Mount Kenya University

BBIT

BBIT 3206 : EVENT DRIVEN PROGRAMMING

AUTHOR : Njuguna Patrick
Phone:0721238570
email : rpwnjuguna@gmail.com

Course content

INTRODUCTION

- The Visual Basic 6 environment
- Defining terms
- Creating a Visual Basic Project
- Practice project - Building a Football Scoreboard

APPLICATION DEVELOPMENT

- Improving the VB application
- Using a step-by-step approach
- Writing a VB procedure
- Calling procedures

BUILDING BLOCK OF VB 6

- Using the Visual Basic 6 code editor
- Adhering to programming standards
- Data types, variables and constants in Visual Basic
- Using operators
- Control structures - IF...THEN, Select Case, DO...LOOP, FOR...NEXT
- Practice assignment - Upgrading the Scoreboard

DESIGNING VB APPLICATION

- Designing the Visual Basic Application
- Working with users
- Guiding principles
- Choosing a Visual Basic interface style

DEVELOPING USER INTERFACE

- Defining the Visual Basic Form
- Standard controls: Picture, Frame, CommandButton, Label, TextBox, CheckBox, etc.
- Visual Basic practice assignment: Creating a Payroll Form

- Arrays
- More controls: ListBox, ComboBox
- Properties and Methods of objects in Visual Basic
- Building a file search application: DriveListBox, DirListBox, FileListBox
- Building a Menu

- Debugging Visual Basic code

- Error trapping

ENHANCING VB APPLICATION WITH CONTROLS

- Manipulating text - string functions
- Visual Basic functions for dates, numbers
- Using the Windows Clipboard and Screen objects
- Creating Copy, Paste, Cut, Delete functions
- Pictures, Graphics and Drawing controls in Visual Basic
- Multimedia - incorporating sounds and pictures
- Building a CD player in code

FILE ACCESS

- Working with Visual Basic files
- Writing and reading a Sequential-access file
- Sample project: the Address Book
- Creating a sequential output form
- Creating and using a Random-access file

DATABASE PROGRAMMING

- Creating a Microsoft Access database - refer to SQL tutorial
- The Project Management example
- The Data control
- Visual Basic Bound controls - TextBox controls linked to database
- Validating data - ensuring database integrity
- Finding a specific record in the database
- Using multiple tables from the database
- Creating multiple data controls
- Using Data Bound List Controls
- Using VISDATA - the Visual Basic Data Manager

Chapter 1: INTRODUCTION

Learning outcome.

At the end of this topic learner should be able to:

- Use basic terms in application development
- Understand visual basic 6.0 development environment
- Develop a basic application

1.1 Defining basic terms

Application

An application is a collection of objects that work together to accomplish something useful. In Visual Basic(VB) the application is called a **Project**. A Project could be a the management of student records, banking application, Video store, the calculation of mortgages, a booking service or the Payroll system for employees etc.

Object

An object is a piece of software that has properties and functions that can be manipulated. Whew! You're here so, you must be somewhat familiar with the Windows environment. A window is an **object**. It has **properties**: size, color, position on the screen, etc. (The purists among you may want to talk about a **class** rather than an **object** but, at this point we just want to keep it simple, and the underlying concept is the same). The window has functions, also called **methods**, that can be manipulated: change the size, move it around, open it and close it. You do not have to write code to resize a window - you just click and drag. But somebody had to write code at some point. Fortunately for us, when they did they put it all in a nice little package and called it a ***window object***. Now, whenever you need a window in your Project you can make a copy of the window object, change its properties for color or size very easily, and paste it where you want it. Then you can use its built-in methods to open it, close it when you want or resize it whenever necessary. When you create an application using objects and combining them to produce results, you are working in an **object-oriented** environment.

Event-driven

To produce an application in COBOL, a procedural language, you write COBOL source programs, you compile them into machine code and then you run them via a control interface such as JCL. A program can contain 1000's of lines of source code and could run for hours with no human intervention. In fact, in large installations, a jobstream can consist of a dozen programs, all automatically accepting input from the previous program and producing output for the next. The programmer can be blissfully unaware that the program has run unless something catastrophic happens.

In a VB project, the processes that occur have to be associated with **events**. An event is something that happens - the user clicks on a button, a form is opened, the result of a calculation is too large. The operation is **event-driven** because everything that executes does so as the result of some kind of event. The role of the programmer is to anticipate the events and to write the code that will be executed when the event occurs. A VB application is **interactive** in the sense that the user is constantly interacting with the program. The user inputs a Customer Id, the program checks the Id in the database and immediately brings up the customer's file or displays a message that the particular Id is invalid.

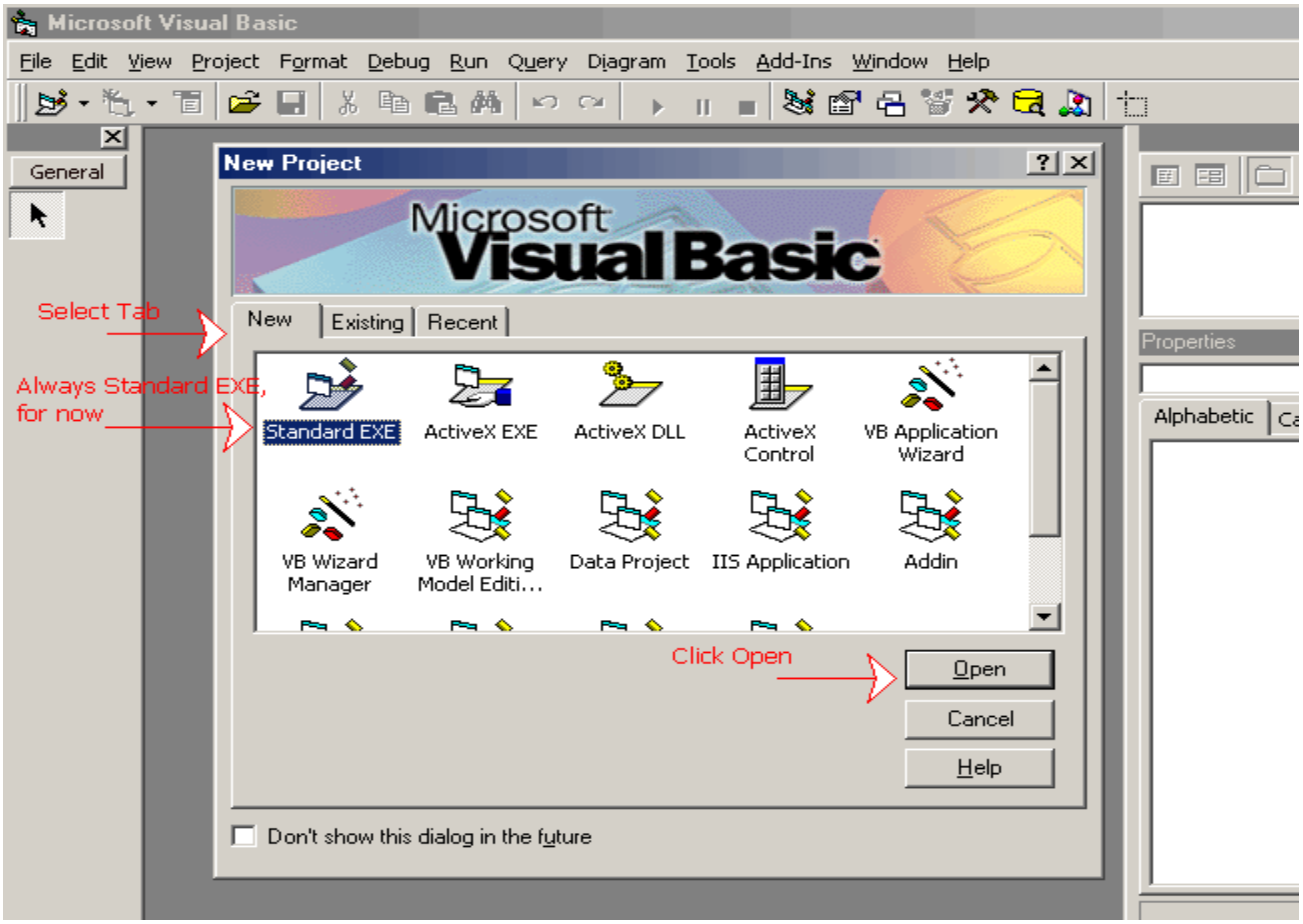
Project description

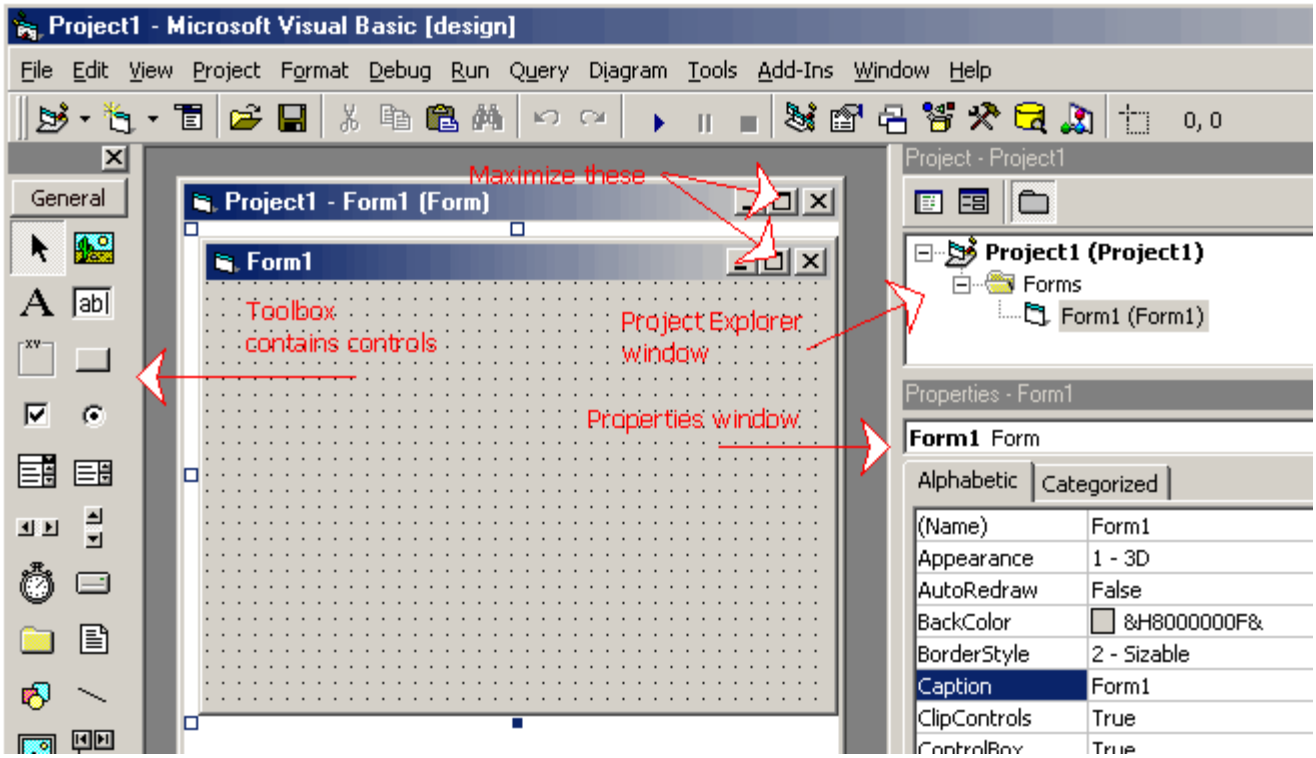
We want to create a Scoreboard for a football game (there it is already!) between the Giants and the Redskins. To begin with the simplest task we will only count the touchdowns and display appropriate messages.

Please note: although we will create a complete functional Project with controls and code and so on, the purpose of this exercise is to show what can be done. In the following lessons we will be explaining scripts and the use of controls in a lot more detail. If you study this example you should be able to relate it to what you already know of programming and judge whether this tutorial will be easy or hard for you to do.

1.2 Creating the Project

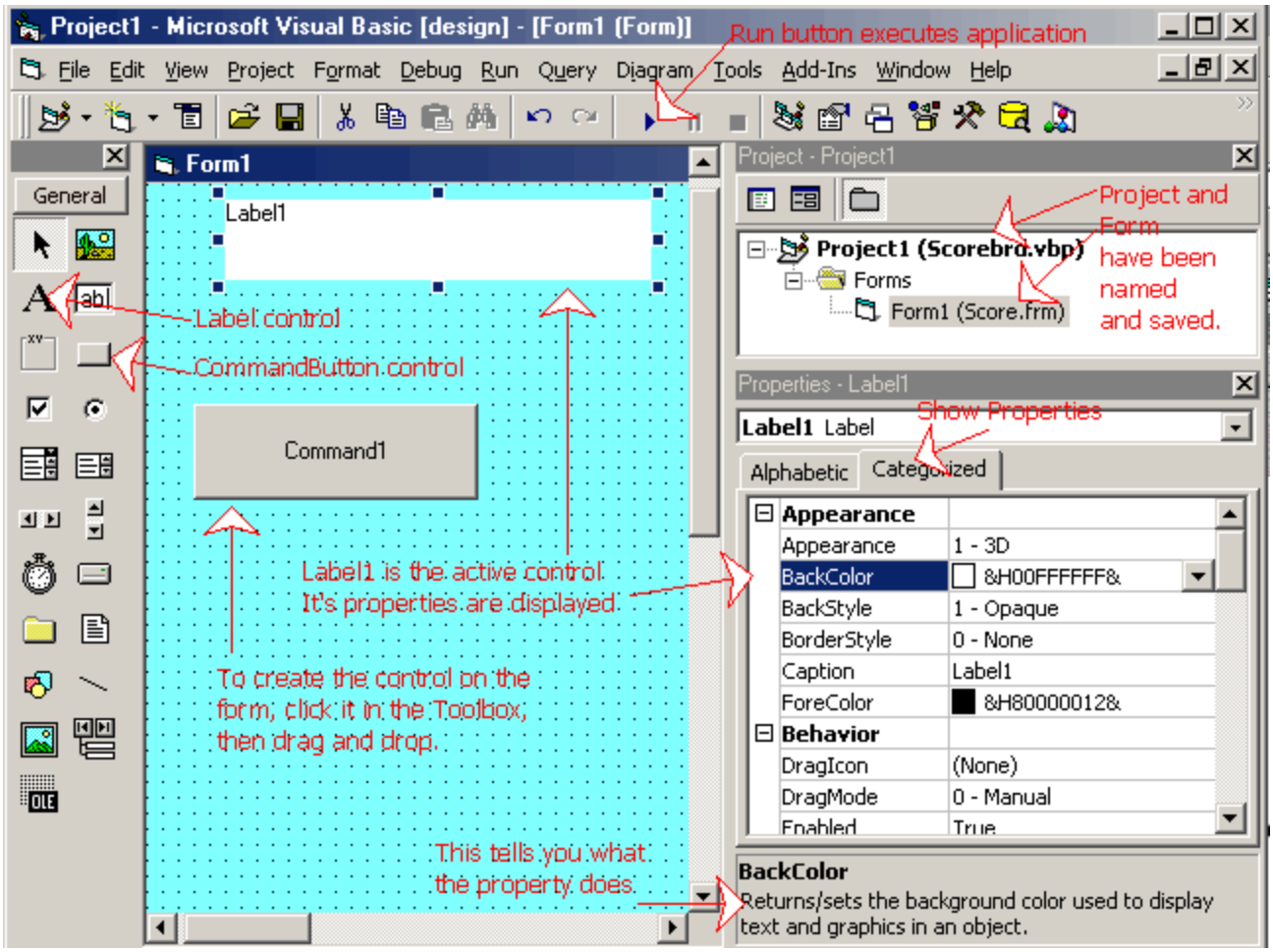
First thing to do is to create a Directory where you will store all your VB Projects. Call it VBApps, for example. Then start VB. The first screen will ask whether you want to open a new project or an existing one - it's obviously a new one and it will be a Standard EXE. Then, maximize all the windows (it's easier to work with - some of the examples in the tutorial had to be reduced for the sake of the presentation). Now, save your project. It will first ask you to save the form - call it Score.frm - and then the Project - call it Scorebrd.vbp. From now on, do File-->Save Project very, very frequently.


Microsoft Visual Basic
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
General
New Project
Microsoft Visual Basic
Select Tab
New
Existing
Recent
Always Standard EXE, for now
Standard EXE
ActiveX EXE
ActiveX DLL
ActiveX Control
VB Application Wizard
VB Wizard Manager
VB Working Model Editi...
Data Project
IIS Application
Addin
Click Open
Open
Cancel
Help
Don't show this dialog in the future
Properties
Alphabetic



Before you start to build-up the form, it will make it easier if you change the color of the form. Otherwise you will be working with grey controls on a grey background. To change the color, just click anywhere on the form, go to the properties window, find the property called BackColor and change it to the standard Window background (teal) or to any color you want in the palette.

In our first example we will need 6 **labels** and 2 **command buttons**. Each one of these objects that you put on a Form is called a **control**. To get a control you go to the **Toolbox**, click on the control you want, come back to the Form and click and drag the control to the size and position you want. Position the controls somewhat like in the diagram below.

