

## Database Sample

Sample Description: [Database Sample](#)

### Points of Interest

[Connecting the DataControl to the Data Source](#)  
[Binding the Form's TextBoxes to the DataControl](#)  
[Binding the Form's OptionGroup to the DataControl](#)  
[Populating the Indented List with Department Numbers](#)  
[Creating a new Record](#)  
[Deleting the Current Record](#)  
[Updating the Current RecordSet](#)  
[Using an IndentedList for Selecting Records](#)

### Controls

Option Button  
DataControl  
IndentedList

For Help on Help, Press F1

### **Database Sample**

The purpose of this sample is to present several common development techniques that are used when creating a database application. This includes adding, deleting, updating and navigating database records.

Our sample consists of a form which is used to maintain information about employees in a company. The sample consists of a DataControl which is connected to a standard ASCII text file located in the sample's directory. There are four TextBoxes and an OptionButton group which present information about each employee. These textboxes and option buttons are bound to the DataControl.

An IndentedList is presented at the top of the form and is used to navigate the employees by the department they belong to. The IndentedList displays a list of Department numbers. Clicking on a specific Department number will expand that entry and present a list of employees from that department.

### Connecting the DataControl to the Data Source

The DataControl is displayed at the bottom of the form. This control is a type of "HyperControl" made up of several individual controls linked to a RecordSet object. See information on DataControl to understand the functionality of the various navigation buttons on the control.

The DataControl is connected to a standard ASCII text file located in the sample's directory. The contents of the file are shown below. The length of each data field are as follows: Dept(5), EID(5), Name(20), Salary(10), Status(7).

Dept	EID	Name	Salary	Status
10	100	Jim Black	900.00	A
15	105	Henry Brown	1264.00	A
10	110	Gloria White	755.00	A
15	200	Sally Hunter	815.00	O
10	120	Jack Henderson	1450.00	T
10	125	Susan Jackson	860.00	A
10	130	Robert Stewert	1000.00	A
20	300	John McHenry	1050.00	O

To connect the DataControl to this ASCII file, you place "edit focus" on the form and double-click on the DataControl. This executes its **DetailedEdit** method which displays a Database Wizard. The Database Wizard is used to connect to the file "employee.txt" in the samples directory.

The data in the employee.txt file is Fixed ASCII and contains a total of (5) fields. Since the first row of the data file contains field names, the size of the field must be at least as long as the field name, otherwise the name of the field would be cropped when the RecordSet is updated.

### **Binding the Form's TextBoxes to the DataControl**

Binding the form's controls to the DataControl is accomplished by the Database Wizard. The (4) TextBoxes on the form are bound to the DataControl on the last step of the Database Wizard. As this is done, the **DataField** and **DataSource** properties are set for each TextBox.

### Creating a new Record

The **New** Button on the EmployeeForm is used to add a new record to the underlying RecordSet. The primary code to do this is:

```
' Add a new entry to the recordset  
DataControl1.RecordSet.AddNew
```

The AddNew method automatically clears the contents of the (4) bound TextBoxes. When a new record is created, the (3) OptionButtons are set to their default values as follows:

```
OptionGroup.Option1.Value = True  
OptionGroup.Option2.Value = False  
OptionGroup.Option3.Value = False
```

Finally the focus of the first TextBox is set so the user may begin entering data about the new employee.

```
' Place the type-in point in the Employee ID textbox  
TxtEmployeeID.SetFocus
```

### Deleting the Current Record

The **Delete** Button on the EmployeeForm is used to remove the current record from the underlying RecordSet. A **YesNoBox** is used to prompt the user is first prompted if they really want to remove the record.

The following code example actually is used to remove the current record.

```
With DataControl1.RecordSet
    If Not .EOF Or Not .BOF Then
        .Delete
        .MoveNext
        If .EOF Then .MovePrev
    End If
End With
```

In addition, this code also causes a move to the next available record. If we are removing the last record, the **EOF** would return True, so we would move to the previous record with the **MovePrev** method. Otherwise we would move the next record with the **MoveNext** method.

After a record is deleted, the data source is updated through the following code:

```
DataControl1.RecordSet.UpdateAll()
```

Finally, the IndentedList is updated. If the employee record being removed is currently selected in the IndentedList, the entry is removed from the IndentedList.

### Updating the Current RecordSet

The **Update** button on the EmployeeForm is used to save the current information stored in the DataControl's RecordSet object to the employee.txt ASCII file. This is accomplished by the BtnUpdate\_Click method. The first step is to check to make sure that we have sufficient data to update. This is done through the following two checks.

```
' Need to validate vital employee information
If TxtEmployeeID.Text = "" Then
    InfoBox.Message("Warning", "No Employee ID has been entered.")
    Exit Sub
End If

If TxtDepartment.Text = "" Then
    InfoBox.Message("Warning", "No Department number has been entered.")
    Exit Sub
End If
```

If all's well, then the following UpdateAll method is executed.

```
' Save the contents of the recordset to disk file
DataControl1.RecordSet.UpdateAll()
```

Once the data has been updated, the IndentedList is cleared and then repopulated with updated information.

```
' Need to update the indented list
PopulateList
```

### Using an IndentedList for Selecting Records

The main purpose of the IndentedList in this sample is to allow the user to click on a department number, see the names of employees in that department and then click on a particular employee to navigate to that employee's record.

