meric Functions

Basic includes functions for many mathematical operations.

Function	Returns
`Abs(x)`	The absolute value of x. $\|x\| = x$ if $x \geq 0$ $\|x\| = -x$ if $x \leq 0$
`Atn(x)`	The angle in radians whose tangent is x.
`Cint(x)`	The even integer closest to x (rounded).
`Cos(x)`	The cosine of x where x is in radians.
`Exp(x)`	The value of e raised to the power of x.
`Fix(x)`	The integer portion of x (truncated).
`Int(x)`	The largest integer $\leq x$.
`Log(x)`	The natural logarithm of x, where $x \geq 0$.
`Rnd`	A random number in the range 0–1 (exclusive).
`Round(x[,DecimalPlaces])`	The rounded value of x, rounded to the specified number of decimal positions. *Note:* Values round to the nearest even number.
`Sgn(x)`	The sign of x. -1 if $x < 0$ 0 if $x = 0$ 1 if $x > 0$
`Sin(x)`	The sine of x where x is in radians.
`Sqr(x)`	The square root of x where x must be ≥ 0.
`Tan(x)`	The tangent of x where x is in radians.

Functions for Working with Strings

Visual Basic provides many string functions for working with text. Although several of the functions are covered in this text, many more are available.

Function	Returns
`Asc(StringExpression)`	The number corresponding to the ASCII code for the first character of String.
`Chr(NumericCode)`	A string character that corresponds to the ASCII code in the range 0–255. Reverse of the `Asc` function.
`Instr([StartingPosition,] StringExpression, SubString)`	A numeric value that is the position within the string where the substring begins; returns 0 if the substring is not found.
`InstrRev(StringExpression, SubString [, StartingPosition[, CompareSetting]])`	A numeric value that is the position within the string where the substring begins, starting from the end of the string; returns 0 if the substring is not found.
`LCase(StringExpression)`	The string converted to lowercase.
`Left(StringExpression, NumberOfCharacters)`	The left-most characters of the string for the indicated number of characters.
`Len(StringExpression)`	A numeric count of the number of characters in the string.
`Mid(StringExpression, StartingPosition [,NumberOfCharacters])`	A substring taken from the string, beginning at StartingPosition for the specified length.
`Replace(StringExpression, FindString, ReplacementString, [, StartingPosition[, Count[, CompareSetting]]])`	A string that has the FindString replaced with the ReplacementString the specified number of times.
`Right(StringExpression, NumberOfCharacters)`	The right-most characters of the string for the indicated number of characters.
`Space(NumberOfCharacters)`	A string of blank spaces for the specified number of characters.
`Str(NumericExpression)`	The string value of the numeric expression; used to convert numeric values to strings. Reverse of the `Val` function.
`String(NumberOfCharacters, StringExpression)`	A string of the named character(s) for a length of the specified number of characters.
`StrReverse(StringExpression)`	A string with the characters reversed.
`UCase(StringExpression)`	The string converted to uppercase.
`Val(StringExpression)`	The numeric value of the string expression; used to convert strings to numeric values. Reverse of the `Str` function.

To display Yes, No, and Cancel buttons along with a question mark icon, use one of these statements:

```
intResponse = MsgBox(strMsg, vbYesNoCancel + vbQuestion, "Good Question")
intResponse = MsgBox(strMsg, 3 + 32, "Good Question")
intResponse = MsgBox(strMsg, 35, "Good Question")
```

MsgBox Example

Earlier you learned to use the `Clear` method to clear the contents of a list box or combo box. In this example we will give the user a chance to confirm whether or not to really clear the list. (Refer to Figure 7.7 for the message box displayed by this procedure.)

```
Private Sub mnuEditClear_Click()
    'Clear the coffee list
    Dim intResponse As Integer

    intResponse = MsgBox("Clear the coffee flavor list?", _
        vbYesNo + vbQuestion, "Clear coffee list")
    If intResponse = vbYes Then
        cboCoffee.Clear
    End If
End Sub
```

Using String Functions

When you need to look at part of a string, rather than the entire string, Visual Basic provides the **`Left`**, **`Right`**, and **`Mid`** **functions** that return the specified section of a string.

