CHAPTER 8

More About Bound Controls

In This Chapter

Bound controls revisited

Using the Masked Edit control

Using the DateTimePicker control

Using the DataCombo control

In the previous chapter, I showed you how to build a program without any code using only bound controls and the ADO Data Control. In this chapter, I will continue discussing bound controls by talking about some other properties and events that you can use to help ensure that only valid data is stored in your database. Then I'm going to discuss a few other useful controls that have special features to help ensure that your data is correct before it gets into the database.

Bound Controls Revisited

Bound controls are not as simple as I led you to believe in Chapter 7. There are several other properties, methods, and events that allow you to fine-tune how the user interacts with the control.

Key properties

All bound controls contain the properties listed in Table 8-1, which affect the way the control works with the user and with the database.

Key methods

While the SetFocus method isn't directly used in binding a control to a data source, it is sometimes used in performing data validation.

Table 8-1
Key Properties of Bound Controls

Property	***Description***
`CausesValidation`	A `Boolean` value, where `TRUE` means that the Validation event for the control previously focused will be fired before the focus is shifted to this control.
`DataChanged`	A `Boolean` value, where `TRUE` means that either the user has changed the value of the control or the program has changed the value of the control.
`DataField`	A `String` value containing the field name to which the control is bound.
`DataFormat`	An object reference to an `StdDataFormat` object which contains information about how to format the value displayed by the control.
`DataMember`	A `String` value that identifies which set of data should be used when a `DataSource` has more than one set of data.
`DataSource`	An object reference to an object that can act as a data source. Common data sources include the `ADO Data Control`, a `Recordset` object or a Class module where the `DataSourceBehavior` property is set to `vbDataSource` (1).

Object.SetFocus ()

The `SetFocus` method is used to transfer the focus to the control specified by *object*. If the control specified by *object* isn't visible or can't accept the focus, a runtime error will occur.

Key events

Most controls that can be bound have a set of events that will be fired as the user interacts with the control. These events can be used to verify that the information a user enters into the control is correct.

Event Change ()

The `Change` event is fired each time the value of a control changes. This can happen if the user updates the value in the control or explicitly sets the `Value`, `Caption`, or `Text` property in code.

Event GotFocus ()

The GotFocus event fired before a control receives the focus.

Event KeyDown (KeyCode As Integer, Shift As Integer)

The KeyDown event is fired whenever the user presses a key while the control has the focus, where KeyCode is an Integer value containing the key code of the key that was pressed. This value is not the ASCII character value associated with the key.

Shift is an Integer value that indicates the status of the Shift, Alt, and Ctrl keys.

Event KeyPress (KeyAscii As Integer)

The KeyPress event is fired whenever a key is pressed and released, where KeyAscii contains the ASCII code of the key that was pressed. Changing this value to zero in the event cancels the keystroke.

Event KeyUp (KeyCode As Integer, Shift As Integer)

The KeyUp event is fired whenever the user releases a key while the control has the focus, where KeyCode is an Integer value containing the key code of the key that was released. This value is not the ASCII character value associated with the key.

Shift is an Integer value that indicates the status of the Shift, Alt, and Ctrl keys.

Event LostFocus ()

The LostFocus event is fired just before the focus is transferred to another control.

Event Validate (Cancel As Boolean)

The Validate event is used to verify that the contents of a control are valid before the focus is shifted to another control. The Validate event will only be triggered if the destination control has the CausesValidation property set to TRUE. This is the case where Cancel is a Boolean value, and TRUE means that the focus should not be shifted to the next control. By default, Cancel is set to FALSE.

Data validation

There are basically four ways to verify that the user has entered the correct data into a control like a TextBox:

- Change event
- KeyPress, KeyUp, and KeyDown events
- GotFocus and LostFocus events
- Validate event

Each approach is discussed below, along with any of its advantages and disadvantages.

Using the Change event

The Change event is triggered each time the value of the control is changed. While at first this may be the ideal event with which to verify changes, it turns out it is the least practical approach.

Consider a TextBox control. Each time the user types a character into the text window, the change event is fired. Also, anytime you change the Text property in your program, you trigger the Change event. This can make things really complicated if you choose to change the value of the control while in the Change event.