When the IndentedList is first populated with Department numbers, each line in the IndentedList is able to expand because each line had the **SetItemCanExpand** method set to True. At the time, since only department numbers were being populated, we know that each line is capable of being expanded. The IndentedList has an **Expand** event handler defined as follows:

```
Sub IndentedList1_Expand(ByVal itemIndex as Integer, ByVal itemData as Long, itemObj as Object)
    Dim i as Integer
    Dim sdept, fdept, eid As String

    sdept = IndentedList1.ItemString(itemIndex)
    ' Search the recordset to find all employees in sdept department
    For i = 0 To DataControl1.RecordSet.RowsRead - 1
        fdept = DataControl1.RecordSet.Row(i).Column(0).Value
        If sdept = fdept Then
            eid = DataControl1.RecordSet.Row(i).Columns(1, 2).Value
            IndentedList1.InsertItem(eid, 2, 1, itemIndex + 1)
        End If
    Next i
    IndentedList1.Reset
End Sub
```

When the user clicks on the blue icon on a department line, this event handler is executed. It basically scans the underlying DataControl's RecordSet to find all employees who are in the expanded department (sdept). When an entry is found, it is added to the IndentedList.

When a department number is expanded, it may later be collapsed by clicking on the blue icon next to the department number. When this is done, the Collapse event is triggered and the following event handler is executed.

```
Sub IndentedList1_Collapsed(ByVal itemIndex as Integer, ByVal itemData as Long, itemObj as Object)
    IndentedList1.SetItemIcon(itemIndex, 1)
End Sub
```

When a department is expanded and displays the employees who are assigned to that department, the user may click on the employee and that employee's record will be made current. This is done through the following Click method:

```
Sub IndentedList1_Click()
    Dim icon_type As Integer
    ' Determine the type of icon the item has
    icon_type = IndentedList1.ItemIcon(IndentedList1.ListIndex)

    ' If the item is an icon_type = 2, it is an employee
    ' so we need to move the datacontrol to the right employee
    If icon_type = 2 Then
        Dim eid As String
        Dim i, row As Integer
        ' Get the indented list line entry
        eid = Trim(Left(IndentedList1.ItemString(IndentedList1.ListIndex), 5))
        ' Find the first blank space
        DataControl1.RecordSet.Column(1).FindFirst(eid)
        DataControl1.RecordSet.CurrentRecordNumber =
DataControl1.RecordSet.CurrentRecordNumber
    End If
End Sub
```

First the icon type is determined. If the icon type is 2, we determine the employee ID (EID) from the selected entry. The employee ID is used since you could have two employees with the same name. The RecordSet's **FindFirst** method is used to locate the employee's record. When it is found, the **CurrentRecordNumber** is set to the found number. In effect, this updates all the controls on the form to display the selected employee's record.



### Binding the Form's OptionGroup to the DataControl

The **OptionGroup** is a data-aware control named **OptionGroupMaster** which contains a form object named "Buttons" and an OptionButton control named "OptButton". Several methods are defined on the Buttons Form which are responsible for adding and removing OptionButton's to and from the form. The OptionGroupMaster object is known as a "self editing container".

The **AddValue** method dynamically embeds an OptionButton on the "Buttons" form. It takes (2) arguments - codex and Caption. The codex String on each OptionButton is responsible for setting the **Value** property on the "Buttons" form. In our example, we have (3) OptionButtons that are grouped together on the "Buttons" form. The corresponding codex and Caption values for these OptionButtons are:

Name	Caption	codex
OptButton1	Active	A
OptButton2	On Leave	O
OptButton3	Terminated	T

The Caption property is displayed on the form while the codex property is used to set the Value of the OptionGroup. The "Buttons" form has a *Procedural Property* named Value which is set by the **Update** event. When the DataControl move to a specific record, the OptionGroup receives an Update event and is passed the value of the employees "status". This value is used to set the Value procedural property. The setValue method is then automatically executed which basically scans all the OptionButton controls to identify the one that has a matching codex property. When the appropriate OptionButton is found, its Value is set to True.

The **RemoveValue** method is responsible for removing an OptionButton from the OptionGroup. It takes a single argument for a codex value and uses the **FindValue** method to located the matching OptionButton. If it is found, it is removed via. the **DestroyObject** method.

A **Resize** method is defined on the "Buttons" form which is used to automatically keep the individual OptionButton's equally sized and spaced apart.

When one of the OptionButtons is clicked, the **DataChanged** property on the parent "Buttons" form is set to True. This property automatically triggers an **UpdateDataSource** method that is executed when the RecordSet is updated. When this method executes, it basically sends the value of the "Buttons" form Value property which indicates which OptionButton is True.

### Populating the IndentedList with Department Numbers

The **EmployeeForm** contains a method named **PopulateList** that is responsible for reading the rows of data contained in the DataControl's RecordSet. This method basically scans the Department field of the RecordSet and for each department number it finds, it checks to see if the department number has already been added to the IndentedList. If it does not current exist, it is added to the list.

