CHAPTER 6

ActiveSync

IN CHAPTER 1, YOU DISCOVERED how to set up ActiveSync and you learned that it provides a conduit between your Pocket PC and your desktop computer. This conduit enables you to run your eMbedded Visual Basic applications on your Pocket PC and it allows you to synchronize things such as your Outlook e-mail, Web pages, files, and databases. Because this is a database book, I'm only going to focus on how ActiveSync enables you to keep desktop and handheld databases in sync with each other.

ActiveSync is a powerful feature with obvious benefits to business users of the Pocket PC. Many companies have mobile workforces that need to have vital corporate data available to them when they're out of the office or on the road. They need to be able to synchronize their Pocket PCs with desktop and corporate databases first thing in the morning so that they have the latest company information at their fingertips while out in the field. Likewise, new data that's captured on the Pocket PC by mobile workers needs to get back to corporate databases so that decisions can be made based on the new information. Professions that would find this technology useful include the following:

- **Real estate agents:** Every morning, a real estate agent can sync up with the MLS database to ensure that he or she has the latest information and photos of all the homes for sale in the area. Having the answers to potential homebuyer questions on the Pocket PC can enhance the agent's responsiveness and increase customer satisfaction.

- **Sales force personnel:** Offline client information on the Pocket PC combined with corporate database synchronization makes for a powerful sales force automation tool. New leads discovered in the field can turn into sales calls when data is replicated back to the office through the Internet or the USB cradle.

- **Doctors:** Rather than filling out patient charts and putting up with mounds of paperwork, doctors can now enter patient data on the Pocket PC and merge that data with hospital patient databases at the end of their shift. Pocket PC handwriting recognition can turn a doctor's illegible scribbling into accurate drug prescriptions.

- **Insurance agents:** When an insurance agent visits a client at his or her home, the agent can dispense with paper forms and enter insurance application information directly into the Pocket PC. The synching of new policy data can happen immediately using remote access or it can happen when the agent gets back to the office.

The list of uses for Pocket PC data that can replicate with corporate databases can go on and on. Let's stop talking about what can be done and start learning how to do it. The first thing I'll do is bore you with the details of data conversion between the desktop and the device so that you can make better decisions about the data types you choose to use. The next thing you're going to do is build an Access database on the desktop and then walk through the necessary steps to get that database on your Pocket PC and converted into Pocket Access format. You'll then build a full-featured DML Pocket Access database manager so you can add, delete, and update synchronized data to your heart's content. Finally, you'll play ping-pong with your data with lots of manipulation thrown in to prove that ActiveSync really works.

Data Conversion Issues

The movement of data between databases on your Pocket PC and your desktop is far from seamless. ActiveSync has its work cut out for it when trying to maintain the integrity of your data during the synchronization process.

Desktop to Device

Starting out on the desktop, ActiveSync can work with Microsoft Access or any ODBC-compliant database. It then has the unenviable task of converting your perfectly good desktop or server database into a Pocket Access database. With its small footprint, Pocket Access doesn't support the enterprise features or the range of data types that SQL Server or Oracle does. ActiveSync is forced to map data types, which may result in the loss of data if the data types don't match up well. Table 6-1 displays Access and ODBC data types and the Pocket Access types they map to.

Table 6-1. Desktop to Device Type Mappings

ACCESS DATA TYPE	ODBC DATA TYPE	POCKET ACCESS DATA TYPE
Text	sql_varchar	Varchar
Memo	sql_longvarchar	Text
LongInt	sql_integer	Integer
	sql_bigint	Integer
Byte	sql_tinyint	Smallint
Int	sql_smallint	Smallint
Single	sql_real	Double
Double	sql_double	Double
	sql_float	Double
ReplID	sql_varbinary	Varbinary
Date/Time	sql_timestamp	Datetime
Currency	sql_numeric	Double
AutoNumber	sql_integer	Integer
YesNo	sql_bit	Boolean
OleObject	sql_longvarbinary	Varbinary
HyperLink	sql_longvarchar	Text
Lookup	sql_varchar	Varchar

Table Issues

- A table will not be converted or copied to your Pocket PC if all its fields use unsupported data types.
- System tables will not be converted or copied to your Pocket PC.
- Table names longer than 31 characters will be truncated.
- If a table with a truncated name exists and you've chosen not to overwrite tables, the last character of that table name will be deleted and replaced with the number 0. If a truncated table name already has a 0, numbers 1 through 9 will be tried. If truncated tables exist with all ten numbers, the table won't be copied.

Field Issues

- Field names longer than 64 characters will be truncated.

Index Issues

- Index names longer than 64 characters will be truncated.
- Only Ascending and Descending index attributes will be copied. All other index attributes will be omitted.
- Only three indexes are allowed in a database. Indexes beyond that number will be skipped.
- Index names are not case sensitive.
- Only the first field of a multicolumn index will be indexed in Pocket Access.
- Pocket Access indexes are created in three stages during the conversion process.

Stage 1: Unique Primary Key Indexes

Any unique index named "PrimaryKey" will be processed first. If this is a single-field index, PrimaryKey will be created in Pocket Access. If PrimaryKey is made up of multiple fields, an index will be created for each field that exists in Pocket Access. If need be, these fields will have 0, 1, or 2 appended to their names. A Boolean cannot be a unique PrimaryKey index.

Stage 2: Unique Indexes

Unique indexes are created in Pocket Access after PrimaryKey indexes. If any of the following statements are true, a unique index will not be created.

- A 64-character, truncated index name matches an index name that is already present in Pocket Access.
- The particular field already has an index in Pocket Access.
- The indexed field is a Boolean data type.

Stage 3: Nonunique Indexes

Nonunique indexes are created after unique indexes. If any of the following statements are true, a nonunique index will not be created.

- The index contains more than one field.
- The indexed field is not selected to be copied to Pocket Access.
- A 64-character, truncated index name matches an index name that is already present in Pocket Access.
- The particular field already has an index in Pocket Access.
- The indexed field is a Boolean data type.

Device to Desktop

When moving from a Pocket Access database to a desktop Microsoft Access or ODBC database, things don't look as bad because you don't have to funnel countless desktop data types into a handful of device data types. The result is a reduced chance of data loss. Table 6–2 lists all the Pocket Access data types as well as the ODBC and Access data types that they map to.

Table 6-2. Device to Desktop Type Mappings

POCKET ACCESS DATA TYPE	ODBC DATA TYPE	ACCESS DATA TYPE
Datetime	sql_timestamp	Date/Time
Double	sql_double	Double
Integer	sql_integer	LongInt
Smallint	sql_smallint	int
Boolean	sql_bit	YesNo
Varbinary	sql_varbinary	Binary
Long Varbinary	sql_longvarbinary	OLEObject
Varchar	sql_varchar	Text
Text	sql_longvarchar	Memo

Table Issues

- A table will not be converted or copied to your desktop PC if the table is a system table.
- A table will not be converted or copied to your desktop PC if it doesn't have an entry in the MSysTables table.

Index Issues

- If the Pocket Access index has a name beginning with the text PrimaryKey, a unique index will be created on the desktop.
- Only Ascending and Descending index attributes will be copied. All other index attributes will be omitted.

Conversion Error Log

When ActiveSync is converting data between your desktop and your Pocket PC, both informational and fatal errors may occur. An informational error occurs when the structure of the data has to be altered as a result of things such as a truncated field or table name. A fatal error occurs when the data can no longer be copied to the desktop or Pocket PC as a result of a break in the communications link or some other anomaly. When either of these kinds of errors occurs, a log file named "Db2ce.txt" is generated in the device partner directory on the desktop PC. The information contained in the error log is displayed in Table 6-3.