Using the KeyPress, KeyUp, and KeyDown events

The KeyPress, KeyUp, and KeyDown events are fired whenever a user presses a key on the keyboard. Like the Change event, these events are somewhat limited in their usefulness for validating data. However, you can trap the keystrokes as they are entered and translate them into something or perform a special task.

For instance, you can include some code in the KeyPress event to translate any lowercase characters into their uppercase equivalents automatically. You can also define certain keystrokes, such as the Esc key, to perform special functions, like restoring the original value for the field from the database (see Listing 8-1).

Listing 8-1: **The Text1_KeyPress event in Customer Information Editor**

```
Private Sub Text1_KeyPress(KeyAscii As Integer)

If Chr(KeyAscii) >= "a" And Chr(KeyAscii) <= "z" Then
   KeyAscii = KeyAscii - 32

ElseIf (KeyAscii = 27) And Text1.DataChanged Then
   Text1.Text = Adodc1.Recordset.Fields("Name").OriginalValue
   KeyAscii = 0

End If

End Sub
```

Note

Looking to the future: One of the properties of the ADO Data Control is a reference to the underlying Recordset object containing the data retrieved from the database. I'll cover the Recordset object in detail starting with Chapter 11.

Using the LostFocus event

You can also use the `LostFocus` event to perform edit checks. This event is triggered when the user or the program transfers the focus to another control. Since the `LostFocus` event of the current control is the last piece of processing that occurs before the next control gains the focus, it is an ideal place to verify that the data in the control is valid.

In Listing 8-2, I use the `LostFocus` event to determine if the user has left the field blank. If so, I turn the background red to indicate that this field has an error. After the user has entered some data and leaves the control again, I return the normal background color for the text box. Of course, since I only change the color in the `LostFocus` event, the error will still be flagged after the user returns the focus to the control. You can leave it this way, or you can add some code in the `GotFocus` event to reset the error flag.

Listing 8-2: **The Text1_LostFocus event in Customer Information Editor**

```
Private Sub Text1_LostFocus()

If Len(Text1.Text) = 0 Then
   Text1.BackColor = &HFFFF&

Else
   Text1.BackColor = &H80000005

End If

End Sub
```

Tip

Make an error stand out: No matter how you determine that the contents of a field are invalid, you should include some code to tell the user in no uncertain terms that the value is incorrect. You could include a line such as `Text1.BackColor = &HFF&` to turn the background of a text box red in the `LostFocus` event and use a line of code like `Text1.BackColor = &H80000005` to restore the proper background color. You could also use other colors to display different status conditions, such as yellow (&HFFFF&) to when a field should not be left blank.

Note

Resetting focus in the `LostFocus` event: When your program handles the `LostFocus` event, the focus has already begun the process of shifting to the next control. If you try to use the `SetFocus` method inside the `LostFocus` event to bring the focus back to the current control, the next control will briefly receive the focus, and its `GotFocus` and `LostFocus` events will be fired.

Using the Validate event

Probably the best way to verify that the data entered by a user is acceptable is to put the test in the `Validate` event. When the user attempts to switch the focus to another control, the `Validate` event is fired. If the value in the control isn't acceptable, all you need to do is set the `Cancel` parameter to `TRUE` and the focus will remain with the current control without triggering the `GotFocus`/`LostFocus` events of the other control.

Unlike the other approaches to validating data which operate on a single control at a time, using the `Validate` method requires that you coordinate all of the controls on a single form. In general, all of the controls on the form that are capable of receiving the focus need the `CausesValidation` property set to `TRUE`. The only controls that shouldn't have `CausesValidation` set to `TRUE` are those that would allow the user to abort any changes.

For example, consider a form like the one in Figure 8-1. There are three text boxes and two command buttons that can receive the focus. Each of these controls has the `CausesValidate` property set to `TRUE`, except for the Cancel button. This means that the `Validate` event will be triggered when the user tries to shift the focus on any of the text boxes or the Update button.

