Display Self-Relational Data in a Microsoft TreeView

Add greater functionality to your Access and Visual Basic applications with the Microsoft TreeView.

By Steve Clark

This article is part two of three in a series describing the storage and usage of self-relational data. The first article (August 2001) focused on the storage and retrieval of self-relational data was the primary focus. In this article, I show you how to display hierarchical data using a Microsoft TreeView control. Also, I use the ImageList control to demonstrate how to specify which icons to display in the TreeView.

For this article, I created examples using ADO, so I assume you're working with Access 2000. But, you can also incorporate the TreeView code and techniques in this article into your Access 97 applications as well. The application I developed for my client was in Access 97.

Build on what you know

Before we get started, there is some unfinished business from the first article. For those of you returning, you were given two problems to solve. The first assignment was to display all employees, their supervisor, and their supervisor's supervisor. Building on what you had learned in the first article, you needed to add a third occurrence of the Employees table, create the Right Outer Join, and add the Next level supervisor's name (listing 1). The results are shown in figure 1. Table 1 lists the employees and their supervisors.

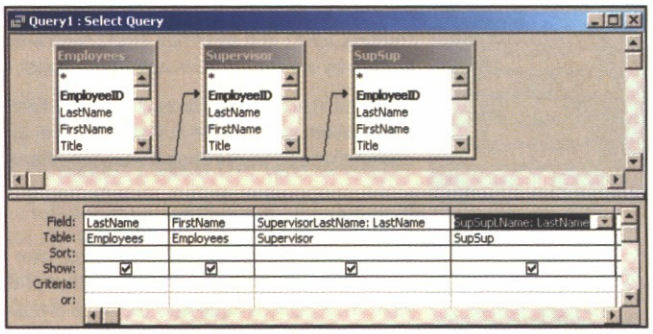
***Listing 1:* SQL statement**—Use this code to display all employees and two levels of supervisors.

```
SELECT Employees.LastName,
Employees.FirstName,
Supervisor.LastName AS SupervisorLastName,
SupSup.LastName AS SupSupLName
FROM (Employees AS SupSup
RIGHT JOIN Employees AS Supervisor
ON SupSup.EmployeeID =
Supervisor.ReportsTo)
RIGHT JOIN Employees
ON Supervisor.EmployeeID =
Employees.ReportsTo;
```

Continued

TECHNOLOGIES

- ADO
- Access 2000/97
- Visual Basic 6.0



Field:	LastName	FirstName	SupervisorLastName: LastName	SupSupLName: LastName
Table:	Employees	Employees	Supervisor	SupSup
Sort:				
Show:	✓	✓	✓	✓
Criteria:				
or:				

***Figure 1:* Query design**—This displays all employees and two levels of supervisors.

Steve Clark is a Senior Systems Analyst on the FMS, Inc. Custom Database Solutions Team. The team creates custom database solutions using VB, SQL Server, Microsoft Access, Excel, Word, and FrontPage, as well as ASP, Cold Fusion, XML, and soon VB.NET. Steve has a Bachelor of Science in Computer Science from the University of Cincinnati. Steve.Clark@FMSInc.com.

Table 1: **Results**—Data results of employees and two levels of supervisors.

Last Name	First Name	Supervisor LastName	SupSupL Name
Davolio	Nancy	Fuller	
Fuller	Andrew		
Leverling	Janet	Fuller	
Peacock	Margaret	Fuller	
Buchanan	Steve	Fuller	
Suyama	Michael	Buchanan	Fuller
King	Robert	Buchanan	Fuller
Callahan	Laura	Fuller	
Dodsworth	Anne	Buchanan	Fuller

The second homework question was how would you display all companies, as well as any of the companies that it owns. In table 2, you can see the data model in question, where tblCompany stores each of the companies, and tblOwnership maintains the information of which companies own which other companies.

Table 2: **Data model**—This table is created using data from the previous article.

tblCompany	tblOwnership
CompanyID	OwnerID
CompanyName	ChildID
CompanyFax	UnitsOfOwnership
CompanyTel	
etc...	