Table 6-3. Conversion Error Log

SECTION	DESCRIPTION
Startup Statistics	Displays the user name, conversion start time, and the user options that were selected for conversion.
Desktop Computer Database	For Access databases, it displays which .mdb file is being copied and where it's located. For ODBC databases, it shows the connection string.
Options Chosen	Displays the sync or overwrite options chosen. Displays 1 for True and 0 for False.
Index Statistics	Displays information about converted indexes.
Table Statistics	Displays the SQL statement used to create the table and shows the number of records copied.
Closing Statistics	Displays the time the conversion was completed and the number of tables, records, packets, and bytes copied.

Even though this section on data conversion issues may not seem that interesting, it's important for you to take it seriously. Every time you build a desktop or ODBC database that you intend to sync your Pocket PC with, let the type mapping tables guide you. Your big database should only use data types that work well with your little database.

Shrinking a Database

Now it's time to dive in and see how you can put a single-user database synching relationship into production. The first thing you'll need to do is construct a simple database on your desktop computer. Let's stick with the simple contact manager database you've been using throughout the book as a model for your desktop database. My database of choice will be SQL Server 2000, which will communicate with ActiveSync through an ODBC connection.

TIP *If you don't have a copy of SQL Server 2000, download a 120-day evaluation copy from the Microsoft Web site (http://www.microsoft.com/sql/productinfo/evaluate.htm). Microsoft Access is also a perfectly acceptable partner in a single-user database relationship between your desktop and your Pocket PC when you work with ActiveSync.*

Building the Database

Bring up the SQL Server 2000 Enterprise Manager and create a new database called "ContactManager." Next, create a table in your new database called "Contacts." The column names, data types, and so on are listed in Table 6-4.

Table 6-4. Contacts Table Data Types

COLUMN NAME	DATA TYPE	LENGTH	ALLOW NULLS	IDENTITY (AUTO NUMBER)
ContactID	int	4	No	Yes
FirstName	varchar	50	Yes	No
LastName	varchar	50	Yes	No
CompanyName	varchar	50	Yes	No
StreetAddress	varchar	50	Yes	No
City	varchar	50	Yes	No
State	varchar	2	Yes	No
Zip	varchar	10	Yes	No

Make sure that the ContactID column is both the Key field and an Identity column in order to keep it unique. If you entered your data correctly, the resulting design view in the SQL Server Enterprise Manager should look like Figure 6-1.

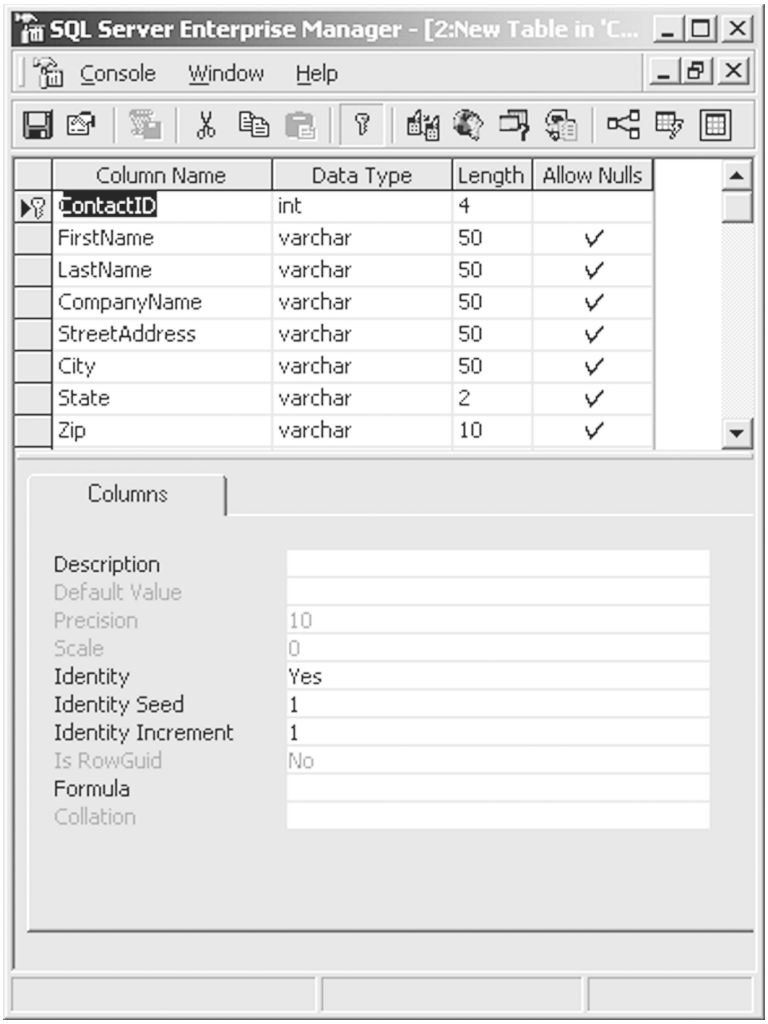

Figure 6-1. The Design view for the Contacts table

Now create the related PhoneNumbers table. The proper column names, data types, and so on are listed in Table 6-5.

Table 6-5. PhoneNumbers Table Data Types

COLUMN NAME	DATA TYPE	LENGTH	ALLOW NULLS	IDENTITY (AUTO NUMBER)
PhoneNumberID	int	4	No	Yes
ContactID	int	4	Yes	No
PhoneNumber	varchar	12	Yes	No

Make sure that the PhoneNumberID column is both the Key field and an Identity column in order to keep it unique. If you entered your data correctly, the resulting design view in the SQL Server Enterprise Manager should look like Figure 6-2.

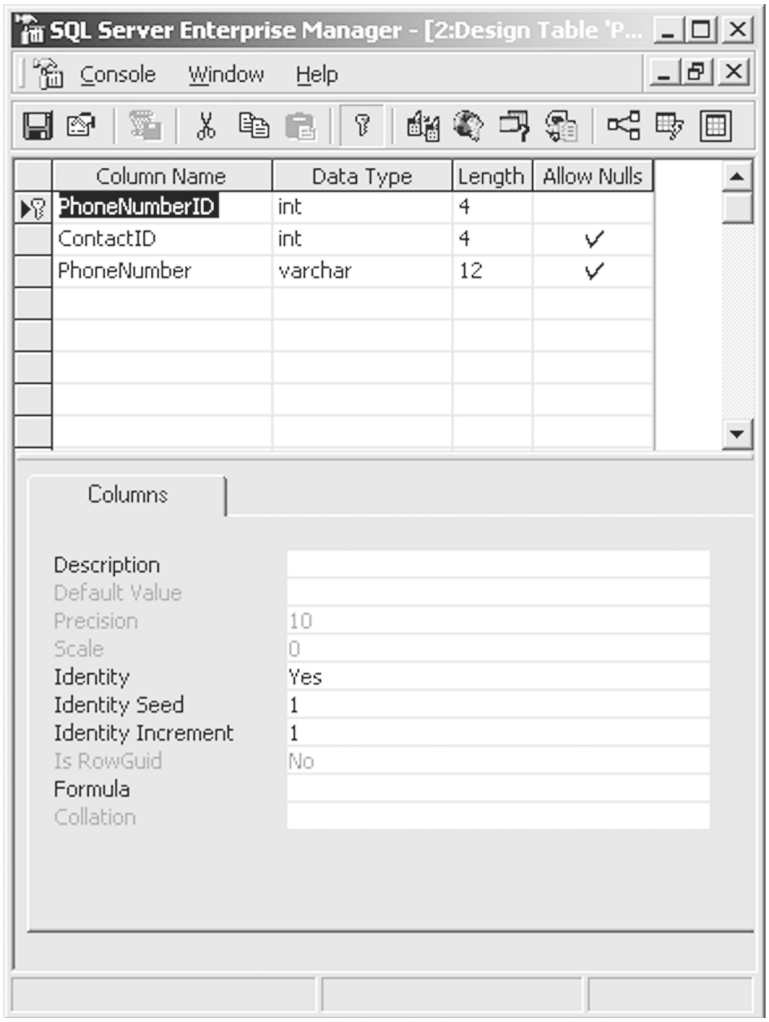

Figure 6-2. The Design view for the PhoneNumbers table

You won't create any explicit one-to-many relationships with the two tables because Pocket Access would ignore the referential integrity rules anyway. Now that you have your ContactManager database and tables constructed, you need to do one more thing before initiating the synching process. Word has it on many of the eMbedded Visual Basic Web sites and newsgroups that your desktop database needs to have at least one row of data entered into it before you run ActiveSync in order to consistently achieve good results. It's not a hard and fast requirement, but having that first row inserted will help you measure success or failure when you view the Pocket Access version of your ContactManager database. Go ahead and enter your own personal contact information into the Contacts table. The value of your ContactID Identity column should be 1 after you've inserted the row of data. Now go to the PhoneNumbers table and enter two phone numbers for yourself. Even though the PhoneNumberID Identity column will autoincrement its numbers, you will need to manually enter the ContactID that was assigned to you in the Contacts table for each phone number that you enter for yourself. Make sure your PhoneNumbers table looks something like Figure 6-3.

