Unit 04:

Basic Active X Controls:

Command Button: 

Command Button control is used to create buttons with a variety of uses on a form. It is probably the most widely used control. It is used to begin, interrupt, or end a particular process. A command button is the most basic way to get user input while a program is running. By clicking a command button, the user requests that a specific action be taken in the program. Or, in Visual Basic terms, clicking a command button creates an event, which must be processed in your program. Here are some command buttons that you would typically find in a program:
OK : Accepts a list of options and indicates that the user is ready to proceed.
Cancel : Discards a list of options.

Command Button Properties:

Appearance : Selects 3-D or flat appearance.
Cancel : Allows selection of button with **Esc** key (only one button on a form can have this property True).
Caption : String to be displayed on button.
Default : Allows selection of button with **Enter** key (only one button on a form can have this property True).
Font : Sets font type, style, size.
Picture : Return/sets a graphic to be displayed in control, if style is set to 1.
Style : Returns/sets the appearance of the control, whether standard (standard windows style) or graphical (with a custom picture).

Command Button Events:
Click : Event triggered when button is selected either by clicking on it or by pressing the access key.

Changing Command Button Properties
You can change command button **properties** (like those of all objects) in two ways:
- By adjusting property settings in the Properties window.
- By changing properties with program code.

Text Box Control Properties: 

A **Textbox** is used to display information entered at design time, by a user at run-time, or assigned within code. The displayed text may be edited. The **Textbox** control is one of the most versatile tools in the Visual Basic toolbox. This control performs two functions:
Displaying output such as operating instructions or the contents of a file on a form.
Receiving text such as names and phone numbers as user input.
How a text box works depends on how you set its properties and how you reference the text box in your program code.

Text Box Properties:

Alignment : Aligns caption within border.
Following are possible values for alignment property
0 – Left Justify
1 – Right Justify
2 – Center

Appearance : Selects 3-D or flat appearance.
BorderStyle : Determines type of border.
0 – None
1 – Fixed Single
Font : Sets font type, style, size.
Locked : Determines whether a control can be edited.
MaxLength : Limits the length of displayed text (0 value indicates unlimited length).
MultiLine : Specifies whether text box displays single line or multiple lines.
ScrollBars : Specifies type of displayed scroll bar(s).
0 – None
1 – Horizontal
2 – Vertical
3 – Both
PasswordChar: Hides text with a single character.
Text : Displayed text.
ToolTipText : Returns/sets the text displayed when the mouse is paused over the control.

Text Box Events:

Change : Triggered every time the **Text** property changes.
GotFocus : Triggered when the user entered the text box.
LostFocus : Triggered when the user leaves the text box. This is a good place to examine the contents of a text box after editing.
KeyPress : Triggered whenever a key is pressed. Used for key trapping, as seen in last class.

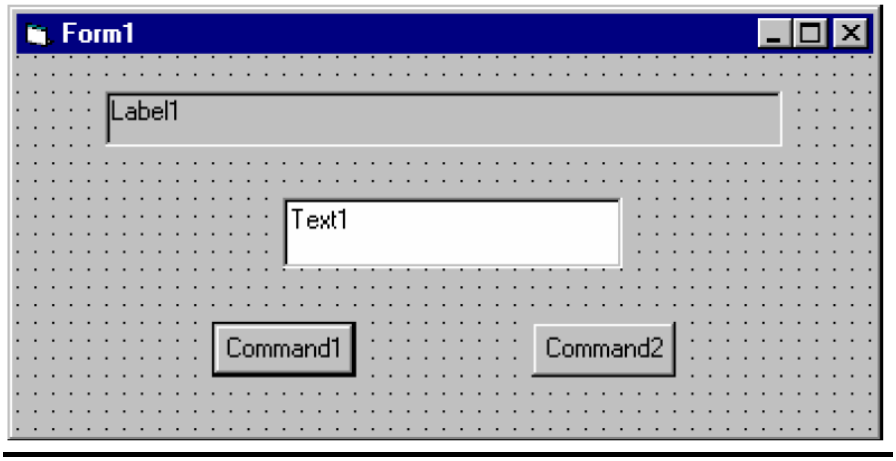
Text Box Methods:

SetFocus : Places the cursor in a specified text box.