The Left, Right, and Mid Functions—General Form

```
Left(StringExpression, NumberOfCharacters)
Right(StringExpression, NumberOfCharacters)
Mid(StringExpression, StartPosition, [NumberOfCharacters])
```

StringExpression in each of these statements may be a string variable, string literal, or text property. NumberOfCharacters and StartPosition are both numeric and may be variables, literals, or numeric expressions. In the `Mid` function, if you omit the NumberOfCharacters argument, the function returns all characters starting with StartPosition.

The Left, Right, and Mid Functions—Examples

```
Left(txtName.Text, 5)          'Returns first 5 characters
Right(strLongString, 1)        'Returns last 1 character
Mid("Mad Hatter", 5, 3)        'Returns 3 characters beginning with character 5
Mid(strProductID, 4)           'Returns all characters beginning with character 4
```

Examples Using Left, Right, and Mid Functions

```
Dim strExample As String
strExample = "It's a wonderful life"
lblMessage.Caption = Left(strExample, 1)      'Returns "I"
lblMessage.Caption = Left(strExample, 6)      'Returns "It's a"
lblMessage.Caption = Left(strExample, 21)     'Returns "It's a wonderful life"
lblMessage.Caption = Left(strExample, 50)     'Returns "It's a wonderful life"
lblMessage.Caption = Right(strExample, 1)     'Returns "e"
lblMessage.Caption = Right(strExample, 4)     'Returns "life"
lblMessage.Caption = Mid(strExample, 8, 9)    'Returns "wonderful"
lblMessage.Caption = Mid(strExample, 8)       'Returns "wonderful life"

If Left(txtDirection.Text, 1) = "N" Then
    lblDirection.Caption = "North"
End If
```

The Len Function

You can use the **Len function** to determine the length of a string expression. You may need to know how many characters the user has entered or how long a list element is.

The Len Function—General Form

```
Len(StringExpression)
```

The value returned by the Len function is an integer count of the number of characters in the string.

The Len Function—Examples

```
Len("Visual Basic")         'Returns 12
Len(txtEntry.Text)          'Returns the number of characters in the text box
Len(strSelection)           'Returns the number of characters in the string variable

If Len(txtName.Text) = 0 Then
    MsgBox "Enter a name", vbInformation, "Data Missing"
End If
```

You can combine the Len function with two new properties of text boxes and the text portion of a combo box—the SelStart and SelLength properties. The SelStart property sets or returns the position of the first selected character; the SelLength property sets or returns the number of selected characters. You can make the current contents of a text box or the Text property of a combo box appear selected when the user tabs into the control and it receives the focus.

```
Private Sub txtName_GotFocus()
    'Select the current entry

        .SelStart = 0                'Begin selection at start
        .SelLength = Len(.Text)      'Select the number of characters
    End With
End Sub
```

Selecting Entries in a List Box

When a list box has a very large number of entries, you can help users by selecting the matching entry as they type in a text box. This method is similar to the way the Help Topics list in Visual Basic works. For example, when you type *p*, the list quickly scrolls and displays words beginning with *p*. Then if you next type *r*, the list scrolls down to the words that begin with *pr* and the first such word is selected. If you type *i* next, the first word beginning with *pri* is selected. The following example implements this feature. See if you can tell what each statement does.

```
Private Sub txtCoffee_Change()
    'Locate first matching occurrence in the list

    Dim intIndex As Integer
    Dim blnFound As Boolean

    Do While Not blnFound And intIndex < lstCoffee.ListCount
        If UCase(Left(lstCoffee.List(intIndex), Len(txtCoffee.Text))) =
            UCase(txtCoffee.Text) Then
            lstCoffee.ListIndex = intIndex
            blnFound = True
        End If
        intIndex = intIndex + 1
    Loop
End Sub
```

Sending Information to the Printer

So far, any printed output has been done with the `PrintForm` method. When you print using `PrintForm`, all output is produced as a graphic, which does not produce attractive text In addition to printing forms, you will need to create reports or print small bits of information on the printer. You can use VB's **`Print` method** to print text on a form, on the Printer object, or in the Debug window.

