DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA, UNAM
CURSOS ABIERTOS

CURSO: **CC65 VISUAL BASIC 3.0 AVANZADO**
FECHA: **1 al 12 de julio de 1996**

EVALUACIÓN DEL PERSONAL DOCENTE

(ESCALA DE EVALUACIÓN: 1 A 10)

CONFERENCISTA	DOMINIO DEL TEMA	USO DE AYUDAS AUDIOVISUALES	COMUNICACIÓN CON EL ASISTENTE	PUNTUALIDAD
ING. ALMA ANDREA GARCIA OLEA				
ING. MARTHA OCAMPO ORTIZ				

Promedio _____

EVALUACIÓN DE LA ENSEÑANZA

CONCEPTO	CALIF.
ORGANIZACIÓN Y DESARROLLO DEL CURSO	
GRADO DE PROFUNDIDAD DEL CURSO	
ACTUALIZACIÓN DEL CURSO	
APLICACIÓN PRACTICA DEL CURSO	

Promedio _____

EVALUACIÓN DEL CURSO

CONCEPTO	CALIF.
CUMPLIMIENTO DE LOS OBJETIVOS DEL CURSO	
CONTINUIDAD EN LOS TEMAS	
CALIDAD DEL MATERIAL DIDÁCTICO UTILIZADO	

Promedio _____

Evaluación total del curso_____

Continúa...2

1. ¿Le agradó su estancia en la División de Educación Continua?

SI ____ NO ____

Si indica que "NO" diga porqué: _____

2. Medio a través del cual se enteró del curso:

Periódico *Excélsior*	
Periódico *La Jornada*	
Folleto anual	
Folleto del curso	
Gaceta UNAM	
Revistas técnicas	
Otro medio (Indique cuál)	

3. ¿Qué cambios sugeriría al curso para mejorarlo?

4. ¿Recomendaría el curso a otra(s) persona(s) ?

SI ____ NO ____

5. ¿Qué cursos sugiere que imparta la División de Educación Continua?

6. Otras sugerencias:

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

COMPLEMENTO DE MATERIAL DIDACTICO

DEL CURSO :

V I S U A L B A S I C

A V A N Z A D O

JULIO, 1996

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Teléfonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 512-5121 521-4020 AL 26

Lab 1: Review

Lab Objectives

The purpose of this lab is to give you a chance to become familiar with Microsoft® Visual Basic™ programming system, to see how your computer has been configured, and to ensure that everyone has the requisite base set of Visual Basic skills.

By the end of this lab, you will be able to:

- Add a VBX file to a project.
- Add a module to a project.
- Write and call a user-written function.
- Write and call a user-written subroutine.
- Place controls on a frame.
- Load and unload a form.
- Build an .EXE file.
- Run a program from Microsoft Windows™ File Manager.
- Customize AUTOLOAD.MAK.

Objects and Subroutines

The partial solution already has the form designed. The following controls are provided for you.

Control name	Purpose
frmConversion	The primary form.
txtMiles	Text box for miles
txtKilometers	Text box for kilometers
cmdReset	Command button to reset
cmdConvert	Command button to convert
frmAbout	The form to be displayed when user chooses About from the Help menu

You will need to add the following controls. The lab instructions will specify details; the list is just provided here for your reference.

Control name	Purpose
fraConvert	Frame to hold options
optEnglish	English to metric option
optMetric	Metric to English option
spnMiles	Spin control next to miles
spnKilometers	Spin control next to kilometers

The following functions are provided for you.

Procedure name	Purpose
MilesToKilometers	Converts miles to kilometers
KilometersToMiles	Converts kilometers to miles
ResetEverything	Resets text boxes to 0; resets option to English To Metric

Exercise 1: Reviewing the Solution

This program converts miles to kilometers and vice versa. The program contains nothing that is beyond the class's prerequisites.

► **Run the lab solution in \VBLABS\REVIEW\SOLUTION**

Test the program by doing the following:

1. Type a number in the Miles text box and choose Convert.

   The corresponding number of kilometers appears in the Kilometers text box.

2. Click the top portion of the spin control to the right of the Miles box.

   The numbering in the Miles box is incremented by 1; the number in the Kilometers box also changes. Clicking the bottom portion of the spin control decreases Miles, with a corresponding decrease in Kilometers. The spin control is a custom control that was added to the project.

3. Click the spin control to the right of the Kilometers box.

   Nothing should happen. The spin control for Kilometers is disabled when the English To Metric option is specified.

4. Choose the Metric To English option and click the Kilometers spin control.

   The appropriate value is displayed in the Miles box. You can also now use the Kilometers spin control. The spin control for Miles is disabled.

5. Choose Reset.

   Everything returns to the way it was when the program started.

6. From the Help menu, choose **About**.

   A form pops up telling something about this program.

When you're satisfied you know how the program works, open the project PREVIEW.MAK in \VBLABS\REVIEW\PARTIAL and go on to the next exercise.

Exercise 2: Writing the Application from the Partial Solution

To save typing time, the partial solution already has the two forms and some controls created for you. You will need to add some controls and add the code.

▶ Add the custom control SPIN.VBX to the project and place two spin controls appropriately.

1. From the File menu, choose Add file.
2. Locate the SPIN.VBX file in the \WINDOWS\SYSTEM directory.
3. Place one spin control adjacent to the Miles box, and name it spnMiles.
4. Place one spin control adjacent to the Kilometers box, and name it spnKilometers.
5. Set the Enabled property of spnKilometers to **False**.

▶ Add the frame with the two option buttons to the form

1. Place a frame on the form, and name it fraConvert.
2. Explicitly draw the two option buttons on the frame.

   Do not use the double-click method to place the option button on the frame. Double-clicking automatically places a control on the form, not the frame, even though it may appear otherwise.

   Use the following names and captions.

Control name	Control caption
fraConvert	
optEnglish	English To Metric
optMetric	Metric To English

3. Set the Value property of optEnglish to **True**.
4. Code the optEnglish_Click option to disable the spnKilometers control and enable the spnMiles control.
5. Code the optMetric_Click option to disable the spnMiles control and enable the spnKilometers control.

▶ Add a module to the project

This module is going to hold two functions that will be used in the program.

1. From the File menu, choose New Module.
2. Choose Save File As from the File menu, move to the C:\VBLABS\REVIEW\PARTIAL directory, and save the module as REVIEW.BAS.

▶ Write the MilesToKilometers function

This function should accept a single argument named miles and return a single-precision variable that is the value of miles expressed in kilometers.

a. Double-click the REVIEW.BAS in the Project window to open a Code window if it is not already open.

b. With the Code window open, choose New Procedure from the View menu.

c. Select the Function option.

d. Type MilesToKilometers in the name box.

e. Choose OK.

f. Edit the function as follows:

```
Function MilesToKilometers (miles As Single) As Single
   MilesToKilometers = miles * 1.609
End Function
```

Another way to create a new function is to just type the function or subroutine heading from anywhere in the code and press ENTER. Visual Basic automatically creates the new procedure stub.

▶ Write the KilometersToMiles function

Perform the same steps to create another function that will convert kilometers to miles. Name this function KilometersToMiles. The formula to convert kilometers to miles is:

```
Function KilometersToMiles (Kilometers As Single) As Single
   KilometersToMiles = Kilometers * .6214
End Function
```

▶ On the form frmConversion, code the cmdConvert command

- In the cmdConvert_Click function, write code that determines which option button has been selected (the value of the option button will be **True** if it has been selected) and then calls the appropriate conversion function.

  For instance, if optEnglish is **True**, then invoke MilesToKilometers and store the result in the txtKilometers box.

  Remember that MilesToKilometers returns a number. You need to convert the number to a string before you can assign it to a text box.

▶ Write the ResetEverything routine

In the Declarations section for frmConversion, write a subroutine named ResetEverything that:

1. Sets the two text boxes to "0".
2. Sets the option English To Metric.
3. Sets the focus to txtMiles.

   Because this function is necessary only on this form, it is most appropriate as a form-level subroutine.

4. Invoke the subroutine ResetEverything from cmdReset_Click.

▶ Code the Spin Up Buttons

Code the spnMiles_SpinUp and spnKilometers_SpinUp functions to automatically add 1 to the Miles or Kilometers text box and invoke the cmdConvert_Click event.

For example, in spnMiles_SpinUp you would:

1. Convert txtMiles.Text to an integer value using the **Val** function.
2. Add 1 to the results of the conversion.
3. Convert that result back to a string using the **Str$** function.
4. Assign the new string to txtMiles.Text.
5. Invoke the cmdConvert_Click procedure that you already coded to convert the value in txtMiles to kilometers and display the results in the Miles box.
6. Copy this code to spnKilometers_SpinUp, and make the necessary changes.

▶ Code the Spin Down Buttons

Code the spnMiles_SpinDown and spnKilometers_SpinDown functions to subtract 1 from the appropriate text box and invoke the cmdConvert_Click function. *Do not allow the value in txtMiles or txtKilometers to fall below 0.*

▶ Design the menu interface

Design the menu interface as follows.

Menu caption	Name	Indentation
File	mnuFile	Not indented
Exit	mnuExit	Indented once
Help	mnuHelp	Not indented
About...	mnuAbout	Indented once

▶ Code the menu interface

1. Code the mnuExit_Click event to end the application.
2. Code the mnuAbout_Click event to load and show the form frmAbout modally.
3. On the form frmAbout, you'll find a command button cmdOK. Code that command button to unload frmAbout.

▶ Set Picture Properties on frmAbout

On the form frmAbout, you'll also find two image controls: image1 and image2. Set the picture property of these images to \VB\ICONS\FLAGS\FLGUK.ICO and \VB\ICONS\FLAGS\FLGFRAN.ICO, respectively.

▶ Test your program

Save and test your program. When you're satisfied it runs correctly, make an .EXE file and run it from the Windows File Manager. If you have trouble doing this, ask the instructor for help. It's important that you be able to do this for some of the remaining labs in the class.

Exercise 3: Optional...

1. There's a file named AUTOLOAD.MAK in \VB. It's an ASCII file and can be edited with a simple text editor like Windows Notepad. The purpose of AUTOLOAD.MAK is to cause Visual Basic to always include certain files in a project. Typically, you'll insert a particular custom control in AUTOLOAD.MAK. Using Notepad, add this line to the beginning of your computer's AUTOLOAD.MAK file.

   ```
   c:\windows\system\cmdialog.vbx
   ```

   Then try New Project from the Visual Basic File menu, and examine the Project Window. The version of AUTOLOAD.MAK on your computer may have been modified from the default one installed by Visual Basic. The default version of AUTOLOAD.MAK installed most of the custom controls. Consult the documentation for more ways you can customize AUTOLOAD.MAK.

2. Add another frame with two option buttons in it that are Distance and Volume. The program currently does only distance conversions. If the user selects Volume, change the titles of the two text boxes to Gallons and Liters. Add two more subroutines to REVIEW.BAS to perform these conversions, and call them from cmdConvert instead of the two you're currently calling. The cmdConvert function will have to figure out not only which way the conversion is going (English to Metric or vice-versa) but also which type of conversion is in effect, Distance or Volume.

   The formulas you'll need are:

   ```
   gallons = liters * .26423
   liters = gallons * 3.7853
   ```

3. If you still have time, extend what you did above to deal with mass. The formulas you'll need are as follows:

   ```
   pounds = kilograms * .4536
   kilograms = pounds * 2.2046
   ```

Lab 2: Input Validation

Lab Objectives

By the end of this lab you will be able to:

- Use available techniques to verify that the data in the controls is correct and complete.
- Set the focus back to an invalid control and display an appropriate error message.

Objects and Subroutines

The following controls are provided for you.

Control name	Purpose
frmInput	The main input form
lblName	Label with caption "Name"
mskName	Masked edit control for name
lblSSN	Label with caption "SSN"
mskSSN	Masked edit control for SSN
lblYears	Label with caption "Years"
mskYears	Masked edit control for years employed
cmdOK	OK button
cmdCancel	Cancel button
cmdReset	Reset button
lblError	Label for error message

Parts of the following routines are provided. You will have to modify the routines. The details are in the instructions.

Routine name	Purpose
DisplayErrorLabel	Displays an error message in lblError: accepts a string argument: sets lblError.Caption to the argument
ValidateOKButton	Determines when to enable the OK button: invoked from the Change event of the masked edit controls

Exercise 1: Testing the Solution

This application is a sample data entry form. Each control has restricted data.

Run the completed lab solution in \VBLABS\USRINPUT\SOLUTION.

When the program begins, you'll see three masked edit controls and three command buttons. Test the program by doing the following:

1. Type at least one character or number in all three masked edit controls.

   The OK button is enabled.

2. Delete data from one of the controls.

   The OK button is disabled.

3. Try to type a letter in Years Employed.

   An error message is displayed.

4. Type a number in Years Employed.

   The error message is cleared.

5. Try to type more than two digits in Years Employed.

   The appropriate error message is displayed.

6. Enter data in all the controls, and choose OK.

   The controls are verified and focus is forced to any control that has invalid contents.

When you're satisfied you know how the program runs, open the partial solution project in \VBLABS\USRINPUT\PARTIAL and go on to Exercise 2.

Exercise 2: Declaring the Array

- The program uses an array of integers to determine when to enable the OK button and to determine whether a control is valid.
  - In the Declarations section of the form INPUT.FRM, declare two integer arrays as follows

```
Dim EnableOkFlags(MSK_COUNT)
Dim ValidDataFlags(MSK_COUNT)
```

Exercise 3: Setting Properties

1. Be sure you have opened the partial solution in \VBLABS\USRINPUT\PARTIAL.

   Note the presence of MSMASKED.VBX in the Project window. The custom control has already been added to the project for you. Three masked edit controls have been added to the form. Set the mask property of each of the controls as follows:

   a. Set mask of mskName to 25 ampersands (&).

   b. Set mask of mskSSN to ###-##-####.

   c. Set the AutoTab property of mskSSN to **True**.

      This enables the user to automatically move to the next control when all nine digits of the SSN have been typed.

   d. Set mask of mskYears to ##.

   e. Assign 0, 1, and 2 to the Tag properties of mskName, mskSSN, and mskYears, respectively. These will be used as subscripts into the EnableOKFlags array later in the application.

2. Run the application. Try to type invalid characters in any of the controls. What happens?

   Remember, at run time any violation of the mask (such as typing a letter when the mask specifies a digit) immediately invokes the ValidationError event for that particular control. You must, however, provide code in the ValidationError event to display an appropriate error message.

   Masks work best for numeric control or for text control of fixed length. Even though Name and Years Employed are not fixed-length controls, as Social Security Number is, masks are used in this program because the ValidationError that occurs when the mask is violated is a convenient place from which to display error messages. The MaxLength property of an ordinary text box does nothing other than beep.

Exercise 4: Displaying Error Messages

1. Display an error message if the user types more that 25 characters in mskName; clear the error message if the user then presses BACKSPACE.
   a. In mskName_ValidationError event if the StartPosition argument is equal to the Maxlength property of mskName, invoke the subroutine DisplayErrorLabel and pass the string "Name limited to 25 characters."
   b. In mskName_KeyPress event, set lblError Visible to **False**. You want to clear the error message when the user types the next character.
   c. In mskName_LostFocus, set lblerror Visible to **False**.
2. Display and clear the appropriate error message for the SSN control.
   a. In mskSSN_ValidationError event invoke DisplayErrorLabel and pass the string "SSN must be filled with 9 digits."
   b. In mskSSN_KeyPress event, set lblError Visible to **False**.
   c. In mskSSN_LostFocus event, set lblError Visible to **False**
3. The code to handle displaying and clearing error messages for Years Employed has already been provided for you.
4. Run your program. Verify that error messages are displayed and cleared properly.

