

Visual Basic and Databases

4. Database Connection with ADO Technology

Review and Preview

- In the last chapter, we discussed the differences between the legacy Visual Basic data control, the DAO control, and the new control shipped with Visual Basic 6, the ADO control. And, we looked at how to use the DAO control to connect to a database and view desired recordsets.
- In this chapter, we cover much of the same material from the previous chapter, however, here we use ADO technology, examining both the data control and data environment. This chapter is self-contained, meaning it assumes you have not necessarily read the chapter on the DAO control. (For those who have read the previous chapter, you will notice lots of repetition). In addition to using the ADO data control for database access, we present a powerful new technology associated with the ADO data control, the data environment. Be aware you can only complete this particular chapter if you are using Visual Basic 6 - ADO technology is not available with Visual Basic 5.

ADO Data Control

- The **ADO** (ActiveX Data Object) **data control** is the primary interface between a Visual Basic application and a database. It can be used without writing any code at all! Or, it can be a central part of a complex database management system. The ADO data control does not appear in the standard Visual Basic toolbox - it must be added. Select **Project** from the main menu, then click **Components**. The Components window will appear. Select **Microsoft ADO Data Control**, then click **OK**. The control will be added to your toolbox. Its icon appears as:



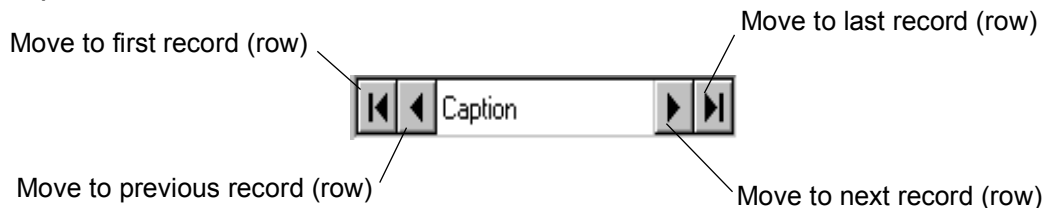
- The data control (or tool) can access databases created by other programs besides Visual Basic (or Microsoft Access). Some other formats supported include dBase, FoxPro, and Paradox.
- The data control can perform the following tasks:
 1. Connect to a database.
 2. Open a specified database table.
 3. Create a virtual table based on a database query.
 4. Pass database fields to other Visual Basic tools, for display or editing. Such tools are bound to the database, or **data bound controls**.
 5. Add new records, delete records, or update records.
 6. Trap any errors that may occur while accessing data.
 7. Close the database.
- As a rule, you need one data control for every database table, or virtual table, you need access to. One row of the table is accessible to each data control at any one time. This is referred to as the **current record**.

ADO Data Control Properties

- The ADO data control is connected to a database simply by setting a few properties. Important properties of this data control are:

Align	Determines where data control is displayed.
Caption	Phrase displayed on the data control.
CommandType	Establishes source of Recordset (table or query).
ConnectionString	Contains the information used to establish a connection to a database.
EditMode	Read-only at run-time. Indicates current state of editing for the current record.
LockType	Indicates the type of locks placed on records during editing (default setting makes databases read-only).
Recordset	A set of records defined by a data control's ConnectionString and RecordSource properties. Run-time only.
RecordSource	Determines the table (or virtual table) the data control is attached to.
Visible	Establishes whether the data control appears on the form at run-time.

- When an ADO data control is placed on a form, it appears with the assigned caption and four arrow buttons:



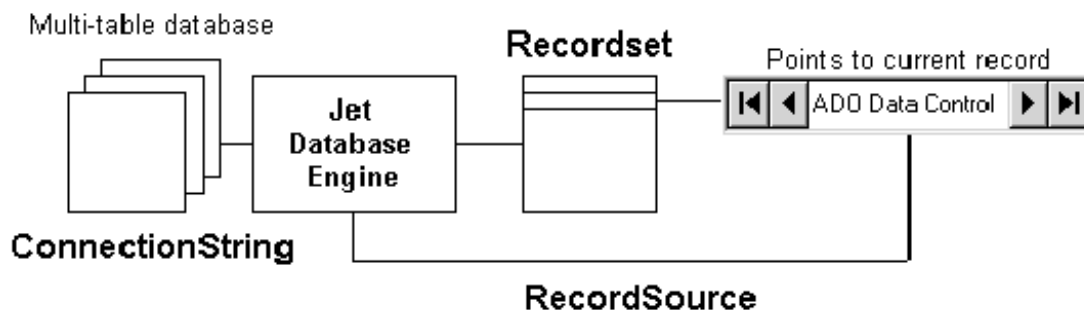
The arrows are used to navigate through the table records (rows). As indicated, the buttons can be used to move to the beginning of the table, the end of the table, or from record to record. In most applications, the data control never appears on the form – its **Visible** property is almost always **False**. In this case, moving from record to record is handled programmatically, a topic discussed later in this chapter.

ConnectionString Property

- After placing a data control on a form, you set the **ConnectionString** property. The ADO data control can connect to a variety of database types. There are three ways to connect to a database: using a data link, using an ODBC data source, or using a connection string. For now, we will look only at connection to a Microsoft Access database using a **connection string**.
- Visual Basic can build the connection string for us. This process is best illustrated by example. We will use the books database (**BIBLIO.MDB**) discussed in Chapter 2. This database is shipped with Visual Basic and is usually installed in the Visual Basic main directory. Using Windows Explorer, find this file. Make a copy of the database and place it in a working directory (you decide on a name – we use c:\vbdb\working) where you will build your applications. We do this to insure there is always a valid copy of BIBLIO.MDB on your computer. You will see that the power of the ADO control also opens up the possibility of doing damage to a database (we, of course, will try to minimize this possibility). So, we are just living by the adage, “Better safe, than sorry.”
- Now, the steps to create our example connection string are:
 1. Start a new Visual Basic project and place an ADO Data Control on the form.
 2. Go to the **Properties** Window, click on the **Connection String**. Click on the ellipsis that appears. The **Property Pages** window appears.
 3. Choose **Use Connection String** and click the **Build** button. The **Data Link Properties** window appears.
 4. Choose the **Provider** tab and select **Microsoft Jet 3.51 OLE DB Provider** (an Access database).
 5. Click the **Next** button to go to the **Connection** tab.
 6. Click the ellipsis and use the **Select Access Database** dialog box to choose the **BIBLIO.MDB** file in your working directory. Click **Open**.
 7. Click **Test Connection**. Then, click **OK** (assuming it passed). Click **OK** in the Property Pages window. The connection string is built and assigned to the **ConnectionString** property of the data control.