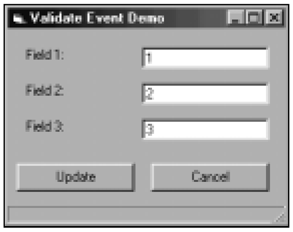


Figure 8-1: Validating data on a form

A typical `Validate` event is shown in Listing 8-3. In this routine, I merely verify that the text box is not empty. If it is, then I'll turn the background to red and display an error message in the status bar. Then I'll set the `Cancel` parameter to `TRUE` to prevent the user from moving the focus to another control.

Listing 8-3: The Text1_Validate Event in Validate Event Demo

```
Private Sub Text1_Validate(Cancel As Boolean)

If Len(Text1.Text) = 0 Then
   Text1.BackColor = &HFF&
   StatusBar1.SimpleText = "Field 1 is blank."
```

```
      Cancel = True
   End If
   End Sub
```

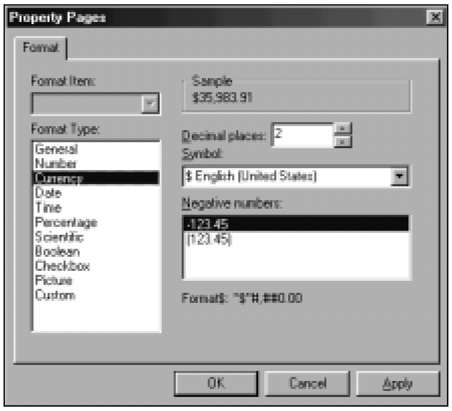
Since the Cancel button has the `CausesValidation` property set to `FALSE`, even if there is an error in one of the text boxes, the focus can be transferred to the Cancel button. Thus, you can use this button to undo any changes or reset the information displayed in the other controls to their default values.

Tip

Explain what caused the error: In addition to making your error stand out, you may want to explain the error in more detail. While you can always display the error message using a message box, consider adding a StatusBar control at the bottom of the form and displaying the error message inside one of the panels. This allows you to display the error message without unnecessarily disrupting the user's work.

Formatting data

Another ability of many bound controls is the ability to format automatically the data they display. You can specify the format information in the `DataFormat` property. You can specify the format information at design time (see Figure 8-2) or at runtime using the `StdDataFormat` object (see Table 8-2).



Format Item:

Format Type: General, Number, Currency (selected), Date, Time, Percentage, Scientific, Boolean, Checkbox, Picture, Custom

Sample: $35,983.91

Decimal places: 2

Symbol: $ English (United States)

Negative numbers: -123.45 (selected), (123.45)

Format$: "$"#,##0.00

Figure 8-2: Specifying a format for your data

Typically, you're going to set the format for a field at design time. Simply choose the appropriate *Format Type* for the value you want to format and a set of options will appear under the *Format* heading corresponding to the *Format Type* you selected.

Table 8-2
Key Properties of the StdDataFormat Object

Property	***Description***
`FalseValue`	A `Variant` containing the value to be displayed when a `Boolean` field is `FALSE`.
`FirstDayOfWeek`	An enumerated value that contains the value of the first day of a week. This information is used to compute values such as the week of the year. Legal values are `fmtDayUseSystem` (0), `fmtSunday` (1), `fmtMonday` (2), `...` and `fmtSaturday` (7).
`FirstWeekOfYear`	An enumerated value that contains information about how to determine the first week of the year. Legal values are `fmtWeekUseSystem` (0), `fmtFirstJan1` (1), `fmtFirstFourDays` (2), and `fmtFirstFullWeek` (3).
`Format`	A `String` value containing a standard format string.
`NullValue`	A `Variant` containing the value to be displayed for a `Null` value.
`TrueValue`	A `Variant` containing the value to be displayed when a `Boolean` field is `TRUE`.
`Type`	An enumerated value that describes the type of information being displayed from the database. Legal values are: `fmtGeneral` (0), `fmtCustom` (1), `fmtPicture` (2), `fmtObject` (3), `fmtCheckbox` (4), `fmtBoolean` (5), and `fmtBytes` (6).

Tip

Numbers and names: Visual Basic often allows you to specify the value of a property by name rather than by number. This is known as an *enumerated value*. You should use enumerated values whenever possible to more clearly document the value you are using.