Exercise 5:
Enabling the OK Button when Controls are Completed

The program uses an array of integers (EnableOKFlags) to determine when to enable the OK button.

1. In the Change event for each of the masked edit controls, invoke the ValidateOKButton subroutine and pass the control name as an argument. The ValidateOKButton subroutine is in the general declarations section of INPUT.FRM.
2. You need to code the ValidateOKButton subroutine to check the contents of the incoming control, set the appropriate flag in the EnableOKFlags array to **True** if the control has data and **False** if it does not have data, and then scan the entire array and enable the OK button if all the members of the array are **True**.

   This subroutine uses the Tag property of the incoming control as a subscript into the array.

   a. Assign the value of the control's ClipText property to the variable TextWOmask (*Fix #1) (this will hold the text without the mask).

   b. Convert the value of the Tag property to a number and store it in the variable named index (*Fix #2).

   c. Set the appropriate member of the EnableOKFlags array to **True** if textWOmask has some data and **False** if it does not contain data (*Fix #3).

   d. Finally, if any one of the three "flags" in EnableOKFlags is **False**, disable the OK button and exit the subroutine. If all the flags are **True**, enable the OK button (*Fix #4).
3. Run the application. Verify that the OK button is enabled when all of the controls have data. The OK button should be disabled if you then delete the data in any of the controls.

Exercise 6: If You Have Time...
Determining Whether Controls Are Valid When User Chooses OK

The application uses an array of integers (ValidDataFlags) to determine whether a control is valid.

From the LostFocus event for each of the masked edit controls, invoke the subroutine DetermineInputValidity and pass the control name as an argument. The DetermineInputValidity subroutine sets the appropriate member of the ValidDataFlags array to **True** if the control has valid data.

The OK button will then scan the array ValidDataFlags, set the focus back to any invalid control, and display an appropriate error message. You will need to do the following:

1. In the Declarations section of the form INPUT.FRM, declare an array of integers named ValidDataFlags. Use the constant MSK_COUNT to size the array.
2. Invoke DetermineInputValidity from the LostFocus event of each masked edit control.
   - In the LostFocus event for each of the three masked edit controls, call the subroutine DetermineInputValidity, and in each case pass the control as an argument.
3. Code the DetermineInputValidity function to set the flags in the ValidDataFlags array. The variable TextWOMask is set for you to contain the text portion without the mask of the current masked edit control. Use this variable when testing for valid data.
   a. Look at the code in DetermineInputValidity for Case 0. This code has been provided for you. Case 0 corresponds to Name. The code sets the flag in the array ValidDataFlags to **True** if the Name is not null, **False** if it is null.
   b. Case 1 corresponds to SSN. Set the flag in the array ValidDataFlags to **True** if the length of TextWOMask is 9, **False** if it's not.
   c. Case 2 corresponds to Years Employed. If the value of Years Employed is between 0 and 50 inclusive, set the appropriate flag to True; otherwise, set it to **False**.
4. Code the cmdOK_Click event to loop through the array ValidDataFlags to find the first **False** integer. Then, set the focus to the control that's invalid. If there are no invalid controls, display a message box stating that all controls are valid.
5. Test your application.

Exercise 7: If You Have Time...

1. Enable your users to use the ENTER key to move from one control to the next. In other words, ENTER and TAB should do the same thing. This requires that you:
   a. Set the KeyPreview property of the form INPUT.FRM to **True.**
   b. Write the code in the form's KeyPress event to detect the ENTER key (ASCII 13).
   c. Set the focus to the next control.

      The easiest way to do this is to write a **Select Case** statement that sets the focus to a particular control. The **Select Case** statement is based on the value of Screen.ActiveControl.TabIndex. When the TabIndex of the active control is 1 (that is, the Name control), set the focus to SSN. Make a similar transition for SSN and Years Employed. When the TabIndex of the current control is any value other than 1, 3, or 5, ignore it.

      Don't forget to consume the keystroke (set KeyAscii to 0) when you do trap the ENTER key.
2. Code the application so the user cannot run two copies of it at the same time.
3. Change the Help form to display the program's name and where it is located on the hard disk.

Lab 3: Coding for Mouse Events

Lab Objective

The purpose of this lab is to add drag and drop capabilities to the BATCHXL application. A user should be able to drag a filename from the file list box to the Files To Print list box. A file icon should be used when the user is dragging the file over a target area. The No symbol should be used when the user is dragging the file over an area that is not a target zone.

Objects and Subroutines

The function GetPath is provided for you. This function correctly builds a path. It takes into account whether the current directory is the root.

Exercise 1: Testing the Solution

Run the solution in C:\VBLABS\MOUSE\SOLUTION.

Drag a filename from the file list box to the Files To Print list box. When you release the mouse, the filename is added to the list box. Notice when the file icon appears and when the No symbol appears.

When you're satisfied you know how the program runs, open the partial solution in C:\VBLABS\MOUSE\PARTIAL and move on to Exercise 2.

Exercise 2: Adding Drag and Drop Capabilities

1. You need to add two picture boxes to store the icons that will be displayed as a filename is being dragged. The picture boxes themselves will be hidden. They will be used only to hold a file icon and a No symbol.
   - Add two picture boxes, and name them picNo and picFile.
2. For both picture boxes, set the following properties:

   Autosize = True
   Visible = False
   TabStop = False
   picNo.Picture = C:\VBLABS\MOUSE\NO.ICO (No symbol)
   picFile.Picture = C:\VBLABS\MOUSE\FOLDER01.ICO (file icon)
3. Set filList.DragIcon to C:\VBLABS\MOUSE\FOLDER01.ICO. This causes the file icon to be displayed when you drag the filList control.
4. To allow programmatic control of dragging, set filList.DragMode to Manual.

   It is better to set DragMode to Manual than to Automatic in this case. If you set DragMode to Automatic, dragging begins every time you click the mouse. Therefore, you would not be able to select a filename by clicking the filename.

5. Because DragMode is Manual, you need to specifically indicate when dragging should occur.

   - In the filList_MouseDown event, add the statement filList.Drag.

   Dragging always stops automatically when the mouse is released, so you do not have to add code to stop dragging.

6. In the filList_DragOver event and the lstPrint_DragOver event set the filList.DragIcon to either a No symbol or a file icon. Add the following code to both event procedures.

   a. When the source is being dragged onto the control, you should set filList.DragIcon to the file icon.

   b. When the source is being dragged out of the control, you should set filList.DragIcon to the no symbol. You can accomplish this with a **Select Case** statement, as follows:

```
Select Case State
   Case ENTER          'Source is entering control
      filList.DragIcon = picFile.Picture
   Case LEAVE          'Source is leaving control
      filList.DragIcon = picNo.Picture
End Select
```

   ENTER and LEAVE are provided for you in the Declarations section of the form.

7. When the user drops the file in the Files To Print list box, add the file currently selected in the file list box to the Files To Print list box.

   - Add the following statement to the lstPrint_DragDrop event:

```
Dim Path As String
Path = GetPath()
lstPrint.AddItem Path + filList.FileName
```

   GetPath is a function we have written for you. GetPath returns the current path and correctly adds a backslash if necessary.

8. Save the project, and test your work.

Exercise 3: If You Have Time... Changing the Mouse Pointer

- When the user selects the Print button, set the screen mouse pointer to the hourglass. Have it hold that shape for an interval that is a product of the number of filenames in the lstPrint control times 1000. Thus five files would mean a five-second delay.

  You'll need a timer that is disabled at design time on the form. In the Print button, set the timer interval to the value derived in the previous paragraph, change the shape of the mouse pointer, and enable the timer.

  Disable the timer and restore the mouse pointer in the timer's event procedure.

Lab 4: File Input and Output

Lab Objective

This lab is an introduction to a program that will extend over several different labs. In addition to file-handling statements, you'll get familiar with the program so that in later versions you'll spend less time learning where variables are defined and so on.

By the end of this lab, you will be able to:

- Create an application that reads and writes data files.

Exercise 1: Testing the Solution

This application allows you to open a file, edit it, and save it.

Run the solution in \VBLABS\FILEIO\SOLUTION. Test the solution by doing the following:

1. Open a file (there should be test files available in the \VBLABS\ directory), make some changes, and save the file.
2. Open a file again, make some changes, and try to exit without saving the file.

   The application should prompt you to save changes.
3. Restart the application. Do not open a file: just start typing text in the text box, and then try to exit.

   The system should prompt you to save the file and should prompt you for a filename.

When you're satisfied you know how the program works, open the partial solution in \VBLABS\FILEIO\PARTIAL and go on to Exercise 2.

Exercise 2: Initial Setup for the Partial Solution

Be sure you are in the partial project in \VBLABS\FILEIO\PARTIAL.

1. The form has already been created for you. In the Declarations section, add these form-level variables:

   lcv as an integer

   dirtyflag as an integer

   fhandle as an integer

   fname as a string

   Note the constant declared in the Declarations section.

2. You will use the common dialog control to open and save a file.

   a. Add the custom control CMDIALOG.VBX to the project.

      This control is located in the \WINDOWS\SYSTEM directory.

   b. Place this control anywhere on the form; use the default name.

3. In the Form_Load event, position the form set the WindowState of the form to 2.

   Note the statement that sets the common dialog control's filter. If you wanted your text editor to always work with files with a particular extension, you'd change this.

4. The variable dirtyflag will determine whether a file needs to be saved.

   - In txtEditBox_Change, set dirtyflag to **True**.

5. You'll want to keep the size of the text box equal to the size of the form but allow room for scroll bars on the text box. Examine the statements in the Form_Resize event that do this.

Exercise 3: Coding the File Open Command

Code the mnuFileOpen_Click event to do the following:

1. Invoke the File Open dialog box by setting the Action property of the common dialog control to 1 (Fix #1).
2. Assign the dialog control's Filename property to fname (Fix #2).

   Notice the code that is provided beeps and exits the subroutine if fname is empty (meaning the user didn't select a filename).
3. Assign the form's Caption property like this (Fix #3):

```
frmMP3E.Caption = FormTitle + " (" + CMDialog1.FileTitle + ")"
```

   FileTitle is just the filename, whereas the Filename property is a fully-qualified filename.
4. Get a fresh value into fhandle by capturing the return value of FreeFile (Fix #4).

   You declared fhandle earlier in the form's Declarations section.
5. Open fname for input with fhandle as the file handle (Fix #5).
6. Read the file in a single statement using the **Input$** statement (Fix #6).
7. Set the dirtyflag to **False** (Fix #7).

   The dirtyflag is used to determine whether the file needs to be saved before the user can exit. Because assigning the file's contents to the text box caused a change event for that text box and because the Change event set dirtyflag to **True**, you need to set the dirtyflag to **False**.
8. Close the file (Fix #8).

Exercise 4: If you have time: Coding the File Save Commands

1. Code the mnuFileSave_Click event to do the following:
   a. Exit the subroutine if dirtyflag is **False**.
   b. Open a file for output.
   c. Print the contents of the text box to the file.
   d. Close the file.
   e. Set the dirtyflag to **False** because the file has just been saved.
2. Code the mnuFileSaveExit_Click event to:
   a. Invoke the mnuFileSave_Click function. This is easier than duplicating its code.
   b. Terminate the application.
3. To be sure that the file is always saved when the user tries to exit the application, invoke the mnuFileSave_Click procedure from the form's QueryUnload event.

Exercise 5: If You Have Time...

1. Add a Save As command to the menu to permit saving the file under another name. You want to be able to tell the user if the new name already exists. The function **Dir$** can be used to tell whether a filename already exists.
2. Many editors have a backup feature, whereby the original file is saved with the extension .BAK. Add a feature to ask the user immediately before saving whether a backup is desired or even do it automatically. If the user wants a backup, you can copy the original file to a file with the .BAK extension. Do this *before* you write the text box back to the original file. You have to build the backup name yourself using string functions. Visual Basic has added a function named **FileCopy** to facilitate doing something like this. **FileCopy** is equivalent to typing **copy** at the Microsoft® MS-DOS® command prompt.

Lab 5: Dynamic Controls

Lab Objectives

After completing this lab, you will be able to:

- Create a control array.
- Dynamically add and delete elements from a control array.

The following controls are provided for you:

Control name	Purpose
txtEditBox	The editing box to hold file
Label1	The label for the buffer
Text1	The paste buffer

Exercise 1: Testing the Solution

This program is an extension of the simple text editor from the previous lab. The editor now has a paste buffer and menu commands to work with that paste buffer. Furthermore, this application will permit the user to have up to four paste buffers. This is where you'll find an array of controls. Each paste buffer is itself editable.

Run the lab solution in C:\VBLABS\CONTROLS\SOLUTION. Test the program by doing the following:

1. From the Buffers menu, choose Add to add several paste buffers.
2. Open a file, highlight some text, and choose Copy To Buffer #1 from the Edit menu.

   The text is copied to the appropriate buffer.
3. Try to copy to a buffer that does not exist.

   The application should beep and display a message.

When you're satisfied you know how the program works, open the project in \VBLABS\CONTROLS\PARTIAL and go to Exercise 2.

Exercise 2: Setting the Properties

Be sure you have opened the partial project in \VBLABS\CONTROLS\PARTIAL.

- The form already has the editing text box, the common dialog control, one label and one paste buffer.

  You'll need to make the labels and buffers into arrays.

  Set the Index properties of Label1 and Text1 to 1.

  Don't worry about sizing or positioning them; you will take care of that in code.

Exercise 3: Coding the Add Buffer Command

1. You need a variable to track the number of paste buffers.
   - In the form's Declarations section, declare an integer named pastecount.
2. In the mnuBufferAdd_Click event, you need to add a new label and a new text box. Add code to do the following:
   a. Load another label using pastecount as the index (Fix #1).
   b. Set the new label's Top to its predecessor's Top + DELTA (Fix #2).
   c. Set the new label's Caption to "Paste Buffer #" + pastecount (Fix #3).
   d. Set the new label's Visible property to **True** (Fix #4).
   e. Load another text box (Fix #5).
   f. Set the new text box's Top to its predecessor's Top + DELTA (Fix #6).
   g. Set the new text box's Visible property to **True** (Fix #7).
   h. Set the new text box's Text property to an empty string. (Fix #8).
3. Run the application, and verify that buffers are added appropriately.

Exercise 4: Coding the Remove Buffer Command

1. In the mnuBufferRemove_Click event, add code to do the following:
   a. Unload the highest label.
   b. Unload the highest text box.
   c. Decrement pastecount.
2. Run the application, and verify that the buffers are deleted appropriately.

Exercise 5: Coding the Edit Menu

On the Edit menu, there are four copy and four paste menus items. The Copy commands copy text from txtEditBox to a buffer. The Paste commands copy text from a buffer to txtEditBox.

