

Exercise 4-1: Build the Vendor Maintenance application

In this exercise, you'll develop the application shown in figure 4-9 in your textbook (see below). You will learn how to use bound controls and parameterized queries in your form. You will need to the textbook for some code for this exercise.

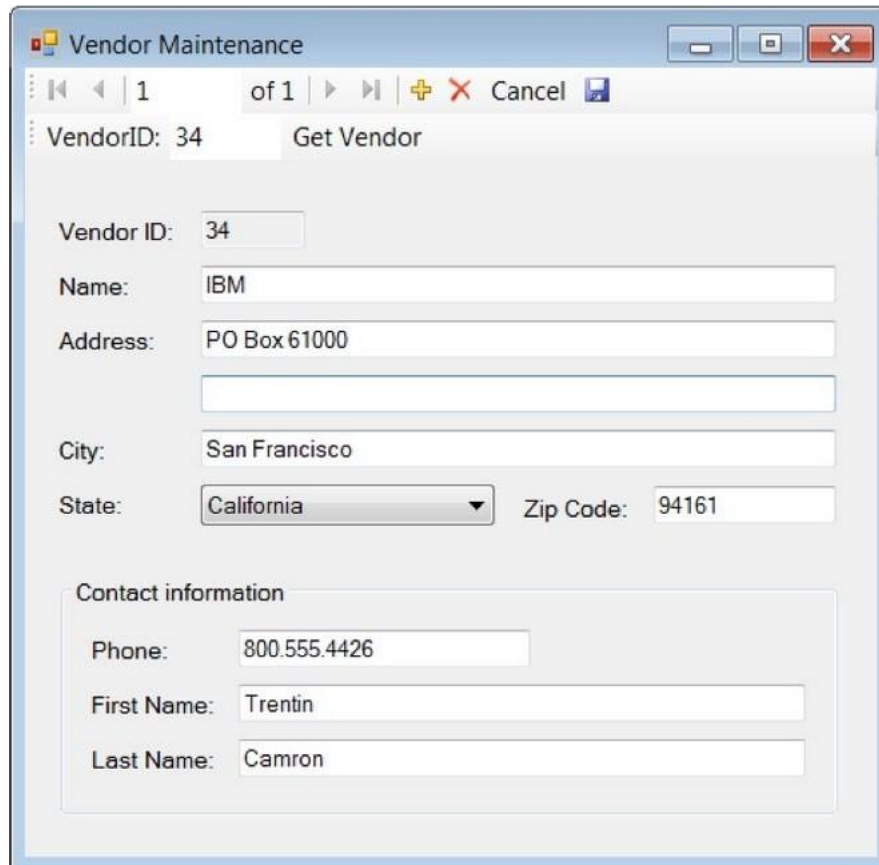
The screenshot shows a Windows-style application window titled "Vendor Maintenance". At the top, there is a toolbar with navigation icons (back, forward, first, last), a status bar showing "1 of 1", and buttons for adding (+), deleting (X), canceling, and saving. Below the toolbar, there is a "VendorID: 34" label and a "Get Vendor" button. The main form area contains several input fields: "Vendor ID: 34", "Name: IBM", "Address: PO Box 61000", "City: San Francisco", "State: California" (a dropdown menu), and "Zip Code: 94161". At the bottom, there is a section titled "Contact information" containing "Phone: 800.555.4426", "First Name: Trentin", and "Last Name: Camron".

Figure 4-9

Build the user interface

1. Start a new project named **VendorMaintenance** in your chapter 4 directory. Then, create a data source for the columns in the Vendors table that are used by the form in figure 4-9, plus the **StateCode** and **StateName** columns in the States table.
2. Drag the first five data source columns in the Vendors table onto the form as text boxes, but leave enough room at the top for two toolbars. Next, drag the State column onto the form as a combo box, and the **ZipCode** column as a text box. Then, rearrange the controls and change any required properties so those controls look like the ones in figure 4-9.

3. Drag a GroupBox control onto the form, adjust its size, and drag the last three Vendor columns from the Data Source window into the group box as text boxes. Then, rearrange them and change any required properties as shown in figure 4-9.
4. Use the procedure in figure 4-3 to bind the State combo box on the form to the States table in the data source. Then, set the DropDownStyle property for the combo box to DropDownList, and set its (DataBindings) – Text property to (none).
5. Test the application to see how this user interface works. Use the combo box to change one of the State entries. End the application, and review the code that has been generated for the form.