Example – 1 (For Practical):
Password Validation

1. Start a new project. The idea of this project is to ask the user to input a password. If correct, a message box appears to validate the user. If incorrect, other options are provided.

2. Place a two command buttons, a label box, and a text box on your form so it looks something like this:

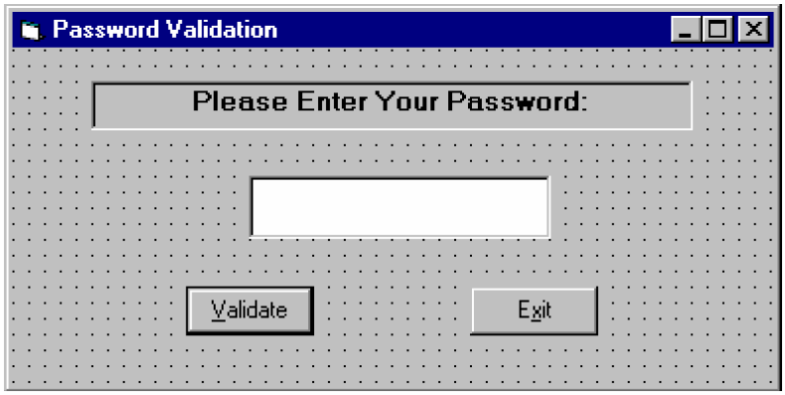


3. Set the properties of the form and each object.

Form1:		**Text1:**	
BorderStyle	1-Fixed Single	FontSize	14
Caption	Password Validation	FontStyle	Regular
Name	frmPassword	Name	txtPassword
		PasswordChar	*
		Tag	[Original password]
		Text	[Blank]
Command1:		**Command2:**	
Caption	&Validate	Cancel	True
Default	True	Caption	E&xit
Name	cmdValid	Name	cmdExit

Label1:	
Alignment	2-Center
BorderStyle	1-Fixed Single
Caption	Please Enter Your Password:
FontSize	10
FontStyle	Bold

Your form should now look like this:



4. Attach the following code to the **cmdValid_Click** event.

```
Private Sub cmdValid_Click()
'This procedure checks the input password
```

```
Dim Response As Integer
If txtPassword.Text = txtPassword.Tag Then
'If correct, display message box
MsgBox "You've passed security!", vbOKOnly + vbExclamation, "Access Granted"
Else
'If incorrect, give option to try again
Response = MsgBox("Incorrect password", vbRetryCancel + vbCritical, "Access
Denied")
If Response = vbRetry Then
txtPassword.SelStart = 0
txtPassword.SelLength = Len(txtPassword.Text)
Else
End
End If
End If
txtPassword.SetFocus
End Sub
```

This code checks the input password to see if it matches the stored value. If so, it prints an acceptance message. If incorrect, it displays a message box to that effect and asks the user if they want to try again. If Yes (Retry), another try is granted. If No (Cancel), the program is ended. Notice the use of **SelLength** and **SelStart** to highlight an incorrect entry. This allows the user to type right over the incorrect response.

5. Attach the following code to the **Form_Activate** event.

```
Private Sub Form_Activate()
txtPassword.SetFocus
End Sub
```

6. Attach the following code to the **cmdExit_ Click** event.

```
Private Sub cmdExit_Click()
End
End Sub
```

7. Try running the program. Try both options: input correct password (note it is case sensitive) and input incorrect password. Save your project. If you have time, define a constant, TRYMAX = 3, and modify the code to allow the user to have just TRYMAX attempts to get the correct password. After the final try, inform the user you are logging him/her off. You‟ll also need a variable that counts the number of tries (make it a Static variable).

Label Controls: 

A **label** is a control you use to display text that a user can't edit directly. **Label**, the simplest control in the Visual Basic toolbox, displays formatted text on a user interface form. Typical uses for the **Label** control include:

- Help text
- Program **splash screen** headings

 Formatted output, such as names, times, and dates
 Descriptive labels for other objects, including text boxes and list boxes.