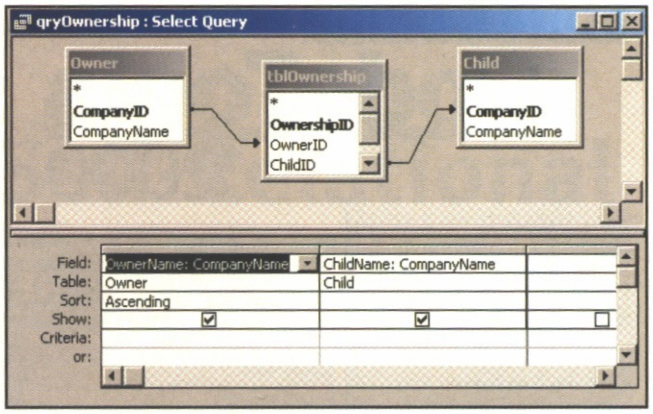
Since you need to see all the companies, you have to make a Left Join from Company to Ownership, and to get the owned company's name (or other non-primary key information), you need to continue the relationship from Ownership to another occurrence of the Company table, this time using a Right Join. Listing 2 shows the SQL to accomplish this, and figure 2 shows the end result in the Query design view.

Listing 2: **All companies**—This SQL statement displays all companies and any others they own.

```
SELECT Owner.CompanyName
AS OwnerName,
Child.CompanyName AS ChildName
FROM tblCompany AS Child
RIGHT JOIN (tblCompany AS Owner
LEFT JOIN tblOwnership
ON Owner.CompanyID = tblOwnership.OwnerID)
ON Child.CompanyID = tblOwnership.ChildID
```

Microsoft TreeView control

Now that you better understand the storage and retrieval of self-relational data, the next step is to focus on displaying the data using a Microsoft TreeView control. Examples of TreeView controls can be found in Windows Explorer, Outlook, and many custom applications where a hierarchy exists. The TreeView control is shipped with most versions of Visual Basic and can also be found in all versions of the Microsoft Office Developers Edition (ODE).



Field:	OwnerName: CompanyName	ChildName: CompanyName	
Table:	Owner	Child	
Sort:	Ascending		
Show:	☑	☑	☐
Criteria:			
or:			

Figure 2: **All companies**—Query Design to display all companies and any others they own.

To implement the TreeView, you must first create a form and insert the TreeView ActiveX control. After the control is placed, it should be named and sized to your desired values. To insert the Microsoft TreeView control:

1. Select "Insert" from the Main Menu
2. Select "ActiveX Control..."
3. Select the Microsoft TreeView control

For this article, I use the TreeView control from Visual Basic version 6.0 with Service Pack 4, but most versions should behave the same. After closing the dialog, the control will be inserted on your form. Change the name to tvxEmployees because you'll be using this control later.

Next, add an ImageList control to the form so you can add icons to the TreeView display. Using the ActiveX controls dialog, add an ImageList control to the form as well.

Customize the ImageList control

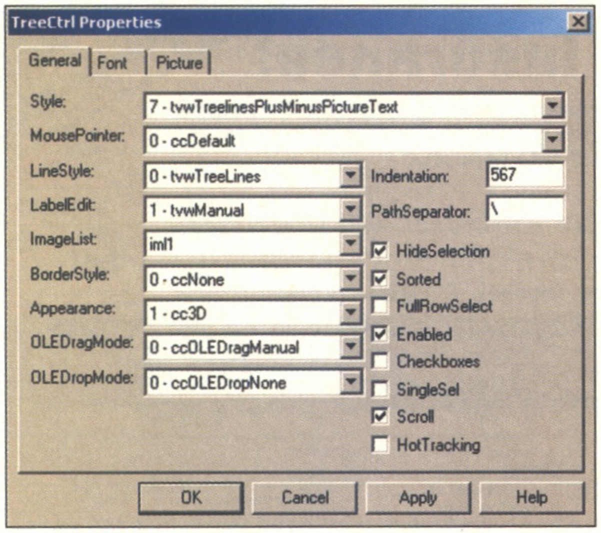
The ImageList control is a storage area for pictures that the TreeView references when it needs to display either the selected or non-selected item. To specify the desired images to be used, right-click on the image control, select ImageListCtrl Object, then select Properties.