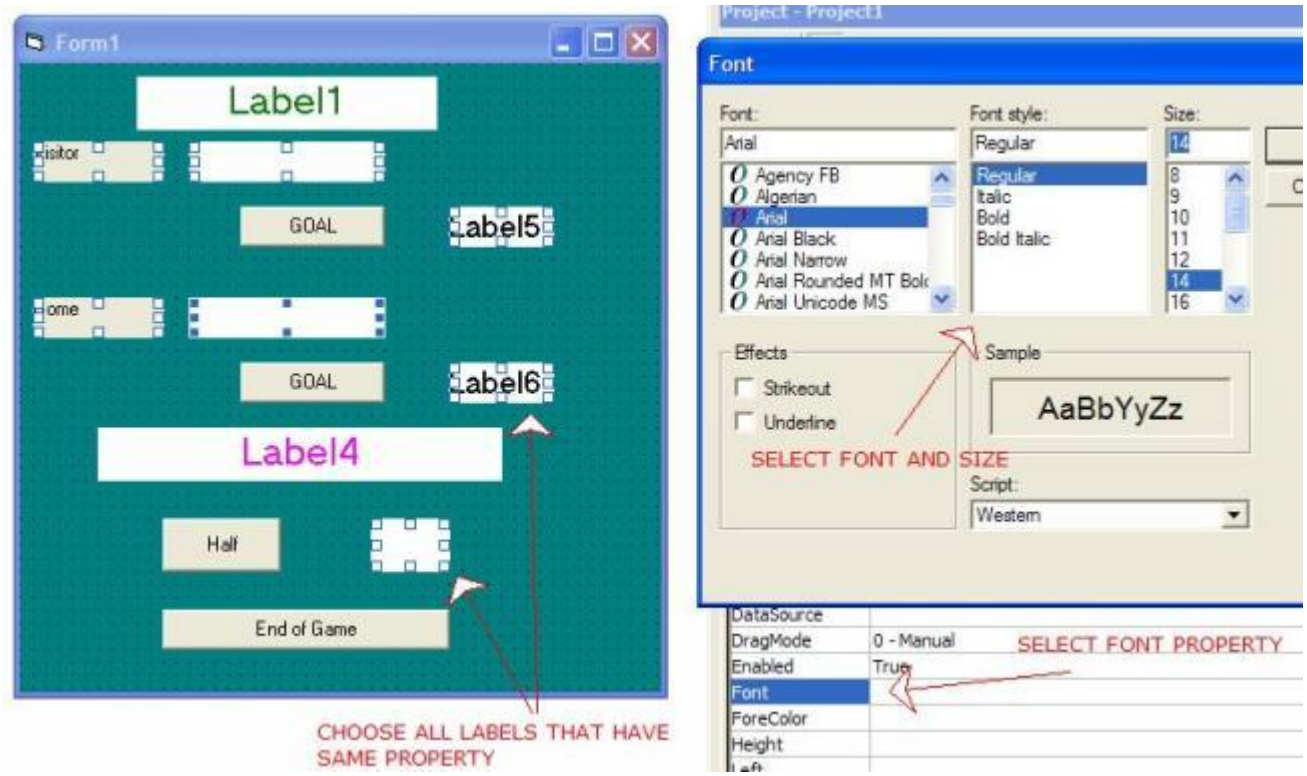


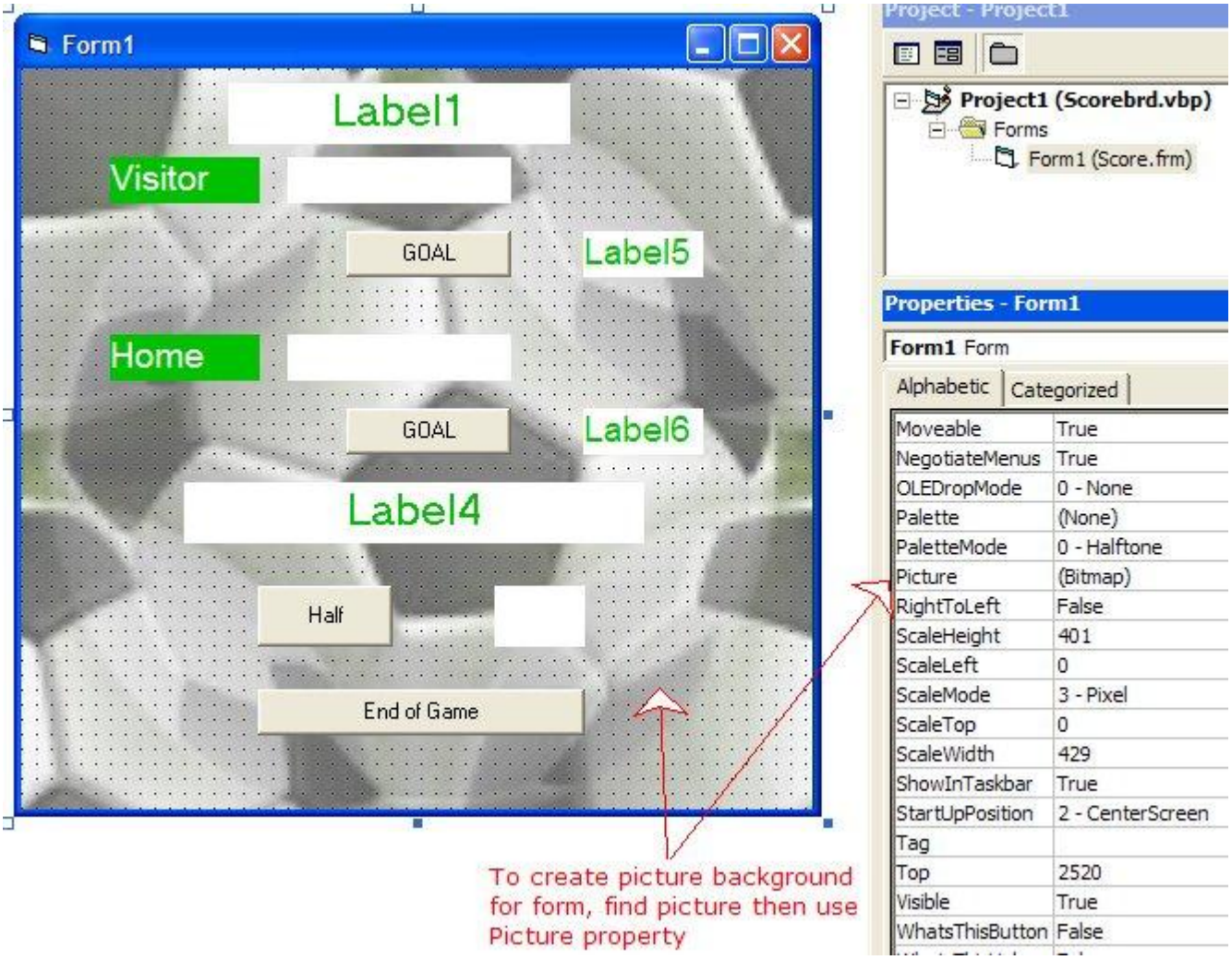
IMPORTANT NOTE: If this is your first experience with VB, don't be afraid to experiment. This is hands-on stuff! Remember that VB is a Microsoft product, therefore it works with the standard Windows interface. All the functions you know from MS-Office work the same way here: Copy, Cut, Paste, (Ctrl)+(Click), (Shift)+(Click), drag the mouse over a group of controls to select them all, etc. The Undo button is a nice one to keep handy - when you modify a control you can always Undo the change - remember this when you get to the part about aligning the controls, making them all the same size and so on. That part can get tricky. If you accidentally end up in the Code window while palying around, go down a few paragraphs and you will see how to get back to the Form. At this point the worst that can happen is that your Form will get all messed up. **So what!** You can just scrap it and start over again, but you will have learned something.

Now that we have a bunch of controls on the form, we have to jazz them up a bit. We do this by changing the **Properties** of the controls in the **Properties window**. Each control has a whole series of properties, most of which we won't need right now. The ones we do need are:

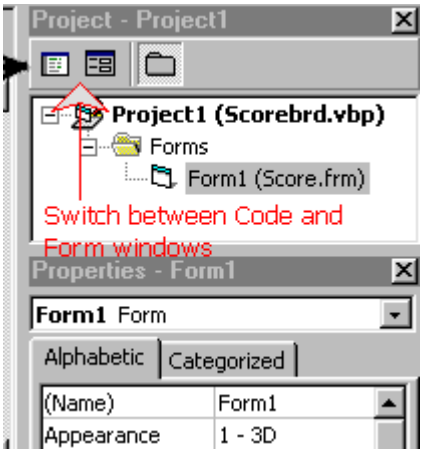
Alignment = how text aligns in the control
BackColor = choose the color of the background
Caption = the text that will appear in the control
Font = choose the font type and size
ForeColor = choose the color of the text (foreground)

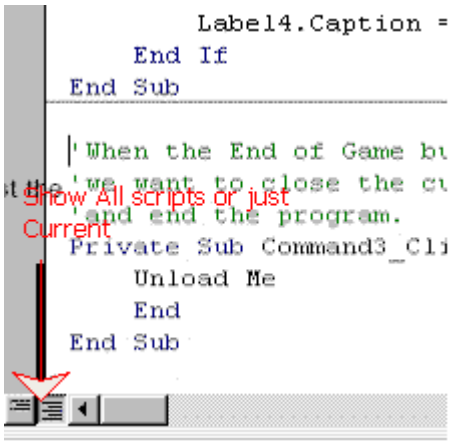
As with all Windows applications, you can select multiple controls with (Ctrl)+(Click) and change a property for all of them at once. For example, if all backgrounds are white, select all controls, change ForeColor to white and all of them are modified. Change your form to look like the one below. Note that you do not have to change the Caption for Label4, Label5 and Label6 and that you can't change the color of the buttons. They insist on being what was called in the old days "IBM grey". **Don't forget to save your project often as you go along!**





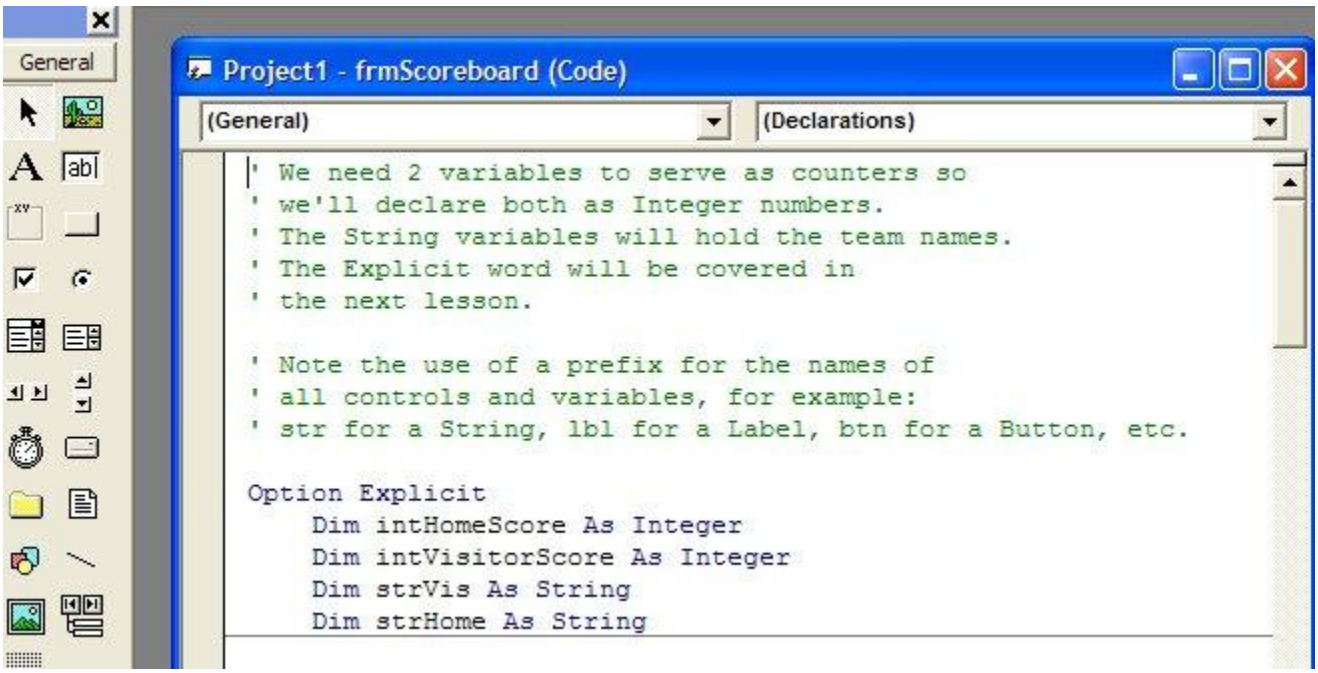
If you **Run** the application at this point, you should see your Form appear, just the way you created it. However if you click on any of the controls, **absolutely nothing happens!** There are **events** that occur; the form opens, a button is clicked, etc. But, there is nothing that tells the form what to do when it sees an event. That is why we have to write code, also called **script**.





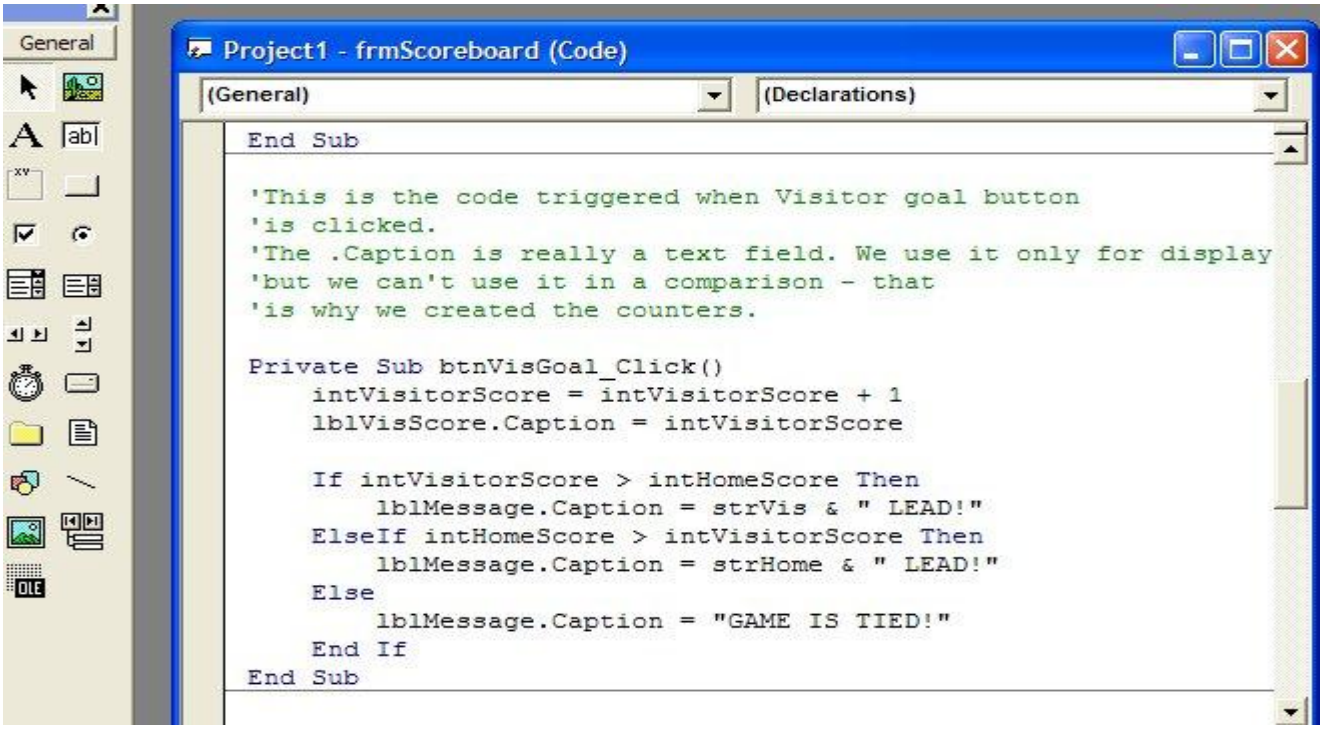
To switch between the Code window and the Form window, use the buttons just over the Project Explorer window (diagram on the left).
Once in the Code window, you have the option of seeing all the code for the Project or the code for one event at a time. Use the buttons in the lower left-hand corner (diagram on the right).
To select the object and the event you wish to code, use the two Listboxes at the top of the Code window. The one on the left for the object and the one on the right for the event. Start with **General ... Declarations** and then **Form ... Load**, etc.

At this point you might want to **download the sample program** and study it. In the following lessons we'll add functionality to the exercice and we'll explain what the code means. But for the moment, a good exercice would be to write part of the code and then try to figure out how to improve certain aspects of the program.



Now we can **Run** it and see something happen. When the Form loads, it will initialize the fields that we specified in the code.

Now code the Command1 button and Run it to see the result.



Getting to know the interface

Any program can stand to be improved, even mine!
But there's a technique to use when building a bigger and better application - you do it step-by-step.

Improving the application

Assuming that you're a beginner with VB, here's the development technique I recommend for you:

- write the simplest program that you understand and make it work - even if it doesn't have color or fancy fonts, test it and then save it;
- make a copy of your previous working program and code one or two improvements in the copy - if you become stuck and can't recover, destroy the copy, go back to the previous version, make a new copy and start again;
- repeat for every improvement you make, using small steps so that if something does go wrong its easier to identify the source of the problem (if you made 12 corrections in a program and then it doesn't work, how will you know which of the changes is causing the problem?)

Avoid repeating code!

In our FootScoreboard example, there is one occasion where there are several lines of code repeated.

OK, so it's only a few lines but, in a large program that can happen a lot and it is very time-consuming both to create the code and the to maintain it.

```
Private Sub btnVisGoal_Click()
    intVisitorScore = intVisitorScore + 1
    lblVisScore.Caption = intVisitorScore

    If intVisitorScore > intHomeScore Then
        lblMessage.Caption = strVis & " LEAD!"
    ElseIf intHomeScore > intVisitorScore Then
        lblMessage.Caption = strHome & " LEAD!"
    Else
        lblMessage.Caption = "GAME IS TIED!"
    End If
End Sub

'To capture the Home goals we code for
'btnHomeGoal button.
'This is done with Copy and Paste of btnVisGoal code.

Private Sub btnHomeGoal_Click()
    intHomeScore = intHomeScore + 1
    lblHomeScore.Caption = intHomeScore

    If intVisitorScore > intHomeScore Then
        lblMessage.Caption = strVis & " LEAD!"
    ElseIf intHomeScore > intVisitorScore Then
        lblMessage.Caption = strHome & " LEAD!"
    Else
        lblMessage.Caption = "GAME IS TIED!"
    End If
End Sub
```

The way to correct that is to take all the code that repeats and put it into a separate procedure. A procedure is identified by the **Private Sub ... End Sub** lines.

Then, whenever you have to execute the code, call the procedure simply by writing its name.

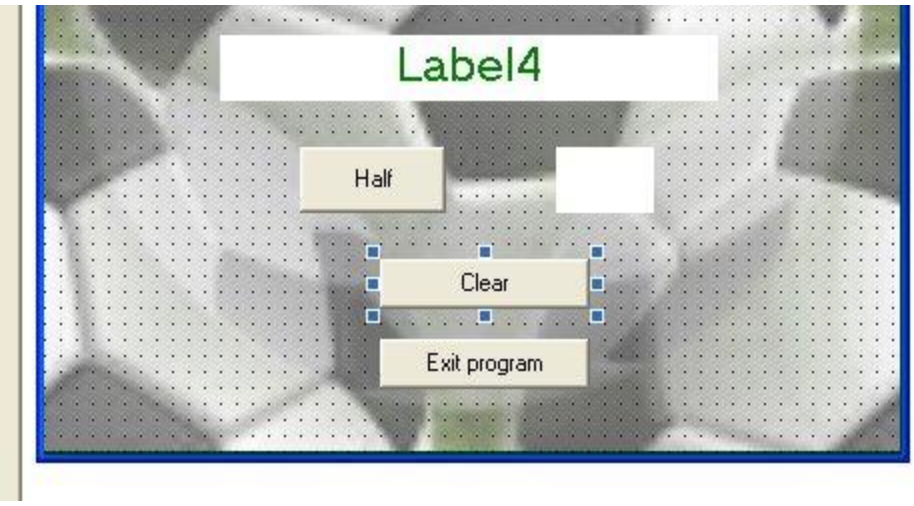
```
Private Sub btnVisGoal_Click()
    intVisitorScore = intVisitorScore + 1
    lblVisScore.Caption = intVisitorScore

    ShowScore      Call procedure
End Sub

'To capture the Home goals we code for
'btnHomeGoal button.
'This is done with Copy and Paste of btnVisGoal code.

Private Sub btnHomeGoal_Click()
    intHomeScore = intHomeScore + 1
    lblHomeScore.Caption = intHomeScore

    ShowScore
End Sub

                                Create procedure for
Private Sub ShowScore()         repetitive code
    If intVisitorScore > intHomeScore Then
        lblMessage.Caption = strVis & " LEAD!"
    ElseIf intHomeScore > intVisitorScore Then
        lblMessage.Caption = strHome & " LEAD!"
    Else
        lblMessage.Caption = "GAME IS TIED!"
    End If
End Sub
```

A second improvement

Another thing we usually need in a program is a re-initialize button.

After one loop of the program, in this case one match, we usually want to clear all the data and start over.

For that we'll create a Clear button on the form.