Label Properties:

Name	**:** Returns the name used in code to identify an object.
Alignment	**:** Aligns caption within border. Following are possible values for alignment property 0 – Left Justify 1 – Right Justify 2 – Center
Appearance	**:** Selects 3-D or flat appearance.
AutoSize	**:** If True, the label is resized to fit the text specified by the caption property. If False, the label will remain the size defined at design time and the text may be clipped.
BackColor	**:** Returns/sets the background color of control.
BackStyle	**:** Indicates whether a background of label is transparent or opaque.
BorderStyle	**:** Determines type of border.
Caption	**:** String to be displayed in box.
Font	**:** Sets font type, style, size.
Enabled	**:** Returns/sets a value that determines whether an object can respond to user- generated events.
ForeColor	**:** Returns/sets the foreground color used to display text and graphics in control.
Visible	**:** Returns/sets a value that determines whether control is visible or hidden.
WordWrap	**:** Works in conjunction with AutoSize property. If AutoSize = True,
WordWrap	: True, then the text will wrap and label will expand vertically to fit the Caption. If AutoSize = True, WordWrap = False, then the text will not wrap and the label expands horizontally to fit the Caption. If AutoSize = False, the text will not wrap regardless of WordWrap value.

Label Events:

Click : Event triggered when user clicks on a label.
DblClick : Event triggered when user double-clicks on a label.

Creating Labels on a Form

To create a label control on a form, we surely refer to the toolbox window to select the label icon, shown as capital letter “A”. When the label control is selected, the label can be placed on a form by creating a rectangle with the mouse, which is held left button clicked. Once the left button is released, the label of size as the rectangle created is placed on the form.

Creating Labels in Code

You can also set label properties with program code, as shown in the following. The program codes below, when command1 button is clicked, will set the caption of label1 as “Welcome” and label2 as “Please enter your name below:”

```
Private Sub Command1_Click ()
Label1.Caption = "Welcome"
Label2.Caption = "Please enter your name below:"
End Sub
```

Option Buttons 

Option buttons provide the capability to make a mutually exclusive choice among a group of potential candidate choices. Hence, option buttons work as a group, only one of which can have a True (or selected) value.

Option Button Properties:

Caption **:** Identifying text next to button.
Font **:** Sets font type, style, size.
Value **:** Indicates if selected (True) or not (False). Only one option button in a group can be True. One button in each group of option buttons should always be initialized to True at design time.

Option Button Events:

Click : Triggered when a button is clicked. **Value** property is automatically changed by Visual Basic.

Checkbox Control 

Check boxes provide a way to make choices from a list of potential candidates. Some, all, or none of the choices in a group may be selected. **Checkbox** displays a list of choices and gives the user the **option to pick multiple items (or none at all) from a list of choices**.

Check Box Properties:

Caption **:** Identifying text next to box.
Font **:** Sets font type, style, size.
Value **:** Indicates if unchecked (0, vbUnchecked), Checked (1, vbChecked), or grayed out (2, vbGrayed).

Check Box Events:

Click **:** Triggered when a box is clicked. Value property is automatically changed by Visual Basic.

List Box Control Properties: 

A **list box** displays a list of items from which the user can select one or more items. If the number of items exceeds the number that can be displayed, a scroll bar is automatically added.

List Box Properties:

Appearance **:** Selects 3-D or flat appearance.
List **:** Array of items in list box.
ListCount **:** Number of items in list.
ListIndex **:** The number of the most recently selected item in list. If no item is selected, ListIndex = -1.
MultiSelect **:** Controls how items may be selected (0 -no multiple selection allowed,

1-multiple selection allowed, 2 - group selection allowed).

Selected **:** Array with elements set equal to True or False, depending on whether corresponding list item is selected.

Sorted **:** True means items are sorted in 'Ascii' order, else items appear in order added.

Text **:** Text of most recently selected item.

List Box Events:

Click **:** Event triggered when item in list is clicked.

DblClick **:** Event triggered when item in list is double-clicked. Primary way used to process selection.

