

About the Author

Chuck Easttom is a software developer and author. You can get to his homepage at www.chuckeasttom.com He currently holds over 20 different computer industry certifications and is continually working on more. He lives with his wife Misty, his son AJ and their two cats Sagan and Monte.

In his spare time (when he is not working, researching, or reading a computer book) Chuck likes to play chess, read voraciously and is a very bad golfer. You can find him on his home web site (that also has a pretty decent VB web site) at www.chuckeasttom.com

Here is a picture of the author at work at his computer (Well Sort of...)



Introduction

This book is meant for the beginner. Its primary purpose is to get you up and writing programs in the shortest possible time. For that reason I take a very practical approach (thus the title). My goal is that you be able to function as a competent Visual Basic programmer in at the end of this book, and write useful programs. That means that I don't go in depth into theory, or into advanced topics. However this book should provide you with the ground work necessary to move on to more advanced programming topics. While some of the code examples and lessons can be accomplished with Visual Basic 4, 5, or 6, much will require Visual Basic 6.0.

When you see the icon



It denotes a tip that can save you a lot of cursing at your code!

New in this edition:

Several things are new in this 3rd edition. The first, and most obvious, are the increased examples. More examples added to several chapters. Then we have the chapter on object orientation reworked and clarified. Finally a few completely new chapters have been added including a chapter on building your own Active X controls and building your own Active X dll's. About 30 extra pages are in this edition

There have also been a few formatting changes. All the examples are now highlighted so they can more easily be found.

Table of Contents

Chapter 1	Introduction to Visual Basic.....	4
Chapter 2	Introduction to Database Programming.....	29
Chapter 3	Essentials of Writing Code.....	56
Chapter 4	In Depth VB Coding.....	84
Chapter 5	Object Oriented Programming.....	111
Chapter 6	VB Printing and More.....	136
Chapter 7	ADO.....	159
Chapter 8	Active X Dll's.....	179
Chapter 9	Active X Controls.....	185
Chapter 10	TCP/IP programming.....	190
Appendix A	Keywords.....	193
Appendix B	Controls.....	196
Appendix C	Other Resources.....	201
Appendix D	DDE & OLE.....	206
Appendix F	Ten Commandments of Coding.....	214.

Chapter One

Introduction

to

Visual Basic

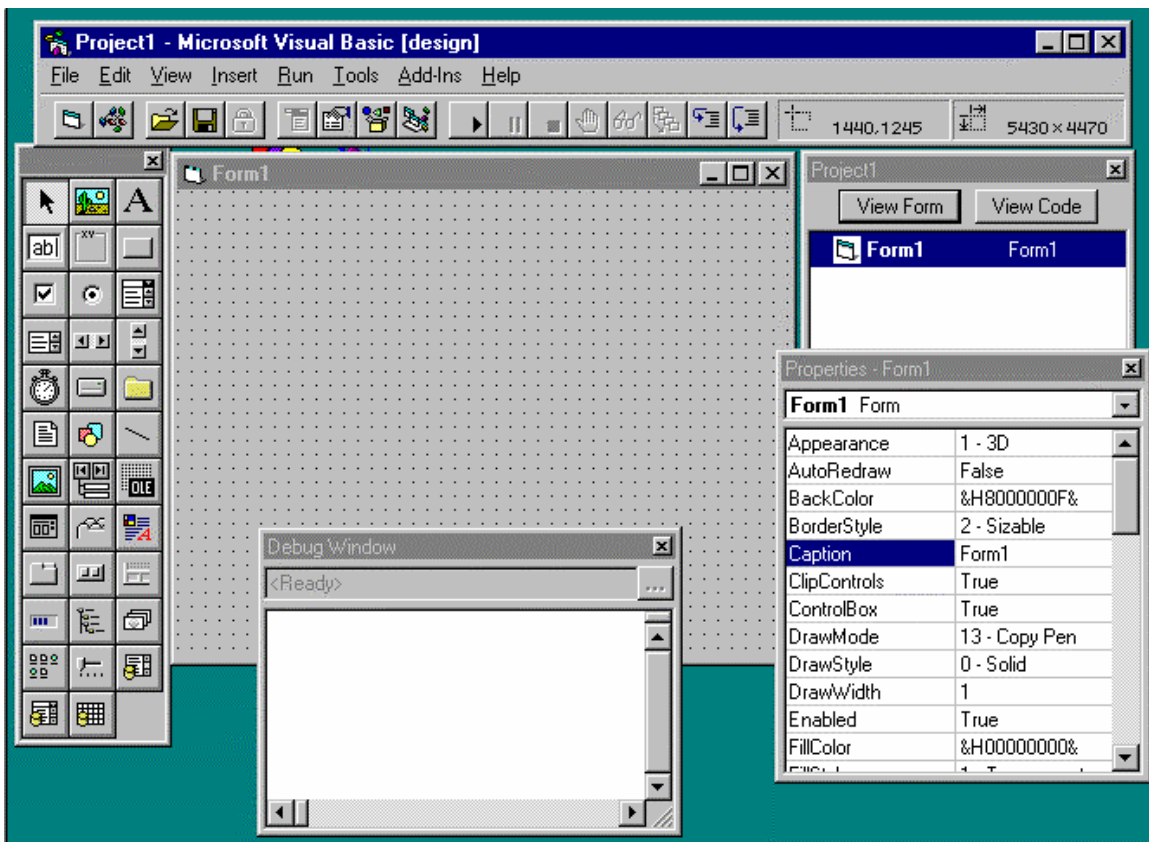
Chapter Objectives:

- Understand Controls and form
- Understand Events
- Understand Control Properties
- Become familiar with terminology
- Become comfortable with the development environment

Chapter 1: Introduction

This chapter is designed to introduce you to the essentials of Visual Basic. At the end of this chapter, you should have a basic understanding of forms, controls, properties, and events. You should also be getting comfortable with the programming interface and know how to move around in the tool bar, tool box, and drop down menus.

When you first launch Visual Basic you will be confronted with a screen that asks you what type of application you wish to make. For now we will always start with the first option “Standard EXE”. This denotes a Standard Executable or program. When you select this, you will then see the following screen:



This is referred to as the IDE or Integrated Development Environment. The top of the screen has the drop down menus and the tool bar. These contain various functions you will use when programming. Functions such as saving, running, and debugging your program. The tool bar looks like this:



The buttons on this toolbar are shortcut keys for common tasks. The first button will open a new form for you to use. You will be prompted to see if you want a blank standard form or one of the pre-made forms that Visual Basic starts with. The second will give you a new blank code module. The third will give you a standard windows dialogue box, allowing you to open an existing file. The fourth will save your current work. Towards the middle of the toolbar is a button with an image of an arrow pointing to the right. This causes your current project to run. Next to it are pause and stop buttons.

Immediately above it is the drop down menu. If you click on the drop down menu, a menu will drop down with other options you can click on. The tool bar simply provides rapid access to some of the most common functions in the drop down menu.

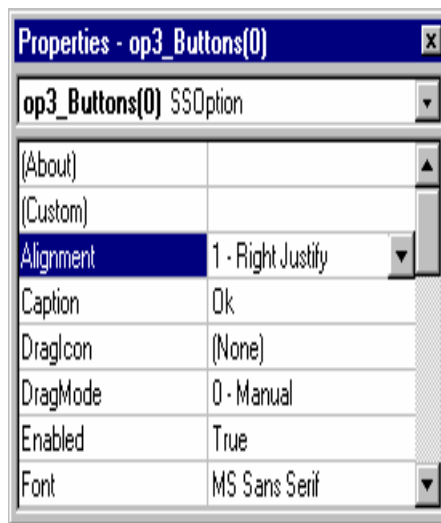
The window on the right is the properties window (see above). The window on the left is the toolbox (See below). The large gray area in the middle is a form. A form is where you lay out your programs visual interface. The interface the user sees is referred to as a Graphical User Interface or GUI (Pronounced G-U-I or “gooie”).



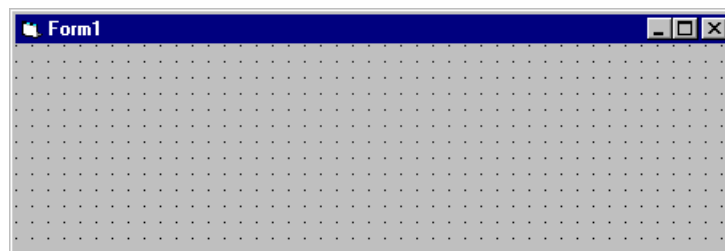
Also, note that at the top of the title bar is the name of the project you are working on. Visual basic gives the new project the default name of “Project1”. You will probably want to go to the drop down menu “File” and select the option “Save As” and save your project under a name that is more appropriate for the application you are developing.

Visual Basic is a RAD or Rapid Application Development tool. This means that you can develop professional applications much faster than you would with some other languages (C/C++, Pascal, COBOL). The way Visual Basic accomplishes this task is by giving you templates for most common items a program might need. You then merely change certain facets of the template to make it do what you want. These templates are called controls and can be found in your toolbox.

The facets of a control that you can change are called properties and are found in the properties box. You can access the properties box by clicking on “View” on the drop down menu at the top of your screen and then clicking on properties, or just by pressing the F4 key. It is important to realize that the properties window will display properties for whatever control currently has focus. You change focus by clicking once on the control you wish to have the focus. The properties are all listed in alphabetical order with the exception of the name property that is listed first. It is vital that you realize that the name property is what the computer will call your control, not what the user will see. The user will see the caption property (for those controls that have one.) Here is an example of a property window:



You can simply drag a control from your toolbox and then place it on the form. You then drag it to any spot on the form you want to place it. You can also simply double click on a control in the tool box and it will appear in the center of the form. All controls can also be stretched to alter their size. After you have placed the control on the form you can then proceed to set the properties in the properties window. Below is a picture of what a form looks like:



Common Controls and their properties

Some controls are used more often than others. The form is the most basic control. It is the container in which you place all the other controls. It is also what your user will see. The most important properties you will set for a form are its name (What the computer will call it); its caption (What appears in the title bar that the user will see), and its back color (what color do you want the background to be). You may also wish to set its border style (this determines if it can be resized by the user or not). Below are listed some other common controls and the properties you will be most concerned with:

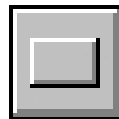
Text box: This control allows the end user to enter data. You will want to set its name (What the computer will call it), its text (the default text that initially appears when you run the program), and possibly its font if you wish a different font in the text box. You can also enter something into the text property as a default text. Finally you may want to set its max length property. This determines the maximum number of characters a user can type into the text box. You can also set its password character property to denote a symbol to show in lieu of passwords. The multi-line and scroll bar properties allow you to have a text box that can handle several lines of text.



Label: Labels are used to display information to the end user, but the end user cannot change any information in the label. The properties you will set in this control are the name (What the computer will call it), its caption (what will the user see), and its font.

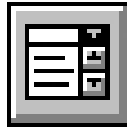


Command Button: Command buttons allow the user to click a button. A lot of the code that you write will be associated with command buttons. The properties you are most interested in are: its name (What the computer will call it), its caption (What the user will see), its style (Will your button have a text caption or a picture, if you choose the graphical style you must set the caption equal to nothing and select a picture in the picture property), and its font.



List Box: A list box simply allows you to list items for a user to select from. The property you are concerned with is the font. The items are listed using code and will be discussed later.

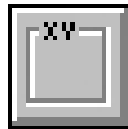
Combo Box: This is like a list box but you can set more options.



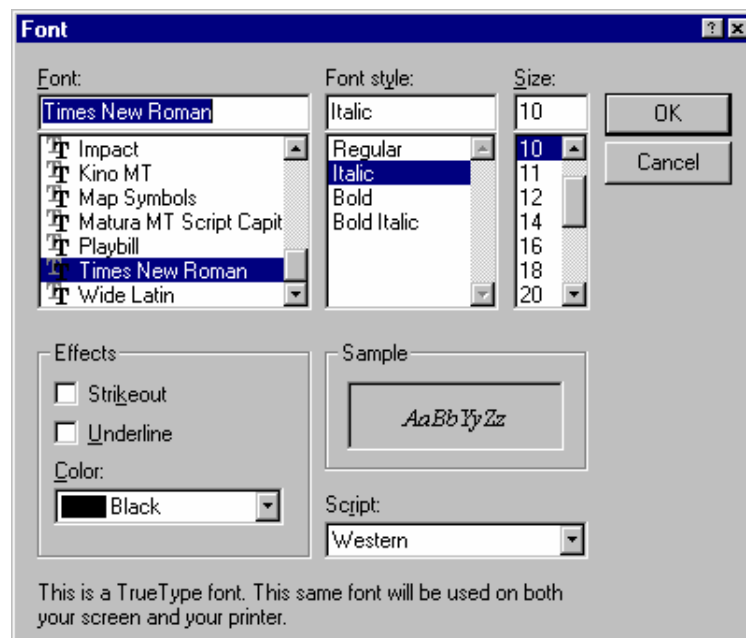
Picture box: Using this control you can place any bitmap (*.bmp), Windows MetaFile (*.wmf), or icon (*.ico) on your form. Starting with Visual Basic Version 5.0 you can also add in JPEG (*.jpg) and GIF (*.gif) image files. All you do is set the stretch property to true (that ensures that the picture you insert will be the size of your picture box) and then use the picture property to select the picture you want to use.



Frame: A frame is basically a container control. It is used to group other controls. The property you will use most with it is its caption.



Font: Many controls have a font property that lets you select the font for that control. Below is what you will see if you select the font property of any control:



A lot of the programming you do will be concerned with using code to alter the properties of some control. The way you do this is you write the control name you wish to alter followed by a period then the name of the property you want to change. A generic example of this is:

```
Controlname. Property = "whatever"
```

A specific example is:

```
Text1.text = "Howdy"
```

```
Form1.caption = "This is my form"
```

```
Picturebox1.image = "c:\folder\image.bmp"
```

```
Data1.recordsource = sSQL
```

```
Label1.caption = "This is a label"
```



Tip: If your code referencing a property does not work, it is most likely because you misspelled either the control or the property name.

Using this basic technique, you can write code to change the properties of any control you want. This is important to remember since much of the code you write in Visual Basic is concerned with altering the properties of various controls.

Example 1-1

Lets try a simple project right now. Start up Visual Basic (If you are using VB 5 or 6 then choose "standard.exe"). It should start off with one form in view. If you don't have the tool box in view, then you can go to the drop down ;menu and select "View" then select "toolbox".

Now on that form place 1 list box, 1 text box, and 1 command button. In the properties window (either select "view" and " properties" or press the f4 key) change the command buttons caption to "add text". Now if you will double click on the command button you will get its code window.

Now go to the "click" event. In it place this code:

```
List1.AddItem text1.text
```

Now run your program. You can do this by clicking on the run button in the toolbar, it looks like a play button on a CD player or by using the f5 key. You will be prompted to save it and give it a name.

Now every time you click on the command button, whatever you have typed into the text box will now be added to the list box!

It is important to note that every control has a default property. That is essentially its primary or most important property. If you refer to a control and do not specify the property you wish to refer to, Visual Basic will assume you intend to refer to the default property. For example, you could write the code above as:

```
Text1 = "Howdy"
```

Since the text property is the default property for the text box. However, in this book we will always explicitly state the property we are referring to.

Note: An event is just a function that the user calls by doing some action.

Glossary:

The following is a list of standard Visual Basic and programming terms that you will need to be familiar with. In programming, proper nomenclature (or naming) is simply vital. You cannot understand instructions and you cannot communicate with other programmers without an understanding of proper nomenclature

Project: This is all the files in your program. It includes forms, code modules, class modules, etc.

Forms: This is the control on which you place other controls and design the User Interface.

Controls: These are the various objects or templates that Visual Basic has for you to use in your program. Examples are the textbox, command button, list box, combo box, and picture box.

Code: This is the actual programming that you write.

Bug: an error in your program.

Debug: to remove errors in your program

Event: This is something the user does. The click event is a good example.

Sub routine: A sub section of code. Events are pre made subroutines in Visual Basic.

Code Module: A module of separate sub routines completely separate from a form.

IDE: Integrated Development Environment.

Application: A fancy word for the program you write

Statement: A single line of code that performs an action.

Function: A group of related statements that perform some action

The Drop Down Menu:

The drop down menu gives you access to several possible functions. On the next page are listed the main drop down options you see at the top of your screen and several of the sub options along with their functions. I do not cover all of the drop down menu options, only the ones that you as a beginner need. We will cover others later in this book.

File:

New Project: This opens a new blank project for you to begin working with

Open Project: This allows you to open an existing project. When you select this option, you will see a dialogue box that you can use to browse your computer to find the project.

Save Project As...: This allows you to save the project your working on under some name you select and in a folder/directory of your choosing.



Tip: Be very careful of what name you give your project and where you save it to. Putting all of a project's files (including database files) in a single folder with an appropriate name is the best idea.

Save Project: This will save the project you are working on under the last name you gave it and in the last location you saved it to.

Save Form: This works just like the Save project option, only it saves the specific form that currently has focus.

Save Form As: This option will allow you to save a specific form under a name of your choosing and in the folder/directory that you select.



Tip: As with the “Save Project” As option you want to make sure you are saving these forms (and any other project files) to the same location

Print: This allows you to print the source code for your project.

Make exe: This makes an executable program from your source code.

Exit: This will exit Visual Basic

Edit:

Undo: This option allows you to undo whatever action you last performed.



Tip: The Undo option exists in most windows programs. If you do something you wish you had not done, don't freak out. Simply go to “Undo”

Redo: This will let you redo whatever action you last used undo on.

Cut: This will cut the code or the control you have selected and place it on the clipboard.

Copy: This works much like the cut command except that the original is still in place.

Paste: This command lets you paste something that you have previously copied or cut.

Find: You can enter a key word and search all your project for it.

Find Next: It finds the next instance of whatever you last used find for.

Replace: This will go through your code replacing one word or phrase with another.

View:

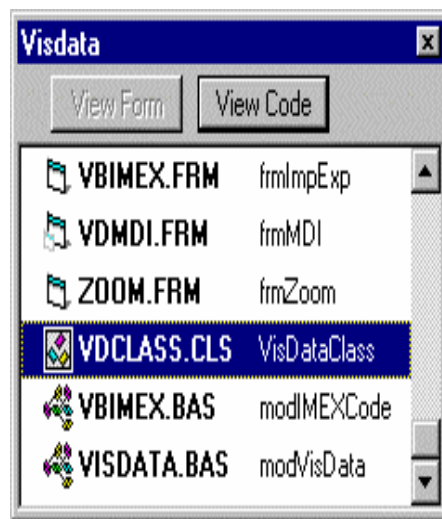


Tip: I always hear at least one student in panic yelling “I can’t find (project/form/code module/etc.). You probably just need to go to “View” and select that object to view.

Code: This will give you the underlying code for any object that currently has focus.

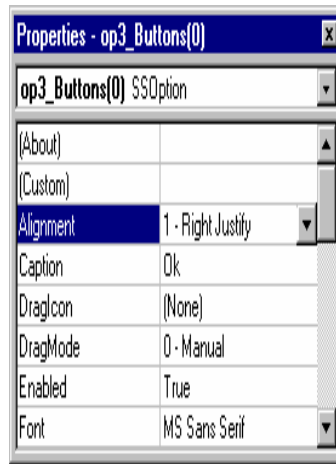
Object: This lets you view the object of the code you are currently viewing.

Project Explorer: Selecting this option will give you the window with a listing of all files in your project. It looks like this



Properties Window: This shows you the properties window. It will usually appear to the right hand side of the IDE. You can get to it using the drop down menu option

“View” and then “Properties” or the shortcut key “F4” It will display the properties of whatever control currently has focus.



Tool Box: You can use this to make your tool box visible. This is usually on the left hand side of the IDE



Project

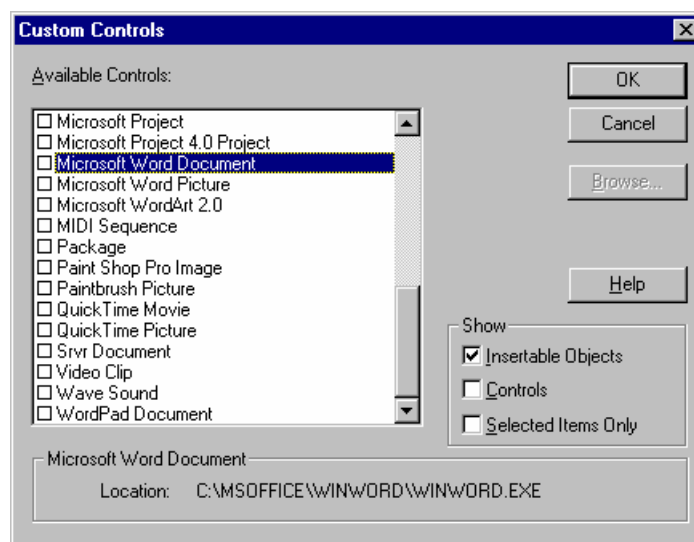
Add Form: This lets you add a new form to your project. When you select this, you will be prompted as to what type of form you wish to add. You will often select the blank form option, but Visual Basic has a number of pre made forms you may want to use.

Add Module: This option lets you add in a blank code module.

Add Control : This is another way to add in a control

Remove: This will remove whatever currently has focus, be it a form, control, or code module.

Components: Visual Basic has many more controls than you see in your toolbox. This option lets you select which controls you want to see in your toolbox. If you select this option you will see a window like the one below from which you can choose which controls you want available for your project:



Run:

This window simply lets you either run your program, pause it, or stop it.

Help:

This will let you ask questions and look up topics from Microsoft help.



Tip: Microsoft has included an awesome help file. If you ever see a keyword you don't understand look it up in the help file. You can learn a lot from the help file. Each version of VB seems to have an even better help file. VB 6.0 has an incredible help file.

Short Cut Keys:

The following are a few of the most commonly used shortcut keys. You may want to memorize them.

F7: View the Code window of an object

F4: View an objects property window

F3: Find

F5: Run

CTRL Z: Undo

CTRL C: Copy

CTRL X: Cut

CTRL V: Paste

Short Cut: You can also right click on your mouse and a pop up window will display the more commonly used functions. The window looks like this:

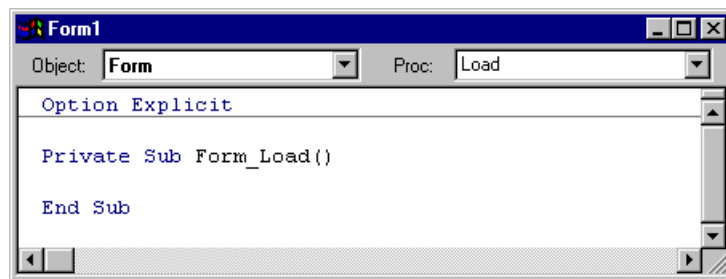


Events:

Events are things that happen. If you double click on any control you will be looking at an event, specifically the default event for that control. The default event for the Command Button is the click event. This means that any code you place in here will be executed when the user clicks that button. Let's do an example. Place a label on your form. By default, it will be named label1. Then place a command button on your form. By default, it will be named command1. Now double click on the command button so you can view the code in the click event. You should see something like this:

```
Private Sub Command1_Click()  
  
End Sub
```

All events are subroutines that Visual Basic has already defined for you. You must place code between these two lines. If you write code before the “Private Sub...” line or after the “End Sub” line, you will get an error. Below is a picture of the form load event:



Now let's put some code in our command1 click event:

```
Private Sub Command1_Click()  
  
    Label1.caption = "Howdy"  
  
End Sub
```

Now let's run our program. You can do this by selecting the button in the middle of the tool bar that looks like a blue arrow pointing to the right. You can also do this by pressing the F5 key, or by going to the drop down menu and clicking on “Run” then clicking on “Start”. You will be asked if you wish to save your new project and your form. If you select “yes”, you will be prompted to name the project and the form. You will also be prompted to select a location in which to save them.

Once the program is running, click on the command button and see what happens to the label. You should see the word “Howdy” appear in the label. This may seem rather

simple, but you will see that a great deal can be accomplished merely by changing properties in your code.

When a user causes any event to occur then the code that you have placed inside the subroutine for that event will be initiated. If there is no code inside the events subroutine then nothing will occur. All events begin with no code. You must place the code in them.

Each control will have several events you can choose from. Each event corresponds to an action that the user takes or to something that happens in the program. Below are some common events found in many controls and what they mean:

Click: This event is triggered whenever a user clicks that control. (this is most often used in command buttons, list boxes, and combo boxes.)



Tip: A lot of your code will be in click events. However, when you first open up a command button the first event displayed is the change event. I have seen lots of beginners put some nice code in an event and then spend hours trying to find out why it won't work, only to find they put the code in the wrong event.

Double Click: This event is triggered whenever a user double clicks on that control.

Mouse Move: When a user passes the mouse over a control this event is triggered.

Get Focus: This event is triggered when you first give focus to a control either by clicking on it or by tabbing to it.

Lost Focus: This event is triggered when you have had focus on a control and then attempt to move focus from that control.

Load: This event is only in the form control and it is initiated when the form is first loaded.

Naming Conventions:

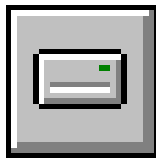
Microsoft has established some naming conventions you should use when naming controls. This makes for a more uniform code that is easy to read. By default each of your controls will have a name like: text1, label1, command4, etc. You must go to the "name" property of each control to change its name. Naming conventions simply make it much easier for another programmer to see what you are doing. Below is a chart of some Microsoft naming conventions:

Control Type	Naming Convention
Text Box	TxtWhatever
Command Button	CmdWhatever
Label	LblWhatever
Panel	PnlWhatever
Frame	FraWhatever
Combo Box	CmboWhatever
List Box	LstWhatever

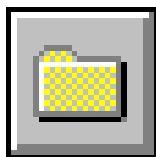
Using these naming conventions will make reading your code much easier. Yes I realize that it is easier and faster to just leave your controls with the default name. And while you are working on simple projects it won't really matter. But when you have to work on real world applications do you really want to spend time trying to figure out that txt1 refers to the account number when you could simply have named it txtAccountNum to start with?

Example 1-2

The following is a very simple program that will allow you to build an image viewer. The program is simple but lets you see how controls and their properties can be used. First, start a new project with a single form on it. On that form you will want to place a



directory list box



and a file list box



You will also need to place a picture box on the form



Now in the change event of the drive list box place this code:

```
Sub drive1_change ()
```

```
Dir1.path =drive1.drive
```

```
End sub
```

This will cause your directory list box to look at those directories that are on the drive, you drive list box points too.

Then in your directory list box change event place this code:

```
Sub dir1_Change()
```

```
File1.path =dir1.path
```

```
End sub
```

Now your file directory list box will only look at files in the same directory that your directory list box is pointing too.

Now how do we make pictures appear? And what type of pictures do we want to look at? Well that's the next step. At the top of your form place 4 command buttons. One will have the caption "Bitmaps", the second will have the caption "Meta Files", The third "Gifs", and the fourth "Icons". Name each command button appropriately (cmdbmp, cmdwmf, cmdgif, and cmdicon.)

Now in the click event of each we will place some code:

```
Cmdbmp_click()
```

```
File1.pattern = "*.bmp" 'when you click this button the file list box will show only  
'maps  
end sub
```

```
cmdwmf_click()
```

```
file1.pattern = "*.wmf"  
  
end sub
```

```
cmdgif_click()
```

```
file1.pattern = "*.gif"  
  
end sub
```

```
cmdicon_click()
```

```
file1.pattern = "*.ico"  
  
end sub
```

Note: You should also place a default file pattern in your form load event:

```
Form1_load()
```

```
File1.pattern = "*.bmp"  
  
End sub
```

Now for the slightly complex part (Don't Panic I said slightly!!). We need to put the real meat of this little program into the file list boxes. Click event:

```
File1_click()
```

```
Dim graphic
```

```
If right(file1.path,1) <> “\” then  
    Graphic = file1.path & “\” & file1.filename  
Else  
    Picturgraphic= file1.path & file1.filename  
End if
```

```
Picture1.picture = loadpicture(graphic)
```

```
End sub
```

Basically, this code just says to point at the file that the file list box is currently on and place that picture file in the picture box. But you have now built a simple program with virtually no writing of code!

Some Notes:

Always start by creating a new folder to put all your project files in. This folder should contain all the files you need to do this project. That means any forms (*.frm), Code Modules (*.bas), Class Modules (*.cls), Project Files (*.vbp), or databases (*.mdb)

Conclusion:

Visual Basic is referred to as an “Event Driven” programming language because all the code is triggered by specific events that the user performs. This puts the user firmly in control of how the program flows and therefore increases user satisfaction with the program.

The key portions of a Visual Basic project are the forms, controls, and code modules. A form or control has properties that you can alter. The program’s code is contained primarily in events.

Additional Practice Exercises

Example 2-3

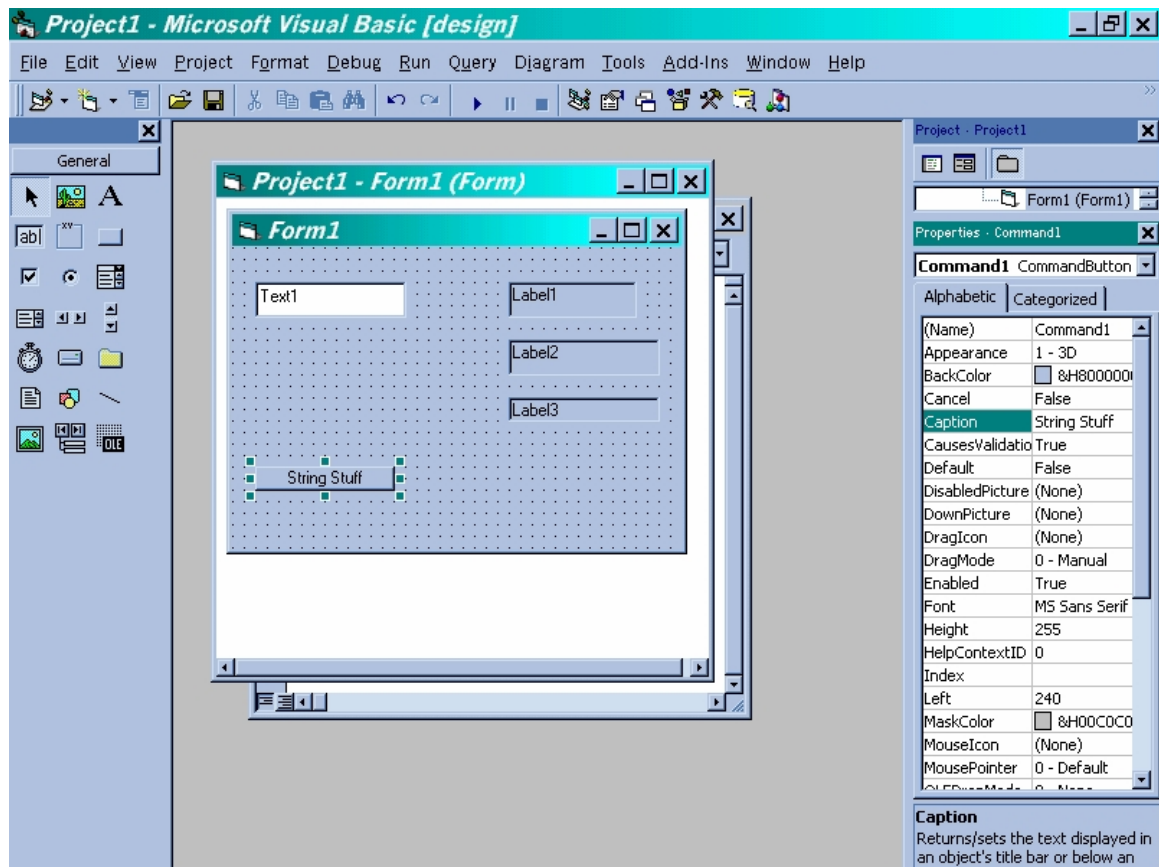
Step 1: Start a new project with a standard executable.

Step 3 place 1 text box and 4 labels on the form. Set the label captions to blank and their border styles to 1.

Step 4: place 1 command button on the form.

Step 5: Change your forms caption to read “String Stuff”

Now your form should look like this:



In the click event of the command button place the following code:

```
Dim mystring as string
```

```
Mystring = text1.text
```

```
Label1.caption = ucase(mystring) ‘ display the upper case version of the string
```

```
Label2.caption = lcase(mystring) ‘ display the lower case version of the string
```

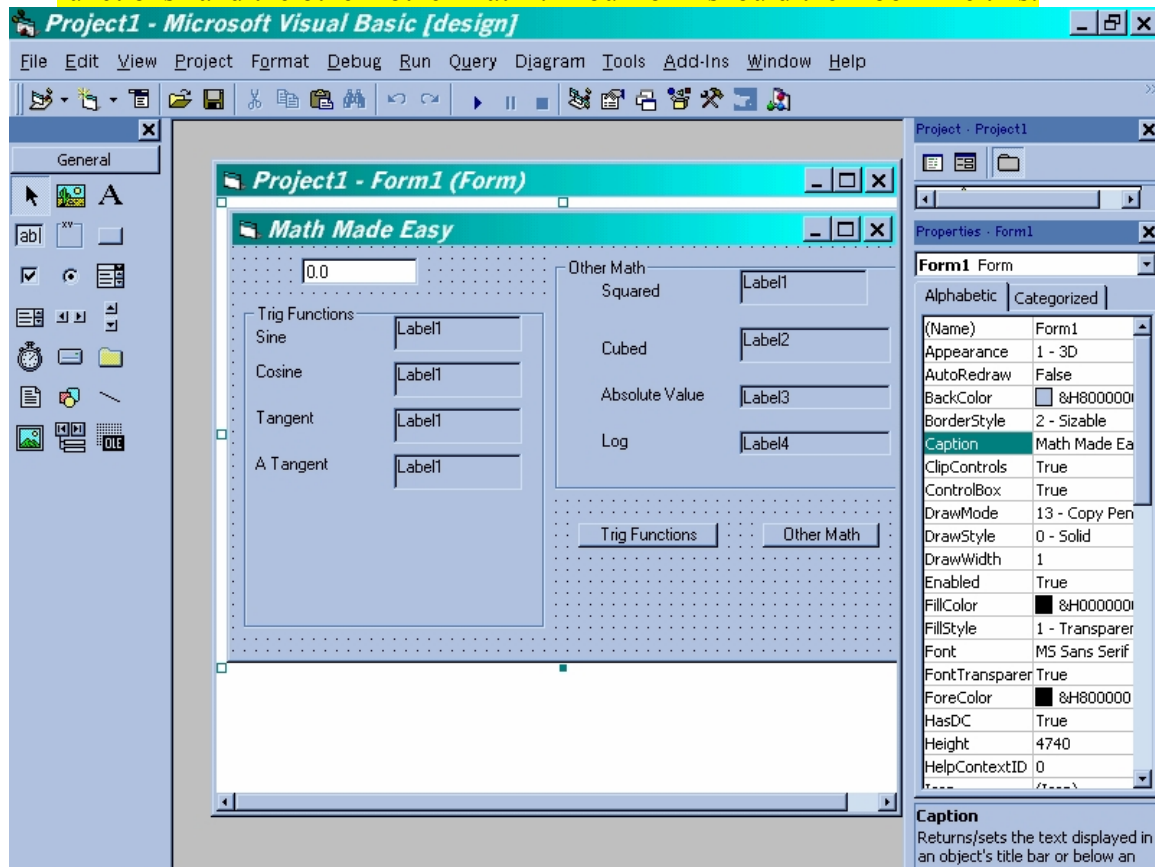
```
Label3.caption = len(mystring) ‘ display the length of the string
```


Now run the application.

This exercise gives you more practice placing controls on forms and using control properties. It also introduces you to some of the many functions built in to Visual Basic.

Exercise 1-3

1. Again we need a new executable with a single form. This form's caption should read "Math Made easy"
2. Place 1 textbox on the form and two frames. The first frame's caption will be "Trig Functions", the second "Other Math functions". The first frame will have 6 labels and the second frame will have 4. Those labels will be given a border. Then some borderless labels will be placed by them with captions. Your form should look like this:
3. You will place two command buttons on the form. One with the caption "trig functions" and the other "other math". Your form should then look like this:



Then in the command button with the caption 'trig functions' you will place this code:

```
Dim mynumber As Double  
mynumber = Val(Text1.Text)
```

```
Label6.Caption = Sin(mynumber)
Label7.Caption = Cos(mynumber)
Label8.Caption = Tan(mynumber)
Label9.Caption = Atn(mynumber)
```

And in the other command button you will place this code:

```
Dim mynumber As Single
mynumber = Val(Text1.Text)
```

```
Label1.Caption = Sqr(mynumber)
Label2.Caption = mynumber * mynumber * mynumber
Label3.Caption = Abs(mynumber)
Label4.Caption = Log(mynumber)
```

Now run your program, place a number in your text box and press your buttons. This exercise is designed to give you even more practice with Visual Basic controls, and also to introduce you to some of Visual Basic's built in math functions.