But, we'll notice that what we do with the Clear button is in fact the same thing we do when we load the form in the first place. So, we'll use the procedure technique to simplify the code.

```
'The Form_Load event is triggered when the Form
'opens. We put in all the stuff we want to do
'right at the beginning of the program. That's
'called "Initializing" controls and variables.

Private Sub Form_Load()
    ClearAll
End Sub

'The Clear button does the same thing as the Form_Load
'so we use the same procedure

Private Sub btnClear_Click()
    ClearAll
End Sub

Private Sub ClearAll()
    lblTitle.Caption = "SCOREBOARD"
    lblVisScore.Caption = 0
    lblHomeScore.Caption = 0
    lblMessage.Caption = "GAME IS TIED!"
    lblHalf.Caption = 1
    intHomeScore = 0
    intVisitorScore = 0
    ' Get team names and display them:
    strVis = InputBox("Enter Visitor team name")
    strHome = InputBox("Enter Home team name")
    lblVisitor.Caption = strVis
    lblHome.Caption = strHome
End Sub
```

1.3 Writing code

The Code Editor

As we saw in the previous lesson, getting to the Code Editor is as simple as hitting the proper button. You may have discovered that you can also call-up the Editor by double-clicking on an object. It is also possible to select "View code" with the right mouse button.

You will note that the Editor has all the functions of a text editor and then some. The most commonly used functions will be **Cut ... Copy ... Paste** which you can call from the Menu, from the Toolbar or from the right mouse button. You also have access to the usual **Find** and **Replace** functions ....

Getting help

There is **a lot** of documentation available on VB. There is so much, in fact, that it's easy to get lost in it. However, the on-line Help available from the Menu should be used regularly. Very often just doing a search on a word in particular will be sufficient to get you out of a jam. If you want to go into more detail check out the Contents part of MSDN (Microsoft Developers' Network) and surf through it.

Writing code

VB is not very particular about presentation - spaces, indents, lower case or upper case, it doesn't make too much difference to the compiler. But it may make a whole lot of difference to the programmer who has to maintain your code in 2 years, after you've moved up to President.

Apply "Best Programming Practices"

/hen you work with RAD (Rapid Application Development) tools like VB in a graphical interface ıvironment, you become more than just a programmer, a writer of code. You are a developer. We will cove ıat in the next lesson.

ut at the moment, you are still a Programmer. And unless you are developing an application for your own ersonal use, that nobody else will see, you have to think of the environment, of the team you are working ith.

"No man (or woman) is an island!"

specially when it comes to programming. The code you write may have to be checked by an Analyst. It wil ıve to go through testing. It may have to be modified by other team members and it almost certainly will go rough modifications, maybe several times, in the months and years ahead when you probably won't be ound to defend yourself. "The evil that men do lives after them...". You do not write code for the VB ompiler. You write it for other developers and programmers. What you want others to say behind your back : "That Jane was blindingly efficient, brilliant, a genius with comments ..."

1.4 Good programming practice/habits

1. Use comments when appropriate but not so many as to overwhelm the code; the apostrophe ' is the comment identifier; it can be at the beginning of a line or after the code.

   ```
   ' This is a comment
   'on 2 lines
   DIM intNumber AS Integer      'This is a comment
   ```

2. Use indents - code must be indented under control structures such as If ... Then or Sub - it makes it so much easier to follow the logic.

   ```
   FOR i = 1 TO 5
       value(i) = 0      ' Indent used in control structures
   NEXT i
   ```

3. Use standard capitalization - keywords like If, Dim, Option, Private start with a capital letter with the rest in lower case; variable names, control names, etc. are usually mixed case: ClientName, StudentId, etc.

4. Write extra-long statements on 2 lines using the continuation character _ (space underscore); in VB each line is assumed to be an individual statement unless there is a continuation at the end of the first line.

   ```
   Data1.RecordSource = _
     "Select * From Titles"      ' One statement on 2 lines is OK
   ```

1.4 Naming conventions

These are the rules to follow when naming elements in VB - variables, constants, controls, procedures, and so on:

- A name must begin with a letter.
- May be as much as 255 characters long (but don't forget that somedy has to type the stuff!).
- Must not contain a space or an embedded period or type-declaration characters used to specify a data type ; these are ! # % $ & @
- Must not be a reserved word (that is part of the code, like Option, for example).
- The dash, although legal, should be avoided because it may be confused with the minus sign. Instead of Family-name use Family_name or FamilyName.

Chapter 2 Data types and operators

Data types are set of values and permitted/allowable operations on those data.

2.1 VB supports variety of data types.

Data type	Storage size	Range
Byte	1 byte	0 to 255
Boolean	2 bytes	True or False
Integer	2 bytes	-32,768 to 32,767
Long (long integer)	4 bytes	-2,147,483,648 to 2,147,483,647
Single (single-precision floating-point)	4 bytes	-3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values
Double (double-precision floating-point)	8 bytes	-1.79769313486232E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values
Currency (scaled integer)	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	14 bytes	+/-79,228,162,514,264,337,593,543,950,335 with no decimal point; +/-7.9228162514264337593543950335 with 28 places to the right of the decimal; smallest non-zero number is +/-0.0000000000000000000000000001
Date	8 bytes	January 1, 100 to December 31, 9999
Object	4 bytes	Any Object reference
String (variable-length)	10 bytes + string length	0 to approximately 2 billion
String (fixed-length)	Length of string	1 to approximately 65,400
Variant (with	16 bytes	Any numeric value up to the range of a Double

numbers)		
Variant (with characters)	22 bytes + string length	Same range as for variable-length String
User-defined (using Type)	Number required by elements	The range of each element is the same as the range of its data type.

In all probability, in 90% of your applications you will use at most six types: String, Integer, Long, Single, Boolean and Date. The Variant type is often used automatically when type is not important. A Variant-type field can contain text or numbers, depending on the data that is actually entered. It is flexible but it is not very efficient in terms of storage.

2.2 Declaring variables

Declaring a variable means giving it a name, a data type and sometimes an initial value. The declaration can be **explicit** or **implicit**.

An **explicit declaration:** variable is declared in the Declarations Section or at the beginning of a Procedure. An explicit declaration looks like:

Dim MyNumber As Integer

Now the variable **MyNumber** exists and a 2-byte space has been reserved for it.

An **implicit declaration:** the variable is declared "on the fly", its data type is deduced from other variables. For example:

Dim Total1 As Integer 'Explicit declaration
Dim Total2 As Integer 'Explicit declaration
Total3 = Total1 + Total2 **'Implicit declaration**

Total3 is not formally declared but is implied, it is "arrived at" from the other declarations.

It is never a good idea to have implicit declarations. It goes against the rules for clarity, readability and ease of use of the code.
To make sure that this rule is followed, start the Declarations with the **Option Explicit** clause. This tells the compiler to consider implicit declarations as errors and forces the programmer to declare everything explicitly.

Other examples of declarations:
Dim MyName As String
Dim StudentDOB As Date
Dim Amount5, Amount6, Amount7

In the last example the type assigned to each variable will be: Variant. It is the default type when none is

specified.

There can be multiple explicit declarations in a statement:

Dim EmpName As String, SalaryMonth As Currency, SalaryYear As Currency

In this final example, what are the types assigned to the three variables:

Dim Amount1, Amount2, Amount3 As Single

All Single-precision floating point, you say. **Wrong!** Only Amount3 is Single. Amount1 and Amount2 are considered Variant because VB specifies that each variable in a statement must be explicitly declared. Thus Amount1 and Amount2 take the default data type. This is different from what most other languages do.

2.3 Constants

A constant is a value that does not change during the execution of a procedure. The constant is defined with:

Const ValuePi = 3.1416

The Scope of variables

The term **Scope** refers to whether the variable is available outside the procedure in which it appears. The scope is **procedure-level** or **module-level**.

A variable declared with Dim at the beginning of a procedure is only available in that procedure. When the procedure ends, the variable disappears. Consider the following example:

```
Option Explicit
    Dim Total2 As Integer

Private Sub Command1_Click ()
    Dim Total1 As Integer
    Static Total3 As Integer
    Total1 = Total1 + 1
    Total2 = Total2 + 1
    Total3 = Total3 + 1
End Sub

Private Sub Command2_Click ()
    Dim Total1 As Integer
    Total1 = Total1 + 1
    Total2 = Total2 + 1
    Total3 = Total3 + 1
End Sub
```

Every time Button1 is clicked, Total1 is declared as a new variable during the execution of that clicked

event. It is a **procedure-level** variable. It will always stay at 1. The same for the Button2 event: Total1 is a new variable in that procedure. When the procedure ends, Total1 disappears.
Total2 is declared in the Declarations section. It is a **module-level** variable, meaning it is available to every control in this Form. When Button1 is clicked, it increments by 1 and it retains that value. When Button2 is clicked, Total2 is incremented from its previous value, even if it came from the Button1 event.
Total3 shows another way of retaining the value of a local variable. By declaring it with **Static** instead of Dim, the variable acts like a module-level variable, although it is declared in a procedure.

Another scope indicator that you will see when you study examples of code is **Private** and **Public**. This determines whether a procedure is available only in this Form (module) or if it is available to any module in the application. For now, we will work only with Private procedures.

2.4 Operators

2.4.1 Mathematical and Text operators

Operator	Definition	Example	Result
^	Exponent (power of)	4 ^ 2	16
*	Multiply	5 * 4	20
/	Divide	20 / 4	5
+	Add	3 + 4	7
-	Subtract	7 - 3	4
Mod	Remainder of division	20 Mod 6	2
\	Integer division	20 \ 6	3
&	String concatenation	"Joan" & " " & "Smith"	"Joan Smith"

Note that the order of operators is determined by the usual rules in programming. When a statement includes multiple operations the order of operations is:
Parentheses ( ), ^, *, /, \, Mod, +, -

2.4.2 Logical operators

Operator	Definition	Example	Result
=	Equal to	9 = 11	False
>	Greater than	11 > 9	True

<	Less than	11 < 9	False
>=	Greater or equal	15 >= 15	True
<=	Less or equal	9 <= 15	True
<>	Not equal	9 <> 9	False
AND	Logical AND	(9 = 9) AND (7 = 6)	False
OR	Logical OR	(9 = 9) OR (7 = 6)	True

Chapter 3 : Control Structures

3.1 Selection

3.1.1 <u>If...Then...Else</u>

```
If condition1 Then
    statements1
Else
    statements2
End If
```

If condition1 is True, then statements1 block is executed; Else, condition1 is not True, therefore statements2 block gets executed. The structure must be terminated with the End If statement.

The Else clause is optional. In a simple comparison, statements1 get executed or not.

```
If condition1 Then
    statements1
End If
```

3.1.2 Select Case

Can be used as an alternative to the **If…Then…Else** structure, especially when many comparisons are involved.

```
Select Case ShirtSize
    Case 1
        SizeName.Caption = "Small"
    Case 2
        SizeName.Caption = "Medium"
    Case 3
        SizeName.Caption = "Large"
    Case 4
        SizeName.Caption = "Extra Large"
    Case Else
        SizeName.Caption = "Unknown size"
End Select
```

3.2 Repetition /loop/ iteration

3.2.1 Do...Loop

Used to execute a block of statements an unspecified number of times.

Do While condition
    statements
Loop

First, the condition is tested; if condition is True, then the statements are executed. When it gets to the Loop it goes back to the Do and tests condition again. If condition is False on the first pass, the statements are never executed.

3.2.2 For...Next

When the number of iterations of the loop is known, it is better to use the For...Next rather than the Do...Loop.

For counter = start **To** end
    statements
Next

1) The counter is set to the value of start.
2) Counter is checked to see if it is greater than end; if yes, control passes to the statement after the Next; if not the statements are executed.
3)At Next, counter is incremented and goes back to step 2).

More will be covered on Control strucures as it becomes necessary in upcoming lessons. Meanwhile,if you want to know more, consult the VB Language Reference.

Assignment

To practise your coding and editing skills, try modifying the Football example by adapting it for different sports. For example, in American football, which is similar to rugby, there are 4 different ways to score, as shown here:

Touchdown	6 points
Field goal	3 points
2-point Convert or Safety	2 points
Convert	1 point

Chapter 4 Designing Application

Introduction

When you start to work on a VB Project you are no longer just a programmer - you are now a developer. You will have to get much more involved in the whole design process. Unless you are designing an application for your own use you will have to work with a team of specialists including, but not limited to, **users, analysts, GUI designer, programmers, testers, network specialist, webmaster and marketing people**. The whole process is iterative - do part of it, check it, get input, go back and correct it, do the next part, and so on. Nobody expects you to do a whole project in one fell swoop - it would probably be a disaster if you did do it that way.

The importance of Users

Any project that you develop has to involve **Users.** They are the people who will sit in front of your interface for eight hours a day and decide if they like it or not. If they don't like it, no matter how efficient the code and how many millions of dollars were spent developing it, they will find ways to sabotage it.

Get users involved from the start. If you are developing a product to specs, that is to be sold to some client eventually, there has to be someone who knows what that eventual client needs. Find a typical user of the product to use as a sounding board. Remember: you are just the developer; no matter how cool you think it would be to use all purple text on orange backgrounds, it is the user who will tell you what is cool and what is not. As you develop more and more parts of the application, run them by the user to check for accuracy, completeness, clarity, etc.

Here's an example of how to design for **clarity**. Given that 01/02/03 is a date, what date is it? If you are an American, you probably automatically assume that it is January 2nd, 2003. If your user is French, however, he would assume that it is February 1st, 2003. And if you are working with this Professor, who has a very definite opinion on the subject, he would say that it is February 3rd, 2001 and should always be written as 2001-02-03. If all your forms are designed as: "Enter date" with a blank box beside it, you are headed for trouble.

race between software engineers striving to build bigger and better idiot-proof programs, produce bigger and better idiots. So far, the Universe is winning.

That's just a joke, by the way. Most users are not idiots. Sometimes they appear confused because they are trying to solve the problem and they can't figure out how. But that's not their job. Their job is to explain clearly what it is they need. Your job is to figure out how to provide it. Don't underestimate users. Be patient, be understanding without being condescending and be humble. There's a lot of things that the user knows how to do that you don't.

4.1 Creating the User Interface

The user interface that you design is the most visible and perhaps the most important part of the application. The term commonly used for this type of interface is: **GUI (Graphical User Interface)**. It's pronounced "goo-wee", not "guy". It is graphical because it consists of buttons, menus, icons, etc. An example of a non-GUI is DOS (remember that?) where everything is text. User interface refers to the fact that it is the part of the application between the user, in front of the screen, and the code behind the screen. How well the user can interact with the code depends on the quality of the interface.

Guiding principles

- **The user is in control**. The user must feel he is in charge of the application. He must have a certain amount of control over such things as window size, window position, choice of fonts, etc. There should definitely be a "Preferences" item in the menu.

- **Consistency is maintained throughout the application**. The user can move to any part of the application and not have to re-learn how things work. Consistency in the choice of icons, in date formats, in error messages means that the user can concentrate on the work. As much as possible, the application should be consistent with Windows standard. For example, "Move to the Recycle Bin" is different from "Delete" - the user has come to expect that an item in the Recycle Bin can be recovered if need be.

- **Application should be "forgiving", or "fault-tolerant**". Users will make mistake. A single error should not bring the application crashing to the floor. If there is no room for errors, users will be afraid to experiment, to discover on their own how to do things. It will slow the learning process considerably.

- **Always supply feedback**. The user should always know that something is going on, especially if it's in the background and may take several minutes to run. Display an hourglass or a progress meter or a status bar so that the user doesn't start to hit keys at random to get something to happen. It only takes a few seconds of inactivity for the user to get frustrated and think that the program is "hanging".

- **Don't neglect esthetics**. The visual aspect is important. The environment should be pleasing to the eye. The presentation style should help in understanding the information presented.

- **Interface should be simple without being simplistic**. There should be a balance between simplicity and functionality. Popup menus, for example, allow you to increase the functionality without having to encumber the screen with all kinds of details which are not used 95% of the time.

On the importance of language

Throughout the project you are going to be doing, you should give some thought to the quality of the language used. As a teacher of technology, I am constantly defending the compulsory language courses included in the curriculum. I have to point out that your mastery of the language, or lack thereof, projects

an image of who and what you are. This is the 21st Century - image is everything!

When I was young, a long, long time ago, in a galaxy far, far away, teachers used to say all the time: "Sloppy work is the sign of a sloppy mind!". There is a lot of truth in that. If you can't be bothered to display the interface correctly, what does that say about the rest of your work? Professionalism should be evident in every part of what you create. If what is seen by the public is of poor quality, there is reason to believe that the work behind the interface (90% of the application) is also poor.

If you are developing an application for yourself, nobody cares what it looks like. If it's a small project for a client that you know, you may be able to get away with some mistakes. Usually however, a project is broader in scope and you don't know the audience. Remember that you are now working in the global village. The interface you write and display may be seen, via the Internet, by millions and millions of critical users. Even if it's not your reputation riding on it, the reputation of your client may be. And you can be sure that he will take it seriously, even if you don't. If language is not your area of expertise, get help from somebody whose area it is.

4.2 Interface style

One of the first decisions you have to make is whether to go **SDI (Single Document Interface)** or **MDI (Multiple Document Interface).** If you have worked with Windows for any length of time, you have probably seen both. Notepad is an SDI - you can only have one document open at any time. Word is an MDI - you can open a number of documents and switch between them at will. An MDI application imposes different constraints on the developer because of the need to juggle several different forms at the same time. It usually requires more experience in the development process. For the purposes of this tutorial, we will work only with SDI applications.

Design considerations

- What is the purpose of the application? Is it a once-a-year thing or one that is in use 24/7? The user will forget the details if he only uses it once a year and you will have to be a lot more specific with the Help functions.
- Who is the intended audience? Beginning users will need more directions than experienced users.
- How many different forms will be needed (you can have several forms without being MDI) and how will they be connected?
- What are the menus going to say? Will toolbars be used to replicate menu functions?
- How are errors going to be identified to the user? How will they be corrected?
- How much Help (in the form of a Help function) is going to be provided?
- How will consistency be maintained across the application? It is important that all forms have the same "look and feel" in terms of colors, fonts, menus, toolbars, etc.

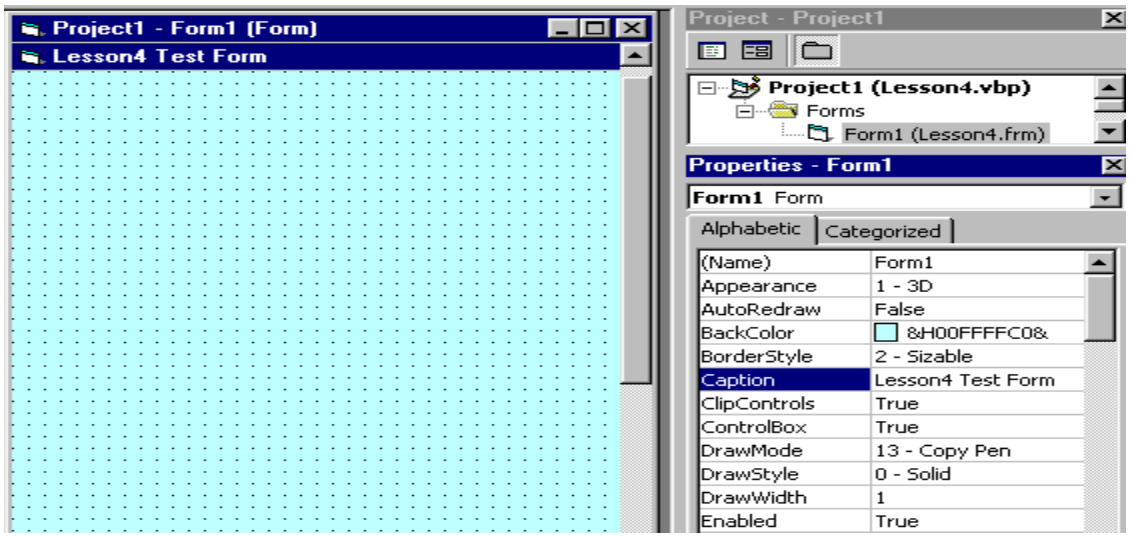
For this lesson we will need a **new Project**, call it **Lesson4.vbp**, which will only be used to create and try out various controls.