Recordset Object

- The **Recordset** object is an important concept. When we set the **RecordSource** property (either select a table from the database or form a virtual table via a query), the data control (using the Jet engine) retrieves the needed records and places them in the **Recordset** object for our use. We will see that this object has its own **properties** and **methods** for our use.
- In this chapter, the Recordset will be one of the native tables in the database. Continuing with the **BIBLIO.MDB** example, let's connect to the **Titles** table:
 1. Go to the **Properties** window for the data control. Select **2-adCmdTable** for the **CommandType** property for the data control (this tells the data control we will be using a native table). Now, click the **RecordSource** property. Click on the ellipsis that appears. The **Property Pages** window appears.
 2. Click the drop-down button under **Tables or Stored Procedure Name**. Choose **Titles**.
 3. Click **OK** in the Property Pages window. The **RecordSource** property of the data control is set, establishing the **Recordset** as the **Titles** table of the books database.
- In summary, the relationship between the **data control**, its two primary properties (**Connection String** and **RecordSource**), and the **Recordset** object is:



Data Bound Controls

- The ADO data control allows us to easily connect to a database and form a Recordset. Yet, that control alone does not provide us with any way to view the information in the database. To view the information, we use **data bound controls** that are special controls with properties established by database fields. A data bound control is needed for each field (column) in the Recordset (database table) you need to view. Most of the standard Visual Basic tools can be used as **data bound** controls.

- Standard data bound data controls are:

Label	Can be used to provide display-only access to a specified text data field. Caption property is data bound.
Text Box	Can be used to provide read/write access to a specified text data field. Probably, the most widely used data bound tool. Text property is data bound.
Check Box	Used to provide read/write access to a Boolean field. Value property is data bound.
Picture Box	Used to display a graphical image from a bitmap, icon, gif, jpeg, or metafile file. Provides read/write access to a image/binary data field. Picture property is data bound.
Image Box	Used to display a graphical image from a bitmap, icon, gif, jpeg, or metafile file (uses fewer resources than a picture box). Provides read/write access to a image/binary data field. Picture property is data bound.

- There are also three 'custom' data bound controls, data bound versions of the standard list box (**DataList**), the standard combo box (**DataCombo**), and the standard grid control (**DataGrid**). We will look at these later.

Data Bound Control Properties

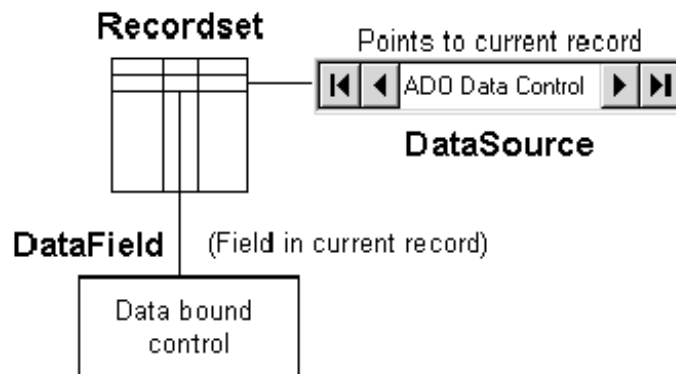
- To establish the connection of the data bound control to a database, we use a few properties:

DataChanged	Indicates whether a value displayed in a bound control has changed.
DataField	Specifies the name of a field in the table pointed to by the respective data control.
DataSource	Specifies which data control the control is bound to (indirectly specifying the database table).

- If the data in any data bound control is changed and the user moves to another record in the database, the database will **automatically** be **updated** with the new data (assuming it is not ReadOnly). Be aware of this - it is an extremely powerful feature of the data control, but also a potential source of problems.
- To make using bound controls easy, follow these steps (in order listed) in placing the controls on a form:
 1. Draw the bound control on the same form as the data control to which it will be bound.
 2. Set the **DataSource** property. Click on the drop-down arrow to list the data controls on your form. Choose one.
 3. Set the **DataField** property. Click on the drop-down arrow to list the fields associated with the selected data control records. Make your choice.
 4. Set all other properties, as needed.

Again, by following these steps in order, we avoid potential data access errors.

- The relationships between a data bound control (**DataSource** and **DataField** properties) and the ADO data control (**Recordset** property) are:



Example 4-1**Accessing the Books Database**

If you haven't made a working copy of the database file (**BIBLIO.MDB**) as explained earlier, please do so now. Note you may be able to skip a few steps in this example, if you've already set the **ConnectionString** and **RecordSource** properties to point to the Titles table.

1. After copying BIBLIO.MDB to your working directory, start a new application. We'll develop a form where we can look through the **Titles** table in the books database. Place an ADO data control, four label boxes, and four text boxes on the form.
2. Set the following properties for each control. For the data control and the four text boxes, make sure you set the properties in the order given.

Form1:

Name	frmTitles
BorderStyle	1-Fixed Single
Caption	Titles Database

Adodc1:

Name	datTitles
Caption	Titles
CommandType	2-adCmdTable
ConnectionString	Use the Build option to point to the BIBLIO.MDB database
RecordSource	Titles (select from list, don't type)

Label1:

Caption	Title
---------	-------

Label2:

Caption	Year Published
---------	----------------

Label3:

Caption	ISBN
---------	------

Label4:

Caption	Publisher ID
---------	--------------

Text1:

Name	txtTitle
DataSource	datTitles (select, don't type)
DataField	Title (select, don't type)
Locked	True
MultiLine	True
Text	[Blank]

Text2:

Name	txtYearPublished
DataSource	datTitles (select, don't type)
DataField	Year Published (select, don't type)
Locked	True
MultiLine	True
Text	[Blank]

Text3:

Name	txtISBN
DataSource	datTitles (select, don't type)
DataField	ISBN (select, don't type)
Locked	True
MultiLine	True
Text	[Blank]

Text4:

Name	txtPubID
DataSource	datTitles (select, don't type)
DataField	PubID (select, don't type)
Locked	True
MultiLine	True
Text	[Blank]

When done, the form will look something like this:

3. Save the application. Run the application. Cycle through the various titles using the data control. Did you notice something? You didn't have to write one line of Visual Basic code! This indicates the power behind the data control and data bound controls.

There's one last thing. If you load this example from the code accompanying the course, you will need to reset the data control's **ConnectionString** property, pointing to the directory in which you have stored the Northwind Traders database on your computer. In fact, you will have to do this anytime you use the examples provided with the course.

ADO Data Control Events

- Like other controls, the ADO data control has **events** that are triggered at various times during database access. In these events, we write BASIC code to perform specific needed tasks. In this chapter, we will not be using these event procedures, but we will define them to make our definition of the data control complete.