Visual Basic was designed to run under Windows, which is a highly interactive environment. It is extremely easy to create forms for interactive programs, but not easy at all to print on the printer. Most professional programmers using Visual Basic use a separate utility program to format printer reports.

Random Data Files

The primary difference between sequential files and random files is that you can read and write the data in any order in a **random file**. With sequential files you must always start at the beginning of the file and proceed in order through the file. Random files offer greater speed as well as the capability for random access.

You can visualize random files as a table in which each entry may be referenced by its relative position. Each entry in a file is one record, which is referred to by its record number. Any record in the file may be read or written without accessing the preceding records. Figure 10.3 illustrates the table concept of random files.

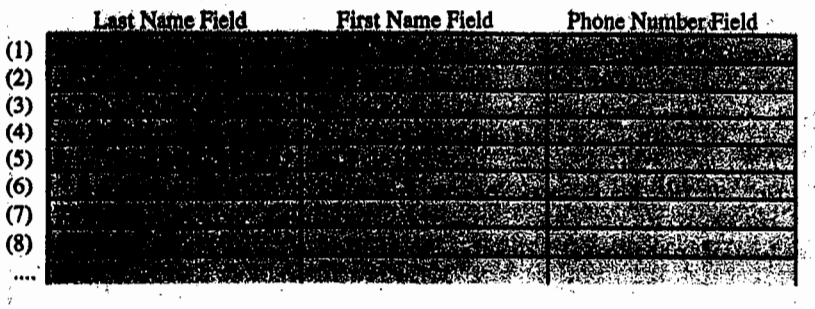


Figure 10.3

Layout of a random file. Each record consists of a Last Name field, a First Name field, and a Phone Number field.

All records in a random file are exactly the same size. The fields within the record are fixed in length and position. That is, if the name takes the first 30 bytes (characters) in one record, every record will allocate the first 30 bytes for the name. This scheme is a departure from sequential files with their variable-length fields and records. Before reading or writing a random file, the record structure or layout must be defined. The `Type/End Type` statements set up record structures. The only modification you will need is to use fixed-length strings.

Fixed-Length Strings

String variables may be variable length or fixed length. Until this point, all strings have been variable length. But for random files you need to specify fixed-length strings. You can define a specific number of characters for elements of user-defined data types and dimension fixed-length single variables and arrays.

If the value you store in a field is less than its specified length, the extra positions are filled with spaces. If you assign a value that is longer than the fixed length, the extra characters are truncated (chopped off) when the value is stored in the fixed-length string.

To specify the string length, add an asterisk (*) followed by the size to the string declaration.

```
Dim strName             As String * 30

Type FullName
    strLastName         As String * 20
    strFirstName        As String * 20
End Type
```

Defining a Record for a Random File

To define a record for a random file, first set up its structure with a Type statement. Then dimension a record variable of the data type.

Note: You can code Type statements in a standard code module or the General Declarations section of a form module or a class module. In a form module or a class module, you must specify Private.

```
Private Type MemberStructure
    strLastName         As String * 20
    strFirstName        As String * 20
    strPhone            As String * 12
End Type
Dim mudtMemberRecord    As MemberStructure
```

Opening a Random File

The Open statement for a random file is the same as for sequential files, using a file mode of Random. This mode allows you to input and output to the same file without closing and reopening it.

```
Open "B:\Data\Names.txt" For Random As #1 Len = 52
Open "A:Members.dat" For Random As #2 Len = Len(mudtMemberRecord)
```

For a random file, the Len (length) entry refers to the length of a single record. In the second example the second Len is actually the Length function that returns the size in bytes of the item enclosed in parentheses. The item in parentheses is the name used for the record variable you declared.

Determining Whether a File Exists

When you open a random file, the file can have both input and output. Therefore, if the file doesn't exist when the Open executes, no error is generated. Instead, VB sets up an empty file so you can start adding records.

Sometimes you want the user to know that no file was found; maybe he or she didn't specify the correct file or load the right disk. You can check for the existence of a file using the **Dir function**. The Dir function returns the path of the named file; if the file doesn't exist, Dir returns an empty string.