To refresh your memory, since the previous two lessons have been rather theoretical, you select the New tab, and Standard EXE for the type. As soon as The Form opens, you **Save** the new Project, **give the Form a name**, let's say it's Lesson4.frm and then you **give the Project a name:** Lesson4.vbp. Note that you do not have to specify the extensions, .frm and .vbp, because they will be assigned automatically.

Chapter exercise

The Form

We covered it too briefly in Lesson1 so we'll go over it again. The Form is the first object you see when you Open the application. It is the window into which all the controls will appear, where you will input data and see results.

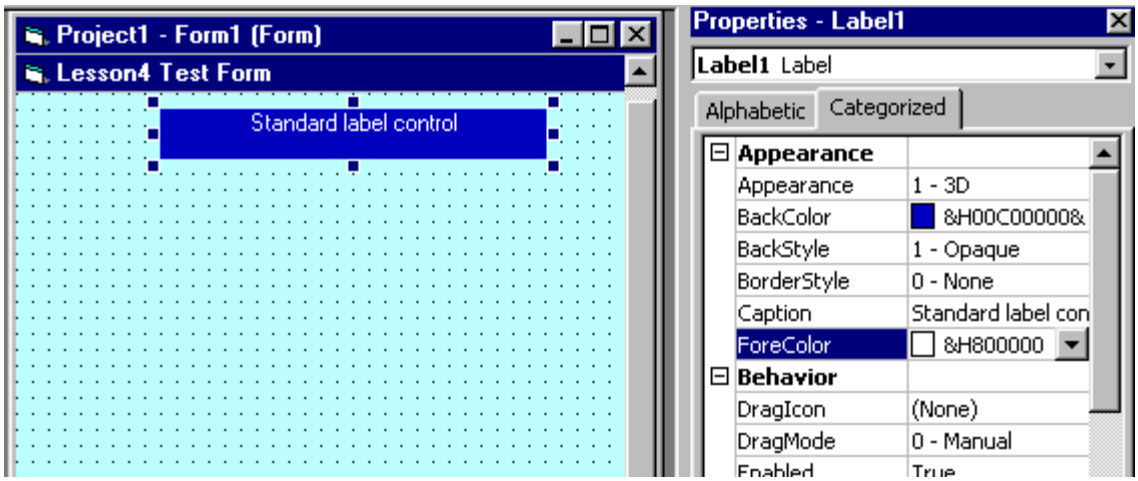


There's not too much you can do with the form, at this time. Basically, you adjust the **BackColor** and the **StartUpPosition** (where it will open on the screen when you Run it) and then you start putting controls on it.

The Label

This is probably the first control you will master. It is used to display static text, titles and screen output from operations. The important properties to remember:

- Caption - the text that is displayed in the label
- BackColor and ForeColor - colors of the background and the text
- BackStyle - Opaque or Transparent - whether the background is visible or not
- Font - font and size of text
- Alignment - text centered, left or right
- Multiline- True or False - if True, you can have several lines of text, delimited by <CR> in the label - by default, it is set to False

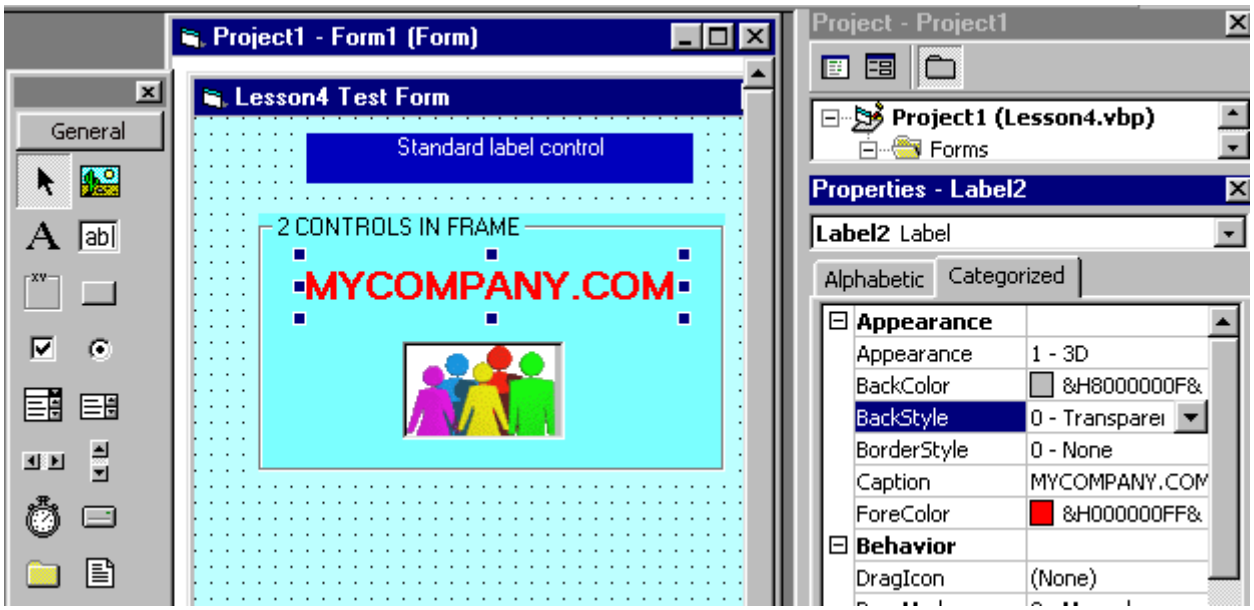


Frame & PictureBox

When you want to group several controls together - name and address, for example - you use a **Frame**. The frame backcolor can be the same as the form's and only the frame borders will be obvious, or it can be a different color and stand out.

You create the frame before the controls. When you create controls in a frame, they are tied to the frame and move with it. The frame caption is the text that appears at the top of the frame - you use it to define the group.

The **PictureBox** is like a Label with a picture in it instead of text. The **Picture property** determines the name of the file, .BMP or .GIF, that will be displayed. It can be used for a company logo, etc.



Top

TextBox & CommandButton

The TextBox is like a Label but, it is used to input data into the program. The data typed in is in the **Text** property of the control.

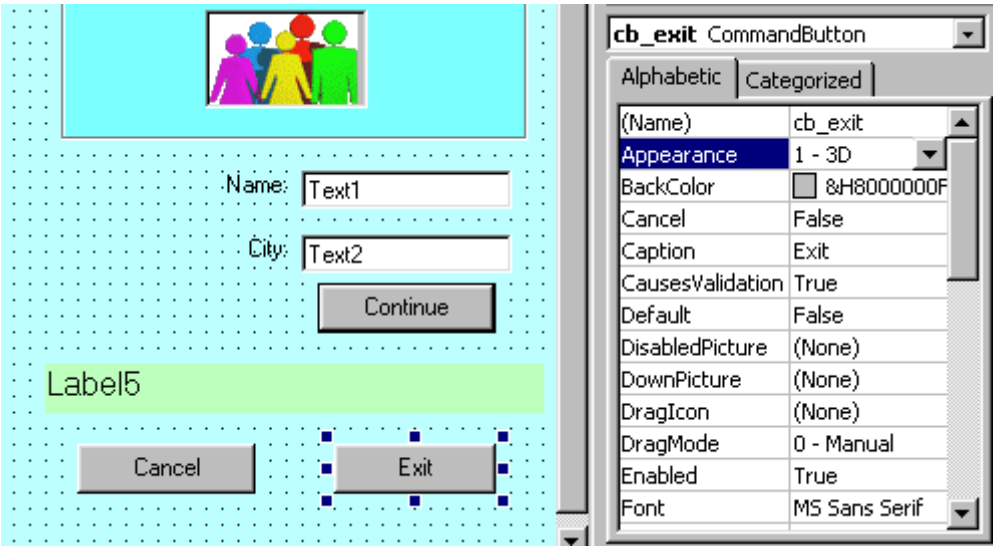
When the program is Run, only the controls that can be manipulated will be activated. For example, if the form contains 3 Labels, 3 TextBoxes and 3 Buttons, when it is Run, the cursor will not stop at the labels.

When the user hits the Tab key, the cursor will go to the first TextBox or Button - not necessarily the first one on the form but, the first one that was created. That is called the **Tab order** and you have to specify it.

On the form there is only one control at any given time that has the cursor on it - it is said to have **Focus**. If you type data, the control with Focus will receive it. You change the Focus with Tab or by clicking on a different control.

> Up until now we haven't bothered with the names of controls (the Name property). Once we start to code, however, it does become important. There are all kinds of occasions in code where you have to call upon a certain control. It can get very confusing when your 12 buttons are called Command1...Command12. What did Command7 do, again? Give each control a name (except for titles, etc. that you never refer to) so that you will be able to identify it easily. It is recommended that you use a prefix when assigning a name; cmd for a CommandButton, lbl for a Label, txt for a TextBox. Thus, txtNumber where you input the value can be distinguished from lblNumber where you display the result.

The CommandButton is used to initiate actions, usually by clicking on it. The Caption property determines the text to display on the face of the button. The **Default** property, if set to true, means that the button will be activated (same as Clicked) if the <Enter> key is hit anywhere in the form. If **Cancel** is set to True, the button will be activated from anywhere in the form by the <Esc> key.



Hopefully, you have now run this program several times, each time you added a new control, in fact. Admittedly, nothing much happened except to confirm that the controls were appearing in the right place on the form.

Here now is an example of the code we could write to perform simple tasks: input name and city and display the information in a label when the Continue button is clicked. The Exit button will end execution of the program and the Cancel button (or the Esc key) will clear the fields.

```
(General)                                   (Declarations)

Option Explicit
    Dim ClientName As String
    Dim ClientCity As String

Private Sub Form_Load()
    lb_result.Caption = " "
    tb_name.Text = " "
    tb_city.Text = " "
End Sub

Private Sub cb_continue_Click()
    ClientName = tb_name.Text
    ClientCity = tb_city.Text

    lb_result.Caption = "Client is " & ClientName _
        & " and lives in " & ClientCity
End Sub

Private Sub cb_cancel_Click()
    Form_Load
End Sub

Private Sub cb_exit_Click()
    Unload Me
    End
End Sub
```

A few explanations: the Form_Load event occurs when the form first opens. This is where we initialize things - we want the TextBoxes and the result Label to be empty when we start off so, we set them to a blank space.

The actual processing is done after the data have been entered and we hit the Continue button. The processing logic is put in the Continue_button_clicked event.

When you hit the <Esc> key or you click on the Cancel button, you want to annul the entry you're doing and start over again. That's the same as opening the form so, we just tell the program to execute the Form_Load procedure which we already have.

The Exit button uses the **pronoun Me** to Unload. **Me** means **the form currently active** without having to name it.

Multiple forms

For the next series of examples we will use a new Form. It is not necessary to create a new Project; any Project can have several Forms in it.

With the Project open on the current Form, go to: Menu --> Project --> Add form. Select New form in the creation window and, voila! Next time you save the Project, it will ask you to name this new Form. You can name it Lesson4B.frm for example.

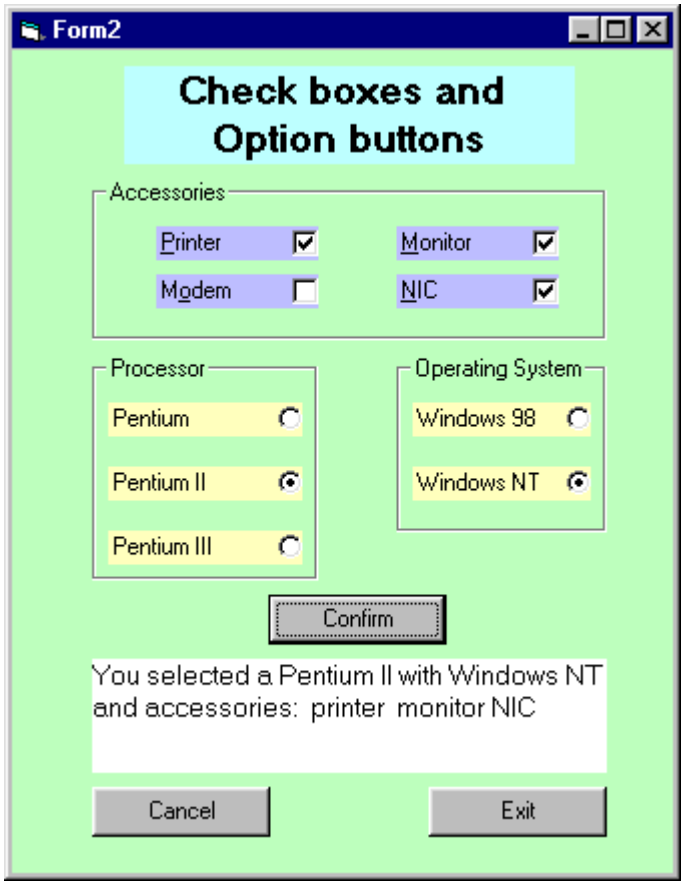
One more detail: when you Run the Project, you want to tell it which Form to open.

Go to the **Project Manager window**, right click on the Project name and select **Project properties**. In the Project properties window, the ListBox on the right says "**Startup object**". Select the Form you want to open when you Run. You can change the Startup object at any time to run the different forms that you created.

Check boxes & Option buttons

These two controls are used when the user must choose from a list of options. The difference between the two is that with Check boxes he can select several at one time but, with Option buttons, he must select only one among several.

The example below illustrates the use of Check boxes and Option buttons. To capture the information entered by means of these controls, you must test the **property: Value.** In a Check box, **Value = 1 if box is checked and = 0 if not.** In an Option button, **Value = True if selected and = False if not.**



The code, although somewhat long, is not very complicated. Most of the time, processing consists of

checking the content of .Value. It is fairly standard procedure, especially with Option buttons, to ensure that at least one button has been selected and to display an error message if it has not.

```
cb_confirm                                  Click

Option Explicit

Private Sub Form_Load()
    ck_printer.Value = 0
    ck_monitor.Value = 0
    ck_modem.Value = 0
    ck_nic.Value = 0
    opt_pent.Value = False
    opt_pent2.Value = False
    opt_pent3.Value = False
    opt_win98.Value = False
    opt_winnt.Value = False
    lb_msg.Caption = ""
End Sub
```

```
cb_confirm                                  Click

Private Sub cb_confirm_Click()
    Dim PrName As String, OsName As String
    Dim AccPr, AccMn, AccMod, AccNic

    'Check if Processor was selected - if no
    'display error message; if yes, get its name.
    If opt_pent.Value = False _
            And opt_pent2.Value = False _
            And opt_pent3.Value = False Then
        MsgBox ("You must select a Processor")
        opt_pent.SetFocus
    Else
        If opt_pent.Value = True Then
            PrName = "Pentium"
        ElseIf opt_pent2 = True Then
            PrName = "Pentium II"
        Else
            PrName = "Pentium III"
        End If
    End If
```

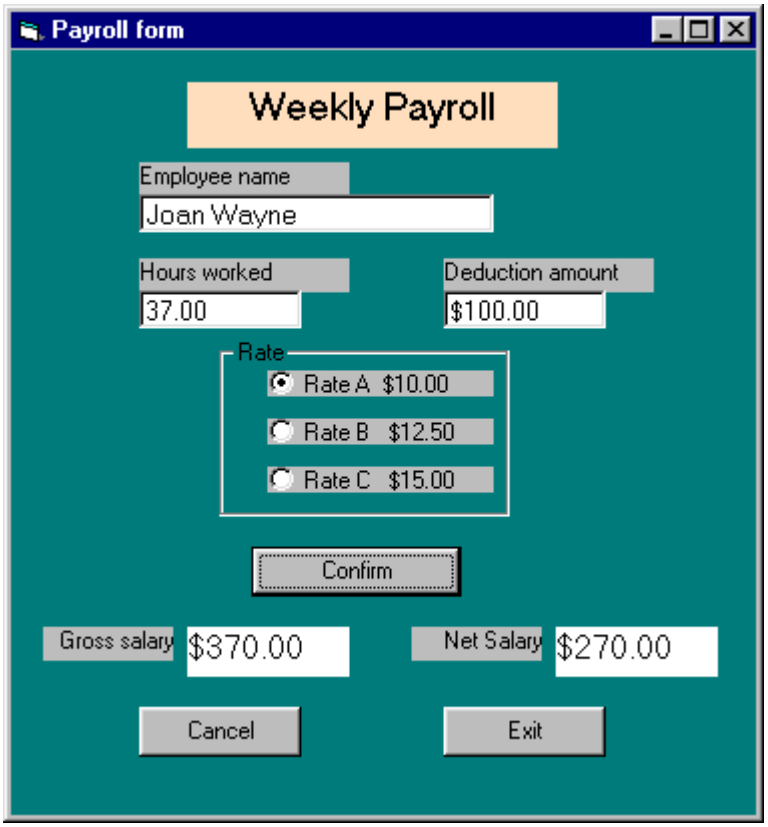
```
    'The MsgBox function allows you to display
    'a message window as a result of some error.
    'See "MsgBox function" in Help for details.

    'SetFocus is a Method that lets you return Focus
    '(the cursor) to a specified object, in this case
    'one of the option buttons, after an error check.
    'See "SetFocus Method" in Help.

    'Check if OS was selected - if no
    'display error message; if yes, get its name.
    If opt_win98.Value = False _
            And opt_winnt.Value = False Then
        MsgBox ("You must select an Operating system")
        opt_win98.SetFocus
    Else
        If opt_win98.Value = True Then
            OsName = "Windows 98"
        Else
            OsName = "Windows NT"
        End If
    End If
```

```
    'Verify which accessories were checked in order
    'to build output label.
    If ck_printer.Value = 1 Then
        AccPr = " printer "
    End If
    If ck_monitor.Value = 1 Then
        AccMn = " monitor"
    End If
    If ck_modem.Value = 1 Then
        AccMod = " modem"
    End If
    If ck_nic.Value = 1 Then
        AccNic = " NIC"
    End If

    lb_msg.Caption = "You selected a " & PrName _
      & " with " & OsName & Chr(13) _
      & "and accessories: " & AccPr & AccMn _
      & AccMod & AccNic

     'If you want to force a line change in a Label,
     'insert a Chr(13) -the carriage return character-
     'in the string.
End Sub
```

```
Private Sub cb_cancel_Click()
    Form_Load
End Sub

Private Sub cb_exit_Click()
    Unload Me
    End
End Sub
```

Assignment 3

Create the Payroll form shown below. Number of hours must be entered as well as the appropriate rate. Gross salary = rate * hours. Net salary = gross salary - deductions.



LESSON 5 - More standard controls

Tuesday, August 02, 2011

Working with arrays

Before we get to today's lesson on common controls, we will cover a bit of programming theory on Arrays.
In VB, arrays work in much the same way as they do in all other languages you have studied. By definition an array is **an indexed variable**, meaning it is one variable with many parts, each part being referenced by an index number. The index number being numeric, it can be manipulated by loop statements, incremented, decremented, etc. An array can contain any valid data type and, if it is of the Variant type, can even contain elements of different types.