The only difference between the commands is which buffer to use. Therefore this is an ideal situation for a general subroutine. The subroutine can be written to accept an argument that indicates the appropriate buffer. You will then just invoke the subroutine from the menus and pass an integer to indicate the buffer number.

For example, from mnuEditCopy 1 you would invoke:

```
my_Copy 1
```

(In the next module you'll learn about menu arrays, which will give you a cleaner way of doing this.)

You'll have to create the two general subroutines my_Copy and my_Paste. The stubs are already available in the Declarations section.

1. Insert a call to my_Copy from each of the copy menu events, and pass the appropriate argument (1, 2, 3, or 4). The Paste menu code has already been provided for you.
2. Add code to my_Copy to do the following:
   a. Check the incoming argument to verify that it's less than or equal to pastecount. If it's not, beep and issue a message and exit the subroutine.
   b. Verify that Screen.ActiveControl is txtEditBox. If it's not, issue a message and exit the subroutine.
   c. Finally, if everything's okay up to this point, assign the selected text from txtEditBox to the appropriate buffer. The incoming argument tells you which paste buffer to use.
3. Add code to my_Paste to do the following:
   - Assign the contents of the appropriate buffer to txtEditBox.SelText. Again, the incoming argument tells you which paste buffer.

Exercise 6: If You Have Time...

Write the logic so that the paste buffers, regardless of how many there are, take up the maximum amount of room on the right side of the screen. For instance, if there's just one paste buffer, it uses the entire right side. Two paste buffers split the right side, three divide it into thirds, and four divide it into fourths.

Lab 6: Menus

Lab Objectives

After completing this lab, you will be able to:

- Enable, check, and hide menu items as needed.
- Dynamically add and delete menu items.

Exercise 1: Testing the Solution

This application is yet another variation on the multiple paste buffer editor; but this time, among other things, the menus change to reflect the number of paste buffers available. The previous lab always displayed the same menus, regardless of the number of paste buffers.

Run the solution in \VBLABS\MENUS\SOLUTION, and do the following:

1. Note the appearance of the Edit menu with just one paste buffer.
2. Add a paste buffer or two, and look at the Edit menu again.
3. Load a file, select some text in it, and look at the Edit menu again.

   Cut and Copy are enabled when some text is selected.
4. Copy something to one of the paste buffers; and look at the Edit menu.

   Paste is enabled if there is something in the buffer.
5. From the Options menu, choose Use Single Buffer.

   Paste buffer 1 expands, the others disappear, the Single Buffer menu item is checked, and the Buffers menu disappears.
6. From the Options menu, choose Use Multiple Buffers.

   This restores the Buffers menu and checks the Multiple Buffers menu item.

   When you're satisfied you know how the application works, open the partial project in C:\VBLABS\MENUS\PARTIAL and move on to Exercise 2.

Tip As you begin this lab, keep in mind one thing beginners have difficulty remembering about dynamic menus in Visual Basic: Typically, it's in the click event of the top-level menu item that you determine the appearance of the menu that is just about to drop down.

Exercise 2: Assigning Properties

1. The first thing to do is to make the three menu items into arrays.
   - In the Menu Design Window, set the indexes of mnuEditCut, mnuEditCopy, and mnuEditPaste to 1.
2. The program should start with the multiple buffer options enabled.
   - In the Menu Design Window, select the Checked property of mnuOptionsMultiple.

Exercise 3: Loading and Unloading Menu Items

1. In the mnuBufferAdd_Click event, you need to load not only a text box and a label, as in the previous lab, but also menu items.
   - Insert the statement to load an additional mnuEditCut menu item, set its caption, and enable it.

     The statements to do the same for mnuEditCopy and mnuEditPaste are already provided. Use them as a model.
2. In the mnuBufferRemove_Click event, add code to:
   - Unload the highest menu items from the menu arrays mnuEditCut, munEditCopy, and mnuEditPaste.

     The statement to decrement pastecount is provided.
3. Run the application. Verify that when you add a buffer, the appropriate menu is added.

Exercise 4: Enabling and Disabling Menu Items

1. In the mnuEdit_Click event you determine whether the Paste, Cut, and Copy commands that are visible are to be enabled.
   a. In the first For loop, enable a Paste command only if the length of its corresponding paste buffer is not 0.

      The **Len** function comes in handy here.

   b. Set the Enabled property of the Cut and Copy commands to the value of selectionmade.

      The variable selectionmade has already been set to **True** or **False**, depending on whether something is highlighted in the editing text box.
2. Suppose all four paste buffers are visible. Then the Edit menu will have 12 selections (Cut, Copy, and Paste * 4). When the user chooses a menu item that is enabled, the Click event for that menu item has an integer as its argument. You can use this integer as the subscript to tell you which paste buffer to work with.
   - In the mnuEditCut_Click event, insert the statement to assign the selected portion of txtEditBox to the appropriate paste buffer.

     The statement to remove the selected text from txtEditBox is already given.

     The other two subroutines, mnuEditPaste_Click and mnuEditCopy_Click, are completed for you.
3. Run the application. Verify that the Edit menu commands work appropriately.

Exercise 5: Coding the Options Menu

1. In this program the user can specify that only a single paste buffer be used. This is done from the Options menu. Add code to the mnuOptionsMultiple_Click event to do the following:

   a. Check whether the user has already enabled multiple options. If so, exit the subroutine (Fix #1).

   b. Set the Checked property of mnuOptionsMultiple to **True** (Fix #2).

   c. Set the Checked property of mnuOptionsSingle to **False** (Fix #3).

   d. Set mnuBuffer's Visible property to **True** (Fix #4).

   As you switched between single and multiple paste buffers, the top-level menu changed from three to four items.

   The statement that restores buffer 0's height to 15% of the height of the editing text box is provided for you. When you ran the application and selected the single buffer option, that buffer expanded to full height. Because in this menu item you're enabling multiple buffers, you need to shrink buffer 0.

2. In the mnuOptionsSingle_Click event, add code to do the following:

   a. Hide the top-level Buffer menu by setting its Visible property to **False**.

   b. Unload all paste buffers, their labels and menu items except the first. This requires five statements.

      There's a **For** loop that iterates from 2 to pastecount to do this. In five statements, you can unload text1, label1, mnuEditPaste, mnuEditCut, and mnuEditCopy. Note that unloading menu items makes them invisible.

   The last two statements of this subroutine are already provided. The variable pastecount is set to 1, and the sole remaining paste buffer is expanded to full height.

Exercise 6: If You Have Time...

The lab solution used Load and Unload when the user added or removed paste buffers.

However, this is not the most efficient way to do this. When the user adds a buffer for the first time, it is necessary to load that text box, its label, and the associated menus. When the user removes a paste buffer or switches to the single buffer option, it is more efficient to just hide the control (set its Visible property to **False**) than to unload it. When the user later re-adds a paste buffer, you can just set its Visible property to **True**, along with its menus.

Add the logic to make this happen. You'll have to track not only how many buffers have been added, but how many are visible. To keep the problem sane, remove buffers only from the high end, never from the middle.

Lab 7: Multiple Document Interface

Lab Objective

The purpose of this lab is to make an application an MDI application. This involves adding a few lines of code, performing certain actions in menus, and setting form properties.

It's important to stress that one doesn't just take an existing application and submerge it into an MDI window in one clean operation. In almost all cases, an existing application has to be tweaked some.

For instance, its menus will have to change, and it may become necessary to make some form-level variables global. This is particularly true if you want your MDI application to use a toolbar.

Objects and Routines

The following routines are provided for you:

Routine	Location	Purpose
SaveCurrentEditor	MDIChild.Frm	Saves current editor. Called from child's File Save menu
CloseCurrentEditor	MDIChild.Frm	Closes current editor. Prompts user to save file if dirtyflag is set. Called from child's File Close menu.
CreateNewEditor	MDI.BAS	Creates a new editor. Called from Parent's menu, and child's menu. Therefore this routine must be global.
GetFileName	MDI.BAS	Invokes common dialog control to prompt user for a file name. Called from Parent and Child's menus. A function - not a sub procedure since it returns a string.

Exercise 1: Testing the Solution

This program allows you to open multiple documents. A new form is created for each document. The program will prompt you if you attempt to exit without saving changes.

Run the solution in \VBLABS\MDI\SOLUTION. Test the program by doing the following:

1. From the File menu, choose New Child to open a new child form. Note how the menus change when a child form appears or disappears.
2. Load several child forms. From the Window menu notice how all the child forms are listed and the current child form is checked.
3. From the File menu, choose Open File to open a file. The common dialog is invoked to allow you to select a file. Open a text file. A new child form is created, and the file is placed in the form.
4. Make some modifications to the data file. From the File menu, choose Exit. The application prompts you to save changes. Respond Yes to save changes.

When you're satisfied you know how the program works, open the partial solution in \VBLABS\MDI\PARTIAL and go to Exercise 2.

Note The solution uses an array to keep track of all the child forms. Each time you select File New - a new form is created and the array is expanded if necessary. When a form is closed - the array entry is not re-used. In a more robust application you could set a flag in the array if the form were closed and reuse that array entry rather than expanding the array.

Exercise 2: Adding Declarations to the .BAS Module

Be sure you have opened the partial project in \VBLABS\MDI\PARTIAL.

Add the following global declarations in MDI.BAS:

- Declare a user defined record type named EdRecord that contains two fields - dirtyflag as integer and filename as string.
- Declare a global array named EditorData as type EdRecord. Do not provide any dimensions to this array. It will be resized later.
- Note the integer declaration EditorCount.

Exercise 3: Creating the MDI Parent Form

1. You'll have to create the parent form from scratch, but this won't be difficult. Most of the work is done by the child form or by the system, so an MDI parent form is usually trivial.
   a. To create the MDI parent form, choose New MDI Form from the File menu.
   b. Try to add another MDI form. Notice that the New MDI Form menu item is disabled.
2. Create the following menu on the parent form.

Menu caption	Name
&File	mnuFile
&New Child	mnuFileNew
&Open File	mnuFileOpen
E&xit	mnuFileExit

3. Set the following properties for the MDI parent form.

Property	Value
Caption	MDI Parent
Name	frmMDIParent
ScrollBars	False

4. Place a picture control on the MDI parent form. Use the default name. Set the picture control's Visible property to False.

   (A picture control is the *only* control you can place on an MDI parent form. Its sole purpose here is to provide a place to put the common dialog control. It will also be used in the extra credit to add a toolbar.)
5. Place the common dialog control on the picture control. Use the default name.
6. Set the startup form to be frmMDIParent.
7. Save this form.
   - Choose Save File As from the File menu, and name this form MDIPARNT.FRM.

Exercise 4: Adding Code to the MDI Parent Form

1. Add the following code to the MDIForm_Load event:

```
Sub MDIForm_Load ()
    'Make the MDI parent as large as possible.
    frmMDIParent.WindowState = MAXIMIZED
End Sub
```

2. Add the following code to the mnuFileExit_Click event:

```
Sub mnuFileExit_Click ()
    Unload Me
End Sub
```

3. Code the File New menu choice to create a new window without opening a file. To do this you can invoke the CreateNewEditor routine and pass a null string. The CreateNewEditor routine creates a new child form. The CreateNewEditor routine receives one argument that indicates the name of a file to open. If the argument is null, the routine does not open a file.

```
Sub mnuFileNew_Click ()
    CreateNewEditor ""
End Sub
```

4 Code the File Open menu to create a child window and open a file. To do this, you can invoke the GetFileName function to get a file name and then invoke CreateNewEditor and pass the filename as an argument. The GetFileName function has been provided for you.

```
Sub mnuFileOpen_Click ()
    Dim fname As String
    fname = GetFileName()
    If fname = "" Then
        Exit Sub
    End If
    CreateNewEditor fname
End Sub
```

Exercise 5: Coding the MDI Child Form

1. Open the form frmMDIChild.
2. Set the MDIChild property to **True**.
3. Code the File New menu to create another child form. You can copy the code from the File New menu on the parent to here.
4. Code the File Open menu to create another child form and open a file. You can copy the code from the File Open menu on the parent to here.
5. Code the File Close menu to unload the current form (UnLoad Me). This causes the Unload event to occur.
6. Code the Form_Unload event to invoke the routine SaveCurrentEditor and pass the form-level constant DO_QUERY.

   If the argument to SaveCurrentEditor is DO_QUERY and the dirty flag is **True** for this form, the routine prompts the user if the file should be saved and then saves the file if the user responds yes.

   If the argument to SaveCurrentEditor is NO_QUERY, the routine does not prompt the user, it just saves the file.

   You will invoke SaveCurrentEditor from Form_UnLoad and from FileSave. From Form_UnLoad, you do want to prompt the user. However, from FileSave, you do not want to prompt the user.
7. Code the File Save menu to invoke the routine SaveCurrentEditor and pass the form-level constant NO_QUERY.
8. Code the File Exit menu to unload the MDIParent form. This causes all the child forms to be unloaded and the application to end.
9. Code the mnuWindowMinimize_Click routine to iterate through all forms and minimize any form that is of type frmMDIChild.
10. The mnuWindowsRestore code has been provided for you.
11. Open the child form's menu design window and check the WindowList property for the Windows menu.
12. The SaveCurrentEditor routine has been provided for you. You may want to review the code in this routine.

Exercise 6: Coding the MDI Child Form: Setting the DirtyFlag

- If data in the text box control is changed, you need to set a flag. When the form is closed, the application checks this flag and prompts the user to save the file.

  In the txtEditBox_Change event, set the dirtyflag of the appropriate member of the EditorData array to **True**. The Tag property of the current form will contain the correct index into the array EditorData. The Tag property was assigned the appropriate value in the CreateNewEditor routine.

Exercise 7: Coding MDI Child Form: CloseCurrentEditor

The CloseCurrentEditor routine is a general procedure on the MDI Child form. This procedure is invoked from Form_Unload. This procedure receives an integer that indicates which form to close. The integer is used as a subscript into the array EditorData.

1. Set this form's dirtyflag to **False**.
4. Set this form's filename to a null string.

Exercise 8: Coding CreateNewEditor routine in MDI.BAS

The CreateNewEditor procedure creates a new child form, sets up the array of forms to point to the newly created form and optionally opens a file into the text box on the form.

The first time this procedure is executed, there is no need to create a new child form. It uses the child form that existed at design time. Remember, when you create an MDI child form at design time you get both a class and one object (an instance of that class).

The following times this procedure is executed - it creates a new child form. It then checks to see if it needs to increase the size of the array (it increases the array in increments of MAX_SIZE, which is currently three). It increases the array if necessary and sets the new entry in the array to point to the newly created child form.

Most of this procedure has been provided for you. You will need to add the following statements:

1. The first time the procedure is run - add the statement to set the array to point to the correct child form.
2. The following times the procedure is run - add the statement to create a new child form and point to it.
3. Each instance of a child form own a cell in the global array EditorData. Set the two variables owned by this instance. Explicit directions will be found in the file at Fix #3.
4. Set the Tag property of the newly created form to the counter. Remember that Tag is a string, so use the StrS function to convert the counter.

Exercise 9: If You Have Time...Adding a Toolbar

The presence of a toolbar greatly complicates an MDI application. It's far easier to design the application from the start with a toolbar in mind than it is to add a toolbar to an existing MDI application. The major problem is that the toolbar is part of the parent form, but the data and procedures are parts of the child form. The parent does not have access to a child form's data and procedures.