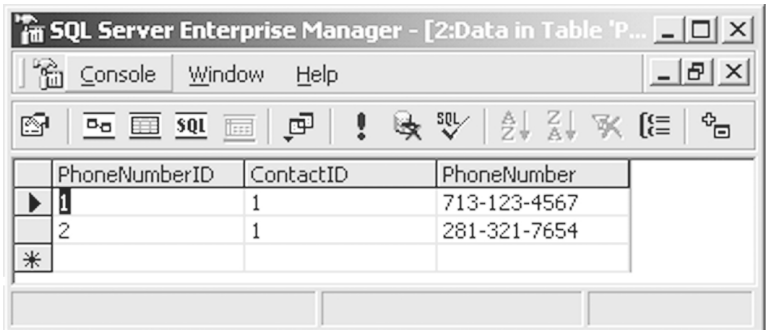

Figure 6-3. The Data view for the PhoneNumbers table

With your desktop database all set and ready to go, you need to create an ODBC DSN for your SQL Server 2000 ContactManager database. Bring up the ODBC Data Source Administrator, click the User DSN tab, and then click the Add button. Highlight SQL Server in the list box and click Finish. On the next screen, type in **ContactManager** for the name, choose (local) from the Server combo box, and then click Next. On the next screen, choose SQL Server Authentication, type in the appropriate login ID and password (with no password if you're bad), and then click Next. Check the option Change the default database to, select ContactManager from the combo box, and then click Next. On the next screen, click Finish. On the last screen, click Test Data Source and then click OK twice if everything worked out with your data connection. Now you're ready for ActiveSync.

ActiveSync Walk-through

If you'll remember back to Chapter 1, I had you check Pocket Access as one of the programs to be synchronized between the desktop and the device. Now you're going to do just that. The other thing I mentioned back in Chapter 1 was that I would make every attempt to ensure that all of the examples in this book could be done with just the emulator. This chapter is going to be the exception because ActiveSync can only work against a real device.

With your Pocket PC sitting in its cradle and actively connected, bring up ActiveSync and select Tools ➢ Import Database Tables (as you see shown in Figure 6-4) to get started with converting your SQL Server database into a Pocket Access database.

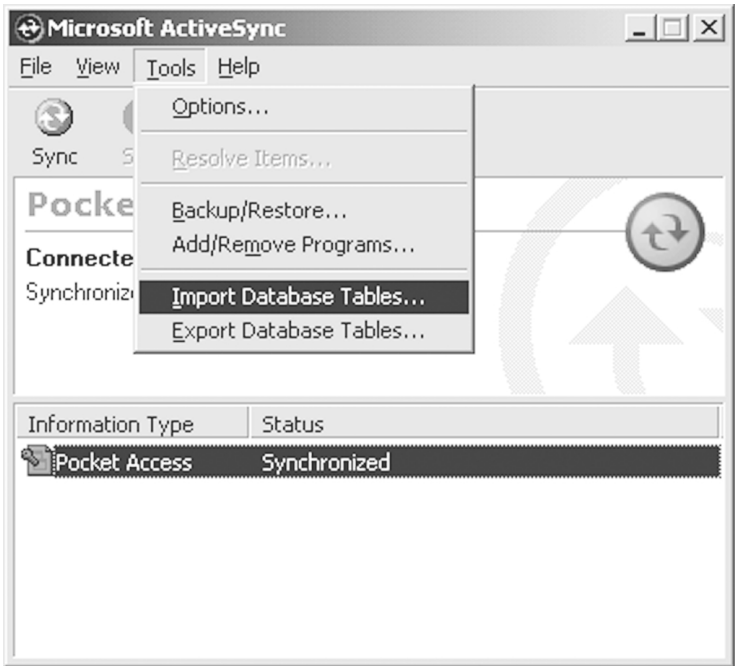

Figure 6-4. Selecting Import Database Tables from the Tools menu

The next thing you'll see is an Open dialog box that enables you to navigate to your desktop database. Its default setting is to look for Microsoft Access databases. Go to the Files of type combo box and select ODBC Database. A Select Data Source dialog box opens for you to select a DSN. Click the Machine Data Source tab, select ContactManager from the list box, and then click OK. When the SQL Server Login dialog box opens, uncheck Use Trusted Connection, type in the appropriate entries in the Login ID and Password text boxes, and then

click OK. You'll briefly see a Copy & Convert dialog box as your SQL Server data is imported. When the import is complete, the Import from Database to Mobile Device dialog box will appear, as shown in Figure 6-5.

NOTE *I don't recommend using the Export Database Tables feature of ActiveSync to create a relationship between a Pocket Access database and a desktop database. This feature takes a Pocket Access database and converts it into a desktop database. Unfortunately, the source database on your Pocket PC only supports a small subset of the features and data types that a desktop or server database supports. This will leave you with a crippled desktop ActiveSync partner that's unable to perform even simple tasks, such as autoincrementing an indexed column.*

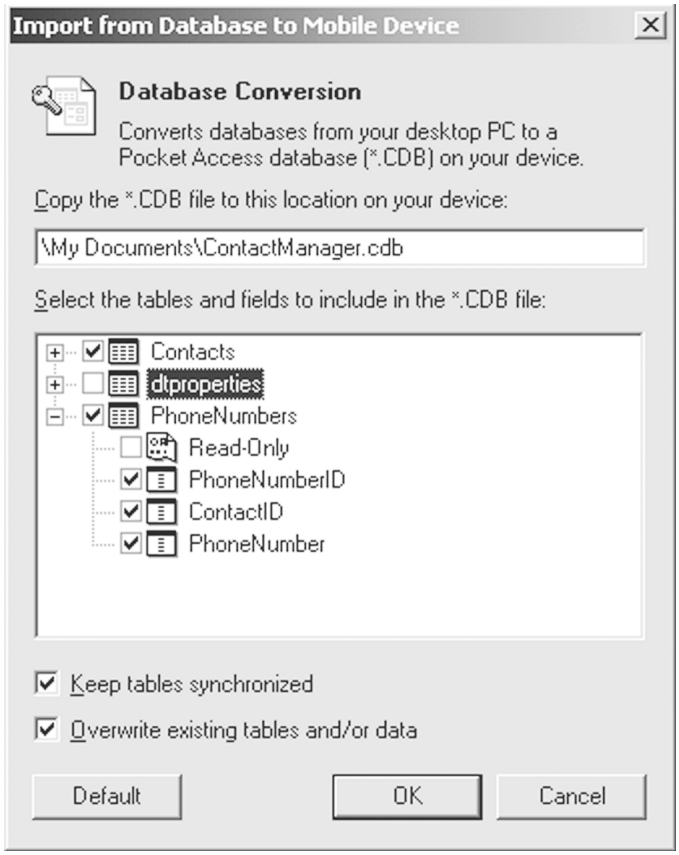

Figure 6-5. The Import from Database to Mobile Device dialog box

In this dialog box, you will specify all aspects of the synchronization relationship between SQL Server and Pocket Access. The text box at the top informs you that it plans to put your ContactManager Pocket Access database in the My Documents folder. This is a good place for your database, but you're free to change the location if you want. A Treeview displays the tables you created and every once in a while it displays a rogue system table called "dtproperties" that you have no interest in. Here you're allowed to select the tables you want converted. You can even decide to select specific fields if you don't want to convert the whole table. Finally, you're given the choice of selecting the Read Only option for each table if you don't want your Pocket Access database to be modified. Under the Treeview, don't change the default settings for keeping the tables synchronized and overwriting existing tables. Make sure that the only tables selected are Contacts and PhoneNumbers and then click OK to get things started. A Copy & Convert dialog box displays, as shown in Figure 6-6, to let you know that the conversion is under way.