An array is declared like any other variable, with the addition of an index:

Dim Department(6) As String

will declare an array of 7 elements of the String type (we assume that it will be 7 Department names). The only problem with this declaration is that the index goes from 0 to 6. So, if you want the name of the sixth Department you have to specify Department(5), which can be confusing at times.
To work around this problem you can specify the starting index in the declaration:

Dim Months(1 To 12) As String

Thus, Months(1) is January and Months(12) is December.
You don't even have to start at 1. If your Sections are numbered 101 - 120 you can define an array:

Dim Sections(101 To 120) As Integer

One common method of manipulating arrays is to use For...Next loops:

```
Private Sub Command1_Click()
    Dim Icnt As Integer           'declare loop counter
    Dim DeptTotal(1 To 20) As Single  'declare array

    For Icnt = 1 To 20          'initialize
        DeptTotal(Icnt) = 0     'all array elements to 0
    Next Icnt

End Sub
```

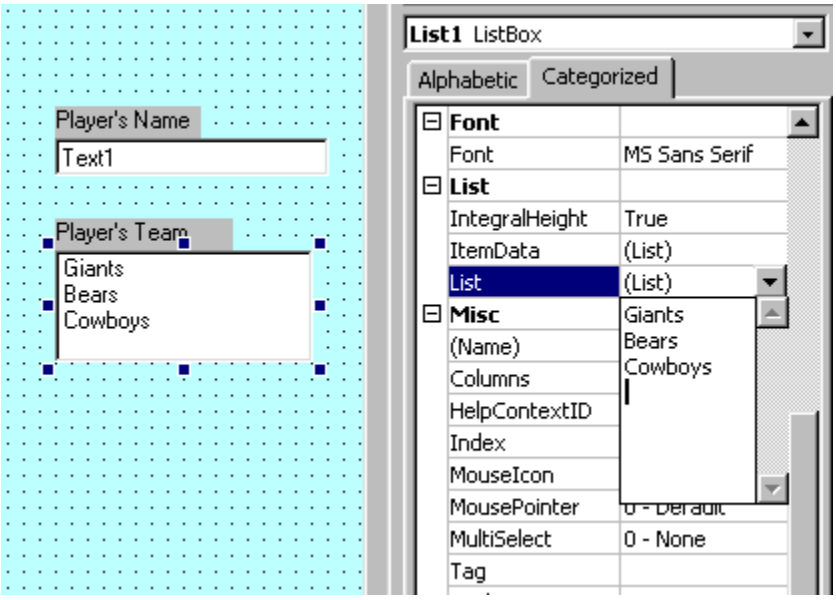
```
Private Sub Command2_Click()
    'To record monthly sales amount in array
    'to be totaled by Division, etc.

    '...
    DeptTotal(DeptNumber) = MonthlySales
    '...
End Sub
```

An array can also be assigned to a variable in code. Look at the following example and understand that that is **not an implicit declaration**. The variable "Week" is declared as Variant. It is assigned an **array value** in code.

```
Private Sub Command3_Click()
    Dim Week, DayOfWeek
    Week = Array("Monday", "Tuesday", "Wednesday")
    DayOfWeek = Week(0)              'first index is 0
    lb_result.Caption = DayOfWeek    'result will be "Monday"
End Sub
```

Now, when we get to the next set of controls, different kinds of Lists, these notions may prove useful.
ListBox

With the ListBox control the user can select items from a list of choices. Although the list of choices can be entered in the List property of the control, as in the example below, this is not a very good habit to get into. It is essentially "hardcoding" data into the program and it can cause maintenance headaches.



The common technique for loading a ListBox is to do it in code, with the Form_Load event. This way, if items have to be added it is a simple matter to add a line of code.

```
Form                                    Load
Private Sub Form_Load()
    'When the form loads, the first thing
    'we do is to assign the names to the
    'list of teams
    lst_team.AddItem "Giants"
    lst_team.AddItem "Redskins"
    lst_team.AddItem "Cowboys"
    lst_team.AddItem "Bears"
    lst_team.AddItem "Jets"
End Sub
```

It is sometimes difficult to distinguish an object's Properties and its Methods. In the example above we used lst_team.AddItem. What is AddItem? It is a Method. How do I know? Well, to tell them apart, think of grammar. A property is a characteristic, something that the object is, a color, a size - it is like an adjective. A Method is an action, something that it does, in fact, a verb. When you see object.something_descriptive, it is a Property. When you see object.some_action, it is a Method.

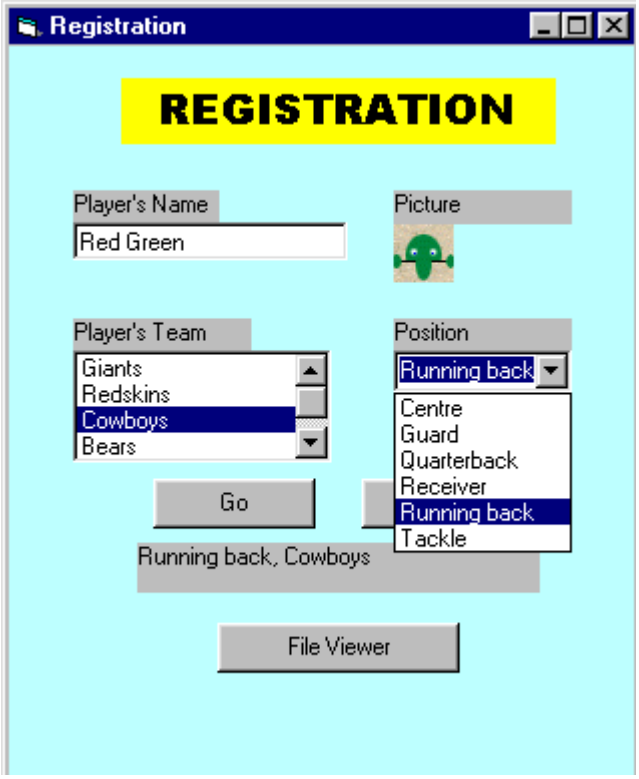
In the example shown, **AddItem is a Method** because it is the action of adding items to the ListBox.
If you wanted to remove an item from the list in code, there is a **RemoveItem Method** for the ListBox.
lst_team.RemoveItem 2 would remove the 3rd team - remember that it starts at 0.
When the Form opens, it will load the list in Form_load before the ListBox is displayed. If there are too many items for the space allocated to the ListBox, it will create a vertical scroll bar.
When the user selects one of the teams from the list, we have to have a way of capturing that information in a variable. That is done with the Text property of the ListBox:

TeamName = lst_team.Text

ComboBox

The ComboBox is called that because it's a combination of a ListBox and a TextBox. It has the advantage over the ListBox of not taking up space until it is actually used which means that it makes it easier to position on the Form.
But the combo has the disadvantage, sort of, that the user can enter his own information, in addition to what is in the list. This may make data validation harder because the choices are not limited. When you want to force the user to make a choice only among the specified items, use a ListBox, even if it is a bit more awkward. If the user is allowed to override the choices, uses a ComboBox.
As in the ListBox, use the **Text** property to get the information input.

Label3.Caption = cbo_position.Text



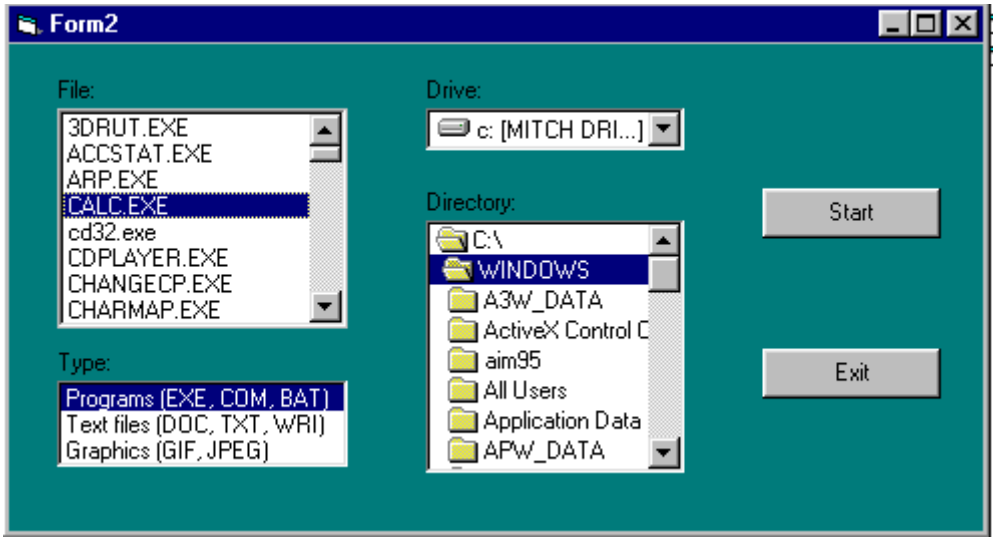

```
Option Explicit

Private Sub Form_Load()
    'When the form loads, the first thing
    'we do is to assign the names to the
    'list of teams
    lst_team.AddItem "Giants"
    lst_team.AddItem "Redskins"
    lst_team.AddItem "Cowboys"
    lst_team.AddItem "Bears"
    lst_team.AddItem "Jets"
    'Next, Load the combo for Position
    cbo_position.AddItem "Guard"
    cbo_position.AddItem "Tackle"
    cbo_position.AddItem "Quarterback"
    cbo_position.AddItem "Receiver"
    cbo_position.AddItem "Centre"
    cbo_position.AddItem "Running back"
End Sub

Private Sub cb_go_Click()
    Label3.Caption = cbo_position.Text _
            & ", " & lst_team.Text
End Sub
```

As you can see, it is fairly simple to load the ListBox and the ComboBox during the From_Load event. The only other detail to note is that the order in which the items appear in the Combo is not the same as the order in which the items were added. That is intentional - it is done with the **Sorted** property for the ComboBox. It can also be done for the ListBox.

DriveListBox, DirListBox, FileListBox

For this next example we need to create a new form, Form2, in the current Project.

Specifications: While in Form1, the Registration form, we need to be able to hit a button which will call-up a new form, the DirList form, which will look like the example below. This form will allow us to select a type of file that we want to see and then to select a file, in a directory, in a drive that will be specified. If the file selected is an executable, we will run the file. If it is a text file we will call-up Notepad to edit it, and if it is a graphics file we will call-up the image editor.
In fact, this allows us to call an external program from inside a form. If, for example, we wish to edit the player's picture before storing it, we can open the picture file with the image editor, change it, and continue with the rest of the form.



There are 3 new controls on this form, plus the buttons and the ListBox. Since you almost always have only one of each of those controls on the form, we won't bother to change the names of the controls in this example - we keep them as: Drive1, Dir1, and File1.

The control that shows the current drive is called a **DriveListBox**. The name of the active drive is in the control's **Drive** property. The selected drive can be changed, in code, by writing: **Drive1.Drive = "D:"**, for example.

> Don't bother looking for the .Drive property in the Properties window for Drive1 - you won't find it. Same with Dir1.Path and List1.FileName. That's because Drive is a runtime property. That is, one that is only available when the program runs. Makes sense when you think about it. You can design the DriveListBox to have the size, the color and the font

you want but you can't tell it which drive will be active at runtime. That will have to come from the system.
VB is full of these details. Every control has properties that are only accessible at runtime, through code. The only way to find them is to look in the documentation. A big Reference Manual is handy and the Help function helps a lot with this, too.

The current directory is in the **DirectoryListBox**. The name is in the **Dir1.Path** property.

The default event associated with Drive1 and Dir1 is called a **Change** event. That's because nothing has to be done with those controls until they are actually changed. Remember, when the program runs they are automatically loaded with the current drive and the current directory active.

The current file selected is in the **FileListBox**, in the File1.FileName property. This one is not automatically loaded because there is no current active file. You select a file by clicking on it, generating a **Click** event.

Study the code and then look at the explanations below. To keep the code section from getting too long, explanations have not been included as comments.

```
(General)                    (Declarations)

Option Explicit
    Dim FileSelected As String
    Dim RunProgram As String
    Dim Result

Private Sub Form_Load()
    lst_types.AddItem "Programs (EXE, COM, BAT)"
    lst_types.AddItem "Text files (DOC, TXT, WRI)"
    lst_types.AddItem "Graphics (GIF, JPEG)"
    lst_types.ListIndex = 0
    If lst_types.ListIndex = 0 Then
        File1.Pattern = "*.EXE; *.COM; *.BAT"
    ElseIf lst_types.ListIndex = 1 Then
        File1.Pattern = "*.DOC; *.TXT; *.WRI"
    Else
        File1.Pattern = "*.GIF; *.JPEG"
    End If
End Sub

Private Sub Drive1_Change()
    Dir1.Path = Drive1.Drive
End Sub

Private Sub Dir1_Change()
    File1.FileName = Dir1.Path
End Sub
```

```
Private Sub cb_start_Click()
    If File1.FileName = "" Then
        MsgBox ("Select a file to run")
        Exit Sub
    End If
    FileSelected = File1.Path
    If Right(FileSelected, 1) = "\" Then
        FileSelected = FileSelected & File1.FileName
    Else
        FileSelected = FileSelected & "\" & File1.FileName
    End If

    Select Case lst_types.ListIndex
    Case 0
        Result = Shell(FileSelected, vbNormalFocus)
    Case 1
        RunProgram = "C:\Program Files\Accessories\Wordpad.exe"
        Result = Shell(RunProgram & " " & FileSelected, vbNormalFocus)
    Case 2
        RunProgram = "D:\Viewer\lviewpro.exe"
        Result = Shell(RunProgram & " " & FileSelected, vbNormalFocus)
    End Select
End Sub
```

```
Private Sub lst_types_Click()
    If lst_types.ListIndex = 0 Then
        File1.Pattern = "*.EXE; *.COM; *.BAT"
    ElseIf lst_types.ListIndex = 1 Then
        File1.Pattern = "*.DOC; *.TXT; *.WRI"
    Else
        File1.Pattern = "*.GIF; *.JPEG"
    End If
End Sub

Private Sub File1_DblClick()
    cb_start_Click
End Sub

Private Sub cb_exit_Click()
    Unload Me
End Sub
```

Program notes:

- First task in Form_Load is to load the list of file types. We only want to display files that are Executable, Text or Graphics. The .EXE is selected by default - ListIndex =0.
- The FileListBox **Pattern** property creates the filter for the selection.

- Whenever we change the Drive selection or the Directory selection, a **Change event** is generated. When the Drive changes, the Directory's path changes and when the Directory changes, the list of files changes.

- When you click on the Start button you first have to check if a file is selected. If not, issue a message.

- The Right() function, which we will look at in Lesson7, checks to see if the rightmost character of the filename is a \. If it is it means that the file is in the root directory. If it isn't, we have to add a \ between the path and the filename.

- Based on the type of file selected, we execute the **Shell function** which runs an executable program. vbNormalFocus is the **window style argument** that tells the program to run in a normal window.

- When we click on a file type, the Pattern property for the FieList must change.

- A double-click on a filename does the same as hitting the Start button.

- Remember, we called this Form from the Registration form. When we're done with this, we want to close it and go back to the calling form. The Exit button does an **Unload** of the current form but, **it does not execute an End statement** because that would cause the Project to end.

This final section of code is in the Registration form. It is the code for the Viewer button which calls the DirList form.

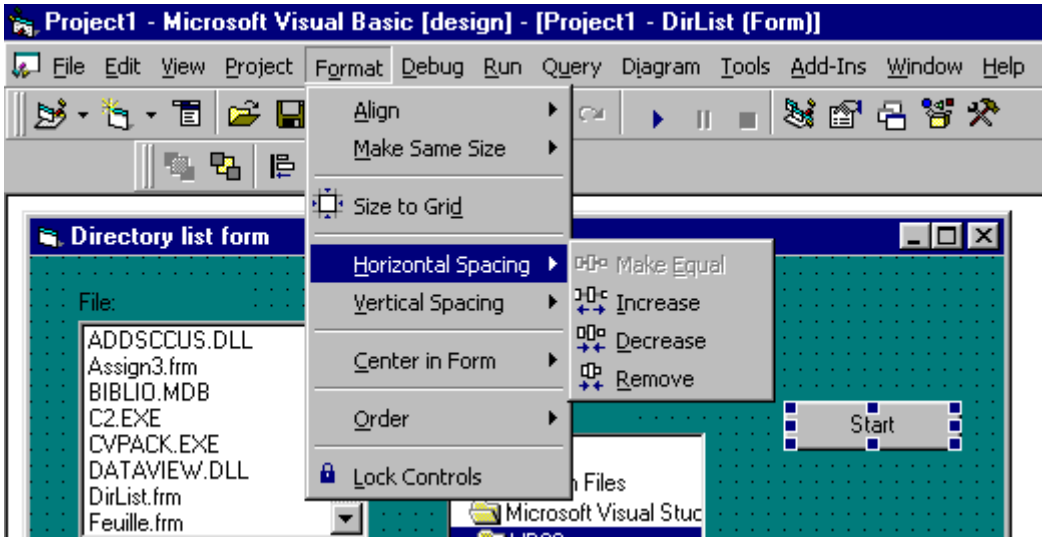
The only thing to do is to Load the form using its FormName (from the Name property) and then to execute its **Show method**. The argument **vbModeless** means that the form does not get exclusive focus. The opposite of vbModeless is **vbModal**. A **modal** form is one which requires action from the user before it can be closed. Usually, error messages are modal - you have to respond, usually by hitting the OK or Cancel button, and you can't click on another form to send this one to the background, and you can't close it with the close box. A **modeless** form can be sent to the background and it can be closed at any time.

```
Private Sub cb_viewer_Click()
    Load DirList
    DirList.Show vbModeless
End Sub

Private Sub cb_exit_Click()
    Unload Me
    End
End Sub
```

5: Menu and Debug

5.1 Creating a Menu

If you've worked with Windows applications before you have worked with menus. Every Windows application has one.
The menu gives the users access to functions that are not defined as controls (editing, formatting, etc) and also repeats certain functions that are coded as controls (Exit button, for example). Menus offer a variety of functionalities to define the application: we can include sub-menus, checked items, enabled/disabled functions, toolbar icons. The VB IDE that you are using certainly displays all of those tools, as in the diagram below.