The FileNumber entry is the file number from a currently open file.

To determine the highest record number in the file, divide the return value of the LOF function by the size of one record (the name dimensioned using the user-defined data type).

The LOF Function—General Form

```
LOF(FileNumber)
```

The LOF Function—Example

```
intNumberRecords = LOF(1) / Len(mudtMemberRecord)
```

You can use the LOF function to find out how many records are in the file prior to using a For/Next loop that might load the data into a table or list.

```
'Read a random file and store member names into a list box
Dim intNumberRecords As Integer
Dim intIndex As Integer 'Index for the loop

intNumberRecords = LOF(1) / Len(mudtMemberRecord)
For intIndex = 1 To intNumberRecords
    Get #1, intIndex, mudtMemberRecord
    lstNames.AddItem mudtMemberRecord.strLastName
Next intIndex
```

When you are adding to the end of the file, you can use a calculation to find the next record number.

```
intRecordNumber = LOF(1) / Len(mudtMemberRecord) + 1
Put #1, intRecordNumber, mudtMemberRecord
```

The Seek Function

At times you may need to determine the position of the file pointer within the file. The **Seek function** returns the current location of the pointer. For a sequential file, the current byte number is returned. For a random file, Seek returns the position (record number) of the *next* record in the file.

The Seek Function—General Form

```
Seek(FileNumber)
```

FileNumber is the file number of a currently open file.

The Seek Function—Example

```
intNextRecord = Seek(1)
```

Using a List Box to Store a Key Field

When you `Get` or `Put` a record in a random file, you need to know the record number. You can use the method introduced in Chapter 9 to use a list box. Display the name or identifying information in a sorted list box, which makes it easy for the user to find the desired record. Each name's corresponding ItemData property can hold the record number. When the user selects a record from the list, you can use the number stored in ItemData to retrieve the correct record. Figure 10.5 illustrates using a list box for record selection.

Figure 10.5

The list box holds employee names in the List property and corresponding keys (record numbers) in the ItemData property.

	List box List property	ItemData property
User selects this name from list.	Brooks, Barbara	5
	Chen, Diana	3
	Dunning, Daniel	1
	Khan, Brad	6
	Lester, Les	2
	Nguyen, Ahn	4
	Potter, Pete	8
	Stevens, Roger	7

Program displays record 6 from collection.

The following code segment concatenates a first and last name, adds to the list box, and then assigns the record number to the ItemData property.

```
'Add to the list box and set ItemData
    strName = txtLastName.Text & ", " & txtFirstName.Text
    With frmMain.lstEmployee
        .AddItem strName
        lngKey = Val(mEmployees.HighestKey) + 1
        .ItemData(.NewIndex) = lngKey
    End With
```

When concatenating fixed-length strings, the entire string length, including the spaces, is used. You can use a `Trim` function to avoid including the extra spaces.

Trimming Extra Blanks from Strings

The **`Trim`**, **`LTrim`**, and **`RTrim` functions** remove extra blank characters in a string. When you have fixed-length strings, such as the fields in a random file, the strings are likely padded with extra spaces. The `LTrim` function

removes extra spaces at the left end of the string, RTrim removes extra spaces on the right, and Trim removes extra spaces from both the left and right ends.

The Trim, LTrim, and RTrim Functions—General Form

```
Trim(StringExpression)
LTrim(StringExpression)
RTrim(StringExpression)
```

The Trim, LTrim, and RTrim functions return a string with the extra spaces removed.

The Trim, LTrim, and RTrim Functions—Examples

```
Trim(" Harry Rabbit ")      'Returns "Harry Rabbit"
LTrim(" Harry Rabbit ")     'Returns "Harry Rabbit "
RTrim(" Harry Rabbit ")     'Returns " Harry Rabbit"
```

Navigating through a Random File

When displaying information from a file, you should allow your user to step through the records.

Because you are using a sorted list box to control the sequence of the records, you will display the records in the same order they appear in the list. This task is easy to accomplish by using the properties of the list box. ListIndex indicates the currently selected record. You can move to the next record by adding 1 to the ListIndex property. Make sure not to allow the ListIndex to exceed the ListCount − 1. Figure 10.6 illustrates using the list box for navigation.