List Box Methods:

AddItem **:** Allows you to insert item in list.

Clear **:** Removes all items from list box.

RemoveItem **:** Removes item from list box, as identified by index of item to remove.

Examples

lstExample.AddItem "This is an added item" ' adds text string to list
lstExample.Clear ' clears the list box
lstExample.RemoveItem 4 ' removes lstExample.List(4) from list box

Combo Box Control Properties:`

The **combo box** is similar to the list box. The differences are a combo box includes a text box on top of a list box and only allows selection of one item. In some cases, the user can type in an alternate response.

Combo Box Properties:

Combo box properties are nearly identical to those of the list box, with the deletion of the MultiSelect property and the addition of a Style property.

Appearance **:** Selects 3-D or flat appearance.

List **:** Array of items in list box portion.

ListCount **:** Number of items in list.

ListIndex **:** The number of the most recently selected item in list. If no item is selected, ListIndex = -1.

Sorted **:** True means items are sorted in 'Ascii' order, else items appear in order added.

Style **:** Selects the combo box form.
Style = 0, Dropdown combo; user can change selection.
Style = 1, Simple combo; user can change selection.
Style = 2, Dropdown combo; user cannot change selection.

Text **:** Text of most recently selected item.

Combo Box Events:

Click **:** Event triggered when item in list is clicked.

DblClick **:** Event triggered when item in list is double-clicked. Primary way used to process selection.

Combo Box Methods:

AddItem **:** Allows you to insert item in list.
Clear **:** Removes all items from list box.
RemoveItem **:** Removes item from list box, as identified by index of item to remove.

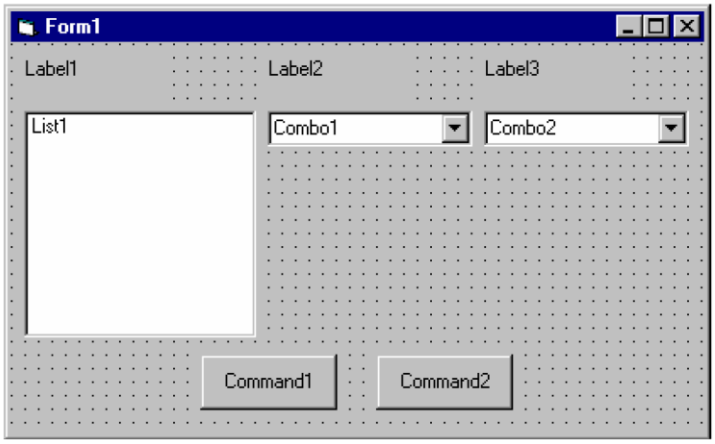
Examples
cboExample.AddItem "This is an added item" ' adds text string to list
cboExample.Clear ' clears the combo box
cboExample.RemoveItem 4 ' removes cboExample.List(4) from list box.

Example – 3 (For Practical):

Flight Planner

1. Start a new project. In this example, you select a destination city, a seat location, and a meal reference for airline passengers.

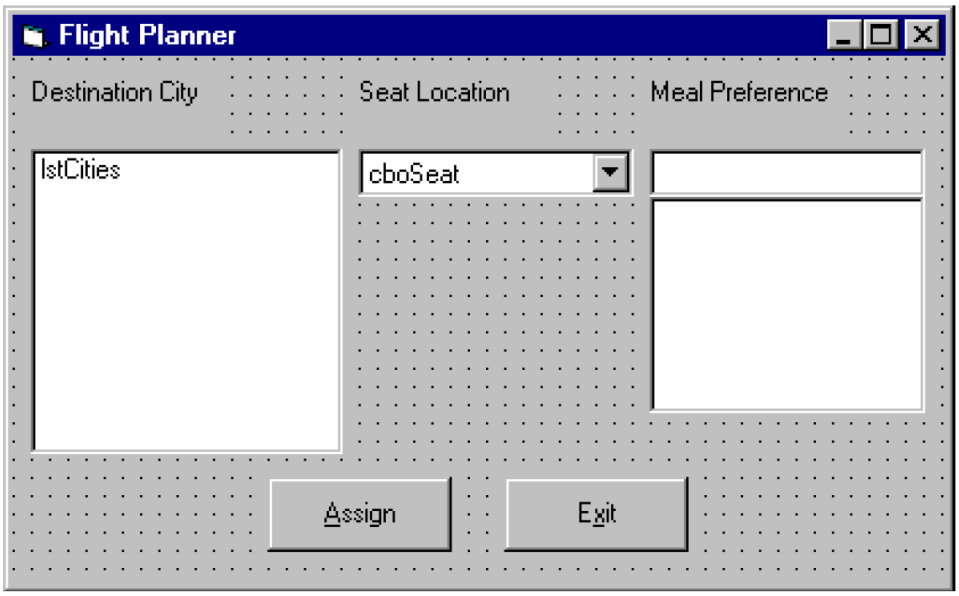
2. Place a list box, two combo boxes, three label boxes and two command buttons on the form. The form should appear similar to this:



3. Set the form and object properties:

Form1:		**Label1**:	
BorderStyle	**:** 1-Fixed Single	**Caption**	**:** Destination City
Caption	**:** Flight Planner	**Label2**:	
Name	**:** frmFlight	**Caption**	**:** Seat Location
List1:		**Label3**:	
Name	**:** lstCities	**Caption**	**:** Meal Preference
Sorted	**:** True		
Combo1:		**Command1:**	
Name	**:** cboSeat	**Caption**	**:** &Assign
Style	**:** 2-Dropdown List	**Name**	**:** cmdAssign
Combo2:		**Command2**:	
Name	**:** cboMeal	**Caption**	**:** E&xit
Style	**:** 1-Simple	**Name**	**:** cmdExit
Text	**:** [Blank]		

(After setting properties for this combo box, resize it until it is large enough to hold 4 to 5 entries.)
Now, the form should look like this:



4. Attach this code to the **Form_Load** procedure:

```
Private Sub Form_Load()
'Add city names to list box
lstCities.Clear
lstCities.AddItem "Mumbai"
lstCities.AddItem "Delhi"
lstCities.AddItem "Pune"
lstCities.AddItem "Latur"
lstCities.AddItem "Bangalore"
lstCities.AddItem "Aurangabad"
lstCities.AddItem "Nanded"
lstCities.AddItem "Nashik"
lstCities.AddItem "Nagpur"
lstCities.AddItem "Goa"
lstCities.AddItem "Jaipur"
lstCities.AddItem "ShriNagar"
lstCities.AddItem "Agra"
lstCities.AddItem "Kolkata"
lstCities.ListIndex = 0
'Add seat types to first combo box
cboSeat.AddItem "Aisle"
cboSeat.AddItem "Middle"
cboSeat.AddItem "Window"
cboSeat.ListIndex = 0
'Add meal types to second combo box
cboMeal.AddItem " Vegetarian "
cboMeal.AddItem " Meat"
cboMeal.AddItem "Chinease"
cboMeal.AddItem " Chicken "
cboMeal.AddItem "Fruit Plate"
cboMeal.Text = "No Preference"
End Sub
```

This code simply initializes the list box and the list box portions of the two combo boxes.

5. Attach this code to the **cmdAssign_Click** event:

```
Private Sub cmdAssign_Click()
'Build message box that gives your assignment
Dim Message As String
Message = "Destination: " + lstCities.Text + vbCr
Message = Message + "Seat Location: " + cboSeat.Text + vbCr
Message = Message + "Meal: " + cboMeal.Text + vbCr
MsgBox Message, vbOKOnly + vbInformation, "Your Assignment"
End Sub
```

When the **Assign** button is clicked, this code forms a message box message by concatenating the selected city (from the list box **lstCities**), seat choice (from **cboSeat**), and the meal preference (from **cboMeal**).