- Important ADO data control events:

WillMove	Triggers before record pointer is moved from one row to another.
MoveComplete	Event triggered after record pointer has been moved from one row to another.
EndofRecordset	Triggers when the record pointer is moved past the last record in the recordset.
WillChangeRecordset	Triggers before a change is made to the recordset. Used to trap unwanted changes.
RecordsetChangeComplete	Triggers after a change is made to recordset.
WillChangeRecord	Triggers before updates for the current row are sent to the data source.
RecordChangeComplete	Triggers after updates for the current row are sent to the data source.
WillChangeField	Triggers before the current field in the recordset is updated.
FieldChangeComplete	Triggers after the current field in the recordset has been updated.

- These events will be discussed further when we begin development of database management techniques in a later chapter.

ADO Data Control Method

- To complete our definition of the ADO data control, we present a single method:

Refresh	Requeries the database based on contents of the RecordSource property.
----------------	---

- Like events, this ADO data control method will be discussed further when we begin development of database management techniques in a later chapter.

ADO Data Control Recordset Properties

- The **Recordset** object of the data control has its own set of **properties**. These properties can only be accessed at **run-time**. To refer to a Recordset property, use a 'double-dot' notation. For example, if you have a data control named **datExample**, to refer to a property named **PropertyName**, the notation is:

datExample.Recordset.PropertyName

- Important data control Recordset properties are:

AbsolutePosition	Long integer that either gets or sets the position of the current record.
BOF	Returns True when the current record is positioned before any data.
Bookmark	Sets or returns a bookmark to the current record. Used as a place marker in database management tasks.
EditMode	Indicates the state of editing for the current record.
EOF	Returns True when the current record is positioned past any data.
RecordCount	The total number of records in the Recordset.

- We will look at the BOF and EOF properties in the section on Recordset Navigation. Other properties will be examined later in this course.

ADO Data Control Recordset Methods

- The data control **Recordset** also has its own set of **methods** that perform functions on the Recordset. These methods are invoked using the double-dot notation introduced for the Recordset properties. So, for a data control (**datExample**) and method (**MethodName**), you invoke the method via:

datExample.Recordset.MethodName

- Important Recordset methods are:

AddNew	Adds a new record to the Recordset. All fields are set to null and this record becomes the current record.
CancelUpdate	Used to cancel any pending updates (either while editing or using the AddNew method)
Close	Closes a Recordset.
Delete	The current record is deleted from the Recordset.
Move	Moves the current record pointer a specified number of records forward or backward.
MoveFirst	Moves the current record pointer to the first record in the Recordset.
MoveLast	Moves the current record pointer to the last record in the Recordset.
MoveNext	Moves the current record pointer to the next record in the Recordset.
MovePrevious	Moves the current record pointer to the previous record in the Recordset.
Requery	Updates the data in a Recordset object by re-executing the query on which the object is based.
Update	Saves the current contents of all data bound controls.

- We will look at the four 'Move' methods in the next section on Recordset Navigation. Other properties will be reviewed later in this course.

ADO Data Control Recordset Navigation

- We have seen that, on the form, the ADO data control has four arrows that allow the user to move to the first, next, previous, and last records in the Recordset. Unfortunately, this control does not have a familiar look to a user and it may not be clear just exactly what functions the arrows perform. For this reason, we usually set the data control's **Visible** property to **False** and provide a programmatic approach to moving from record to record, or **Recordset navigation**.
- Four Recordset methods replicate the capabilities of the arrow buttons on the data control: **MoveFirst**, **MoveNext**, **MovePrevious**, and **MoveLast**. For each function we need, a command button is added to the form, with a **Click** event procedure attached to the corresponding Recordset method.
- When programmatically navigating through the records, you need to be aware of the position of the current record. For example, if you are at the first record and try a **MovePrevious** method, you will move past the beginning of the file. You can use the **BOF** property to see you are at the beginning of file and disallow such a move. You need a similar check at the end of a file to disallow an invalid **MoveNext** method.

Quick Example 1 - Recordset Navigation

1. Load the project built in Example 4-1. Set the ADO data control's **Visible** property to **False**. Add two command buttons with the following properties:

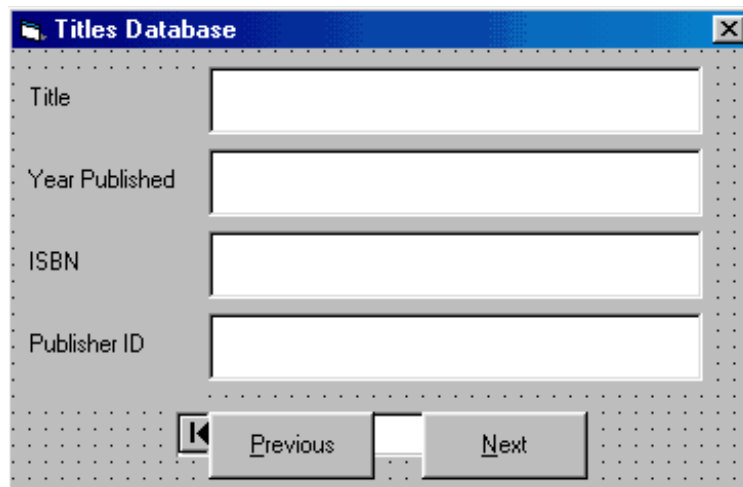
Command1:

Name	cmdPrevious
Caption	&Previous

Command2:

Name	cmdNext
Caption	&Next

The form should look like this (notice the data control is still there under the command buttons- it will only disappear at run-time).

The screenshot shows a Windows-style application window titled "Titles Database". Inside the window, there are four text input fields stacked vertically, each with a label to its left: "Title", "Year Published", "ISBN", and "Publisher ID". Below these fields, there are two buttons. The left button is labeled "Previous" and has a small icon to its left. The right button is labeled "Next". The buttons are positioned such that they appear to be partially overlapping or very close together. The background of the form has a dotted pattern.

2. Place this code in the **cmdPrevious_Click** event:

```
Private Sub cmdPrevious_Click()  
    datTitles.Recordset.MovePrevious  
    If datTitles.Recordset.BOF Then  
        datTitles.Recordset.MoveFirst  
    End If  
End Sub
```

3. Place this code in the **cmdNext_Click** event:

```
Private Sub cmdNext_Click()  
    datTitles.Recordset.MoveNext  
    If datTitles.Recordset.EOF Then  
        datTitles.Recordset.MoveLast  
    End If  
End Sub
```

4. Save and run the application. Make sure the newly added buttons work as they should. Try adding buttons to move to the first and last records. Can you write the code?