Using the Picture and Image Controls

Displaying graphic images in your database is very easy when you use a bound `Picture` or `Image` control. Unlike the other bound controls, these controls are strictly one way. They can display data from your database, but changing the image displayed in the control will not update the image in your database when the rest of the row is updated.

The key to using these controls is to bind them to a column in your database containing the raw image. The image can be in any format that is acceptable to the `LoadPicture` function, including `.BMP`, `.JPG`, and `.GIF`. The type of image will automatically be detected and picture will automatically be loaded into the control.

You can't assign an image stored in your database directly to the `Picture` property of the `Picture` or `Image` controls. You must either save the column's value to a file and then use the `LoadImage` function or bind the control directly to the database.

Note

Aren't they the same? The `Image` and `Picture` controls can both be used to display images. All of the data binding properties are the same. However, the `Picture` control has the ability to act as a container for other controls, and it also includes a rich set of drawing methods that are not found on the `Image` control.

Using the Masked Edit Control

The `Masked Edit` control is an alternative to of the text box control. It includes the ability to compare incoming keystrokes against an input mask that determines whether the keystroke is valid or not. Using a simple mask can ensure that numeric fields will contain only numeric values, and with more complex masks, you can enter and display more complex values, such as telephone numbers and social security numbers (see Figure 8-3).

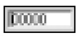


Figure 8-3: Entering a numeric value for the ZIP code.

Tip

Invalid social security numbers: The middle two digits of a social security number (AAA-BB-CCCC) are never 00. The programmers at the Social Security Administration use social security numbers with 00 in the middle to test their applications.

Key properties

The Masked Edit control contains all of the standard properties found in an average ActiveX control, such as Top, Left, Width, Height, Enabled, ToolTipText, etc. However there are a few key properties that affect the way the control works (see Table 8-3).

Table 8-3
Key Properties of the Masked Edit Control

Property	***Description***
AllowPrompt	A Boolean value, where TRUE means that PromptChar is a valid input character.
AutoTab	A Boolean value, where TRUE means that the control will automatically tab to the next field when this field is full.
ClipMode	An enumerated data type that determines if the literal characters displayed in the input mask will be included in a copy or cut operation. A value of mskIncludeLiterals (0) means that the characters will be included, while a value of mskExcludeLiterals (1) means that the characters will not be included.
ClipText	A String containing the contents of the control, without any literal characters.
Format	A String containing up to four format strings separated by semicolons that will be used to display the information in the control.
FormattedText	A String containing the text that will be displayed in the control when another control has the focus.
HideSelection	A Boolean value, where TRUE means that selected text will not be highlighted when the control loses focus.
Mask	A String value containing the input mask. Table 8-4 contains a list of legal mask characters. Any other character in the input mask is considered a literal character.
MaxLength	An Integer containing the maximum length of the input data.
PromptChar	A String containing a single character that is used to prompt the user for input.

Property	***Description***
`PromptInclude`	A `Boolean` value, where `TRUE` means that `PromptChar` is included in the `Text` property. For bound controls `TRUE` means that the value in the `Text` property will be saved in the database and `FALSE` means that the value in the `ClipText` property will be saved.
`Text`	A `String` value containing the value that is displayed in the control while the control has the focus.

Table 8-4
Mask Characters

Character	***Description***
#	A required numeric character.
.	A decimal point indicator as defined in Windows. It is treated as a literal.
,	A thousands separator as defined in Windows. It is treated as a literal.
/	A date separator as defined in Windows. It is treated as a literal.
:	A time separator as defined in Windows. It is treated as a literal.
\	Treat the next character as a literal.
&	A character placeholder.
>	Convert the following characters to uppercase.
<	Convert the following characters to lowercase.
A	A required alphanumeric character.
A	An optional alphanumeric character.
9	An optional numeric character.
C	Same as & (ensures compatibility with Microsoft Access).
?	A required alphanumeric character.
Other	Any other character is treated as a literal.

Creating an input mask

When programming the `Masked Edit` control, the first step is to build an input mask. The best way to start is to choose the mask characters that reflect how your users enter their data and insert them into the `Mask` property. You can add literal characters such as parentheses and dashes to make the input mask easier to use. Table 8-5 lists some sample values for the `Mask` property.