This opens the ImageListCtrl Properties dialog. Here is where you specify the attributes of the icons, as well as the icons themselves. On the General tab, you can select the size of the icons. I recommend the 16 x 16 choice, as it is the smallest, and most non-obtrusive choice. The Images tab is where you specify the desired images, and the colors tab lets you play with the colors of the icons.

To add an image to the Image List control, click on the Images Tab and click on the Insert Picture button. This opens the standard Windows File Open dialog, and you can navigate to your favorite opened and closed icons.

After you have saved the icons in the ImageList, you must alert the TreeView that this is the ImageList you would like to use for it. It is possible to have more than one ImageList per form, so be specific.

To associate the Image List control to the TreeView control, right-click on the TreeView control object, select the TreeCtrl Object option, and select the Properties choice. This will open the Properties dialog, where you can customize the behavior



TreeCtrl Properties

General | Font | Picture

Style:	7 - tvwTreelinesPlusMinusPictureText		
MousePointer:	0 - ccDefault		
LineStyle:	0 - tvwTreeLines	Indentation:	567
LabelEdit:	1 - tvwManual	PathSeparator:	\
ImageList:	iml1	✓	HideSelection
BorderStyle:	0 - ccNone	✓	Sorted
Appearance:	1 - cc3D	☐	FullRowSelect
OLEDragMode:	0 - ccOLEDragManual	✓	Enabled
		☐	Checkboxes
OLEDropMode:	0 - ccOLEDropNone	☐	SingleSel
		✓	Scroll
		☐	HotTracking

***Figure 3:* Display icons**—Here you can specify the icons to display on the TreeView control.

of the TreeView (figure 3). Locate the ImageList property, and set it to the name of the control.

Mission objective: Employee TreeView

Let's create a TreeView of employees where the first level of data are all supervisors, and all subsequent levels of data are employees assigned to that supervisor.

Populating the TreeView control

Any entry in a TreeView is considered a node. To populate the tree with nodes of your data, you will need to use the .Add method of the Nodes collection of the TreeView control. For example:

```
tvx.Nodes.Add Key:="S105", Text:="Ulrich, John", _
   Image:= 1, SelectedImage:=2
```

This example will add the text "Ulrich, John," to the top level of the tree, assigned to the TreeView object variable tvx, and assign to it a unique key of S105. Image and Selected Image values come from the ImageList control. When an image is inserted into the ImageList control, each image has an index automatically entered. Image is used when the node isn't selected, SelectedImage is used when the node is the current node.

It is important to note at this point that the Key parameter is of type Text, and it must start with a letter. In this example, the letter S is used to represent a supervisor. To make matters more complicated, the key must be unique, so if any values in your data have the possibility of repeating, you have to plan for it. This will be dealt with later in code, but it's something to prepare for.

Populating the top level

When you populate your tree, there may be hundreds of records, or hundreds of thousands of records. Depending on your situation, you may not be able to populate the entire tree at once, as it may be too time consuming at startup for the user. In this case, implementing an "On Demand" approach will mean waiting until the user opens a node before attempting to gather the child data for it.

Code disclaimer: Naming conventions and error trapping. I use the prefix tvx for a TreeView to differentiate between custom objects I've made and the intrinsic constants used by Access. For example, tvwChild denotes the index of the first child node of a parent. So, it makes life easier when attempting to use Auto List Members (Ctrl-Space) to find your own objects. I assume you'll include your own error trapping system, so I have omitted it from my coding examples. If you're using the Northwind database that ships with Access 2000, be sure to add a reference to the "Microsoft ActiveX Data Object 2.1 Library" or else the ADO code in listing 3 won't compile.

For the supervisor-to-employee example, the top level will be all the supervisors. First, you must load all the supervisors, then wait for the user to expand a supervisor node, prior to gathering any employee information.

ShowSupervisors()

Listing 3 assumes you're using the Northwind database and the Employees table. In the first article, you added a Yes/No field called Supervisor and a created a query called qlkpSupervisors. On a form, you added a TreeView and ImageList control. The TreeView control is named tvxEmployees. The ImageList has two pictures inserted with the indexes of 1 and 2.