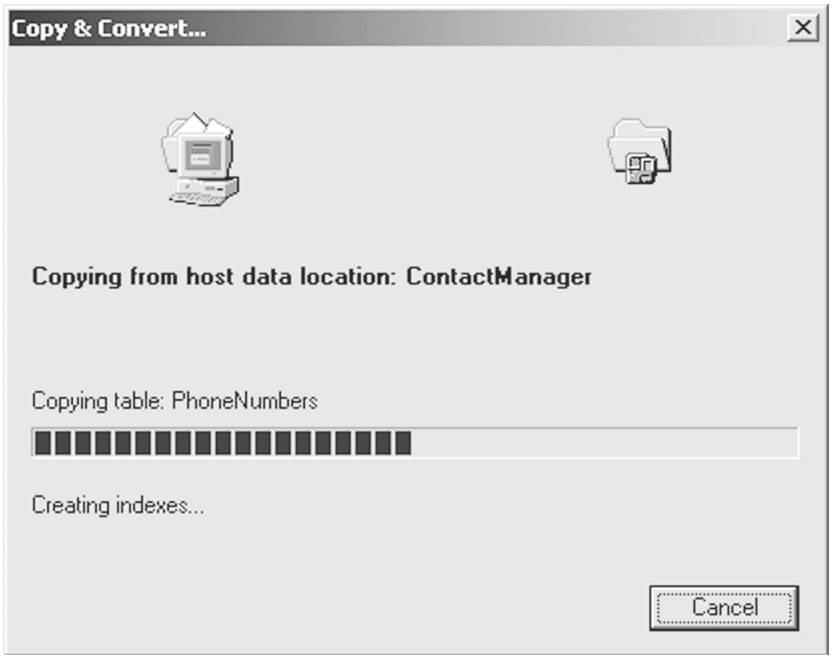

Figure 6-6. Database conversion in progress

To verify the existence of the new Pocket Access database, bring up the File Explorer in your Pocket PC and look for a file named "ContactManager" located in the My Documents folder, as shown in Figure 6-7.

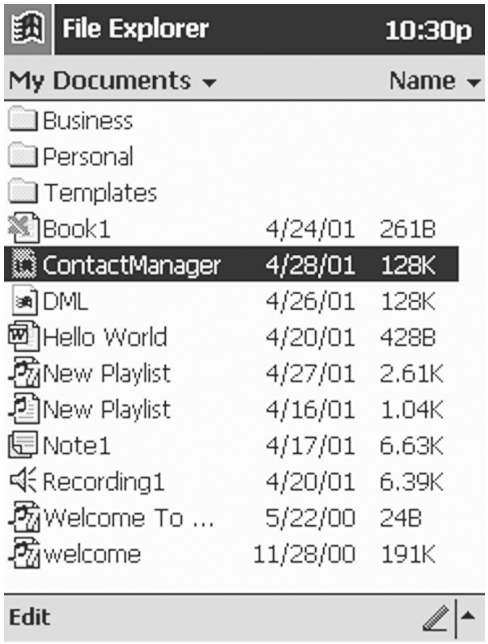

Figure 6-7. File Explorer displaying ContactManager

From now on, every time you return your Pocket PC to its cradle, it automatically synchronizes with the desktop database. Furthermore, you can manually synchronize your desktop and Pocket PC databases anytime you want by clicking the Sync button. Now that you've found the file, you need a flexible way to open and manipulate it. This sounds like a golden opportunity to write some code.

The Database Manipulator

Having the new Pocket Access database on your Pocket PC is only half the battle. You need to see for yourself that you can make additions, updates, and deletions on your Pocket PC and then see those changes reflected in your desktop database the next time you run ActiveSync. You're going to build an eMbedded Visual Basic program to do just that. This time around, you're not going to write code that's specific to the database that you're expecting. Throughout the book, I've made references to building a complete Pocket Access database manager that will give you DDL and DML features similar to what you're used to having in Microsoft Access 2000. In this chapter, you'll build a small piece of the program that will enable you to make additions, updates, and deletions to any Pocket Access database you like. It's about time we wrote a flexible program around here!

Bring up eMbedded Visual Basic and create a new project called "InSync." Check a reference to the Microsoft CE ADO Control 3.0 and add both the Common Dialog control and the Grid control to your project. Finally, add a Module to your project and call it "Module1."

Declaring Globals

Go to your Module and declare public object variables for both the ADOCE Connection object and the Recordset object. Additionally, declare a String variable to maintain the path to whatever Pocket Access database you choose to open.

```
Option Explicit

Public CN As ADOCE.Connection
Public RS As ADOCE.Recordset

Public PocketAccessDatabase As String
```

Instantiating Globals

In the Load event of your main form, you need to add the code necessary to instantiate both the Connection and the Recordset objects.

```
Private Sub Form_Load()

    'Instantiate the Connection Object
    Set CN = CreateObject("ADOCE.Connection.3.0")

    'Instantiate the Recordset Object
    Set RS = CreateObject("ADOCE.Recordset.3.0")

End Sub
```

Closing and Dereferencing Globals

In the OKClick event of your main form, you need to add code to close the Connection object as well as code to dereference both the Connection object and the Recordset object.

```
Private Sub Form_OKClick()

    If CN.State = 1 Then 'Open
        'Let's close the database
        CN.Close
    End If

    'Dereference the Recordset
    Set RS = Nothing

    'Dereference the Connection
    Set CN = Nothing

    App.End

End Sub
```

Opening and Closing the Database

Now things start to get interesting. You need to have a button that enables you to toggle between opening and closing your Pocket Access database. In order to actually open the database, you'll need the Common Dialog control to enable a user to navigate to and open Pocket Access database files. Therefore, you need to drag the Common Dialog control from the Toolbox and drop it on your form. This code also uses a Grid control as well as a combo box, so drag both of those items from the Toolbox and drop them on your form. Name the combo box "cboTableSelect" and set its Text property to "Select a Table." The Grid control can keep its default name and no property adjustments are necessary. To implement the code necessary to open and close the database, drag a CommandButton from the Toolbox and drop it on your form. Name this control "cmdOpenDatabase" and set its caption to read "Open Database." In the click event of this CommandButton, insert the following code:

```
Private Sub cmdOpenDatabase_Click()

    Select Case cmdOpenDatabase.Caption
        Case "Open Database"

            Dim fileflags As FileOpenConstants
            Dim e As Variant
            Dim i As Integer
```

```
'Set the text in the dialog box title bar
CommonDialog1.DialogTitle = "Open Database"

'Set the default filename and filter
CommonDialog1.InitDir = "\"
CommonDialog1.FileName = ""
CommonDialog1.Filter = "Pocket Access (*.cdb)|*.cdb"

'Verify that the file exists
CommonDialog1.Flags = cdlOFNFileMustExist

'Show the Open common dialog box
CommonDialog1.ShowOpen

'Return the path and filename selected or
'Return an empty string if the user cancels the dialog box
PocketAccessDatabase = CommonDialog1.FileName

If PocketAccessDatabase <> "" Then

    'Open the database
    CN.Open PocketAccessDatabase

    'Display Connection Errors
    For Each e In CN.Errors
        MsgBox e.Description
    Next

    'Display Tables
    RS.Open "MSysTables", CN
    While Not RS.EOF

        'Disregard all System Tables
        If RS("TableName") <> "MSysTables" _
        And RS("TableName") <> "MSysIndexes" _
        And RS("TableName") <> "MSysFields" _
        And RS("TableName") <> "MSysProcs" Then
            'Add table to combo box
            cboTableSelect.AddItem RS("TableName")
        End If
```

```
                RS.MoveNext
            Wend

            RS.Close

            'Change caption
            cmdOpenDatabase.Caption = "Close Database"

        End If

    Case "Close Database"

        If CN.State = 1 Then 'Open
            'Let's close the database
            CN.Close
        End If

        'Remove existing data from Grid
        GridCtrl1.Redraw = False
        For i = 1 To GridCtrl1.Rows
            GridCtrl1.RemoveItem 0
        Next
        GridCtrl1.Redraw = True

        'Clear the combo box
        cboTableSelect.Clear

        'Reset combo box text
        cboTableSelect.Text = "Select a Table"

        'Zero out the path to the database
        PocketAccessDatabase = ""

        'Change caption
        cmdOpenDatabase.Caption = "Open Database"

End Select


End Sub
```