Figure 10.6

The list box ItemData property holds the record key. To move through the file sequentially, use the ListIndex property to determine the next customer and then use the ItemData property to display the correct record.

ListIndex property	List box List property	ItemData property
(0)	Brooks, Barbara	5
(1)	Chen, Diana	3
(2)	Dunning, Daniel	1
(3)	Khan, Brad	6
(4)	Lester, Les	2
(5)	Nguyen, Ahn	4
(6)	Potter, Pete	8
(7)	Stevens, Roger	7

When record 6 displays, the ListIndex property is 3.

Add 1 to ListIndex, and then display record #2.

indicate that Bold is currently selected. Choosing the *Bold* command a second time should remove the check mark and turn off the Bold option.

The check boxes in the menu editor for *Checked* and *Enabled* determine the beginning state for these options. By default, menu commands are enabled. If you wish the item to be grayed, the check box must be empty.

Toggling Check Marks On and Off

If you create a menu option that can be turned on and off, you should include a check mark to indicate the current state. You can set the initial state of the check mark in the menu editor (refer to Figure 5.2). In code you can change its state by setting the menu item's Checked property.

```
Private Sub mnuFormatBold_Click()
    If mnuFormatBold.Checked = True Then
        mnuFormatBold.Checked = False
        txtChangeable.Font.Bold = False
    Else
        mnuFormatBold.Checked = True
        txtChangeable.Font.Bold = True
    End If
End Sub
```

Standards for Windows Menus

When you write applications that run under Windows, your programs should follow the Windows standards. You should always include keyboard access keys; if you include keyboard shortcuts (Ctrl + key), stick with the standard keys, such as Ctrl + P for printing. Also follow the Windows standards for placing the *File* menu on the left end of the menu bar and ending the menu with an *Exit* command; if you have a *Help* menu, it belongs at the right end of the menu bar.

Plan your menus so that they look like other Windows programs unless your goal is to confuse people.

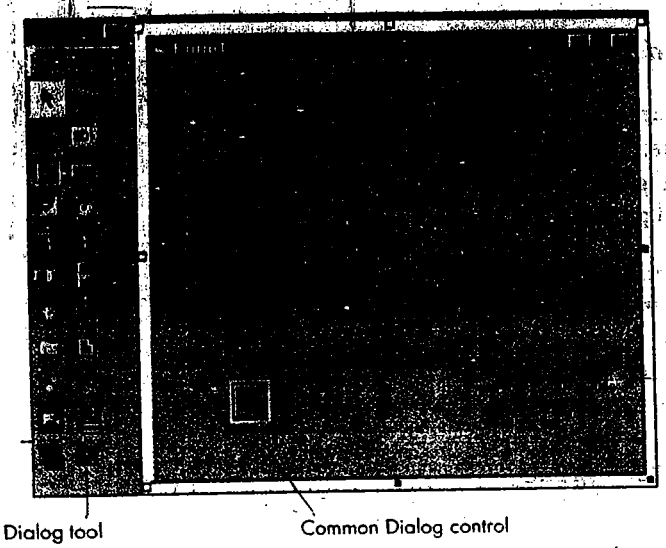
Common Dialog Boxes

You can use a set of predefined standard dialog boxes in your projects for such tasks as specifying colors and fonts, printing, opening, and saving. The **common dialog** control, which is a custom control, allows your project to use the dialog boxes that are provided as part of the Windows environment.

To use the common dialog control, you first place a control on your form (Figure 5.11). You cannot change the size of the control, and its location doesn't matter, since it will be invisible when your program runs. In code, when you wish to display one of the standard dialog boxes, you will refer to properties and methods of the control. You don't need more than one common dialog control on your form even if you plan to use more than one type of dialog box; each time you refer to the control, you specify which dialog box you want.

Note: Windows determines the size of the dialog boxes; you cannot modify their size.