Review Questions:

1. What is a property?
2. Name 3 properties of the text box
3. What is an event?
4. What is the default event of the command button
5. Name 4 controls
6. Give an example of code that changes another controls property.
7. What is the mouse move event?
8. What does GUI mean?
9. What does IDE mean?
10. How do you add controls to your tool box?
11. What is a bug?
12. What is the default property of a text box?
13. What is the drop down menu option to add new controls to your toolbox?
14. Give the functions of at least 2 buttons on the tool bar.
15. What is the difference between “Save” and “Save as”
16. How do you remove a form from a project?
17. What is the shortcut key to get to the properties window?
18. What is the shortcut key to make the program run?

Chapter Project # 1:

Make a standard executable(.exe) with 1 form. Place a label on the form. On that form place a command button with the caption “Press Me”. In the click event of that command button place code that will change the labels property to “Ahh that’s good”. The save the project under the name firstapp.vbp and run it.

Chapter Project #2:

Create the picture viewer outlined earlier in this chapter. It should allow the user to view at least 4 file times, but you could have more.

Chapter Project #3:

Create a program with a single form that allows a person to type text into a text box, then click on a command button and have that text display in a label.

Chapter Two

Database Programming

Chapter Objectives

- **Understand and Use the Data Control**
- **Understand and Write Basic SQL command**
- **Understand the basic's of the Data Access Object**
- **Understand and utilize a variety of data access methods**

Chapter 2: Introduction to Database Programming

Most business programming is, at some level, database programming. What is an employee control program but a database program? What is a Point of Sale or Inventory Control program but a database program? It is assumed that you can create a table in Access or in the Data Manager. If this is the case then you can skip the “Introduction to databases section”. However if this is not the case I do provide you a quick introduction to database concepts and to Microsoft Access.

Introduction to Databases: This section is designed to give you a very brief introduction to creating tables in Access. Access is one of the most popular relational database systems on the market today (at least for small to medium sized databases). First of all let me define some database terms:

Field: This is a particular piece of data like: Last Name, Address, Phone, etc. In SQL Server or Oracle this is referred to as a ‘Column’.

Record: This is all the fields for a particular entry. For example, all the fields that make up a single employee’s file are a record.

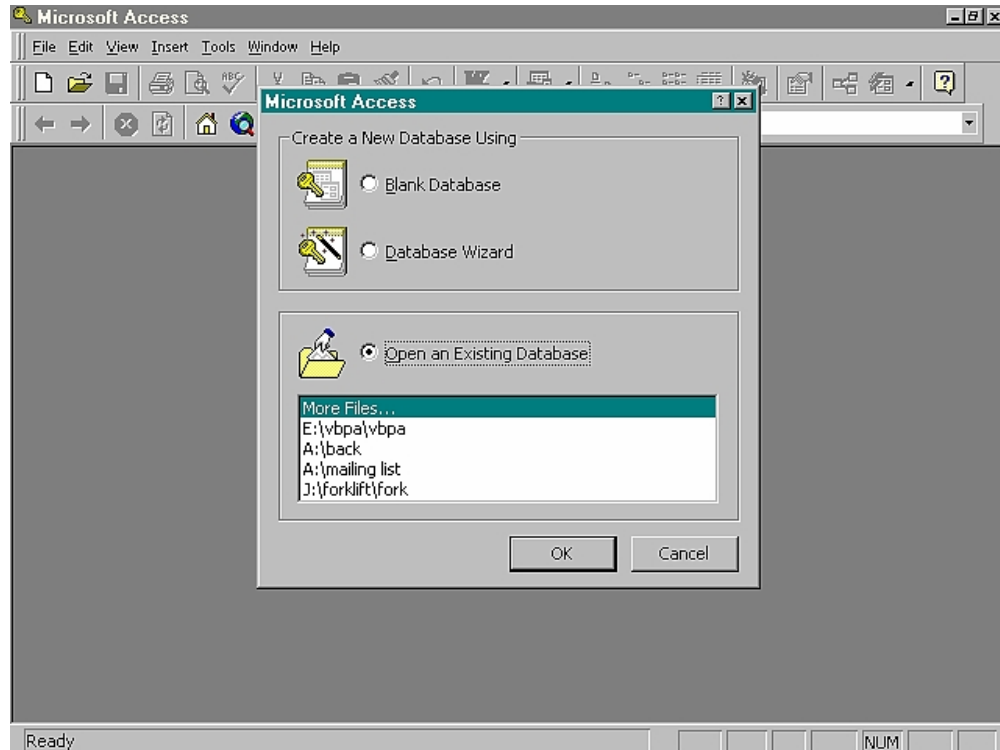
Table: These are related records stored together. For example, all the records of all the employees might be stored in an “Employee” table.

Database: These are related tables stored together. For example the employee table, inventory table, and finance table might be stored together in a common small business database.

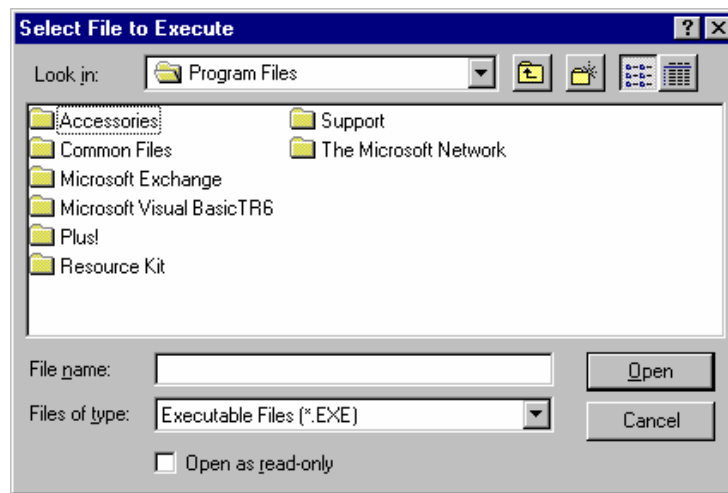
The Relational Database System (RDMS) refers to data that is related. Lets say you have a database that included employee data. Now for each employee you would want a job description. What if you have 20 people with the same job description (20 Visual Basic programmers, for example)? Do you really want to re-write the job description 20 times? In the relational database model you simply have a table of job descriptions and each employees record in the employee table is related or linked to their job description. This drastically reduced the amount of overhead a database requires. It also makes for a very logical lay out to your database.

This method of storing data is different than the previously used flat file. A flat file simply stores information one record after another. There is no relationship between records or between fields. This makes for a inordinately large data file and one that has a lot of duplication. It is also quite difficult to perform searches.

Creating a Database/Table: To do this in Access you first open Access. You will be confronted with a dialogue box. Like the one you see below:



For our purposes, choose the first option “Blank Database”. (NOTE: You could choose the database wizard to have access create an entire database for you!) You will then be asked where to create that database and what to name it. This is a dialogue box you will see:



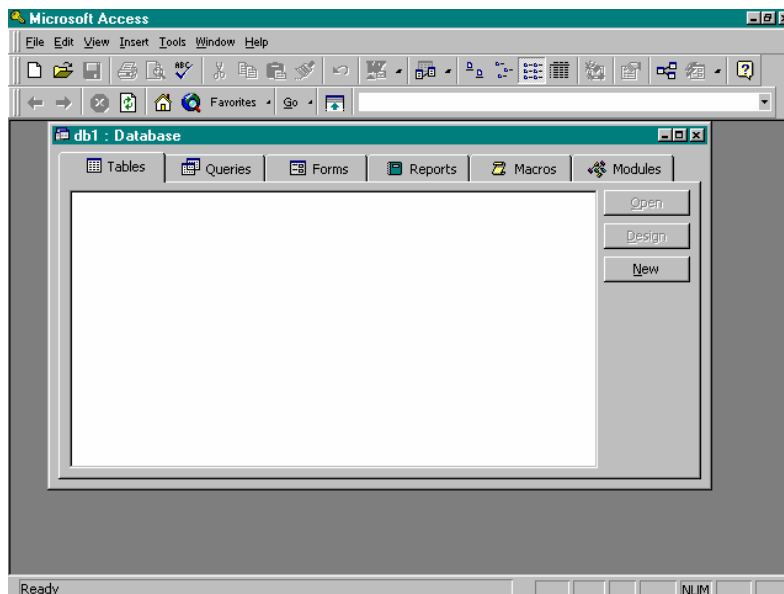
Dialog Box Basics

In case you are not familiar with dialogue boxes (You need to be since all Windows programs use them!!) here are some tips:

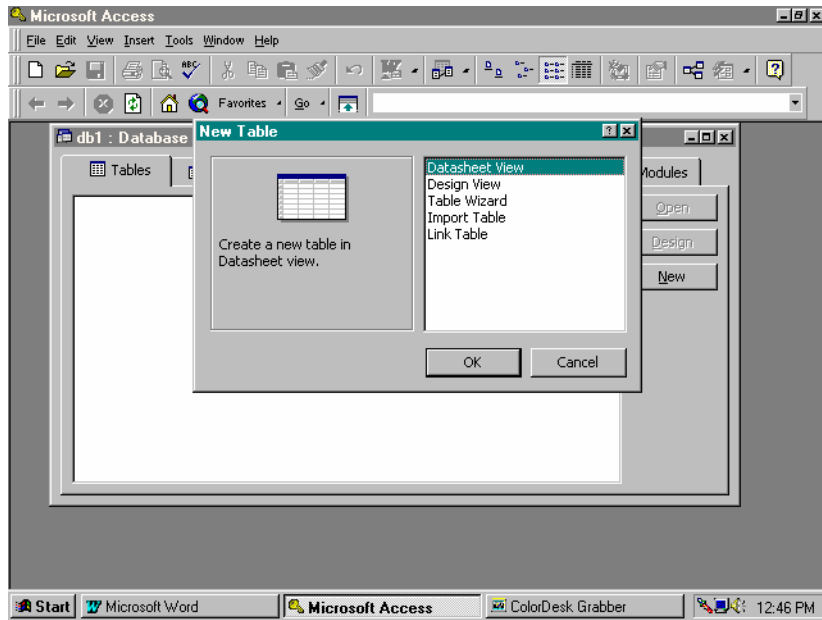
- *The top drop down box that says “look in: “ allows you to change directories and drives searching for files.*
- *The small folder button to the right of that drop down box will allow you to go up one level in the directory structure.*
- *The large white box in the middle of the dialogue box will display files and folders that are in the location you are viewing.*
- *The small box towards the bottom that says “File Name” is where you place the name of your file.*
- *The small box below that that says “File Type” is where you can select a variety of file types.*

This dialogue box is the same for opening files or saving them. The only difference will be the two buttons on the right and the caption at the top. If you are saving files the two buttons will be “Save “ and “Cancel” and the caption will be “Save As...”. If you are attempting to open files the two buttons will be “Open” and “Cancel” and the caption will be “Open..” .

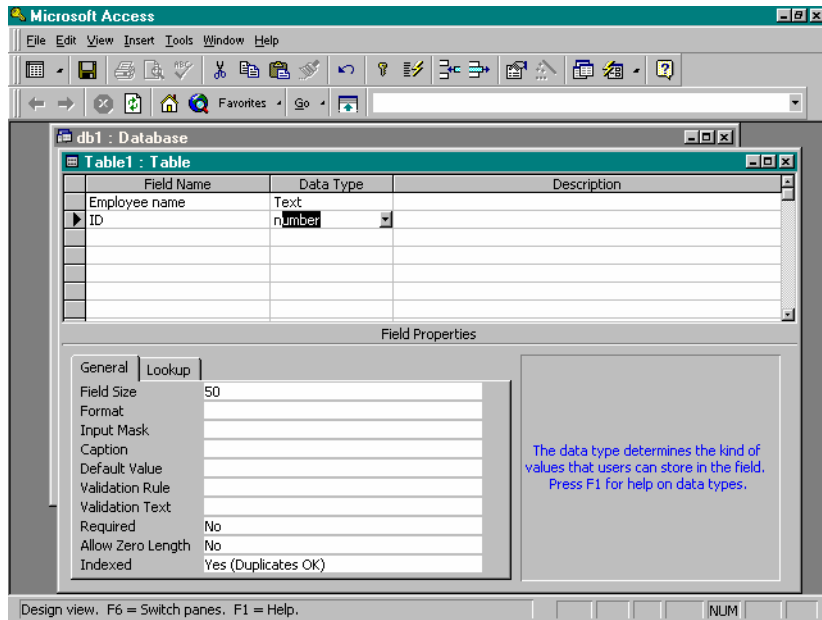
To create a new table select “New” under the tables tab



and then choose “design view”.



You are now looking at a grid for designing a database table.



Put in the names of the fields you want to include and what data types you want the fields to hold. When you have added in all the fields you need, you can close Access. Note each field needs a name and data type. The description is optional, but is a good idea.

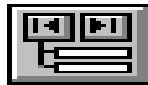
Now access will allow you to do a number of neat things with database design. Unfortunately a thorough coverage of Access is not within the scope of this book. However if you intend to truly maximize your Visual Basic programming skills I strongly recommend that you become at least moderately familiar with Microsoft Access.

It is also important to remember that Access 97 has a great many “wizards” that automate the process of developing tables, queries, or even entire databases.

Practice

Create an table in MS Access. The table should be called “contacts” and should have fields for : Last Name, First name, Street Address, City, State, Zip Code, Phone Number, and Email.

Section I: The Data Control

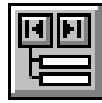


The Data control is a very special control. Using its properties you can very quickly and easily access a database. Basically after you set its properties, the data control then represents a set of records (commonly called a recordset) in that database and you can proceed to perform any database operation you wish. For our purposes, we will use access databases as examples. At the end of this chapter you should be able to create a simple program that will connect to an Access database and allow you to add, delete, and save records to that database.

Most Business programming will involve databases. For a visual basic programmer this usually means a database designed using Microsoft Access or the Data manager that accompanies Visual Basic. The Data manager is essentially the Access engine without some of the neat extras.

Now, assuming you have made a database table(s) you wish to access from your Visual Basic program you have two routes you can use: the data control, or the recordset object. Lets start with the former:

The data control is one of the controls in your toolbox. It looks like this:



When you place it on a form, you must set the following properties:

Database Name : This tells VB what database you wish to connect to. It will also give the path to that database

Record source: This tells VB what table in the database you wish to connect to.

Recordset Type: Do you want to open this as a snapshot (The user can view only) as a dynaset (basically a picture of the database at the time you load your form, but not showing subsequent changes) or a table (real time data base view)

Exclusive: This tells VB that if it is a shared database, say on a LAN, can others view it??

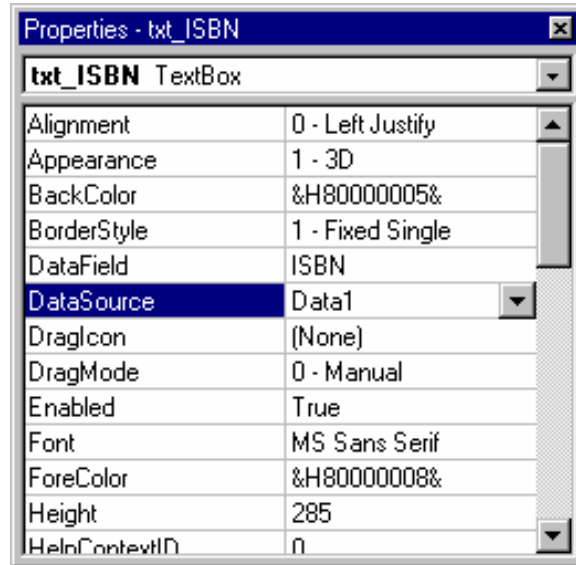


Tip: You usually won't want to have a data control exclusive. This means that no one else can open the database. Imagine the headaches you will cause if you write a program that locks everyone on the network out of a database!

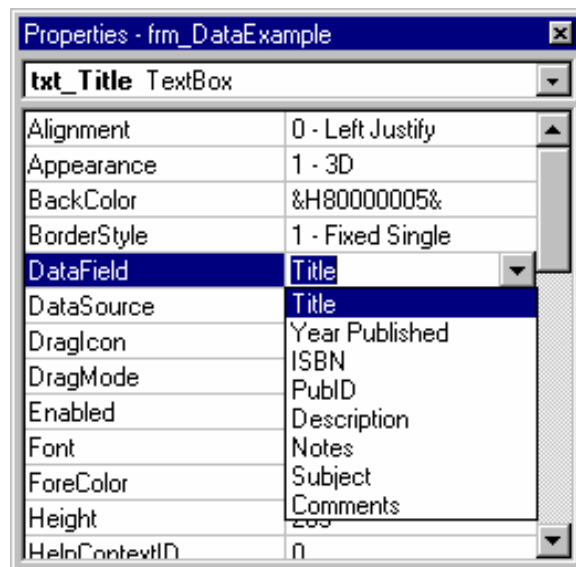
There are of course many other properties that may be of interest to you , but are not essential to our project. Some of these such as font and back color are merely cosmetic. The visible property can be quite nice if you wish to hide the data control from the user.

Now once you have done that you will want to allow the user to view data, edit data, and save new data. The quickest way to do this is to make some data bound controls. A data bound control is a control that is tied to a database, so that changes to it are automatically made to the database and visa versa. The most common example is a data bound text box. But you can bind labels, picture boxes, grids, and combo boxes to a database.

Let's say our database is an employee table. We then can make some text boxes tied to the database. One text box for each field in our table. Set that text boxes data source property equal to the name of the data control (i.e. data1, data2)



Then set its data field property to the field you wish that textbox to be tied to.



Now with that and the data control we can view the database, move first, move previous, move next, and move last. But hey, we want more than that!! Well here is a list of some things you can do:

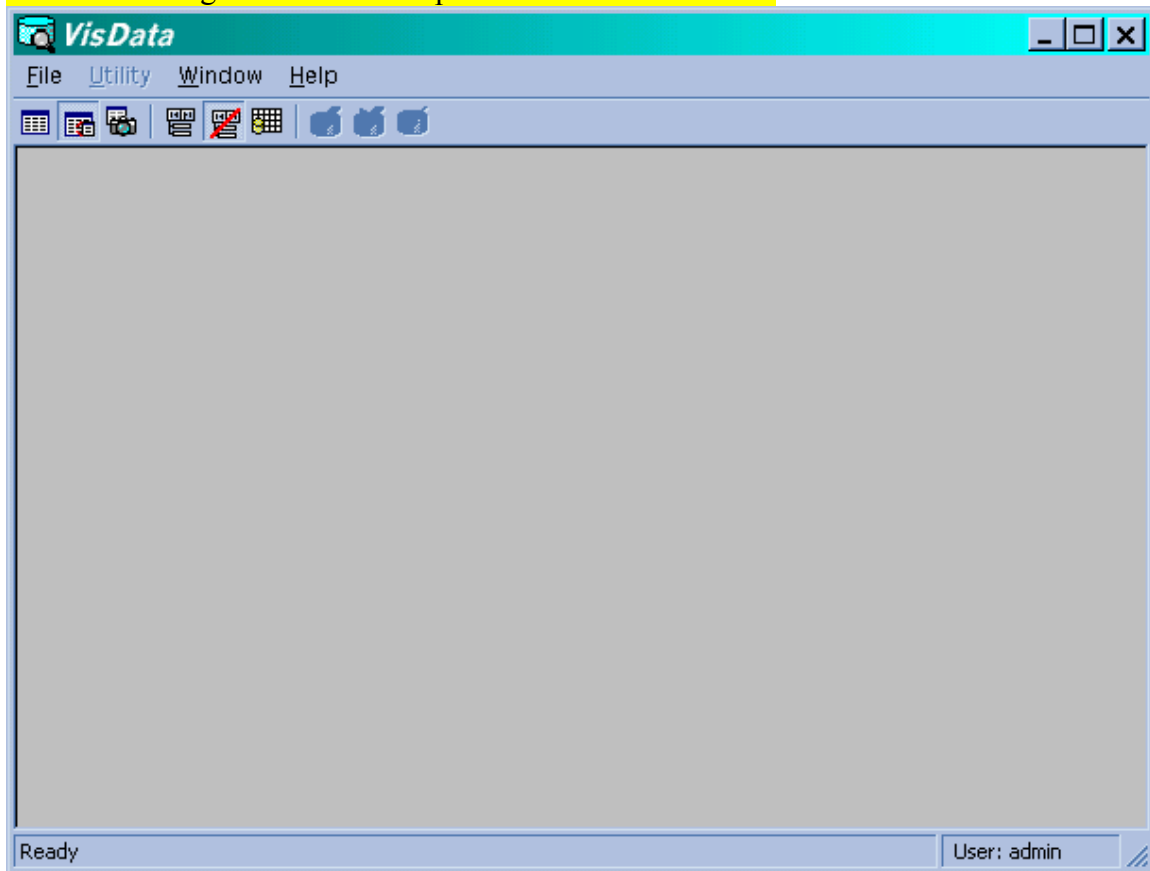


Tip: If nothing appears in your data field window, you probably forgot to set the record source property of the data control.

Example 2-1

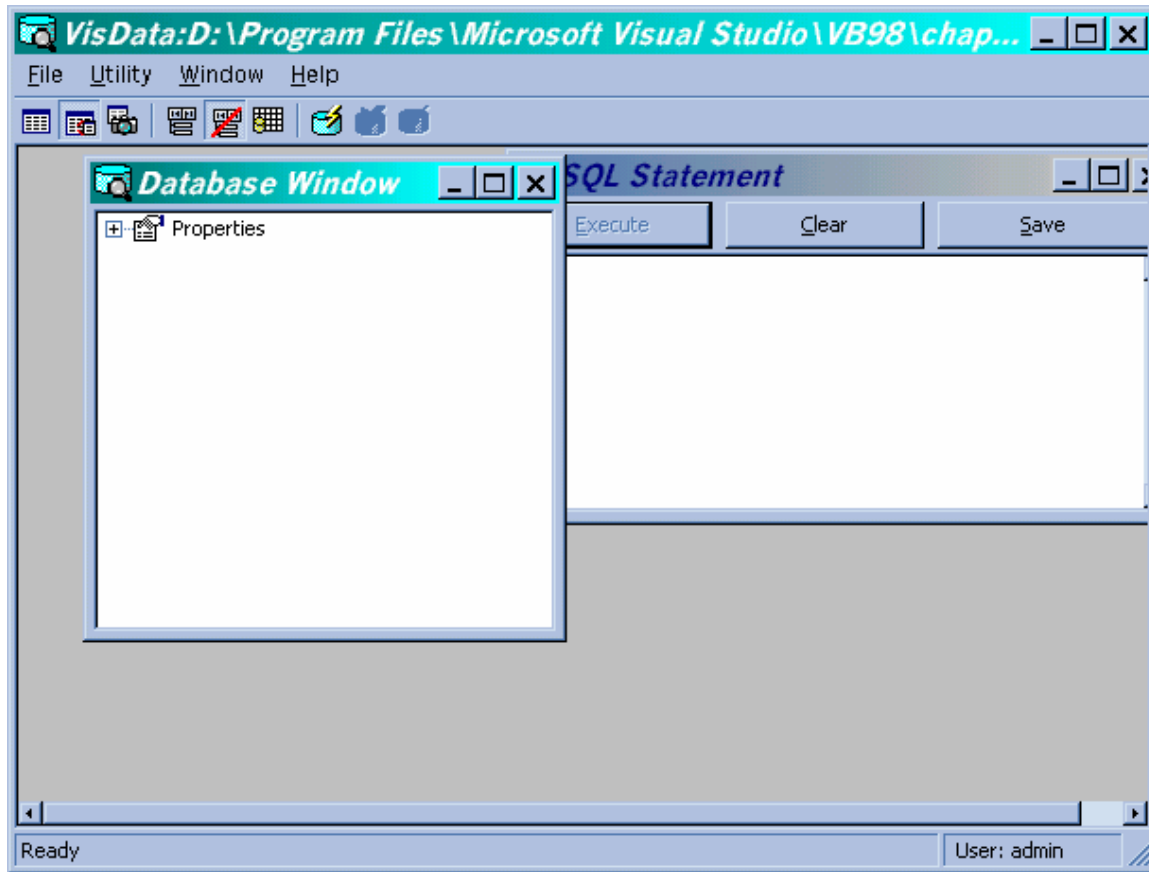
Open Visual Basic to a new project. You should have a single form on the screen. Place on that form 1 data control that you will link to a recordset in a sample that we will create with the DataManager add in. Follow along with each step:

Step1: Launch the data manager by going to the drop down manager , 'choosing add-ins' and 'data manager'. You will be presented with this screen:

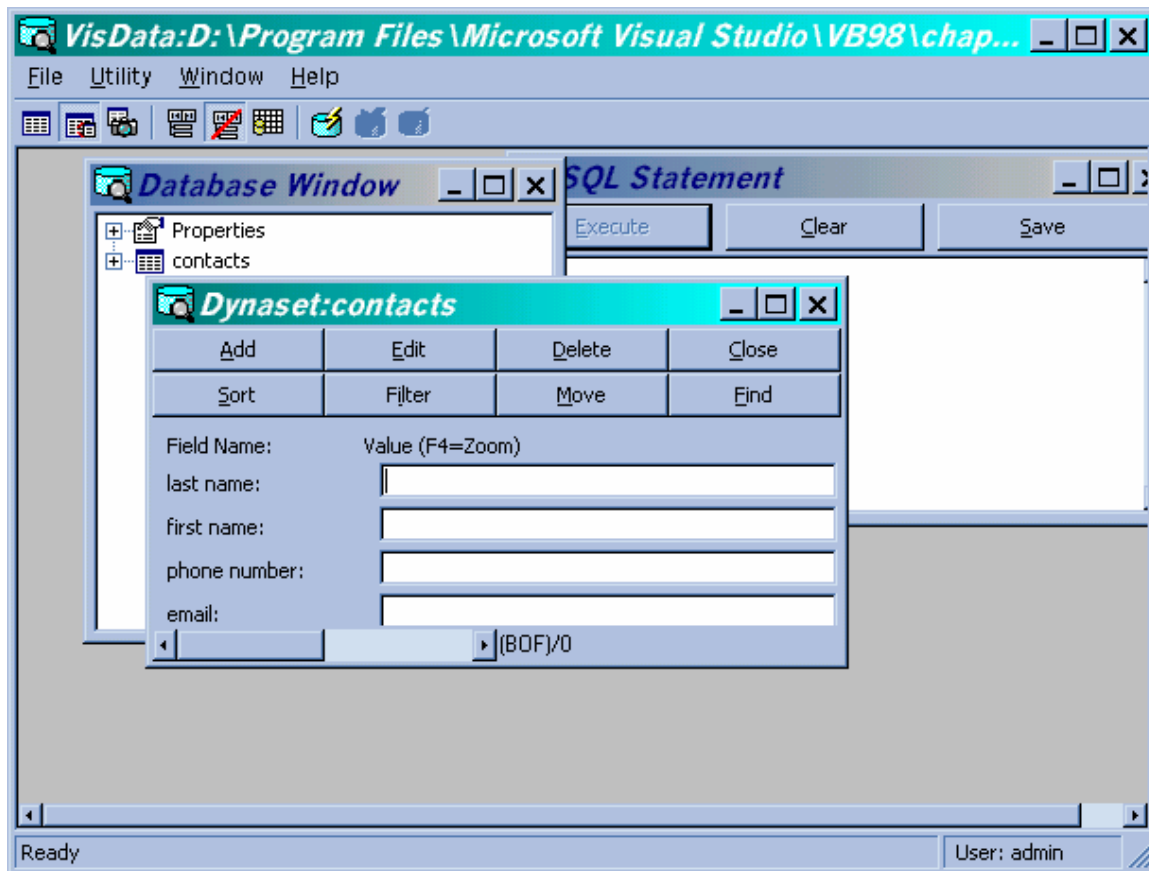


2. Choose 'File' > 'New' > 'Microsoft Access' > 'Version 7.0 MDB' This will create an Access 97 style database for us to work with. At this stage you will be presented with a dialog box asking you to name your database and save it. Lets give the name 'chapter2'.

Now you should see the following screen



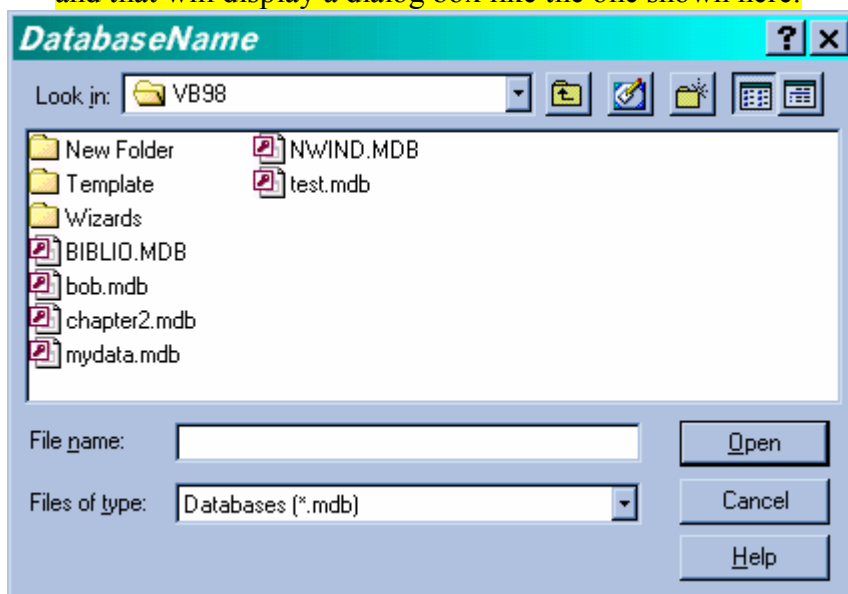
4. To add a table we right click on properties and choose 'new table' This will take us to the following screen:
5. Name the table 'contacts' then using the 'Add Field' button add 4 fields. One each for last name, first name, phone number, email.
6. Now close the fields window and click "build the table' on the main window. Now you are done. You have created a database with one table. Congratulations. Now lets add 1 sample record.
7. Right click on the table name you just created and choose 'open' . That will take you to the following screen:



8. Now click the add button and place data in each field, then click update.

Now our database is complete. Lets connect to it! First close out the data manager.

9. Back on our form place a data control . The click on its database name property and that will display a dialog box like the one shown here:



10. Just double click on the database we want to connect to (chapter2.mdb).
11. Now go and click on the record set property. A drop down box will show all the tables in the database. Since we just have one table in our database choose it. Now you have connected your data control to the database table!
12. Since we have four fields in our database table, put four text boxes on the form. With each one of them set their 'data source' field to data1 (When you click on it, all the data controls on your form will display). Then click on the 'data field' property and set that equal to the particular field you wish to connect that text box to. All the text boxes will connect to the same data source (the data control we made) but to different fields in the database.
13. Now run it and see what you have!.
You can now scroll back and forth and view data in a real database.
Well now all we have to do is allow you to add, delete, and update.

14. Make a command button (give it a caption of 'Add New') and in its click event place the following code:

```
data1.recordset.addnew
```

When you do this your data bound text boxes will blank out and they are ready for a new entry! Basically you have just created a new blank record set in the database and are ready to put data in it.

15. Make another command button (its caption should be 'Save') with the following code:

```
data1.recordset.update
```

When you do this your changes are saved!



Tip: If you attempt to use this update method without first doing an add new or edit method you will get an error.

16. Make another command button(make its caption 'delete'), this time with:

```
data1.recordset.delete  
data1.recordset.movenext
```




Tip: You must have the move next line. If you do not do this then your data control will be pointing to a null record.

17. The following code prepares a record to be edited:

```
Data1.recordset.edit
```

After making changes, you then must save them. However when using data bound controls all records are automatically in edit mode and any changes are made automatically as soon as you move to another record.

Once you have done all this, you have just created your first database application! Congratulations you are now a database programmer.

Bookmark: You may wish to bookmark a specific record so that you can return to it. You do that by setting some variable equal to the bookmark property:

```
Let mvariable = data1.recordset.bookmark.
```

Other properties:

You can get even more information from the data control and exercise more control by looking at its properties. For example if you wish to know how many records are in a database table that your data control is tied to then you could have:

```
Label1.caption = data1.recordset.recordcount
```

The record count of that database table would then be displayed in the label.

You may also have some need to set the exclusive property to true, this would keep any one else from accessing that table while you have that property set.

MovePrevious Moves to the previous record in the Recordset.

MoveFirst Moves to the first record in the Recordset.

MovePrevious Moves to the previous record in the Recordset.

MoveLast Moves to the last record in the Recordset.

MoveNext Moves to the next record in the Recordset .

FindFirst This will find the first record that matches your requirements. For example:

```
Let mysearch = "LastName = " & "Smith"
```

```
Data1.recordset.findfirst mysearch
```

FindLast This works just like find first only it finds the last matching record

```
Let mysearch = "LastName = " & "Smith"
```

```
Data1.recordset.findlast mysearch
```

FindNext: This is often used in conjunction with a findfirst command.

```
Let mysearch = "LastName = " & "Smith"
```

```
Data1.recordset.findlast mysearch
```

You can use any of these commands in the following manner such as:

```
Data1.recordset.movefirst or
```

```
Data1.recordset.movenext
```

```
Data1.recordset.moveprevious
```

```
Data1.recordset.movefirst
```

Now you probably noticed that your data control has some arrows on it. The arrows on the extreme right and extreme left are move last and move first buttons. The arrows on the inner right and inner left are move next and move previous buttons. So if you choose to leave the data control visible you will not need to code in any of the move functions, they are built into the data control!

BookMarking:

The bookmark denotes a specific place in the database. You can dimension a variable to hold a bookmark value and then you can return to it. Usually a variant is used to hold the bookmark value.

```
Private vReturnPoint as Variant
```

```
vReturnPoint = Data1.Recordset.Bookmark
```

Even if new rows are added to, or existing rows are deleted from, the Recordset, the Bookmark Property of a record doesn't change. To return to the same record after moving away from it, use this command:

```
Data1.Recordset.Bookmark = vReturnPoint
```

You can also access a specific field in a record with the following code:

```
Data1.recordset.fields("fieldname")
```



Tip: You must absolutely spell the fieldname correctly to include any spaces.

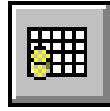
This line of code now references a specific field in the record that your data control is currently pointing to.

How do I?	Answer
Add a Record	Data1.recordset.addnew
Delete a record	Data1.recrodset.delete Data1.recordset.movenext
Save changes	Data1.recordset.update
Move to the next record	Data1.recordset.movenext
Edit a record	Data1.recordset.edit
Address a certain field in a record	Data1.recordset.fields("fieldname")
Bookmark a record	Let myvariable = Data1.recordset.bookmark
Get the record count in a table	Data1.recordset.count

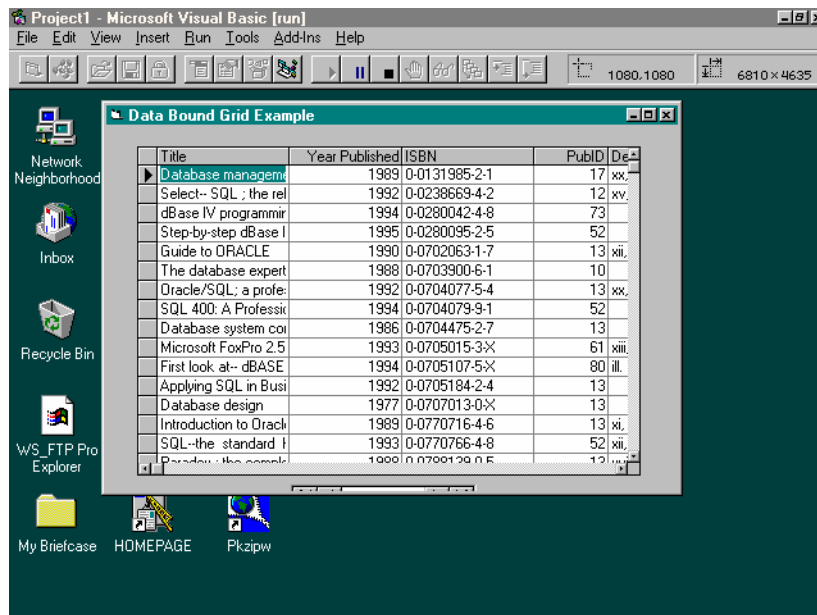
More Data bound Controls:

A data bound control is a very quick and efficient way to connect your program to a database. Visual Basic has offered several controls that can easily be bound to a database. To get these controls you must first add them to your toolbox. You can do this by choosing "project" from the drop down menu and then selecting "components". You will then check the "Microsoft Data Bound Grid" and "Microsoft Data Bound List" boxes and click the "OK" button. You will now see three new controls in your tool box and we will talk about each one.