The beginning of this block of code starts out with a Case statement based on the value of the CommandButton caption. If the caption reads "Open Database," you execute the appropriate code to get the database loaded into your program. On the other hand, if the caption reads "Close Database," you execute the code necessary to unload the database from your program. The next bit of code deals with the Common Dialog control. You set its filter to ensure that it only looks for files that end in .cdb. You also set a flag that makes sure that the file you're trying to open truly exists. Once the Common Dialog is open and the user has chosen the database they want to open, set the PocketAccessDatabase string equal to the path given to you by the Common Dialog control. Based on that path, you open a database connection and then proceed to list all the tables in the database in a combo box. The last bit of code is concerned with closing the database and performing cleanup operations, such as clearing out the Grid and combo box.

Choosing a Table

Once the database is open, the combo box named "cboTableSelect" is filled with the names of the tables in the database. Selecting one of these tables from the combo box will cause the Grid to be filled with the metadata and data associated with the selected table. In order to make this happen, insert the following code in the click event of this combo box:

```
Private Sub cboTableSelect_Click()

    'Declare variables
    Dim i As Integer
    Dim ColumnNames As String
    Dim ColumnValues As String

    'A keyset-based, forward and backward, read and write Recordset
    RS.Open cboTableSelect.List(cboTableSelect.ListIndex), CN, adOpenKeyset,
    adLockOptimistic, adCmdTable

    'Remove existing data from Grid
    GridCtrl1.Redraw = False
    For i = 1 To GridCtrl1.Rows
        GridCtrl1.RemoveItem 0
    Next
    GridCtrl1.Redraw = True

    'Set the Grid columns equal to the field count
    GridCtrl1.Cols = RS.Fields.Count
```

```
    'Get the column names
    For i = 0 To RS.Fields.Count - 1
        ColumnNames = ColumnNames & RS.Fields(i).Name & vbTab
    Next

    GridCtrl1.Redraw = False

    'Add the column headers to the Grid
    GridCtrl1.AddItem ColumnNames

    'Loop through the Recordset
    While Not RS.EOF

        'Get the column values for this row
        For i = 0 To RS.Fields.Count - 1
            ColumnValues = ColumnValues & RS.Fields(i).Value & vbTab
        Next

        'Add the column values to the row
        GridCtrl1.AddItem ColumnValues

        'Set ColumnValues to a zero-length string
        'so it can be refilled with the next row
        ColumnValues = ""

        RS.MoveNext
    Wend

    GridCtrl1.Redraw = True

    'Close the Recordset
    RS.Close

End Sub
```

The first thing that happens in this block of code is that a Recordset is opened based on the name of the table you selected from the combo box. This Recordset is designed to return all the columns from the table in question. The next thing you'll notice is that I've added some new code to complement the standard Grid-clearing code. At the beginning of the operation, I set the Grid's Redraw property to False and then I set it back to True once all the rows have been removed. Doing this causes the Grid to both clear and fill with data much faster. You may not notice a performance difference in your emulator, but you

sure can see a difference when you run the application on your Pocket PC. The next thing that happens is that you iterate through and dynamically display the column names as well as the column values for each row. Finally, you close the Recordset.

Adding a Record

Now that you have a Grid full of data based on the table you've chosen, you may want to add an additional record to it. In order to do this, you should drag a CommandButton from the Toolbox and drop it on your form. Name this control "cmdAdd" and set its caption to read "Add a Record." In the click event of this CommandButton, insert the following code:

```
Private Sub cmdAdd_Click()

    cmdAdd.Enabled = False

    'Declare variables
    Dim i As Integer
    Dim ColumnNames As String
    Dim ColumnValues As String

    'A keyset-based, forward and backward, read and write Recordset
    RS.Open cboTableSelect.List(cboTableSelect.ListIndex), CN, adOpenKeyset,
    adLockOptimistic, adCmdTable

    'Call the AddNew method
    RS.AddNew

    'Get dynamic user input
    For i = 0 To RS.Fields.Count - 1

        'If the field is an integer. . .
        If RS.Fields(i).Type = adInteger Then

            'Ask the user if the field is autoincrementing
            If MsgBox("Is " & RS.Fields(i).Name & " an autoincrementing field?",
            vbYesNoCancel) = vbYes Then

                'Code to Auto Increment
                Dim AutoNumber As Integer
                Dim Identity As ADOCE.Recordset
                Set Identity = CreateObject("ADOCE.Recordset.3.0")
```

```
                Identity.Open "SELECT " & RS.Fields(i).Name & " FROM " &
cboTableSelect.List(cboTableSelect.ListIndex) & " ORDER BY " &
                RS.Fields(i).Name & " DESC", CN
                If Not RS.BOF And Not RS.EOF Then
                    AutoNumber = CInt(Identity(0)) + 1
                Else
                    AutoNumber = 1
                End If
                Identity.Close
                Set Identity = Nothing

                'Set new field equal to a new autoincremented number
                RS(RS.Fields(i).Name) = AutoNumber

            Else

                'Set new field equal to user input
                RS(RS.Fields(i).Name) = InputBox(RS.Fields(i).Name, "Add")

            End If

        Else

            'Set new field equal to user input
            RS(RS.Fields(i).Name) = InputBox(RS.Fields(i).Name, "Add")

        End If

    Next

    'Ask user if he or she wants the record added
    If MsgBox("Do you wish to add this record?", vbYesNoCancel) = vbYes Then
        RS.Update
    Else
        RS.CancelUpdate
        MsgBox "No new record added."
    End If

    'Query the database again to refresh the Grid
    RS.Requery

    'Remove existing data from Grid
    GridCtrl1.Redraw = False
    For i = 1 To GridCtrl1.Rows
```

```
        GridCtrl1.RemoveItem 0
    Next
    GridCtrl1.Redraw = True

    'Set the Grid columns equal to the field count
    GridCtrl1.Cols = RS.Fields.Count

    'Get the column names
    For i = 0 To RS.Fields.Count - 1
        ColumnNames = ColumnNames & RS.Fields(i).Name & vbTab
    Next

    GridCtrl1.Redraw = False

    'Add the column headers to the grid
    GridCtrl1.AddItem ColumnNames

    'Loop through the Recordset
    While Not RS.EOF

        'Get the column values for this row
        For i = 0 To RS.Fields.Count - 1
            ColumnValues = ColumnValues & RS.Fields(i).Value & vbTab
        Next

        'Add the column values to the row
        GridCtrl1.AddItem ColumnValues

        'Set ColumnValues to a zero-length string
        'so it can be refilled with the next row
        ColumnValues = ""

        RS.MoveNext

    Wend

    GridCtrl1.Redraw = True

    RS.Close

    cmdAdd.Enabled = True

End Sub
```

When you learned how to add a record to a table back in Chapter 5, you knew the table structure in advance and therefore wrote rigid code based on that fact. Because everything about the program here in Chapter 6 is dynamic, the code is a lot trickier. Your code block starts out normally enough with the opening of a Recordset based on the currently selected table. After you call the AddNew method, things start to get a little crazy. The goal is to pop up Input boxes to ask the user to type in the new record data. The problem is that you don't know the names of the table columns in advance of doing so. As a result, you have to iterate through the Fields Collection to determine the column names that the user is entering data into—but don't get too cozy just yet.