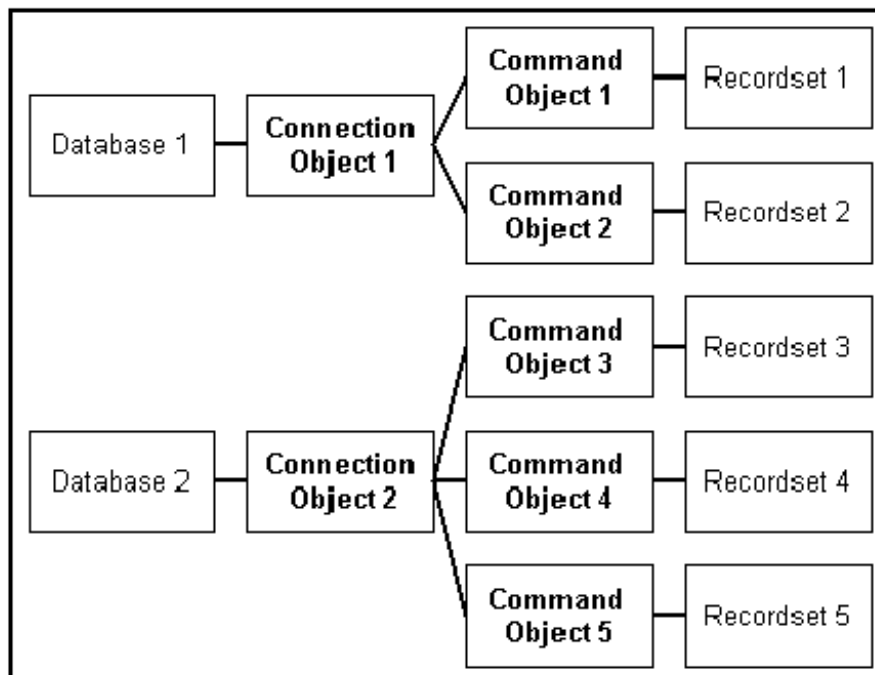
DAO or ADO – What's the Difference?

- We asked this same question in Chapter 3 before we looked at both data controls. Now that we've seen how the two controls are used for database connection, it's fair to ask the question again.
- Use of the two controls is nearly identical – you can see that in the fact that the two sets of notes are nearly identical. The primary difference between the two data controls is in the properties used to connect to a database. The **DAO** control connects to the database using the **DatabaseName** property while the ADO control uses the **ConnectionString**.
- To construct a Recordset, the **DAO** control uses the **RecordSource** property. The **ADO** control requires setting two properties: **RecordSource** and **CommandType**.
- The ADO data control offers more event procedures than the DAO control to allow more complete control over database management.
- So, the question still may be – which control should you use? You should understand the use of both data controls because you will see them both as you progress as a Visual Basic programmer. For simple projects, the DAO control is adequate. For more detailed projects and for all new projects, we would recommend the ADO data control. It is new technology and will receive the bulk of Microsoft's support with future releases of Visual Basic. And, as seen in the next section, it has some hidden powers we have yet to see!

ADO Data Environment

- One big advantage of the new ADO data control is that you don't even need it to work with databases! Yes, that's right. A new technology has emerged – the **ADO Data Environment**. The Data Environment acts like multiple data controls you can access from anywhere in your application. You can connect to multiple databases and form multiple views of the data in those databases. Data bound controls can bind to any data view in the Data Environment.
- The Data Environment is a shareable and reusable connection file that can be used in any Visual Basic project. It is added to your project just like a form (it has a single property of interest – **Name**). The Data Environment provides a greatly simplified programming environment used to connect to data sources. It provides 'drag-and-drop' functionality for building interfaces and developing database reports (discussed in a later class).
- The Data Environment has two primary objects: the **Connection** object (specifies the database similar to the **ConnectionString** property of the ADO data control) and the **Command** object (specifies the Recordset like the **RecordSource** property of the ADO control).
- To see the versatility of the Data Environment, here is a diagram of one with five different recordsets being generated from two different databases:

Data Environment



Connection Object

- The primary feature of the Data Environment is the **Connection** object. It specifies the information needed to connect to a particular database.
- The Connection object has several properties:

Attributes	Needed connection string properties passed to the Connection object.
CommandTimeout	Time in seconds the server will wait for a command to return a reply.
ConnectionString	String that describes the path to the data source connection.
ConnectionTimeout	Time in seconds the server will wait for a connection to open on the destination server.

Don't worry if some of these properties don't make sense right now.

Command Object

- The Data Environment Connection object specifies a database. The **Command** object specifies a database table (either a native table or a virtual table formed using a database query).
- The Command object is very flexible and can be used for very advanced database applications. For now, we will use it to form a Recordset for our use. Some Command object properties are:

CommandText	Specifies table to display or gives valid database query statement.
CommandType	Specifies whether object is connected to database native table or virtual table formed using a query.
ConnectionName	Name of associated Connection object.
LockType	Controls how data in table may or may not be changed. Set to Read Only by default. Must be changed if you need to edit data.

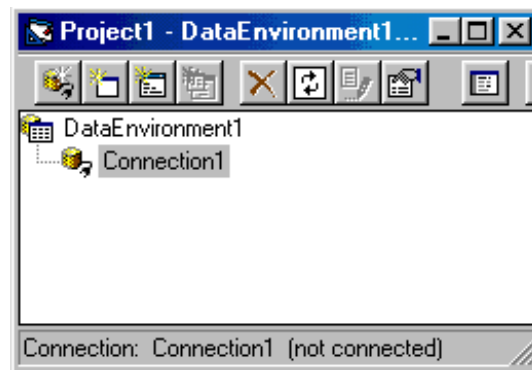
- Like most things with Visual Basic, the Data Environment and associated Connection and Command objects are best illustrated via example.

Quick Example 2 - Use of ADO Data Environment

In this example, we will form a **Connection** object to the books database (**BIBLIO.MDB**) database and a **Command** object for the **Titles** table. This is the same table used in the example with the ADO data control. This should illustrate the differences between using the Data Environment and the ADO data control.

1. Start a new project. Add a **Data Environment** to your project. To do this, either right-click the Project Explorer window and choose **Add**, then **Data Environment**. Or, choose **Add Data Environment** under the **Project** menu item. There's a chance that **Add Data Environment** may not be one of the menu choices. If this is the case, the problem is easy to solve. Select **Components** under the **Project** menu item. In the window that appears, choose the Designers tab and check the boxes next to **Data Environment** and **Data Report**, then click **OK**. The Data Environment will now be available.