Databound Grid: This is a grid that is tied to your entire table. It will display the whole table not just one record. It is on your toolbar and looks like:



Like any other control, it has properties. You will of course want to set its name and caption. But there are other properties you will be concerned with. I will list them in order of importance though on the properties window they appear in alphabetical order. When you run your program the data bound grid will display your data in a nice grid format:



Data source: This tells the data bound grid which data control you wish to get data from.

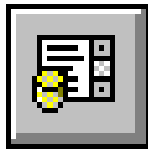
Data Mode: The grid can operate in bound mode or in unbound mode. You may wish to have it bound while it loads up data then unbound while the user makes changes.

Allow Add New: This determines whether or not the user can add new records to the database via this grid.

Allow Delete: This determines whether or not the user can delete records via this grid.

Allow Update: This determines whether or not the user can update the database via this grid.

Data Bound List Box: This works just like the normal list box, except that you don't have to add items to it, they are taken from the database. It looks like this:



The properties you are most concerned with on this control are:

Data Source: This determines what data control the data will come from.

Data Field: The data bound list box is bound to a specific field, just like a data bound text box.

Data Bound Combo Box: This works like a normal combo box. It is bound to a specific field. It looks like this:



The properties you are most concerned with on this control are:

Data Source: This determines what data control the data will come from.

Data Field: The data bound list box is bound to a specific field, just like a data bound text box.

Style: This is the same as a normal combo box and simply denotes what type of combo box you want.

With these additional controls you can expand any database program. I think you will find them very use

Section II: SQL

Now for the big news: SQL or Structured Query Language. This is basically the language of Databases. You can use visual basic to write SQL statements. Let me give you an example:

```
Dim sSQL as string 'used to hold the sql statement
```

```
sSQL = "SELECT * FROM [MYTABLE]"
```

```
Let Data1.Recordsource = sSQL
```

```
Data1.recordset.refresh
```

This statement will select all files from the table.

What we did here is take a string variable and set it equal to an SQL (pronounced S-Q-L or sequel) statement. We then set the data controls recordsource property equal to that SQL statement. The brackets are used to denote a table within the database. The asterick (*) symbol is computer shorthand for “everything”. So basically we told the data control to give us all the files in the table named “mytable”. The capitilization is not absolutely necessary but is an often used convention.

Before we go any further you should realize that Structured Query Language is NOT a part of Visual Basic, that’s why we have to put SQL statements in a string. It is the language used by relational databases. Most relational databases utilize Structured Query Language, not just MS Access. It is a good idea to become very comfortable with SQL since it is used by MS SQL Server, Oracle, MS Access, SYBASE SQL Anywhere, and many other relational database systems.

Now more often you will have a more complex sql statement with clauses like
"...WHERE [state] =California"

The Basic format is

```
"SELECT [field1],[field2],.... FROM [sometable] WHERE [somefield] = SomeCriteria"
```

The Where clause denotes some condition. Basically you are saying that you want all the specified fields if a certain criteria is met.

Basically for SQL you will have to declare a string variable (most people just call it strSQL) and then you will assign SQL statements to that variable. You can then assign the SQL statements value to whatever data access object you are using, be it a data control or a recordset object. Here is an example

SELECT: strSQL = "SELECT Publishers.[Company Name], Publishers.State FROM Publishers WHERE Publishers.State ="

```
strSQL = strSQL & Chr(34) & txtVariable & Chr(34)
```

Note: The chr(34) is inserting quotes into your string. You have to do it this way because if you try to literally place a quote into a string, the quote will terminate the string!

```
Data1.RecordSource = strSQL  
Data1.Refresh
```

Please note several things about this statement:

1. You can have lines added to each other by just making the next line of SQL: SQL = SQL + "...."
2. Notice how you can add in a variable contained within a text box. This is vital as that is how you will most often do searches.
3. Note that once you have built your SQL statement you merely have to set an object equal to it. You can set recordset objects or data controls equal to an SQL statement.

You can also build a variable into your sql statement like this:

```
"SELECT * FROM [Employees] WHERE [State] = " & sState & ""
```

Now the above statement would let you select the company names of publishers from a database based upon the state the company is in (i.e. you can select all the publishing companies in Texas). Also notice that we used two different examples of including a variable into an SQL statement. In the first example we used ascii character codes (Chr(34)) to place in the proper quotation marks, and in the second example we used a combination of single quotes ', double quotes ", and the ampersand & symbol to accomplish the same goal. This is just one more reminder that there is often more than one way to accomplish a desired task.

ORDERBY: This statement simply orders the records based on a particular field you wish to order by. Here is an example:

```
strSQL = strSQL & "ORDER BY[Publishers.State]"
```

This statement says to sort the records returned by our SQL query by the state of the publisher. I frequently use this statement simply to arrange data in a particular manner. For example I might add this statement to an employee database:

```
strSQL = strSQL & "ORDER BY [Employee ID]"
```

This way all the records are now in order by Employee ID Number. I use this statement a lot since it is the quickest and easiest way to sort records. In some cases I will create a drop-down menu or command buttons that each sort by a different criteria. Using the "ORDER BY" statement in this way, I let my users sort their data in any fashion they wish.

INNER JOIN: What if you want to search more than one table for the publishers records? Well have no fear SQL offers the INNER JOIN Statement:

```
SELECT Publishers.[company Name] FROM Publishers  
INNER JOIN Titles ON Publishers.PubID = Titles.PubID
```

This statement just says to get all the company names of publishers from either the Publishers table or the Titles table if the publishers ID's match!

DISTINCT: What about Duplicate records? Especially in the above examples the same record may be duplicated many times. One way around this is to simply add the DISTINCT statement to your SELECT such as:

```
strSQL = "SELECT DISTINCT..."
```

Now we have just covered the SELECT, INNER JOIN, and DISTINCT statements. While there is more to SQL this will give you a start. You will probably use the select statement on a regular basis. Consider the order by statement as a good way to sort the records in your data control in any way you wish.



Tip: Always double check the spelling of your SQL statements. A single misspelling and it will NOT work.

A Brief History of SQL

SQL or Structured Query Language is the language used to talk to Relational Databases. The Relational Database model for database design was invented by Dr. E.F. Cobb in 1969 and published in Computer World in 1985.

SQL was first implemented in 1974 at IBM the San Jose Research Laboratory. SQL is essentially a non-procedural language that uses English-like words to talk to a database. The American National Standards Institute (ANSI) is continually updating their versions of SQL standards.

This is just a brief working introduction to Structured Query Language. It should get you up and running so that you can use SQL in your programs. I strongly suggest that early on in your programming career you make a point of learning more about SQL. You can find some good resources for SQL and other programming topics at the end of this book.

Section III: Using Data Access Objects

Another Way to connect to a database is via data objects. These are specific object variables that you can use to connect with databases. Basically you use object variables to represent the data table instead of using the data control. Before you can use the DAO object library you must go to 'project' on the drop down menu and select 'references', then choose the DAO library. If you have more than one version, just choose the latest version.

The connection is quite simple:

```
Dim myDB as Database  
Dim rs as recordset
```

You now have a variable that can point to any database (myDB) and one that can point to any type of recordset (rs). Getting them to point to the right things is pretty easy:

```
Set mbDB = OpenDatabase("c:\pathname\mydata.mdb")
```

```
And then set rs = dbopenrecordset("tablename",dbopentable)
```

The last parameter simply states that you wish to use this recordset as a table instead of as a snapshot or dynaset. Opening it as a table means that all changes made either by you in your program, or by someone else that is actually in the database are reflected immediately in "real time". Opening as a dynaset means that you basically have a picture of the database as it looked at the time you connected. Any changes made to

that underlying data after that will NOT be reflected in your program. A snap shot is simply a picture of the database (like the dynaset) that is “read only”. That means your users can look, but they can’t touch. You can also use

`DbOpenDynaset` and `dbopenSnapshot`

You can also include some options when opening the actual database. You can set it to exclusive (only one user) or read only. You simply add a comma and the word true in the appropriate place:

Generic Example: `Set myDB = opendatabase(“c:\pathname\mydata.mdb”,exclusive, readonly)`

Specific Example: `Set mydb = opendatabase(“c:\vb\biblio.mdb”,false, true)`

Now once you have a recordset object you can use it just as you would the data control:

`Rs.addnew`

`Rs.delete`
`Rs.movenext`

`Rs.update`

You can even use SQL statements like:

`Set rs = sSQL`

Just as you would with a data control.

However there are differences between the recordset object and the data control. The recordset object can NOT be bound to controls. You have to manually place the data in the controls from the record set object. However the recordset object gives you complete control of the database programing.

Manually placing data into the controls is, however, simple:

`TxtLastName = rs.fields(“Last Name”)`

Also getting data from the fields to the database is simple:

`Rs.addnew`

```
Rs.fields("Last Name") = txtLastName
```

Remember that the recordset (rs) and database (db) variables are object variables. When you are done with them, make very sure that you close them and destroy them:

```
Rs.close
```

```
Set rs = nothing
```

The recordset object is actually used much more frequently than the data control because you have total control over all programming functions. The data control is usually used when you want to bind controls to your data, like a databound grid.

Example 2-2

Step 1: Start a new executable.

Step 2: Go to 'Project' > 'References' and select Microsoft DAO library.

Step 3: In the general declarations section place these variable declarations

```
Dim db as database
```

```
Dim rs as recordset
```

Step 4: In the form load event place this code

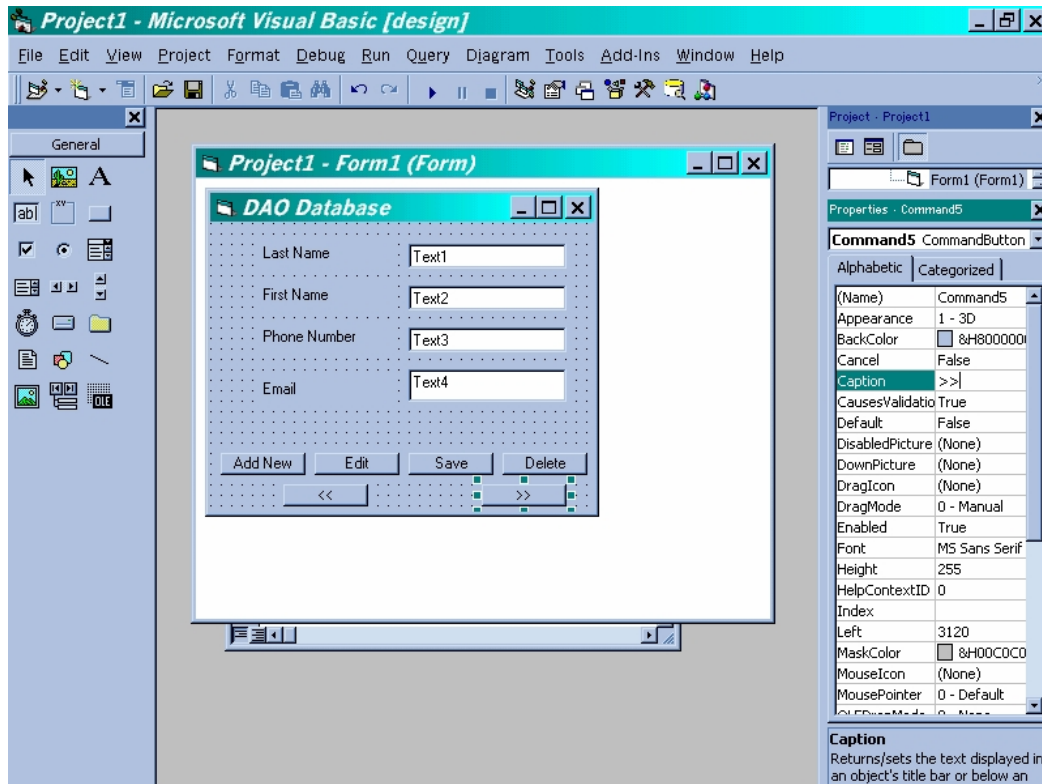
```
Set db = OpenDatabase("databasepath")
```

' Substitute the path and name of the database you created in exercise 2-1.

```
Set rs = db.OpenRecordset("tablename",dbOpenTable)
```

' Substitute the table name you created in exercise 2-1

Step 5: Place 5 text boxes on your form with 5 labels. Also put 6 command buttons
At the bottom of the form. Your form should look like this:



Step 6: Now we add code to each of our buttons

Add new:

`Rs.addnew`

Delete:

`Rs.delete`

`Rs.movenext`

Save:

`Get_text()`

`Rs.update`

Edit:

`Rs.edit`

`<<`

`rs.moveprevious`

`>>`

`rs.movenext`

Now we will need to have two functions . One to load the textboxes with data from the recordset and the other to take data from the textboxes and put it in the recordset. You should right these in the general declarations section of your form.

`Public sub fill_text()`

`Text1.text = rs.fields(0)`

```
Text2.text = rs.fields(1)
Text3.text = rs.fields(2)
Text4.text = rs.fields(3)
End sub
```

```
Public sub get_Text()
Rs.fields(0) = text1.text
Rs.fields(1) = text2.text
Rs.fields(2) = text3.text
Rs.fields(3) = text4.text
```

```
End sub
```

The difference between using the data access objects and the data control is simply that the data control allows you to fill controls automatically, but by doing that it limits the level of control you can programmatically exert over the database operations.

Conclusions:

This of course is not an exhaustive look at data access objects but perhaps it will give you enough information to get you started on this topic. In a later chapter we will examine the Remote Data Access Object and the Active Data Object.

For this chapter I merely wanted to introduce you to the idea of connecting your forms and controls to a data source via either the data control or the Data Access Objects.

Review Questions:

1. What property of the data control do you set to select a database to connect to?
2. What is the code to delete a record?
3. What property of the data control do you use to select a specific table in a database?
4. What is the code to save changes to a record?
5. Why is there a “move next” command in the delete?
6. What code will tell me how many records are in the data control?
7. How would you write a line of code that specifically recognized the employeeID field of a record?
8. What properties of a text box must you set in order to bind it to a data control.
9. What property of a text box must you set to pick a specific field to bind to?
10. What is the code to move to the first record in a table?
11. How would I connect an object variable to a database?
12. What SQL statement is used to connect data from two different tables?
13. What SQL statement is used to get records from a table?
14. What SQL statement is used to filter out duplicate records?
15. What SQL statement is used to place records in a particular order?

Chapter Project:

Create a form that connects to two different data sources via the Data Access Object (hint: if you really want to be efficient you can use the same objects to connect to both and just re point them.).

Chapter Three

The Essentials of

Writing Code

Chapter Objectives

- Understand and use loop structures
- Understand and use naming conventions
- Understand the declaration and scope of variables
- Understand and use error handling and data validation
- Understand basic programming conventions

Chapter 3: Essentials of Writing Code:

So far this book has concentrated on how to make interesting things happen with visual basic. Now it is time to introduce you to some standard coding methods. These methods are present in most programming languages only their implementation is different. Using these techniques, you will be able to begin to do more serious programming. Also I feel the need to interject a point here. Visual Basic is an incredible programming tool. With it you can create professional quality Windows applications without having an in-depth knowledge of computer science, operating systems, etc. However, this is also the flaw with Visual Basic. It allows you to do some pretty nasty looking programming. I implore all of you to work hard to not settle for programs that just run, but strive for high quality programs that can also be easily read and followed by other programmers.

Comments:

Let me begin by introducing you to comments. A comment is a line that the computer will ignore but that another programmer viewing your source code can read. Comments are vital to good programming. They allow people reading your code to get an idea for what you meant. You can put a comment anywhere by simply placing an apostrophe before the line such as:

‘ This is a comment

Commented lines will appear in Green. Actual key words that Visual Basic recognizes will show up in Blue and errors will be in red.

NOTE: *It is absolutely vital that you use comments in your code. Comments can explain to other programmers what you intended to accomplish. As a rule there is no such thing as too much comments in your code. Here are some examples:*

Dim x as integer ‘this x is simply used as a loop counter

Let iAcct = left(iAccountNum,4) ‘get the first 4 digits of iAccount Num and put them in the variable “iAcct”

From here on I will use comments in my code examples so that you can become acquainted with commenting.

If-Then loop.

If-Then loops are a method by which your program can make decisions based on either the conditions existing, or on user input. Let me first give you a generic example then I will use a specific example:

```
If some condition exists then  
    Do this code  
End If
```

A specific example would be:

```
If text1.text = "Howdy" then  
  
    text1.text = "Hi back at ya"  
  
End if
```

Basically this code looks at the text in the text1 control and checks its value. If that value is "Howdy" then it changes the text to say "Hi back at you". If not then it simply proceeds with the rest of the program

Another example would be

```
If myage > 21 then  
    Text1.text = "come on in"  
Else  
    Text1.text = "Sorry, you are not old enough to enter"  
End if
```

The concept is simple and is found in all programming languages. If some condition is true then execute the specified code.

Lets try placing a text box and a command button on a form. In the command buttons, click event, place the above if-then loop. Then run the program. First click the button. Nothing should happen. Then type the word "Howdy" in the text box and then click the button. You should see the text box change to "Hi back at you".

Of course you may have need for more options in your if-then loop. That is where the Else and Elseif statements come in. You could rewrite the above loop like this:

```

If text1.text = "Howdy" then
    Let text1.text = "Hi back at ya"
Elseif text1.text = "Greetings"
    Let text1.text = "Good Day "
Else
    Let text1.text = "Bye Now"
End if

```

You can continue to add many else-if statements if you so desire but remember that too many will make your program harder to read.

Do Loops (Do-Until and Do-While)

We also have do loops. The two most common are the Do-While loop and the Do-until loop. For example if you wished to scroll through a table referenced by a data control looking for a specific entry, you could use a do-Until loop:

```

Data1.recordset.movefirst           'move to the beginning of the document
Do until data1.recordset.eof        'eof stands for end of file
    If data1.recordset.fields("name") = "smith" then
        Exit do                     'get out of the do until loop
    End if
    Data1.recordset.movenext        'move to next record

```

Loop

This code will loop through all the records until it finds one with a name "smith" at which point it will exit the loop. Loops are very important in programming, you will find that you often utilize them.

NOTE: You notice that I left spaces between some lines and that I indented lines inside loops. This is not necessary in order for your program to run. However it is absolutely

essential if you want other people to read your code. If you don't use blank lines (called white space) and indents you will not last long on any programming team.



Tip: The most common mistake is to leave out the `data1.recordset.movenext` line of code. Without that line the loop will continually loop around the first record and will never move forward.

You could re-write this as a Do-While loop:

```
Data1.recordset.movefirst           'move to the beginning of the document
Do while data1.recordset.eof = false 'eof stands for end of file
    If data1.recordset.fields("name") = "smith" then
        Exit do                       'get out of the do until loop
    End if
    Data1.recordset.movenext         'move to next record
Loop
```

Either the Do-while or the Do-until loop can be used interchangeably. However, I suggest that you consider what is most logical. In the second example, we are moving through a database to the end. It makes sense to use a do-until loop.

Here is an example from an actual application where the do-while loop combined with some keywords is used to search a data control.

```
Public Sub item_search()
Dim search As String

search = InputBox("Please enter the Item you wish to search for:")

frminven.data1.Recordset.MoveFirst

Do Until frminven.data1.Recordset.EOF
```

```
If Trim(UCase(frminven.data1.Recordset.Fields("Item"))) = Trim(UCase(search)) Then
```

```
    Exit Sub
```

```
End If
```

```
frminven.data1.Recordset.MoveNext
```

Loop

Basically this example moves to the beginning of a data control on a specific form and then loops through that data control looking to see if some item matches the item that the user input with the input box. The “Ucase” statement basically causes the search comparison to look at the upper case values of both items, in this way it is not case sensitive. In other words, “Wrench” will be the same as “WrEnCH”. The Trim statement allows our search to ignore any spaces the user may have inadvertently left at the end of their input. It means to trim off the blanks at the beginning or the end.

With – End With loops

These loops are very useful when you wish to do a lot of code with a single object. Fore example if you wanted to write code that dealt with a record set object called rs then you might try

```
With rs
```

```
    .delete  
    .movenext
```

```
end with
```

Select Case

When you are face with multiple choices, an if-elseif-end if scheme can get very convoluted. For that purpose we use the Select Case Statement

Assume you have some variable called tempvalue and it is an integer. Depending on the value of that integer we wish to take some action.

```
Select case tempvalue
```

```
    Case 1  
        ‘ Place code here  
    Case 2  
        ‘ Place code here  
    Case 3  
        ‘ Place code here
```

Case 4

‘ Place code here

End select

If tempvalue is 1 then the first case will be executed and the others ignored. If, however, it is 3, then case 3 will be executed and the others ignored. You can have a virtually unlimited number of cases, and you can make switches base on strings as well as numeric values.

Variables & Datatypes:

A variable is basically a space in memory that is set aside to hold some value. We use variables frequently to temporarily hold information. For example if you wanted to compute sales tax on a purchase you might have 1 variable that holds the pre tax total and on that holds the tax rate. When you multiply them together, you can set the result equal to another variable.

A variable must be of a certain Data Type. Each data type has a different purpose and each takes up a different amount of memory and has a different range.

Variable Type	Storage Size	Range
Integer	2 bytes	-32,768 to 32,767
Long	4 bytes	-2,147,483,648 to 2,147,483,647
Single	4 bytes	-3.402823 E38 to 1.402823 E38
Double	8 bytes	-1.7976931348 E308 to 1.7976931348 E308
Currency	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5808
String	1 byte per Character	0 to 65,000 Characters
Byte	1 byte	0 to 255
Boolean	2 bytes	TRUE or FALSE
Date	8 bytes	Jan 1, 100 to Dec 31 9999
Object	4 bytes	Any Object reference
Variant	16 bytes + 1 byte per character	Anything, this is the catch all.

Integers and long integers are used for whole numbers. Singles and doubles are used for numbers containing a decimal place. The currency will take only dollar amounts. Strings

can hold letters or numbers as well as any other character. The byte only holds small numbers but works great as counter for loops.

The boolean simply holds true and false values, it is often used as a flag (i.e. is some condition true or false). The date holds only date values within the specified range. The Variant is a catch all and is NOT recommended as it wastes memory.

The object variable is a very special variable. It can be used to represent any object even forms and controls. You could for example have:

```
Private f as object
```

```
Set f = mydataform
```

This would make f a reference or pointer to the form called mydataform.

It is not important that you memorize the exact ranges of all variables. It is, however, vital that you remember what each variable is used for.

You create a variable by allocating space for it, declaring its scope, and giving it a name and type. This is much easier than it sounds. Below is an example:

```
Private iAccount as integer
```

This tells the computer that the variable needed will be private in scope, iAccount will be its name, and it will be an integer type.



Tip: Some programming languages allow you to dimension several variables on one line like this example from C that dimensions 4 integers:

```
Int accountnum, indexnum, loopcounter, j
```

In Visual Basic this will not work. Only the first variable will be an integer, all the rest will be variants. In Visual Basic you must use this method:

```
Private accountnum as integer, indexnum as integer, loopcounter as integer, j as integer
```

Scope refers to usage. Any variable declared in an event should be private. Any variable declared in the declarations section of the form module should also be private. However, any variable declared in a code module that you may wish to use in more than one form should be declared public. In earlier editions of Visual Basic you used the terms Dim and Global instead of Private and Public. You will still see these used by many programmers and they do work, Microsoft discourages their use. The private just means that only that level and below can use that variable. For example a privately declared variable in an

event can only be accessed within that event. A privately declared variable in a form can be accessed by events within that form.

NOTE: You cannot declare a variable as public within an event, sub routine, or function.

It is customary to give a variable a descriptive name beginning with a letter designating the type of variable. For example

IAccount is an integer variable holding the account number

SName is a string variable holding the name

You can use the underscore character if you so wish like this:

IAccount_number

As you have probably already seen there are also naming conventions for the variables. Here are the most common ones:

Variable	Naming Convention
Integer	IName
Long	LName
Variant	VName
Double	DName
Object	objName

User Defined Types

This is one of the coolest features in Visual Basic. It is very much like structures are in C/C++ . Basically it is a conglomerate of variables that you define. For example in our rolodex program you might want to have each entry in a single user defined variable. You would declare it like this:

```
User_Data as type
  Name as string
  Phone as string
  Email as string
  Address as string
End type
```


Now at this point you have simply created a new data type within your application. In order to use it you must set some variable equal to it:

```
Public Employee_Data as user_data
```

Then to address one of these items you simply use:

```
Txtemail= Employee_Data.email
```

The Message Box:

The message box is your friend. With it, you can give the user lots of feedback and you can get responses. Basically to use the message box function you must first declare an integer variable, this variable will hold the response code from your message box. It is a good idea to name it something like:

```
Private imessage as integer
```

Or

```
Private iResponse as integer
```

Now you declare the message box function in the following manner:

```
Imessage = msgbox("your message", buttons,"Your title")
```

This basically says make a message box using the built in msgbox function. Give it whatever message you decide, give it the buttons you tell it, then give it your title. Whatever button the user presses will return a code to the variable imessage. Now for the really great part: you don't have to memorize any numeric codes. In version 4.0, Visual Basic began implementing names for you to use. For example if you want only an ok button then in the above code place vbokonly where I have the word buttons. If you want a Yes button and a No button then place vbyesno where I have the word buttons. If you want a complete listing look under help for msgbox and you will get a complete list. The most common however are: VBOkOnly, VBYesNo, and VBOKCancel.

Now when the user clicks a button it will return a code to your variable "imessage" that will indicate what button was pushed. The error handling section of this chapter makes extensive use of the message box, but here is an example you can put in an exit button to see if the user really wants to exit. Place this code in the click event of a command button with the caption "Exit":

Private iMessage as integer

IMessage = msgbox("Are you sure you want to quit?", vbyesno, "Exiting")

If I message = vbyes then 'if the use types in the word end then stop the program

 End

End if

The basic structure of a message box function call is this:

 Return value variable = msgbox(prompt for user, buttons, title, help file(if it exists))

The message box may well be the most commonly used function in Visual Basic. I certainly could not imagine writing programs without it. You can dig further into the message box function in the help file. I strongly suggest you do some experimenting with it and get comfortable with it.

Input Box

Much like the message box except that it prompts the user to input data. For example:

Dim s_name as string

S_name = inputbox("Please Enter Your Name")

You can expand the call using more variables. The three most commonly used parts of the input box are:

(prompt, Title, Default Value)

Error Handling and Data Validation

This could be the most important thing you learn about visual basic. This section is devoted to giving you a working basic knowledge of these topics. You can certainly do more than I will mention here, but this should get you started.

Error Handling:

For the purpose of learning, let us consider a command button named cmdSave. This command button uses a data control update method to update a database. Now the problem is that if there is any error at all (such as the database is locked by another user)

you will get an error and your program will crash. The simplest type of error trapping we can do is:

```
CmdSave_Click()  
  
    On error Resume Next  
  
    data1.recordset.update  
  
end sub
```

Now what this will do is simply skip any line that has an error. Your program won't crash, however the user won't know that the data was not updated! So how about we notify the user there was a problem:

```
CmdSave_Click()  
  
    on error goto errorhandler  
  
    data1.recordset.update  
  
    exit sub  
  
errorhandler:  
  
    imessage = msgbox("I could not update the database!",vbcritical,"my program has an  
    message")  
  
end sub
```

Now this code tells the program that if there is an error skip down to our error handler. If there is no error, then exit the sub after the data was updated. Then the error handler will use the message box function to display a message to the user. Now your program does not crash and the user knows the data was not updated!! Only one problem, when they call you for support you won't have a clue WHY the data did not update!

```
CmdSave_Click()  
  
    on error toto errorhandler  
  
    data1.recordset.update
```

```
exit sub
```

```
errorhandler:
```

```
imessage = msgbox ("I could not update the database because : " & error,vbcritical, "My  
program has a message!")
```

Now this code gives an error message for the user that includes the actual error!! Now your application does not crash, the user knows what happened, and you get a specific message for technical support!!!! Congratulations you are now using error trapping!



Tip: Notice in every example there is an “Exit Sub” Prior to the error handling routine. If you omit this, then the error handler will be triggered NO MATTER WHAT!!

You may also prefer to place errors in a log file so that you can look at that log file to see past errors and find program bugs. In a later chapter we explore reading and writing from a flat file. You can use that methodology in your error handler to create an error log.

Data Validation: Another way to keep from getting errors is to make sure you don't send invalid data. The simplest method is to make sure that text boxes max length is set to the max length of the data field. In that way, you will not send data that is too big.

The next method is to look at the data in the lost focus event. Lets say you have a program that has the user input a number in txtNum1 and then divides that number by whatever the user enters in txtNum2. Now division by zero gives an error, so in the lost focus event on the txtNum2 we put

```
if txtNum2.text = "0" then
```

```
    imessage = msgbox("You cannot divide by zero, please enter another number")
```

```
end if.
```

Now the user will be instructed NOT to enter Zero! And you will get valid data! You can also check for numerical values, improper dates, etc in this same manner. I prefer checking right in the lost focus event, but some programmers wait until just before the data is sent to the database and place a series of if-then loops checking all the data. Either method is acceptable

Now remember these are just a few things to get you started. I hope you will continue to explore these topics since error handling and data validation will give you more robust applications!! (Not to mention less support calls!)

It is also a good idea to verify actions before doing them. You can use message boxes and their return values to verify before deleting, saving, or exiting. For example:

```
iReturn = msgbox("Are you sure you want to delete the current record?",vbYesno,"Confirm Deletion)
```

```
If iReturn = vbYes then  
    Data1.recordset.delete  
End if
```

Key Words:

Key words are programming terms that Visual Basic recognizes as a command. Below are listed several of these that you will need, along with an example of each:

Len: If you want to get the length of some variable or text you can use the len command to get that value. Here are two examples

```
Let iLength = len(txtname) 'this will take the length of the text in txtname  
                          ' and place it in the variable iLength
```

```
Let iLength = len(iAccount) 'This will take the length of the variable iAccount  
                            ' and place it in the variable iLength
```

Let: This assigns a value to a control or variable. Here are a few examples:

```
Let iCounter = 0          'this sets the variable iCounter to zero
```

```
Let iAccount = txtAccount 'This takes the value of txtAccount and copies it to  
                          'iAccount.
```

Load: This will load a form. If you have a main form and you wish to load other forms you can use this command. For example if you have a form called frmOrders and you have a command button you wish to use to load it in, place the following code in that command buttons click event:

```
Load frmOrders
```

Hide: This makes whatever you are referring to invisible. Here is an example:

```
FrmOrders.hide
```

The form is still loaded its just not visible

Show: This shows something that is hidden. If you use it in conjunction with a form that is not loaded then it will first load the form then show it.

```
FrmOrders.show
```

Ucase: What if you wish to compare to values but you don't care if their case matches (lower or upper case)? You can compare the upper case values of both. For example in the Do-Until example previously used what happens if the value you are searching for is "Smith" and the value in the database is "Smlth"? You won't get a match. If you use the Ucase statement then the upper case values of both will be compared. Here is an example:

```
If Ucase(sSearchValue) = ucase(txtLastName) then...
```

You can also compare the lower case values using the same technique with the keyword **Lcase**.

Msgbox: This key word simply tells the computer that you want to display a message box.

SetFocus: This tells the program to change focus to another control. You might place something in your cmdAddNew button that sets focus to the first text box on your form:

```
CmdAddnew_Click()
```

```
    TxtFirst.setfocus
```

```
End sub
```

Additem: This method allows you to add items to a list box or a combo box. For example if you wanted a list box that allowed a user to add a title to some persons last name you could place this code in the forms load event:

```
List1.additem "Mr. "  
List1.additem "Mrs. "  
List1.additem "Ms. "  
List1.additem "Dr. "  
List1.additem "Rev. "
```

Then in the double click event of that list box you could add this title to a text box that had the last name like this:

```
Let txtlastname.text = list1 & txtlastname.txt
```

Or you can make a separate subroutine to add the list items:



```
Private Sub AddList(objListBox As Control)

With objListBox
    .AddItem "Smith, Joe"
    .AddItem "Jones, Bob"
    .AddItem "Jackson, Rick"
    .AddItem "Doe, John"
    .AddItem "Meyers, Al"
    .AddItem "Dobson, Sid"
    .AddItem "Mitchell, Charlie"
    .AddItem "Noble, Mike"
End With 'objlistbox

End Sub
```

Removeitem: This works just like the above example only it removes an item from a list or combo box.

Clear: This key word causes all items in a list box or combo box to be erased:
List1.clear

Date: This simply returns the current system date. You might try some code like:
Let txtdate.text = Date

Time: This returns the current system time. You can display this to the user with some code like:
Let lbltime.caption = Time

Format: This allows you to change the format of any variable, textbox, or label. Let's look at some examples:

If I want the date to be yymmdd then I use the format command like this:

```
Let txtdate.text = format(date,"yymmdd")
```

You see, I place an open parentheses after the format comand then I tell it what I wish to format. In this case, that is "Date". The in quotation marks I tell it how to format it. We can also format numbers. For example if you want a number to appear as a dollar amount:

```
Let txtmynumber = format(txtmynumber,"###,###.##")
```

Your number will show a dollar sign then six digits a decimal point and two additional digits.

Here are some examples of Date formats and their outputs

<i>Format</i>	<i>Output</i>
<i>Format(Now,"m/d/yy")</i>	1/21/98
<i>Format(Now,"dddd,mmmm,dd,yy")</i>	Wednesday, January 21, 1998
<i>Format(Now,"d-mmmm")</i>	21 Jan
<i>Format(Now,"mmmm-yy")</i>	January 1998
<i>Format(Now,"hh:mm am/pm")</i>	08:45 AM
<i>Format(Now,"h:mm:ss a/p")</i>	7:15:04 a
<i>Format(Now,"d-mmmm h:mm")</i>	3-January 7:18

Shell: This is a very useful key word. By use of the shell command, you can start other programs. Look at this example:

Private lshell as long

```
Lshell = shell("c:\path\name.exe",1)
```

This tells the computer to take the path name and start the program "name.exe" in standard mode (1) and return its windows handle to the variable entitled lShell.



Tip: You must use a long in 32 bit operating systems like Windows 95, 98, or NT because they return a long windows handle. Windows 3.1 was 16 bit and returned an integer handle.

App.Activate: This statement allows you to activate a program that is already running. You can either, activate a program by using its name as it appears in the tool bar or by using the windows handle returned to lShell.

```
App.activate lshell
```

Or

```
App.activate "MS Word"
```


Sendkeys: After app.activate you can send keystrokes to a program. For example if you have activated the notepad.exe you can then

Sendkeys "My Dog Has Fleas"

And that phrase will appear in the notepad!

Left: This command allows you to get the left portion of a string. For example, if you want to get the left three characters in a text box you can use this code:

```
Private sMystring as string
```

```
Let sMystring = left(txtMytext,3)
```

This will place the left 3 characters of txtMytext into the string sMystring. So, if the text box txtMytext had the word "Hello" in it, sMystring would now have "Hel" in it.

Right: This is very similar to the left command, except that it returns the right side characters requested. For example:

```
Private sMystring as string
```

```
Let sMystring = right(txtmytext,2) would result in :
```

```
"lo"
```

I think you will find the Left and Right command very useful

Trim: This command will remove all blank spaces from a string

Ltrim: This command will remove all trailing spaces from the left side of a string.

Basic Math:

The basic mathematical operators are available in visual basic. The following is a list with examples:

Addition: You simply place the + (shift =) sign between the variables or controls you wish to add. Note that using this symbol in conjunction with strings will simply concatenate or connect them.

```
Let itotal = iSubtotal + iSubtotal2
```

Subtraction: This works by simply placing the – symbol between the variables you wish to subtract:

Let INetIncome = IGrossIncome – IDeductions

Multiplication: This is accomplished by placing the * (shift 8) between the two variables you wish to multiply:

Let ITax = IGrossPay * TaxRate

Division: You divide two variables by placing the / between them like this:

Let dQuotient = dDivident / dDivisor

Example 3-1

In this example we are going to combine many of the coding techniques you have covered in this book along with the mathematical functions you have just learned, to create a windows calculator.