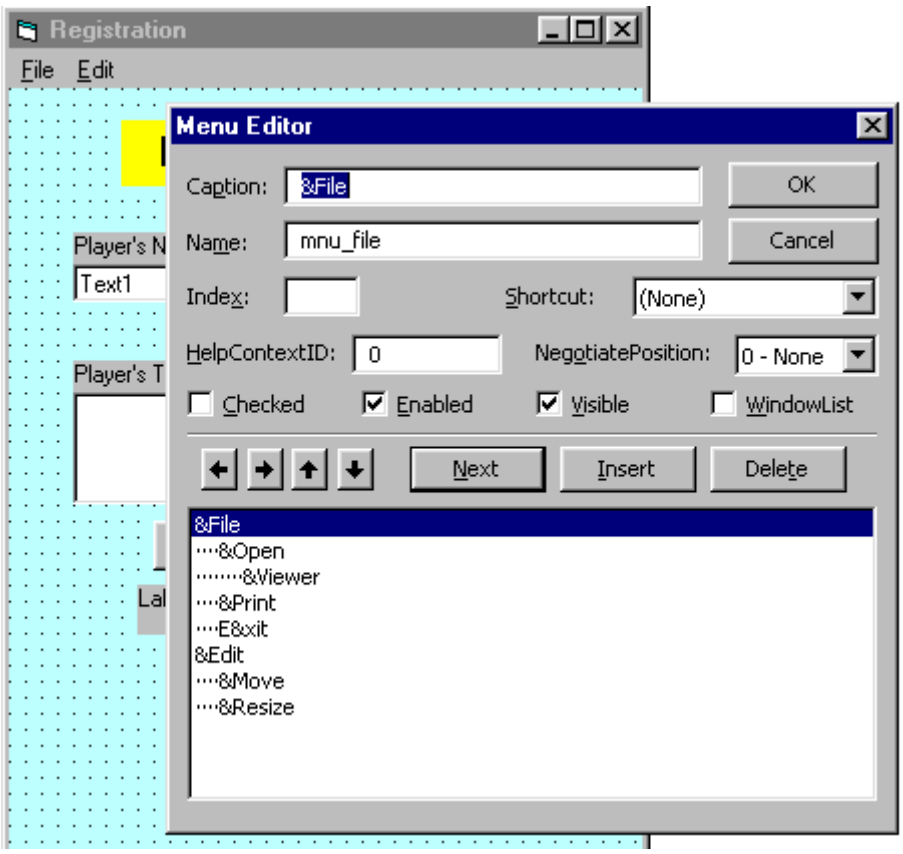
For this lesson, we will use the Registration form we created in Lesson 5 and we will add a menu to it.

The easiest way to create a menu is to do it with the Application wizard when creating the application. But since we're not here to do things the easy way, we'll have to rough-it. In this case, roughing-it is not much harder. We use the **Menu Editor** that can be found in the Menu bar --> Tools. Using the Editor is fairly obvious. We just build up the menu bar on the first level and then, we add sub-menus using the arrow keys to add an elipsis before the captions. Thus, &File is on the menu bar and ...&Open is under &File. Items can be inserted anywhere using the Insert button.

You may have noticed the use of the ampersand (&) in the captions (the **Caption** is the part that will display in the menu bar, not the name). That is standard Windows practice. It creates a Hot-key, meaning a function that can be called from the keyboard using the <Alt> key. Putting an & before a letter in a

caption makes that letter the hot-key for the function; <Alt><F> will call-up File, <Alt><E> will call-up Edit, and so on. Just make sure that the same hot-key is not used for 2 functions on the same level. In the menu bar for VB above, note that <Alt><F> is used for File but, <Alt><o> is used for Format. The hot-key for each function is the letter underlined so there should'nt be any confusion.

The other consideration when creating the menu is to give each menu item a specific **name**. In this case we use the prefix mnu_ to identify menu items. These are important because they will be refered to in code and it should be clear that mnu_exit is the Exit function in the menu whereas cb_exit is the Exit command button.



You can run the application at any time while you create the menu, just to verify that it displays correctly. Of course, if you click on a menu item, nothing happens. Just like controls, menu items have to be coded to work. So, we go to the code window and write the code for each of the menu items that we want to activate. Fortunately, some of it is automatic. Clicking on a menu item will automatically open lower-level items, if there are any. We just code for the lowest-level item. For example, for File-->Open-->Viewer, there is no code for File, nor for Open but, we must write the code to execute for when Viewer is clicked.

For this example we will code a few simple operations to show how it is done. From this it is just a question of expanding the menu to display more functions.

```
'Click on menu Exit same as click on button Exit
Private Sub mnu_exit_Click()
    cb_exit_Click
End Sub
'Invoke Move method for this form (Me)
'Look at Form object --> Methods in Help
Private Sub mnu_move_Click()
    Me.Move 0, 0
End Sub
'Invoke PrintForm method for this form (Me)
'Sends image of form to printer - useful for hardcopy
Private Sub mnu_print_Click()
    Me.PrintForm
End Sub
'Parameters of Move are: left edge, top edge, width, height
'Measurements in twips (see Lesson 7)
Private Sub mnu_resize_Click()
    Me.Move 3000, 3000, 6000, 5000
End Sub
'Load and Show form DirList
Private Sub mnu_viewer_Click()
    Load DirList
    DirList.Show vbModeless
End Sub
```

When working with forms, there is always a certain amount of data validation that has to be done. Data validation consists of making sure the data is correct before doing calculations and so on. Usually, until the data is all correct certain functions such as calling the database or going to the next form have to be made unavailable - we say that the function is **disabled**.

To complete the example, let's say that we want to disable the Viewer option if the player's name has not been entered. To do this we add some code in the Go button. The code consists of setting the **Enabled property to False** if we want to disable a control or menu item; we set the property to True to enable the control again. When disabled, the Caption goes gray and the code cannot be executed. In the case of Viewer, where we have both a button and a menu function, we must remember to disable both.

```
Private Sub cb_go_Click()
    Label3.Caption = cbo_position.Text _
            & ", " & lst_team.Text
    If tb_name.Text = "" Then
        mnu_viewer.Enabled = False
        cb_viewer.Enabled = False
        tb_name.SetFocus
    Else
        mnu_viewer.Enabled = True
        cb_viewer.Enabled = True
    End If
End Sub
```

Debugging code

After six lessons of this tutorial, it would be surprising if you had never had a bug in the code you've been writing. By now you should have written some original code for yourself, not just the examples supplied with the lessons. Whenever you write code there is the possibility of making mistakes. Heck! even Professors make them, although very rarely. It can be very frustrating to try to find an error when you don't know where to begin to look. But there are techniques that can help.

> Do you know where the term "**bug**", for a program error, comes from? In case you've never heard the story before, here it is, as told by Grace Hopper, one of the pionneers of programming.
> In the late 40's even a simple computer was a big thing: 1000's of vacuum tubes and 1000's of square feet of floor space. A group of programmers were working late one hot summer night. To help to dissipate all the heat generated by those tubes, all the windows were open. At one point the program that they were working on bombed-out. Eventually they found the problem: a moth had flown in and had become lodged in the wiring, creating a short-circuit. Afterwards, every time a program would crash the programmer would exclaim, "There must be a bug in the machine!". To this day that has remained one of the mainstays of programmers: when the program goes wrong, blame the hardware!

One of the first techniques to master is the use of **breakpoints.** A breakpoint is a flag at a given point in code where program execution will be suspended to give you time to look at the content of variables or at the status of properties. When VB hits a breakpoint when running a program, the code window opens and an **immediate window** opens at the bottom of the screen. You can look at variables or properties in the immediate window and then, either do **Start** to resume execution or do **Step**, using **<F8>** to step through the execution, one statement at a time.

Again we will use the code from Lesson 5. In the code window, click the column to the left of a line of code. This will create a breakpoint indicated by a red dot (you remove the breakpoint by clicking on the red dot). When you run the program it will stop at the breakpoint. In the immediate window, look at the content of different variables or properties. Step through the code with <F8>; the active statement is indicated by the yellow arrow. All the logic represented by IF or LOOP or DO statements will be executed according to the conditions present. If the yellow arrow jumps to a line that you don't expect, find the reason why.

```
Private Sub cb_go_Click()
    Label3.Caption = cbo_position.Text _
            & ", " & lst_team.Text
    If tb_name.Text = "" Then
        mnu_viewer.Enabled = False
        cb_viewer.Enabled = False
        tb_name.SetFocus
    Else
        mnu_viewer.Enabled = True
        cb_viewer.Enabled = True
    End If
End Sub
```

```
Immediate
print label3.Caption
Label3
? tb_name.Text       ' ? same as print

? mnu_viewer.Enabled  'done on first brekpoint
True
? mnu_viewer.Enabled  'done after stepping-through
False
```

Another technique to learn is called **"error trapping"**. It consists in intercepting errors that can occur at execution rather than programming mistakes, although not providing for user errors can be considered a programming mistake.

Let's build a simple example. The user will input 2 numbers, a numerator and a denominator. The program will divide the numerator by the denominator and display the result. Easy so far. However, if the user inputs 0 for the denominator, the program crashes because programming cannot make sense of division by zero. So, we want to **trap the error** and process it before it displays an error message to the user. We will use the **On Error GoTo ...** statement. This tells the program that if there is some kind of run-time error, go to the error-processing-routine named. We have to create a **line label** to identify the error routine; a line label has a colon at the end, like error_rtn:, in the example. At the same time, there is an **Err object** created and it contains, among other things, a **Number property** that will identify the error. For example, if Err.Number = 11, the error was a division by zero; Err.Number = 6 represents an overflow situation.

```
Private Sub cb_calc_Click()
    Dim numer, denom, answer, msg
    On Error GoTo error_rtn

    numer = Text1.Text
    denom = Text2.Text
    answer = numer / denom
    Label2.Caption = answer

    Exit Sub    'needed, otherwise continue thru error_rtn
                    'try it with breakpoint and step
error_rtn:      'this is a line label
    If Err.Number = 11 Then     'error 11 = divide by 0
        msg = MsgBox("Cannot divide by zero", vbOKCancel)
    End If
    If msg = 1 Then     'OK button was pressed in msgbox,
        Text2.SetFocus  'get another number
    Else                'Cancel button was pressed,
        cb_exit_Click   'end program
    End If
End Sub

Private Sub cb_exit_Click()
    Unload Me
    End
End Sub
```

It is worth noting that line labels in code do not end processing in any way. When the logic gets to a line label it keeps on going. The programmer has to make sure that the processing of errors in the error_rtn is not done automatically every cycle (that is called "falling through" the next routine and it's a common error).

Manipulating text

Whenever you are entering data, creating files or databases, you are working with text strings. Text strings contain characters that can be copied, deleted, cut and reassembled but they also have important visual characteristics: size, color, weight, transparency, etc. In this lesson we will look at different ways of manipulating those text strings.

String functions

Here is a list of the basic functions that work with strings:

- **Len(string)**: returns the length of *string*, the number of characters it contains.
- **Left(string, number)**: returns the number of characters specified by *number* from the left end of *string*.

- **Right(string, number)**: returns the number of characters specified by *number* from the right end of *string*.
- **Mid(string, position, number)**: returns the number of characters specified by *number* starting at character number *position* from the left end of *string*.
- **InStr(string1, string2)**: returns the position of *string2* in *string1* - returns 0 if *string2* is not found in *string1*.
- **LTrim(string), RTrim(string) and Trim(string)**: returns *string* with non-significant spaces removed from the left, the right or both, respectively.
- **LCase(string), UCase(string)**: returns *string* converted to lower-case or upper-case, respectively.

```
Option Explicit
    Dim string1 As String, string2 As String
    Dim answer As String
    Dim result As Integer, number As Integer
    Dim position As Integer

Private Sub cb_go_Click()
    string1 = "It was the best of times, " & _
        "it was the worst of times."
    string2 = "best of times"

    result = Len(string2)   'will return 13

    answer = Left(string1, 6)   'will return "It was"

    answer = Right(string1, 6)  'will return "times."

    answer = Mid(string1, 8, 8) 'will return "the best"

    result = InStr(string1, string2) 'will return 8

    answer = UCase(string2) 'will return "BEST OF TIMES"

End Sub
```

Formatting Numbers, Dates and Times

The Label control is still the easiest way of displaying the result of calculations. Whatever the answer is, just move it to Label5.Caption and it will appear on the form. Unfortunately, it does not always appear the way you want to see it. No problem if the result is a string but, what if it is a dollar amount or a date of some kind. That will require some formatting of the result before displaying it. We use the **Format function**:

Label5.Caption = Format(result, "formatting characters")

Numbers

For example, given that:

```
Dim result As Single

result = 3456.7

Label5.Caption = Format(result, "00000.00")        'displays: 03456.70

Label5.Caption = Format(result, "#####.##")        'displays: 3456.7

Label5.Caption = Format(result, "##,##0.00")        'displays: 3,456.70

Label5.Caption = Format(result, "$##,##0.00")       'displays: $3,456.70
```

Here is a list of what the formatting characters mean:

0	Represents a digit, with non-significant leading and trailing zeros
#	Represents a digit, without non-significant leading and trailing zeros
.	Decimal placeholder
,	Thousands separator
$ + - ( ) space	Literal character; displayed as typed

When displaying dollar amounts, it is good practice to always use the 0 placeholder with the decimal so that **10 cents** does not come out as **$.1** or **$0.1** Using the formatting string **"$#0.00"** ensures that the value follows standard rules and comes out as **$0.10**.

Dates and Times

When working with dates and times, you need to know that there is a function that will obtain the current date and time from the system clock. The function is: **Now( )** and you can use it directly as in:

Label5.Caption = Now( )

The result is the current date and time formatted according to what you specified in the Windows Control Panel for your system. If you want to format the result, because you don't want to see the time, for example, there are formatting characters for date and time, as there are for numbers. The main characters are:

yy	year without the century - eg: 00
yyyy	year with century - eg: 2000
m	month number - eg: 12
mmm	short name of month - eg: dec
mmmm	long name of month - eg: december
d	day of the month, without zero - eg: 8
dd	day of the month, with zero - eg: 08
dddd	name of the day of the week - eg: Monday
h	hour, without zero - eg: 7
hh	hour, with zero - eg: 07
mm	minutes - eg: 45
ss	seconds - eg: 55

Thus, if Now( ) is July 8, 2000 ,
Label5.Caption = Format(Now( ), "dddd, yyyy mmmm dd")
returns: Saturday, 2000 July 08

Of course any other date can be formatted for display:

Label5.Caption = Format(DateOfBirth, "yyyy-mm-dd")

Named Formats

In addition to the formatting string, there are several **named formats** that can be used to determine the output format. These named formats are VB constants that you call when you need them:

General Number	Number with no thousands separator

Currency	Thousands separator, two digits to the right of decimal
Fixed	Displays at least one digit to the left and two digits to the right of decimal
Standard	Thousands separator, at least one digit to the left and two digits to the right of decimal
Percent	Multiplies by 100, add percent sign to the right
General Date	Display determined by Control panel settings; displays date and time
Long Date	Long date format specified for system
Short Date	Short date format specified for system
Long Time	Long time setting specified by system; includes hours, minutes, seconds
Short Time	Shows hours and minutes

```
Dim DateHired As Date
DateHired = "1995-10-25"
Label5.Caption = Format(DateHired, "Long Date")
```

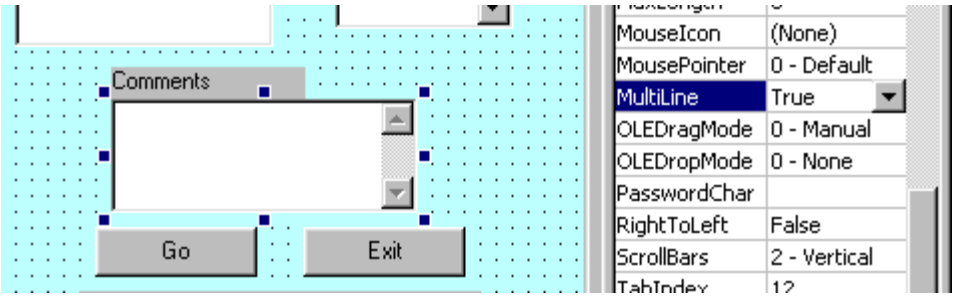
returns: October 25, 1995

Manipulating blocks of text

The **TextBox** and the **ComboBox** controls contain several properties which will allow you to manipulate blocks of text, in addition to single characters.

If you have to input a large quantity of text in a TextBox, for example, you do not want to see it all in a single line. There are 2 properties that you set that will make the data easier to see:

- **MultiLine = True** allows you to have several lines of input, all separated by <Enter>.
- **ScrollBars = 2 - Vertical** will create scrollbars, useful to read text.



Then there are 3 properties to work with a **block of selected text** in the control:

- **SelStart** an integer number identifying the start of selected text, the position of the first character in the block - starts at 0.
- **SelLength** an integer number identifying the number of characters selected in the block.
- **SelText** a string containing the selected text.

Note that this kind of manipulation is usually done with the mouse. However, you do not have to code for the mouse events. It is automatic - when you select text in a control, the appropriate events, MouseDown, MouseUp and MouseMove, are triggered by the control.

Useful objects: Screen and Clipboard

The **Screen object** represents the complete Windows environment. It allows access to all Forms and Controls. It has **2 important properties** that we need:

- **ActiveForm** returns the name of the Form currently active.
- **ActiveControl** returns the name of the Control that currently has **focus**.

In the example that follows we will use these properties to avoid having to name the form and the control in code. This is a way of implementing **re-usability of code**, an important design principle - we can write code that can be re-used in many applications without having to be re-written.

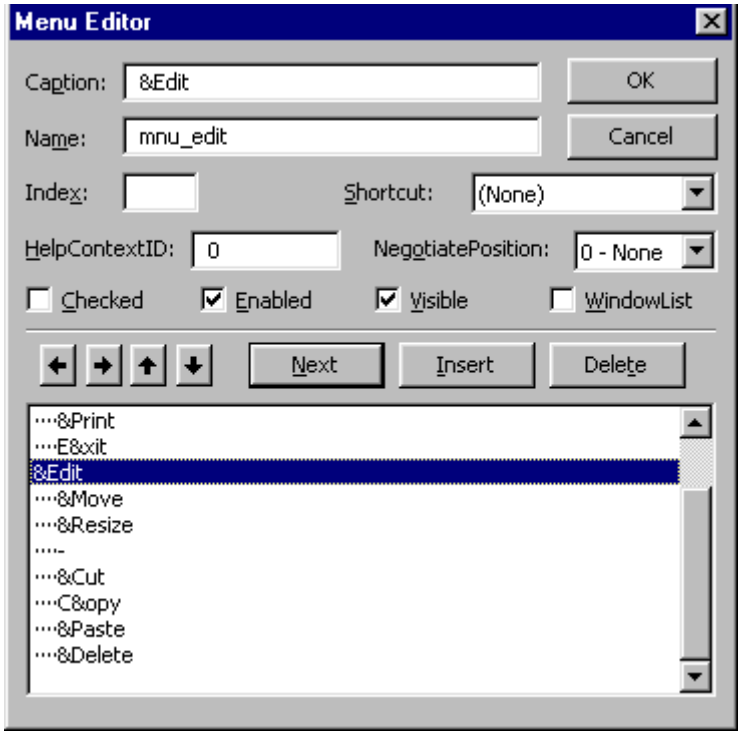
The **Clipboard object** is the system clipboard that you know from all your other Windows applications. It is the object that temporarily stores text or graphics between applications. In the case of the Clipboard object, it has **3 important methods** that we will use:

- **Clear** empties the Clipboard.
- **SetText** puts the selected text into the Clipboard.
- **GetText** gets the contents of the Clipboard.

Example

For the purposes of this example, we will use the Registration Form from Lesson 5.
We will add a Comment TextBox to the form. This textbox will be multiline, with a vertical scrollbar. Then, we will add items to the menu to allow us to edit the text entered in Comments. We want to be able to Cut, Copy, Paste and Delete blocks of text.