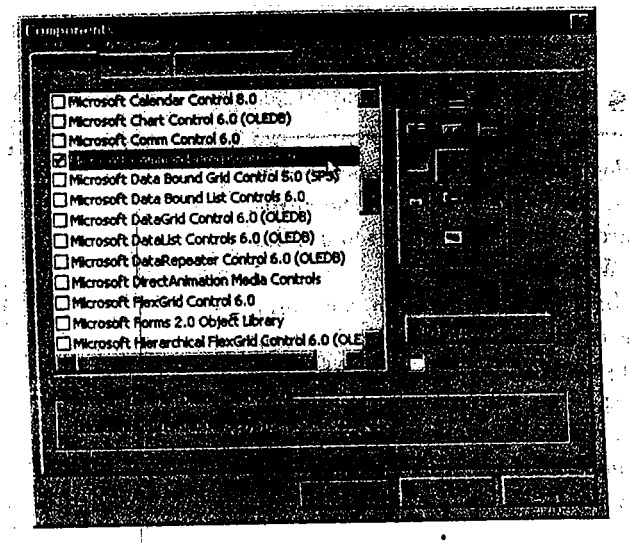
Figure 5.11



Use the common dialog tool in the toolbox to place a control on the form.

The common dialog control may not appear in your toolbox. It is a custom control, and you must add it to Visual Basic before you can use it. Custom controls are stored in files with an extension of .ocx. To use the common dialog, you must have Comdlg32.ocx in your Windows\System folder. Open the *Project* menu and choose *Components*; you should see *Microsoft Common Dialog Control 6.0* in the list (Figure 5.12). Make sure that a check appears next to the item, and its tool should appear in your toolbox.

Figure 5.12



Select the common dialog control in the Components dialog box to make the Common Dialog tool appear in the toolbox.

When you name a Common Dialog control, use "dlg" as its three-character prefix.

Using a Common Dialog Box

After you place a Common Dialog control on your form, you can display any of its dialog boxes at run time. In code you specify which one you want with the Show method.

Show Method—General Form

```
Object.ShowMethod
```

The method can be one of the following:

Dialog Box	Method
Open	ShowOpen
Save As	ShowSave
Color	ShowColor
Font	ShowFont
Print	ShowPrint

For example, assume you have a common dialog control called dlgCommon and a menu item named mnuEditColor. You could display the *Color* dialog box in the Click event for the menu command:

```
Private Sub mnuEditColor_Click()
    'Display the Color Dialog Box

    dlgCommon.ShowColor 'Display the Color dialog box
End Sub
```

The Windows dialog box for choosing colors (Figure 5.13) will appear when the user clicks on the *Color* menu option.

Figure 5.13

The Color common dialog box.

Using the Information from the Dialog Box

Displaying the *Color* dialog box doesn't make the color of anything change. You must take care of that in your program code. What *does* happen is that the user's choices are stored in properties that you can access. You can assign these values to the properties of controls in your project.

Color Dialog Box

The color selected by the user is stored in the Color property. You can assign this property to another object, such as a control or the form.

```
frmMain.BackColor = dlgCommon.Color
```

Because Basic executes the statements in sequence, you would first display the dialog box with the ShowColor method. (Execution then halts until the user responds to the dialog box.) Then you can use the Color property:

```
Private Sub mnuEditColor_Click()
    'Display the Color dialog box

    dlgCommon.ShowColor
    'Assign dialog box color to the form
    frmMain.BackColor = dlgCommon.Color
End Sub
```

When the *Color* dialog box displays, it shows black as the default color. If you want to initialize the box to show the form's currently selected color when it appears, you must set the control's Flags property to cdlCCRGBInit and set its Color property before showing the dialog box.

```
Private Sub mnuEditColor_Click()
    'Select the color and set initial color
```

```
    With dlgCommon
        .Flags = cdlCCRGBInit                    'Initialize the dialog box
        .Color = frmMain.BackColor               'Set initial color
        .ShowColor                               'Display dialog box
    End With

    'Assign dialog box color to the form
    frmMain.BackColor = dlgCommon.Color          'Set color of form
End Sub
```

Font Dialog Box

Before you can use the ShowFont method, you must set the Flags property of the control. This coding step installs the fonts to appear in the list box. The value of the Flags property may be cdlCFScreenFonts, cdlCFPrinterFonts, or cdlCFBoth. If you forget to set the flags, an error will occur at run time that says "There are no fonts installed."