Table 8-5
Sample Masks

Input Mask	*Description*
(###) ###-####	A telephone number.
####-####-####-####	A credit card number.
###-##-####	A social security number.
>A<AAAAAAAA	A name field with the first character always in uppercase and the following characters always in lowercase.
>AAAAAAAAAA	A name field with all characters converted to uppercase.
?#:##	A time value.
##/##/##	A date value.

Caution

The mask didn't work: Just because you use an input mask to ensure the input is in the proper format doesn't mean that the value entered will always be correct. For instance, someone could enter (000) 000-0000 as a telephone number or they could enter 99/99/99 as a date. Both of these values are invalid, yet they meet the requirements specified by the input mask. In these cases, you can use the `Validate` event or another control, such as the `DateTimePicker`, to ensure that the user's data is correct.

As the user enters characters into the control at runtime, each character is validated against the input mask. Any literal characters are frozen on the screen and are automatically skipped over. If the user enters a character that isn't compatible with the input mask, it is ignored, unless you have coded the `ValidateError` event. Then the event will be fired and you can respond in whatever fashion you wish.

Tip

A real multimedia solution: I like to use the full multimedia capabilities of a PC/XT to let the user know that they typed an invalid character in the `Masked Edit` control. Therefore, in the `ValidateError` event, I'll include a Beep statement to generate an audible signal to the user.

Prompting the user

A prompt character can be defined using the `PromptChar` property. This character will be displayed in the each position of the field where the user is expected to enter a value. Generally, you will want to use the underscore (_) character as the prompt character, but you may want to supply a value like a zero or a space depending on the input mask. For instance, if you set `PromptChar` to 0 and Mask to ###-##-####, the user will see 000-00-0000 in the field. The user would then overtype the prompt characters with the appropriate values. Note that if you use a value for PromptChar that is also legal in the input mask, then you need to set the AllowPrompt property to TRUE.

Database considerations

When using the Masked Edit control as a bound control, you need to decide which value you want to store in the database. If you set the PromptInclude property to TRUE, the value in Text property, which will include any literal values from the input mask, will be saved in the database. Otherwise, the value from the ClipText property will be used.

Using the DateTimePicker Control

While you can use the TextBox or the Masked Edit controls to enter date and time values, using the DateTimePicker control is a much better way. The sole purpose of the DateTimePicker control is to help users enter legal date and time values. From a programming point of view, it works just like a TextBox control. You can use the standard data binding properties to connect it to your database.

Key properties

The DateTimePicker control contains all of the standard properties found in an average ActiveX control, such as Top, Left, Width, Height, Enabled, ToolTipText, etc. However, there are a few key properties that affect the way the control works (see Table 8-6).

Table 8-6
Key Properties of the DateTimePicker Control

Property	***Description***
CalendarBackColor	A Long value containing the background color of the calendar.
CalendarForeColor	A Long value containing the foreground color of the calendar.

Continued

Table 8-6 *(continued)*

Property	***Description***
CalendarTitleBackColor	A Long value containing the background color of the calendar's title bar.
CalendarTitleForeColor	A Long value containing the foreground color of the calendar's title bar.
CalendarTrailingForeColor	A Long value containing the foreground color for the dates before and after the current month.
CheckBox	A Boolean value. When TRUE, a check box will be displayed next to the value. If not checked, NULL will be returned as Value.
CustomFormat	A String value containing an alternate format to display date and/or time values chosen from the characters listed in Table 8-7. Also, you must set Format to dtpCustom (3).
Day	A Variant value containing the currently selected day of month.
DayOfWeek	A Variant value containing the currently selected day of week.
Format	An enumerated type specifying the standard or custom format that will be used to display the date and/or time value. Possible values are dtpLongDate (0) to display a date in a long format; dtpShortDate (1) to display a date in a short format; dtpTime (2) to display the time; and dtpCustom (3) to display the value using the format string in CustomFormat.
Hour	A Variant value containing the currently selected hour.
MaxDate	A Date value containing the maximum date value a user can enter.
MinDate	A Date value containing the minimum date value a user can enter.
Minute	A Variant value containing the currently selected minute.
Month	A Variant value containing the currently selected month.