What do you do about those pesky autoincrementing indexes that work on the desktop but not on the Pocket PC? The workaround is to check the data type of each column as you iterate through the Fields Collection. When you find an Integer data type, you prompt the user with a Yes/No message box and ask if the field is autoincrementing. Don't worry, when you build your Pocket Access database manager in the next chapter, I promise to be more scientific about the determination of autoincrementing fields. Anyway, if the user chooses Yes, you open up a second Recordset to figure out the highest number in the given field and add 1 to that number to get your new AutoNumber value. If the user chooses No, you prompt the user to enter his or her own Integer in an Input Box.

After getting all the non-Integer user inputs, you ask the user if he or she is sure he or she wants to add this new record. If the user chooses Yes, you call the Update method. If the user chooses No, you call the CancelUpdate method. The rest of the code is similar to what you've done before. You'll empty the Grid and then dynamically refill it to display the new record.

NOTE *Because there's no referential integrity enforcement in Pocket Access, you'll have to follow up on adding, updating, and deleting records in related tables manually.*

Updating a Record

The second DML function you'll want to perform after adding records is updating records. For the purposes of this program, I'll allow you to click any cell in the Grid and let you update the contents of that cell. To accomplish this tall order, drag a CommandButton from the Toolbox and drop it on your form. Name this control "cmdUpdate" and set its caption to read "Update the Selected Record." In the click event of this CommandButton, insert the following code:

```
Private Sub cmdUpdate_Click()
```

```
'Declare variables
Dim i As Integer
Dim ColumnNames As String
Dim ColumnValues As String
Dim SQL As String

If GridCtrl1.RowSel > 0 Then

    If GridCtrl1.TextMatrix(GridCtrl1.RowSel, GridCtrl1.ColSel) <> "" Then

        'Build a query to return just the column and value
        'reflected in the user's Grid selection
        SQL = "SELECT " & GridCtrl1.TextMatrix(0, GridCtrl1.ColSel) &
        " FROM " & cboTableSelect.List(cboTableSelect.ListIndex) & " WHERE " &
        GridCtrl1.TextMatrix(0, GridCtrl1.ColSel) & " = " &
        GridCtrl1.TextMatrix(GridCtrl1.RowSel, GridCtrl1.ColSel)

        'A keyset-based, forward and backward, read and write Recordset
        RS.Open SQL, CN, adOpenKeyset, adLockOptimistic, adCmdText

        'Get dynamic user input
        For i = 0 To RS.Fields.Count - 1

            'Set new field equal to user input
            RS(RS.Fields(i).Name) = InputBox(RS.Fields(i).Name, "Add")

        Next

        'Ask user if he or she wants the record added
        If MsgBox("Do you wish to update this record?", vbYesNoCancel) = vbYes
Then
            RS.Update
        Else
            RS.CancelUpdate
            MsgBox "No new record updated."
        End If

        RS.Close

        'Query the database again to refresh the Grid
        'A keyset-based, forward and backward, read and write Recordset
        RS.Open cboTableSelect.List(cboTableSelect.ListIndex), CN,
        adOpenKeyset, adLockOptimistic, adCmdTable
```

```
'Remove existing data from Grid
GridCtrl1.Redraw = False
For i = 1 To GridCtrl1.Rows
    GridCtrl1.RemoveItem 0
Next
GridCtrl1.Redraw = True

'Set the Grid columns equal to the field count
GridCtrl1.Cols = RS.Fields.Count

'Get the column names
For i = 0 To RS.Fields.Count - 1
    ColumnNames = ColumnNames & RS.Fields(i).Name & vbTab
Next

GridCtrl1.Redraw = False

'Add the column headers to the Grid
GridCtrl1.AddItem ColumnNames

'Loop through the Recordset
While Not RS.EOF

    'Get the column values for this row
    For i = 0 To RS.Fields.Count - 1
        ColumnValues = ColumnValues & RS.Fields(i).Value & vbTab
    Next

    'Add the column values to the row
    GridCtrl1.AddItem ColumnValues

    'Set ColumnValues to a zero-length string
    'so it can be refilled with the next row
    ColumnValues = ""

    RS.MoveNext

Wend

GridCtrl1.Redraw = True

RS.Close
```

```
        End If

    End If

End Sub
```

The first thing that happens at the top of the code block is you make sure that your Update code is only executed if the user selects a nonmetadata row in the Grid. The next thing you have to do is build a complicated SQL statement using the TextMatrix property of the Grid that determines the column name as well as the value of the cell selected by the user. You open a Recordset that contains a single column and a single row to update. You then dynamically prompt the user to enter the new value for this cell. After that, you call the Update or CancelUpdate method depending on whether or not the user wants to commit the change to the database. The final bit of code refreshes the Grid so that you can see the results of the cell update.

Deleting a Record

The Delete function in this program works similarly to the Update function. When a user clicks a cell in the Grid, the row that cell belongs to is deleted. To make this operation a reality, drag a CommandButton from the Toolbox and drop it on your form. Name this control "cmdDelete" and set its caption to read "Delete the Selected Record." In the click event of this CommandButton, insert the following code:

```
Private Sub cmdDelete_Click()

    'Declare variables
    Dim i As Integer
    Dim ColumnNames As String
    Dim ColumnValues As String
    Dim SQL As String

    If GridCtrl1.RowSel > 0 Then

        If GridCtrl1.TextMatrix(GridCtrl1.RowSel, GridCtrl1.ColSel) <> "" Then

            SQL = "SELECT * FROM " & cboTableSelect.List(cboTableSelect.ListIndex) &
            " WHERE " & GridCtrl1.TextMatrix(0, GridCtrl1.ColSel) & " = " &
            GridCtrl1.TextMatrix(GridCtrl1.RowSel, GridCtrl1.ColSel)

            'A keyset-based, forward and backward, read and write Recordset
```

```
RS.Open SQL, CN, adOpenKeyset, adLockOptimistic, adCmdText

'Ask user if he or she wants the record deleted
If MsgBox("Do you wish to delete this record?", vbYesNoCancel) = vbYes Then

    RS.Delete

    RS.Close

    'Query the database again to refresh the Grid
    'A keyset-based, forward and backward, read and write Recordset
    RS.Open cboTableSelect.List(cboTableSelect.ListIndex), CN,
    adOpenKeyset, adLockOptimistic, adCmdTable

    'Remove existing data from Grid
    GridCtrl1.Redraw = False
    For i = 1 To GridCtrl1.Rows
        GridCtrl1.RemoveItem 0
    Next
    GridCtrl1.Redraw = True

    'Set the Grid columns equal to the field count
    GridCtrl1.Cols = RS.Fields.Count

    'Get the column names
    For i = 0 To RS.Fields.Count - 1
        ColumnNames = ColumnNames & RS.Fields(i).Name & vbTab
    Next

    GridCtrl1.Redraw = False

    'Add the column headers to the Grid
    GridCtrl1.AddItem ColumnNames

    'Loop through the Recordset
    While Not RS.EOF

        'Get the column values for this row
        For i = 0 To RS.Fields.Count - 1
            ColumnValues = ColumnValues & RS.Fields(i).Value & vbTab
        Next

        'Add the column values to the row
        GridCtrl1.AddItem ColumnValues
```

```
                    'Set ColumnValues to a zero-length string
                    'so it can be refilled with the next row
                    ColumnValues = ""

                    RS.MoveNext
                Wend
                GridCtrl1.Redraw = True

            Else

                MsgBox "The record is unchanged."
            End If
            RS.Close
        End If
    End If

End Sub
```

After checking to make sure that the user doesn't click a metadata cell in the Grid, a complicated SQL string is constructed with the help of the Grid's TextMatrix property. This query causes the Recordset to open only the row of data the user clicked on in the Grid. The user is then prompted with a Yes/No message box asking whether or not to delete the selected record. If the user chooses Yes, the Delete method is called and the code proceeds to refresh the Grid so you can see that the row in question has been removed.