The Font properties for the common dialog control are different from those used for other controls. Instead of a group of properties for a Font object, each Font property is listed separately.

Font Object of Other Controls	Font Property of Common Dialog Control	Values
Font.Bold	FontBold	True or False (Boolean)
Font.Italic	FontItalic	True or False (Boolean)
Font.Name	FontName	System dependent
Font.Size	FontSize	Font dependent
Font.StrikeThrough	FontStrikeThru	True or False (Boolean)
Font.Underline	FontUnderline	True or False (Boolean)

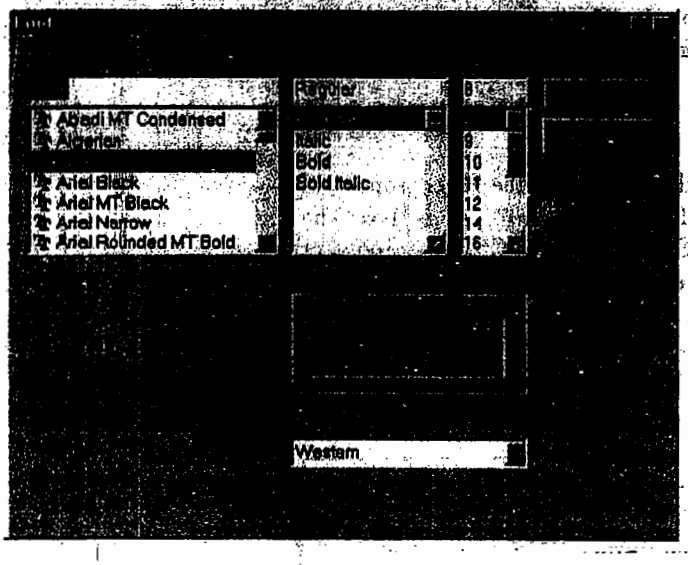
Assign the FontName property before assigning the size and other attributes. The following code allows the user to change the font of a label.

```
Private Sub mnuEditFont_Click()
    'Display the Font dialog box

    With dlgCommon
        .Flags = cdlCFScreenFonts
        .ShowFont
    End With

    'Assign dialog box font to the label
    With lblEmployee.Font
        .Name = dlgCommon.FontName
        .Bold = dlgCommon.FontBold
    End With
End Sub
```

When the user clicks on the *Font* menu command, the *Font* dialog box appears on the screen (Figure 5.14).

Figure 5.14

The Font common dialog box.



Setting Current Values

Before calling the common dialog box for colors or fonts, you should assign the existing values of the object properties that will be altered. This step will provide for the display of the current values in the dialog box. It also ensures that if the user selects the Cancel button, the property settings for the objects will remain unchanged.

```
Private Sub mnuEditFont_Click()
    'Display the Font common dialog box
    '  and make the changes in the label settings

    'Assign the current settings to the dialog box
    With dlgCommon
        .FontName = lblEmployee.Font.Name
        .FontBold = lblEmployee.Font.Bold
        .FontItalic = lblEmployee.Font.Italic
        .FontSize = lblEmployee.Font.Size
    End With
    'Set the method for Font dialog box
    With dlgCommon
        .Flags = cdlCFScreenFonts 'Specify screen fonts
        .ShowFont
    End With
```

Figure 10.10

The InputBox function produces a dialog box with a prompt and a text box for entering program input.

The InputBox Function—General Form

```
VariableName = InputBox("Prompt" [, "Title"] [, Default] [, XPos] [, YPos])
```

The prompt must be enclosed in quotation marks and may include NewLine characters (vbCrLf) if you want the prompt to appear on multiple lines. The Title displays in the title bar of the dialog box; if the Title is missing, the project name appears in the title bar. Any value you place in *Default* appears in the text box when it is displayed; otherwise, the text box is empty. (If *Default* is a string, it must be enclosed in quotation marks.) *XPos* and *YPos*, if present, define the measurement in twips for the left edge and top edge of the box.