Property	***Description***
`Second`	A `Variant` value containing the currently selected second.
`UpDown`	A `Boolean` value when `TRUE` displays an updown (spin) button to modify dates instead of a drop-down calendar.
`Value`	A `Variant` value containing the currently selected date.
`Year`	A `Variant` value containing the currently selected year.

Table 8-7
Mask Characters

Character	***Description***
D	Displays the day of month without a leading zero (1-31).
dd	Displays the day of month as two digits using a leading zero if necessary (01-31).
ddd	Displays the day of week as a three-character abbreviation (Sun, Mon, etc.).
dddd	Displays the day of week with its full name (Sunday, Monday, etc.).
H	Displays the hour without a leading zero (0-12).
hh	Displays the hour with two digits, using a leading zero if necessary (00-12).
H	Displays the hour in 24-hour format without a leading zero (0-23).
HH	Displays the hour in 24-hour format, using a leading zero if necessary (00-23).
M	Displays the month without a leading zero (1-12).
MM	Displays the month as two digits, using a leading zero if necessary (01-12).
MMM	Displays the month as a three-character abbreviation (Jan, Feb, etc.).
MMMM	Displays the month with its full name (January, February, etc.).
m	Displays the minutes without a leading zero (0-59).

Continued

Table 8-7 *(continued)*

Character	***Description***
mm	Displays the minutes as two digits, using a leading zero if necessary (00-59).
s	Displays the seconds without a leading zero (0-59).
ss	Displays the seconds as two digits, using a leading zero if necessary (00-59).
t	Displays AM or PM as a single character (A or P).
tt	Displays AM or PM as two characters (AM or PM).
x	Uses the Callback events (`CallbackKeyDown`, `Format`, and `FormatSize`) to get the information needed to format the custom date/time value.
y	Displays the day of year (1-365).
yy	Displays the year as a two-digit number (00-99).
yyyy	Displays the year as a four-digit number (0100-9999).

Choosing a user interface

The `DateTimePicker` control has two different ways to allow users to edit date and time values. If you want to edit only date values, you can display a drop-down calendar, which allows the user to select a particular date (see Figure 8-4). Set `UpDown` to `TRUE`.

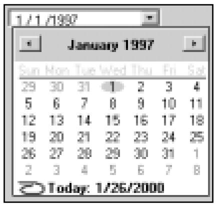


Figure 8-4: Using a drop-down calendar to enter a date

If you want to edit only time values or date and time values, you can set `UpDown` to `FALSE` and the user can edit by clicking on a value and using the spinner arrows at the end of the field to adjust the value (see Figure 8-5). This option works with all formats, including any custom format you may choose to build.

22-Jun-1997 08:52 PM

Figure 8-5: Entering a date and time value

Using the DataCombo Control

The `DataCombo` control looks and works just like a regular `ComboBox` control, except that the data for the control comes directly from the database. This control is extremely useful when you want to translate a coded data value into its text equivalent. It uses two database tables: one table containing the data you want to edit, and a translation table that contains both the encoded and translated values. The beauty of this control is that no code is required to perform the translation process.

Note

They come in pairs: The `DataList` control is very similar to the `DataCombo` control, in the same way a list box is the same as a combo box. Thus, they share many common properties and methods.

Key properties

The `DataCombo` control has a number of properties that affect the way the control works (see Table 8-8). It also supports all of the standard properties, methods, and events listed.

Table 8-8
Key Properties of the DataCombo Control

Property	*Description*
`BoundColumn`	A `String` value that contains the name of the column that will supply that value to the control.
`BoundText`	A `String` value that contains the current value of the `BoundColumn`.
`DataBindings`	An object reference to a `DataBindings` collection.
`ListField`	A `String` value that contains the name of the column used to fill the drop-down list.
`MatchedWithList`	A `Boolean` value that indicates the contents of the `BoundText` matches one of the entries in the list.