To change the Menu, we again call upon the **Menu Editor**. We add the new functions under the Edit item. To insert a **separator bar**, just put a single hyphen in the Caption and give it a Name, mnu_sep1, for example. The menu should look like this:



Then we code the menu events. Note that we use the Screen properties exclusively in this example. Even if we are working in a control which is called txt_comments, there is nothing in the code that refers specifically to that control. We can copy this whole section to any form in any application and it will work without a hitch.

```
mnu_cut                                   Click

Private Sub mnu_cut_Click()
    'Make sure clipboard is empty
    Clipboard.Clear
    'Copy selected text to Clipboard
    Clipboard.SetText Screen.ActiveControl.SelText
    'Delete selected text
    Screen.ActiveControl.SelText = ""
End Sub

Private Sub mnu_copy_Click()
    'Make sure clipboard is empty
    Clipboard.Clear
    'Copy selected text to Clipboard
    Clipboard.SetText Screen.ActiveControl.SelText
    'Selected text is not deleted - stays in Control
End Sub

Private Sub mnu_paste_Click()
    'Place text from clipboard into active control
    Screen.ActiveControl.SelText = Clipboard.GetText
End Sub

Private Sub mnu_delete_Click()
    'Delete selected text
    Screen.ActiveControl.SelText = ""
End Sub
```

Graphics

When working with graphics (pictures, diagrams, images) the first thing to master in the environment is the **coordinate system.** That is the 2-dimensional grid that defines locations on the screen in terms of (x,y) coordinates. x is the horizontal position as measured from the left edge and y is the vertical position as measured from the top. (0,0) is the upper left-hand corner of the screen, form or other container.

By default, all VB measurements are in **twips**. A twip is 1/20 of a printer's point (of which there are 72 per inch). Thus, there are **1440 twips in an inch, 567 in a centimeter**. The measurements refer to printed size; because there are great variations between monitors, sizes may vary.

You can change the units of measure yourself by setting the **ScaleMode property** for objects or controls.

ScaleMode	Meaning
0	User-defined.

1	Twips - 1440 per inch.
2	Points - 72 per inch.
3	Pixels - number per inch depends on monitor.
4	Characters - character = 1/6 inch high and 1/12 inch wide.
5	Inches.
6	Millimeters.
7	Centimeters.

Examples:

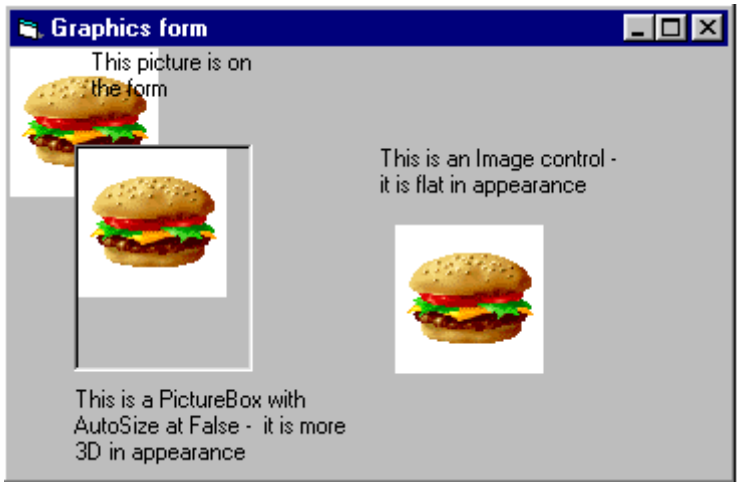
Me.ScaleMode = 5 'sets scale to inches for this form
pic_EmployeePic.ScaleMode = 3 'sets scale to pixels for control

Adding Pictures

You can display pictures in 3 places in your application:

- On a form
- In a picture box
- In an image control

The **PictureBox** control and the **Image** control are very similar in operation. However the PictureBox is more flexible in its methods and properties and is the one you should use most of the time. There is little use for the Image.
In all 3 cases you will display a picture contained in a graphics file (.BMP, .GIF, .JPEG, .WMF, etc). The name of the file containing the picture will be entered in the **Picture property** of the control or form.
In a form, the picture cannot be resized or moved, so it should be of the correct size before you use it. The picture will serve as background and other controls that you put on the form will be displayed over it.
The PictureBox's **AutoSize property** must be set to **True**, otherwise the control's size will not change to accomodate the picture, as in the example below.



In the above example the pictures were all added to the controls at design time.

You can also insert or remove a picture at run time. You use the **LoadPicture function**, as in:

pic_departmentlogo = LoadPicture("C:\Pictures\acctnglogo.bmp")

Removing the picture is done with the LoadPicture function without a file name:

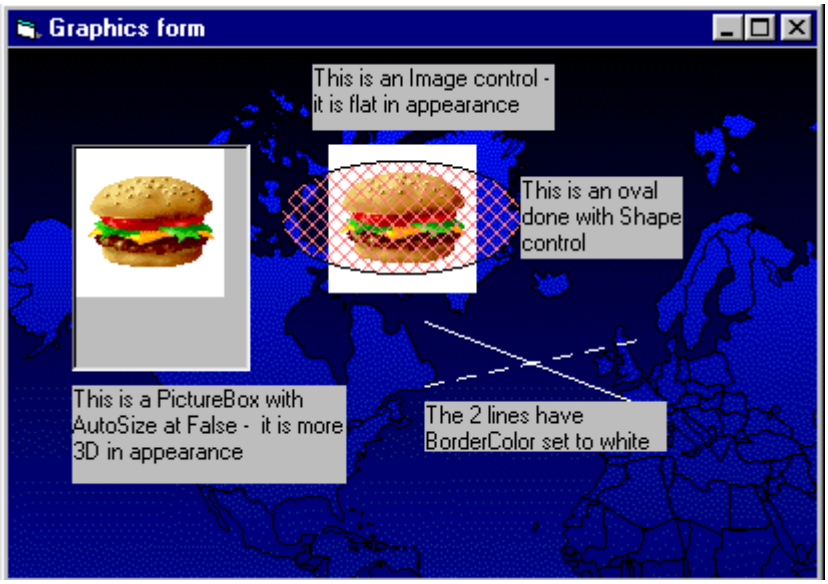
pic_departmentlogo = LoadPicture ("")

Drawing controls

There are 2 controls in the toolbox which allow you to draw directly on the form - the **Line control** and the **Shape control**.

Both are easy to use and fairly obvious. The main properties of each that have to be manipulated are: **BorderColor** for the color of the line or shape and **BorderStyle** to use a solid or dashed line.

In addition, the Shape control has: **Shape** for rectangle, circle, etc., **FillColor** and **FillStyle** to determine how the shape will be filled and **BackStyle** for transparent or opaque.



Multimedia

Multimedia refers to devices other than the screen or the printer to play sounds, watch videos or record music. This is done through the use of the **Multimedia control**. Don't look for it in the toolbox, it's not there. It is an additional control that you must load.

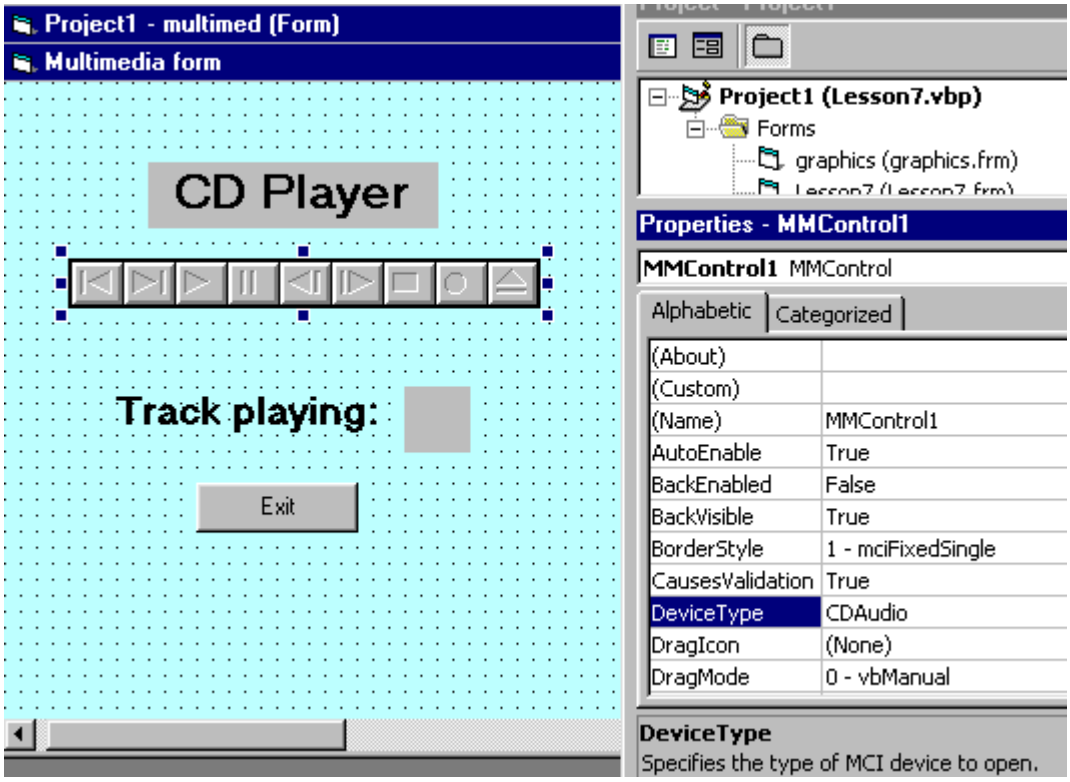
First, create anew form in Project Lesson7 and call it "multimed.frm". Then, in the menu, at **Project --> Components**, find the item "**Microsoft Multimedia Control 6.0**" and check the box next to it. Hit OK and that's it. The Multimedia control should now appear in your toolbox.

If you select the multimedia control and put it down on the form, you will have a button bar like all the ones you've seen on CD players, recorders, etc. In the **DeviceType property** you specify what type of device this control controls:

DeviceType	**Device**
CDAudio	CD Audio player
DAT	Digital audio tape player
Overlay	Overlay
Scanner	Scanner
Vcr	Videotape player and recorder
Videodisc	Videodisc player
Other	Other devices not specified

Example: a simple CD player

We create a new form in Lesson7 and call it multimed.frm. After adding the Multimedia control to the toolbox, we put a MM control on the form. Since we will only be using the one MM control, we'll leave its name as MMControl1. The only property we have to change at this time is the **DeviceType**, to tell it that we are using the CD player, so we write CDAudio in DeviceType. We add a few labels to complete the form and we get:



Now we have to write the code to operate the CD player.

Before we start to write the code there are a few things to know about the MM control. There is a **Track property** which contains the number of the current track. But its most important property is called the **Command property** and it can take on several values that, in fact, operate the device.

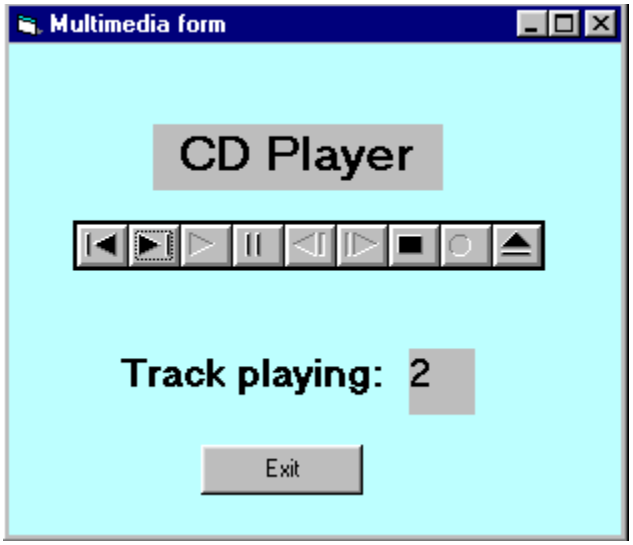
Command value	**Meaning**
Open	Opens the device
Close	Closes the device
Eject	Ejects the CD

Play	Plays the device
Pause	Pauses the device
Next	Goes to next track
Prev	Goes to beginning of current track. If used within 3 seconds of most recent Prev, goes to beginning of previous track
Record	Initializes recording
Save	Saves the open device file
Seek	Step backward or forward a track
Stop	Stops the device
Step	Step forward through tracks

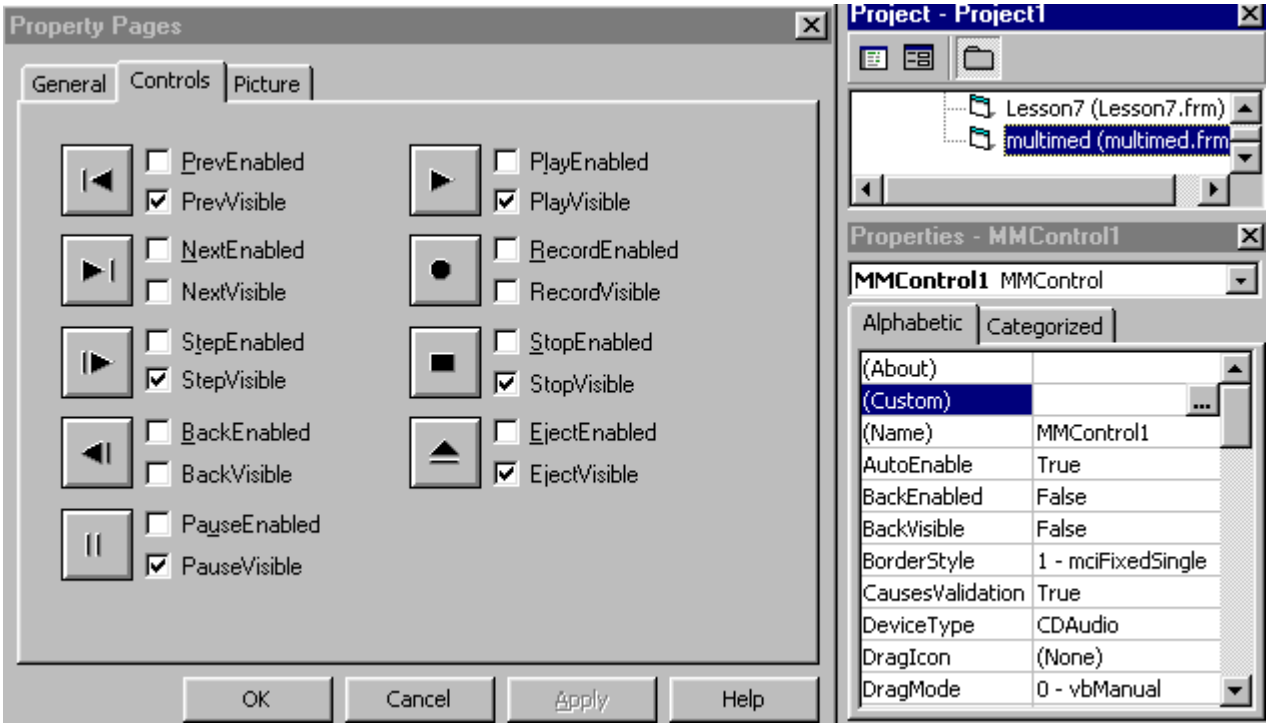
Understand that both Track and Command are run-time properties because they are meaningless at design time. For example, to open the CD player:
MMControl1.Command = "Open" 'we assign the value "Open" to the Command property
To pause:
MMControl1.Command = "Pause" 'we assign the value "Pause" to the Command property

Now, as you have seen, the trick is to know with which events to associate the code that has to be written. The first one is fairly obvious: when we load the form, the **Form_Load event**, we will open the device. Now, one we haven't used before. When we unload the form, we will close the device. The reason is that, once launched, the device will keep on playing, even if the form is closed. So, just click on the **Form_Unload event** and write the code there. Finally, just to see that things are running smoothly, we will use the StatusUpdate event for the MM control to display the track being played every time the status of MMControl1 changes (change track, pause, play are all status changes).
As you will see, once the CD starts playing, you can use the buttons in the MM toolbar to control what it does.

```
MMControl1                    StatusUpdate

Private Sub Form_Load()
    MMControl1.Command = "Open"
    MMControl1.Command = "Play"
End Sub

Private Sub Form_Unload(Cancel As Integer)
    MMControl1.Command = "Stop"
    MMControl1.Command = "Close"
End Sub

Private Sub MMControl1_StatusUpdate()
    lbl_track.Caption = MMControl1.Track
End Sub

Private Sub cb_exit_Click()
    Unload Me
    End
End Sub
```



You may notice that some of the buttons for the CD Player are not used during play. If you want you can hide these buttons from the control by using the **(Custom) property**. This will open an editor window that will allow you to customize the MMControl.



LESSON 8 - Working with files

Storing data

Data comes in many forms. It can be a list of DVDs you own and want to keep track of, the description of all the online college courses you intend to take or even the movie stars you intend to date!

In the previous lessons, you have learned how to manipulate the VB environment to produce forms, do calculations, edit text and so on. However, everything you've done so far is a one-shot deal. Even if you did create the Payroll form, you can use it to calculate the net pay for any number of employees but, you can't save any of that information.

That's where data storage comes in. There are many ways to store data for future use. The most popular and powerful method is to create a database. But that can get quite involved and it does require a certain amount of analysis knowledge and skill. The next two lessons will cover how to create and use databases in VB.

A much more accessible method and one which you have certainly used many times before, is to create a **data file**. A file is a collection of data on a given subject, stored on a storage medium, usually a disk or CD. There are executable files, usually with the .EXE extension, library files (.DLL), Word document files (.DOC) and a hundred other types. Many applications call for data to be stored and then read back later for further processing. Think of a simple application: an Address book to store people's names, addresses and phone numbers. You could create an Address book database and indeed, it is often the first one you learn how to do in database courses. However, the task is more suited to data file processing. You just want to create a form to input names, addresses and phone numbers and then you want to store all the information entered in a file so that you can print it or look-up numbers when needed. In this lesson we will learn how to create our own files to store and retrieve data.

Defining new terms

- **Record:** one logical section of a file that holds a related set of data. If the file contains Student information, a record would hold the information on one student: name, address, studentID, etc. If there are 5,000 students registered, the file contains 5,000 records.

- **Field:** part of a record that defines a specific information. In the Student record, FirstName, LastName, StudentID, are fields. The field is the lowest element in the file. Even if the information consists of one character, Sex is M or F, it is still considered a separate field. The field is the equivalent of the variable - we call it a variable when it is used to store data in memory and call it a field when it stores in a file.

- **I/O:** stands for Input/Output. Whenever you work with a file you have to have ways of reading data from the file (that's **Input**) and ways of writing data to the file (that's **Output**). I/O operations consist of all those commands that let you read and write files.

Types of files

There are basically three types of files you can work with:

- **Sequential file**: this is a file where all the information is written in order from the beginning to the end. To access a given record you have to read all the records stored before it. It is in fact like listening to a tape - you can go forward or back but you can't jump directly to a specific song on the tape. In fact, in the old days, magnetic tape was the most commonly used medium to store data and all files were organized this way. Now, it is still useful when there is a small amount of data to store, a file of application settings, for example. It can even be of use when there is a large amount of data to be stored, provided it all has to be processed at one time, eg: a file of invoices to produce a statement at month-end.

- **Random file**: a file where all records are accessible individually. It is like a CD where you can jump to any track. This is useful when there is a large quantity of data to store and it has to be available quickly: you have to know if a part is in stock for a customer who is on the phone; the program doesn't have time to search through 10,000 records individually to locate the correct one. This method of storage became popular when hard-disk drives were developed.

- **Binary file**: this is a special, compacted form of the random file. Data is stored at the byte level and you can read and write individual bytes to the file. This makes the file access very fast and efficient. We won't be covering this type of file in these exercises. If you need to find out more about it, go to the VB Reference Manual.

Opening and closing files

To begin our work on files we will look at some commands that are common to both Sequential and Random files. After that we will look at the specific processing commands for each type of file.