Step 1 : Create a new project with a single form

Step 2: Change the forms caption to “VB Calculator”

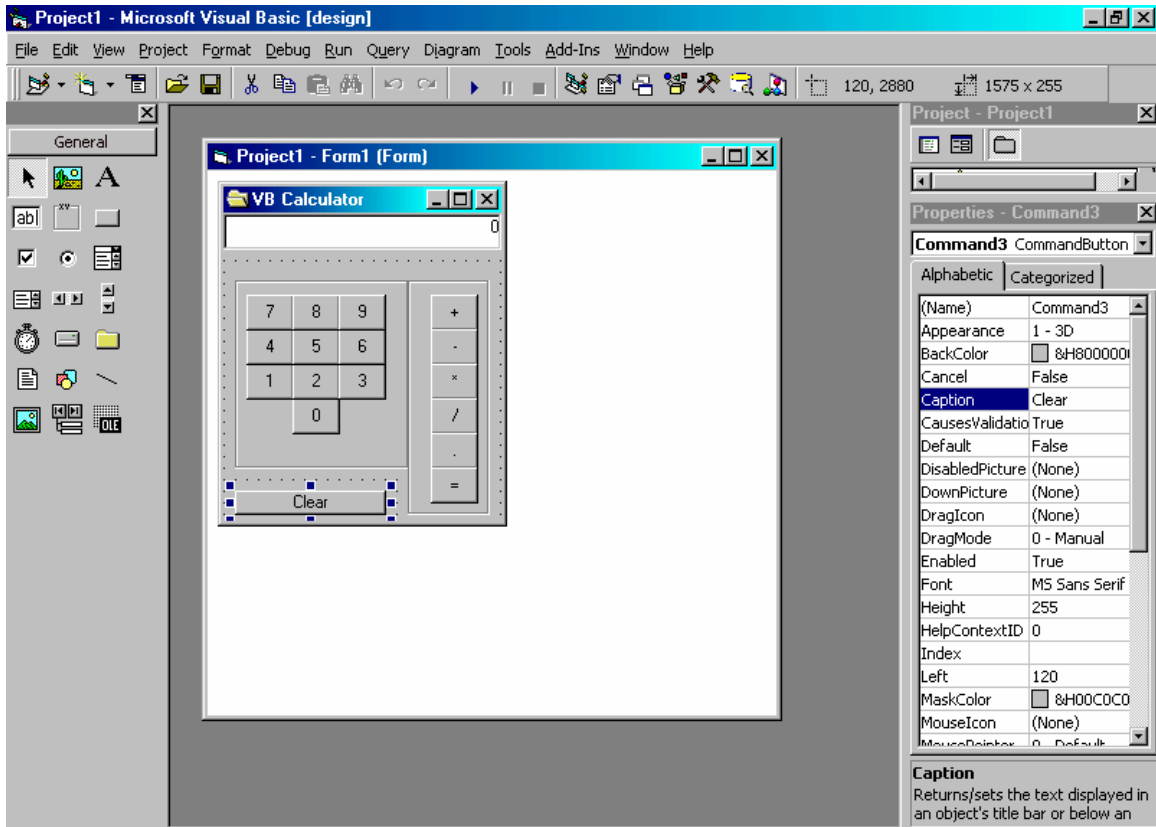
Step 3: You are going to place to control arrays of buttons on the form. Each on a separate frame. Each frame will have no caption. The way you do this is placing one button on a frame then doing a copy and paste of that button to the frame. You will be asked if you wish to create a control array, say yes. You will first create an array with 10 buttons (0 through 9). You will then, on a separate frame, create a second array of 6 buttons(+, -, *, /, ., =). Make sure that with the numeric buttons that each caption matches the buttons array index.

Step 4: Then you will place a label across the top of your form and set its caption to ‘0’. It will have a border style of ‘fixed single’, and a background color of white. Its alignment should be set to right justify. Assign it the name ‘display’.

Step 5: Now add one button under the number buttons, and make its caption ‘Clear’. In its click event place the code:

```
Display = “0”
```

When you are done your form should look like this:



Step 5: Since we made our buttons as control arrays, we will be able to place all the code for each button group in one event. First let's declare some global variables in the general declarations section of the form module.

Option Explicit

```
Dim firstnum As Single
Dim secondnum As Single
Dim answer As Double
Dim operation As Integer
```

```
Const PLUS = 0
Const MINUS = 1
Const MULTIPLY = 2
Const DIVIDE = 3
Const DECPOINT = 4
const EQUALS = 5
```

Step 6: Now in the click event of the command1 array (the button array with the numbers on it) place this code:

```
display = display & Index
```

Step 7: Now we need to add some code in the click event of the command2 array (the one with the math operators). This is where we will use those constants we created earlier. They should match the index of the button with the matching symbol (Since DIVIDE = 3, the button with the caption '/' should have an index of 3).

On Error GoTo errorhandler

Select Case Index

Case PLUS

```
firstnum = Val(display)
display = "0"
operation = PLUS
```

Case MINUS

```
firstnum = Val(display)
display = "0"
operation = MINUS
```

Case MULTIPLY

```
firstnum = Val(display)
display = "0"
operation = MULTIPLY
```

Case DIVIDE

```
firstnum = Val(display)
display = "0"
operation = DIVIDE
```

Case DECPOINT

```
display = display & "."
```

Case EQUALS

```
secondnum = Val(display)
do_math
```

End Select

Exit Sub

errorhandler:

```
MsgBox ("Whoops there was a problem" & Err.Description)
```

Step 8 Now for that subroutine 'do_math' that we referenced above. You will write this in the general declarations section of your form module.

Public Sub do_math()

On Error Resume Next

Select Case operation

Case PLUS

```
answer = firstnum + secondnum
```

```
display = answer
```

Case MINUS

```
answer = firstnum + secondnum
```

```
display = answer
```

Case MULTIPLY

```
answer = firstnum + secondnum
```

```
display = answer
```

Case DIVIDE

```
answer = firstnum + secondnum
```

```
display = answer
```

End Select

End Sub

Making a Tag Tip:

NOTE: This technique is only needed in older versions of VB since VB 6.0 has a tag tip property for most controls.

In many windows programs you will notice that as you move your mouse over a button a small (often yellow) tag will appear that will tell you what that button does. You can make one of those for your own programs. First create a label and place it at the bottom of your left most button. Set the labels background property to a color you like then set its auto size property so that it will contract or expand to fit the text it holds. Also, set its visible property to false. Now in the mouse move event of each of your buttons you can just change some properties and you tag tip will show up with an appropriate message. Below is an example:

Make a command button called cmdSave and a label named lblTagTip with its properties set as above. Then place the following code in the mouse move event.

```
CmdSave_mousemove()
```

```
    LblTagTip.caption = "This will save the new record"
```

```
    Lbltagtip.left =xxxx (place the number here that you wish)
```

```
    Lbltagtip.visible = true
```

```
End sub
```

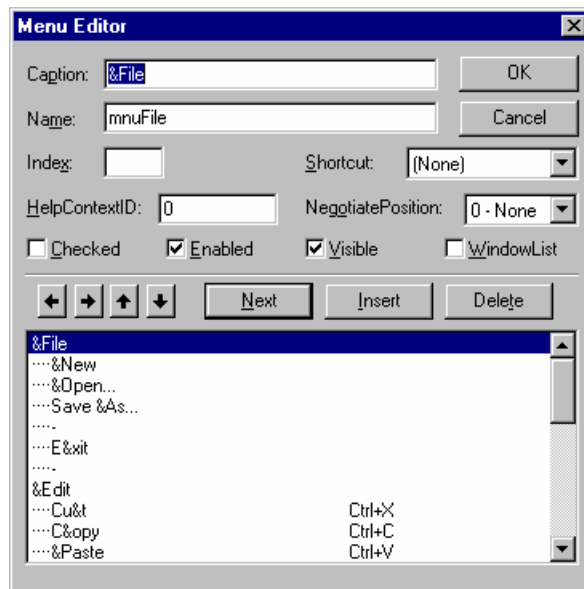
By using the labels left property to move it around you will also be able to use just one label and simply change its caption and position so it can be the tag tip for all of your buttons.

Now if you want your tag tip to go away when you move off of the button and you have grouped your buttons together in a frame you can place this code in the mouse move event of the frame:

```
FraToolBar_mousemove()  
  
    Lbltagtip.visible = false  
  
End sub
```

The Drop Down Menu: Most windows programs make use of Drop Down Menus. In this section, you will learn how to make a drop down menu.

First of all, you must select tools from the drop down menu at the top of the IDE. Then you must select “Menu Editor”. You will then be presented with the menu editing screen.



Note the items you need to be concerned with:

Caption: This will be what the user sees

Name: This is what the program will use to refer to that menu item

Shortcut: This allows you to assign shortcut keys to your menu items.

Arrow Keys: Indented items are sub menu's

This screen prompts you to give your menu option a caption and a name. As with all controls the caption is what the user sees, the name is what the program sees. The standard naming convention with menu items is "mnuName".

You will then need to use the arrow keys to place your menu item. A menu item that is flush with the left hand side of the box and is not preceded by any "... " is a top level menu. Any menu's immediately following that have a "... " and are not flush with the left hand side, are sub options for this menu item. You can also select a short cut key for your drop down menu at this point.

When you have placed all the menu items you wish you will then click the OK button and you will notice that your form now has drop down menu items at the top. Now all you need to do is to place code in each menu items. Click event just as you would with a command button.

Some Routines:

The following are a few sample routines just for your edification. Hopefully you will find some of them useful in your programming. They have been very useful for me as a professional programmer. Hopefully by examining them you will get a better idea of how to write code.

Centering a Form

```
Public Sub CenterWindow(f As Form)
```

```
'This routine will center the form calling it on the  
'screen. Just place the line " centerwindow.me" in the  
'form load event of the form you wish to center
```

```
    f.Top = (Screen.Height / 2) - (f.Height / 2)
```

```
    f.Left = (Screen.Width / 2) - (f.Width / 2)
```

```
End Sub
```

Highlighting Text

```
Public Sub SetSelected()
```

```
'This sub simply highlights/selects the text  
'in a textbox/label/etc when that control has  
'focus. You can place it in the got focus event of the text box. Or you could  
'place it in a separate subroutine then in the got focus event of all your text boxes  
'simply place the word SetSelected.
```

```
Screen.ActiveControl.SelStart = 0
```

```
Screen.ActiveControl.SelLength = Len(Screen.ActiveControl.Text)
```

```
End Sub
```

Waiting a Second

```
Public Sub SlowDown(itime As Integer)
```

```
'This subroutine will simply wait a number of seconds  
' for example you can place the code slowdown 1 in your app  
'to make it wait 1 second. It will only recognize integer values
```

```
Dim lStartTime&
```

```
lStartTime = Timer
```

```
Do While Not Timer >= lStartTime + itime
```

```
    DoEvents
```

```
Loop
```

```
End Sub
```


Preventing Multiple copies

Public Sub Main()

'use this as your main sub in order to prevent multiple copies of your
'app from running.

'if the application is already running then just bring the
'main form to the forefront , if not then load the main form

'You would place this in a separate code module. If you have more than one form
'that code module would also be a good place to put your set selected subroutine.

If App.PrevInstance Then

 BringWindowToTop frmMain.hwnd

Else

 Load frmMain

End If

End Sub

Review Questions:

1. How do you tell a message box to display a yes and a no button?
2. What line must appear in your code prior to the first line of your error handler?
3. Name 3 types of variables
4. What is a double variable used for?
5. Write an example of a Do-Until loop.
6. How do I get the length of a variable?
7. Give an example of an If-Then loop.
8. How can I prevent a user from typing more than 10 characters in a text box?
9. How do I declare a variable?
10. Under what drop down menu option do I find the menu editor?
11. What does the shell command do?
12. How do I send a keystroke to another application?

Chapter Project:

1. Go back to the Rolodex from chapter 2 and change the max length property on all text boxes so they will not allow entries that are longer than the fields in your database table.
2. Also, add in error handling in all command buttons.
3. You will need to add a new command button that will search by last name for a particular entry.
4. Use the SetSelected subroutine to cause all of your textboxes to be highlighted when they have focus. Place this code in a separate code module.
5. Place the sub main in that code module so that you can prevent multiple copies of your application from being ran.
6. Place tag tips on your buttons
7. Add in input boxes to get data from the user and message boxes to display data and confirm actions.
8. Go to “file” on the drop down menu, then choose “make exe” and make a Standard executable. Also make a short cut on your windows screen for that Executable.

Chapter Four

In Depth VB Coding

Chapter IV: In Depth Visual Basic Coding

Chapter Objectives

- Understand and use additional Controls
- Understand and use arrays
- Understand and use the Select-Case Statement
- Become familiar with the API
- Learn additional key words

By now, you should have a basic working knowledge of how to write simple Visual Basic programs, specifically database programs using the Data Control. You should also have a working understanding of variables, error handling, and basic coding standards/techniques. In this chapter, you will be introduced to some new controls and new command words. These will round out your programming skills enabling you to make even better programs.

Section I More Controls:

Check Box: The check box is a very useful control. When you place it on your form it will allow the user to select an option. For example if your form is for employee information and you are inquiring about an employees educational background, you might have check boxes for Bachelors, Masters, Technical School, Doctoral, and Corporate training. A person might need to select more than one. A person may have a Bachelors and have attended a tech school. The check box allows you to select more than one option.



The properties you are concerned with are: caption and style. The style denotes either a standard check box or a graphical check box.

Option box: The option box works in much the same way as the check box except that only one can be selected at a given time. It is used when the user must select 1 option from several choices.



Here is an example of using an option box: Let us assume that you have an employee tracking program that is connected to a database via a data control (data1) and you wish the user to select the employees status. You could place three option boxes on the form named optSalaried; optFullHrly; and optPart. Then in your cmdsave button you could have an if then loop that selects which option box was selected and then assigns an appropriate value to the database:

Cmdsave_click()

If optsalaried.value = true then

Data1.recordset.fields("Employee Status") = "salaried"

Elseif optfullhrly.value = true then

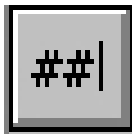
Data1.recordset.fields("employee Status") = "Full Time Hourly"

Else

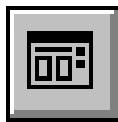
Data1.recordset.fields("Employee Status") = "Part Time"

End if

Masked Edit Box: This is like a text box but you can format the way in which user can input data. You just type into the masked property what format you want the data in. For example: ##/##/## .



Common Dialog: This is a very useful control. With it you set up a dialog box, as is standard in all windows programs, whereby the user can search their computer for a particular file of a particular type.



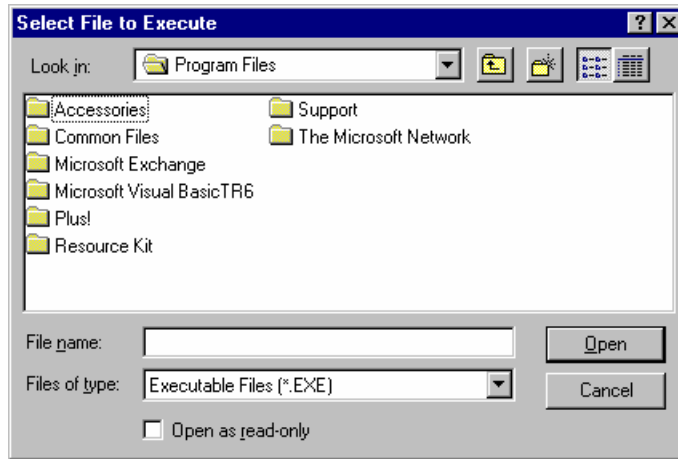
The properties you are concerned with are:

Dialog Title: This gives a title to your dialogue box

Init Dir: This selects an initial directory to open in

File Ext: This gives a file extension for files that are saved

This control is used in the MDI example under the VB/Samples directory. Once you have set these appropriate properties (and the caption) you can utilize the control in your applications. Your users will see this when the common dialog control is executed:



The filter is that property which tells it what kind of files to open or save. For example you might want a dialog box that looks for text files (*.txt) then you would set the filter as follows:

```
CommonDialog1.filter = "Text Files (*.txt)|*.txt"
```

The first section – “Text Files (*.txt) is what your user will see in the “Files of Type” section of the dialog box. The second part - |*.txt is the part the program needs to read. You can have several file types. Simply add another | then do the same for each file type you want. For example if you wanted your dialog box to look for text files and for batch files:

```
CommonDialog1.filter = "Text Files (*.txt)|*.txt | Batch Files (*.bat)|*.bat"
```

Then you will need to tell the dialog box whether you wish to look for a file to open, or to save. You do this with a very simple method:

```
CommonDialog1.showopen
```

Or

```
CommonDialog1.showsave
```

You will probably find many uses for the common dialog box.

DirListBox: A DirListBox control displays directories and paths to the user. This control can display a hierarchical list of directories. You can create dialog boxes that allow a user to open a file from a list of files in all available directories.

DriveListBox: A DriveListBox control enables a user to select any disk drive (floppy, CD, DVD, Hard disk, Zip, Jazz, networked, etc.) You can create dialog boxes that allow the user to open a file from a list of files on a disk in any available drive

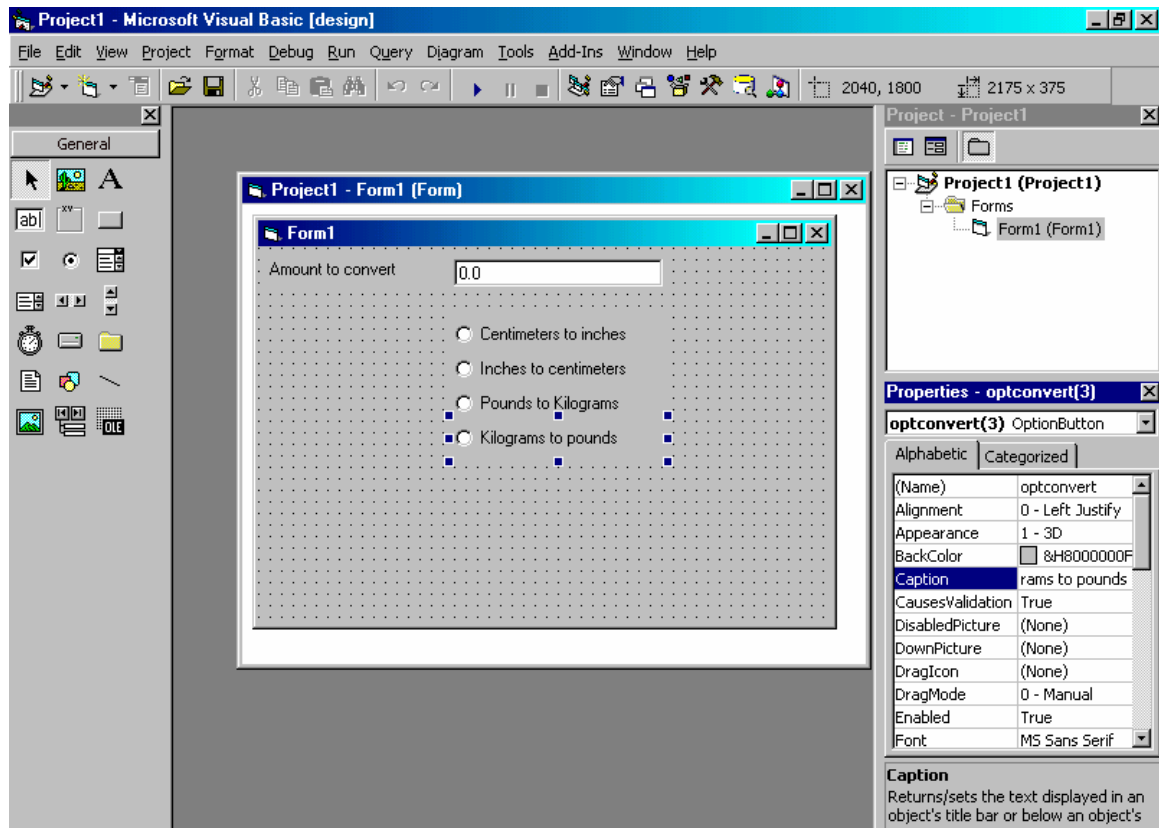
ImageControl: Use the Image control to display a graphic. An image control can display jpeg, gif, bitmaps, metafiles, and icons. This control uses fewer system resources and repaints faster than a PictureBox control, but it has fewer properties and methods than the picture box.. Use the Stretch property to cause the image to scale to fit the image control.

Example 4-1

We are going to write a simple program that will simply do some conversions

Step 1 Start a new standard exe project and place 1 text box and one label on the form.

Step 2: Place an option button on the form and name it optconvert. Then highlight that button and copy it (either via 'edit' > 'copy' or using the ctrl-c shortcut) then past it back on the form. You will be asked if you wish to create a control array, say yes. Continue pasting until you have a total of 4 option boxes. Change their captions so that your screen looks like this picture:



Step 3: Add a single command button, name it cmdconvert, and make its caption 'convert'

Step 4: Place the following code inside the command click event
On Error Resume Next

```
Dim amount As Single  
Dim answer As Double
```

```
amount = Val(Text1.Text)
```

```
If optconvert(0).Value = True Then  
    answer = amount / 2.54  
ElseIf optconvert(1).Value = True Then  
    answer = amount * 2.54  
ElseIf optconvert(2).Value = True Then  
    answer = amount / 2.2  
ElseIf optconvert(3).Value = True Then  
    answer = amount * 2.2  
End If
```

```
MsgBox ("The answer is " & answer)
```

Now the purpose of this example was to expose you to using option buttons and arrays of controls.

As you probably suspect, these controls also have naming conventions:

<u>Control</u>	<u>Naming Convention</u>
<i>Check Box</i>	ChkName
<i>Option Box</i>	OptName
<i>Common Dialog</i>	cmdlgName

Section II Coding Issues:

In chapter 3, we looked at many basic command words and loop structures. In this chapter, we are going to look at some more advanced items. It is imperative that you have a firm grasp of the items discussed in Chapter 3 before you move on to the following items. If you need to review, please do so before continuing:

Arrays: An array is a group of similar items. In code, it is not often used in Visual Basic but it can be very useful with controls. Let's say you have four buttons you need to create and they all are related and must be the same size. Let's say they are buttons on a toolbar. You can create one button then copy it and paste it to a frame. You will get a message box that informs you that you already have such a control would you like to make an array. Click the "Yes" button.

You can then paste as many buttons as you like each will be designated by the name of the first button followed by an index number in parentheses: cmdToolbar(1) or cmdToolbar(2). All arrays start at zero so your original button will now be cmdToolbar(0). There are three reasons why you may want to do this:

1. Less memory is used by an array of controls than by several individual controls.
2. You group controls with a related purpose into one area. For example all database related command buttons can be put in an array called cmdDatabase.
3. The code is easier to read.

The third reason is because all the code will now appear in the same place. These command buttons have a common click event, mouse move event, etc. This is also why less memory is required. Now this raises a question: When clicking a button, if all the buttons share a click event, how do you write the code?

Well the most common method is to use the select case method. This is much like the if-then loop. Let's look at an example of data control related command buttons in an array named cmddata:

```
Cmddata_click()
```

```
Select case index ' the index is the number designating which button we are addressing it  
    ' begins with zero for the first button in the array and continues  
    ' sequentially.
```

```
Case 0 'first button in the array
```

```
    Data1.recordset.addnew
```

```
Case 1 'second button in the array
```

```
    Data1.recordset.delete
```

```
    Data1.recordset.movenext
```

```
Case 2
```

```
    Data1.recordset.update
```

End select

Note how all the database code can now be found in one place, you can also have one error handler for all of the database related buttons. The use of control arrays and the Select Case statement will make your code smaller and easier to read.



Tip: (actually two tips) 1. Arrays start with 0 not 1. Your first button will be `cmdbutton(0)`. Also, notice that ALL events will be shared in common by a control array and will need to use select-case statements.

Sub Routines and Functions

Subroutines

Basically events where merely sub routines and functions that Visual Basic had already included for you. You can go to the general declarations section of any form or code module and declare a new sub routine like this:

```
Private sub mysub()
```

```
End sub
```

Now you can place any code you want in your subroutine. Then to access that code at any time in your program you just place the line of code “mysub” and that subroutine will be launched. NOTE: Subroutines created in a form are only accessible to that form, subroutines created in a code module are available to the entire program. A good example is the subroutine from chapter three used to highlight text. You can place this in a code module in your project:

```
Public Sub SetSelected()
```

```
'This sub simply highlights/selects the text  
'in a textbox/label/etc when that control has  
'focus. You can place it in the got focus event of the text box. Or you could  
'place it in a separate subroutine then in the got focus event of all your text boxes  
'simply place the word SetSelected.
```

```
Screen.ActiveControl.SelStart = 0
```

```
Screen.ActiveControl.SelLength = Len(Screen.ActiveControl.Text)
```

```
End Sub
```

Now in any text box on any form you can place in the got focus event:

```
Setselected
```

And the text in that text box will be highlighted as soon as the text box receives focus. You could of course place highlighting code in each and every text box, but using a subroutine is a much better way.

Functions:

Now a function is merely a subroutine that accepts a value passed to it. So, the function is in essence a special case of the subroutine. Any function must return a value. The key points to remember in using a function is to make sure you pass it an appropriate value. For example if your function is looking for an integer and you pass it a currency variable you will get an error.

A good example is the slow down function from the previous chapter:

```
Public Sub SlowDown(itime As Integer)
```

```
'This subroutine will simply wait a number of seconds  
' for example you can place the code slowdown 1 in your app  
'to make it wait 1 second. It will only recognize integer values
```

```
Dim lStartTime&
```

```
lStartTime = Timer
```

```
Do While Not Timer >= lStartTime + itime
```

```
    DoEvents
```

```
Loop
```

```
End Sub
```

Notice that in the parenthesis we now have a value “itime as integer” this means that an integer variable called itime will be passed. Now when you call this function rather than just referring to its name, as you do with subroutines, you must follow the name with the value you wish to pass it:

```
Slowdown 1
```

You can use functions for any type of subroutine in which you wish to pass a value to that subroutine. This is very important.



Tip: Make sure that you are only passing to the function the proper type of variable. Don't try passing 1.5 to the slowdown function since it is looking for an integer.

You might even do some error checking before sending the variable. For example you might have the user input a number in a text box and you then check to make sure that value is an integer before you pass it. You can also use a special key word called "cint" that converts a value to the nearest integer equivalent. Such as
Dim iSeconds as integer

Let iSeconds = Cint(mytextbox.text)

Slowdown iSeconds

API(Applications Programming Interface)

The API is a set of about 600 functions that are built into windows and you can call them from your program at any time. Using API calls can drastically enhance your programming skills. API stands for Application Programming Interface. It is a rather large set of functions built into Windows that you can use. Basically you declare the API function, usually in the declarations section of a code module, and then you call the function in your program as you would any other function. Let me give you a few examples:

1. One I use often simply brings the form calling it to the forefront. I find it easier than using the zorder:

The Declaration is

```
Declare Sub BringWindowToTop Lib "User" (ByVal hwnd As Integer)
```

Then anywhere you want to use it you just call

```
BringWindowToTop me.hwnd
```

And viola! Your form will be in the forefront of all windows!

2. How much free memory do I space do I have, well there is an API call for that:

The Declaration is:

```
Declare Function GetFreeSpace& Lib "Kernel" (ByVal flag%)
```

and the call is:

```
x& = GetFreeSpace(0)
```

3. How much System resources do I have open? Well guess what you can find out with an API call:

The Declaration is:

```
Declare Function GetFreeSystemResources& Lib "User" (ByVal flags%)
```

The call is just

```
x& = GetFreeSystemResources(0)
```

4. Put my window on top:

```
Declare Function SetWindowPos Lib "user32" Alias "SetWindowPos" _  
(ByVal hwnd As Long, ByVal hWndInsertAfter As Long, _  
ByVal x As Long, ByVal y As Long, ByVal cx As Long, _  
ByVal cy As Long, ByVal wFlags As Long) As Long
```

Now there are over 600 API calls you can use. The good news is that in addition to the many books written on the subject, VB ships with the API Viewer. This will let you look up API calls then copy and paste the declarations into your code!! See how neat VB is, it does a lot for you. Now remember that 32 bit API calls are different than 16 bit (Actually they just have a slightly different declaration). However, the good news is that Windows 95 will have support for both the 16 bit and 32 bit API calls! Remember that the declarations for API calls must go in the declarations section of a Code Module. Then you can call them from anywhere in the program you wish to.

I hope these three examples give you an idea of how to use the API and I hope you delve deeper into this topic.



Tip: Spelling, Spelling, Spelling!! The API calls are even worse than the SQL statements! One minor misspelling and it will not work. And with all the weird spelling its hard not to misspell *something*

Now you can program all you want simply copying and pasting API calls without really understanding all that goes into them, but I think a few notes of explanation are in order. You may have noticed a lot of words in an API declaration such as “Lib”, “Alias”, and others (if you didn’t notice, then go back up and take a look). Well let me at least explain to you the various things you might find in an API declaration and what they mean:

Private or Public: This is just like variable declaration. It tells you the scope of the declaration.

Declare: This is the key word for API calls. When you see this you know that we are talking about an API call.

Sub or Function: Just like in routines, is this a sub or will it return a value (a function).

Lib: This is a keyword required by API declarations.

Name: This is the name that will be used in your code to reference this procedure.

Alias: This is not actually required but can often be useful.

As Type: What kind of value will be returned if this is a function.

Now it is important to realize that API’s are really designed for C and C++ programmers and that’s why some of the declarations seem odd to Visual Basic programmers. Things like “Lib” (C uses libraries). If you want to explore this topic more fully I recommend “The VB6 Win32 API tutorial” from Wrox Press. You might also want to get just a basic grasp of C programming (it will also make you really appreciate VB!). If you are interested in that, then I recommend “Teach yourself C” by Herbert Schildt. After you have experimented with Visual Basic for a few months, it would be a good idea to at least play with C a little bit. It will give you a better understanding of API calls and a better appreciation for all the headaches Visual Basic handles for you!

More keywords:

There are several other key words you can use in Visual Basic to do a number of useful tasks.

Cbyte: This converts a variable of one type into a byte. If the expression you are converting lies outside the acceptable range for a byte you will get an error. The syntax is: Cbyte(myvariable)

Cdate: Much like Cbyte, this expression converts a variable to the Data type. Its syntax is: Cdate (myVariable)

There are also statements like Clong, Cint, etc to convert a variable to an integer or a long value.

Chr: This returns the actually character denoted by an ASCII character number for example chr(65) returns 'A'

Clear: This keyword can be applied to the clipboard object, a combobox, or a listbox. It causes all items stored therein to be cleared.

```
List1.clear
```

Cos: This will return the cosine of a number. For example you can say "Cos(15) to get the cosine of 15.

FileCopy: This command causes a file to be copied from its source to a destination file. You might do something like : FileCopy "sourcefile.txt" ,"destinationfile.txt"

FileLen: This function returns the length of a file in bytes. Its syntax is : Filelen("myfile.txt")

IsDate: This function will return a boolean value (True/False) indicating whether or not a variable can be converted to a date. The syntax is: Isdate(myvariable).

IsNull: This function is used to see if another variable is null. You use it like this: isnull(myvariable)

Sin: This will return the Sine of a variable such as sin(45) returns the sine of 45.

Tips and Tricks:

The following are some simple tips and tricks that you can use. Many of these are simple subroutines of functions that you can put in your code in order to make it more effective:

System Modal Forms

If you want a form to be system modal* then:

Declare this:

Declare Function SetSysModalWindow Lib "User" (ByVal hWnd As Integer) As Integer
and call this:

```
oldSysModal = SetSysModalWindow([Form].hWnd)
```

*System modal means that the user cannot do any other function until they have finished with this form. This is an excellent way to password protect your program.

What kind of Drive is it?

```
Declare Function GetDriveType Lib "Kernel" (ByVal nDrive As Integer) As Integer  
Global Const DRIVE_REMOVEABLE% = 2, DRIVE_FIXED% = 3  
Global Const DRIVE_REMOTE% = 4
```

Centering a Form

```
Public Sub CenterWindow(f As Form)
```

'This routine will center the form calling it on the
'screen. Just place the line " centerwindow.me" in the
'form you wish to center

```
    f.Top = (Screen.Height * .5) - (f.Height * .5)  
    f.Left = (Screen.Width * .5) - (f.Width * .5)
```

```
End Sub
```

Dimensioning Variables

Some programmers try the following

```
Dim iNum, iNextNum, iLastNum as Integer
```

Believing that all these 3 variables end up as Integer. However the first two end up as default data type: Variant.

Instead you should do this

```
Dim iNum as Integer
```

```
Dim iNextNum as Integer
```

```
Dim iLastNum as Integer
```

Highlighting Text

```
Public Sub SetSelected()
```

```
'This sub simply highlights/selects the text
```

```
'in a textbox/label/etc when that control has
```

```
'focus
```

```
Screen.ActiveControl.SelStart = 0
```

```
Screen.ActiveControl.SelLength = Len(Screen.ActiveControl.Text)
```

```
End Sub
```

Closing another Program:

```
' Close an existing program
```

```
title = "MyAPP" 'The title of the app you want to close.
```

```
ihWnd = findWindow(0&, Title)
```

```
ihTask = GetWindowTask (ihWnd)
```

```
iRet = PostAppMessage(ihTask, WM_QUIT, 0, 0&)
```

Now this code requires the use of a few API calls, but you can use the API wizard to grab their declarations and paste them into your code!

Does a file Exist?

```
Function FileExist(Filename as string) as Boolean
```

```
FileExist = IIf(Dir(Filename) <> "", True, False)
```

```
End Function
```

More Effective Programming:

The strength and weakness of Visual Basic lies in the fact that you don't have to code well to make a program work. Visual Basic will let you write pretty sloppy programs that will still run. This is a strength in that a novice programmer can create functional applications. However, it is also a weakness in that code maintenance can be a nightmare and that applications performance is often much less than it could be. We have already discussed issues such as commenting and indentation. Now lets look at some methods to really improve your code.

OPTIMIZING FOR SPEED

At times, you will need to improve the speed of execution for your application. There are a number of techniques you may find useful. The first and most important technique is to use integers whenever possible. Visual Basic considers all variables to be of the default Variant type. Unless you specify otherwise, everything, text and numeric, will be a Variant.

A 50-Million-Loop Time Test

```
Dim i As Long
```

```
Dim b As Byte
```

```
Dim start As Double
```

```
Dim Elapse As Double
```

```
start = Timer
```

```
For i = 1 To 200000
```

```
    For b = 1 To 254
```

```
        Next b
```

```
    Next i
```

```
    Elapse = Timer - start
```

```
Print Elapse
```

```
End Sub
```

When this Form loads, the inner loop counter b is defined as a Byte variable type for the first test. The starting and ending times are held in the variables start and Elapse. We make them so they will permit great precision. Then we begin with start = Timer to get the current time. We use the Timer function in VB (unrelated to Timer Controls). It calculates elapsed time since midnight each day, then resets. Then we start running the nested loop. When the loop finishes, we subtract the starting time from the current (Timer) time and print the results on the Form.

To test each variable type, all we have to do is change the type of j in the first line of the program: Dim j as Integer, Dim j as Long, and so forth. Below you can see which variable is fastest. (Actual times will vary on different machines)

<i>Integer</i>	1
<i>Long</i>	2
<i>Byte</i>	3
<i>Currency</i>	4
<i>Variant</i>	5
<i>Single</i>	6
<i>Double</i>	7

As you can see, the type of variable you use can make a difference in application performance. The Integer variable type runs quite a bit faster than the Floating-Point types. The Variant-considering that it must analyze context and requires more storage in memory than any other type-runs surprisingly fast. However, you can gain about 40 percent in speed by simply using an Integer instead of the default Variant.

Obviously, you can't use integers in all your programs, everywhere. Calculations requiring greater precision (a result including a fraction) must be made with Floating-Point variable types (Single or Double). Calculations requiring a large range cannot be made with integers (which can manipulate only the numbers between -32768 and 32767). In that case, you have to resort to "long integers" (Long), Floating Point (Single or Double) or Currency data types.

One way to speed up animation is to use an Image Box rather than a Picture Box (which has many more Events). Unfortunately, a 15 percent improvement is often barely noticeable to the viewer. People don't notice minor changes in speed. Our concern is not pure computational speed, but the perceived, speed.

There are a number of ways to increase the users perception of the applications speed. One method is to load all forms at the outset of the program. For example if your application has 5 forms in it, rather than simply load the main form then load each additional form when you need it, load them all. Just set the visible property of all forms to false. Then when you need the form set its visible to true. This will make its to appear to load instantly.

The AutoRedraw Property is available only for a Form or a Picture Box. AutoRedraw is by default False. Set it to True, and VB will create everything into the Form or Picture Box. Nothing will be redrawn on the fly. VB will keep a copy, a bitmap

picture, of the entire Form or Picture Box in memory. This will increase speed of the application, but remember that, like the method of loading the forms all at once, this does tax the memory of the machine. Fortunately, most modern machines have plenty of RAM!

Also, remember a few simple tricks like an image box displays faster than a picture box. Also, do not use subroutines for code that is not being called from multiple places. If code will only be used in one or two events, put that code in that event. It is faster for an event to execute code that is there in the event, than to call out to an external (external to the control that is) subroutine.

Reading a Flat File:

Flat files are basically text files that can be created and read through virtually any word processor, including the windows notepad. The System.ini, Config.sys, Autoexec.bat, and win.ini are all examples of flat files. Below is an example of how you might read a flat file, like the system.ini into a textbox.

```
Public Sub Read_Ini()  
Dim get 'A variable that holds the string  
  
Open "C:\WINDOWS\SYSTEM.INI" For Input As #1  
Do While Not EOF(1)  
  
    get = Input(1, 1)  
    txtIni.Text = txtIni.Text & get  
  
Loop  
  
Close #1
```

Now this can be very useful in your programs. You can, for example, store some user-defined data in a flat file that your program reads. For example if your program accesses a database stored on a network, you can use an ini file to point to that network location. Just place in your ini file this line:
DB x:\myfolder\

Then you can use this code to read the location:

```
Open "C:\myprogram\my.ini" for input as #1  
  
Do While Not EOF(1)
```

```

Get = input (1,1)

If Left(Get,2) = "DB" then

    Let iLength = Len(get)           'get the length of the line
    Let sDBPath = right(get,iLength -4) 'get all but the left 4 characters

End if

```

Basically this code will see if the first to letters of the line in your ini file are "DB" if they are then take the rest of that line (minus the equal sign and spaces) and use that as the sDBPath variable to tell your program where to find the database.