Continued

Table 8-8 *(continued)*

Property	***Description***
`MatchEntry`	An enumerated data type that controls how the user's keystrokes will be matched with the values in the control. A value of `dblBasicMatching` (0) means that when the user presses a key, the list of values is searched for by the first item whose first character matches the key that was pressed. If the same key is pressed again, the second item whose first character matches will be displayed. A value of `dblExtendedMatching` (1) means that the control will search for an item in the list whose value matches all of the characters entered. Typing additional characters refines the search.
`RowMember`	A `String` value that identifies which set of data should be used when a `RowSource` has more than one set of data.
`RowSource`	An object reference to an object that can act as a data source. Common data sources include the `ADO Data Control`, a `Recordset` object, or a Class module where the `DataSourceBehavior` property is set to `vbDataSource` (1).
`SelectedItem`	A `Variant` value containing a bookmark for the currently selected record.
`Style`	An `Integer` value that controls how the user interacts with the control. Values are `dbcDropdownCombo` (0), which allows the user to enter a value in the text box or select a value from the drop-down box; `dbcSimpleCombo` (1), which allows the user to enter a value into the text box or select a value from the list below the text box; and `dbcDropdownList` (3), which allows the user to only select a value from the drop-down list.

Tip

Sort, please: You should use a **Select** statement with the **Order By** clause when defining your data source so that the list of values can be displayed in sorted order.

Key methods

The `DataCombo` control supports the usual assortment of methods; however, the `ReFill` method is unique to this and the `DataList` control.

DataCombo.ReFill ()

The ReFill method gets a fresh copy of the data from the RowSource and recreates the list of items in the list.

Note

It's not the same: Don't confuse the ReFill method with the Refresh method. The Refresh method merely repaints the data on the screen. It doesn't get a fresh copy of the data from the database.

Configuring the control

Using the DataCombo control is more complicated than most bound controls, because it needs to interact with two database tables. It also can be used in two different fashions. First, the user may choose a value from a list of values. Second, the user may choose from a list of values that are automatically encoded and decoded as needed.

Selecting from a list

To function as a normal bound control, you need to specify values for both the DataSource and DataField properties. The DataSource property must contain an object reference to an OLE DB data source such as an ADO Data Control or a Recordset object that references the table you want to update. The DataField property specifies the column name whose value is displayed in the text part of the combo box.

The list part of the combo box is populated using the RowSource and ListField properties. The RowSource property is an object reference to an ADO Data Control or a Recordset object corresponding to the list of entries to be displayed in the list part of the combo box. The ListField property contains the name of the column whose values will be displayed in the list part of the control. Before you can use the DataCombo control, you need to set one more property, BoundColumn. This property should be set to the same value as ListField.

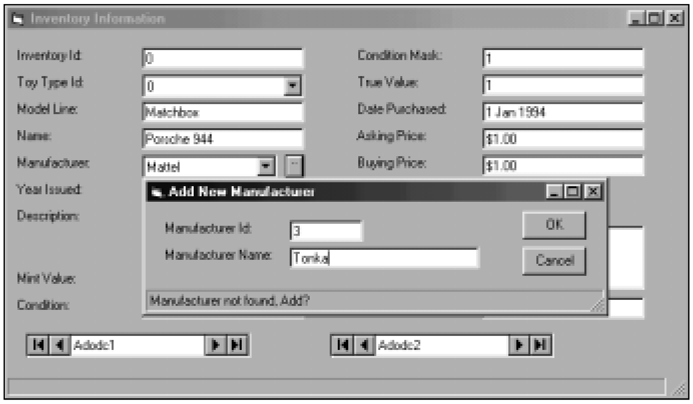
Translating a value

When designing a database it is common to *codify* a value in order to reduce redundancy in your database. When you do this, you typically create a field called ManufacturerId in one table and use another table to translate ManufacturerId into the manufacturer's Name. Thus the ManufacturerId value is known as the *codified* value, while Name is known as the *translated* value.

In the example used in this book, each row in the Inventory table contains a reference to the ManufacturerId, while the Manufacturers table contains ManufacturerId and the Name associated with the ManufacturerId. Because ManufacturerId is a numeric value, it is difficult to remember which value is associated with each manufacturer's name. This situation is an ideal candidate for the DataCombo control.

The properties of the DataCombo control should be set as follows: DataSource = Adodc1 (the table with the raw data); DataField = ManufacturerId (the codified field in the raw data table); RowSource = Adodc2 (the translation table); ListField = Name (the translated field in the translation table); and BoundColumn = ManufacturerId (the codified field in the translation table).