Once selected, a Data Environment will appear in the Project Explorer window. The Data Environment window should also appear. If it doesn't, double-click the Data Environment listing in the Project Explorer window. It should look like this:



2. Go to the **Properties** window and assign a **Name** of **denBooks** to DataEnvironment1. Assign a **Name** of **conBooks** to Connection1.

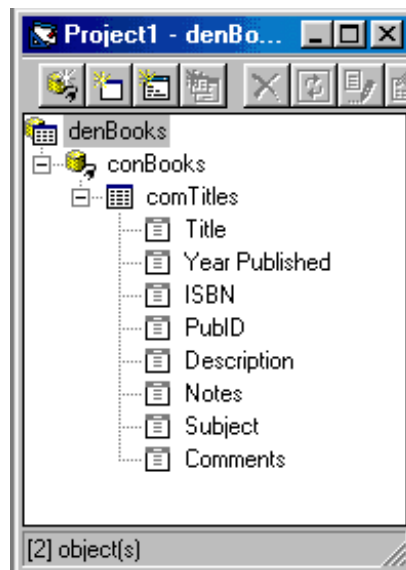
3. We'll now establish the **Connection** object:

- ⇒ Right-click on **conBooks** and select **Properties**. A **Data Link Properties** window (identical to that used to set the `ConnectionString` with the data control) will appear.
- ⇒ Choose the **Provider** tab and select **Microsoft Jet 3.51 OLE DB Provider** (an Access database).
- ⇒ Click the **Next** button to go to the **Connection** tab. Click the ellipsis and use the **Select Access Database** dialog box to choose the **BIBLIO.MDB** file in your working directory. Click **Open**. Click **Test Connection**. Then, click **OK** (assuming it passed).
- ⇒ Click **OK** in the **Data Link Properties** window.

4. And now, we will establish the **Command** object:

- ⇒ In the Data Environment window, right-click on **conBooks**. A pop-up menu will appear. Select **Add Command**. A Command object is added to the environment.
- ⇒ Right-click **Command1** and select **Properties**. A Properties window will appear. Make sure the **General** tab is active.
- ⇒ Give the Command object a name of **comTitles**. Make sure the **Connection** is **conBooks** (the Connection object you just created).
- ⇒ In the **Source of Data**, choose **Database Object**, then select **Table**. Finally, under **Object Name**, choose the **Titles** table from the drop-down list.

5. The Command object is complete. Right-click **Titles** and choose the **Expand All** option. All fields in the Titles table will be listed and the Data Environment window should look like this:



The Data Environment is now configured to allow access to the **Titles** table in the books database (**BIBLIO.MDB**) database. This configuration may have required more steps than simply using the ADO data control, but the steps are worth it. You'll see that now where we attach some data bound controls to view the Titles table.

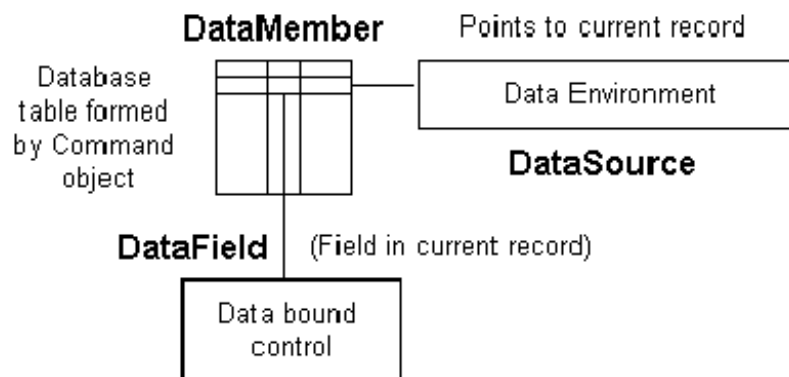
Data Bound Controls with the Data Environment

- To establish the connection of a data bound control to a database using the Data Environment, we use three properties:

DataField	Specifies the name of a field in the table established by the Command object.
DataMember	Specifies the Command object establishing the database table.
DataSource	Specifies which Data Environment the control is bound to.

Note there is one more property needed when compared to using the ADO data control - **DataMember** is not used with the data control.

- The relationships between a data bound control (**DataSource**, **DataMember** and **DataField** properties) and the Data Environment are:



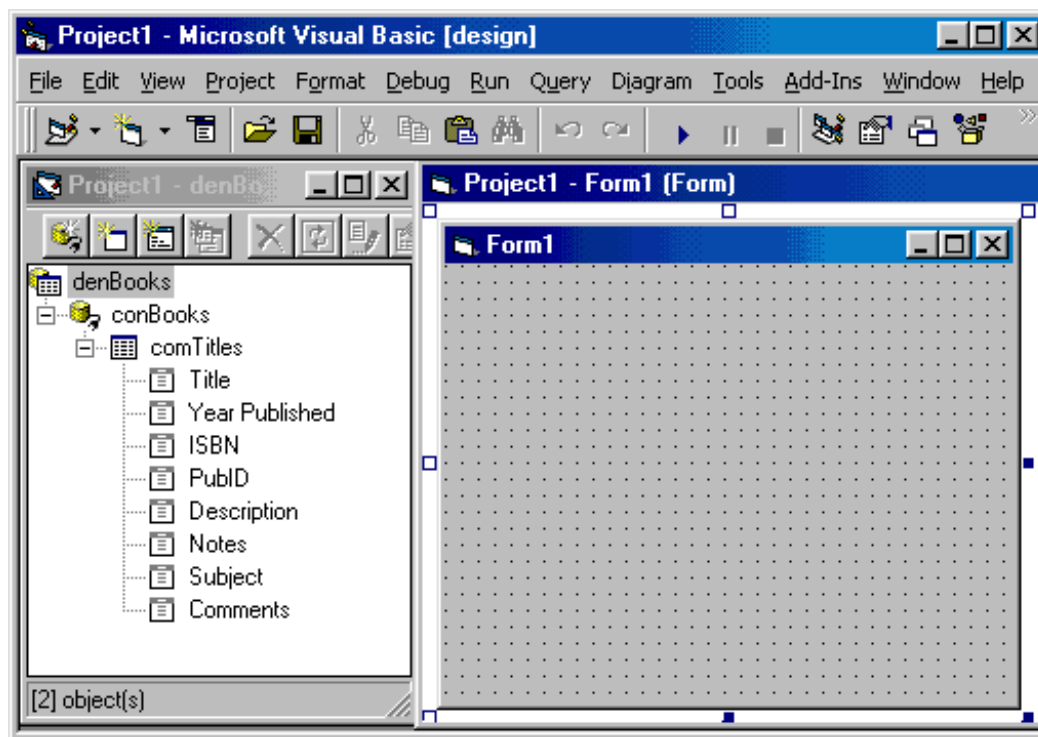
- To use bound controls with the Data Environment, we could follow these steps (in order listed) in placing the controls on a form:
 1. Draw the bound control on the a form.
 2. Set the **DataSource** property. Click on the drop-down arrow to list the Data Environments in your project. Choose one.
 3. Set the **DataMember** property. Click on the drop-down arrow to list the Command objects in the selected Data Environment. Choose one.
 4. Set the **DataField** property. Click on the drop-down arrow to list the fields associated with the selected DataMember. Make your choice.
 5. Set all other properties, as needed.
- The steps above are just one way to connect data bound controls to the Data Environment. But, now let's look at one of the powerful features of the Data Environment - '**drag and drop**' data bound controls.