Run the solution from \VBLABS\MDI\XCREDIT. The file folder is for File Open, and the diskette is for File Save.

If you want to convert your completed MDI application from the \VBLABS\MDI\PARTIAL directory to use a toolbar, here's a list of the things you should do:

1. Place two image controls on the picture control that's already on the MDI parent. You may either make these two controls into an array, or leave them as separate controls. The extra credit solution places them into an array. The icons you will use are \vb\icons\office\folder02.ico and \vb\icons\computer\disk06.ico.
2. Make the picture control visible. (It's currently invisible, since it only serves to hold the common dialog control.)
3. The code for the file open icon already exists in the parent's Open File menu, so just copy it to the click event of the appropriate image control.
4. The routine SaveCurrentEditor is currently a form-level routine on the child form. You need to make it globally available by placing it in MDI.BAS. It's not enough to just place it there. The routine uses Me to refer to the current child form. When this routine is no longer a form-level routine, Me is no longer available. You must use frmMDIParent.ActiveForm in place of Me.

   Also, there are several form-level constants that must be placed in MDI.BAS.
5. You may now call SaveCurrentEditor from the Save image control's click event. However, you need a means of disabling this control when no child forms are loaded. There are several ways to do this, but perhaps the easiest is to only invoke SaveCurrentEditor if Screen.ActiveForm is of type frmMDIChild. The keyword TypeOf will provide this information.
6. You may want to place code in the parent's Form_Load event to position the two image controls.

Lab 8: Trapping Run-Time Errors

Lab Objective

The purpose of this lab is to implement error handling for the Text Editor application.

You will write an application that handles the following two run-time errors:

- "Drive not ready"—User selects a disk drive without a disk.
- "File too big"—File exceeds maximum length of a string (64K).

In this lab, you will focus on the following statements:

```
On Error Goto label
On Error Goto 0
Resume
Resume Next
Resume label
```

Exercise 1: Testing the Solution

Run the solution in C:\VBLABS\ERRORS\SOLUTION.

Choose Open from the File menu, and try to open a file from drive A without a disk in drive A. Notice the error message. Place a disk in drive A and select Retry. Try to open a file from drive B without a disk in drive B; this time select Abort.

There are numerous files in C:\VBLABS\ERRORS. Their names indicate their lengths. Try to open each of these files, and note what happens.

When you are satisfied you understand how the application works, close the solution, open the partial solution in C:\VBLABS\ERRORS\PARTIAL, and continue with Exercise 2.

Exercise 2: Handling "Drive Not Ready" Error

In this exercise, you will add error-handling code to check for invalid drives on the form frmFileList.

The partial solution in \VBLABS\ERRORS\PARTIAL does not use the common dialog control. If you use the common dialog control to open or save a file, you do not have to worry about handling invalid drive specifications. The common dialog control handles it for you. However, if you had your own file open dialog form with just the standard drive, directory, and file list boxes, you would need to include an error-handling routine for invalid drives.

► Enable the error handler in the Drive1_Change event

1. In the Drive1_Change event on the frmFileList form, add the following statement at the beginning of the routine (Fix #1):

   ```
   On Error GoTo OpenDriveError
   ```

   This causes the program to branch to the label OpenDriveError if any run-time error occurs during this subroutine.

2. Place the label OpenDriveError: at Fix #2.

3. Place the statement Exit Sub at Fix #3.

   This ensures that you do not fall through and execute the error routine if there was not an error.

► In the error handler - Determine which error occurred and Display message

- At Fix #4, right after the OpenDriveError label, place the code to determine which error occurred. You can do this with a **Select Case** statement that checks the value of ERR.

  If the error number is 68, that indicates a "Drive not ready" error. In this case, you should display an Abort, Retry or Ignore message box to the user with the message that a drive is not ready.

  If the error number is any number other than 68, you should display an OK message box indicating "Some Other Error Occurred" and then end the program.

```
1 'Determine the File Error
2  Select Case Err
3   Case 68, 76 'Device Unavailable
4        mymsg = " Drive Not Ready    "
5        msgtype = ABORTRETRYIGNORE + WARNINGMESSAGE
6        msgtitle = "DRIVE ERROR"
7   Case Else ' Any other error
8        mymsg = "Some other error " + Str$(Err) + " " + Error$
9        msgtype = OK
10       msgtitle = "Unknown Error"
11 End Select
13    KeyPressed = MsgBox(mymsg, msgtype, msgtitle)
```

▶ **In the error handler - Determine how user responded to message**

At Fix #5, query the value of KeyPressed and take appropriate action. For example, if KeyPressed = KEYRETRY (4), you will want to use **Resume**, to retry the statement that caused the error. If KeyPressed = KEYIGNORE, you should use **Resume Next**. If KeyPressed = KEYABORT you should set the Drive1.Drive to the first two characters of Dir1.Path and use **Resume**.

```
'Process which key user pressed
Select Case KeyPressed
  Case KEYRETRY
      'Try again
       Resume
  Case KEYIGNORE
       'Ignore, go on to next line
       'Note that the drive has not really changed
        Resume Next
  Case KEYABORT
       'Return to previous drive setting & Resume
        Drive1.Drive = Left$(Dir1.Path, 2)
        Resume
  Case Else
       'Unexpected key value
        mymsg = "Unexpected results, key = " + Str$(KeyPressed)
        msgtype = CRITICAL
        msgtitle = "KEY ERROR"
        MsgBox mymsg, msgtype, msgtitle
       'End the Application
        End
End Select
```

▶ **Save your project, and test your application.**

Exercise 3: Checking for "File Too Large"

There is a limit to how big the string in a text box can be. This is a statement found in mnuFileOpen_Click of the TextEdit form:

```
entirefile = Input$(LOF(fhandle), fhandle)
txtEditor.Text = entirefile
```

When you read a text file and assign it to a text control in this way, there are several things that can go wrong.

The first thing that can go wrong is not a trappable error but is still a problem. The biggest file that can be placed in a text box *and* edited is 29,999 bytes. When the contents of a text box exceeds that amount, you won't be allowed to add any more characters. You can, however, browse in that text box. There is no trappable error that occurs when a file of, say, 32,000 bytes is placed in a text box.

Thus, for files between 30,000 and about 45,000, all you can do is browse.

At about 45,000 bytes, when you read the file into the text box as in the statement above, you'll get the trappable error "Out of Memory". The exact size at which this error occurs depends on what your program is doing elsewhere regarding variables. You'll get this error for files up to about 60,000 bytes.

Between about 60,000 and 65,535 bytes, the statement above generates the trappable error "Out of String Space".

When the file size exceeds 65,535 bytes, you'll get error #5, illegal function. This is because the first argument to Input$ cannot exceed 65,535.

There are numerous files in \VBLABS\ERRORS for you to use to observe all of these errors and problems.

In this exercise, you will write error-handling code to deal with all three of these trappable errors for the form frmTextEdit.

▶ Enable the error handler in the File Open routine

1. In the mnuFileOpenClick routine on the form frmTextEdit, place this statement *right before* the call to open the file:

```
On Error GoTo OpenError
```

2. At the end of the mnuFileOpenClick routine, insert this statement and label:

```
Exit Sub
OpenError:
```

▶ In the Error Handler - Determine which error occurred

- Immediately after the label OpenError, install your error-handling code. Your code should handle the three errors described above. The necessary constants and variable declarations are provided at the top of this event procedure or in a .BAS file.

```
1  OpenError:
2      msgtype = RETRYCANCEL + WARNINGMESSAGE + SECONDBUTTON
3      msgtitle = "FILE SIZE ERROR"
4
5      'Determine the File Error
6      Select Case Err
7          Case ILLEGAL_FUNCTION_CALL
8              mymsg = "File's WAY too big."
9          Case OUT_OF_MEMORY
10             mymsg = "Not enough memory."
11         Case NO_STRING_SPACE
12             mymsg = "Out of string space."
13         Case Else     'Any other error
14             mymsg = "Some other error " + Str$(Err) + " " + Error$
15             msgtype = OK
16             msgtitle = "Unknown Error"
17     End Select
18
19     KeyPressed = MsgBox(mymsg, msgtype, msgtitle)
```

► **In the Error Handler - Determine how user responded to message**

- Query the value of KeyPressed and take appropriate action. For example:

```
1
2    Select Case KeyPressed
3        Case KEYRETRY                'Try again
4            Resume
5        Case KEYCANCEL               'Cancel attempt to read file
6            Resume FileTooBig
7        Case Else                    'Unexpected key value
8            mymsg = "Unexpected results, key = " + Str$(KeyPressed)
9            msgtype = CRITICAL
10           msgtitle = "KEY ERROR"
11           MsgBox mymsg, msgtype, msgtitle
12           'End the Application
13           End
14   End Select
```

A simpler way of handling these three errors would be to just check the value of the LOF function before trying to read the file into the text box.

Exercise 4: If You Have Time...

You might need to invoke the drive-error-handling routine from several forms. Instead of including the code directly in the Drive1_Change event, you could place this code in a general procedure and then call that procedure from the Drive1_Change event. If you have time, modify your application to do this.

A centralized version of error handling is provided in the \VBLABS\ERRORS\XCREDIT directory. Examine the code in Drive1_Change () for file FILELIST.FRM.

Lab 9: Dynamic-Link Libraries

Lab Objectives

The purpose of this lab is to implement various Windows API calls.

After completing this lab you will be able to:

- Write an application that displays the Windows system path.
- Write an application that saves information to an initialization file.

In this lab, you will focus on the following Windows API calls:

- **GetWindowsDirectory**
- **GetPrivateProfileString**,
- **GetPrivateProfileInt**,
- **WritePrivateProfileString**

Exercise 1: Testing the Solution for Windows System Directory

Run the solution in \VBLABS\DLL\SOLUTION.

Notice the Information menu. From the Information menu, choose Windows Path. A message box will appear with the path for the Windows files.

When you are certain you understand the purpose of the application, open the project in \VBLABS\DLL\PARTIAL and move on to Exercise 2.

Exercise 2: Getting the Windows System Directory

In this exercise, you will add a menu item to display the path of the Windows files.

▶ Add the appropriate menus to the form frmMPBE

1. Add a menu titled Information, and set the name to mnuInfo.
2. Under that menu add a submenu titled Windows Path; set the name to mnuWinPath.

▶ Declare the API routines

You should always copy the declaration from the WIN31.HLP file into your application.

1. Double-click the icon "WIN 3.1 API Help" in the VB group to view the help file.
2. Copy the declaration for the **GetWindowsDirectory** function and paste it into the Declarations section of the INIWIN.BAS module.
3. If you would like more information about the **GetWindowsDirectory** function, double-click the icon "Win SDK Help" in the VB group and search for GetWindowsDirectory.

▶ Declare the variable to be used in the API routines

The **GetWindowsDirectory** function accepts a string argument lpbuffer. This is the buffer that receives the string indicating the Windows directory.

- Declare the string as a global variable in the Declarations section of a module as follows:

```
Global lpBuffer As String
```

- In the form load event, fill the string with the null characters
  lpBuffer = String$(255,0)

▶ Invoke the routine and display a message box with Windows directory

1. In the mnuWinPath_Click event, add the invocation to **GetWindowsDirectory**.

   Pass lpBuffer as the first argument

   The second argument specifies the maximum length of the buffer. To be sure you have enough room for the returned string, pass 255 for the second argument.
2. Display a message box that indicates the Windows path.
3. Save your project, and test your application.

Exercise 3: Testing the Solution — Creating an Application .INI File

There are many cases where you want to save some information between executions of an application. For example, if a user moves or resizes a window, you might want to save the position for use the next time the application starts. The easiest way to accomplish this is to create an application .INI file.

When the application ends, you write any information you would like saved to the .INI file. When the application starts, you read the .INI file and take appropriate action.

In the multiple paste buffer editor, you will record the form's position. Each time the application starts, you will position the form to wherever the user last left it. You will also add the appropriate number of buffers.

Run the solution in \VBLABS\DLL\SOLUTION\INIWIN.MAK.

Resize and move the form. Add two more buffers. From the File menu, choose Exit. An .INI file should be created. Run the application again. Notice the form is positioned in the same place and two more buffers are added.

Use Notepad to view the file.C:\WINDOWS\ED2.INI. Notice the information that is stored in the file.

When you're satisfied you understand the purpose of the application, close the solution, open \VBLABS\DLL\PARTIAL\PINIWIN.MAK, and move to Exercise 4.

Exercise 4: Creating an Application .INI file

1. The first step is to declare the API functions in the Declarations section of a module.

   a. Create a module if necessary.

   b. Copy the following declarations from the WIN3.1 API Help file into the Declarations section of the module: Notice the declarations allow arguments to be passed "As Any". Remember to use ByVal when you invok these procedures.

   WritePrivateProfileString

   GetPrivateProfileInt

2. These functions require arguments that indicate the name of the .INI file and the name of a section in the .INI file.

   The .INI filename will be ED2.INI. The section name will be [Text Editor].

   Declare two global constants in the Declarations section of a module to store these values:

```
Global Const INIFILENAME = "ED2.INI"
Global Const SECTION = "TEXT EDITOR"
```

3. In the Form_Unload event, invoke the **WritePrivateProfileString** to store the Top, Left, Width, and Height of the form. This requires four invocations of **WritePrivateProfileString**. For example, to record the top of the form, the invocation will look like this:

```
Temp = WritePrivateProfileString(SECTION, ByVal "Top",
    ByVal Str$(frmMPBE.Top), INIFILENAME)
```

4. Save your project at this point.

5. Test the program. Run the program. Move the form. End the program. From an MS-DOS prompt, view the file C:\WINDOWS\ED2.INI. The file should look something like:

```
[TEXT EDITOR]
Top= 0
Left= 0
Height= 7200
Width= 7125
```

   If the file is not correct, stop now and fix your application before going on.

Note Remember, because invalid calls to API functions can cause general protection (GP) faults, always save your work before testing your application.

---

6. Once you have created the .INI file properly, you can start coding the Form_Load event to read the .INI file and take the appropriate action.

   In the Form_Load event, invoke the **GetPrivateProfileInt** function to set the Top, Left, Width, and Height for the form. This requires four invocations of **GetPrivateProfileInt**.

   If an .INI file does not exist, set the Top and Left of the form to 0 and set the Width to Screen.Width and Height to Screen.Height. The third argument in the **GetPrivateProfileInt** call is the value to use if the .INI file was not found.

   For example, the following two calls to **GetPrivateProfileInt** set the values for Top and Width:

```
frmMPBE.Top = GetPrivateProfileInt(SECTION, "Top", 0, INIFILENAME)
frmMPBE.Width = GetPrivateProfileInt(SECTION, "Width", Screen.Width,
  - INIFILENAME)
```

7. Save your project, and test your work.

Exercise 5: If You Have Time...

You may want to expand the application to also save the number of paste buffers that the user has opened. You will need to add another call to **WritePrivateProfileString** in the Form_Unload event to store the value of PasteCount. You will also need to add another call to **GetPrivateProfileInt** to retrieve the stored value. Finally, you will need to invoke mnuBufferAdd_Click the appropriate number of times. If PasteCount was stored as 1, you will want to add 1 buffer, and so forth.