Trying It Out

Now that you've wired all this code together, it's time to see what your little program can do. By the way, you can put your Grid, combo box, and buttons anywhere you want, but mine looks like Figure 6-8.

Figure 6-8. The InSync program without an open database

You're about to start playing ping-pong with your data as you bounce it from Pocket Access to SQL Server. The InSync program in conjunction with the SQL Server Enterprise Manager will help you accomplish the following tasks:

- Verify the original conversion and row of data
- Add data to Pocket Access
- Update data in SQL Server
- Update data in Pocket Access
- Delete data in SQL Server
- Delete data in Pocket Access

Bring up the InSync program on your Pocket PC so you can get started putting it through its paces.

Verifying Original Conversion and Row of Data

The first thing you need to do is verify that the database you built and the row of data you added in SQL Server has made it to your Pocket PC successfully. Click the Open Database button to display the Open common dialog box. It should default to All Folders and should display only files that end in .cdb, as you see shown in Figure 6-9.

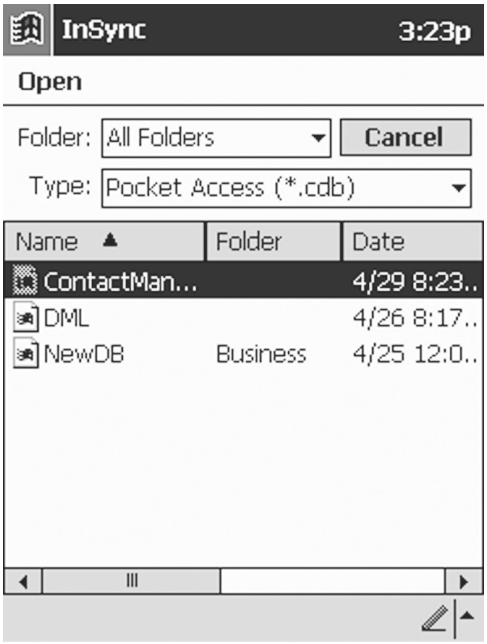

Figure 6-9. The Open common dialog box displaying the ContactManager database

Hopefully, you'll see your ContactManager database. Tap on the database to open it. The next thing you need to do is click the Select a Table combo box (see Figure 6-10) and choose the Contacts table.

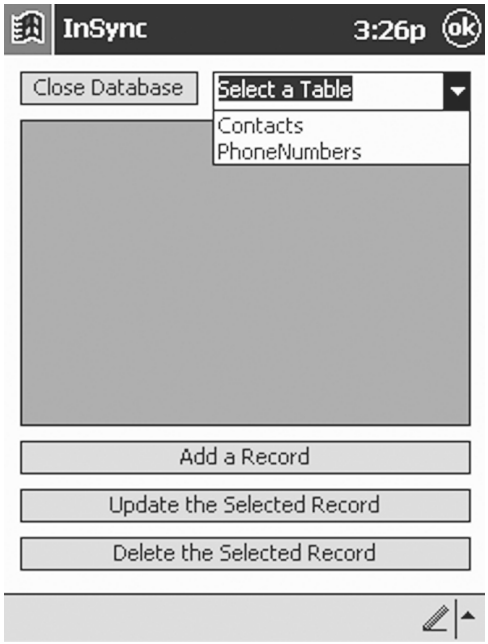

Figure 6-10. Select a table

Once you've clicked Contacts, the Grid should fill with the same metadata and data you entered in SQL Server. Your Grid should look something like Figure 6-11.

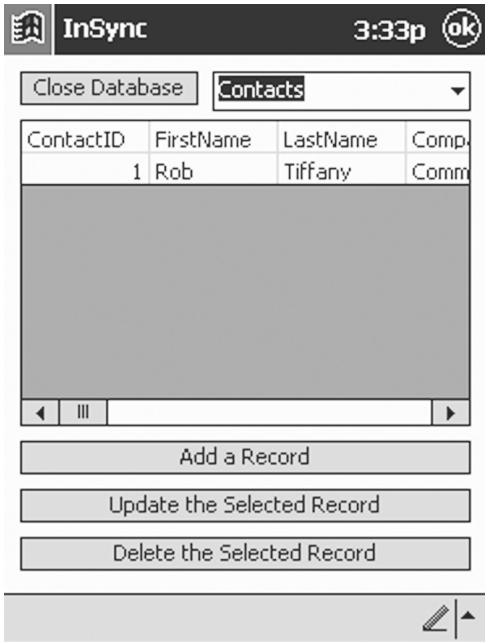

Figure 6-11. The Grid displaying the Contacts table

Be sure that you also verify the contents of the PhoneNumbers table as well before you move on.

Adding Data in Pocket Access

Now you get to try out your Add a Record function and see how its dynamic adding capabilities work. Reselect the Contacts table from the combo box so that it's displayed in the Grid. Now click the Add a Record button to start the process. As your code loops through the metadata, it should display the message box shown in Figure 6-12 almost immediately.

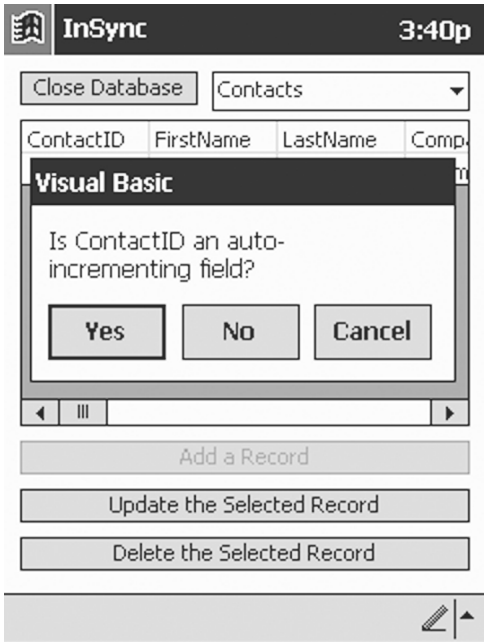

Figure 6-12. Checking for autoincrementing fields

Because you created ContactID as an Identity column in SQL Server, you know that it is an autoincrementing field and you should therefore click Yes. From there on, you will see a series of Input Boxes prompting you to enter values for the given column name as shown in Figure 6-13.

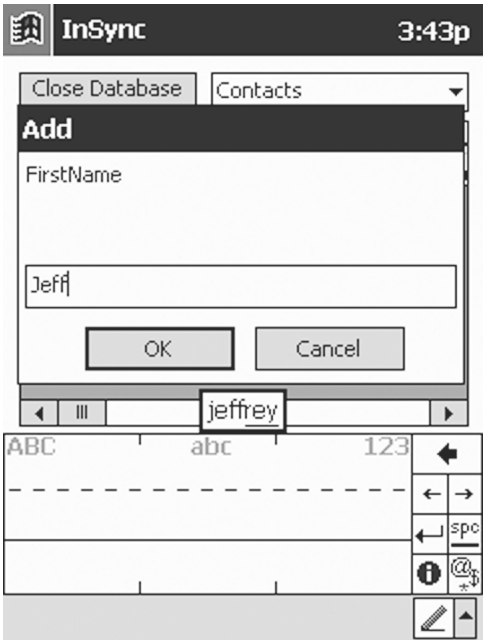

Figure 6-13. Adding a value in the FirstName column

After you've entered all the necessary data for this new record, click Yes when you're asked if you want to add the record. If all went well, you should be looking at a new row of data in your Grid with a ContactID of 2. Now I want you to close the database but leave InSync running. Go to ActiveSync on your desktop and click the Sync button. Once it's finished synchronizing, take a look at your Contacts table in SQL Server through the Enterprise Manager. It should look just like your InSync Grid looked a moment ago (mine did).