You have an even easier way to open and write to flat files if you know exactly what kind of data you are putting in there. Lets say you have a program that takes 5 variables about an employee and stores them in a flat file so that they can be attached to an email and sent (any email reader will be able to read a flat file). Here is how you would take those 5 variables (perhaps from an MS Access database) and write them to a flat file:

```

Public sub write_file()
Dim sName as string
Dim sSSN as string
Dim sPhone as string
Dim sTitle as string
Dim sSalary as currency

Open Filename for output as #1

Write #1, sName, sSSN, sPhone, sTitle, sSalary

Close #1

```

Now lets analyze this code. The first part simply says to open a file designated by the variable "filename" for output. You might utilize a common dialog box to let the user select a file name and location. You simply have

```
Filename = commondialog1.filename
```

Then you call the subroutine to write to the file

A file of that name will be created in the location designated by the common dialog box. Then you write those variables to the file. Now lets say you wanted to go through your entire employee database table and write it to a flat file. Well let's just make a few modifications to our code:

```
Private sub Write_file()
```

'dimension your variables

Dim DB as database

Dim RS as recordset

'point to the appropriate database and table

Set DB = OpenDatabase(spath & "\mydatabase.mdb")

Set RS = db.OpenRecordset("mytable", dbOpenTable)

'move to the beginning of that table

Rs.movefirst

Open Filename for output as #1

Do until rs.eof

 Sname = rs.fields("Employee Name")

 SSSN = rs.fields("Employee Social")

 SPhone = rs.fields("Employee Phone Number")

 STitle = rs.fields("Employee Title")

 SSalary = rs.fields("Employee Salary")

 Write #1, sName, sSSN, sPhone, sTitle, sSalary

 Rs.movenext

Loop

Close #1

Now reading a flat file is just the opposite:

Open filename for input as #1

 Read #1, sname,sssn, sphone, stitle, ssalary

Close #1

With the strength and versatility of modern relational database systems flat files are not generally used for data storage. However they can be great if you want to send data via email or even modem. Flat files are also good for storing program variables you would like the user to be able to easily set. They can open a flat file in Windows Notepad and set variables.

Some Simple Graphics

The image box and the picture box both provide a method for placing graphics on your programs. You have probably already discovered that you can take images and put them in an image box or picture box. I prefer the picture box because its stretch property will cause the picture to conform to the dimensions of the picture box. In earlier versions of Visual Basic, only bitmaps (*.bmp), Icons (*.ico), and metafiles (*.wmf) could be used. Now you can also add in gifs (*.gif) and jpeg files (*.jpg). This is important because jpeg files and gifs are the type file you will find on the internet.

Now you probably use some static images, at least in your splash screens or on your about screens. You can make a simple animation by moving the picture box. For example if you have a picture of an arrow in a picture box and you wish it to travel across the screen:

```
For x = 1 to 1000
```

```
    Let picturebox1.left = picturebox1.left + 10
```

```
Next x
```

Now note a few things here. First, how far your for-next loop will need to progress depends entirely upon how wide your form is. Secondly, the amount you increment the left property by is very important. The larger the number the less smooth your animation will seem. However too small a number and the image will move too slowly. You can play with this a bit and decide what works for you.

A simple Graphics Program:

Here is a very simple animation program you can write to get a feel for using graphics in visual basic.

Example 4-2

Place three image boxes on your form each one has a picture of a butterfly. Name Each one "Main", "OpenWings", and "CloseWings". You will also need to place a timer control on the form. Now for the code:

In the timer control timer event place this code:

Static iPicture As Integer

```
Main.Move Main.Left + 20, Main.Top - 5
```

```
' For a variation, use the following line instead of the preceding one.
```



```
' Main.Move (Main.Left + 20) Mod ScaleWidth, (Main.Top - 5 + ScaleHeight) Mod  
' ScaleHeight
```

```
If iPicture Then
```

```
    Main.Picture = OpenWings.Picture ' Display the open butterfly picture.
```

```
Else
```

```
    Main.Picture = CloseWings.Picture ' Display the closed butterfly picture.
```

```
End If
```

```
iPicture = Not iPicture
```

Basically this routine causes you to move the image across the screen and to change from the open wings to the close wings image and back several times. Simply using a few complimentary images and the move event and you can make a nice little animation!

Debugging:

Let's face it, you will get bugs in your programs. Its that simple. But how do you find them. Well there are a few techniques to help you track them down.

1. Compile on Demand : On your drop down menu under "tools" you have an "options" section. This will present you with a series of tabs. If you select the "General" tab, you will see a check box on the bottom left hand side that says "Compile on Demand". This means that when you run your program that Visual Basic will only compile the parts you actually utilize. In other words when this option is checked, if you have errors in subroutines you don't happen to use you will never know it. This option is only meant to be used when you are building your application. When you are ready to test, turn it off.

2. If you have an error in a subroutine you can "step – through" your program. The way you do this is to go to the last line of code that you are sure works and highlight it . Then you go to the drop down menu and select "De-Bug" and then select "Toggle Breakpoint". Now when you run the program it will stop at this line of code. You can then go through the code line by line simply by selecting "De-Bug" and "Step Into". The program will execute one line at a time. At each line you can highlight any variable and select "De-Bug" and "Instant Watch". This will tell you what is currently in this variable.

These two simple debugging techniques allow you to more thoroughly check out your program before you make an executable and distribute it. One of the biggest problems in software development today is software that has not been thoroughly tested.

Don't forget a few other concepts:

Structure: Too often Visual Basic programmers just slap together some forms and code modules in a goo they call code. You must have some structure and design to your program. Lack of structure is one of the most common causes of bugs. Take the time to sit down and think out a program before you start to develop. Maybe even draw out a chart of the major routines and how they connect to each other and to the data. Think about how you want the program to flow.

Spaghetti Code: This is a term for code that is so convoluted that it is nearly impossible for someone to follow its flow. An example of spaghetti code would be a routine that calls another routine that in turn calls another routine.... You get the idea. You would be better off to have each routine called from the main event that triggers them (like the click event of a command button) something like:

```
CmdButton private sub_click()  
  
    First_sub 'call the first sub  
  
    Second_Sub 'Call the second subroutine  
  
    Third_Sub 'Call the third sub  
  
End sub
```

Here are my personal 10 commandments of software quality control:

1. Always test every single option. Click every button no matter how trivial!
2. Place Error handling in EVERY subroutine.
3. Make your error handling give useful error messages.
4. Have someone else run and test your program.
5. Install your program on another machine and run it.
6. Try entering invalid data, your program should not allow it.
7. Try clicking buttons that you should not click yet (like a save button before you have added or edited anything).
8. Read through your code to see if anything looks inappropriate.

9. If you get an error at any stage in this process, correct that error then start over from the beginning.
10. Remember that no matter how good your program is, a single bug will cause serious user dissatisfaction.

Planning:

Before you write a single line of code, try planning your project out. There are a number of formal planning strategies available but at a minimum you should try the following:

1. Clearly write out the end user needs of your application. Make sure you have clearly defined the goals.
2. Write out a plan for the major points in the flow of your application.
3. Research any new technologies or techniques you may require.
4. Plan out your code in moderate detail. Decide how many forms and what content those forms will have. Decide what code should be in events and what should be in code modules. Decide on the scope of variables. Decide on what methods and algorithms you will use.
5. Now you can start writing code.

Microsoft recommends you use the Microsoft Solutions Framework which consists of 4 phases:

1. Envisioning : do some brainstorming, figure out what you want to have happen in your code.
2. Planning: Now do formal planning showing how you are going to make your vision a reality.
3. Coding: Actually write the code.
4. Stabilizing: Test and debug.

Testing:

You have to test your code!! I do not care how simple the application is, bugs can get in there. There are also a number of sophisticated formal testing strategies available but for beginners just try to follow these steps:

1. First re read your code. Make sure you have proper error handling. Make sure there are no obvious flaws in your logic.
2. Refer back to your user defined goals and check to see that your application meets them.
3. Now run the program. You should have a written table showing the exact testing conditions (the PC, Operating system, etc.) and exactly what items you will test.
4. Try everything. Specifically try doing things the WRONG way, trust me users will.

Remember also that your testing plan should account for 3 things:

1. Valid Data: When you put correct data in, and do it in the correct format, does your program work?
2. Invalid Data: What does your program do if you put in bad data? What if you enter a string where you were supposed to enter an integer?
3. Extreme Data: How does your program handle extreme data? If an employee tracking program is given an employee age of 174 what does the program do?

Review Questions:

1. What variable is the fastest?
2. What is a common dialog control used for?
3. What properties must I set with a common dialog control?
4. What is the main difference between a checkbox and an option box?
5. What does API stand for?
6. List one API call.
7. What is the difference between a subroutine and a function?
8. What is the got focus event?
9. Why would I use a control array?
10. Discuss the Select Case Statement
11. What statement is used to see if a variable is empty?
12. What does cint do?
13. How do I write code to get the sin of 57?
14. What does the FileLen keyword tell me?
15. Give an example of FileCopy?

Chapter Project #1:

Go back to your rolodex program and add in the following:

1. An “about” screen with a picture box that moves across the screen
2. A flat file that holds the users name, read that in and display it in the caption of The main form.
3. Add in a shell command that launches notepad, then use keystrokes to send some message to it. This can be something like a note generator in your rolodex program.
4. Make sure that all textboxes are highlighted when they receive focus.
5. Use an API call to center the main form when it is loaded.

Chapter Project #2:

Thoroughly test your program using the 10 commandments of quality control. Then prepare a word document that outlines your testing result. This quality control report should indicate

- the number of times a complete test was run,
- the results of each test run,
- any errors that where found.
- Steps taken to correct those errors.

- All operating systems and machines you tested on

The purpose of this project is to introduce you to the concept of Quality Control in software development.

Chapter Five Object Oriented Programming

Chapter Objectives

- **Understand the basic concepts of object orientation**
- **Become familiar with terminology**
- **Be able to create a class**
- **Be able to use Visual Basics standard collections**
- **Be able to build a user defined collection**

Chapter V: Object Orientation

Introduction: You have probably heard this term tossed around quite a bit. It sounds pretty mysterious and has probably left you with many questions. Well let me tell you a little secret: you have already been doing object oriented programming since you started this book! Objects are basically templates that you manipulate. Every control you put on a form (and, incidentally, the form itself) is an object. When you change the place a text box on a form and change one of its properties, what you have actually done is created an *instance* of the textbox object and changed one of its *members*.

It is important to realize that you deal with objects every single day. The car you drive is an object. It has properties such as size, shape, color, etc. It also has methods such as accelerate, brake, etc. You don't have to know how a car works, you just have to know how to call its methods. That is how an object works in programming.

Perhaps this example is showing you why object oriented programming is so cool. You can create a text box, manipulate its properties, and utilize its events without having the slightest idea how a text box is made. This is often referred to as the "Black Box" approach. You can utilize the black box without knowing how it works inside. You just have to know what data to feed it.

Before we move forward into showing you how to create your own objects, lets discuss a little object-oriented terminology:

OOP: This means object oriented programming.

Instance: This is basically the current incarnation of your objects template. Think of the general object "car" and you own care as a specific instance of the "car" object.

Class: This is a special kind of code module that is used for you to make your own objects. It is a template for objects you create.

Encapsulation: This is probably the single most important part of object oriented programming. It basically refers to the fact that an object stores both data and the methods to manipulate that data in the same place. This way you deal both with functions and data simultaneously and all the relevant code is in a single place.

Inheritance: This refers to the ability to make classes from other classes. This is not available in Visual Basic.

Polymorphism: This refers to the ability to define a new object based on an existing object and then simply alter the new object. This is not available in Visual Basic.

New: This keyword denotes a new instance of an object. For example:

Dim myclass as new original class

Classes:

Lets talk about classes, since that is how you do object oriented programming. A class is a special kind of code module. A class will have certain items. Any subroutines or functions that you add will be your class(objects) methods. Any variables you add will be your class (objects) properties.

When you first add a class to your project it will have a declarations section and a class section. Under the class section two events are there. One begin initialize and the other terminate. The initialize event is where you place any code that you wish to execute as soon as an instance of the class is created. The terminate event is where you place code when you wish to execute when the class is destroyed. This is a good place to put any “clean up” code. Under the general declarations section you will declare any variables (properties) of your class (object) that you might need. All variables should be declared as private. One of the reasons you use a class is to protect data. The variables can ONLY be accessed via methods(subroutines and functions).

Class modules are different from standard modules primarily in the way their data is stored. There's only one copy of a standard module's data. Since you cannot create separate instances of a standard module. This means that when one part of your program changes a public variable in a standard module, and another part of your program subsequently reads that variable, it will get the same value. This means that the standard code modules variables are global and that any change done to them is changed for the entire application.

Class data, on the other hand, exists separately for each instance of the class. Each instance of a class is a separate entity. Remember that the class module itself is simply a template for the class objects you create. Therefore if you have more than one instance

of a class module, a change in the data of one module has absolutely no bearing on the data in other class modules.

Below I have outlined for you a simple class used to access databases

Example 5-1

Option Explicit

```
*****  
' This class provides easy access to any type of database via ADO  
*****
```

' Database Variables

```
Private m_strSQL As String           ' Used for SQL Queries  
Private m_ado_command As Command    ' Ado Command object  
Private m_ado_conn As Connection     ' Ado Connection Object  
Private m_recordset As Recordset     ' Ado Recordset Object
```

' Variables exposed via properties

```
Private m_strServer As String        ' The name of the server that  
                                     ' holds the database  
Private m_strPath As String          ' the database path
```

```
Private m_cursorlocation As Integer
```

' General use variables not exposed via properties

```
Private m_boolDone As Boolean        ' indicates that a process is complete  
Private m_strMessage As String       ' used in certain external events
```

' Public events

```
Public Event DataAccessError(sz_errmsg As String)
```

' Database type constants

```
Const DTACCESS = 1  
Const DTSQL70 = 2
```

' cursorlocation constants

```
Const DTLOCAL = 1  
Const DTSERVER = 2
```

```

'database constants
Private Const adStateClosed = 0
Private Const adStateOpen = 1
Private Const adStateConnecting = 2
Private Const adStateExecuting = 4
Private Const adStateFetching = 8
Public Sub connect_to_database(DBType As Integer)
'*****
' This function takes an integer that will tell it what type of
' database connection to make. It then uses ado to connect to that
' database. Right now it only connects to Access and SQL Server 7.0
' but adding new types on is a trivial matter.
' - Chuck Easttom May 2001
'*****
Dim strConnect As String

```

```

On Error GoTo errorhandler

```

```

' select the connection parameters based on the database
' type passed to this method
Select Case DBType
Case DTSQL70
' add server name to connection string
strConnect = "Provider=SQLOLEDB.1;Persist Security Info=False;User
ID=sa;Initial Catalog=LERG;Data Source="

```

```

' add the database path name to connection string
strConnect = strConnect & m_strServer

```

```

' set connection object provider string
m_ado_conn.Provider = "SQLOLEDB.1"

```

```

' set connection object connection string
m_ado_conn.ConnectionString = strConnect

```

```

' set up a client side cursor
m_ado_conn.cursorlocation = adUseClient

```

```

' sixty second timeout
m_ado_conn.CommandTimeout = 60

```

```

' set mode to read and write

```

```

        m_ado_conn.Mode = adModeReadWrite
    Case DTACCESS

        ' add server name to connection string
        strConnect = "Provider=Microsoft.Jet.OLEDB.3.51;Persist Security
Info=False;Data Source="

        'add the database path name to connection string
        strConnect = strConnect & m_strPath

        ' set connection object provider string
        m_ado_conn.Provider = "Microsoft.Jet.OLEDB.3.51"

        ' set connection object connection string
        m_ado_conn.ConnectionString = strConnect

        ' set up a client side cursor
        m_ado_conn.cursorlocation = m_cursorlocation

        ' sixty second timeout
        m_ado_conn.CommandTimeout = 60

    Case Else
        RaiseEvent DataAccessError("Database Type not recognized")

    End Select

    ' Open Connection
    m_ado_conn.Open

Exit Sub

errorhandler:
    m_strMessage = "Connect Error : " & Err.Description
    RaiseEvent DataAccessError(m_strMessage)

End Sub

Public Property Get cursorlocation() As Integer
    On Error Resume Next
    cursorlocation = m_cursorlocation
End Property
Public Property Let cursorlocation(location As Integer)
    On Error GoTo errorhandler

```

```

Select Case location
  Case DTLOCAL
    m_cursorlocation = adUseClient
  Case DTSERVER
    m_cursorlocation = adUseServer
  Case Else

End Select

Exit Property
errorhandler:
  RaiseEvent DataAccessError("Invalid cursor location")

End Property

```

```

Public Property Get DatabaseConnectionState() As String
  '*****
  ' This property simply allows the user/programmer to find
  ' out what the current database connection state is.
  ' Chuck Easttom May 2001
  '*****
  On Error Resume Next

  Select Case m_ado_conn.State
    Case adStateClosed
      DatabaseConnectionState = "Connection is closed"

    Case adStateOpen
      DatabaseConnectionState = "Connection is opened"

    Case adStateConnecting
      DatabaseConnectionState = "Connection in progress"

    Case adStateExecuting
      DatabaseConnectionState = "Executing SQL statement"

    Case adStateFetching
      DatabaseConnectionState = "Retrieving data"

    Case Else
      DatabaseConnectionState = "Unknown State"

  End Select

```

```
End Property
```

```
Public Property Let DatabasePath(path As String)  
On Error Resume Next
```

```
    m_strPath = path  
End Property
```

```
Public Property Get DatabasePath() As String  
On Error Resume Next
```

```
    DatabasePath = m_strPath  
End Property
```

```
Public Sub disconnect_database()
```

```
*****  
' This subroutine simply closes the ado connection object  
' thus terminating the connection to the  
' database - Chuck Easttom May 2001  
*****  
    ' close database object  
    m_ado_conn.Close
```

```
End Sub
```

```
Public Function ExecuteSQLQuery(SQL As String) As Recordset
```

```
    Set m_recordset = New Recordset
```

```
    'set recordset parameters  
    m_recordset.CursorType = adOpenStatic  
    m_recordset.cursorlocation = adUseClient  
    m_recordset.LockType = adLockOptimistic  
    m_recordset.ActiveConnection = m_ado_conn
```

```
    ' execute query and return recordset  
    m_recordset.Open SQL
```

```
    Set ExecuteSQLQuery = m_recordset
```

End Function

Public Property Get SERVERNAME() As String
On Error Resume Next

SERVERNAME = m_strServer

End Property

Public Property Let SERVERNAME(Name As String)
On Error Resume Next
m_strServer = Name

End Property

Private Sub Class_Initialize()
' initialize variables to default values
m_cursorlocation = adUseClient
m_strPath = App.path

' create database connection object,
' it gets destroyed in the terminate event

Set m_ado_conn = New Connection
End Sub

Private Sub Class_Terminate()
Set m_ado_conn = Nothing
Set m_recordset = Nothing

End Sub

' Database type constants
Public Const DTACCESS = 1
Public Const DTSQL70 = 2

' cursorlocation constants
Public Const DTLOCAL = 1
Public Const DTSERVER = 2

Now to use this class, you first insert it into a new class module in your project. Then you have to create an instance of the class in a form you wish to use it in. You create a new instance like this:

```
Dim objdata as new clsdata
```

You can then access any of the items inside the class by simply calling objdata then a '.'? And then the name of the method or property you wish to use. Here is an example of using this object in a program.

```
' set up database connection
objdataaccess.cursorlocation = DTLOCAL
objdataaccess.DatabasePath = App.path & "\somedatabase.mdb"
objdataaccess.connect_to_database (DTACCESS)
```

```
sz_SQL = "SELECT * FROM [records]"
Set adorecordset = objdataaccess.ExecuteSQLQuery(sz_SQL)
```

With this little bit of code you will then be accessing the database you selected and getting a set of records from it based on the SQL query you passed it. While this particular class did not use all the methods a class might use here is a list of those methods and what they are used for:

Let	The property let would be used to change a property (variable) to a standard value (integer,string,etc). In our deckofcards example this was unnecessary since you would not want to change a cards value from outside the class.
Get	The get procedure is used when you want to get the value of a property (variable). For example if you wish to get the value of a card in our deckofcards example you can call the class and its get method: get current card: Mydeck.getcurrentcard*
Set	The property set is very much like the property let, but is used with object variables.

Important notes about classes:

DataBindingBehavior

Sets a value that determines if an object can be bound to a data source. All classes use the default of *vbNone* because the object will not be bound to a data source.

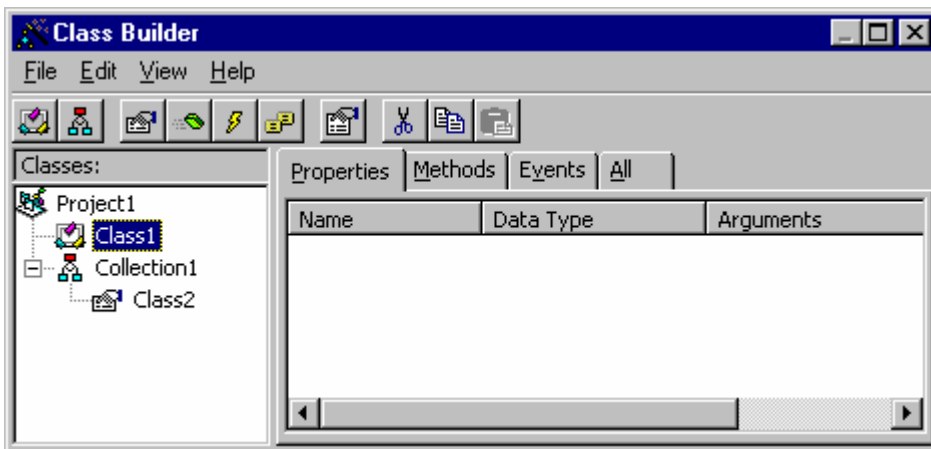
DataSourceBehavior

Sets a value that determines if an object can act as a source of data for other objects. All class modules use the default of vbNone because the objects are never sources of data for other objects.

Instancing

Sets a value that specifies whether you can create instances of a public class outside a project, and if so, how it will behave.. This allows other applications to create objects from the class . One instance of your class can provide any number of objects and the objects will be provided by the same instance of the component. Multi Use is more efficient with memory because it prevents additional copies of the component from being executed.

The class builder utility (shown below) is meant to help you set up classes more quickly and easily. You simply tell it what variables you want and it will create the appropriate properties. You also tell it what methods you want and it will create the basic frame work for them.



Simulated Inheritance:

Visual Basic has often been accused of not being truly object oriented in that it does not support inheritance. It is certainly true that Visual Basic does not support inheritance in a strict sense. For example in C++ you can have a base class and then any derived classes can access the properties and methods of that base class as if they were part of the derived class. A frequent example of inheritance would be the use of animal classes. Lets assume a base class we call

Animal properties: height width length
Methods: eat move reproduce

All animals will have these properties and methods. With inheritance you need not re write that code with each specific animal class. For example if I create a class called fish it will use the base animal class move method and I merely write additional code to handle the specifics of fish swimming. For a specific class horse I would still use the base class method move and simply add code to handle the specifics of running.

This very useful technique is not available in Visual Basic in a strict sense. However when we discuss any facet of software development we can take two approaches:

1. The strictly theoretical and academic approach
2. The practical approach.

If we take the second approach we are forced to ask the question: Why do I want inheritance? The answer is simple: code reusability.

If that is your goal with inheritance then let me suggest this. Create a base class with the methods and properties that will be commonly needed. Then *within* the other classes they can each create a private instance of that base class and access all of its methods and properties. In this way you now can inherit all the properties and methods of any class. In fact you create a private instance of that class that the inheriting class can utilize as needed.

The obvious disadvantage to this method is that it requires more memory. Rather than simply access certain parts of a base class you create an entire instance of a class. But if you consider that a class will probably take up less than 1K of memory, and that memory will be released as soon as the inheriting class is done with it, it is really not much of an obstacle.

Beyond the simple fact of giving Visual Basic programmers the powerful methodology of inheritance this method actually provides a more flexible implementation of inheritance than does C++. In C++ there are certain rules governing the implementation of base classes and how they can be inherited. With this methodology the programmer is free to use any given class as a base class for any other class and can in fact have as many classes as he might wish inherited in a single class. This means that not only does the Visual Basic programmer not only has access to the powerful tool of inheritance but he also has a very flexible way to use multiple inheritance.

Here is a sample illustrating this concept:

The Base Class

Option Explicit

Private m_object As Object

Public Property Let borderstyle(style As Integer)

```
m_object.borderstyle = style
End Property
```

```
Public Property Get borderstyle() As Integer
    borderstyle = m_object.borderstyle
End Property
```

```
Public Sub hide(thisobject As Object)
    thisobject.hide
End Sub
```

```
Public Sub move()
    With frmdemo.Image1
        .Left = .Left + 300
        .Top = .Top + 100
    End With
End Sub
```

```
Public Sub show(thisobject As Object)
    thisobject.show
End Sub
```

```
Public Property Set targetobject(target As Object)
    m_object = target
End Property
```

The child class (The one that uses the base classes properties)

```
Option Explicit
```

```
Dim cls_baseclass As New clsAnything
```

```
Public Property Get borderstyle() As Integer
    borderstyle = cls_baseclass.borderstyle
End Property
```

```
Public Sub move()
    cls_baseclass.move
```

End Sub

```
Private Sub Class_Initialize()  
Dim f As Form  
Set f = frmdemo  
Dim i As Object  
Set i = frmdemo.Image1
```

```
Set cls_baseclass.targetobject = f.i
```

End Sub

Now create a form called frmdemo with an image box and a command button. In the command button place this code:

```
Private Sub cmdmove_Click()  
Dim cls_localClass As New clsimage1  
cls_localClass.move  
End Sub
```

We now have one class using a method from another class. This is a trivial example but it illustrates the principle of using inheritance in Visual Basic. There are other techniques for simulating inheritance in Visual Basic, I chose this one for three important reasons:

1. Its easy to follow and understand
2. It illustrates the concept of inheritance
3. And last but not least, its my own invention!

Function Based Object Oriented Design for Visual Basic

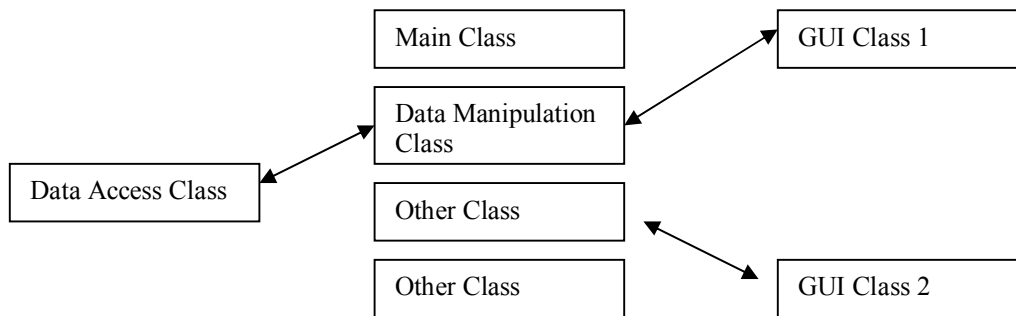
This is my own design methodology. You may not wish to follow it, but at least it should get you thinking about design methodologies.

Introduction

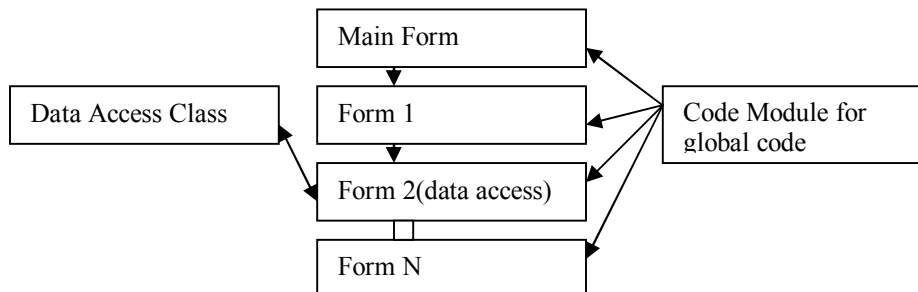
In the C programming language a program is essentially a series of functions connected together. When the C++ superset of the C language developed it carried this concept into object orientation. A C++ program is basically a conglomeration of classes. This even carried into Windows Programming where the various user interface elements are generated by certain classes.

This means that any C program (including windows Visual C++ programs) first starts from a design methodology emphasizing the functionality of the program and the graphical user interface is based from that.

Example of Visual C++ Structure for a simple database front end:



Visual Basic programs, however, are designed differently. With early versions of VB (2.0, 3.0), it was primarily used to design the GUI and most functionality was done in C/C++ DLL's. Because of this the design methodology is quite different. Most of the design and coding starts with the visual interface and the functionality is connected to that interface. As VB Maturated a tremendous amount of functionality was added to the programming language. There is very little that Visual C++ can do that Visual Basic 6.0 cannot. However the original design approach stayed. It is in fact the essential design approach used even in advanced object oriented visual basic programs. Below is an example of a typical Visual Basic database front end:



Now this design methodology has basic flaws:

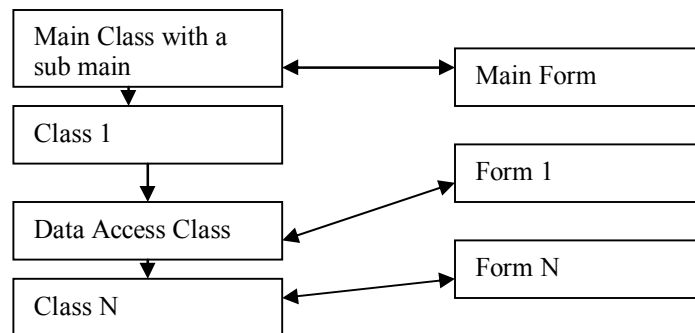
1. Since code can be initiated from any event in any control on any form, or in response to external triggers, it is very difficult to follow the flow of the program.
2. The design begins with the GUI, which, with a tool like VB, is a trivial part of the coding.
3. This design is not truly object oriented since the heart of the program is still forms and code modules, classes are simply used to encapsulate certain functions.

Frankly in many cases classes are used in Visual Basic programs simply so the developer can state that they have done object oriented programming. In those cases where the Classes are actually utilized it is usually in Active X programming to expose certain properties and methods. This, in my opinion, is not object oriented programming because it is not object oriented thinking and it is not object oriented design.

What I propose is a design methodology based on functionality of the code. The entire core functionality of the program can be designed independent of the GUI. This will accomplish the following goals:

1. Ease upgrading between versions. The hardest part of an upgrade is when your controls are changed. With this design the controls do NOT contain much code. Therefore if you wish to complete replace a control it is very easy.
2. The flow of the program is very easy to follow.
3. The programmer is required to think about the code functionality before considering the user interface.
4. The programmer is required to think in an object-oriented fashion, and therefore is more likely to actually utilize the full benefits of object orientation.

Here is an example:



The most important obstacle to utilizing this sort of coding method is that it requires a very close watch on the creation and destruction of instances of classes. It is vital that the programmer take into account that every reference to a class must be destroyed prior to that class actually being unloaded from memory.

However this also lends itself very well to multi-user multithreaded programming. Since multiple instance of the same class can be generated and each class will generate instances of the forms it needs to interact with the user, this framework is ready made for multiple process/thread applications. To implement such an approach you simply need to add the appropriate code into the main class to launch and monitor various threads.

Details

The conceptual foundation for this design methodology is quite simple; it rests upon three basic propositions:

1. Start the design process with the functional flow of the code and make the graphical user interface a secondary consideration.
2. Create smaller classes than are normally found in Visual Basic in order to make their scope more clearly defined.
3. Make the class the foundation unit of the application.

The first point is seemingly the simplest, but it is often not the way that Visual Basic programs are written. When Visual Basic was first introduced it was a major change in the way windows applications were developed. The original concept was to take the often-tedious task of creating the graphical user interface and to make it simple. Then rather than the code being procedural as in traditional languages (COBOL, Pascal, C, Quick Basic) the code was user event driven. The user determined the flow of the application by initiating the events in various controls.

Unfortunately this led to a trend wherein programmers concentrated on the user interface and their code was often tied to particular events. This can lead to a few issues:

- a. Following the flow of the code can be very difficult especially in large Visual Basic programs.
- b. Upgrading can be very difficult since much of the code is contained within controls that are being upgraded or even replaced.

With the advent of Visual Basic version 4.0 , VB developers could utilize classes and collections as well as other objects. Each successive version of Visual Basic brought more object oriented tools. Many purists criticized Visual Basic as not being truly object

oriented because it lacked inheritance and polymorphism (an accusation that is not entirely correct, see the paper Visual Basic Inheritance). However, in my view, this was not the reason that Visual Basic was not truly object oriented. The problem was that Visual Basic design strategies did not lend themselves to a truly object based applications. These paradigms also did not lead to programmers thinking in a truly object oriented fashion.

My concept is rather simple and is based in a large part on C++ object oriented designs. I also feel that, when applied properly, this methodology will definitely reduce memory leaks.

The first change is that all the functionality of a program is contained in classes. These classes should be smaller than you usually see in a Visual Basic program. In Visual Basic, classes are treated as a sort of super code module and often have extensive coding. I propose that a class be used as a small set of closely related functions. Each class should have a clearly definable purpose and the only entry and exit points to that class should be clearly defined public methods and properties. The rest of the code for the class should be private.

If a function requires a user interface then that class can launch whatever form or forms it might need.

Example:

```
Dim f as form
Set f = thisform
f.show
```

Now the advantage we have at this point is that any part of the graphical user interface is clearly tied too and dependent upon the underlying class.

There should of course be a single code module that contains the main subroutine and any global variables necessary. It will create instances of any classes it needs. There are two approaches to this.

- a. Have each class in turn create instances of other classes it might need. This creates a type of domino effect. The only problem is that if you are not careful with your code you could waste lots of memory.
- b. You can have the main code module launch each class instance as needed. This provides a centralized location to regulate all the objects in a project. This is the approach I would most recommend.

One of the variables I suggest you place in the main code module is a reference counter for each class you have. The counter should be incremented each time a class is created and decreased each time a class is destroyed.

When we turn our attention to the individual classes, we see how this methodology is truly object oriented in a way that Visual Basic has never been. A given class is the basis for all the functionality and the user interface for a particular process. Also in the

terminate event of a class you can ensure that all forms or other objects that class has created are destroyed. This will actually improve your ability to track memory leaks via objects. If each class cleans up its objects and the main code module cleans up the classes there is little chance of a memory leak.

We can also see that this architecture paradigm lends itself well to multithreaded applications. You see since the main code module can launch multiple instances of the same class, and each class can launch instances of whatever objects it might need, you have a ready made frame work for multithreaded applications.

Classes are very important for you . The only way you will ever be able to move on to advanced Visual Basic topics such as Active X controls, Active Server Pages, and IIS applications is by learning all you can about classes.

Collections:

Collections are basically object-oriented arrays. Visual Basic has several built in collections. The forms collection contains a reference to all the form in a project and the controls collection contains a reference to all the controls on a given form. For example, if you wanted to disable all the text boxes on a given form you could use the controls collection like this:

```
Private sub disable_text()  
  
Dim I as integer 'used as a counter  
Dim c as control 'use c as the generic control object  
  
For each c in controls  
  
    If typeof c is textbox then  
        c.enabled = false  
    end if  
  
next c
```

You can also create your own collections simply dimension a collection variable then add stuff to it:

```
Dim myCollection as new collection  
  
Now add stuff to your collection using the add method  
  
Mycollection.add somevariable
```

Now you may ask why use a collection instead of an array. Well that is a good question, the answer is that a collection is far more flexible. In an array all the elements must be of a single data type. You can have an array of integer, and array of strings, an array of doubles, but you cannot have a single array that contains more than one type of variable. A collection can contain diverse types of variables! Let's look at another example.

```
Dim empcollection as new collection
```

```
Let sName = "Employee Name"
```

```
Let iNum = 3848457
```

```
Let sPhone = "1-555-555-5555"
```

```
Empcollection.add sName
```

```
Empcollection.add iNum
```

```
Empcollection.add sPhone
```

This added flexibility makes the collection a far better alternative to the array. In the real world, it is often necessary to use several different data types. In my personal programming, I now only use arrays with controls, in code I use only collections.