Lab 10: Dynamic Data Exchange

Lab Objective

The purpose of this lab is to use dynamic data exchange (DDE) with Visual Basic and Microsoft Excel.

In part 1, you will create a Visual Basic destination application that links to an Excel worksheet.

In part 2, you will create a Visual Basic source application that links to a Visual Basic destination application.

In part 3, you will expand the XLBatch application to be a Visual Basic destination application that uses Microsoft Excel to print a list of selected files.

In this lab, you will focus on the following Properties, Events, and Methods:

- Properties: LinkTopic, LinkItem, LinkTimeout, LinkMode
- Events: LinkOpen, LinkClose
- Methods: **LinkPoke, LinkRequest**

Part 1: Visual Basic to Microsoft Excel

Exercise 1: Testing The Solution

This application links a Visual Basic form to a Microsoft Excel worksheet. The application assumes Excel is already running.

1. Start Excel; open the document C:\VBLABS\DDE\SOURCE.XLS.
2. Minimize Excel.
3. From Visual Basic, open the solution and run the application C:\VBLABS\DDE\VBFROMXL\SOLUTION.
4. Select the Automatic option.

   The text box should be filled with information from Excel.
5. Switch to Excel, type some new data, and press ENTER.

   The Visual Basic form should be automatically updated.
6. Switch to the Visual Basic form, and choose the Manual option.
7. Switch to Excel, and type new data.
8. Switch to Visual Basic.

   The new data is not reflected in Visual Basic until you choose the Update button.
9. From Visual Basic, type some new data in the text box and choose Poke.
10. Switch to Excel, and verify the new data has been received.

When you are sure you understand the purpose of the solution, close the project and open the partial solution in C:\VBLABS\DDE\VBFROMXL\PARTIAL.

Leave Excel running and the SOURCE.XLS file open.

Exercise 2: Coding the Visual Basic Destination Application

Be sure you have opened the partial solution.

In this exercise you will create a form with a text box that is linked to a Microsoft Excel worksheet. The form includes the following:

- A text box linked to a cell in an Excel worksheet SOURCE.XLS
- Option buttons to select Automatic or Manual
- A command button that updates the text box from the Excel worksheet for a Manual link
- A command button that pokes data from the text box into the Excel worksheet.

1. Be sure you have opened the partial solution in C:\VBLABS\DDE\VBFROMXL\PARTIAL.

   Examine the form VBDEST.FRM.

   Observe the existing objects and their given names and properties.

Object name	Purpose
frmDest	Visual Basic destination to Excel—main form
txtData	Text box to receive data from DDE link
cmdUpdate	Button to request data
cmdPoke	Button to poke data to Excel
optAutomatic	Option to set DDE link to automatic
optManual	Option to set DDE link to manual
optNone	Option to terminate link

2. Set the following properties for the text box at Form_Load:

```
txtData.LinkMode = NONE
txtData.LinkTopic = "EXCEL|SOURCE.XLS"
txtData.LinkItem = "R1C1"
txtData.LinkTimeout = 30
```

3. Disable the Update button and the Poke button at Form_Load.
4. In the optManual_Click event, set the text box LinkMode to manual, and enable the Update button and the Poke button.
5. In the optAutomatic_Click event, set the text box LinkMode to automatic, disable the Update button and the Poke button.
6. In the cmdUpdate_Click event update the contents of the text box.
7. In the optNone_Click event, set the link to NONE, disable the Update button and Poke button and clear the text box.
8. The code in the cmdPoke_Click event has already been provided for you. Review the code.
9. This application assumes that Excel is already running and the file C:\VBLABS\DDE\SOURCE.XLS is already open. A run-time error occurs if this is not true.

   When testing this application, be sure you have Excel started and the file SOURCE.XLS open.

Exercise 3: If You Have Time...
Adding a Picture Box

1. Add a picture box to your form. Set the name to picData.
2. Set the following properties for the picture box at Form_Load:

```
picData.LinkMode = NONE
picData.LinkTopic = "EXCEL|CHART1.XLC"
picData.LinkItem = "Chart"
picData.LinkTimeout = 30
```

3. If the user clicks the Manual option, set the LinkMode for the picture box to MANUAL. Remember to set LinkMode to NONE before setting it to MANUAL.
4. If the user clicks the Automatic option, set the LinkMode for the picture box to AUTOMATIC.
5. If the user clicks the Update button, execute a LinkRequest to get the new data for the picture box.
6. Add a command button that will allow the user to end the application.
7. To clear the picture box when NONE is selected, use picData.Picture = LoadPicture ()
8. To test this application, you need to start Excel and open the chart CHART1.XLC file. You will find this file in C:\VBLABS\DDE.
9. Once Excel is running and the chart is open, run your application.

Part 2: Visual Basic to Visual Basic

Exercise 1: Testing the Solution

This application links a Visual Basic form with another Visual Basic form.

1. Open the project DDE-S.MAK in C\VBLABS\DDE\VB2VB\SOLUTION and make an executable.

   From Program Manager, run the Visual Basic source application C:\VBLABS\DDE\VB2VB\SOLUTION\DDE-S.EXE. Choose the Source option so the application will act as a source. Minimize the application.

2. From Visual Basic, open the project under C:\VBLABS\DDE\VB2VB\SOLUTION\DDE-D2.MAK, and run it.

   Choose the Automatic option. The form should receive information from the other Visual Basic application.

When you are certain you understand how the application works, open the partial solution for the source application in C:\VBLABS\DDE\VB2VB\PARTIAL, and move on to Exercise 2.

Exercise 2: Coding the Visual Basic Source Application

In this exercise you will create two Visual Basic applications. One will be the source application and one will be the destination application.

Be sure you have opened the partial solution in C:\VBLABS\DDE\VB2VB\PARTIAL\PDDE-S.MAK.

1. Examine the form, DDE-S.FRM. Observe the existing objects and their given names and properties.

Object	Purpose
frmSource	Visual Basic source form
txtSource	Text box to act as source in DDE link
cmdDone	Command to end the application
optSource	Option to set the application to act as a source application
optNone	Option to cancel the DDE conversation.

   The constants NONE, AUTOMATIC, MANUAL, and SOURCE are provided as form-level constants.

2. Set the LinkMode property of frmSource to Source. This is required at design time in order for the application to be a DDESource.
3. Code the Source option button to:
   a. Clear any previous server link.
   b. Set the source LinkTopic of the form to "frmSource".
   c. Set the LinkMode of the form to SOURCE.
4. Code the None option button to clear any existing links.
5. Run your application to be sure it can compile and execute.
6. Make your source application into an .EXE called DDE-S.EXE. (You will use this name in the client application.)

Exercise 3: Coding the Visual Basic Destination

1. Open the project, PDDE-D2.MAK.
2. Examine the single form of this project.

   Observe the existing objects and their given names and properties.

Object	Purpose
frmDest	Visual Basic destination form
txtDest	Text box to server as destination link
cmdUpdate	Command to update the contents of the text box
optAutomatic	Option to set the DDE link to automatic
optManual	Option to set the DDE link to manual
optNone	Option to terminate the DDE link

3. Code Form_Load to:
   a. Set txtDest.LinkMode to NONE.
   b. Set txtDest.LinkTopic to the name of the source application plus the value of the LinkTopic in the Visual Basic source ("DDE-S1frmSource").
   c. Set txtDest.LinkItem to the name of the text box on the source ("txtSource").
   d. Set txtDest.LinkTimeout to 30.
4. All code for the option buttons and the command buttons is already provided for you. Examine it to see what it does.
5. Make your application into an .EXE named DDE-D2.EXE.

Exercise 4: Testing Your Applications

1. Run your source application. Choose the Source option before continuing.
2. Run your destination application. Verify that your destination application can receive and send data to the source application.

Part 3: Microsoft Excel Batch Print Utility

Exercise 1: Testing the Solution

This application implements the Print button in the BATCHXL application. The application uses Microsoft Excel to print a group of files. The solution automatically starts Excel if necessary. The solution sends **Print** statements to Excel to print all the files in the Files To Print list box. Test the solution:

1. Close Microsoft Excel.
2. Run the solution in C:\VBLABS\DDE\BATCHXL\SOLUTION.
   a. Place several .XLS files in the Files to Print list box. Some .XLS files can be found in C:\VBLABS\DDE.
   b. Place only .XLS files in the Files To Print list box. If you try to print any other type of file, Excel will send back an error message.
   c. You may need to use the Windows Control Panel to set your printer to FILE if there is no printer available in class.
   d. Choose the Print button to cause the Print command to be sent to Excel.

When you are certain you understand how the program works, open the partial solution in C:\VBLABS\DDE\BATCHXL\PARTIAL.

Exercise 2: Coding the BatchXL Print Command

Be sure you have opened the partial solution in C:\VBLABS\DDE\BATCHXL\PARTIAL.

1. Add a form-level string variable named ExcelCmd. Note the constants that have been provided.
2. Add a text box to the BATCHXL.FRM. Set the following properties:

```
Name = txtHidden
Visible = False
```

3. Code the Print button (cmdExcel_Click) to:
   a. Set txtHidden to be the link to Microsoft Excel. Set the following properties of txtHidden:

      LinkMode = NONE

      LinkTopic = "Excel|System"

      LinkMode = MANUAL

      No LinkItem is needed, because we only want to link to Excel for LinkExecutes, not data exchanges.

   b. Code the three command strings and send them to Excel by means of LinkExecute. Remember the file name will need to be in quotes. You'll need the four quotes to equal an actual quote in the final string. For example:

```
Dim Q as String
Dim fname As String
Q = Chr$(34)     'This is the quote symbol "
fname = lstPrint.List(lstPrint.ListIndex)
ExcelCmd = "[OPEN(" + Q + fname + Q + ",1)]"
txtHidden.LinkExecute ExcelCmd
```

      The other commands will need to be created as follows:

```
[PRINT(1,,,1,FALSE,FALSE,1,FALSE,1)]
[CLOSE(FALSE)]
```

   c. Place `rc = DoEvents()` after each LinkExecute command.

   d. Set the LinkErrorTimeout to 30 seconds for the Open and Close commands. Set the LinkErrorTimeout to 60 seconds for the Print command.

   e. Change Screen.MousePointer to an hourglass while printing and back to normal when done. Insert these two statements at appropriate places in the event procedure. Use the constants DEFAULT and HOURGLASS.

4. Terminate the DDE link when the files have printed.
5. Display a message box when all files have printed.

Exercise 3: If You Have Time... Handling a DDE Run-Time Error

Verify that Microsoft Excel is started before trying to print. If it is not, you will get the following run-time error:

```
282  ' No application responding to DDE
```

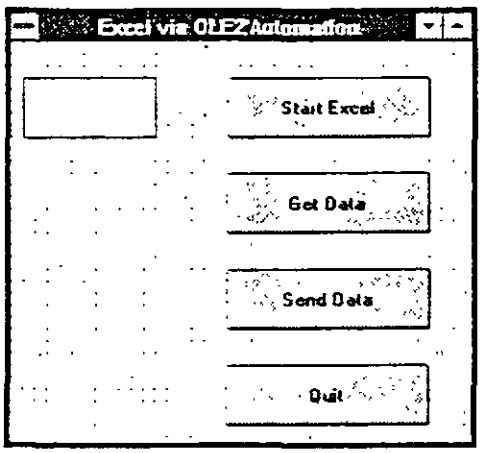
Ask whether the user wants Excel started (YES/NO). Handle user responses.

Lab 11: Object Linking and Embedding

Lab Objectives

The purpose of this lab is to write an application that can send and retrieve data to and from Microsoft Excel. The program is a variation on the DDE lab that did the same thing, except that it uses DDE instead of OLE.

Your form should look like this:



Exercise 1: Reviewing the Solution

► **Run the lab solution in \VBLABS\OLE\SOLUTION\OLERW.MAK**

Test the program by doing the following:

1. Choose the Start Excel button.

   Microsoft Excel is started, a workbook is opened, and Microsoft Excel is minimized

2. Choose the Get Data button.

   The value of cell A1 is placed in the text box.

3. Modify the value in the text box and choose the Send Data button.
4. Switch to Microsoft Excel to verify the new data in A1.

When you're satisfied that you know how the program works, start a new project and go on to the next exercise.

Exercise 2: Writing the Application

1. Design the form as shown in the previous diagram.
2. Save the form and project in the directory \VBLABS\OLE.
3. Code the Open button to:
   a. Load Microsoft Excel
   b. Open the workbook C:\VBLABS\OLE\SOURCE.XLS
   c. Select the worksheet SOURCE

Note If you have time, code the application to start Microsoft Excel only if it is not already running and open the workbook only if it is not already open.

---

5. Code the Get command button to retrieve the value from cell A1 into the text box.
6. Code the Send command button to send the value from the text box to cell A1.
7. Test the application.

Exercise 3: Optional...

1. A well-behaved program should leave the computer in the same state as when the program began. That is, you should close Microsoft Excel if you started it and close the workbook if you opened it. Add the code to do this.
2. Create functions for starting Microsoft Excel, for determining if a workbook is open, and for opening a workbook. Make these functions generic so they can be used in other OLE programs. This will require passing arguments into the functions.

Lab 12: Data Access Using the Data Control

Lab Objectives

In this lab you will write an application that displays data from the BIBLIO.MDB database. The BIBLIO database is a sample database that ships with Visual Basic.

The BIBLIO database has the following tables:

Authors
- Au_ID
- Author

Titles
- Title
- Year Published
- Au_ID
- ISBN
- PUBID

Publishers
- PUBID
- Name
- Company Name
- Address
- City
- State
- Zip
- Telephone
- Fax

Publishers Comments
- PUBID
- Publisher
- Comments

Exercise 1

1. Create an application that displays the title and author from the BIBLIO.MDB database. The BIBLIO.MDB database is in the VB directory.

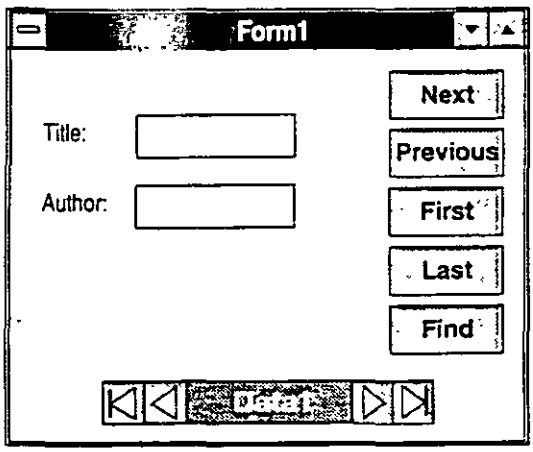
   Include Move buttons to move first, last, next and previous.

   Include a Find command button to find a particular author.