Add a parameterized query

6. Use the procedure in figure 4-5 to add a parameterized query named **FillByVendorID** that finds the Vendor row for a specific vendor ID. Then, note the toolbar that has been added to the form. Now, review the code that has been added to the application, and review the schema for the application.
7. Test the application to see how it works. First, use the binding navigator toolbar to move through the rows. Then, enter a vendor ID of 8 in the second toolbar and click the FillByVendorID button. Now, go back to the binding navigator toolbar, and you'll discover that you can't use it to go through the rows anymore because the dataset contains only one row.
8. With the application still running, use the second toolbar to select the vendor with VendorID 10. Choose a new state from the combo box, but don't click the Save button. Then, go to VendorID 20, select a new state, and click the Save button. Now, go back to VendorID 10 to see that the state has reverted to what it was originally. Finally, go to VendorID 20 to see that the state has been changed. This shows that you must click the Save button after each change if you want the changes to be made to the database. That's because the dataset consists of only one row at a time.
9. Add a valid row to the dataset and click the Save button. Then, note that the binding navigator toolbar lets you navigate between the previous row and the one you just added because two rows are now in the dataset. As soon as you use the second toolbar to go to a different row, though, the first toolbar shows only one row in the dataset.
10. Delete the row that you added in step 9 by going to that row and clicking the **Delete** button, which makes the row disappear. Then, click the Save button to apply the deletion to the database. If you don't do that, the row won't be deleted. When you're done, end the application.

Modify the toolbars

11. Use the procedure in figure 4-7 to add a Cancel button to the binding navigator toolbar and to change the text on the **FillByVendorIDToolStripButton** button in the second toolbar to Get Vendor. Then, use the procedure in figure 4-8 to start the event handler for the Cancel button, and add the one line of code that it requires.
12. Test these changes to see how they work. At this point, the application should work like the one in figure 4-9. You just need to enhance the code so it provides for formatting and data validation.

Enhance the code

13. Add the FormatPhoneNumber procedure in figure 4-9 to the form. Then, in the Load event handler for the form, add the code for wiring the Format event of the **Phone** text box to the FormatPhoneNumber procedure. Now, test this enhancement to make sure the phone number is formatted correctly.
14. Add the UnformatPhoneNumber procedure in figure 4-9 to the form. Then, wire it to the Parse event of the of the **PhoneTextBox** Binding object.
15. Comment out the code for filling the **Vendors** table in the Load event handler, and run the application. As you'll see, only the **State** combo box has a value when the application starts because no vendor has been selected. However, this value is simply the first state in the list, not necessarily the correct state for the first vendor. To remove this from the display, use this statement in the Form Load event to set the index for the **State** combo box to -1:

```
StateComboBox.SelectedIndex = -1
```

Then, test this change.

16. At this point, you have a prototype of the application. Although you could add the data validation and error handling code that's shown in figure 4-9, that isn't always necessary for a prototype.

Add another parameterized query to the form

17. Add a parameterized query named **FillByState** that gets all the vendor rows for a specific state based on the state code. Next, run the application and use the third toolbar to get all of the rows for a specific state code like CA. Note that the binding navigator toolbar lets you navigate through these rows.
18. Add a separator at the right of the controls on the **FillByVendorID** ToolStrip, followed by a label, text box, and button that look like the three controls on the **FillByState** ToolStrip. Then, delete the **FillByState** ToolStrip.

19. Modify the code for the form so the **FillByState** button in the **FillByVendorID** ToolStrip gets the vendor rows by state. Then, test this enhancement. When you've got it working right, close the project.

Exercise 4-2: Build the Invoice Maintenance application

This exercise will guide you through the development of the application in figure 4-16. Here again, you'll learn a lot by doing that. But you may find that this takes a lot longer than you thought.

The Invoice Maintenance form

The screenshot shows the 'Invoice Maintenance' application window. At the top, there's a 'VendorID' field with the value '122' and a 'Get Vendor' button. Below this, there are fields for 'Name' (United Parcel Service), 'Address' (P.O. Box 505820), and 'City, State, Zip' (Reno, NV, 88905). The main part of the window is a table with the following columns: InvoiceNumber, InvoiceDate, InvoiceTotal, PaymentTotal, CreditTotal, DueDate, and PaymentDate. The table contains 10 rows of data. To the right of each row is a 'View Line Items' button. At the bottom right of the window is an 'Update Database' button.

