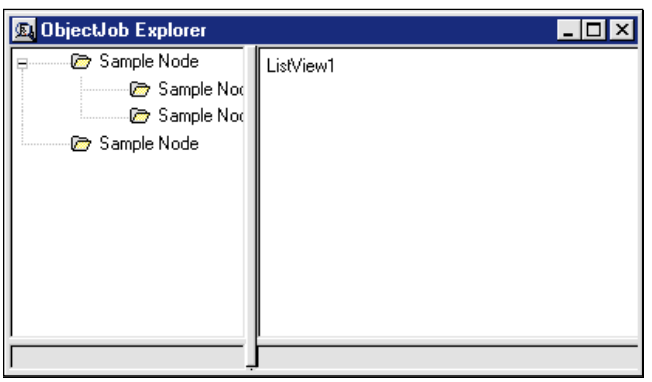

FIGURE 4 ***The ObjectJob Explorer Form.*** *These controls will be automatically realigned by the Csplitter class.*

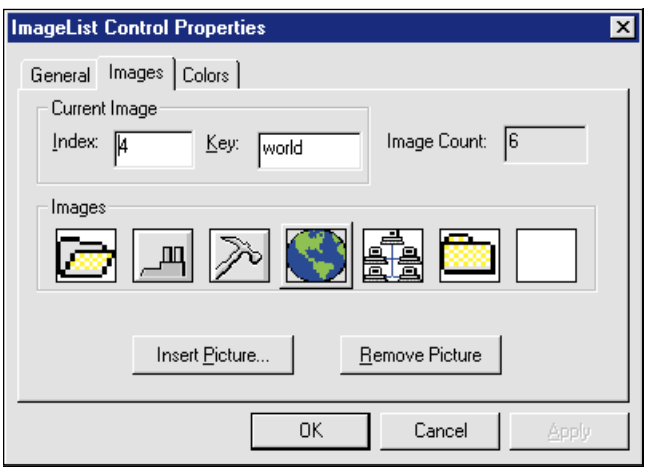

FIGURE 5 ***The ImageList Control.*** *It is easy to add your own images to this control and display them in the TreeView or ListView controls.*

TreeView control contain different images. Both the TreeView and ListView controls can display any of the images contained in the ImageList control. I've loaded six images into the ImageList control and given each image a Key that you can reference to load that image into one of the controls (see Figure 5). You'll need to set the ImageList property of the TreeView and ListView controls to point to this ImageList control.

Now add the CSplitter and CSplittee class modules to your project. Don't worry if you haven't had a chance to download those source files yet; your application will work, but you won't have the functionality of a splitter window.

The code for the ObjectJob Explorer begins by declaring four private variables for the application to use. MySplitter is your instance of the CSplitter class, and the Boolean variable Splitting is set to True when the mouse is moving the splitter control to change how the window is split. Similarly, the InDrag variable is set to True when the user is dragging a node to another location. Define the variable nodX as a Node object and use it to hold the node being dragged:

```
Option Explicit
Private MySplitter As New CSplitter
Private Splitting As Boolean
'True when splitting
Private InDrag As Boolean
'True when Dragging.
Private nodX As Node
'Item that is being dragged.
```

The code in the Form_Load event registers the controls with the CSplitter class and calls the LoadTree procedure to load the TreeView control with the information from the database:

```
Private Sub Form_Load()
'register the splitter control
MySplitter.Register butSplit

'indicate the left window controls
MySplitter.Add TreeView1, True, True
MySplitter.Add lbLeft, True, False

'indicate the right window controls
MySplitter.Add ListView1, False, True
MySplitter.Add lbRight, False, False

'load the tree from the database
LoadTree
End Sub
```

LOADING THE TREE

Because the OBJobExp database may contain many jobs, but only a small number of machines, you can load the entire TreeView control in a single procedure. Begin by loading data from the top-level Organization table into the TreeView control. The first step is to open the database and open a record set containing the proper data. The OpenDatabase statement uses the Path property of the App object to point to the database in the same location as the application. The OpenRecordset method of the Database object opens the highest-level table in the hierarchy, the Organization table:

```
Dim db As Database
Dim rs As Recordset
Dim SQL$
Set db = OpenDatabase(App.Path & "\OBJobExp.mdb")
Set rs = db.OpenRecordset("Organization")
```

Like most of the other new VB4 controls, the TreeView control is made up of a collection of objects—in this case, Node objects. Use the Add method of the Nodes collection to add Node objects to the control. The Add method can take six arguments, but only one is required: the text to display for the Node. In your code for the top-level nodes you'll only need to use two additional arguments: the image to display and the unique key for this node.

Warning: there is a trick to creating a unique key for a Node. The Key is a string property, but you can't just convert a number to a string with the CStr function and use that as the Key. The control requires that the Key be alphanumeric. The technique I used is to concatenate the numeric ID from the database to the first letter of the table name to create a unique key.

LIKE MOST OTHER NEW VB4 CONTROLS, THE TREEVIEW CONTROL IS MADE UP OF A COLLECTION OF OBJECTS—IN THIS CASE, NODE OBJECTS.