2. You will need to join the authors table and the titles table to get the author and title. To do this, set the RecordSource property of the data control as follows:

```
Data1.RecordSource = "Select author, title from authors, titles
                     ➥where authors.au_id = titles.au_id"
```

3. The application should look as follows:



Lab 13: Data Access Using the Data Object Variables

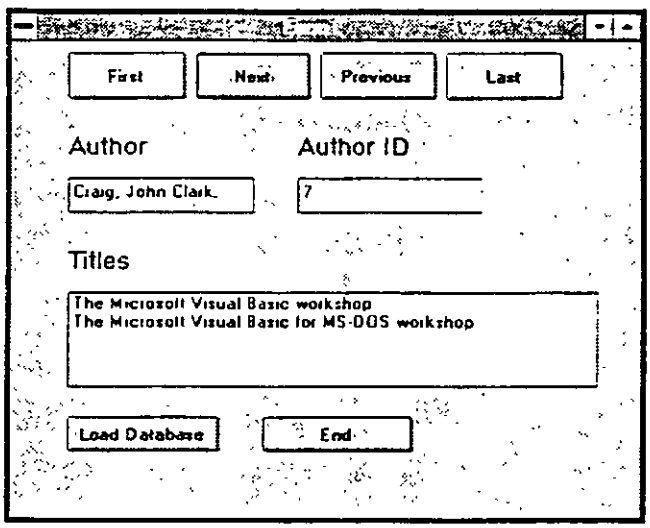
Lab Objectives

In this lab you will write an application that displays authors and titles from the BIBLIO.MDB database.

For each author, we want to see a list box that contains all the titles written by that author.

You will use the object variables **Database** and **Dynaset**.

Your form should look like this:



Exercise 1 - Test the Solution

1. Open the solution in C:\VBLABS\DATA2\SOLUTION.
2. Choose Load Database to load the database.
3. Choose Next to move to the next record.
4. The titles for each author are displayed in a list box.

Exercise 2 - Display the Author and Author ID

To make things simple, start by just displaying the author and author id and getting the buttons to work. In the next step, you can add the title information.

The form has been designed for you. No code has been provided.

1. Open the partial solution C:\VBLABS\DATA2\PARTIAL.
2. In the general declarations section, declare a **Database** variable and a **Dynaset** variable.
3. Code the cmdloaddb_Click event to:

   Set the database variable to C:\VB\BIBLIO.MDB

   Create a **Dynaset** that contains the author and au_id from the authors table. You can use the following SQL statement to create the **Dynaset:**

   ```
   Select Author, Au_ID from Authors
   ```

   Display the first record.
4. Code the First, Next, Previous and Last command buttons to move through the **Dynaset** and display the information in the text boxes.
5. Code the cmdEnd button to end the application. What do you need to close before ending the application?
6. Run your application and make sure it works correctly.

Exercise 3 - Display the Titles for Each Author

Now, let's display the titles for each author.

The titles are in the Titles table. You will need a separate **Dyanset** variable to hold the title information.

1. In the general declarations section, declare another **Dynaset** variable.
2. Create a general procedure that creates a **Dynaset** of titles for a particular author. You can use the following SQL statement when creating the **Dynaset** for the titles:

```
Dim sql As String
sql = "Select Title from Titles Where Au_ID = " & txtAuID.Text
set dsTitles = db.CreateDynaset (sql)
```

   This procedure should also loop through the titles **Dynaset** and copy all of the titles to the lstTitles list box.
3. Invoke the general routine you just created from each of the move command buttons.
4. Test your application.

Exercise 4 - If You Have Time...

Code the Solution using Snapshot and Filter

Another way to solve this program is to create a snapshot of all the titles when the database is loaded. Then set the Filter property to display only the titles for a particular author. Modify your program to use this method.

A solution has been provided for you in C:\VBLABS\DATA2\XCREDIT\SOLUTION.

1. In the general declarations section of the form, declare two snapshot variables.
2. In the cmdloaddb_Click event, set a snapshot variable to contain all of the titles from the titles table.
3. In the general procedure to get the titles:

   Set the filter property for the **snapshot** variable to specify only the titles for the current author.

   Create another snapshot **based on the first** snapshot.

   Copy all the titles from the **filtered snapshot** to the lstTitles list box.
4. Invoke the general procedure **from each** of the move buttons.
5. Test your application.

Lab 14: Adding Online Help

Lab Objectives

The purpose of this lab is to add online help to a Visual Basic application.

There are four steps in adding Help to a Visual Basic Application:

1. Create the .RTF file.
2. Create the help project (.HPJ) file.

   Assign the context numbers.
3. Compile the .RTF files by passing the .HPJ the first command line parameter to the Help compiler.
4. Create the Visual Basic program.

   Assign the HelpContextId properties to the appropriate context numbers.

   Assign the HelpFileName property to the application.

   Help will be invoked automatically when the user presses F1.

   If you want to invoke Help explicitly, you can use the common dialog control or call the WinHelp Windows API function.

Exercise 1: Creating the .RTF File

The .RTF file is the file that contains the actual help text. This file can be created in any word processor that can save files in rich text format.

The easiest way to create the file is to use the Windows Help Authoring Template (WHAT.DOT). The Windows Help Authoring Template makes creating help files easier by providing dialog boxes and macros to enter help features that would normally be entered using specialized help coding and arbitrary word-processing functions. For example, instead of typing footnotes for a topic's context string and title, you can simply type the information in a dialog box.

In this exercise you will use Microsoft Word for Windows and WHAT to create a simple help file. The help file will have an index page with two jump topics and one popup definition.

► **Create a new document based on the WHAT6 template.**

1. Open Word for Windows.
2. From the File menu, choose New.
3. From the template list, choose WHAT6.

   Some new options are added to the Insert menu to assist with creating the help file. You are now working on Document2.

► **Insert a new topic named GEN_TOPIC**

1. From the insert menu, choose Topic.
2. Fill in the Topic table as follows:
   a. Title: General Information
      Context String: GEN_TOPIC
      Keywords: General;Index
   b. Check the option "Place new topic at end of file"
   b. Choose OK.

   The topic is inserted. You should now see:

   S.#.K <paragraph marker>

   You should be on a line that has style Heading 9. You would insert text as Heading 9 if you wanted some See Also topics under the main heading. We are not adding those.

   Press enter to start a line as style Normal.

   If the paragraph mark is not visible, select the paragraph symbol in the upper-right corner of the toolbar.
3. In the document, type the text for the topic as follows:

   This is the general information. This is the first page of Help that the user will see when they call Help from my program.
4. Press ENTER to start a new line.

▶ Add a Jump to COM_TOPIC

1. From the Insert menu, choose Jump or Popup Hotspot.
2. Fill in the Topic table as follows:

Text:	List of Commands
Context String:	COM_TOPIC

3. Be sure JUMP is selected.
4. Choose OK. The jump entry is added to your document.
5. Press ENTER to start a new line.

▶ Insert the topic for COM_TOPIC

1. From the Insert menu, choose Topic.
2. Fill in the Topic table as follows:

Title:	List of Commands
Context String:	COM_TOPIC
Keywords:	Commands

3. Choose OK.

   The topic is inserted.

4. In the document press enter to start a new line with style Normal

   Then type the text for the topic as follows:

   This information is displayed when a user selects List of Commands from my help file.

5. Press ENTER to start a new line.

▶ Include a popup on this topic

1. In the document type the text:

   Here is a definition of

2. From the Insert menu, choose Jump or Popup Hotspot.

Text:	multitasking
Context string:	WORD_TOPIC

3. Select Popup.
4. Choose OK.

   The popup entry is inserted in your document.

5. Press ENTER to start a new line.

▶ Insert the topic for WORD_TOPIC

1. From the Insert menu, choose Topic.
2. Fill in the Topic table as follows:

Title:	multitasking definition
Context String:	WORD_TOPIC
Keywords:	multitasking

3. Choose OK

   The WORD_TOPIC is inserted.

4. In the document, press enter to start a new line styled as Normal.

   Type the text for the topic as follows:

   This information will pop up when the user clicks "multitasking".

5. Press ENTER to start a new line.

► **Save the file and close Word for Windows**

1. Save the file as C:\VBLABS\HELP\HELP1.RTF. Be sure the save file type is RTF.
2. Close Word for Windows.

Exercise 2: Creating the Help Project File

► **Create the help project file:**

You can create a help project file using Notepad.

1. Open Notepad.
2. Create the .HPJ file as follows:

```
[FILES]
help1.rtf

[MAP]
GEN_TOPIC 1
COM_TOPIC 2
WORD_TOPIC 3
```

3. Save the file as C:\VBLABS\HELP\HELP1.HPJ.

Exercise 3: Compile the Help File to Create the .HLP File

► **Compile the help file**

To compile the help file, from an MS-DOS prompt type:

```
CD \VBLABS\HELP
HCP  HELP1.HPJ
```

The file HELP1.HLP is created and saved in C:\VBLABS\HELP.

Exercise 4: Testing Your New HELP File

Before using the help file from Visual Basic, you might want to just run WinHelp against the help file and verify it was created correctly.

1. Choose Run from the Windows Program Manager File menu, and type WINHELP.
2. Open your help file C:\VBLABS\HELP\HELP1.HLP.

   Test your help file connections between topics, keywords, and popups.

Exercise 5: Creating the Visual Basic Program

1. Start Visual Basic, and place three text boxes on a form.
2. Assign the HelpContextId property for each text box to 1, 2, and 3, respectively.
3. From the Options menu, choose Project. Assign the HelpFileName C:\VBLABS\HELP\HELP1.HLP.
4. Run your application.
5. Press F1 from each of the text boxes. Notice how Help is automatically invoked with the correct context number.

Exercise 6: Calling WinHelp from a Visual Basic Program

Visual Basic automatically invokes Help for you when the user presses F1. If you want to invoke Help at other times, you can explicitly call WinHelp. In this exercise, you will create a Help menu with three submenus. The first submenu will display the index page of your help file. The second submenu will display Help on "commands" and the third submenu will display help on the topic 2.

1. Add a Help menu with two submenus named Index and Search respectively.
2. Add the following declarations to a module of your Visual Basic program:

```
Declare Function WinHelp Lib "User" (ByVal hWnd As Integer,
- ByVal lpHelpFile As String, ByVal wCommand As Integer,
- dwData As Any) As Integer

'Help Constants
Const HELP_CONTEXT = &H1          'Display topic
Const HELP_QUIT = &H2             'Terminate Help
Const HELP_CONTENTS = &H3         'Display index
Const HELP_PARTIALKEY = &H105     'Call the search engine

Dim rc As Integer
Dim Search As String
Const Helpfile = "C:\VBLABS\HELP\HELP1.HLP"
```

3. From the Index menu command, invoke the **WinHelp** function to get the contents of your help file.

```
rc = WinHelp (Form1.hwnd, HELPFILE, HELP_CONTENTS, ByVal 0&)
```

4. From the Search menu command, invoke the **WinHelp** function to execute a search in Help.

```
Search = "multitasking"
rc = WinHelp (Form1.hwnd, HELPFILE, HELP_PARTIALKEY, ByVal search)
```

You could also program the Search menu to search for a selected keyword as follows:

```
If TypeOf Screen.ActiveControl is TextBox Then
    Search = Screen.ActiveControl.SelText
Else
    Search = "Contents"
End If
```

Exercise 7: If You Have Time... Using the Common Dialog

Use the common dialog control to invoke Help. Refer to the Student Workbook for examples.

Optional Lab 15: Creating Graphical Effects

Lab Objective

The purpose of this lab is to implement graphics in a Visual Basic application.

In exercises 1A and 1B, you will create a graph using the GRAPH.VBX custom control from the Visual Basic Professional Edition.

In exercises 2A and 2B, you will simulate a 50 button control with a bitmap.

Exercise 1A: Testing the Solution—GRAPH.VBX

Run the solution in \VBLABS\GRAPHICS\GRAPH\SOLUTION.

When you are certain you understand the purpose of the application, open a new project and move on to the next exercise.

Exercise 1B: Using GRAPH.VBX

In this exercise you will create a graph using GRAPH.VBX. You will create a 3-D pie chart.

In this lab, you will focus on the following Objects:

GRAPH.VBX
- .DrawMode
- .NumPoints
- .GraphType
- .GraphStyle
- .GraphTitle
- .GraphData

1. Start a new project. Add the GRAPH.VBX custom control to the project. Visual Basic automatically starts a graphics server when you add the graph control to a project.
2. Select the graph icon from the toolbox, and draw it on the form. It creates a 2-D bar chart automatically using five random data points. This assists you in seeing the graph type and style with which you are working.
3. To see more than five points, change the NumPoints property. Set NumPoints to 6. It will now generate the chart using six random data points.

4. Change the type of graph to a 3-D pie chart by changing the GraphType property. You will now see a 3-D pie chart of six random data points.
5. Change the style of the graph to % Colored Labels by changing the GraphStyle. This will place colored labels of percentages on the chart, with lines.
6. Add a title "My First Pie Chart" to the Graph by changing the GraphTitle property. Resize the form and chart if desired.
7. To use your own data instead of the random points, select the property GraphData, and type your own values. Press ENTER after each data point. For this exercise, use the following points:

   10, 15, 25, 5, 35, 10
8. To change the graph at run time, place a button on the form. Set the name to CmdBar, and set the Caption to "Change to Bar Chart".
9. In cmdBar_Click, write the following code to alternate between a pie chart and a bar chart.

```
Sub cmdBar_Click ()
    ' Bar chart
    If cmdBar.Caption = "Change to Bar Chart" Then
        Graph1.DrawMode = 1 ' Clear
        Graph1.GraphType = 4 'Bar Chart
        Graph1.GraphTitle = "My First Bar Chart"
        cmdBar.Caption = "Change to Pie Chart"
        Graph1.DrawMode = 2 ' Draw
    Else
        Graph1.DrawMode = 1 ' Clear
        Graph1.GraphType = 2  'Pie Chart
        Graph1.GraphTitle = "My First Pie Chart"
        cmdBar.Caption = "Change to Bar Chart"
        Graph1.DrawMode = 2 ' Draw
    End If
End Sub
```

Exercise 2A: Testing the Solution—Simulating 50 Buttons

In this exercise you will simulate a 50-button control using a bitmap. By using a user-defined coordinate system and detecting the X, Y coordinates of your mouse pointer, you can calculate which button is "pressed."

- Run the solution in C:\VBLABS\GRAPHICS\BUTTONS.

  Click a button, and note the message that is displayed. The 50 buttons are all in a single bitmap that is in a picture control. The MouseDown event for the picture control converts the mouse location to a number and displays that to you by means of a message box.

  When you are comfortable you know how the program works, open the partial solution in C:\VBLABS\GRAPHICS\BUTTONS\PARTIAL.

Exercise 2B: Coding the Application—Simulating 50 Buttons

Be sure you have opened the partial solution in C:\VBLABS\GRAPHICS\BUTTONS.

1. Create a picture box, size it, and set the picture property to \VBLABS\GRAPHICS\BUTTONS\MEMORY.BMP. Set the AutoSize property to **True**.

2. You'll want to set up a user-defined scale on the picture box. The idea is to establish a coordinate system that corresponds to the bitmap. The bitmap is a "5x10" bitmap, so to speak, so you want a 5x10 coordinate system. This way, any mouse event that occurs on the picture box will be reported with X and Y coordinates in the range 0–5 for Y and 0–10 for X.

   You'll need to set ScaleLeft and ScaleTop as well. Here, you're establishing X and Y values for the origin. You may either use a 0,0 or 1,1, but this will affect what you do later.

   a. Set ScaleMode to UserDefined
   b. Set ScaleWidth to 10
   c. Set ScaleHeight to 5
   d. Set ScaleLeft to 0
   e. Set ScaleTop to 0

   Note that these properties could be set in Form_Load instead of at design time. Also, the four statements to set ScaleWidth, ScaleHeight, ScaleLeft, and ScaleTop can be reduced to a single call to the **Scale** method.