Updating Data in SQL Server

With the Contacts table displayed in the Enterprise Manager, change the last name of ContactID 2 by clicking in the cell and making the edit. Now close the Contacts table Grid, bring up ActiveSync and click the Sync button. When the synchronization is complete, go back to the InSync program on your Pocket PC, open the ContactManager database, and view the Contacts table in the Grid. Sure enough, your ContactID 2 should have a new last name. Mine went from Swantkowski to Johnston—try and top that!

Updating Data in Pocket Access

Not to be outdone by SQL Server, let's do some updating with Pocket Access. With the Contacts table displayed in the InSync Grid, tap the FirstName of ContactID 1. The cell should have a dotted outline around it signifying that it has been selected. Now click the Update the Selected Record button. If everything is working correctly, it should only prompt you to enter a new FirstName. When a message box asks if you want to update the record, tap the Yes button. You should now be looking at a refreshed Grid with a new FirstName for ContactID 1. This time, I changed my name from Rob to my wife's name, Cathy. I guess I'll inform her that she's the new CTO over at CommonVision. Now let's find out if Cathy survives the trip back to SQL Server. Close the InSync database and click the Sync button in ActiveSync. Display the Contacts table in the SQL Server Enterprise Manager and see if your Pocket Access change is reflected in SQL Server. In my case, my wife's name managed to beam over to SQL Server without any loss of molecular structure.

Deleting Data in SQL Server

It's now time for one of the two remaining contestants in the Contacts table to get voted out. In the Contacts table Grid in the Enterprise Manager, select the entire row of ContactID 1 and push the Delete key on your keyboard. With only contestant number two remaining, click the Sync button in ActiveSync. When the synchronization is finished, go back to your InSync program on your Pocket PC, open the ContactManager database, and select the Contacts table. If your code is working as good as mine, you should only see ContactID 2 displayed in the Grid.

Deleting Data in Pocket Access

Because no one gets out of this world alive, it's time to delete the remaining row of data in your Contacts table. Tap any one of the cells in ContactID 2's row so that the cell is outlined with dots. Now click the Delete the Selected Record button and definitely tap the Yes button when asked if you want to delete this record. You should now be staring at a Grid displaying only column metadata. Well, let's try to replicate your empty table back to SQL Server. Close the database on the Pocket PC and click the Sync button in ActiveSync. When the synchronization finishes, bring up the Contacts table Grid in the Enterprise Manager. If you did it right, your Grid should be empty with the exception of column metadata, as shown in Figure 6-14.

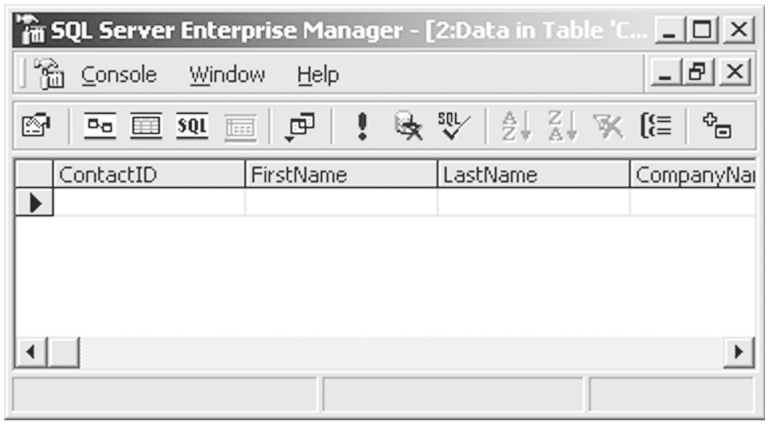

Figure 6-14. An empty table

Now that you've accomplished the previous tasks, you should feel confident in your ability to build single-user Pocket PC database applications that use ActiveSync. You've learned all the important points regarding establishing a relationship between a desktop or server database and a Pocket Access database. What about multiuser ActiveSync database relationships, you might ask? It's important to differentiate what you've done in this chapter with single-user issues versus what might happen in multiuser scenarios and how best to handle those issues. The single-user scenario assumes that only you are using the desktop and Pocket PC databases at any given time. When you're at your desk, only you make additions, updates, and deletions to your desktop database, whether that's Microsoft Access or SQL Server. This scenario also dictates that you'll run ActiveSync to replicate those changes to your Pocket PC *before* you decide to modify the data residing in Pocket Access. Likewise, when you're on the road making additions, updates, and deletions to your Pocket Access database, you must run ActiveSync upon your return to the office to replicate those changes to your desktop database *before* you decide to modify the data residing in Microsoft Access or SQL Server. At this point, you're probably wondering what the big deal is.

In a multiuser scenario, all kinds of people back at your office can make additions, updates, and deletions to your Microsoft Access or SQL Server database while you're on the road with your Pocket PC, and that *is* a big deal. At a minimum, you can just imagine the conflicts that will arise with your autoincrementing fields when you're making additions to Pocket Access while others are making additions to your desktop database at the same time. You'll end up with clashing Identity column numbers that ActiveSync won't be able to resolve and will therefore not synchronize. Go ahead and try this out for yourself by adding a new record to both your desktop and your Pocket Access databases. With new records added in both places with differing data but the same ID number, run ActiveSync to see what happens. You're now staring at an ActiveSync dialog box

reporting the database synchronization conflict that it's going to write to its error log. I know you're hoping that I'm about to show you a slick workaround for this issue, but sadly, Pocket Access is now officially out of its league. Replication usually involves moving from autoincrementing key fields to using randomly generated numbers to uniquely identify a particular row. The only way to be certain that you won't have any conficting key fields is to use 128-bit GUIDs as your key field data type. The ability to autogenerate GUIDs for a key field whenever a new row is added is supported in SQL Server and Access but unfortunately not in Pocket Access. Since I don't know of a way to make the CoCreateGuid call work on the Pocket PC, you wouldn't even be able to generate the GUIDs yourself like you do with the autoincrementing numbers. Don't throw in the towel just yet, there is a solution.

Sophisticated data merge and replication code is used all the time in enterprise products such as Lotus Notes, Oracle, and Microsoft SQL Server. Using the right tools for the job can solve your multiuser replication issues. Those tools are SQL Server 2000, Windows 2000, Internet Information Server 5, and SQL Server CE 1.1. Firewall-friendly HTTP is used as the transport mechanism to replicate changes from SQL Server CE on your Pocket PC to SQL Server 2000 through IIS 5. Best of all, using merge replication, these products are designed to automatically handle the conflicts that arise in multiuser scenarios where the SQL Server 2000 database is being modified at the same time as the SQL Server CE database that's out on the road. I won't delve into SQL Server CE any further because it deserves its own book and this is a book on Pocket Access, but I would like to point you in the right direction to help you get started if you require this greater level of database sophistication. You can download a trial version of SQL Server CE 1.1 on the Microsoft Web site (http://www.microsoft.com/sql/evaluation/trial/CE/download.asp). Additionally, you can find several good SQL Server CE articles and tutorials on the deVBuzz Web site (http://www.devbuzz.com/).

Summary

You now know more about programming a flexible Pocket Access application in eMbedded Visual Basic then you probably ever cared to know. Well, it only gets better. You're going to take everything you've learned up to this point to build the closest thing you can get to having the Microsoft Access 2000 design environment running on your Pocket PC.