Destroy Your Objects:

It is a good idea to destroy all objects before you exit a program. If you do not then they are still floating around in memory. For example if you used:

```
Dim myclass as new deckofcards
```

```
Then you must have
```

```
Set myclass = nothing
```

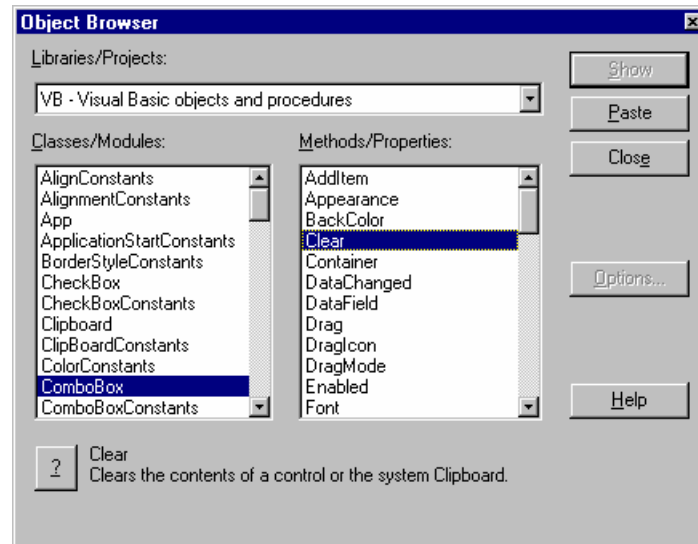
The same is true with collections. The nothing Keyword will cause the new instance of the object to be destroyed.



Tip: If you do not destroy your objects they may still be in memory!

Object Browser

Visual Basic also comes with an object browser that allows you to view any object (whether it is a Visual Basic standard object or one you created) and its properties. If you choose “View” from the drop down menu and then select “Object Browser” you will see the following:



You can select any object from the left-hand list box and its properties will appear in the right hand list box. If you then click on a particular property, the bottom of the screen will display a description of that property. This tool will be very helpful for you to understand objects.

Pointers in VB

Here is how:

Lets suppose that you have a list of employee data. Each entry needs lastname, first name, SSN, Job Title.

Create a class called clsEmployee that has each of these four variables (property get/let or just make them public variables) and a method called next.

Now for each entry you want to add : dim Employee1 as new clsEmployee, dim Employee2 as new clsEmployee. This will create a list of employee data. The way you use a pointer to work with the data is this:

Dim pointer as clsEmployee

Then any time you wish to point to an entry:

set pointer = EmployeeN. You see the set foo = NEW foo, the keyword new creates a new instance. Without that keyword you actually have a pointer to the object. It works in much the same way a C pointer does. Unfortunately only object variables can be pointed to in this fashion.

Now I would add a twist to this and store all the entries in a collection (classes can be added to collections). This way my entire list is in a single variable that can be passed to and from functions, through dll's and into classes.

UML

You will also find that there are specific design strategies for Object Oriented programming. One very popular one that I personally use is Unified Modeling Language. UML is essentially a methodology for creating a series of diagrams to plan out your project. Each diagram becomes successively more detailed. Below is a brief outline of the various UML diagrams. This is not meant to make you proficient at UML but to give you enough information so that you can decide whether or not you wish to invest the time to learn UML.

Unified Modeling language is a methodology of systematically planning large scale object oriented development. With this method you gradually proceed from very general user needs to very specific programming specifications. Each model either concentrates on a different view of the development process or takes the previous model and adds more detail.

The UML process is an iterative design process whereby you begin with user defined goals and business rules in the Use-Case Model And move down to actually programming specifications in Class Diagrams, Activity Diagrams, State Charts, and Deployment diagrams. This approach allows you to begin with general ideas of the desired function of the application and gradually develop specifications based on the desired functionality.

Most UML experts encourage iterative development to proceed concurrently with the latter stages of UML design. For example by the time you begin making class diagrams you can actually begin coding the frame work of your application. As the design becomes more detailed, so will the code.

Below are the various models used in UML.

Use-Case Models: This is a diagram showing the actual functionality of an application from a users perspective. The elements in this diagram are actors (anything which can act on the application) and the business rules that govern the application.

Interaction Diagrams: This is the next step in UML modeling. This shows This basically shows how the elements of this application interact. This will show the interaction of the application, user, database, etc.

Sequence Diagrams: This diagram basically takes the Interaction Diagram and begins to translate that into actual objects that correspond directly too programming objects.

Collaboration Diagrams: This diagram concentrates on the actual distribution of the objects involved.

Class Diagrams: This is where you take the data from previous diagrams and actually plan the classes you will use in your application.

Activity Diagrams: These diagrams show the actual flow and function of the application.

State Charts: This is a model of the various states the applications objects may be in.

Component Diagrams: These diagrams show the various components of s distributed application.

Deployment Diagrams: This model shows the actual physical location of various components.

The File System Object

Visual Basic has a number of built in objects, you will learn to use over time. One that is very useful is the file system object. It is actually a series of objects arranged in a hierarchy. This object hierarchy allows you to access any file, drive, or folder on your machine, quickly and easily. Lets walk through the process of accessing this object hierarchy and then use it.

Step 1: First you will have to go to project > references and choose “Microsoft Scripting Runtime” (This is the sccrun.dll).

Now we have to create an actual instance of the file system object, and from it we can then create and access each of the sub objects. Consider this example:

Example 5-2

First create a form with one command button and one text box on it. Also place a common dialog box on the form. If you don't remember the common dialog box then just go to project > components and select it. It does not matter where you place it on your form as it will not be visible at run time.

Now in the command button place this code:

```
CommonDialog1.filter = "Text Files (*.txt)*.txt | All Files (*.*)*.*"
```

```
CommonDialog1.action = 1
```

```
' file system object variables
```

```
Dim fs As FileSystemObject
```

```
Dim tStream As TextStream
```

```
Dim objFile As File
```

```
Dim strFileLine As String
```

```
On Error GoTo errorhandler
```

```
' Set the file system object
```

```
Set fs = CreateObject("Scripting.FileSystemObject")
```

```
Set objFile = fs.GetFile(commonDialog1.filename)
```

```
Set tStream = objFile.OpenAsTextStream(ForReading, TristateFalse)
```

```
Do Until tStream.AtEndOfStream
```

```
    strFileLine = tStream.ReadLine & " " & vbCrLf
```

```
    text1.text = text1.text & strFileLine
```

```
Loop
```

```
Exit Sub
```

```
errorhandler:
```

```
MsgBox (Err.Description)
```

```
End Sub
```

Now whatever file you point to with the common dialog (preferably a text file) you will open and place into the text box.

Conclusion:

This is by no means meant to be an exhaustive examination of the topic of object orientated programming. Entire books have been written on the topic. The purpose of this chapter is simply to let you know what object-oriented programming is and provide you with a start on some of its more basic uses.

Hopefully now you at least have a clue what object oriented programming is and can get started with a little bit. It is also vital that you have some object oriented programming before you attempt to create active X components. Probably the best book on object oriented programming is Peter Wrights "Beginning Objects" by Wrox Press. Another good source is any object orientation book written by Deborah Kurata.

Review Questions:

1. What does the “new” keyword do?
2. What is encapsulation?
3. What advantage does a collection have over an array?
4. Why should all variables in a class be declared private?
5. What are methods of a class?
6. What do you place in the terminate event of a class?
7. What does the “nothing” keyword do?
8. What is the difference between the “let” and “get” methods?
9. What does the get method do?
10. What is polymorphism?
11. Does visual basic support polymorphism?

Chapter Six

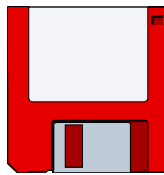
Printing and More

Chapter Objectives

- **Understand Printing in Visual Basic**
- **Understand the basic's of Crystal Reports**
- **Understand the Basics of Encryption**
- **Using other applications objects**

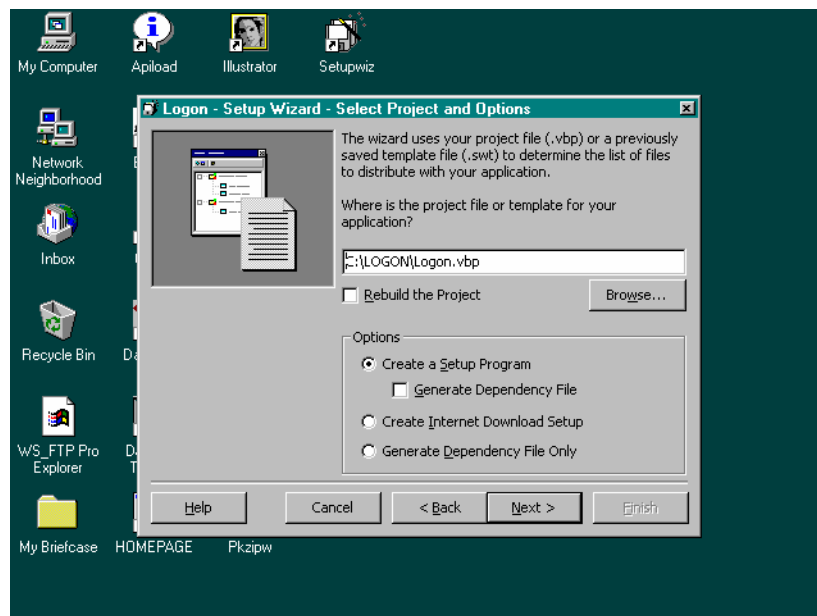
Chapter VI: Printing and More

By this point, you should be comfortable writing database applications, printing with Visual Basic and at least familiar with object oriented programming. But is this all? Is there no more to Visual Basic? Well actually, there is a lot more. The purpose of this chapter is to give you a taste, just a taste, mind you, of several different topics.



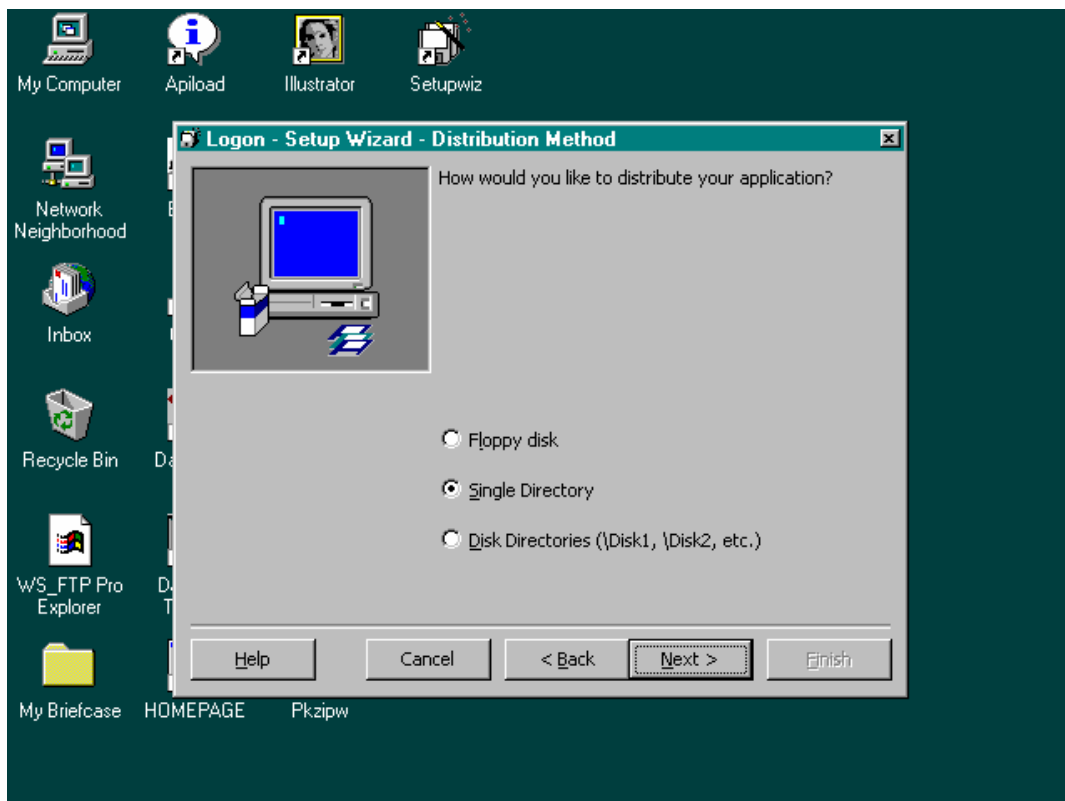
Application distribution:

So you have created a cool application. Now you want a setup utility to make distribution disks. Well Visual Basic Ships with a Setup Wizard. When you first click on the Setup Wizard. The first step will simply ask you to browse the hard drive and point to a program you wish to make a setup file for.

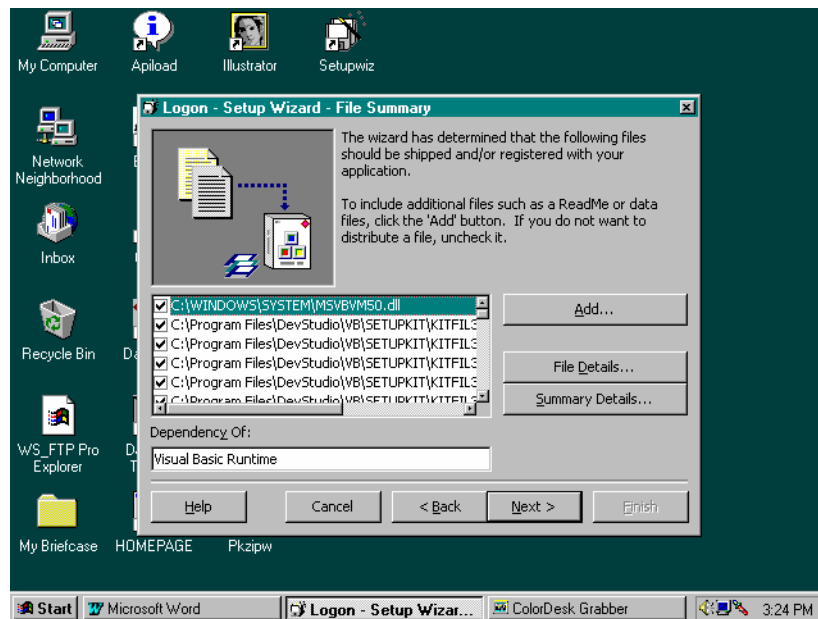


As you walk through the setup wizard you will be prompted for a little information, but it really does most of it for you. You may see information about funky things like dependency files, but at this stage in your VB programming I would not worry about it. You really just need to get your program out the door for now.

You will be prompted to tell the setup wizard whether you wish to make your installation file on a disk or in a folder on your PC:



About the 5th step, the setup wizard will work on its own for a little while running your program and trying to see what supporting files it will need (for your controls to work you need certain .OCX and .DLL files). Then you will be presented with a screen that will show you what files are included in your setup. There is a problem. The Setup Wizard will not pick up database files, flat files or crystal report files. You can add them at this step. Make sure you do or else your program won't work correctly.



It is important to make sure you include everything your user will need. Especially check for any crystal reports files (*.rpt), Access database files (*.mdb) and text files (*.txt).

It is a good idea to try to install your program on several machines. Make sure your install works fine.



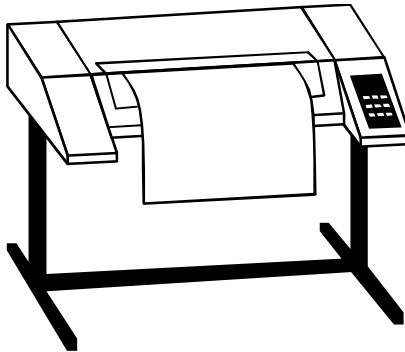
- If you have a data control pointing to a database, it may not work on a users machine. The reason for this is that the path may not be the same. For example if your database path on your development machine is: c:\project\database.mdb and your user installs on their machine, windows 95, by default will install the program to c:\program files\project. So your data control will generate an error. You could include in your documentation that the program must be installed to a particular directory or you could develop it in c:\program files\project on your own machine.

Another option is to go back to your project files and set the database name of the data control to nothing. Then in the forms load event have this code:

```
Data1.databasesname = app.path &"\mydatabase.mdb"
```

This tells the data control to look in the folder that it is in and search for a database called mydatabase.mdb.

Printing:



You will, of course need to print things. There are three methods to print. The first two I will briefly introduce and then ignore since the third method is absolutely fantastic for all printing needs.

PrintForm: This command caused the current form to be printed in its entirety. This is useful for capturing the look of a specific form.

Printer: The printer object can be used to print lines of code to the printer. You may need to manually set font, font size and other properties. The Printer object is one of those built in objects that Visual Basic has. In this case, it represents your default printer. You can set many printer options and settings via the printer object. Things Like print quality which is an integer value ranging from - 4 to -1 or an integer value corresponding to a specific dots per square inch (dpi) setting, as in these examples:

```
Printer.printquality = 300 '300 dpi  
Printer.printquality = -1
```

There are a number of printer object options you can utilize to print with:

<i>Option</i>	<i>Effect</i>	<i>Example</i>
<i>Killdoc</i>	Kills the current document being printed	Printer.killdoc
<i>Print</i>	Prints a string or a variable	Printer.print "This is cool"
<i>Enddoc</i>	Stop the current document from printing	Printer.enddoc
<i>Printquality</i>	Sets the quality of print	Printer.quality = 300
<i>Zoom</i>	Indicates "zooming" in and out of a document	Printer.zoom = 50 Note that these numbers are percentages of the total therefore 100 (i.e. 100%) is the default.

The -1 to -4 print quality values represent specific levels of printer quality:

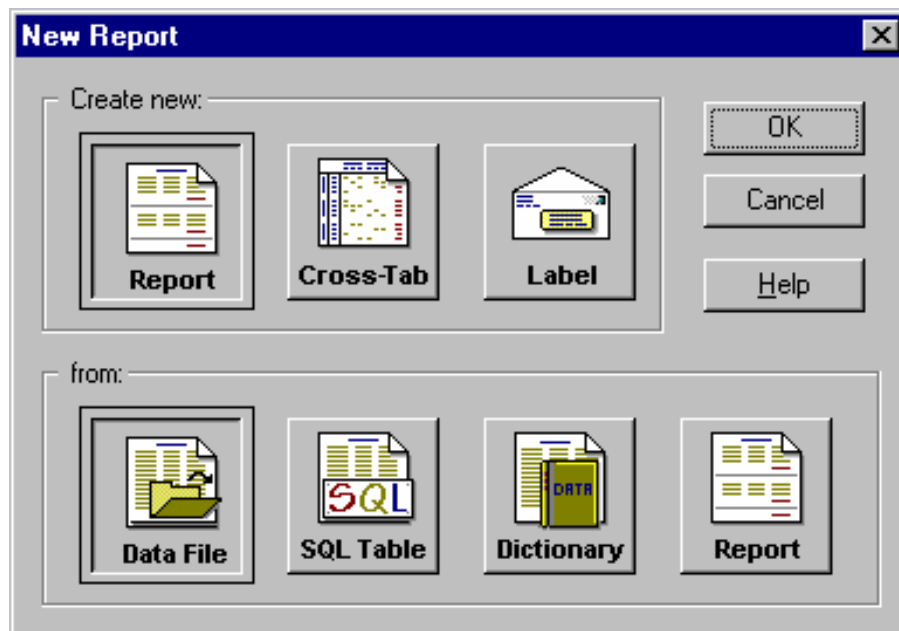
- 1 denotes Draft resolution
- 2 denotes Low Resolution
- 3 denotes Medium resolution
- 4 denotes High Resolution

I definitely recommend this over using the specific DPI designation since you do not know the resolution capabilities of the printer your application may need to print to.

Crystal Reports:

Visual Basic versions prior to version 6 shipped with a very powerful tool called Crystal Reports. With Crystal Reports you can easily place data fields from a database onto a form, alter fonts, include formulas, and then print. It allows you to add in full-featured windows printing to your programs. Even though it no longer ships with Visual Basic, many VB programmers still use this tool.

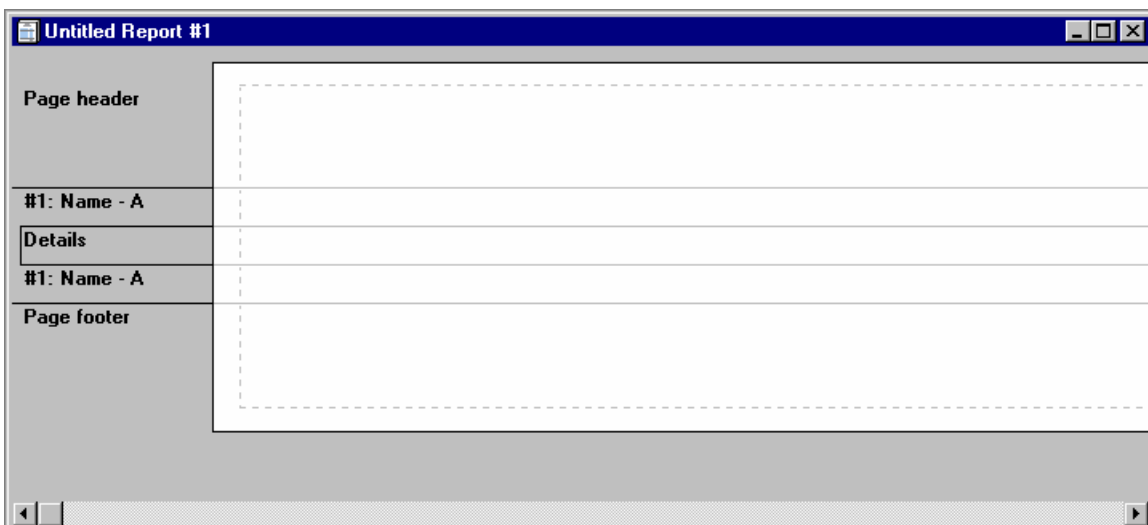
Now when you first open Crystal Reports you will be asked to register the software, whether or not you do this is up to you. Once you are past the registration screen, you can select various options from the drop down menu's. Under "File" you can select "New", and you should then select the first or "standard" option. In earlier versions of Crystal reports, you saw a similar window only you would select the first button, which was captions "Report"



Next you are presented with a screen inquiring as to what data you wish to use for this report. If you are using an access database, simply select the datafile button. You will then see a standard windows 95 dialogue box. With this, you can select any database on your computer. When you have completed this process, click the “done” button.

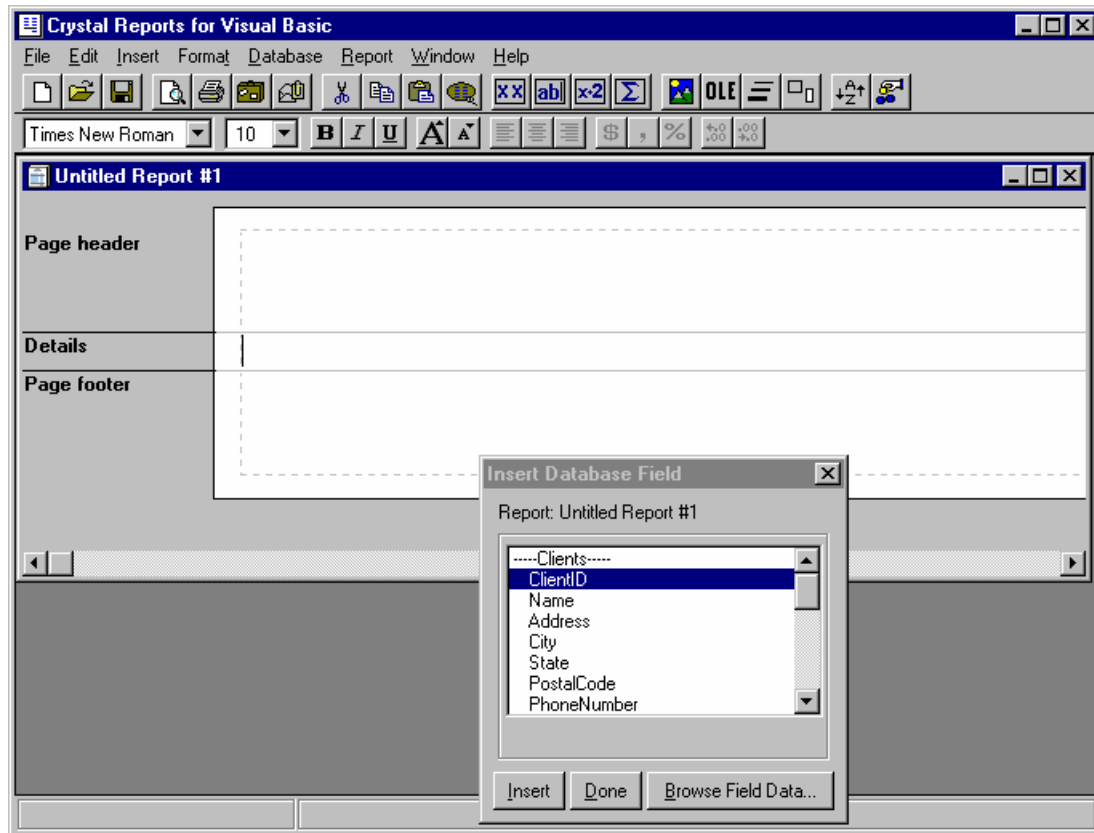
The only other tab we are interested in, for our purposes, is the fields tab. Which fields would you like on your report? Once you have selected that, you can choose to preview the report. It will show you what the standard report will look like. If you are not happy with the layout, you can then use the design tab to move any fields to any place you wish, just drag and drop. You can also change texts and fonts as you would in any word processor.

A basic Crystal Report looks like this:



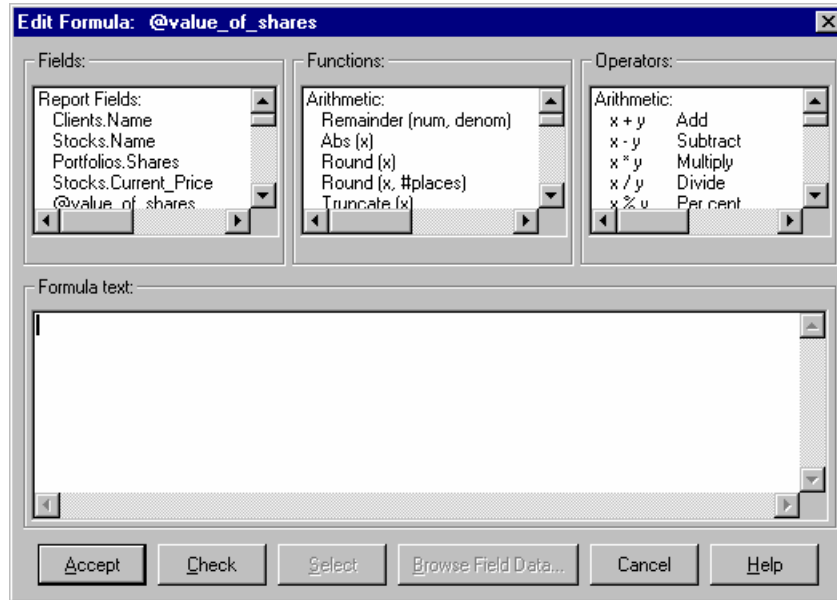
The Page header contains those items you wish at the top of the report, often things such as the title, print date, etc. The page footer contains items to be printed at the end of the report such as totals and statistics. Most data fields are placed in the details section.

You can also go back to the “insert” option and select “database field” then add in more database fields. You will be presented with a screen much like the one below:



You may also wish to sort which records to display based on whatever filter criteria you can devise. Select “Report” and then select “Record Selections Expert”. This will walk you through the process of setting up selection criteria. This allows you to make printable reports for a wide variety of purposes. Examples would include printing all employees with a hire date this year, all employees with a salary over 30,000, etc.

Formulas will also greatly enhance your reports. If you select “Insert” and “formula” you can then add in formulas. Things like the sum of a certain field, the standard deviation, average, etc.



The fields list box on the left displays all the fields in your report. If you double click on one, it will then be copied to the formula text. You can then double click on any operation or function and it will be placed in the formula text. A formula might look like:

Sum ({Temptable.Retail Price})

This tells Crystal Reports to sum up all the items in the “retail price” field of the “temptable”

This is just an introduction to Crystal Reports. I suggest you play with it for a little bit and see how you can manipulate the various options. But the big question is how do you get this thing to print from your Visual Basic Program. There is a control you can place on your form called the crystal control. This control will not be visible at run time. The properties you will need to set up are:

Reportfilename : This property is like the databasename in the datacontrol. It points to the report you wish to use.

Destination: This tells the control whether to print directly to the printer or to the window first.

Now wherever you want to start the printing from (say a cmd_save button) place code such as: `crystalreport1.action = 1` and viola! You are printing.

Now this is by no means an exhaustive look at crystal reports but hopefully it will show you the essentials. Hopefully you will find this useful. As you explore Crystal Reports more thoroughly, you will find it to be the most robust printing and reporting tool you have ever seen.

Other Options in VB:

There is a lot of stuff in the Visual Basic IDE you can use. Let's look at just a few items here and their uses:

Under the drop down menu "Tools" you have an item called "options". This has a lot of neat things in it you will want to use such as:

Require Variable Declaration: This option means that you must declare each and every variable before you can use it. If this option is not selected then anytime you write a word VB does not recognize it will just assume this is a new variable. ALWAYS have this option checked!

Auto Syntax Checking: If this is checked then Visual Basic will check your syntax as you go. This is very useful, I usually have it checked.

Default to Full Module View: This determines whether or not all your subroutines appear as a continuous bunch of code with lines separating each one, or if each subroutine, event, and function appears in a separate window.

Auto Data Tips: This is very helpful as VB will try to guess what kind of code your writing and will prompt you with the most likely next word in code.

Under the drop down menu "Project" You will see the option "Properties". This lets you set a number of properties for the current project including version number and how to compile the program (optimize for speed, optimize for speed).

Communications in Visual Basic:



By using the MSComm control your application can communicate over the phone line:
First, you need to set the properties of your MSComm control. You may wish to do this at design time or at run time using user input. The properties you will need to set are:

Comport: What port do you need to use?
Settings: baud rate, parity, data bits, stop bits

Here is a practical example:

```
Mscomm1.commport =1  
Mscomm1.commport = "28800,N,8,1"
```

This is for a modem on comma port 1 running at 28.8 baud, with no parity, 8 data bits, and 1 stop bit.

Now that you have set a comm port number and its settings, we have to open that port and set up an initializations string:

```
Mscomm1.portopen = true
```

```
Mscomm1.output = "AT" & chr(13)
```

Do

Do events

```
Loop until mscomm1.inbuffer count >= 2
```

‘This code basically says to hold your horses until we get a response from the modem.

Now you can send stuff over the modem using the output command!

<i>Properties</i>	<i>Description</i>
<i>comport</i>	Sets and returns a comm port number
<i>Settings</i>	Sets and Returns the baud rate, parity, data bits, and stop bits as a string.
<i>Port Open</i>	Sets and returns the state of a comm port i.e. <code>commport.open = true</code>
<i>Input</i>	Returns as a string the stuff received through the comm port
<i>Output</i>	Sends an output string through the comm port.



Encryption:

Encryption is an exciting word in programming, and also a daunting one. It probably brings up visions of complex mathematics and machine level programming. Encryption and Cryptography have long been important topics, especially in this day of increasing computer security concerns.

During World War II Mathematician Alan Turing developed a method to decrypt Nazi military messages. He employed the aid of dozens of women in the Woman's Royal Naval Service (WREN) to each perform a very simple task, repeated many times on each shift around the clock. Each day these women were given a template that contained a word or phrase that was likely to be embedded within a German message. These women basically worked assembly line style to decrypt messages.

Computer programs that perform encryption for you have supplanted this kind of assembly line work. But Alan Turing's work with mathematical concepts and encryption has earned him a valued place in the history of encryption.

A number of methods have been developed through the years to encrypt a message. We will look at a practical method to do this in Visual Basic

Now with that background of encryption and decryption lets look at some visual basic techniques you can use.

Let's say we have some message we wish to encrypt. We have placed this message in a variable we will call sMessage

```
Private sub encrypt()  
  
Dim I as integer 'counter  
Dim x as integer 'used in encryption  
  
Smessage = txtinput  
  
For I = 1 to Len(sMessage)  
  
    Z = Mid(sMessage,I,1)  
    Z = Asc ( Z)  
    Z = z - 1  
    Txtoutput = Chr(z)  
Next I  
  
End Sub
```

Now what this sub routine is doing is taking your message grabbing on character of it changing it to ASCII format (with the Asc() function) subtracting 1 from that ASCII code and then changing it back to a character and displaying the new message.

To decrypt messages you would simply reverse this process. This is basically an elementary substitution code. It will work fine for most basic encryption needs, but don't think that this kind of encryption will foil the CIA for more than a minute or two! But this gives you an idea of how computer encryption works. You create an encryption key, which you provide to those people you want to be able to decrypt your stuff. You could easily write a simple program with a form that allows a person to input a message and then click on either an "encryption" or "decryption" button.

You could complicate your encryption in many ways. You could change the increment step (z -1) to something more complex. You could make your output backwards (Last character first) . There are a vast number of methods you could use to make your own message encryption far more effective.

Advanced Encryption Concepts

Encryption is basically some form of substitution. For example if I use the number 1 in place of a and 2 in place of B and 3 in place of C, etc I am using the simplest form of encryption. A more complex sequence might be to take the number corresponding to the place in the alphabet (such as a =1) and then perform some mathematical operation on it. I might add 2 and divide by 3. Take the arc cosine of the number. No matter what the process the idea is to give some new symbol for the letter. The problem with basic encryption such as this, is that it still leaves a back door to crack the code. That back door is frequency. Basically certain letters appear in the English language with certain frequency. For example the phrase “a dog eats the bone” . If I use the basic encryption (a =1, b =2,etc.) I get 1- 4 -15 -7 -5- 1 -20 -19 -20 -8- 5- 2 -15 -14- 5, The vowels a and e are most likely giving my code away do to their frequency. How can we prevent this from happening?? Well the answer is multiple encryption alphabets. Let me show you:

A	B	C	D	E	F
1	2	3	4	5	6
4	5	7	R	T	y
Z	X	4	3	2	W

Now if I attempt to encrypt the phrase “A Cab” using just a simple substitution with the first row I get:

1 3-1-2

What multiple alphabet substitution says, is that I will use the first substitution alphabet for the first letter, the second alphabet for the second letter, and so forth. Thus “A Cab” becomes:

1 7-Z-2

Now there is no frequency for any decoder to pick up on. You can utilize this with as many substitution alphabets as you like, but I think 5 should be sufficient to stop most snoopers.

My Encryption Algorithm for cryptography beginners

Encryption is simply a method of changing a text via some predefined algorithm so that its content is no longer readily apparent . ASCII codes could be considered a simple type of encryption whereby each character can be represented by either a hexadecimal or decimal number. However since ASCII codes are so commonly known, it would not be a good choice to use that code.

One of the simplest types of encryption known is the substitution alphabet. In this encryption method you set up a predefined simple algorithm to change each letter of a

word. For example you might use the method of adding 2 to the numeric value of every letter. For example the phrase

A CAT (letter 1, space, letter 3, letter 1, letter 19) becomes:

C ECV

The problem with this method is that certain letters and phrases appear with certain frequency in any given language. It would not take a particularly insightful cryptologist to deduce that the letter C is actually A and in short time he would have your message completely decoded.

A variation on this theme, which is used widely even by the National Security Agency, is the multi substitution alphabet. In this scheme you set up a series of alphabet shifts. For example you might have three alphabets the first being +1, the second -1, the third + 3. The first letter is adjusted according to the first encryption code, the second letter by the second code, etc. When you reach your last substitution alphabet you start over. In this case the above phrase:

A CAT encodes to

B B DU

This means that there will be no regular frequency of letters since a letter may be encrypted to differing values at different points in the message. Utilizing 5 or more alphabets makes a message virtually unbreakable.

Now usually the alphabets consist of shifting a letter forward or backwards a given number of spaces. However you can use any mathematical function you like. You could square the numeric value of the letter, take the square root of its negative (thus generating a complex number), or any other function you might care to use.

My Cryptography Algorithm for more serious cryptographers

My encryption algorithm is actually simply a variation on the traditional multi-alphabet encryption. What I do is simply use the following steps:

1. Convert the character to its ASCII equivalent.
2. Adjust that value according to the first substitution alphabet.
3. Convert that value to hexadecimal.
4. Adjust that value according to the first substitution alphabet.
5. Repeat the process for each character.

I also encrypt the key and place a value inside the encrypted message denoting what key will match it. This last part makes having an exact key essential to decoding the message.

Using Office Object Models to write integrated applications

Object Models

Object oriented programming has been one of the hottest buzzwords in programming for the past 4 or 5 years. Unfortunately most programmers are not sure why its such a cool thing, they just want it on their resume. Perhaps the best example of why object orientation is such a powerful technology can be seen by examining object models. Object models are simply a hierarchical representation of the objects that are exposed by a given application. All Office products, Internet Explorer, many Back Office Products, and Visual Basic all have an object model that you can utilize to manipulate that application in any way you wish to.