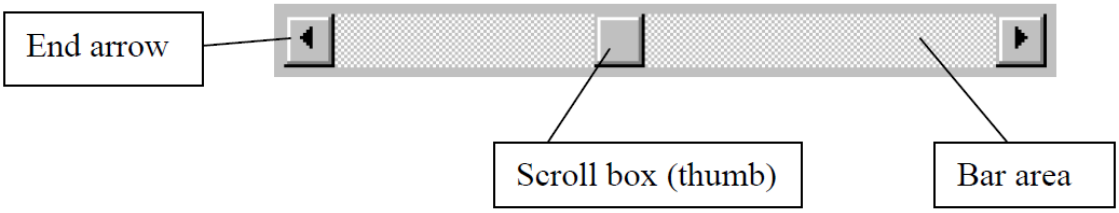
6. Attach this code to the **cmdExit_Click** event:

```
Private Sub cmdExit_Click()
End
End Sub
```

7. Run the application. Save the project.

Scroll Bar Control Properties: 

Horizontal and vertical scroll bars are widely used in Windows applications. Scroll bars provide an in-built way to move through a list of information. Both types of scroll bars are comprised of three areas that can be clicked, or dragged, to change the scroll bar value. Those areas are:



Clicking an end arrow increments the scroll box a small amount, clicking the bar area increments the scroll box a large amount, and dragging the scroll box (thumb) provides continuous motion. Using the properties of scroll bars, we can completely specify how one works. The scroll box position is the only output information from a scroll bar.

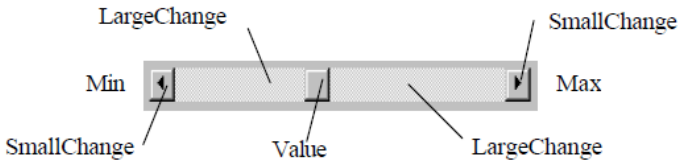
Scroll Bar Properties:

LargeChange : Increment added to or subtracted from the scroll bar Value property when the bar area is clicked.
Max : The value of the horizontal scroll bar at the far right and the value of the vertical scroll bar at the bottom. Can range from -32,768 to 32,767.
Min : The other extreme value - the horizontal scroll bar at the left and the vertical scroll bar at the top. Can range from -32,768 to 32,767.

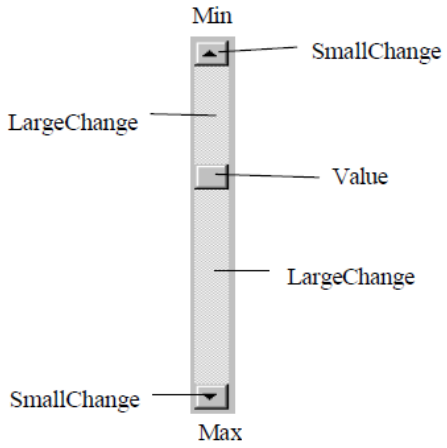
SmallChange : The increment added to or subtracted from the scroll bar Value property when either of the scroll arrows is clicked.
Value : The current position of the scroll box (thumb) within the scroll bar. If you set this in code, Visual Basic moves the scroll box to the proper position.
If you ever change the Value, Min, or Max properties in code, make sure Value is at all times between Min and Max or and the program will stop with an error message.
Scroll Bar Events:
Change : Event is triggered after the scroll box's position has been modified. Use this event to retrieve the Value property after any changes in the scroll bar.

Scroll : Event triggered continuously whenever the scroll box is being moved.

Properties for horizontal scroll bar:

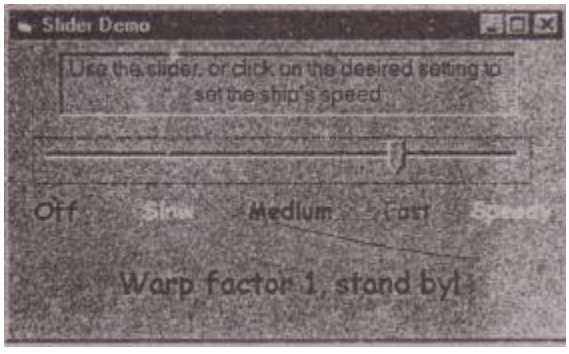


Properties for vertical scroll bar:



Slider Control- Properties:

The Slider control is similar to the ScrollBar control, but it doesn't cover a continuous range of values: The Slider control has a fixed number.of tick marks, which the developer can label (e.g., Off, Slow, Speedy,). The user can place the slider's indicator to the desired value. While the ScrollBar control relies on some visual feedback outside the control to help the user position the indicator to the desired value, the Slider control forces the user to select from; a range of valid values.