3. Code the Picture_MouseDown event to display a message box indicating which button was "pressed."

   In the MouseDown event, the incoming arguments X and Y tell the function where the mouse pointer was. The values of X and Y will be in the range you specified in the previous step 0–5 for Y, and 0–10 for X. Note that X and Y are single precision numbers, which means you'll get values for X like 5.551.

   You can determine the current row by using `Row = Int(Y)`. You can determine the current column by using `Col = Int(X)`.

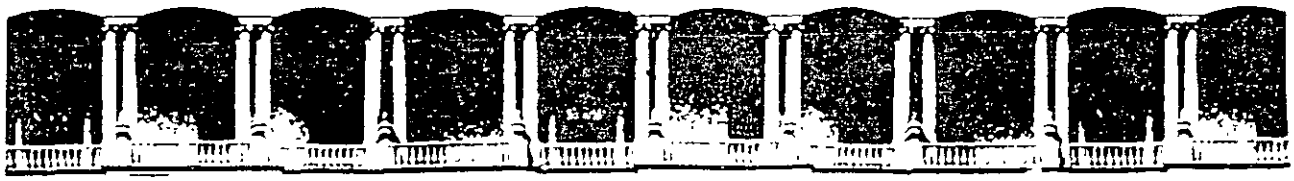
   The calculations to determine which "button" corresponds to the mouse pointer are as follows if ScaleLeft and ScaleTop are both set to 0:

   Button selected = (Col * 5) + Row + 1.

   However, you'll have to determine whether the mouse button was pressed while the mouse pointer was between buttons. Consider the picture control as being comprised of a grid of size 5 by 10. Each button occupies only 84% of the width of its grid cell and 74% of the height. (These values were derived empirically.) It's easier to show by example than by explanation. Look at the solution if you need help.

   In most systems, a button's code is not executed until a MouseUp occurs and the mouse pointer is still on the button. You could expand this application to trap the X,Y location in the MouseDown and execute the code in the MouseUp only if the user was still in the same location.

4. Display the current X, Y mouse coordinates in the label that is already provided. Add this code to the Picture_MouseMove event so the label is constantly getting updated.



FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

MATERIAL DIDACTICO

COMPLEMENTO

V I S U A L B A S I C

A V A N Z A D O

JULIO, 1996

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg Cuauhtémoc 06000 México, D.F. APDO. Postal M-2285
Teléfonos: 512-8955 512-5121 521-7335 521-1987 Fax 510-0573 512-5121 521-4020 AL 26

Getting Started

Microsoft® ODBC Desktop Database Drivers

Version 2.0

Open Database Connectivity for the
Microsoft Windows™ Operating System

Microsoft Corporation

Document No. DB61492-0994
Printed in the United States of America.

Contents

CHAPTER 1

Welcome to the Microsoft ODBC Desktop Database Drivers Pack

Welcome to the Microsoft® Open Database Connectivity (ODBC) Desktop Database Drivers Pack version 2.0, a set of ODBC drivers that you can use to access the most common database formats found on your desktop computer system.

You can use the ODBC Desktop Database Drivers with your ODBC-enabled Windows-based applications. Through these drivers, your applications can access data in seven database formats: Microsoft Access®, Microsoft FoxPro®, dBASE, Paradox, Btrieve, Microsoft Excel®, and formatted text files.

You can also use the ODBC Desktop Database Drivers to access data with custom applications written with the C, C++, or Visual Basic programming languages and the ODBC Software Development Kit.

About This Manual

This manual describes ODBC and the ODBC drivers for accessing desktop data. It explains how to set up ODBC drivers on your computer system. It also provides information about redistributing the drivers for software developers who are licensed to do so.

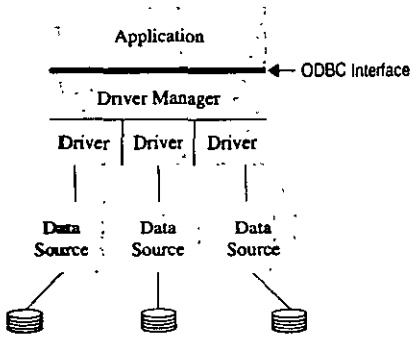
This manual assumes that you are familiar with the basic terminology and procedures for using Microsoft Windows™ version 3.1 or later (for 16-bit drivers) or Microsoft Windows NT version 3.5 or later (for 32-bit drivers). It also assumes that you have set up the Windows operating environment.

How ODBC Works

ODBC has four components: an ODBC-enabled application, the ODBC Driver Manager, ODBC database drivers, and data sources.

- **Application** A program that accesses data through ODBC drivers. For example, a program that retrieves dBASE data with the ODBC driver for dBASE files and Microsoft Excel data with the ODBC driver for Microsoft Excel files.
- **Driver Manager** Manages drivers on behalf of an application.
- **Drivers** Process ODBC requests and return data to the application. If necessary, drivers modify an application's request into a form that is understood by the data source.
- **Data sources** The files or databases accessed by a driver. For the ODBC Desktop Database Drivers, a data source is a default directory or a single database file. If the data source is a directory, the data source includes the files in that directory, and the information needed to access the data in those files, such as the format of a text file.

The following diagram shows the relationship between the four required components.



When you set up the ODBC Desktop Database Drivers, the Driver Manager and one or more ODBC drivers are installed on your desktop computer system. Your ODBC-enabled applications can then use the drivers to access your existing data sources.

Using Applications with ODBC Drivers

Because many data sources differ in the SQL statements and capabilities they support, ODBC provides several levels of functionality that a driver can support. The ODBC Desktop Database Drivers support the functionality required by most ODBC-enabled applications. Furthermore, the drivers provide (with few exceptions) a consistent level of functionality for all the databases represented by the ODBC Desktop Database Drivers.

For more information about the ODBC functions and SQL statements supported by each driver, see the driver-specific Help files, described in "Getting Help about ODBC Drivers," in Chapter 2.

CHAPTER 2

Using the ODBC Desktop Database Drivers

The ODBC Setup program decompresses and copies ODBC drivers and other ODBC files to your hard disk. Before you can start using ODBC drivers, you must use ODBC Setup

In this chapter, you'll learn how to set up ODBC on your hard disk, configure data sources, and get help about the drivers.

Hardware and Software Requirements

To use the ODBC Desktop Database Drivers, you need:

- An IBM-compatible personal computer with an 80386 or higher processor and a VGA or higher-resolution graphics card
- Microsoft Windows™ operating system version 3.1 or later (for 16-bit drivers) or Microsoft Windows NT operating system version 3.5 or later (for 32-bit drivers).
- A hard disk with 6MB of free disk space for 16-bit or 32-bit drivers.
- At least 4MB of random-access memory (RAM).

Application developers who write C language programs that call functions in the drivers also need the Microsoft Open Database Connectivity Software Development Kit.

Note Some drivers have additional software requirements. See Appendix A for a list of driver-specific software requirements.

Setting Up ODBC Desktop Database Drivers

The ODBC Setup program allows you to set up drivers from the distribution disk or automatically from a network drive. (The automatic setup procedure can only be used to set up 16-bit drivers, not 32-bit drivers.)

Setting Up ODBC Desktop Database Drivers from Disk

The ODBC Setup program allows you to select the drivers you want and installs them on your disk. In addition to the drivers, Setup installs the ODBC files needed to configure and use the drivers with ODBC-enabled applications.

The ODBC Desktop Database Drivers Pack includes two setup disks: one for 16-bit drivers and one for 32-bit drivers. Each disk contains all the files needed to set up the associated drivers. Make sure you use the 16-bit setup disk to set up 16-bit drivers and the 32-bit setup disk to set up 32-bit drivers. The setup process you need to follow (as explained below) is the same for each disk.

▶ To set up ODBC Desktop Database Drivers

1. If Windows 3.1 or later, or Windows NT 3.5 or later, is not already running, do the following:
   - For Windows 3.1 or later, type **win** at the command prompt and then press ENTER.
   - For Windows NT 3.5 or later, you need only boot your workstation and log in. If you are running a dual-boot system, then reboot your workstation and choose "Microsoft Windows NT" during the system countdown.
2. After your Windows operating system begins running, close any open applications except the Program Manager.
3. From the Program Manager File menu, run SETUP.EXE.
4. Follow the instructions displayed in the Setup program's dialog boxes.

Note Microsoft ODBC Setup installs ODBC components and drivers in your Windows directory and the directory that contains system dynamic-link libraries (DLLs) (for Windows 3.1 or later, usually the C:\WINDOWS and C:\WINDOWS\SYSTEM directories; for Windows NT 3.5 or later, usually the C:\WINDOWS and C:\WINDOWS\SYSTEM32 directories).

---

5. Select the drivers that you want to install from the list and then click OK.
6. If Setup detects a copy of a driver already installed and it is the same or a newer version than the one you are installing, Setup displays a warning. Do one of the following:
   - To overwrite the existing driver, click the Yes button.
   - To keep the existing driver, click the No button.

7. After the installation of the ODBC Desktop Database Drivers is complete, you can add data sources for the drivers you installed. To add data sources now, follow the instructions in "Adding, Modifying, and Deleting Data Sources," later in this chapter. To exit the Setup program without adding data sources, click the Close button.

Note You must add one or more data sources for each ODBC driver before you can access data with that driver. For more information, see "Adding, Modifying, and Deleting Data Sources," later in this chapter.

---

8. When Setup is complete, click the OK button, or press ENTER. You are returned to Windows.

Setting Up 16-Bit ODBC Desktop Database Drivers Automatically

In a multiuser environment, a network administrator can configure the 16-bit ODBC Desktop Database Drivers to be set up automatically from a network drive. Automatic setup provides all users with the same ODBC drivers and data sources.

Note The automatic setup procedure can be used to set up the 16-bit ODBC Desktop Database Drivers, but not the 32-bit drivers. The 32-bit ODBC drivers must be set up from disk (see "Setting Up ODBC Desktop Database Drivers from Disk").

▶ To set up ODBC Desktop Database Drivers automatically

1. Ask your network administrator for the address of the network drive containing the ODBC Desktop Database Drivers.
2. If Windows 3.1 or later is not already running, type **win** at the command prompt and then press ENTER.
3. After your Windows operating system begins running, close any open applications except the Program Manager.
4. Connect to the network drive containing the ODBC Desktop Database Drivers. If you have any questions, ask your network administrator.
5. From the File menu, choose Run (ALT, F, R).
6. Type the full path of the ODBC Setup program, followed by the **/auto** switch, and then press ENTER. For example, if your network drive is O: and the ODBC Setup program is in the \ODBC directory, type **o:\odbc\setup /auto**. After the software is set up, you are returned to Windows.

▶ **To configure the ODBC Desktop Database Drivers for automatic setup**

1. Connect to a network directory to which all users have access.
2. Create a directory from which other users will install the ODBC Desktop Database Drivers. For example, if the network drive is O: and other users will install the software from the \ODBC directory, type **mkdir o:\odbc**.

Note The ODBC Desktop Database Drivers Pack includes two setup disks: one for 16-bit drivers and one for 32-bit drivers. Only the 16-bit driver setup disk should be used for automatic setup of the 16-bit drivers.

---

3. Copy all the files on the ODBC Desktop Database Drivers disks (except the 32-bit driver setup disk) to the newly created directory. For example, if other users will install the software from the O:\ODBC directory, type **copy a:*.* o:\odbc** for each disk.
4. On a system that does not have ODBC installed, install the ODBC Desktop Database Drivers and add all the data sources your users need. For more information, see "Setting Up ODBC Desktop Database Drivers," earlier in this chapter
5. Copy the ODBC.INI and ODBCINST.INI files from the system used in the previous step to the network directory created in step 2. For example, if the network drive is O: and other users will install the software from the \ODBC directory, type:

   copy c:\windows\odbc.ini o:\odbc
   copy c:\windows\odbcinst.ini o:\odbc

   If Windows is installed in a directory other than C:\WINDOWS, use that directory
6. With a text editor such as Notepad, add the following two lines to the [Files] section of the SETUP.LST file on the network directory:

   odbcinst.ini = odbcinst.ini
   odbc.ini = odbc.ini

Additional Setup Information

ODBC Setup copies the ODBC driver files into the directory that contains system DLLs (for Windows 3.1 or later, usually the C:\WINDOWS\SYSTEM directory; for Windows NT 3.5 or later, usually the C:\WINDOWS\SYSTEM32 directory). For Windows 3.1 or later, it also creates two files in your Windows directory:

- ODBCINST.INI stores information about the ODBC drivers installed on your system.
- ODBC.INI stores information about data sources created through the ODBC control panel device or ODBC Administrator.

For Windows NT 3.5 or later, this information is stored in the registry.

The ODBC control panel device is added to the [MMCPL] section, or the CONTROL.INI file in your Windows directory (for Windows 3.1 or later) or the registry (for Windows NT 3.5 or later). Setup does not modify your AUTOEXEC.BAT, CONFIG.SYS, or WIN.INI files.

If you already have ODBC drivers on your system, installing the ODBC Desktop Database Drivers:

- Preserves your existing data sources.
- Preserves your existing ODBC drivers and adds the ODBC Desktop Database Drivers
- Automatically overwrites existing ODBC components *only* if the components shipped with the ODBC Desktop Database Drivers are newer.

Adding, Modifying, and Deleting Data Sources

A data source associates a particular ODBC driver with the data you want to access through that driver. For example, you might create a data source to use the ODBC dBASE driver to access one or more dBASE files found in a specific directory on your hard disk or a network drive.

You can create multiple data sources, each one associating a driver with some data you want to access using that driver. You need to give each data source a name that uniquely identifies that data source. For example, if you create a data source for a set of dBASE files that contain customer information, you might name the data source "Customers." Applications typically display data sources names for users to choose from.

▶ **To add a data source**

1. In the Main group in the Program Manager window, double-click the Control Panel icon. In the Control Panel window, double-click the ODBC icon.

Note For Microsoft Windows NT 3.5 or later when using 16-bit drivers, start the ODBC Administrator by double-clicking the Microsoft ODBC Administrator icon in the Microsoft ODBC group

---

2. Click the Add button.

   In the Add Data Source dialog box, the Installed ODBC Drivers list contains the names of currently installed drivers. (To install additional ODBC drivers, use the Drivers button in the Data Sources dialog box.)
3. From the Installed ODBC Drivers list, select the driver for the data source to use.

4. Click the OK button.

   A driver-specific dialog box appears.

5. Enter information about the data source, such as its name and file version (for example, dBASE III or dBASE IV).

   Each driver-specific data source dialog has a Help button that provides information about adding a data source for the driver you have chosen.

6. Click the OK button

▶ **To modify a data source**

1. In the Main group in the Program Manager window, double-click the Control Panel icon. In the Control Panel window, double-click the ODBC icon.

   Note For Microsoft Windows NT 3.5 or later when using 16-bit drivers, start the ODBC Administrator by double-clicking the Microsoft ODBC Administrator icon in the Microsoft ODBC group.

2. Select a data source from the Data Sources (Driver) list.
3. Click the Setup button.

   A driver-specific dialog box appears.