Example 4-2

Drag and Drop Controls

In this example, we will build the same interface used with the ADO data control in **Example 4-1** using the drag and drop capabilities of the Data Environment.

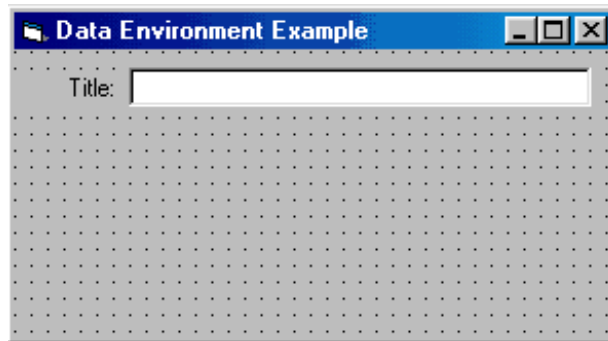
1. Return to the previous example (**Quick Example 2**) where we set up the Data Environment for the Titles table of the books database. Set up the Visual Basic environment so the Data Environment window and the Project window with an empty Form appears on the screen. Your screen should look something like this:



2. Set the following properties for the form:

Name	frmDataEnvironment
BorderStyle	1 - Fixed Single
Caption	Data Environment Example

- Left-click on the **Title** field in the **Data Environment** window and drag it to the **Form**. When the field is over the Form an icon will appear. This icon holds a label identifying the field and a text box for displaying the field. Release the mouse button when the icon is in the desired position. At this point, my form looks like this:



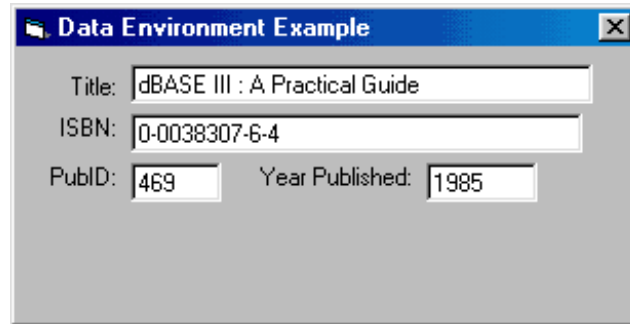
- Go to the Properties window and look at the **Name**, **DataField**, **DataMember**, and **DataSource** properties for the text box on the form:

Name	txtTitle
DataField	Title
DataMember	comTitles
DataSource	denBooks

It's magic! By dragging the Title field onto the form, the data bound control properties are automatically established by the Data Environment. This makes building an interface much easier. At this point, of course, you can change any properties regarding size and appearance.

- Complete the interface by dragging the **Year Published**, **ISBN**, and **PubID** fields onto the form. Move and resize any controls, as needed.

6. Save and run the application. Your screen should look something like this:



The screenshot shows a Windows-style application window titled "Data Environment Example". Inside the window, there are four text input fields arranged in a form. The first field is labeled "Title:" and contains the text "dBASE III : A Practical Guide". The second field is labeled "ISBN:" and contains "0-0038307-6-4". The third field is labeled "PubID:" and contains "469". The fourth field is labeled "Year Published:" and contains "1985". The window has a standard Windows 95/98 style title bar with a close button (X) in the top right corner.

Note we have a problem. We can only view the first record in the database. The Data Environment provides no means for navigating between records. But, recall, we have looked at how to programmatically navigate between records when discussing the ADO data control. Those same methods can be used here.

Recordsets in the Data Environment

- As configured, the Data Environment returns a Recordset that has properties and methods identical to those of the ADO data control. A Recordset is returned for each Command object in the environment. Since there are many possible recordsets in one Data Environment, each is assigned a unique name by the environment, based on the Command object name. Make sure you give each Command object in a Data Environment a unique name.
- For a **Command** object named **comExample**, the Recordset is assigned the name **rscomExample** (note the addition of the two letter prefix, rs). To read or set the property of this Recordset in a **Data Environment** named **denExample**, we use the double-dot notation of:

denExample.rscomExample.PropertyName

- Likewise to reference a method (such as one of the Move methods) for this Recordset, we use the notation:

denExample.rscomExample.MethodName

Quick Example 3 - Record Navigation with the Data Environment

1. Continue with Example 4-2 and add two command buttons with the following properties:

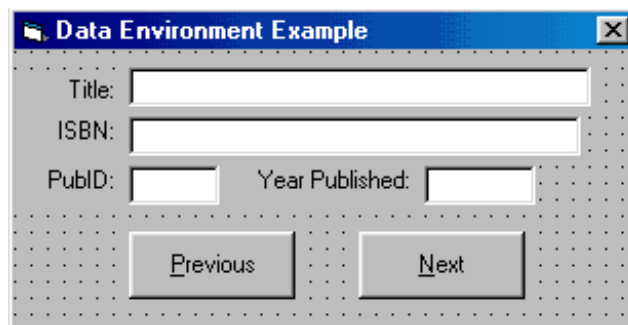
Command1:

Name	cmdPrevious
Caption	&Previous

Command2:

Name	cmdNext
Caption	&Next

2. The form should look like this:



The screenshot shows a Windows-style window titled "Data Environment Example". Inside the window, there is a grid background. At the top, there is a "Title:" label followed by a text box. Below that is an "ISBN:" label followed by a text box. Then, there are two labels, "PubID:" and "Year Published:", each followed by a text box. At the bottom of the form, there are two command buttons. The left button is labeled "Previous" and the right button is labeled "Next".