The InputBox Function—Examples

```
strName = InputBox("Enter your name.")
intQuantity = InputBox("How many do you want?", "Order Quantity")
```

Using the InputBox to Randomly Retrieve a Record

You will find the input box to be a great tool when you need to retrieve a record from a random file. Many applications use the record number as a method of identification, such as customer number or product number. If you request this number, you can read the correct record in the random file directly.

```
Dim intRecordNumber As Integer
intRecordNumber = Val(InputBox("Enter Customer Number"))
If intRecordNumber >= 0 And intRecordNumber <= LOF(1) / Len(udtCustomer) Then
    Get #1, intRecordNumber, udtCustomer
Else
    MsgBox "Invalid Customer Number"
End If
```

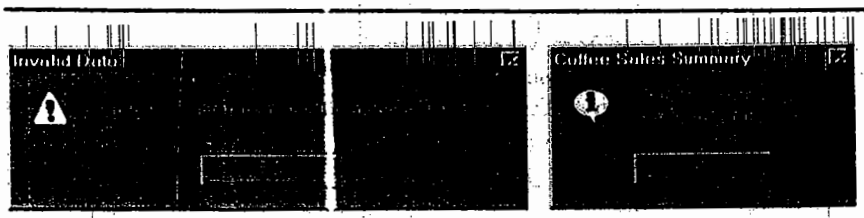
that you have check boxes for Discount, Taxable, and Delivery. You will need separate If statements for each condition.

```
If chkDiscount.Value = Checked Then
    'Calculate the discount
End If
If chkTaxable.Value = Checked Then
    'Calculate the tax
End If
If chkDelivery.Value = Checked Then
    'Deliver it
End If
```

Displaying Messages in Message Boxes

A **message box** is a special type of Visual Basic window in which you can display a message to the user. You can display a message, an optional icon, a title bar caption, and command button(s) in a message box (Figure 4.8).

Figure 4.8



Two sample message boxes created with the MsgBox *statement.*

You may want to display a message when the user has entered invalid data or neglected to enter a required data value. Later in this chapter you will see several techniques for checking for valid input, called *validating* input data.

The MsgBox Statement—General Form

```
MsgBox "Message string" [, Buttons/icon][, "Caption of title bar"]
```

The *Message string* is the message you want to appear in the message box. The *Buttons* portion is optional; it determines the command buttons that will display and any icons that will appear. If you omit the *Caption of title bar*, the project name will appear in the message box title bar.

The MsgBox Statement—Example

```
If txtName.Text = "" Then
    MsgBox "Please Enter your name.", vbOKOnly, "Name Missing"
End If
```

Selecting the MsgBox Icon

For the button/icon entry, you can choose to use the numeric values in the following table or to use the Visual Basic constant. For example, you can display the Warning Query icon with either of these statements:

```
MsgBox "Let this be a warning", vbQuestion, "Error"
```

or

```
MsgBox "Let this be a warning", 32, "Error"
```

Button/Icon	Value	Constant
OK button	0	vbOKOnly
Critical Message icon	16	vbCritical
Warning Query icon	32	vbQuestion
Warning Message icon	48	vbExclamation
Information Message icon	64	vbInformation

Note: `MsgBox` can be used as a statement as explained here, which displays only an OK button. It can also be used as a function. When you use `MsgBox` as a function, you can choose the buttons to display (such as Yes, No, Cancel; or OK and Cancel). The function returns a value indicating which button was pressed. The `MsgBox` function is covered in Chapter 7.

Displaying a Message String

The message string you display may be a string literal enclosed in quotes or it may be a string variable. You may also want to concatenate several items, for example, combining a literal with a value from a variable. If the message you specify is too long for one line, Visual Basic will wrap it to the next line.

Combining Values into a Message String

You can concatenate a literal such as "Total Sales" with the value for the total sales:

```
Dim strMessage As String

strMessage = "Total Sales" & mcurTotalSales
MsgBox strMessage, vbOKOnly, "Sales Summary"
```

This example does not format the number. To remedy this condition, consider formatting the number before concatenating it to the string.