In short, the ScrollBar control should be used when the exact value isn't as important as the value's effect on another object or data element. The Slider control should be used when the user can type a numeric value and the value your application expects is a number in a specific range; for example, integers between a and 100, or a value between a and 5 inches in steps of 0.1 inches (0.0, 0.1, 0.2 inches, and so on, up to 5 inches). The Slider control is preferred to the TextBoxcontrol in similar situations because there's no need for data validation on your part. The user can only specify valid numeric values with the mouse.

Understanding Visual data manager:

The Visual Basic 6 Data Manager is a complete program (written in Visual Basic!) that ships with Visual Basic 6.0. This program can be used to create new Microsoft Access databases and edit, convert, compact, repair, encrypt, and decrypt existing databases. You can use Data Manager to create or delete data tables and indexes. You can also use the Visual Basic 6 Data Manager to perform simple data entry on data tables.
The Visual Basic 6 Data Manager can create databases in the Microsoft Access database format. It can also be used to attach to and perform field maintenance and data entry on Paradox, dBASE, FoxPro, Btrieve, and ODBC data sources. It can even attach to Excel spreadsheets and DOS Text files.

Using the Data Manager

Today you will learn how to use the Data Manager that is shipped with Visual Basic 6. This utility program gives you the power to create and maintain basic databases without leaving Visual Basic 6 design mode. You will learn how to use the Data Manager program to do the following:

- Create a new database
- Open existing databases
- Add data tables to a database
- Link to information contained in other databases
- Add fields and indexes to a database
- Set relationships between data tables
- Enter and find data in data tables
- Enter and save SQL statements
- Compact and repair databases
- Encrypt and decrypt databases

Plus, today is the day you start building your first extended Visual Basic 6 database project—The Company Database Project. You will use the Data Manager to construct the first data table in the database—the CompanyMaster table.

Microsoft Access Database Support

The Visual Basic 6 Data Manager provides nearly complete support for Microsoft Access databases. It allows you to create databases and create and delete tables, indexes, and data fields. You will not, however, be able to delete data fields that are used in indexes.

Advantages and Disadvantages of Using the Data Manager

- The Data Manager program has several key features that make it an excellent tool for constructing and maintaining databases for your Visual Basic 6 applications. First, you can launch this program directly from the Visual Basic 6 Add-Ins menu. As long as Visual Basic 6 is up and running in design mode, you can call up the Data Manager and create new databases, open existing databases, or modify data tables and indexes without having to leave Visual Basic 6 or close down your Visual Basic 6 project.

- Another advantage of having the Data Manager is that you can use it to do quick data entry into existing data tables. This allows you to quickly create test data for your Visual Basic 6 applications. Do you

need to see whether a database lookup routine you wrote really works? You can pop up the Data Manager add a few records to the appropriate data table, and then return to Visual Basic 6 and run your application.

- You can also use the Data Manager to compact out deleted records (in other words, physically remove spaces left by deleted records), and for those occasions when you get the dreaded "corrupted database" error, you can use the Data Manager to repair existing Microsoft Access type databases.

- You can even use Data Manager to build and test SQL statements. This is an extremely handy tool to have in order to test the logic of SQL statements as you need to incorporate them into your Visual Basic 6 code. These statements, once tested and working properly, can then be saved by Data Manager.

- A major disadvantage of using the Data Manager to create databases for your Visual Basic 6 applications is that it is not a complete database administration tool. Although you can use the Data Manager to construct and maintain data tables and indexes, you cannot print out data structures or index parameters.

- Even with this limitation, the Data Manager is a very useful tool. Let's go through a short course on how to use the Data Manager to construct and maintain Microsoft Access-type databases.