***Listing 3:* ShowSupervisors()**—This code populates the top level of the tree with supervisor information.

```
Sub ShowSupervisors()

  'Notice that MSComctlLib must be used
  Dim tvx As MSComctlLib.TreeView
  Dim rst As ADODB.Recordset

  'Notice that .Object must be used to make Auto List
  'Member appear.
  Set tvx = tvxEmployees.Object
  Set rst = New ADODB.Recordset

  With rst
    'Open the Supervisor Lookup Query
    .Open "qlkpSupervisors", _
       Application.CurrentProject.Connection

    Do While Not .EOF
      'Make Key = "S" & EmpID, and Display LN, FN.
      tvx.Nodes.Add Key:="S" & !EmployeeID, _
         Text:=!LastName & ", " & !FirstName, 1, 2
      'Go to next Supervisor.
      .MoveNext
    Loop
  End With

End Sub
```

To trigger this code, call the ShowSupervisors procedure from the Form's On Load event property:

```
Private Sub Form_Load()
  ShowSupervisors
End Sub
```

Populate the children on demand

With the top level in place, the user can click on a supervisor and reveal the employees, except that there is no data there to display, as it hasn't yet been populated. To populate a supervisor's employees on demand, you can execute a populating procedure when the user triggers either the NodeClick or Expand event property of the TreeView control. Note: The TreeView has both a Click and a NodeClick property. If you need to trap for a particular node, you must use NodeClick. The Click event property is used for the entire TreeView.

In the following procedure headers, a Node object is exposed as a parameter so you can reference which node was triggered:

```
Private Sub tvxEmployees_Expand(ByVal Node As Object)
Private Sub tvxEmployees_NodeClick(ByVal Node As Object)
```

Note: Because the TreeView is an ActiveX control, you can't generate the above procedure headers from the form's property window. To create them, open the Code Editor window, select the TreeView control name from the Object combo box on the left side, then select either Expand or NodeClick from the Event combo box on the right side.

Suppose the user has clicked on a Supervisor node. This can be used as the trigger to populate the supervisor with all his or her employees. Using the TreeView's NodeClick event properties, a call to a user-defined procedure called ShowEmployees can populate the data using the exposed node as the parent.

```
Private Sub tvxEmployees_NodeClick(ByVal Node As Object)
  ShowEmployees Node
End Sub
```

The code for ShowEmployees is in listing 4.

Adding a child node

In a previous example of the .Add method, nodes were added to the top level of the TreeView. Now, let's add nodes as children of existing nodes. To accomplish this, there are two other parameters of the .Add method that specify the Parent node to be used and the relationship between new node and the existing one. For example:

```
tvx.Nodes.Add nodParent.Index, tvwChild, "E205", "Ulrich,
John", 1, 2
```

With this example, the index of the parent node is specified, and the relationship is defined using the intrinsic constant tvwChild. This example is used in listing 4 with the node-passed to the procedure as the parent node. Table 3 defines the node properties used in listing 4.

***Listing 4:* Add a child node**—Populate the employees for a selected supervisor.

```
Sub ShowEmployees(pnod As Node)

  Dim tvx As MSComctlLib.TreeView
  Dim cnn As ADODB.Connection
  Dim cmd As ADODB.Command
  Dim rst As ADODB.Recordset
  Dim intPos As String
  Dim strEmpID As String

  'Used to generate a unique ID.
  Static slngCnt As Long

  Set tvx = tvxEmployees.Object
  Set cnn = Application.CurrentProject.Connection
  Set cmd = New ADODB.Command

  'Check if already populated.
  If pnod.Children = 0 Then
    'Even if no children, we may have attempted
    'to populate previously.
    If len(pnod.Tag) = 0 Then
      'Extract the EmployeeID from .Key.
      If Left$(pnod.Key, 1) = "S" Then
        'Extract EmpID from the format "S####"
        strEmpID = Mid$(pnod.Key, 2)
      Else
        'Find the period.
        intPos = InStr(pnod.Key, ".") - 1
        'Extract the EmpID from the format "E####.##".
        strEmpID = Mid$(pnod.Key, 2, intPos)
      End If

      'Prepare the query to extract children info.
      Set cmd.ActiveConnection = cnn
      cmd.CommandType = adCmdText
      cmd.CommandText = _
        "SELECT * FROM Employees WHERE ReportsTo = _
        " & strEmpID

      Set rst = New ADODB.Recordset
      With rst
        'Get children for this employee.
        .Open cmd
        Do While Not .EOF
          'Make Key = E + EmpID + Period(.) + UniqueID
          'and display Lastname, Firstname.
          tvx.Nodes.Add pnod.Index, tvwChild, _
            "E" & !EmployeeID & "." & slngCnt, _
            !LastName & ", " & !FirstName, 1, 2
          'Bump unique index.
          slngCnt = slngCnt + 1
          'Go to next child.
          .MoveNext
        Loop

      End With

      'Mark node as populated.
      pnod.Tag = "X"
      If pnod.Children > 0 Then
        'Ensure that first child is visible.
        pnod.Child.EnsureVisible
      End If 'pnod.Children > 0
    End If 'pnod.Tag = vbNullString
  End If 'pnod.Children = 0

End Sub
```