InvoiceNumber	InvoiceDate	InvoiceTotal	PaymentTotal	CreditTotal	DueDate	PaymentDate
989499-743	3/10/2010	\$426.12	\$426.12	\$0.00	4/9/2010	4/9/2010
989500-399	4/15/2010	\$208.85	\$208.85	\$0.00	5/15/2010	5/15/2010
989672-101	5/7/2010	\$195.40	\$195.40	\$0.00	6/6/2010	6/6/2010
989699-682	6/11/2010	\$958.76	\$958.76	\$0.00	7/11/2010	7/17/2010
989715-213	7/12/2010	\$223.45	\$223.45	\$0.00	8/11/2010	8/11/2010
990113-886	8/26/2010	\$388.00	\$388.00	\$0.00	9/25/2010	9/25/2010
991562-022	9/20/2010	\$206.22	\$206.22	\$0.00	10/20/2010	10/19/2010
992609-278	10/5/2010	\$328.05	\$0.00	\$0.00	11/4/2010	
993376-631	11/12/2010	\$98.50	\$0.00	\$0.00	12/12/2010	

The Line Items form

The screenshot shows the 'Line Items' application window. It contains a table with three columns: AccountNo, Amount, and Description. The first row is highlighted, showing AccountNo 553, Amount \$2,051.59, and Description Freight.

AccountNo	Amount	Description
553	\$2,051.59	Freight

Figure 4-16: The user interface for the Invoice Maintenance application

Build the user interface for the Invoice Maintenance form

1. Start a new project named **InvoiceMaintenance** in your chapter 4 directory. Then, create a data source that includes the tables and columns shown in the schema in figure 4-17.