Now you can add a top-level Node to the Nodes collection for each record in the Organization table by looping through the record set while the EOF property is not True. The ID is in the first field of the record set, and the name is in the second field. I used the image from the ImageList control with the Key of "world." After adding the node, use the MoveNext method of the Recordset and then close the record set when finished:

```
Do While Not rs.EOF
   Set nodX = TreeView1.Nodes.Add(, , "O" & rs(0), _
      CStr(rs(1)), "world")
   rs.MoveNext
Loop
rs.Close
```

Now that you have nodes for each of the Organizations in your database, you can add nodes for each of the Plants and link them to their Organization node. To make this link and insert the node in the proper location, use two of the optional arguments in the Add method—the Relative and Relationship arguments. Set the Relative argument to the ID of the parent organization and set the Relationship argument to tvwChild. Use the same trick to create a unique key for the plant nodes by beginning the key with the letter P. Use the "plant" image for these nodes:

```
Set rs = db.OpenRecordset("Plants")
Do While Not rs.EOF
   Set nodX = TreeView1.Nodes.Add("O" & rs(2), _
      tvwChild, "P" & rs(0), CStr(rs(1)), "plant")
   rs.MoveNext
Loop
rs.Close
```

You can use almost this same code for the Departments, Cells, and Machines tables. Don't forget to close your database at the end with code like this:


```
rs.Close
Set rs = Nothing
db.Close
Set db = Nothing
End Sub
```

Try running your app now. You should see the filled TreeView control and you should be able to expand and contract all the nodes in the tree. Neat!

CLICK AND LIST

Now you can add the functionality to display the Jobs when the user clicks on a Machine node. Add this code in the TreeView control's Node click event to call the LoadList procedure and pass the Node object:

```
Private Sub TreeView1_NodeClick(ByVal Node As Node)
'here add the next level down
'and fill ListView
LoadList Node
End Sub
```

The LoadList procedure will be called anytime a node is clicked on, whether it is a Machine node or another node in the hierarchy. The best way to handle this is to call different procedures from this procedure based on the Key of the Node. Remember that you used the first letter of the table to create a unique key for the node. Now you can use a Select statement on the first character of the Key property of the passed Node. Call the LoadJob procedure in the machine case. You can add procedures for the other node types later. This is a good place to set the right label to the FullPath property of the Node object, so the user can easily see which node is selected:

```
Private Sub LoadList(Node As Node)
lbRight = Node.FullPath
'select by record type
Select Case Left(Node.Key, 1)
Case "M" 'Machine
   LoadJob Mid(Node.Key, 2)
End Select
End Sub
```

The LoadJob procedure will fill the ListView control with five columns of data: the JobID, ItemCode, DueDate, Quantity, and Load. Like the TreeView control, the ListView control is made up of collections of other objects. The ColumnHeaders collection contains ColumnHeader objects. You can use the Add method of the ColumnHeaders collection and set the text in the column and the column width.

First dimension the database and record-set variables as well as variables to hold the ListItem and ColumnHeader objects:

```
Private Sub LoadJob(Key$)
Dim db As Database
Dim rs As Recordset
Dim SQL$
Dim itmX As ListItem
Dim clmX As ColumnHeader, cWidth As Integer
```

Then open the database and use the ListView control to clear the ColumnHeaders collection:

```
Set db = OpenDatabase(App.Path & "\OBJobExp.mdb")
ListView1.ColumnHeaders.Clear
```

Add ColumnHeader objects and set their widths based on the current width of the ListView control:

```
cWidth = (ListView1.Width - 1500) \ 5
Set clmX = ListView1.ColumnHeaders. _
   Add(, , "JobID", cWidth)
Set clmX = ListView1.ColumnHeaders. _
   Add(, , "ItemCode", cWidth)
Set clmX = ListView1.ColumnHeaders. _
   Add(, , "DueDate", cWidth, lvwColumnRight)
Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Quantity", cWidth, lvwColumnRight)
Set clmX = ListView1.ColumnHeaders. _
   Add(, , "Load", cWidth, lvwColumnRight)
```

Now that you have added the columns, you can read the database and add the job information. Build a SQL statement to select all fields from the Jobs table where the MachineID field is equal to the MachineID passed in the Key argument:

```
'load Jobs
SQL = "select * from Jobs where MachineID = "
SQL = SQL & Key
Set rs = db.OpenRecordset(SQL)
ListView1.ListItems.Clear
```

Loop through this record set and use the Add method of the ListItems collection to add the JobID to the ListView control. Each ListItem object has an array of SubItems to contain the other information in the columns. Set these SubItems to the appropriate fields in the Jobs table. Note the use of the placeholders "@" in the Format function to facilitate sorting of numeric data:

```
Do While Not rs.EOF
   Set itmX = ListView1.ListItems.Add(, "J" _
      & rs(0), Format(rs(0), "J000"), "closed")
   itmX.SubItems(1) = rs(1)
   itmX.SubItems(2) = Format(rs(3), "yy-mm-dd")
   itmX.SubItems(3) = Format(rs(4), "@@@@@")
   itmX.SubItems(4) = Format(rs(5), "@@@@@")
   rs.MoveNext
Loop
rs.Close
Set rs = Nothing
db.Close
Set db = Nothing
End Sub
```

Run your application and expand the tree until you can click on one of the machine nodes. If you click on any of the machine nodes in the "SunUS\Sarasota #1\Machining\FDG" path, you should see the ListView fill with job data.

Now that you have a working application, it is easy to add more functionality. I added code to the LoadList procedure to call the LoadSummary procedures for other nodes and display the percent utilization of the child nodes. I also added code in the TreeView events to support the Node drag-and-drop actions and code in the ListView events to support both ascending and descending sorts by column. Although the code for the entire project is not listed here, it is available on line in a file called GS0696.ZIP. Download the file from *VBPJ's* Development Exchange on the World Wide Web at http://www.windx.com, or from the *VBPJ* CompuServe Forum, or MSN site. For details, see "How to Reach Us" in Letters to the Editor. See what else you can add to the application and e-mail me a copy of your best new features. ×