ROCK SOLID
INSTALLATIONS
We specialize in building and troubleshooting Installations.
SageKey
Software Installation Experts
Access 2002 Runtime Scripts
Access 2000 Runtime Scripts
Access 97 Runtime Scripts
Satellite Forms Runtime
Access 2000 Object
Extended Security
Palm Pilot Installer
www.sagekey.com
1-888-248-0055

There are several techniques used in this procedure that need further explanation, such as marking the node as populated, and creating a unique key for each node, while incorporating the EmployeeID for later reference.

A node is marked as populated by setting the node's tag property with the character "X." This technique is included because you can attempt to populate a node, but it doesn't have any children. If the user attempts to populate this node again, you'll see that it has none, and cease any further processing. This flag saves you time by preventing unnecessary data accessing. When a parent node is passed to the ShowEmployees procedure, you need to know the parent's EmployeeID, so you can create the query to find the correct children. So, adding the EmployeeID to the key seems like the best way to retain the information for later use. I would have used the tag property, but it was already assigned a task.

Because each employee has the possibility of being in the tree more than once, a plan must be devised to ensure that all node keys would be unique. First, all keys must start with a letter, so E was chosen to represent employee. From there, you add the EmployeeID. Unfortunately, you can't stop there because "E101" won't be a unique key of the employee appears in the tree again. To ensure a unique key, I add a unique ID by maintaining a static variable throughout the life of the form. So, after the EmployeeID, add a period and the unique ID (i.e., E101.56). Later, when you need to call the EmployeeID, you can extract all the characters after the E, but before the period using a series of string manipulation functions.

Table 3: **Nodes**—These are the node properties used in code and their descriptions.

.Index – Unique Number assigned to a node by VBA	
.Key – Unique Value assigned by developer, must start with a letter	
.Children – Number of children a node has	
.Child – The first Child Node of a parent	
.Tag – Text field for any use	
.EnsureVisible – Make node visible in TreeView	

Wrap up

Now that you've been exposed to the inner workings of the TreeView control, you can see that adding a significant amount of functionality (and cool eye candy) to your Access or Visual Basic applications can be done without extensive programming. The TreeView object is easy to manipulate, even with minimal amount of knowledge of its inherent properties. It is useful in self-relating situations, but can also be incorporated in more simple situations, such as customers to orders to order details.

In the next article, I'll show you how to export the self-relating data to an Excel file and use Microsoft Visio to produce an organizational chart from it. ADVISOR

NEED PROFESSIONAL HELP?

Whether you need a quick tip or want us to develop a database solution, we can help you. Just drop us an e-mail that describes your problem or need, attach all the necessary files, and we will send you a quote and time table for the solution.

Small problems will be solved in the same or next day.

We are experts in the following fields:

VB • Access • Oracle • SQL Server • Web* • Excel & Word Automation
Finance-related databases • Interfacing with existing systems

Click on Dev
Expert Consulting Services

web: http://www.clickondev.com
email: sales@clickondev.com
telephone: 212-859-3521

*Unique ability to deliver fast, complex database-related Web sites.