When using the DataCombo control to automatically translate a value, you should set the Style property to dbcDropdownList (3) in order to prevent problems. The other values for Style allow the user to enter their own value into the control. This can cause a translation error, since the value in the codified column doesn't have translation value in the translation table. The easiest way to handle this situation is to place a button beside the DataCombo control to add the new value to the translation table and then refresh the data displayed in the control using the Refill method (see Figure 8-6).



Inventory Id:	0	Condition Mask:	1
Toy Type Id:	0	True Value:	1
Model Line:	Matchbox	Date Purchased:	1 Jan 1994
Name:	Porsche 944	Asking Price:	$1.00
Manufacturer:	Mattel	Buying Price:	$1.00
Year Issued:			
Description:			
Mint Value:			
Condition:			

Add New Manufacturer

Manufacturer Id:	3
Manufacturer Name:	Tonka

Manufacturer not found, Add?

Figure 8-6: Adding a value to the translation table

In the Command2_Click event in the Add New Manufacturer form, which occurs when the OK button is pressed, you begin by adding the new manufacturer to the Manufacturers table (see Listing 8-4). Then you use the Requery method to get a fresh, reordered copy of the data from the database, and then use the data combo's ReFill method to get a fresh copy of the data for the drop-down line. Finally, you should save the newly added manufacturer into the data combo's Text property to save the user from having to select the newly added information.

Listing 8-4: **The Command2_Click event in Inventory Information — Add New Manufacturer form**

```
Private Sub Command2_Click()

Form1.Adodc2.Recordset.AddNew
Form1.Adodc2.Recordset.Fields("ManufacturerId").Value = _
   CLng(Text1.Text)
Form1.Adodc2.Recordset.Fields("Name").Value = Text2.Text
Form1.Adodc2.Recordset.Update
Form1.Adodc2.Recordset.Requery
Form1.DataCombo1.ReFill
Form1.DataCombo1.Text = Text2.Text
Unload Me

End Sub
```

Thoughts on Reducing Data Errors

If you were to write a database program in a different programming language, you would probably have to spend a lot of time moving data from database buffers to the various display fields. In some languages such as COBOL, you might spend as much as fifty percent of your programming effort writing this type of code.

By using bound controls in Visual Basic, you can do two things. First, you reduce the overall size of your program, since you don't have to move all that information to and from the database buffers. Second, you improve the reliability of your program, since the code that handles the data movement isn't yours. Of course, you might make a mistake when specifying which field is associated with a particular control, but this is much easier to find and fix than looking through a complex program and trying to find why a particular field wasn't updated.

Data validation is a big part of the process also. The best way to help prevent your database from becoming corrupt is to verify that all of the data that is entered into the database is acceptable. Using controls such as the `MaskedEdit` and the `DateTimePicker` control means that the data that is entered is at least of the right type and the proper format. This goes a long way towards preventing bad data from reaching your database.

Sometimes bad data will get into your database no matter how much you check and recheck your data before it is entered. For instance, someone could mistype an item number yet still end up with a bad record. Even though the item number was valid, it wasn't the product the customer wanted. To help prevent this type of problem, you should provide as

Continued

Continued

much visual feedback as possible. Perhaps you could display a picture of the item, which would allow the customer to see that they entered the wrong item. Izf a picture isn't practical, you should at least provide a description of the item from which the customer may be able to recognize their mistake.

While providing feedback isn't that important if the information is being entered by a person whose full-time job is to use this application, it is very important for a casual user. A casual user typically isn't very comfortable with the application and is prone to making more mistakes, yet with the proper feedback mechanisms, they will do a better job in the long run.

Summary

In this chapter you learned the following:

- You can validate data in a bound control by using the `Change`, `KeyPress`, `LostFocus`, and `Validate` events.
- You can use the `MaskEdit` control to prompt users for textual information.
- You can use the `DateTimePicker` control to help users select date and time values.
- You can use the `DataCombo` box in place of a normal `ComboBox` control to allow a user to select from a series of values extracted directly from your database.

✦ ✦ ✦