For example purposes we will examine the Outlook object model.

Outlooks Object Model

The last page of this document has a chart describing the Outlook object model. We are going to address some of the most important objects and a few of the things you can do with them.

Application: This object represents Outlook itself. With this object you can access and gain control of a currently running instance of Outlook, or you can launch an instance of Outlook.

Namespace: This object represents the current MAPI session. This object will allow you to manipulate, start, or end a MAPI session.

Explorer: This object represents the current window.

Folders: This collection represents all of the folders in Outlook.

Folder: This object represents the currently selected folder

Items: This object represents all the items in a folder

Item: This object represents a specific item in a folder

Sample Code

Sample Code that utilizes the Outlook object model. This code is simply to give you an idea of what you can do with this particular object model.

```
Private Sub Outlook_Object_Model_Demo()  
Dim objApp As Outlook.Application  
Dim objNameSpace As Outlook.NameSpace  
Dim objFolders As Outlook.Folders
```

```
Dim objExplorer As Outlook.Explorer
Dim objItem As Outlook.MailItem
Dim myItem(10) As Outlook.MailItem
Dim objItems As Outlook.Items
Dim objFolder As Outlook.MAPIFolder
```

```
Dim icount As Integer
```

```
' Set your application object equal to the current instance of
' Outlook.
```

```
Set objApp = GetObject("", "Outlook.Application")
```

```
' Set your namespace object equal to the mapi session
```

```
Set objNameSpace = objApp.GetNamespace("MAPI")
```

```
' Set the explorer object to the active view
```

```
Set objExplorer = objApp.ActiveExplorer
```

```
' Get the collection of all folders
```

```
Set objFolders = objNameSpace.Folders
```

```
' Set your folder object to the current folder
```

```
Set objFolder = objApp.ActiveExplorer
```

```
' set your items object equal to the collection of items in the current
```

```
' folder
```

```
Set objItems = objFolder.Items
```

```
For Each objItem In objItems
```

```
    Set myItem(icount) = objItem
```

```
    count = icount + 1
```

```
Next objItem
```

```
' some of the things you can access via the items object
```

```
objItem.Attachments
```

```
objItem.BCC
```

```
objItem.CC
```

```
objItem.CreationTime
```

```
objItem.MessageClass
```

```
objItem.Subject
```

```
objItem.Size
```

```
objItem.Move (destinationfolder)
```

```
' some of the things you can access via the folder object
```

```
objFolder.MoveTo (someotherfolder)
```

objFolder.Items

' some of the things the explorer object can provide you with

objExplorer.Caption
objExplorer.CurrentFolder
objExplorer.CommandBars

' Some of the things the namespace object can provide you with

objNameSpace.AddressLists
objNameSpace.CreateRecipient
objNameSpace.GetDefaultFolder
objNameSpace.Logoff

' some things that the application objec can provide you with

objApp.COMAddIns
objApp.CreateObject (objectsname)
objApp.CreateItem (typeofitem)
objApp.LanguageSettings
objApp.Quit

End Sub

All the office applications have objects representing their various items and functions. The coding can be a little bit tricky and its important to use the object hierarchy in the proper fashion. I will use the outlook model as an example but the same principles can be extrapolated to the rest of Office. To do any of this, howeve, you must manually set a reference to the office objects dll

At the top level we have the application object. This represents the outlook application. You can create a new instance of Outlook using the create object method

```
Set outApp = CreateObject("Outlook.Application")
```

You can access a currently running instance of outlook with

```
Set outApp = GetObject("", "Outlook.Application")
```

once you have the application object set you then will get the namespace object. That object represents the actual folders contained in Outlook. It has default properties for accessing any default folders. However you need to do a little more to access custom folders.

```
Set outNameSpace = outApp.GetNamespace("MAPI")
```

The next object in the hierarchy is the explorer object. This represents the window that is currently open

```
Set outExplorer = outApp.ActiveExplorer
```

The current folder is represented by:

```
Set outview = outApp.ActiveExplorer.CurrentFolder
```

you have a fairly large array of other objects you can use such as the
item, object that represents an individual item
inspector object, that represents a form or property page in outlook
command bars, that represents the tool bar in outlook
exception, represents errors

These are just a few of the objects you have access too. Each object has a host of methods and properties. Essentially, if you familiarize yourself with outlooks object model, you can do anything in your code that Outlook might do. You can take complete control of outlook

Review Questions:

1. What is the code to cause the crystal reports control to print.
2. What property do you set in the crystal control to point to a particular database?
3. What drop down menu option do you select to add in a new database field?
4. What drop down menu option do you select to filter records?
5. What is the command to print the current form.
6. Why might you not want to print with the printer object.
7. What is the Setup Wizard
8. What kind of files does the setup wizard not automatically add in for you?
9. What is DDE?
10. Name the types of DDE Links.
11. What do you call the application that provides the information in a DDE Link?
12. What do you call the transfer of data via DDE?
13. How do I set the comm port for an Ms Comm control?

Chapter Project 1:

Return to your rolodex project and add in an option to print lists of people. Do this using the printer object.

Chapter Project 2:

Return to your rolodex project and add in an option to print all people in your rolodex. Use Crystal reports and the crystal control. Also, make an executable of your program with installation disks prepared by the Setup Wizard.

Chapter Project #3:

Write a simple encryption/decryption program. This program should have the following features:

- It reads flat text files into a text box and then either encrypts or decrypts them
- It uses a common dialog box to find these flat files

Hint: You can even give your flat files a unique file extension (i.e. other than *.txt)

Chapter VII

ADO

Chapter VII Active Data Objects

Chapter Objectives

- To understand the basics of ADO
- To get familiar with the ADO Object Model
- To be able to access a database using ADO

ADO Basics

Active Data Objects are the latest method for access databases. You can think of ADO as the Active X version of DAO (Data Access Objects). An object hierarchy as shown in the table below defines ADO.

ActiveX Data Objects (ADO) are an easy-to-use yet extensible technology for adding database access to your Web pages. You can use ADO to write compact and scaleable scripts for connecting to OLE DB compliant data sources, such as databases, spreadsheets, sequential data files, or e-mail directories. OLE DB is a system-level programming interface that provides standard set of COM interfaces for exposing database management system functionality. With ADO's object model you can easily access these interfaces (using scripting languages, such as VBScript or JScript) to add database functionality to your Web applications. In addition, you can also use ADO to access Open Database Connectivity (ODBC) compliant databases.

It is important to be familiar with the various objects and what they are for before we proceed.

ADO Object Summary

Object	Description
Command	A specific command you execute against a data source.
Connection	Represents an open connection to a data source.
Data Control	Binds a data query Record set to one or more controls to display the Record set data on a Web page.
Data Factory	Implements methods that provide read/write data access to specified data sources for client-side applications.
Data Space	Creates client-side proxies to custom business objects located on the middle tier.

Error	Contains details about data access errors that pertain to a single operation involving the provider.
Field	Represents a column of data with a common data type.
Parameter	Represents a parameter or argument associated with a Command object. stored procedure.
Property	Represents a dynamic characteristic of an ADO object that is defined by the provider.
Record	Represents a row of a Record set, or a directory or file in a file system.
Record Set	Represents the entire set of records from a base table or the results of an executed command. At any time, the Record set object refers to only a single record within the set as the current record.
Stream	Represents a binary stream of data.

Perhaps the most often used, and hence the most important ADO objects are the connection, command, recordset, and record objects. Frankly you can work with ADO for quite a while and only use connection and recordset.

ADO Methods

Add New Creates a new record for an updatable Recordset object

Append Appends an object to a collection. If the collection is Fields, a new Field object may be created before it is appended to the collection.

Append Chuck Appends data to a large text or binary data Field, or to a Parameter object.

BeginTrans Begins a new transaction.

CommitTrans Saves any changes and ends the current transaction. It may also start a new transaction

RollbackTrans Cancels any changes and ends the current transaction. It may also start a new transaction.

Cancel Cancels execution of a pending, asynchronous method call..

Cancel Batch Cancels a pending batch update.

Cancel Update Cancels any changes made to the current or new row of a Recordset object, or the Fields collection of a Record object, before calling the Update method.

Clear Removes all the Error objects from the Errors collection.

Clone Creates a duplicate Record set object from an existing Record set object. Optionally, specifies that the clone be read-only.

Close Closes an open object and any dependent objects

CopyRecord Copies a file or directory, and its contents, to another location.

CopyTo Copies the specified number of characters or bytes (depending on Type) in the Stream to another Stream object.

CreateRecordset Creates an empty, disconnected Recordset.

Delete Deletes an object from the Parameters collection.

DeleteRecord Deletes a file or directory, and all its sub directories.

Execute Executes the query, SQL statement, or stored procedure specified in the Command Text property. It can also be used to execute a specified query, SQL statement, or stored procedure,.

Find Searches a Recordset for the row that satisfies the specified criteria

GetChunk Returns all, or a portion of, the contents of a large text or binary data Field object.

GetRows Retrieves multiple records of a Recordset object into an array.

GetString Returns the Recordset as a string

Move Moves the position of the current record in a Recordset object.

MoveFirst ,MovePrevious, MoveNext, and MoveLast Moves to the first, last, next, or previous record in a specified Recordset object and makes that record the current record.

MoveRecord Moves a file, or a directory and its contents, to another location.

Open Opens a connection to a data source.

Refresh Updates the objects in a collection to reflect objects available from, and specific to, the provider.

Requery Updates the data in a Recordset object by re-executing the query on which the object is based.

Save the Recordset in a object.

Seek Searches the index of a Recordset to quickly locate the row that matches the specified values, and changes the current row position to that row.

Now will all that out of the way, most of which is probably just confusing to you at this point, the real question is: How do I connect to a database with ADO?

Well you have too choices. You can do this via code alone, or with an ADO data control. Lets do it the hard way (but also the more flexible way) first, lets do it in just code!

ADO Code:

The first thing you need is a connection object. This makes good sense because you can't do much with a database if you can't connect to it. Here is an example:

```
Dim m_Connect as adodb.connection  
Set m_connect = New adodb.connection
```

NOTE: You can condense this into one line of code like this:

```
Dim m_connect as new adodb.connection
```

```
M_connect.connectionstring = "File Name c:\myfolder\mydata.mdb"
```

```
M_connect.open
```

Now the first part is simple, you are creating a new connection object. The second part is a bit trickier, but you are simply pointing that connection object to some data source. The cool thing about ADO is that you can point to ANY ODBC or OLEDB compliant data source, or even to a flat file!

Now that you have connected lets talk about the command object. This is the object that tells you connection object to do something. Here is an example:

```
Dim m_command as new adodb.command
```

```
M_command.commandtype = adCmdText
```

```
M_command.commandtext = s_sql ' this is just a string with some sql statement in it.
```

```
M_command.ActiveConnection = m_connect
```

```
M_command.execute
```

Again the first line is pretty straight forward. You simply create a new connection object. The second line simply states that you are using the command type of command text. There is a full table of command types below:

<i>Command Type</i>	<i>Used for</i>
AdCmdFile	This is for using a file as you command object.
AdCmdStoredProc	This type is used to invoke a stored procedure such as in an SQL Server or Oracle database.
AdCmdTable	This type is used to connect directly to a database table.
AdCmdText	This type is used with SQL commands that are given via a string.
AdCmdUnknown	This type is used , as the name suggests, when you are not sure

The next line of code simply assigns this command to some particular connection object. This is necessary because you might have any number of connection objects.

The last line of code simply causes the command to execute.

Now we can move our discussion on to the Record set object. This is very much like the Record set object we say with DAO. Here lets look at an example:

```
Dim m_record as new adodb.recordset
```

```
M_record.source = "SELECT * FROM TABLE"
```

```
M_record.open
```

This record set has all the old familiar properties that the DAO record set had including addnew, delete, move next, etc.

ADO Code Module

An obvious way to do ADO data connection is to put all your data access code into a single module. Here is an example of all the module in a code needed to connect to the "titles" table of the "biblio" database that ships with Visual Basic. In this case the code has been incorporated in the form. You can follow this practice or move all the data access code to a separate code module.

Option Explicit

```
Dim WithEvents adoPrimaryRS As Recordset
```

```
Dim mbChangedByCode As Boolean
```

```
Dim mvBookMark As Variant
```

```
Dim mbEditFlag As Boolean
```

```
Dim mbAddNewFlag As Boolean
```

```
Dim mbDataChanged As Boolean
```

```
Private Sub Form_Load()
```

```
    Dim db As Connection
```

```
    Set db = New Connection
```

```
    db.CursorLocation = adUseClient
```

```
    db.Open "PROVIDER=Microsoft.Jet.OLEDB.3.51;Data Source=C:\Program  
Files\Microsoft Visual Studio\VB98\Biblio.mdb;"
```

```
    Set adoPrimaryRS = New Recordset
```

```
    adoPrimaryRS.Open "select
```

```
Comments,Description,ISBN,Notes,PubID,Subject,Title,[Year Published] from Titles",  
db, adOpenStatic, adLockOptimistic
```

```
    Dim oText As TextBox
```

```
    'Bind the text boxes to the data provider
```

```
    For Each oText In Me.txtFields
```

```
        Set oText.DataSource = adoPrimaryRS
```

```
    Next
```

```
    mbDataChanged = False
```

```
End Sub
```

```

Private Sub Form_Resize()
    On Error Resume Next
    lblStatus.Width = Me.Width - 1500
    cmdNext.Left = lblStatus.Width + 700
    cmdLast.Left = cmdNext.Left + 340
End Sub

```

```

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    If mbEditFlag Or mbAddNewFlag Then Exit Sub

```

```

    Select Case KeyCode
        Case vbKeyEscape
            cmdClose_Click
        Case vbKeyEnd
            cmdLast_Click
        Case vbKeyHome
            cmdFirst_Click
        Case vbKeyUp, vbKeyPageUp
            If Shift = vbCtrlMask Then
                cmdFirst_Click
            Else
                cmdPrevious_Click
            End If
        Case vbKeyDown, vbKeyPageDown
            If Shift = vbCtrlMask Then
                cmdLast_Click
            Else
                cmdNext_Click
            End If
    End Select
End Sub

```

```

Private Sub Form_Unload(Cancel As Integer)
    Screen.MousePointer = vbDefault
End Sub

```

```

Private Sub adoPrimaryRS_MoveComplete(ByVal adReason As
ADODB.EventReasonEnum, ByVal pError As ADODB.Error, adStatus As
ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
    'This will display the current record position for this recordset
    lblStatus.Caption = "Record: " & CStr(adoPrimaryRS.AbsolutePosition)
End Sub

```

```
Private Sub adoPrimaryRS_WillChangeRecord(ByVal adReason As
ADODB.EventReasonEnum, ByVal cRecords As Long, adStatus As
ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
```

```
'This is where you put validation code
```

```
'This event gets called when the following actions occur
```

```
Dim bCancel As Boolean
```

```
Select Case adReason
Case adRsnAddNew
Case adRsnClose
Case adRsnDelete
Case adRsnFirstChange
Case adRsnMove
Case adRsnRequery
Case adRsnResynch
Case adRsnUndoAddNew
Case adRsnUndoDelete
Case adRsnUndoUpdate
Case adRsnUpdate
End Select
```

```
If bCancel Then adStatus = adStatusCancel
```

```
End Sub
```

```
Private Sub cmdAdd_Click()
```

```
On Error GoTo AddErr
```

```
With adoPrimaryRS
```

```
    If Not (.BOF And .EOF) Then
```

```
        mvBookmark = .Bookmark
```

```
    End If
```

```
    .AddNew
```

```
    lblStatus.Caption = "Add record"
```

```
    mbAddNewFlag = True
```

```
    SetButtons False
```

```
End With
```

```
Exit Sub
```

```
AddErr:
```

```
    MsgBox Err.Description
```

```
End Sub
```

```
Private Sub cmdDelete_Click()
```

```
On Error GoTo DeleteErr
```

```
With adoPrimaryRS
```

```
    .Delete
```

```
    .MoveNext
```

```

    If .EOF Then .MoveLast
End With
Exit Sub
DeleteErr:
    MsgBox Err.Description
End Sub

Private Sub cmdRefresh_Click()
    'This is only needed for multi user apps
    On Error GoTo RefreshErr
    adoPrimaryRS.Requery
    Exit Sub
RefreshErr:
    MsgBox Err.Description
End Sub

Private Sub cmdEdit_Click()
    On Error GoTo EditErr

    lblStatus.Caption = "Edit record"
    mbEditFlag = True
    SetButtons False
    Exit Sub

EditErr:
    MsgBox Err.Description
End Sub
Private Sub cmdCancel_Click()
    On Error Resume Next

    SetButtons True
    mbEditFlag = False
    mbAddNewFlag = False
    adoPrimaryRS.CancelUpdate
    If mvBookMark > 0 Then
        adoPrimaryRS.Bookmark = mvBookMark
    Else
        adoPrimaryRS.MoveFirst
    End If
    mbDataChanged = False

End Sub

Private Sub cmdUpdate_Click()
    On Error GoTo UpdateErr

```



```

adoPrimaryRS.UpdateBatch adAffectAll

If mbAddNewFlag Then
    adoPrimaryRS.MoveLast      'move to the new record
End If

mbEditFlag = False
mbAddNewFlag = False
SetButtons True
mbDataChanged = False

Exit Sub
UpdateErr:
    MsgBox Err.Description
End Sub

Private Sub cmdClose_Click()
    Unload Me
End Sub

Private Sub cmdFirst_Click()
    On Error GoTo GoFirstError

    adoPrimaryRS.MoveFirst
    mbDataChanged = False

Exit Sub

GoFirstError:
    MsgBox Err.Description
End Sub

Private Sub cmdLast_Click()
    On Error GoTo GoLastError

    adoPrimaryRS.MoveLast
    mbDataChanged = False

Exit Sub

GoLastError:
    MsgBox Err.Description
End Sub

Private Sub cmdNext_Click()
    On Error GoTo GoNextError

```

```

If Not adoPrimaryRS.EOF Then adoPrimaryRS.MoveNext
If adoPrimaryRS.EOF And adoPrimaryRS.RecordCount > 0 Then
    Beep
    'moved off the end so go back
    adoPrimaryRS.MoveLast
End If
'show the current record
mbDataChanged = False

Exit Sub
GoNextError:
    MsgBox Err.Description
End Sub

Private Sub cmdPrevious_Click()
    On Error GoTo GoPrevError

    If Not adoPrimaryRS.BOF Then adoPrimaryRS.MovePrevious
    If adoPrimaryRS.BOF And adoPrimaryRS.RecordCount > 0 Then
        Beep
        'moved off the end so go back
        adoPrimaryRS.MoveFirst
    End If
    'show the current record
    mbDataChanged = False

    Exit Sub

GoPrevError:
    MsgBox Err.Description
End Sub

Private Sub SetButtons(bVal As Boolean)
    cmdAdd.Visible = bVal
    cmdEdit.Visible = bVal
    cmdUpdate.Visible = Not bVal
    cmdCancel.Visible = Not bVal
    cmdDelete.Visible = bVal
    cmdClose.Visible = bVal
    cmdRefresh.Visible = bVal
    cmdNext.Enabled = bVal
    cmdFirst.Enabled = bVal
    cmdLast.Enabled = bVal
    cmdPrevious.Enabled = bVal
End Sub

```

ADO Class

It is common practice to use classes to encapsulate your ADO data access. I will include here sample code for an ADO class that connects to the “authors” table of the “biblio.mdb” sample database that ships with Visual Basic.

Option Explicit

```
Dim WithEvents adoPrimaryRS As Recordset
Private DoingRequery As Boolean
Public Event MoveComplete()
```

```
Private Sub Class_Initialize()
    Dim db As Connection
    Set db = New Connection
    db.CursorLocation = adUseClient
    db.Open "PROVIDER=Microsoft.Jet.OLEDB.3.51;Data Source=C:\Program
Files\Microsoft Visual Studio\VB98\Biblio.mdb;"
```

```
    Set adoPrimaryRS = New Recordset
    adoPrimaryRS.Open "select Au_ID,Author,[Year Born] from Authors", db,
adOpenStatic, adLockOptimistic
```

```
    DataMembers.Add "Primary"
End Sub
```

```
Private Sub Class_GetDataMember(DataMember As String, Data As Object)
    Select Case DataMember
        Case "Primary"
            Set Data = adoPrimaryRS
    End Select
End Sub
```

```
Private Sub adoPrimaryRS_MoveComplete(ByVal adReason As
ADODB.EventReasonEnum, ByVal pError As ADODB.Error, adStatus As
ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
    RaiseEvent MoveComplete
End Sub
```

```
Private Sub adoPrimaryRS_WillChangeRecord(ByVal adReason As
ADODB.EventReasonEnum, ByVal cRecords As Long, adStatus As
ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
    'This is where you put validation code
```

'This event gets called when the following actions occur

```
Dim bCancel As Boolean
Select Case adReason
Case adRsnAddNew
Case adRsnClose
Case adRsnDelete
Case adRsnFirstChange
Case adRsnMove
Case adRsnRequery
Case adRsnResynch
Case adRsnUndoAddNew
Case adRsnUndoDelete
Case adRsnUndoUpdate
Case adRsnUpdate
End Select
```

```
If bCancel Then adStatus = adStatusCancel
End Sub
```

```
Public Property Get EditingRecord() As Boolean
EditingRecord = (adoPrimaryRS.EditMode <> adEditNone)
End Property
```

```
Public Property Get AbsolutePosition() As Long
AbsolutePosition = adoPrimaryRS.AbsolutePosition
End Property
```

```
Public Sub AddNew()
adoPrimaryRS.AddNew
End Sub
```

```
Public Sub Delete()
adoPrimaryRS.Delete
MoveNext
End Sub
```

```
Public Sub Requery()
adoPrimaryRS.Requery
DataMemberChanged "Primary"
End Sub
```

```
Public Sub Update()
With adoPrimaryRS
.UpdateBatch adAffectAll
If .EditMode = adEditAdd Then
MoveLast
```

```
End If
End With
End Sub
```

```
Public Sub Cancel()
With adoPrimaryRS
.CancelUpdate
If .EditMode = adEditAdd Then
MoveFirst
End If
End With
End Sub
```

```
Public Sub MoveFirst()
adoPrimaryRS.MoveFirst
End Sub
```

```
Public Sub MoveLast()
adoPrimaryRS.MoveLast
End Sub
```

```
Public Sub MoveNext()
If Not adoPrimaryRS.EOF Then adoPrimaryRS.MoveNext
If adoPrimaryRS.EOF And adoPrimaryRS.RecordCount > 0 Then
Beep
'moved off the end so go back
adoPrimaryRS.MoveLast
End If
End Sub
```

```
Public Sub MovePrevious()
If Not adoPrimaryRS.BOF Then adoPrimaryRS.MovePrevious
If adoPrimaryRS.BOF And adoPrimaryRS.RecordCount > 0 Then
Beep
'moved off the end so go back
adoPrimaryRS.MoveFirst
End If
End Sub
```

ADO Data Control

Now the ADO data control is very much like the DAO data control. It has some similar properties. Rather than rehash all those properties and methods I will simply show you the properties of an ADO data control that is set to the “publishers” table of the sample “biblio” database that ships with Visual Basic. I will also give you the source code for the form that this control is on:

Command type	8 - adCmdUnknown
Connection String	PROVIDER=Microsoft.Jet.OLEDB.3.51;Data Source=C:\Program Files\Microsoft Visual Studio\VB98\Biblio.mdb;
Cursor Location	AdUseClient
Cursor Type	AdOpenStatis
Lock Type	AdLockOptimistic
Record Source	select Address, City, Comments, [Company Name], Fax, Name, PubID, State, Telephone, Zip from Publishers

Now for the forms code:

Option Explicit

```
Private Sub Form_Unload(Cancel As Integer)
    Screen.MousePointer = vbDefault
End Sub
```

```
Private Sub datPrimaryRS_Error(ByVal ErrorNumber As Long, Description As String,
ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal
HelpContext As Long, fCancelDisplay As Boolean)
    'This is where you would put error handling code
    'If you want to ignore errors, comment out the next line
    'If you want to trap them, add code here to handle them
    MsgBox "Data error event hit err:" & Description
End Sub
```

```
Private Sub datPrimaryRS_MoveComplete(ByVal adReason As
ADODB.EventReasonEnum, ByVal pError As ADODB.Error, adStatus As
ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
    'This will display the current record position for this recordset
    datPrimaryRS.Caption = "Record: " & CStr(datPrimaryRS.Recordset.AbsolutePosition)
End Sub
```

```
Private Sub datPrimaryRS_WillChangeRecord(ByVal adReason As  
ADODB.EventReasonEnum, ByVal cRecords As Long, adStatus As  
ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
```

```
'This is where you put validation code
```

```
'This event gets called when the following actions occur
```

```
Dim bCancel As Boolean
```

```
    Select Case adReason  
        Case adRsnAddNew  
        Case adRsnClose  
        Case adRsnDelete  
        Case adRsnFirstChange  
        Case adRsnMove  
        Case adRsnRequery  
        Case adRsnResynch  
        Case adRsnUndoAddNew  
        Case adRsnUndoDelete  
        Case adRsnUndoUpdate  
        Case adRsnUpdate  
    End Select
```

```
    If bCancel Then adStatus = adStatusCancel  
End Sub
```

```
Private Sub cmdAdd_Click()  
    On Error GoTo AddErr  
    datPrimaryRS.Recordset.AddNew
```

```
Exit Sub  
AddErr:  
    MsgBox Err.Description  
End Sub
```

```
Private Sub cmdDelete_Click()  
    On Error GoTo DeleteErr  
    With datPrimaryRS.Recordset  
        .Delete  
        .MoveNext  
        If .EOF Then .MoveLast  
    End With  
Exit Sub  
DeleteErr:  
    MsgBox Err.Description  
End Sub
```

```
Private Sub cmdRefresh_Click()
```

```

This is only needed for multi user apps
On Error GoTo RefreshErr
datPrimaryRS.Refresh
Exit Sub
RefreshErr:
MsgBox Err.Description
End Sub

Private Sub cmdUpdate_Click()
On Error GoTo UpdateErr

datPrimaryRS.Recordset.UpdateBatch adAffectAll
Exit Sub
UpdateErr:
MsgBox Err.Description
End Sub

Private Sub cmdClose_Click()
Unload Me
End Sub

```

Conclusion:

Now this is just a very simple overview of ADO and is not designed to make you a database guru. It should however, peak your interest in ADO. After you have finished this textbook you should have a basic idea of two different data access approaches: DAO and ADO. You will, of course, want to explore ADO in more depth.

If you carefully study this chapter (after having a firm grasp on the DAO chapter) and you carefully do the end of chapter projects you should have a basic working knowledge of ADO.

Review Questions:

1. What does ADO mean?
2. What is a connection string?
3. What is a command object?
4. Does ADO support ODBC databases?
5. How do you actually make a command object do its command?
6. Can an ADO object connect to an Oracle Database?
7. What is a stored procedure?
8. What does the property adCmdText tell you?
9. What does the requery command do?
10. What does the rollbacktrans command do?

Chapter Project #1: Go back to the rolodex application and completely redo it from scratch using ado code to access your data

Chapter Project #2 Go back to the rolodex application and completely redo it from scratch using an ADO class to access the data.

Chapter Project #3 Go back to the rolodex application (surprise!) and completely redo from scratch using the ADO data control to access the data.

Active X DLL's

Chapter VIII Active X Dll's

Objectives

- Understand the basics of what a dll is.
- Understand the basics of what COM is
- Be able to create a standard Active X dll

Introduction I am sure you have seen the job ad's demanding knowledge of Active X technology. Well this chapter, and the next, will get you the basics of Active X. First I will introduce you to the concepts behind Active X, then I will walk you through the process of actually creating your own Active X dll's and Active X controls. These two chapters won't make you an Active X guru, but will get you started.

What is a DLL? A Dll is a Dynamic Linked Library. You can think of a Dll as a bag of functions that you link your program too. It is frequently a good idea to take the business logic of your program and place it in a dll. That way if you need to change the business logic, you can simply update the dll' without making any changes to the user interface.

What is Active X: First of all we must answer the question: What is COM? COM is Component Object Model. In short, it is a pre defined way for various applications to communicate with each other, regardless of what language they are written in, or where they are located.

Active X is simply an implementation of COM. So in short Active X is a means for exposing the functions of an application, so that other applications can use them.

Creating an Active X dll:

Creating an Active X dll is really pretty simple. The following example illustrates this.

Example 8-1

Step 1 Start a new application with 'Active X dll' as type.

Step 2 You will now have a class1 , change its name to clsmath.

Step 3 Go to project properties and change the project name to MathWiz

Step 4 Now we are going to add several public subs to this class.

Option Explicit

Public Function cube(number As Single) As Double

Dim answer As Double

answer = number * number * number

cube = answer

End Function

```
Public Function power(number As Single, pow As Single) As Double  
    Dim answer As Double
```

```
        answer = number ^ pow
```

```
        power = answer  
End Function
```

```
Public Function square(number As Single) As Double  
    Dim answer As Double
```

```
        answer = number * number
```

```
        square = answer
```

```
End Function
```

Now simply go to File > make MathWiz.dll

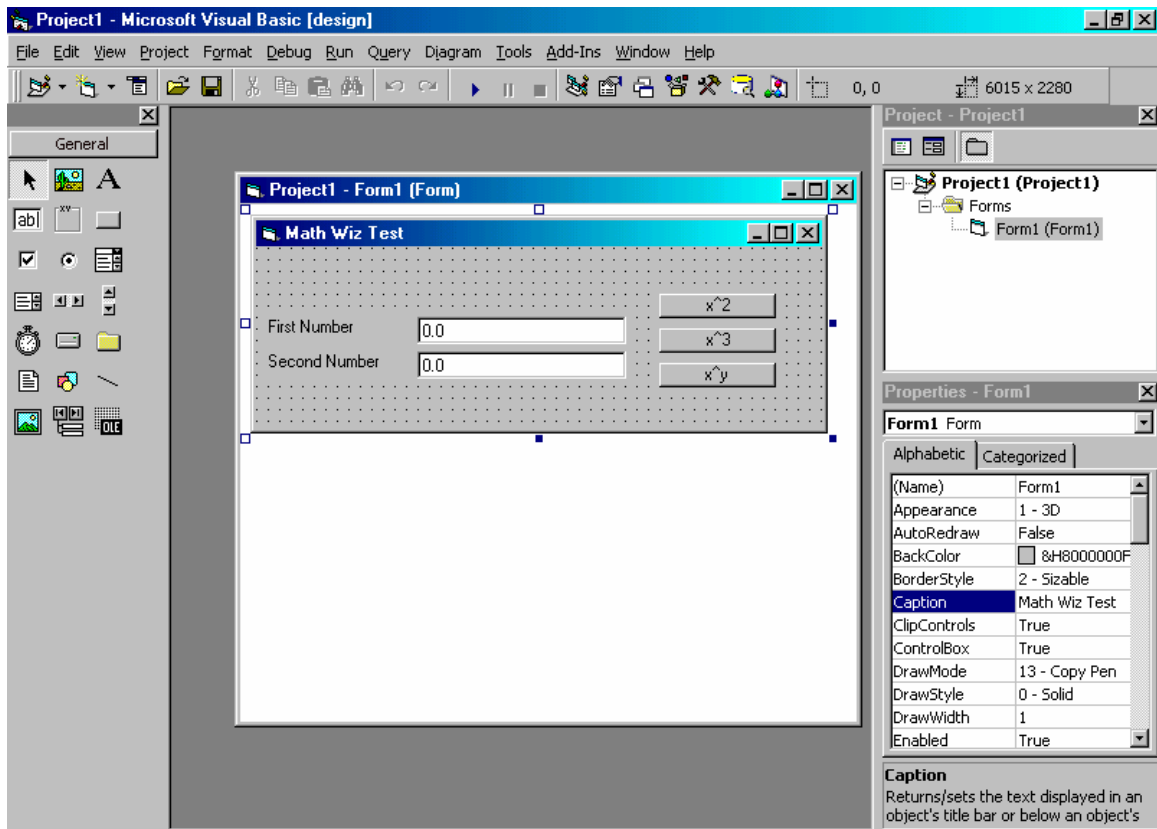
Now you will start a new application that will be a standard executable. Inside of it you will go to its project > references and select browse to find mathwiz.dll and select it.

Step 5 In the general declarations section of this form place this code

Option Explicit

```
Dim mathobj As New MathWiz.clsmath
```

Step 6 Now you are going to add a few buttons, textboxes, and labels to your form. It should look like this.



Step 7 Place the following code in your three command buttons

```
Private Sub Command1_Click()
```

```
    Dim num As Single
```

```
    Dim answer As Double
```

```
    num = Val(Text1.Text)
```

```
    answer = mathobj.square(num)
```

```
    MsgBox (num & " squared is " & answer)
```

```
End Sub
```

```
Private Sub Command2_Click()
```

```
    Dim num As Single
```

```
    Dim answer As Double
```

```
    num = Val(Text1.Text)
```

```
    answer = mathobj.cube(num)
```

```
MsgBox (num & " cubed is " & answer)
```

```
End Sub
```

```
Private Sub Command3_Click()  
Dim num As Single, num2 As Single  
Dim answer As Double
```

```
num = Val(Text1.Text)  
num2 = Val(Text2.Text)
```

```
answer = mathobj.power(num, num2)
```

```
MsgBox (num & " raised to " & num2 & " power is " & answer)
```

```
End Sub
```

Conclusion

So essentially creating an Active X dll in Visual Basic is as simple as starting a new Active X dll and making some public subroutines or functions in that dll. That is all it takes. Now for the big question: Why? Well its simple you can make dll's that handle complex operations (such as database connectivity) then whenever you need that functionality, just put a reference in your project to that dll.

Review Questions:

1. What is COM?
2. What is Active X
3. What is a DLL?
4. How do you create a reference to a dll in an normal project

Chapter Project

Remember the DAO code we did earlier? Well now you will place all of that into an Active X dll class and compile it. Then you can use it in any program that needs database access.

Active X Controls

Chapter IX Active X Controls

Introduction: In the last chapter I briefly introduced you to the concepts of COM and Active X. In this chapter We are going to apply those same concepts to an Active X control. Essentially an Active X control is like any of the control's in the toolbox that you use on a regular basis. What you will do in this chapter is simply make your own controls to use.

General Concepts: First of all you should think of the way you communicate with current controls in the toolbox. Essentially you set their properties, call their methods, and then they respond via events. For example the command button has properties such as caption, and it has events such as click().

Example 9-1

The easiest way to learn how to create Active X controls is to create a compound control. To create a compound control you simply take 2 or more existing controls, combine their features and perhaps add some new features.

Step 1: Open a new project as an Active X control project.