3. Place this code in the **cmdPrevious_Click** event:

```
Private Sub cmdPrevious_Click()  
denBooks.rscomTitles.MovePrevious  
If denBooks.rscomTitles.BOF Then  
    denBooks.rscomTitles.MoveFirst  
End If  
End Sub
```

4. Place this code in the **cmdNext_Click** event:

```
Private Sub cmdNext_Click()  
denBooks.rscomTitles.MoveNext  
If denBooks.rscomTitles.EOF Then  
    denBooks.rscomTitles.MoveLast  
End If  
End Sub
```

5. Save and run the application. Make sure the newly added buttons work as they should. Do you see how much easier it was to construct the form with the data bound controls, as compared to setting all the properties back in Example 4-1?

DAO to ADO – ADO to DAO

- What if you have a database application using the DAO data control that you would like to update to the ADO data control or ADO data environment? Or, what if you need to adapt a Visual Basic 6 application using ADO to Visual Basic 5 and DAO? In this section, we offer a few tips for the conversions.
- Replacing one data control with another is simple if you pay attention to just a couple of details. To switch from the **DAO control** to the **ADO control**, follow these steps:
 - ⇒ Make note of the DAO control **Name**, **DatabaseName**, and **RecordSource** properties.
 - ⇒ Delete the DAO data control.
 - ⇒ Add the ADO data control to the form.
 - ⇒ Assign the new control the same **Name** property, insure the **ConnectionString** is attached to the database specified by the DAO **DatabaseName** and use the same **RecordSource** property.

By following these steps, all data bound controls should still be properly connected. If not, it is a simple matter of re-establishing the **DataSource** and **DataField** properties for each control.

- Switching from the **ADO control** to the **DAO control** follows similar steps:
 - ⇒ Make note of the ADO control **Name**, **ConnectionString**, and **RecordSource** properties.
 - ⇒ Delete the ADO data control.
 - ⇒ Add the DAO data control to the form.
 - ⇒ Assign the new control the same **Name** property, set **DatabaseName** to the database referenced in the ADO **ConnectionString** property, and use the same **RecordSource** property.

By following these steps, all data bound controls should still be properly connected. If not, it is a simple matter of re-establishing the **DataSource** and **DataField** properties for each control.

- Updating from the **DAO data control** to the **ADO data environment** requires as a minimum:
 - ⇒ Make note of the DAO control **DatabaseName** and **RecordSource** properties.
 - ⇒ Delete the DAO data control.
 - ⇒ Add an ADO data environment to the project.
 - ⇒ Configure the **Connection** object so it is attached to the database given by DAO **DatabaseName**.
 - ⇒ Configure a **Command** object such that it's **Source of Data** is the same as the DAO **RecordSource** property.
 - ⇒ Modify data bound control properties – **DataSource** is bound to **data environment**, **DataMember** is bound to **Command** object, **DataField** is bound to appropriate field (should be the same value used by DAO control).
- Moving from the **ADO data environment** to the **DAO data control** requires as a minimum:
 - ⇒ Make note of the database the ADO **Connection** object is connected to and the **Source of Data** for the **Command** object.
 - ⇒ Delete the ADO data environment.
 - ⇒ Add a DAO data control to the form.
 - ⇒ Set the **DatabaseName** property such that it is connected to the same database and set the **RecordSource** property to the same value used by the ADO **Command** object.
 - ⇒ Modify data bound control properties – **DataSource** is bound to the DAO data control, **DataMember** is not used (blank it out), **DataField** is bound to appropriate field (should be the same value used by ADO data environment).
- All of the above conversions only address **design mode** setup. If you have written any BASIC code that uses data control methods and properties or recordset methods and properties, you will have to make sure necessary modifications are made to insure your code works with the new data access technology.

Summary

- In this chapter, we used the ADO data control to connect to and view a database using data bound controls. The procedure was seen to be nearly identical to that of the DAO data control studied in the previous chapter.
- A new technology, the **Data Environment**, was also studied. The Data Environment is a shareable and reusable connection file that can be used in any Visual Basic project. It is added to your project just like a form. The Data Environment provides a greatly simplified programming environment used to connect to data sources. It provides 'drag-and-drop' functionality for building interfaces.
- The ADO data control is still useful for quick prototyping of database applications and we will occasionally use it for this purpose. But, for any serious application, we will forego the ADO data control in favor of the ADO Data Environment.
- At this point in the course, we have learned a lot, but still can only view a native table in a database. In the next chapter, we learn the language that allows us to form any virtual view of data we wish. That language is **SQL** (structured query language), the heart of any database management system.

Exercise 4-1

Northwind Traders Database

A second sample database is included with Visual Basic 5 and Visual Basic 6. It is a database (**NWIND.MDB**) used by a fictional company (**Northwind Traders**) to handle its commerce. It has eight tables. In this exercise, we repeat the tasks of Example 4-1, using one table (**Customers**) in this database. This and the next exercise give you further practice in using the ADO data control and data bound controls and allows you to study the structure of another database.

1. Copy NWIND.MDB to your working directory and start a new application. We'll develop a form where we can look through the **Customers** table in the Northwind Traders database. Place an ADO data control, four label boxes, and four text boxes on the form.
2. Set the following properties for each control. For the data control and the four text boxes, make sure you set the properties in the order given.

Form1:

Name	frmCustomers
BorderStyle	1-Fixed Single
Caption	Customers Database

Adodc1:

Name	datCustomers
Caption	Customers
CommandType	2-adCmdTable
ConnectionString	Use the Build option to point to the NWIND.MDB database
RecordSource	Customers (select from list, don't type)

Label1:

Caption	Customer ID
---------	-------------

Label2:

Caption	Company Name
---------	--------------

Label3:

Caption	Contact Name
---------	--------------

Label4:

Caption	Contact Title
---------	---------------

Text1:

Name	txtCustomerID
DataSource	datCustomers (select, don't type)
DataField	CustomerID (select, don't type)
Locked	True
MultiLine	True
Text	[Blank]

Text2:

Name	txtCompanyName
DataSource	datCustomers (select, don't type)
DataField	CompanyName (select, don't type)
Locked	True
MultiLine	True
Text	[Blank]

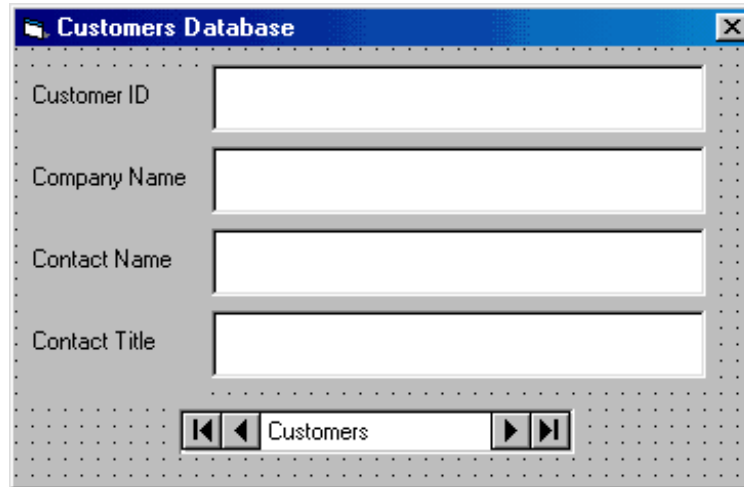
Text3:

Name	txtContactName
DataSource	datCustomers (select, don't type)
DataField	ContactName (select, don't type)
Locked	True
MultiLine	True
Text	[Blank]

Text4:

Name	txtContactID
DataSource	datCustomers (select, don't type)
DataField	ContactTitle (select, don't type)
Locked	True
MultiLine	True
Text	[Blank]

When done, the form will look something like this:



The screenshot shows a Windows-style application window titled "Customers Database". Inside the window, there are four labeled text input fields stacked vertically: "Customer ID", "Company Name", "Contact Name", and "Contact Title". At the bottom of the window, there is a data control consisting of a list box containing the word "Customers", flanked by navigation buttons: a double left arrow, a single left arrow, a single right arrow, and a double right arrow.

3. Save the application. Run the application. Cycle through the various customers using the data control.

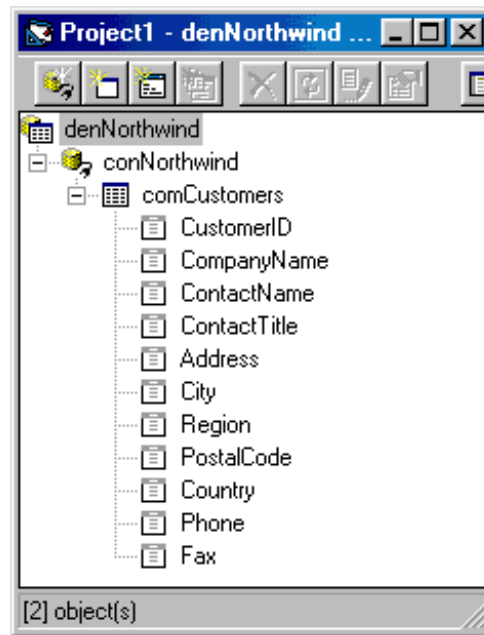
Exercise 4-2

Data Environment with Northwind Traders Database

For more practice in connecting to databases, we repeat Example 4-3 and Quick Example 4, using the Northwind Traders database. In this exercise, we will first form a **Connection** object to the Northwind Traders (**NWIND.MDB**) database and a **Command** object for the **Customers** table.

1. Start a new project. Add a **Data Environment** to your project. Go to the **Properties** window and assign a **Name** of **denNorthwind** to **DataEnvironment1**. Assign a **Name** of **conNorthwind** to **Connection1**. We'll now establish the **Connection** object:
 - ⇒ Right-click on **conNorthwind** and select **Properties**. A **Data Link Properties** window (identical to that used to set the **ConnectionString** with the data control) will appear.
 - ⇒ Choose the **Provider** tab and select **Microsoft Jet 3.51 OLE DB Provider** (an Access database).
 - ⇒ Click the **Next** button to go to the **Connection** tab. Click the ellipsis and use the **Select Access Database** dialog box to choose the **NWIND.MDB** file in your working directory. Click **Open**. Click **Test Connection**. Then, click **OK** (assuming it passed).
 - ⇒ Click **OK** in the **Data Link Properties** window.
2. And now, we will establish the **Command** object: In the Data Environment window, right-click on **comNorthwind**. A pop-up menu will appear. Select **Add Command**. A Command object is added to the environment.
 - ⇒ Right-click **Command1** and select **Properties**. A Properties window will appear. Make sure the **General** tab is active.
 - ⇒ Give the Command object a name of **Customers**. Make sure the **Connection** is **conNorthwind** (the Connection object you just created).
 - ⇒ In the **Source of Data**, choose **Database Object**, then select **Table**. Finally, under **Object Name**, choose the **Customers** table from the drop-down list.

The Command object is complete. Right-click **Customers** and choose the **Expand All** option. All fields in the Customers table will be listed and the Data Environment window should look like this:



3. Set the following properties for the form:

Name	frmDataEnvironment
BorderStyle	1 - Fixed Single
Caption	Data Environment Example

4. Drag and drop the following fields from the **Data Environment** window to the **Form: CustomerID, CompanyName, ContactName, and ContactTitle**. Add two command buttons with properties:

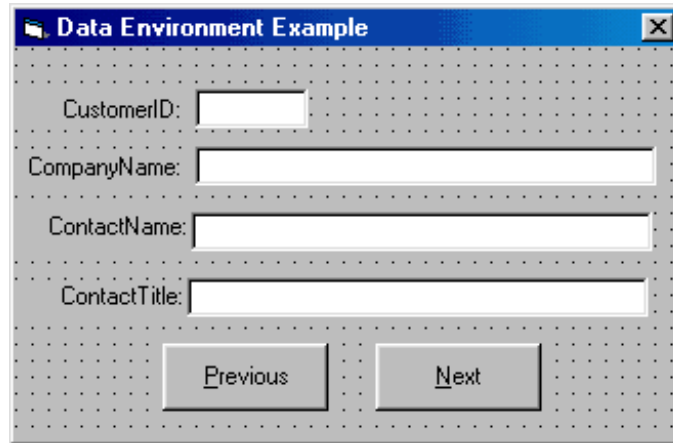
Command1:

Name	cmdPrevious
Caption	&Previous

Command2:

Name	cmdNext
Caption	&Next

The form should look like this:



The screenshot shows a Visual Basic form titled "Data Environment Example". The form has a dotted background. It contains four text boxes with labels: "CustomerID:", "CompanyName:", "ContactName:", and "ContactTitle:". At the bottom of the form, there are two buttons labeled "Previous" and "Next".

5. Place this code in the **cmdPrevious_Click** event:

```
Private Sub cmdPrevious_Click()  
denNorthwind.rscomCustomers.MovePrevious  
If denNorthwind.rscomCustomers.BOF Then  
    denNorthwind.rscomCustomers.MoveFirst  
End If  
End Sub
```

6. Place this code in the **cmdNext_Click** event:

```
Private Sub cmdNext_Click()  
denNorthwind.rscomCustomers.MoveNext  
If denNorthwind.rscomCustomers.EOF Then  
    denNorthwind.rscomCustomers.MoveLast  
End If  
End Sub
```

7. Save and run the application.

This page intentionally not left blank.