4. Modify the data source information, such as its name and file version (for example, dBASE III or dBASE IV).

   Each driver-specific data source dialog box has a Help button that provides information about modifying a data source for the driver you have chosen.

5. Click the OK button.

▶ **To delete a data source**

1. In the Main group in the Program Manager window, double-click the Control Panel icon. In the Control Panel window, double-click the ODBC icon

   Note For Microsoft Windows NT 3.5 or later when using 16-bit drivers, start the ODBC Administrator by double-clicking the Microsoft ODBC Administrator icon in the Microsoft ODBC group.

2. Select a data source from the Data Sources (Driver) list.
3. Click the Delete button.
4. A message box appears asking you to verify that you want to remove the data source you have selected:
   - Click the Yes button to delete the data source.
   - Click the No button to return to the Data Sources dialog box without deleting the data source.

Getting Help About ODBC Drivers

One Help file covers all of the ODBC Desktop Database Drivers. This file (ODBCJET.HLP) contains information about each driver. For example, the Help file tells you how to set up each driver, what functionality it supports, and how it is implemented. Most information in the Help file applies to all of the ODBC Desktop Database Drivers. When information is applicable to one or more, but not all, drivers, the Help file identifies the driver(s) to which the information applies. The Help file contains three levels of information:

- **For All Users** How to set up and use the driver.
- **For Advanced Users** What SQL statements and data types the driver supports.
- **For Programmers** How the driver is implemented and how to call the driver programmatically.

The driver Help file is placed in the directory that contains Windows system DLLs (usually the C:\WINDOWS\SYSTEM directory).

Using the Driver Help File

To use the driver Help file, you can either run it from the Program Manager or create an icon for the Help file and double-click the icon.

▶ **To run the driver Help file from the Program Manager**

1. If Windows is not already running, type **win** at the MS-DOS prompt.
2. From the File menu, choose Run (ALT, F, R).
3. Type **winhelp odbcjet.hlp**, and then press ENTER
4. After you have finished reading the Help file, choose Exit from the File menu (ALT, F, X).

▶ **To create an icon for the driver Help file**

1. Open the group to which you want to add the icon.
2. From the File menu in Program Manager, choose New (ALT, F, N).

   The New Program Object dialog box appears.

3. Select the Program Item option, and then choose the OK button.

   The Program Item Properties dialog box appears.

4. In the Description box, type a description of the driver Help. For example, **ODBC Driver Help**.
5. In the Command Line box, type **winhelp odbcjet.hlp**.
6. Choose the OK button. You can now double-click the icon to view the driver's Help file.

Using the Driver Help File from Your Application

If you are an application developer, you may want to access the driver Help file from your application or from your application's Help file.

To access the driver Help file from your C language application, call the **WinHelp** function in the Windows SDK and specify the full path of the Help file in the *lpszHelpFile* argument and HELP_CONTENTS in the *fuCommand* argument. For more information, see the *Windows SDK Programmer's Reference*

To access the driver Help file from your application's Help file, create a jump to the contents' context ID of the driver Help file. For more information, see the *Windows SDK Programmer's Reference*. The context ID of the contents of the ODBCJET HLP Help file is ODBCJetContents.

Microsoft Product Support Services

CompuServe

The Microsoft Connection on CompuServe provides online technical information for Microsoft products, including the ODBC Desktop Database Drivers With the Microsoft Connection, you can exchange messages with Microsoft professionals and experienced Microsoft users, and you can download free software—such as patches, tools, and add-ons—provided by Microsoft and CompuServe members

By using the Microsoft Connection, you can access the Microsoft Developer Services area You are encouraged to use this area to speak directly to Microsoft about developer-related issues. The Microsoft Developer Services area offers the following advantages:

- **Developer Forums** The ODBC section of the Windows Extension (WinEXT) forum provides information about the ODBC API, application development, and driver development (GO WINEXT). The ODBC Dsktop Drivers section of the WinEXT forum provides information about the use of the ODBC Desktop Database Drivers. The section leads for these sections are from Microsoft Product Support and can help answer your questions about the Desktop Database Drivers.
- **Confidential Technical Service Requests** Microsoft offers private (fee-based per incident) technical support to help solve your more complex development problems. For more details, see the Microsoft Developer Services area.
- **Microsoft Knowledge Base** This up-to-date reference tool, compiled by Microsoft Product Support, contains developer-specific technical information about Microsoft products (GO MSKB).

To connect to the Microsoft Connection, type **GO MICROSOFT** at the CompuServe "!" prompt. For information about establishing a CompuServe account, call (800) 848-8199, 8:00 A M to 10.00 P M EST Ask for operator 230 and receive a $15 connect-time usage credit.

Support Plans

Microsoft also offers pay-as-you-go telephone support from a Microsoft Engineer and a variety of annual paid support plans. In the United States, call (800) 936-3500 for further information on Microsoft support options. Outside of the United States, contact your local Microsoft Subsidiary for information regarding the availability of these products and services.

APPENDIX A

Additional Driver-Specific Software

With the exception of the ODBC Btrieve driver, no additional database software is required for any of the ODBC Desktop Database Drivers. For example, to access Paradox data, the Paradox driver does not require that you have a copy of Paradox.

The ODBC Btrieve driver requires the stand-alone Btrieve for Windows dynamic-link library (DLL), WBTRCALL.DLL. This file must be in the directory that contains Windows system DLLs (usually the C:\WINDOWS\SYSTEM directory).

▶ **To check if you already have a copy of this file on your hard drive**

1. In the Main group, choose the File Manager icon.
2. From the File menu, choose Search.

   The Search dialog box appears.
3. In the Search For box, type WBTRCALL.DLL.
4. In the Start From box, type C:\.
5. Choose the OK button.

If this file exists, copy it to the directory that contains Windows system DLLs (usually the C:\WINDOWS\SYSTEM directory).

Important If you already have a copy of WBTRCALL.DLL in your C:\WINDOWS\SYSTEM directory, contact your system administrator before copying over it. This file may be used by other programs on your computer.

If you don't have a copy of this file, contact Novell, Inc. at (800) 453-1267 or your local software dealer. Outside of the United States, contact your local Novell distributor.

APPENDIX B

Redistributing ODBC Desktop Database Drivers (Application Developers Only)

If you are a commercial application developer and you want to redistribute the ODBC Desktop Database Drivers, follow the instructions for redistribution of drivers in your license agreement. If your license agreement does not cover redistribution of drivers, contact your Microsoft sales representative for more information.

Setting Up ODBC Components for Your Application

The easiest way to set up the ODBC Desktop Database Drivers for your commercial application is to redistribute copies of the disk on which they were shipped and instruct users to follow the setup procedures listed in Chapter 2, "Setting Up the ODBC Desktop Database Drivers."

You can also integrate the setup of ODBC components into your application's setup program by calling functions in the ODBC Installer DLL (ODBCINST.DLL for 16-bit applications or ODBCCP32.DLL for 32-bit applications). The ODBC Installer DLL installs ODBC components and is shipped with the ODBC Desktop Database Drivers. It includes both high-level and low-level functions for installing ODBC components. For more information about the ODBC Installer DLL, see the *Microsoft ODBC SDK Programmer's Reference*.

Note The ODBC Installer shipped with the ODBC Desktop Database Drivers supports the functions described in Chapter 24, "Installer DLL Function Reference," of the *Microsoft ODBC SDK Programmer's Reference, Version 2.0*. It also supports several new functions that provide a graphical user interface from which users can install ODBC components and configure data sources.

ODBC Component Files

If you intend to redistribute all of the ODBC Desktop Database Drivers with your commercial application and you intend to use the ODBC Setup program, you must ship all the files on the ODBC Desktop Database Drivers disks. Otherwise, you need to ship only the files for the ODBC components your application requires; the ODBC.INF file you ship must reflect these components. For more information about the ODBC.INF file, see the *Microsoft ODBC SDK Programmer's Reference*.

Note You must ship the Driver Manager and the Installer DLL with any ODBC-enabled application. If you wish to configure 16-bit drivers in Windows NT, you must also ship the ODBC Administrator (ODBCADM.EXE).

The following table lists the files required by each component of the ODBC Desktop Database Drivers. Some files are required by more than one component. Although you need to ship only a single copy of a common file, you must list it in the section of the ODBC.INF file of every component that requires it.

ODBC component	Required files
Btrieve Driver	BTRV200.DLL CTL3DV2.DLL ODBCJET.HLP MSAJT200.DLL ODBCJT16.DLL COMPOBJ.DLL OLE2.DLL OLE2.REG OLE2NLS.DLL OLE2DISP.DLL SCP.DLL STDOLE.TLB STORAGE.DLL TYPELIB.DLL VBAR2.DLL VAEN2.DLL VAEN2.OLB VBAJET.DLL

ODBC component	Required files
dBASE Driver	XBS200.DLL(16-bit)/MSXB2032.DLL(32-bit) CTL3DV2.DLL(16-bit)/CTL3D32.DLL(32-bit) ODBCJET.HLP MSAJT200.DLL(16-bit)/MSJT2032.DLL(32-bit) ODBCJT16.DLL(16-bit)/ODBCJT32.DLL(32-bit) COMPOBJ.DLL OLE2.DLL OLE2.REG OLE2NLS.DLL OLE2DISP.DLL SCP.DLL(16-bit)/SCP32.DLL(32-bit) STDOLE.TLB STORAGE.DLL TYPELIB.DLL VBAR2.DLL(16-bit)/VBAR232.DLL(32-bit) VAEN2.DLL(16-bit)/VAEN232.DLL(32-bit) VAEN2.OLB(16-bit)/ VAEN232.OLB(32-bit) VBAJET.DLL(16-bit)/ VBAJET32.DLL(32-bit)
Driver Manager	CTL3DV2.DLL(16-bit)/CTL3D32.DLL(32-bit) ODBC.DLL(16-bit)/ODBC32.DLL(32-bit) ODBCCURS.DLL(16-bit)/ODBCCR32.DLL(32-bit)
Microsoft Access Driver	CTL3DV2.DLL(16-bit)/CTL3D32.DLL(32-bit) ODBCJET.HLP MSAJT200.DLL(16-bit)/MSJT2032.DLL(32-bit) ODBCJT16.DLL(16-bit)/ODBCJT32.DLL(32-bit) COMPOBJ.DLL OLE2.DLL OLE2.REG OLE2NLS.DLL OLE2DISP.DLL SCP.DLL(16-bit)/SCP32.DLL(32-bit) STDOLE.TLB STORAGE.DLL TYPELIB.DLL VBAR2.DLL(16-bit)/VBAR232.DLL(32-bit) VAEN2.DLL(16-bit)/VAEN232.DLL(32-bit) VAEN2.OLB(16-bit)/ VAEN232.OLB(32-bit) VBAJET.DLL(16-bit)/ VBAJET32.DLL(32-bit)

ODBC component	Required files
Microsoft Excel Driver	CTL3DV2.DLL(16-bit)/CTL3D32 DLL(32-bit) ODBCJET HLP ODBCJT16 DLL(16-bit)/ODBCJT32.DLL(32-bit) MSXL2016 DLL(16-bit)/MSXL2032.DLL(32-bit) MSAJT200.DLL (16-bit)/MSJT2032 DLL (32-bit) COMPOBJ DLL OLE2 DLL OLE2.REG OLE2NLS.DLL OLE2DISP.DLL SCP.DLL(16-bit)/SCP32.DLL(32-bit) STDOLE TLB STORAGE DLL TYPELIB.DLL VBAR2.DLL(16-bit)/VBAR232.DLL(32-bit) VAEN2.DLL(16-bit)/VAEN232.DLL(32-bit) VAEN2.OLB(16-bit)/ VAEN232 OLB(32-bit) VBAJET.DLL(16-bit)/ VBAJET32.DLL(32-bit)
Microsoft FoxPro Driver	CTL3DV2.DLL(16-bit)/CTL3D32 DLL(32-bit) ODBCJET.HLP MSAJT200.DLL(16-bit)/MSJT2032.DLL(32-bit) ODBCJT16 DLL(16-bit)/ODBCJT32 DLL(32-bit) XBS200.DLL(16-bit)/MSXB2032.DLL(32-bit) COMPOBJ.DLL OLE2.DLL OLE2.REG OLE2NLS DLL OLE2DISP DLL SCP DLL(16-bit)/SCP32 DLL(32-bit) STDOLE TLB STORAGE.DLL TYPELIB DLL VBAR2 DLL(16-bit)/VBAR232 DLL(32-bit) VAEN2.DLL(16-bit)/VAEN232 DLL(32-bit) VAEN2 OLB(16-bit)/ VAEN232 OLB(32-bit) VBAJET.DLL(16-bit)/ VBAJET32.DLL(32-bit)
ODBC Administrator	ODBCADM.EXE(16-bit)/ODBCAD32.EXE(32-bit) (ODBCADM EXE must be shipped for 16-bit applications that run on Windows NT)
ODBC Installer/ Control Panel Device	CTL3DV2.DLL(16-bit)/CTL3D32.DLL(32-bit) ODBC.INF ODBCINST DLL(16-bit)/ODBCCP32 DLL(32-bit) ODBCINST.HLP WINHELP.EXE WINHELP.HLP

ODBC component	Required files
ODBC Setup Program (16-bit or 32-bit)	ODBCSTP EXE SETUP.EXE SETUP.LST BOOTSTP EXE DRVSETUP DLL
Paradox Driver	CTL3DV2 DLL(16-bit)/CTL3D32.DLL(32-bit) ODBCJET.HLP MSAJT200 DLL (16-bit)/MSJT2032.DLL(32-bit) PDX200.DLL(16-bit)/MSPX2032.DLL(32-bit) ODBCJT16 DLL(16-bit)/ODBCJT32.DLL(32-bit) COMPOBJ.DLL OLE2.DLL OLE2 REG OLE2NLS DLL OLE2DISP.DLL SCP.DLL(16-bit)/SCP32 DLL(32-bit) STDOLE TLB STORAGE DLL TYPELIB.DLL VBAR2 DLL(16-bit)/VBAR232.DLL(32-bit) VAEN2.DLL(16-bit)/VAEN232.DLL(32-bit) VAEN2.OLB(16-bit)/ VAEN232.OLB(32-bit) VBAJET.DLL(16-bit)/ VBAJET32.DLL(32-bit)
Text Driver	CTL3DV2 DLL(16-bit)/CTL3D32 DLL(32-bit) ODBCJET.HLP ODBCJT16 DLL(16-bit)/ODBCJT32.DLL(32-bit) MSTX2016 DLL(16-bit)/MSTX2032 DLL(32-bit) MSAJT200 DLL (16-bit)/MSJT2032.DLL (32-bit) COMPOBJ DLL OLE2 DLL OLE2.REG OLE2NLS.DLL OLE2DISP.DLL SCP.DLL(16-bit)/SCP32 DLL(32-bit) STDOLE.TLB STORAGE.DLL TYPELIB.DLL VBAR2 DLL(16-bit)/VBAR232.DLL(32-bit) VAEN2.DLL(16-bit)/VAEN232 DLL(32-bit) VAEN2.OLB(16-bit)/ VAEN232.OLB(32-bit) VBAJET.DLL(16-bit)/ VBAJET32.DLL(32-bit)