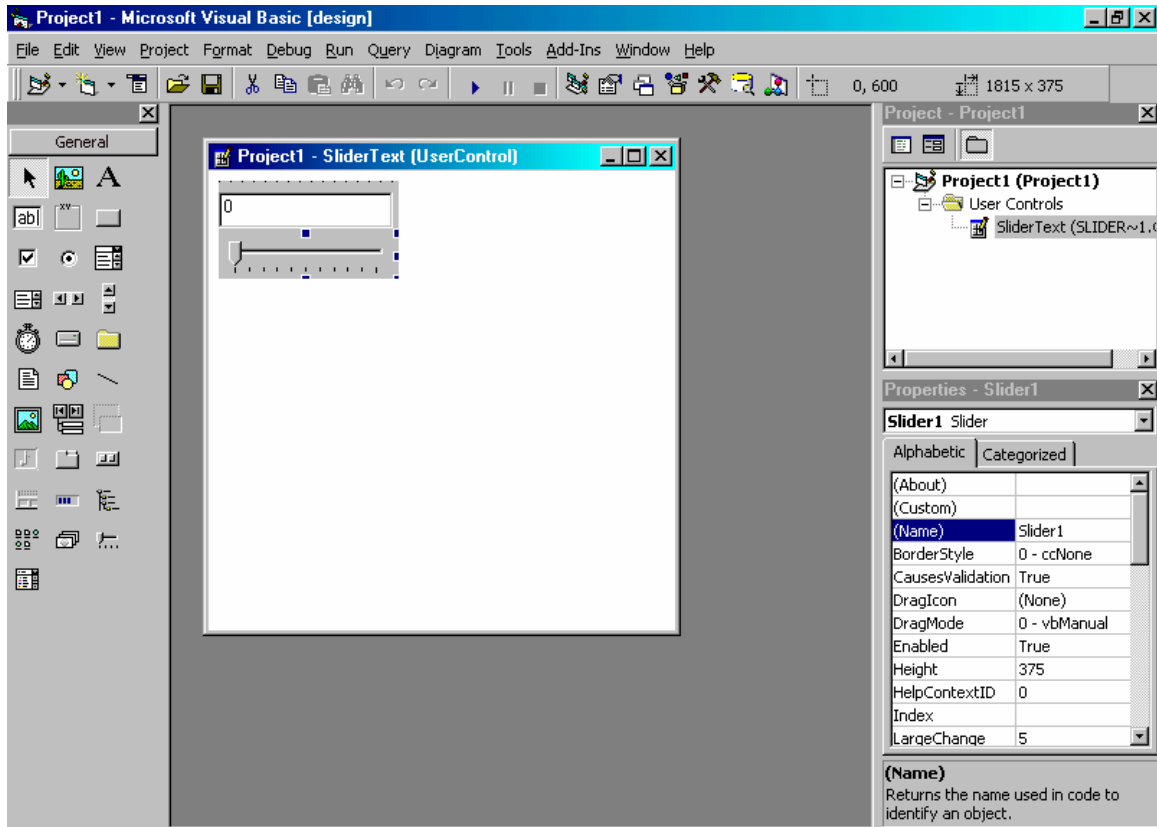
Step 2: Set the following properties of the User Control Window:

InvisibleAtRuntime = false

Set the ToolBoxBitmap and the MaskPicture to any pictures you wish

Step 3: In project > properties change the projects name to slidertext

Step 4: On the User Control window place a text box and a slider control (to place the slider control you will have to go to your project > components and add "Microsoft Windows Common Controls". When you have done this your screen should look like this:



Step 6:

We are going to put some code in a few places

```
Private Sub Slider1_Change()
```

```
    On Error Resume Next
```

```
    Text1.Text = Slider1.Value
```

```
    If bflagextreme = True Then
```

```
        If ((Slider1.Value) >= (0.9 * (Slider1.Max))) Then
```

```
            Text1.ForeColor = RED
```

```
        Else
```

```
            Text1.ForeColor = BLACK
```

```
        End If
```

```
    Else
```

```
        Text1.ForeColor = BLACK
```

```
    End If
```

```
End Sub
```

```

Private Sub Text1_Change()
    On Error Resume Next
    Slider1.Value = Text1.Text
End Sub
Private Sub UserControl_Resize()
    Text1.Width = UserControl.Width
    Slider1.Width = UserControl.Width
End Sub

```

Option Explicit

```

Private bflagextreme As Boolean

```

```

Private Const RED = &HFF&
Private Const BLACK = &H0&
Public Property Let flagextreme(Extreme As Boolean)
    bflagextreme = Extreme
End Property

```

```

Public Property Get flagextreme() As Boolean
    flagextreme = bflagextreme
End Property
Public Property Let Max(MaxValue As Integer)
    On Error Resume Next

```

```

    Slider1.Max = MaxValue
End Property

```

```

Public Property Get Max() As Integer
    On Error Resume Next
    Max = Slider1.Max

```

```

End Property

```

Now compile your active X control a

Step 7

Now make a new project that is a standard executable. Go to project > components, and use browse to add this component to your tool box. Then place it in the form. Now when you move the slider , the textbox will reflect its value, and visa versa. Also if you set the flagextreme property to true, whenever your value is within 10% of the maximum, the text will turn red.

See how easy that was!

Now there may be occasions where you want your component to communicate something back to the host application. This is done via events. You first declare an event in the general declarations section of your Active X Control (BTW Active X dll's can also have events, and they are created in exactly the same way).

This is a declaration for an event in the previous example. This event is designed to let the host application know that they have attempted to enter a value that is beyond the max value.

```
Public event ValueOutOfRange(message as string)
```

Now any where you would like to fire off this event you simply call it and pass it a string parameter. Lets do an example inside the text1.lost_focus event

```
Raiseevent ValueOutOfRange("The value you entered it too large!")
```

Conclusion:

I hope you saw, in these few pages, just how easy it is to create an Active X component of your own. You should consider making an Active X component any time there is a situation where you routinely have to set a standard control (s) in a certain way.

TCP/IP Communications

Chapter X TCP/IP Communication

Objectives

- Understand the TCP/IP protocol
- Be able to use the Winsock Control to create TCP/IP communications

Creating TCP/IP communications inside of Visual Basic is actually a trivial matter. The first step is to add the Winsock control to your project. To do this go to "Project" then "Controls" and add in the Winsock control:

Now on to how to use it. In order for TCP/IP communications to work one end needs to be a server and then clients communicate with it. In order to make a server you place the following code in the form load event of the form that has the Winsock control on it.

```
Winsock1.LocalPort = 100
```

```
Winsock1.Listen
```

Now I picked port 100 at random. TCP/IP ports can be anywhere from 1 to a little over 32000. However for custom applications you will want to avoid certain ports such as 21 (always used for FTP) and 80 (used for http). Now what you have is a Winsock that is listening. But what do you do when someone wants to send it a packet?

In the Winsock Winsock_ConnectionRequest event place this code:

```
If Winsock1.State <> sckClosed Then _
```

```
Winsock1.Close
```

```
' Accept the request with the requestID
```

```
' parameter.
```

```
Winsock1.Accept requested
```

Now this allows your Winsock to accept packets from that particular TCP/IP client.

Now how do you get the data? Well you go to the Winsock_DataArrival event and place this code:

```
Dim strBuffer as string
```

```
Winsock1.getdata strBuffer
```

Now the entire data portion of the incoming TCP/IP packet will be placed in the string strBuffer.

Note: If you open a socket but don't close it then you will have to reboot that machine before you can use that socket again. So I suggest when you are done placing a

```
Winsock1.close
```

Now what about sending data? Well that's pretty easy:

```
Winsock1.remoteport ' What port do you want to send it too. It has to match the port  
' that the server is listening in on.
```

```
Winsock1.remoteip ' What IP are you sending it to?
```

```
Winsock1.senddata strBuffer ' Just put your data into the string and away it goes!!
```

This is just an introduction to basic TCP/IP communications in Visual Basic with the Winsock control. With various API's and Controls you can do any type of internet communication you want inside of Visual Basic.

Appendix A: Key Words

This list is by no means exhaustive. It is simply a quick reference of those keywords covered in this textbook. These are the keywords that I have found beginners are most likely to use.

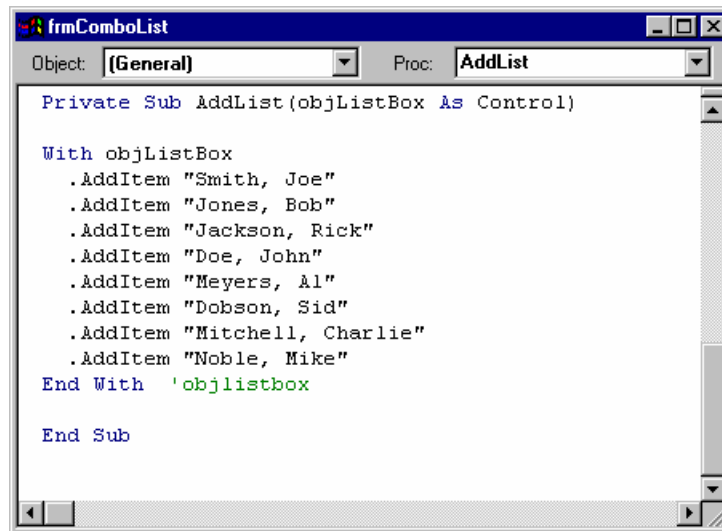
Additem: This method allows you to add items to a list box or a combo box. For example if you wanted a list box that allowed a user to add a title to some persons last name you could place this code in the forms load event:

```
List1.additem "Mr. "  
List1.additem "Mrs. "  
List1.additem "Ms. "  
List1.additem "Dr. "  
List1.additem "Rev. "
```

Then in the double click event of that list box you could add this title to a text box that had the last name like this:

```
Let txtlastname.text = list1 & txtlastname.txt
```

Or you can make a separate subroutine to add the list items:



```
frmComboList  
Object: (General) Proc: AddList  
Private Sub AddList(objListBox As Control)  
  
With objListBox  
    .AddItem "Smith, Joe"  
    .AddItem "Jones, Bob"  
    .AddItem "Jackson, Rick"  
    .AddItem "Doe, John"  
    .AddItem "Meyers, Al"  
    .AddItem "Dobson, Sid"  
    .AddItem "Mitchell, Charlie"  
    .AddItem "Noble, Mike"  
End With 'objlistbox  
  
End Sub
```

App.Activate: This statement allows you to activate a program that is already running. You can either, activate a program by using its name as it appears in the tool bar or by using the windows handle returned to lShell.

```
App.activate lshell
```

Or

```
App.activate "MS Word"
```

Cbyte: This converts a variable of one type into a byte. If the expression you are converting lies outside the acceptable range for a byte, you will get an error. The syntax is: Cbyte(myvariable)

Cdate: Much like Cbyte, this expression converts a variable to the Data type. Its syntax is: Cdate (myVariable)

There are also statements like Clong, Cint, etc to convert a variable to an integer or a long value.

Chr: This returns the actually character denoted by an ASCII character number for example chr(65) returns 'A'

Clear: This keyword can be applied to the clipboard object, a combobox, or a listbox. It causes all items stored therein to be cleared.

```
List1.clear
```

Cos: This will return the cosine of a number. For example you can say "Cos(15) to get the cosine of 15.

Date: This simply returns the current system date. You might try some code like:

```
Let txtdate.text = Date
```

FileCopy: This command causes a file to be copied from its source to a destination file. You might do something like : FileCopy "sourcefile.txt" ,"destinationfile.txt"

FileLen: This function returns the length of a file in bytes. Its syntax is : Filelen("myfile.txt")

Format: This allows you to change the format of any variable, textbox, or label. Let's look at some examples:

If I want the date to be yymmdd then I use the format command like this:
Let txtdate.text = format(date,"yymmdd")

You see I place an open parentheses after the format command then I tell it what I wish to format. In this case, that is "Date". The in quotation marks I tell it how to format it. We can also format numbers. For example if you want a number to appear as a dollar amount:

```
Let txtmynumber = format(txtmynumber,"###,###.##")
```

Your number will show a dollar sign then six digits a decimal point and two additional digits.

Hide: This makes whatever you are referring to invisible. Here is an example:

```
FrmOrders.hide
```

The form is still loaded its just not visible

IsDate: This function will return a boolean value (True/False) indicating whether or not a variable can be converted to a date. The syntax is: Isdate(myvariable).

IsNull: This function is used to see if another variable is null. You use it like this:
isnull(myvariable)

Left: This command allows you to get the left portion of a string. For example if you want to get the left three characters in a text box you can use this code:

```
Private sMystring as string
```

```
Let sMystring = left(txtMytext,3)
```

This will place the left 3 characters of txtMytext into the string sMystring. So if the text box txtMytext had the word "Hello" in it, sMystring would now have "Hel" in it.

Len: If you want to get the length of some variable or text you can use the Len command to get that value. Here are two examples

```
Let iLength = Len(txtname) 'this will take the length of the text in txtname  
' and place it in the variable iLength
```

```
Let iLength = Len(iAccount) 'This will take the length of the variable iAccount  
' and place it in the variable iLength
```

Let: This assigns a value to a control or variable. Here are a few examples:

Let iCounter = 0 ‘this sets the variable iCounter to zero

Let iAccount = txtAccount ‘This takes the value of txtAccount and copies it to
‘iAccount.

Load: This will load a form. If you have a main form and you wish to load other forms, you can use this command. For example if you have a form called frmOrders and you have a command button you wish to use to load it in, place the following code in that command buttons click event:

Load frmOrders

Ltrim: This command will remove all trailing spaces from the left side of a string.

Msgbox: This key word simply tells the computer that you want to display a message box.

Removeitem: This works just like the add.item example only it removes an item from a list or combo box.

Right: This is very similar to the left command, except that it returns the right side characters requested. For example:

Private sMystring as string

Let sMystring = right(txtmytext,2) would result in :

“lo”

I think you will find the Left and Right command very useful

Sendkeys: After app.activate you can send keystrokes to a program. For example, if you have activated the notepad.exe you can then

Sendkeys “My Dog Has Flees”

And that phrase will appear in the notepad!

SetFocus: This tells the program to change focus to another control. You might place something in your cmdAddNew button that sets focus to the first text box on your form:

```
CmdAddnew_Click()
```

```
    TxtFirst.setfocus
```

```
End sub
```

Shell: This is a very useful key word. By use of the shell command, you can start other programs. Look at this example:

```
Private lshell as long
```

```
Lshell = shell("c:\path\name.exe",1)
```

This tells the computer to take the path name and start the program "name.exe" in standard mode (1) and return its windows handle to the variable entitled lShell.

Show: This shows something that is hidden. If you use it in conjunction with a form that is not loaded then it will first load the form then show it.

```
    FrmOrders.show
```

Sin: This will return the Sine of a variable such as sin(45) returns the sine of 45.

Time: This returns the current system time. You can display this to the user with some code like:

```
    Let lbltime.caption = Time
```

Trim: This command will remove all blank spaces from a string

Ucase: What if you wish to compare to values but you don't care if their case matches (lower or upper case)? You can compare the upper case values of both. For example in the Do-Until example previously used what happens if the value you are searching for is "Smith" and the value in the database is "SmIth"? You won't get a match. If you use the Ucase statement then the upper case values of both will be compared. Here is an example:

If Ucase(sSearchValue) = ucase(txtLastName) then...

You can also compare the lower case values using the same technique with the keyword

Lcase

Appendix B: Control Reference

DirListBox: A DirListBox control displays directories and paths to the user. This control can display a hierarchical list of directories. You can create dialog boxes that allow a user to open a file from a list of files in all available directories.

DriveListBox: A DriveListBox control enables a user to select any disk drive (floppy, CD, DVD, Hard disk, Zip, Jazz, networked, etc.) You can create dialog boxes that allow the user to open a file from a list of files on a disk in any available drive

ImageControl: Use the Image control to display a graphic. An image control can display jpeg, gif, bitmaps, metafiles, and icons. This control uses fewer system resources and repaints faster than a PictureBox control, but it has fewer properties and methods than the picture box.. Use the Stretch property to cause the image to scale to fit the image control.

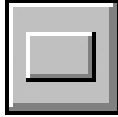
Text box: This control allows the end user to enter data. You will want to set its name (What the computer will call it), its text (the default text that initially appears when you run the program), and possibly its font if you wish a different font in the text box. You can also enter something into the text property as a default text. Finally you may want to set its max length property. This determines the maximum number of characters a user can type into the text box. You can also set its password character property to denote a symbol to show in lieu of passwords. The multi-line and scroll bar properties allow you to have a text box that can handle several lines of text.



Label: Labels are used to display information to the end user, but the end user cannot change any information in the label. The properties you will set in this control are the name (What the computer will call it), its caption (what will the user see), and its font.



Command Button: Command buttons allow the user to click a button. A lot of the code that you write will be associated with command buttons. The properties you are most interested in are: its name (What the computer will call it), its caption (What the user will see), its style (Will your button have a text caption or a picture, if you choose the graphical style you must set the caption equal to nothing and select a picture in the picture property) , and its font.



List Box: A list box simply allows you to list items for a user to select from. The property you are concerned with is the font. The items are listed using code and will be discussed later.

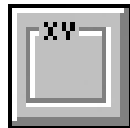
Combo Box: This is like a list box but you can set more options.



Picture box: Using this control you can place any bitmap (*.bmp), Windows MetaFile (*.wmf), or icon (*.ico) on your form. Starting with Visual Basic Version 5.0 you can also add in JPEG (*.jpg) and GIF (*.gif) image files. All you do is set the stretch property to true (that ensures that the picture you insert will be the size of your picture box) and then use the picture property to select the picture you want to use.



Frame: A frame is basically a container control. It is used to group other controls. The property you will use most with it is its caption.

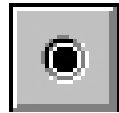


Check Box: The check box is a very useful control. When you place it on your form it will allow the user to select an option. For example if your form is for employee information and you are inquiring about an employees educational background, you might have check boxes for Bachelors, Masters, Technical School, Doctoral, and Corporate training. A person might need to select more than one. A person may have a Bachelors and have attended a tech school. The check box allows you to select more than one option.



The properties you are concerned with are: caption and style. The style denotes either a standard check box or a graphical check box.

Option box: The option box works in much the same way as the check box except that only one can be selected at a given time. It is used when the user must select 1 option from several choices.



Here is an example of using an option box: Let us assume that you have an employee tracking program that is connected to a database via a data control (data1) and you wish the user to select the employees status. You could place three option boxes on the form named optSalaried; optFullHrly; and optPart. Then in your cmdsave button you could have an if then loop that selects which option box was selected and then assigns an appropriate value to the database:

```
Cmdsave_click()
```

```
  If optsalaried.value = true then
```

```
    Data1.recordset.fields("Employee Status") = "salaried"
```

```
  ElseIf optfullhrly.value = true then
```

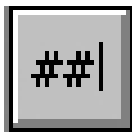
```
    Data1.recordset.fields("employee Status") = "Full Time Hourly"
```

```
  Else
```

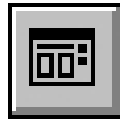
```
    Data1.recordset.fields("Employee Status") = "Part Time"
```

```
  End if
```

Masked Edit Box: This is like a text box but you can format the way in which user can input data. You just type into the masked property what format you want the data in. For example: ###/###/### .



Common Dialog: This is a very useful control. With it you set up a dialog box, as is standard in all windows programs, whereby the user can search their computer for a particular file of a particular type.



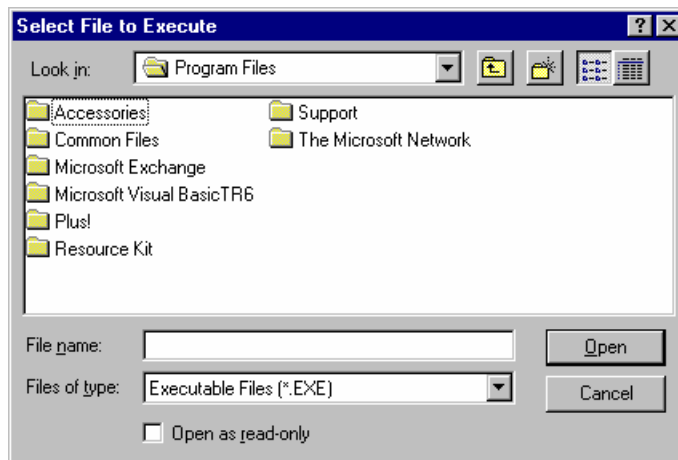
The properties you are concerned with are:

Dialog Title: This gives a title to your dialogue box

Init Dir: This selects an initial directory to open in

File Ext: This gives a file extension for files that are saved

This control is used in the MDI example under the VB/Samples directory. Once you have set these appropriate properties (and the caption) you can utilize the control in your applications. Your users will see this when the common dialogue control is executed:



The filter is that property, which tells it what kind of, files to open or save. For example you might want a dialogue box that looks for text files (*.txt) then you would set the filter as follows:

```
CommonDialog1.filter = "Text Files(*.txt)|*.txt"
```

The first section – “Text Files (*.txt) is what your user will see in the “Files of Type” section of the dialog box. The second part - |*.txt is the part the program needs to read. You can have several file types. Simply add another | then do the same for each file type you want. For example if you wanted your dialogue box to look for text files and for batch files:

```
Commondialog1.filter = “Text Files (*.txt)|*.txt | Batch Files (*.bat)|*.bat”
```

Then you will need to tell the dialog box whether you wish to look for a file to open, or to save. You do this with a very simple method:

```
Commondialog1.showopen
```

Or

```
Commondilaoge1.showsave
```

You will probably find many uses for the common dialog box.

Appendix C: Other Resources

This book is simply meant to get you started with Visual Basic. There is a lot more you can learn. Below is a list of some of my favorite resources this list is not exhaustive, but most of the Web Pages have links to other Web pages.

Internet:

Ask the VB Pro: (<http://www.inquiry.com/thevbpro/>) A rather famous Visual Basic page

Comp.lang.basic.visual.misc: This is a usenet newsgroup, you can ask questions and usually get answers

Chuck Easttoms Visual Basic Page (<http://www.geocities/~chuckeasttom/vb/Vb.htm>) This is my own page for Visual Basic beginners.

Carl And Gary's VB Page (<http://www.cgvb.com>) This is the mother of all Visual Basic Pages.

Ed's VB Page: (<http://www.tiac.net/users/efields/vbhp.htm>) This is a pretty good page with lots of resources

Microsoft Certification Page: (<http://www.microsoft.com/mcp/ie30.htm>) If you are interested in getting certified from Microsoft this is the page for you

Zarr's VB Site: An excellent Visual Basic web site with lots of content and links.
www.zarr.net

The Visual Basic Web Magazine: This is an online magazine with articles you can read!
www.vbwm.com

Each of these excellent web sites will provide you with links to dozens more web sites.

Books:

The Database How To– To book from the Waite Group is an awesome book for serious Database programming.

Visual Basic 6 Database Programming by John Connel. This is an outstanding book on database programming and it does a fine job of discussing ADO.

Visual Basic 6 Objects by Peter Wright. This outstanding book will take you through all the detailed ins and outs of Object Orientation.

Murach's Visual Basic 6: This is a great book to round out your introduction to Visual Basic. It is an excellent starting place for beginners.

Daniel Appleman's Guide to the API – the definitive book on the Windows API

The Practical SQL Handbook – this is the bible of SQL. If you read and practice the stuff in this book you will be an SQL guru!

The VB SuperBible: A really good reference book.

Appendix D: Get Certified and Get Ahead

Training is vital to your career. It is also vital to differentiate yourself from the pack. One of the best ways to do this is to pursue industry recognized certifications. Below I discuss some of the most widely recognized and give you links.

Microsoft Certifications

Microsoft: has many different types of certifications. We will discuss those that have the most bearing on the software developer. The first level of Microsoft Certification is the Microsoft Certified Professional (MCP). You earn this title by successfully completing any single Microsoft test (excluding the networking essentials test). For the new Visual Basic Developer I would recommend you take the: Exam 70-176: Designing and Implementing Desktop Applications with Microsoft® Visual Basic® 6.0 . After you successfully complete this one test, you could complete 3 more and become a Microsoft Certified Solutions Developer.

New MCSD Track

Core Exams (3 required)

Desktop Applications Development (1 required):

Exam 70-016: Designing and Implementing Desktop Applications with Microsoft® Visual C++® 6.0

Exam 70-156: Designing and Implementing Desktop Applications with Microsoft® Visual FoxPro® 6.0

Exam 70-176: Designing and Implementing Desktop Applications with Microsoft® Visual Basic® 6.0

(Designing and Implementing Desktop Applications with Microsoft Visual J++ available in 1999.)

Distributed Applications Development (1 required):

Exam 70-015: Designing and Implementing Distributed Applications with Microsoft® Visual C++® 6.0

Exam 70-155: Designing and Implementing Distributed Applications with Microsoft® Visual FoxPro® 6.0

Exam 70-175: Designing and Implementing Distributed Applications with Microsoft® Visual Basic® 6.0

(Designing and Implementing Distributed Applications with Microsoft Visual J++ available in 1999.)

Solution Architecture (required):

Exam 70-100: Analyzing Requirements and Defining Solution Architectures

Elective Exams (1 required)

Exam 70-015: Designing and Implementing Distributed Applications with Microsoft® Visual C++® 6.0
Exam 70-016: Designing and Implementing Desktop Applications with Microsoft® Visual C++® 6.0
Exam 70-019: Designing and Implementing Data Warehouses with Microsoft® SQL Server™ 7.0
Exam 70-024: Developing Applications with C++ Using the Microsoft® Foundation Class Library
Exam 70-025: Implementing OLE in Microsoft® Foundation Class Applications
Exam 70-027: Implementing a Database Design on Microsoft® SQL Server™ 6.5
Exam 70-029: Designing and Implementing Databases with Microsoft® SQL Server™ 7.0
Exam 70-055: Designing and Implementing Web Sites with Microsoft® FrontPage® 98
Exam 70-057: Designing and Implementing Commerce Solutions with Microsoft® Site Server 3.0, Commerce Edition
Exam 70-065: Programming with Microsoft® Visual Basic® 4.0
(This exam has retired. Please see the "Retirement of Exams" notice.)

Exam 70-069: Application Development with Microsoft Access for Windows® 95 and the Microsoft Access Developer's Toolkit
Exam 70-091: Designing and Implementing Solutions with Microsoft® Office 2000 and Microsoft® Visual Basic® for Applications
Exam 70-097: Designing and Implementing Database Applications with Microsoft Access 2000
Exam 70-105: Designing and Implementing Collaborative Solutions with Microsoft® Outlook® 2000 and Microsoft® Exchange Server 5.5
Exam 70-152: Designing and Implementing Web Solutions with Microsoft® Visual InterDev™ 6.0
Exam 70-155: Designing and Implementing Distributed Applications with Microsoft Visual FoxPro® 6.0
Exam 70-156: Designing and Implementing Desktop Applications with Microsoft Visual FoxPro® 6.0
Exam 70-165: Developing Applications with Microsoft® Visual Basic® 5.0
Exam 70-175: Designing and Implementing Distributed Applications with Microsoft® Visual Basic® 6.0

Exam 70-176: Designing and Implementing Desktop Applications with Microsoft® Visual Basic® 6.0

Core exams that can also be used as elective exams can only be counted once toward a certification; that is, if a candidate receives credit for an exam as a core in one track, that candidate will not receive credit for that same exam as an elective in that same track.

For more information see www.microsoft.com/mcp/

TekMetrics/ Brainbench

Another, less expensive way to get certified is to use the TekMetrics (recently renamed to the “Brain Bench”)certifications. These are 3rd Party tests over a variety of computer topics including Visual Basic.

For more information see www.brainbench.com

Appendix F DDE and OLE

With the advent of Active X, OLE and DDE are rarely used anymore. This appendix is provided for two reasons. The first being that you may encounter older code using these techniques, and the second being that getting a basic understanding of these methodologies may help you understand Active X a bit better.

DDE (Dynamic Data Exchange)

Basically DDE allows you to link up different applications so they can transfer information between each other. Such a transfer is referred to as a DDE Conversation. The application providing data is the source and the application receiving data is the destination. To establish a DDE Conversation the destination application must specify:

- The name of the source application. This could be Excel, Word, or perhaps another Visual Basic application.
- The topic of the conversation. This could be a specific word document, Excel spreadsheet, or Visual Basic form.
- The item of the conversation: This might be a Word bookmark, a specific cell in Excel, or a control (like a text box) in visual basic.

Now the quick and dirty way to set up a DDE conversation is to go to the application you wish to be a source (such as an excel spreadsheet) and highlight the specific item of conversation (such as a cell) and then select copy. You then can go to the destination (such as your Visual Basic application) and select the particular destination (a textbox). This time when you go to edit you will see a new option enabled: "Paste Link". When you choose this you have created an automatic DDE link between those two items. There are three types of DDE Links:

Automatic: The destination is automatically updated whenever a change is made in the source.

Manual: The destination must request an update.

Notify: The source will notify the destination if there is a change and the destination must then confirm the update.

Now you may not want to have a link always on. You may wish to include a subroutine to initiate the link when you are ready for the DDE conversation. Here is an example from an actual program:

```

Public Sub SetLinkMode()
    'This module is responsible for resetting the Linkmode. If a source application
    ' terminates a conversation with a Visual Basic destination control, the value for that
    ' controls LinkMode setting changes to 0 (None).

    Dim i%                'used as a counter
    Dim c As Control
    Dim f As Form
    Dim LoopCount As Integer

    Set f = frmGetDataLink

    On Error Resume Next

    'Scroll through all controls on the form and sets the link mode if the control is a text
    ' box
    For i = 0 To f.Controls.Count - 1
        Set c = f.Controls(i)

        If typeof c Is TextBox Then
            c.LinkMode = 1
            DoEvents
        End If

    Next

    LoopCount = 1
    Do While LoopCount < 50
        If frmGetDataLink!txtDDEReady.Text = "1" Then Exit Do
        LoopCount = LoopCount + 1
        DoEvents
    Loop

End Sub

```

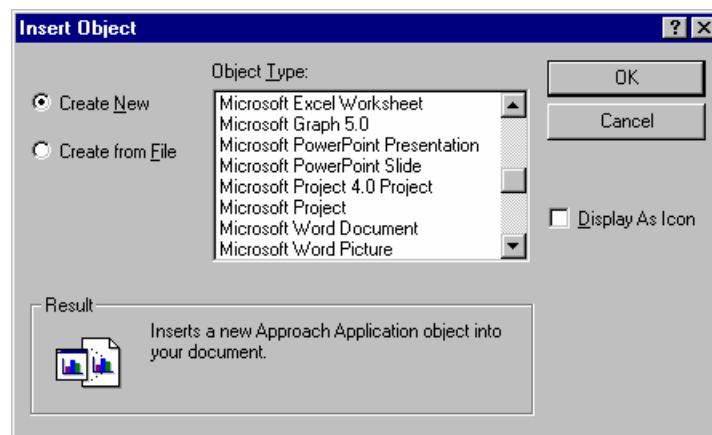
In this example we cycle through all the controls and set their LinkMode to automatic. Each textbox has been paste linked to the source application at design time, now at runtime this routine will fire off the DDE conversation.

Basic OLE

OLE means Object Linking and Embedding. By the use of OLE you can basically connect your visual basic application to other windows applications. Now the quick and dirty way to do this is to place the OLE control on your form and set a few properties:



After you have placed the OLE control on your form, simply click on it. You will be presented with a window that asks you to choose either “Create New” or “Create From File”.



Create New: This will create a new blank object. This could be a blank word document or an excel spreadsheet.

Create from File: This prompts you to browse your hard drive and find some object to connect to, such as a Word Document.

For our purposes choose “Create from File” and select some document on your hard drive. Then you will select either to check the “link” box or not. If you do not then a copy of that object is actually Embedded in your form. If you do check it, then your form will contain an image which is linked to that object (see where we get the term Object

Linking and Embedding.) I will choose not to check it. You then can choose whether or not to display this embedded document as an icon. For our example choose yes.

Now when you run your program the object you selected is in your application. Any changes to the source object will be reflected in your application and visa versa. This can be a powerful tool allowing you to connect a variety of office applications to your Visual Basic Application.

OLE Automation:

OLE Automation breaks down the barriers between individual applications. It allows you to do so much more than simply communicate (as you do with DDE) it allows you to bring that applications functionality into your application.

Since computing began, it has usually been awkward to move information from one application to another. If you were writing a quarterly report. If you kept your company's inventory in a database,(Like Access) Then you retyped the data into a word processor(Like Word) to incorporate it into your report.

MS Office took this a step further by allowing you to export data from one office application to another (you can export your excel spread sheet to Word, for example) .With dynamic data exchange (DDE) and object linking and embedding (OLE 1.0), data transfers became relatively automatic and invisible to the user. But the user still had to manually activate and specify the links.

Now, OLE Automation goes far beyond simple data transfer: actually giving you control one application from another. OLE permits virtually unlimited manipulation of outside "objects. With OLE, you get essentially unrestricted access to the innards of any application that permits its objects to be used. The technical phrase for an application that allows you to communicate with it via OLE is to say it "exposes its objects."

A much more advanced example: This following example is for the adventurous soul. If you don't feel like trying your limits, this will be tough for you:

Now that you have heard the OLE sales pitch, lets look at a specific example. Let's say you wanted to borrow MS Word's Spell checker for your own application. Well you can have Word for Windows do a spell-check of the text in a VB Text Box. VB cannot check spelling, so we'll borrow the services of Word.

The way we do this is to create an object variable Create. The object will be Word's macro language, WordBasic. Here's how to create the WordBasic object:

```
Dim myWord As Object  
Set myWord = CreateObject("word.basic")
```

Once the object is created, we can use all the commands and facilities of WordBasic from within VB. But how do we use those command? Which commands will accomplish our goal? The answer is that you record a macro in WordBasic, then copy the result to the Clipboard. WordBasic will do the work itself, providing the necessary commands and their parameters.

OLE Automation from VB to Word involves "calling up" WordBasic and then feeding WordBasic commands that it understands (commands written in WordBasic). The easiest way to get the correct commands to accomplish things in Word is to record a macro using WordBasic from within Word. Then copy and paste those commands into the Sub or Function in VB where we want to trigger the OLE Automation.

Let's create our spell-check automation, step by step:

1. Start Word, and then type in a few lines of text. Highlight (select) these words by dragging the mouse across them.
2. Copy the highlighted words to the Clipboard by pressing Ctrl+Ins. (We'll send our VB Text Box's contents to Word via the Clipboard, so we want something that can substitute for the actual text while we're recording this macro.)
3. Turn on Word's macro recorder by pressing Alt+T, M.
4. Type in the macro name ForVB (or whatever name you want to give the macro).
5. Press Alt+O, Enter to start recording the macro.
6. Now, with WordBasic recording your every move, do this:
Press Alt+F, N to create a new document.
Press Shift+Ins to paste the clipboard text into the Word document.
Invoke the SpellCheck by pressing Alt+T, S, Enter.
To select the entire document, press Alt+E, L. Then press Alt+E, C to copy the spell-checked text back into the clipboard.
Finally, close the document by pressing Alt+F, C, and stop the macro from recording by pressing
Alt+T, M, Alt+O.

At this point, we want to see the results, so press Alt+E to Edit the ForVB macro.

You should see this:

```
Sub MAIN
FileNew .Template = "Normal", .NewTemplate = 0
EditPaste
ToolsSpelling
EditSelectAll
EditCopy
FileClose
End Sub
```

In our VB Sub, we can use this entire piece of programming as is, except for the first line. VB doesn't understand named parameters like `.Template = "Normal"`, so our next task will be to translate the FileNew command so that VB will know what to do with it.

VB Parameters

A parameter (sometimes called an argument) is a modification or list of specifications for a command. For example, the VB Move command has four parameters: left, top, width and height.

```
Move left[, top[, width[, height] ] ]
```

The actual names of the parameters (left, top, width and height) never appear in your programming. You merely list the values of any parameters you use:

```
Move 55, 35
```

```
FileNew .Template = "Normal", .NewTemplate = 0
```

But it does accept the traditional:

```
FileNew
```

("Normal" and 0 are the default parameters for FileNew, so we can leave them off.)

So let's copy the WordBasic macro. Select the text in Word's Macro Edit screen and press Ctrl+Ins (don't copy the Sub Main or End Sub). Now switch over to VB and double-click on the Command Button to bring up its Click Event. Then paste the WordBasic right into VB:

```

Sub cmdSpellCheck_Click ()
  Dim Wordobj As Object

  Set Wordobj = CreateObject("word.basic")

  FileNew .Template = "Normal", .NewTemplate = 0
  EditPaste
  ToolsSpelling
  EditSelectAll
  EditCopy
  FileClose

End Sub

```

Now we go through and translate the WordBasic into VB-compatible programming. First, strip off the unneeded default parameter list (.Template = "Normal", .NewTemplate = 0). Then, to let VB know that each command should be sent to the WordBasic object, add Wordobj. to the start of each command:

```

Sub cmdSpellCheck_Click ()
  Dim Wordobj As Object
  Set Wordobj = CreateObject("word.basic")

  Wordobj.FileNew
  Wordobj.EditPaste
  Wordobj.ToolsSpelling
  Wordobj.EditSelectAll
  Wordobj.EditCopy
  Wordobj.FileClose 2

  Set Wordobj = Nothing
End Sub

```

Now all that remains is to insert the VB commands that will copy the Text Box's contents to the Clipboard and, at the end, paste the spell-checked result back into the Text Box.

Here's the finished Automation routine:

```

Sub cmdSpellchck_Click ()
  On Error Resume Next

  Dim wordobj As object

  Set wordobj = CreateObject("Word.Basic")

  clipboard.Clear           'clear anything currently in the clipboard

  clipboard.SetText text1.Text, 1  'take the text from your text box and put it in
                                   'the clipboard

  wordobj.FileNew
  wordobj.EditPaste
  wordobj.ToolsSpelling
  wordobj.EditSelectAll
  wordobj.EditCopy
  wordobj.FileClose 2

  Set wordobj = Nothing       'destroy your word object
  text1.Text = clipboard.GetText() 'get the text back in your text box
End Sub

```

Now this may seem a little complicated for a beginners book and it probably is, That's why I added it at the end. But you can see where having your application "borrow" the functionality of other applications can be a very powerful programming technique.

Appendix F: The 10 Commandments of Visual Basic

1. Thou shalt comment heavily. Not only on the first through sixth days, but even on the seventh day, thou shalt comment thy code.
2. Thou shalt name thy variable with sensible names. Names like `iAccountNum`, `sLastName`, these shalt thou use, and thou shalt avoid confusing names like `I`, `x`, and `s`.
3. Thou shalt not use the latest technology or buzzword without some good reason and a good understanding of it.
4. Thou shalt write easy to read code with good indentations.
5. Thou shalt use proper naming conventions for all controls and variables.
6. Thou shalt Always use the “option explicit”
7. Thou shalt avoid nested loops that are too deep
8. Thou shalt strive to create code that other programmers can read and understand
9. Thou shalt strive for a user friendly interface.
10. Thou shalt go forth and program fruitfully!!