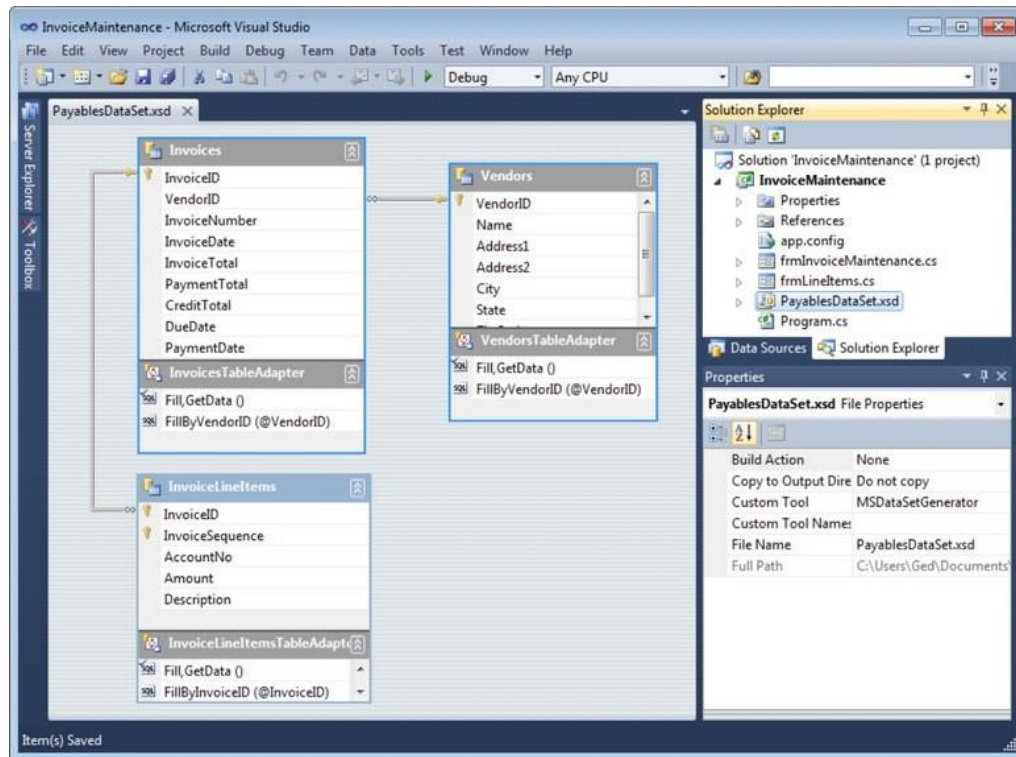


Figure 4-17: The dataset schema for the Invoice Maintenance application

2. Drag the columns in the **Vendors** table onto the form as text boxes, and adjust them as shown in figure 4-16. Then, drag the Invoices table that's subordinate to the **Vendors** table onto the form as a DataGridView control.
3. Run the application and use the binding navigator toolbar to scroll through the vendor rows. When a row that has related Invoice rows is selected, you'll see that the invoice rows are displayed. Now, close the application, and review the code that has been generated for it.
4. Delete the binding navigator toolbar, then comment out the code in the Load event handler for the form that fills the Vendors table (but not the Invoices table). Next, use the procedure in figure 4-5 to create a parameterized query named **FillByVendorID** that gets the Vendor data for a specific VendorID. Then, run the form, enter VendorIDs, and see how the Vendor and Invoice data is displayed.
5. Use the procedures in figures 4-10 and 4-11 to disable adding and deleting rows in the DataGridView control, to turn off the Visible property of the **InvoiceID** column, to remove the **VendorID** column, to set the header text widths of the visible columns as appropriate, to format the date columns, and to set the ReadOnly property for the **InvoiceNumber**, **InvoiceDate**, and **InvoiceTotal** columns to True. Then, set the formatting for the three total columns to Numeric with 2

decimal places, and set their Alignment properties to MiddleRight. Now, run the form to make sure that you've got everything right.

6. Use the procedure in figure 4-13 to add an unbound DataGridViewButtonColumn named ViewLineItemsButton to the **DataGridView** control. Then, set the Text property for this column to "View Line Items", set the UseColumnTextForButtonValue property to True, and adjust the column width. Now, run the form to make sure this is working right.

Build the user interface for the Line Items form

7. Add a new form to the project and name it frmLineItems. Next, drag the **InvoiceLineItems** table onto this form. Then, create a parameterized query named **FillByInvoiceID** that gets the InvoiceLineItems rows for a specific InvoiceID.
8. Delete the toolbars that were generated by the queries, and set the properties of the **DataGridView** control so the form looks like the one in figure 4-16.
9. Add the code for the CellContentClick event of the **DataGridView** control on the Invoice **Maintenance** form, as shown in figure 4-18. This code starts by checking whether column 8 (counting from 0) was clicked, which should be the column that the View Line Items buttons are in. If so, it gets the invoice ID from the row that was clicked, saves it as the Tag property of a new Line Items form, and displays a new Line Items form.
10. Modify the code for the Load event of the Line Items form, as shown in figure 4-18. This code starts by getting the invoice ID from the Tag property of the new form. Then, it fills the **InvoiceLineItems** table in the dataset. For this statement, you can just copy and modify the statement that was generated for the event handler for the Click event of the **FillByInvoiceID** ToolStrip button. Now, delete the code for the other event handlers for this form.
11. Test the application to make sure that the Line Items form is displayed when you click on a View Line Items button for an invoice.

Change the way the application gets the invoice data

12. Review the code for the Load event handler for the Invoice Maintenance form. There, you can see that all the rows in the Invoices table are loaded into the dataset when the form is loaded. Then, the dataset rows for a specific VendorID are displayed in the **DataGridView** control each time the VendorID changes. For some applications, that may be okay, but if there are thousands of invoice rows in the dataset that may be inefficient.

13. To change the way that works, create a parameterized query named **FillByVendorID** for the **DataGridView** control that gets the Invoice rows for a specific VendorID. Delete the ToolStrip that gets generated and delete the code for the Click event of its **ToolStripButton**. Then, modify the code in the event handler for the Click event of the **FillByVendorIDToolStripButton** so it looks like the code in figure 4-18.
14. Delete the Load event handler for this form. Then, test the application again. It should work the same as it did before, but now only the invoice rows for the selected vendor are in the Invoices table in the dataset.

Complete the application

15. At this point, you have a prototype of the application, and you should have learned a lot about how building applications with data sources and datasets works. Now, if you want to finish this application, you just need to: (1) add an **Update Database** button to the bottom of the Invoice **Maintenance** form as shown in figure 4-16; (2) code the event handler for this button's Click event; and (3) make sure all of the properties for both forms and all of the controls on both forms are set right.