The first command to include in a program that needs to work with files is the **Open** command. Open assigns the file to a numbered **file handle**, also called a **channel**, or sometimes a **buffer**. The format of the command is:

Open "Filename" [For Mode] [AccessRestriction] [LockType] As #FileNumber

For example:

Open "MyFile.txt" For Random Read Lock Read As #1

- **MyFile.txt** is the name of the file in the disk directory.
- **For Random** means that access to the records can be random; if access is not specified, For random is the default value.
- **Read** restricts access to Read-only - the user cannot write or change the records.
- **Lock Read** means that only the person reading the record can have access to it at any given time; it is not shared among users.
- **As #1** means the file is assigned file handle #1; for all processing in the program, it will always be refered to as #1, not its Filename.

AccessRestriction and **LockType** are parameters that are used mostly with files in a network environment. You use them when you want the file to be shared or not, and you want to prevent certain users from changing or deleting things that they shouldn't. For the rest of this lesson we will not be using those parameters.

Access Mode

For Mode in the Open statement indicates how the file will be used. There are five access modes:

- **Input:** open for sequential input; the file will be read sequentially starting at the beginning.

- **Output:** open for sequential output; records will be written sequentially starting at the beginning; if the file does not exist, it is created; if it does exist, it is overwritten.

- **Random:** open for random read and write; any specific record can be accessed.

- **Append:** sequential output to the end of an existing file; if the file does not exist it is created; it does not overwrite the file.

- **Binary:** open for binary read and write; access is at byte level.

If access mode is not specified in the Open statement, **For Random is used by default**.

Once processing is finished, you need to **Close** all the files that have been opened. The format for the **Close statement** is:

Close #FileNumber1 [, #FileNumber2] ...

You can close any number of files with one Close statement. Eg:

Close #1, #2, #3

The following statement closes all open files:

Close

Writing and Reading a Sequential file

There are two commands that allow you to write data to a sequential file: **Print** and **Write**. They work in almost the same way but, the Print command does not separate the fields in the file in quite the same way which makes the data harder to read afterwards. There is really no valid reason to use Print when creating a sequential file. In the rest of this lesson **we will use Write** exclusively.

The format of the Write command is:

Write #FileNumber, OutputList

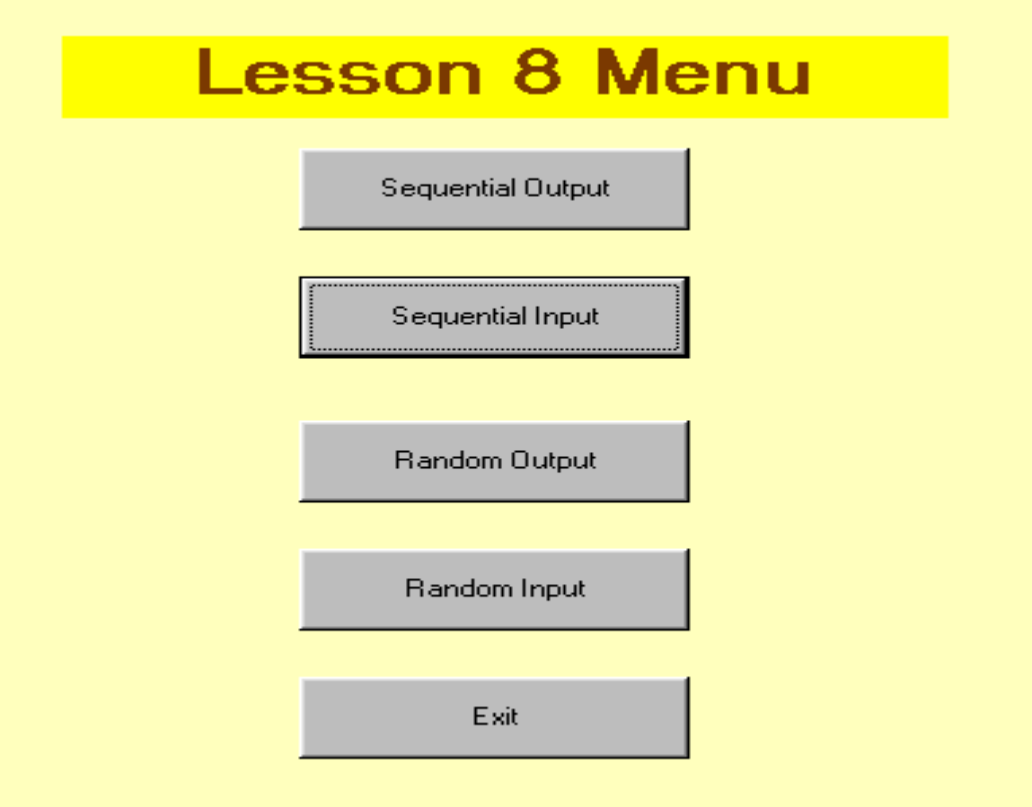
where FileNumber is the number the file was opened with and OutputList is one or more variables you want to write to the file.

Address Book Example

In this exercise we will create a simple address book file to keep track of people's names, addresses and phone numbers.

To handle the various forms that we have to use, we will develop a new technique for these lessons: the use of a **Menu** of choices. Note that that is not the same as a Menu bar used in a form. In this case we are just going to line-up a series of buttons for the different forms that have to be called. There has also been a small change to the display format - from now on all the forms are maximized (they occupy the full screen) - this is often easier for the user to work with, rather than have a number of different forms overlapping on the screen. To get the form to run maximized, change the **Form property WindowState -> 2 - Maximized**.

This is what the menu should look like:



The code for the menu consists of loading and showing the various forms. The Exit button exits the Menu itself. Any open files are closed by the individual forms.

```
cb_SOut                                   Click
Private Sub cb_SOut_Click()
    Load SAdresOut
    SAdresOut.Show vbModeless
End Sub

Private Sub cb_SIn_Click()
    Load SAdresIn
    SAdresIn.Show vbModeless
End Sub

Private Sub cb_ROut_Click()
    Load RAdresOut
    RAdresOut.Show vbModeless
End Sub

Private Sub cb_RIn_Click()
    Load RAdresIn
    RAdresIn.Show vbModeless
End Sub

Private Sub cb_Exit_Click()
    Unload Me
    End
End Sub
```

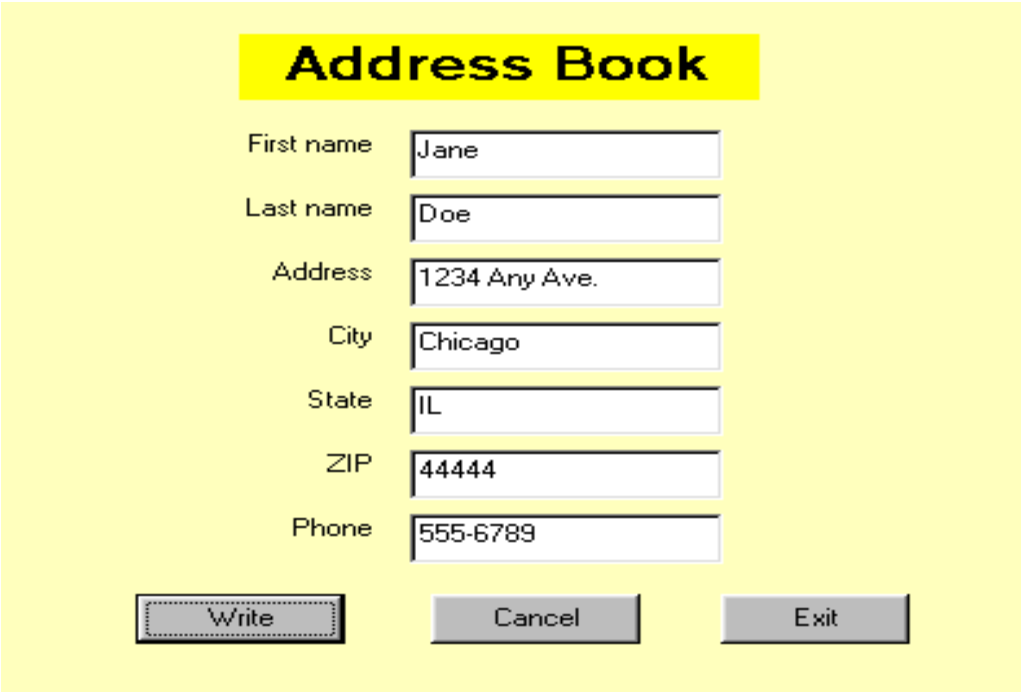
File design

It has been determined that the file will store 7 fields of information. First and last names could be together and we could have a work phone number but, the Analyst (who gets paid big bucks to think this stuff up) has determined that 7 is what is required. It has also been decided that the file will be called "AdrsBook.txt" and will be stored in "C:\VBApps" - we need to know this for the Open statement.

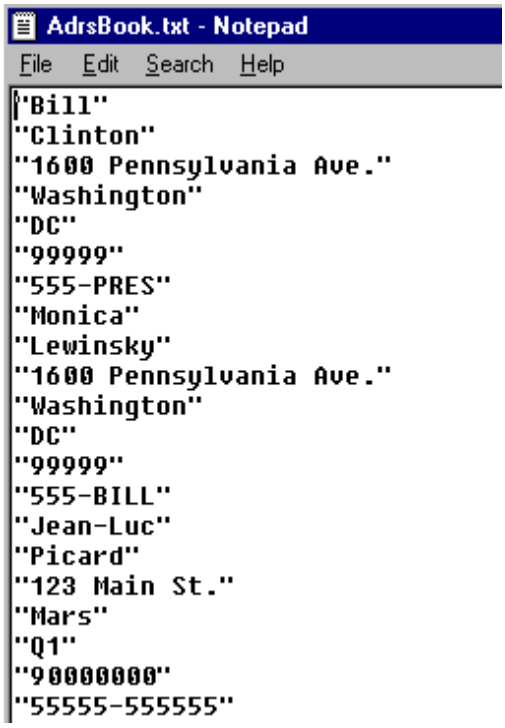
It must also be determined, before we start to code, what the File mode is going to be when we output to the file. We could use "Output" but that would mean that every time that we want to add a new listing, we wipe-out the file. Not very practical! Therefore, we will use "Append" so that all new entries are added to the end of the existing file.

Finally, once the controls are in place on the form, we have to finalize the order in which we Tab through them when working from the keyboard. That is called the **Tab order**. To set the tab order, we use the **TabIndex property** for each control. It starts at 0 and goes up for every control in order. When the form opens, the control with TabIndex=0 gets focus; when you tab from that, focus goes to TabIndex=1, and so on. Controls that don't get focus - Labels, Pictures, etc. - do have a TabIndex but their **TabStop property** is set to False. If you don't want Tab to stop on a control, set its TabStop to False.

Here is what the Sequential Output form will look like when we use it:



Once the file has been created we can use Notepad to look at it. Notice that the last entry, the one on the form above, is not yet in the file. It gets written only when you hit the Write button. Each field entered is stored as a separate line in the file. When we read them, we read in the same order as that in which they were written.



Creating the Sequential Output form

The form SAdresOut is used to capture data from the user and then output that data to the AdrsBook.txt file. The design of the form is what you see in the diagram above.

As you can see, we need 7 TextBox controls to capture the 7 fields. To simplify the code, we will use a technique we haven't used before in these lessons: the **Control Array**. You may have seen that come up before if you tried to copy and paste controls. What we do is: create one TextBox control, give it a name - we call it "txt_field" -, and then copy that control and paste it 6 times on the form. When you paste a control, since it has the same name as the existing one, the editor asks whether you want to give it a new name or create a control array. In this case we tell it to create the control array. This means that, instead of 7 different TextBoxes, we will have an array of TextBoxes, named txt_field(0) to txt_field(6). As you can see from the code, this allows us to use For ... Next loops to do things like clear the controls and write to the file.

The Cancel button simply clears all the TextBoxes and does not executes a Write operation. The Exit button closes the open files and unloads the form which returns us automatically to the Menu form. There is no End statement, as that would cause the program to end.

```
Form                                    Load

Option Explicit

Private Sub Form_Load()
    'Make sure all TextBoxes are blank

    Dim intCnt As Integer
    For intCnt = 0 To 6
        txt_field(intCnt).Text = ""
    Next intCnt

    'You can specify any path you want for the file:
    'Open "C:\VBApps\AdrsBook.txt" For Output As #1

    'Now, if we want to keep adding to the file instead of
    'overwriting it every time, we change the Open
    'to read: For Append instead of For Output
    'and use this next statement for the Open:

    Open "C:\VBApps\AdrsBook.txt" For Append As #1
End Sub
```

The code to write to the file is fairly straightforward. Once information has been entered into the 7 TextBoxes, we use a FOR ... NEXT loop to execute the Write command. The reason for this is that the Write command outputs only one field at a time. So, we have to do 7 writes to output the whole record. After the TextBoxes have been written-out, we clear them to create the next record.

```
Private Sub cb_write_Click()
    Dim intCnt As Integer
    Dim intCnt2 As Integer

    For intCnt = 0 To 6
        Write #1, txt_field(intCnt).Text
    Next intCnt

    For intCnt2 = 0 To 6
        txt_field(intCnt2).Text = ""
    Next intCnt2
End Sub

Private Sub cb_cancel_Click()
    Dim intCnt As Integer
    For intCnt = 0 To 6
        txt_field(intCnt).Text = ""
    Next intCnt

    txt_field(0).SetFocus
End Sub

Private Sub cb_Exit_Click()
    Close
    Unload Me
End Sub
```

Top

Working with a Random file

For this exercise we will use the same Menu form that we started with but we'll create a new output file which we will call "PhoneBook.txt". Since this file format is different from the sequential, we can't use the same file to test the code. The PhoneBook file will have almost the same content as the AdresBook file. The only difference is that we'll add a field for PersonId at the beginning of each record. That will allow us to retrieve records using a record number.

User-defined data type

In addition to data types like String, Integer, Date and so on, you can also define your own data type. This type is called **structure** or **structs** in other languages. We will use it in our application to simplify our I/O operations since our I/O commands, **Put** and **Get** only handle one field at a time. What we do with the

user-defined data type is to create a new variable which contains a whole record.

The user-defined variable must be declared in a **module**. That's a program at the application level, not tied to any specific event. To create a module: **Menu bar --> Project --> Add module --> Open**. When you save the module, it will take the .BAS extension. The information contained in modules is available to all the forms in the application. This is what your first module should contain:

```
(General)                    (Declarations)

Type PhoneRec
    PersonId As Integer
    Fname As String * 15
    Lname As String * 15
    Street As String * 20
    City As String * 15
    State As String * 2
    Zip As String * 5
    Phone As String * 8
End Type
```

The **Type** statement creates a new data type; in this case, it's **PhoneRec**. Once it's been defined, the new type can be used like any other type, String or Integer, etc. to declare a variable:

Dim InRecord As PhoneRec

The individual fields in the structured variable can be accessed using dot notation:

Label5.Caption = InRecord.Fname
txt_city.Text = InRecord.City

When you define the fields within the new type, it's important to determine the length of each string. Random access is sensitive about record lengths. When you define a String field like: **Fname As String * 15** you determine that the size of the field will always be 15 characters. **This is important for the processing to work properly!** Just make sure that the size you assign is big enough to handle most situations. You do not have to worry about the Integer field because its size is standard.

Writing and Reading records

The command to write records to the Random file is **Put**. Its format is:

Put #Filenumber, [RecordNumber], Variable

RecordNumber is optional and, if it's omitted, variable is written in Next record position after last Put or Get statement.

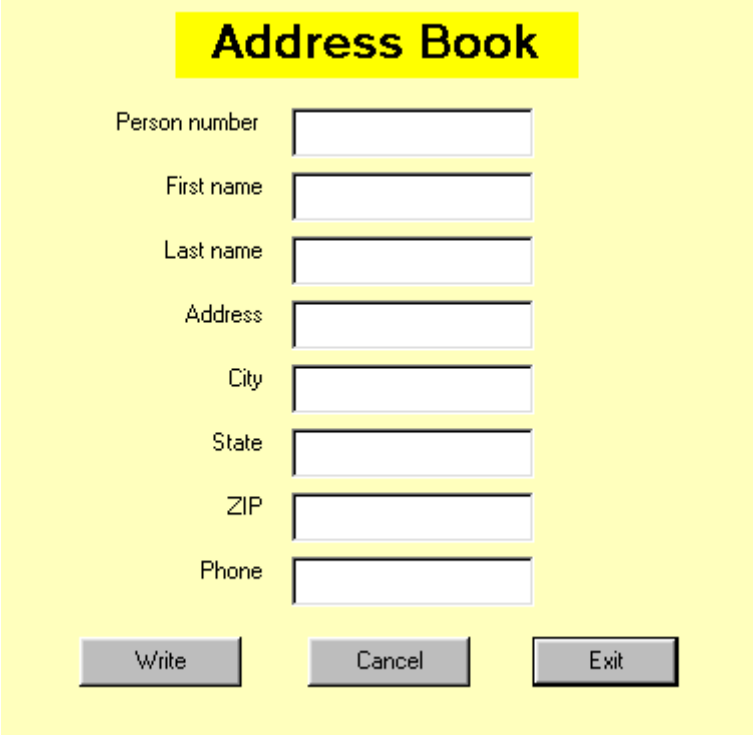
The command to read records from a Random file is: **Get**. Its format is:

Get #FileNumber, [RecordNumber], Variable

If RecordNumber is omitted, next record is read from the file.

Creating the Random file

To create the PhoneBook file, we will need a new form which is just a copy of the SAdresOut form with the additional Person number TextBox, which is in fact the record number. Then we'll write the code, making use of the user-defined data type "**PhoneRec**" described earlier. This form, "**RAdresOut**", obtains the next record number from the file, accepts input from the user and writes- the record out to the file.



```
Form                                    Load

Option Explicit
    Dim OutRec As PhoneRec
    Dim position As Integer
    Dim lastrecord As Integer

Private Sub Form_Load()
    'Make sure all TextBoxes are blank
    Dim intCnt As Integer
    For intCnt = 0 To 7
        txt_field(intCnt).Text = ""
    Next intCnt

    'To keep this example separate from
    'the Sequential file create a new file.
    Open "C:\VBApps\PhoneBook.txt" For Random As #1

    'Read the file until the end
    'Get without position is a "Read next".
    Do While Not EOF(1)
        Get #1, , OutRec
    Loop
    'Seek(1) returns number of current record,
    'which is End so, subtract 1 to get last valid.
    lastrecord = Seek(1) - 1
    position = lastrecord
End Sub
```

```
Private Sub cb_write_Click()
    Dim intCnt As Integer

    On Error GoTo errRtn

    position = position + 1   'add 1 to record number
    txt_field(0).Text = position
    OutRec.PersonId = position
    OutRec.Fname = txt_field(1).Text
    OutRec.Lname = txt_field(2).Text
    OutRec.Street = txt_field(3).Text
    OutRec.City = txt_field(4).Text
    OutRec.State = txt_field(5).Text
    OutRec.Zip = txt_field(6).Text
    OutRec.Phone = txt_field(7).Text


    Put #1, position, OutRec    'write output

    For intCnt = 0 To 7         'clear TextBoxes
        txt_field(intCnt).Text = ""
    Next intCnt
errRtn:
    Resume Next     'basically, ignore error
End Sub
```

To read records from the file, we have to specify a record number. This number is accepted into the Person number TextBox and then used to locate the appropriate record in the file.

The error-trapping routine is useful in this procedure because you are almost certain to encounter the "Reading past End-of-file" error when you enter a Person number that does not exist.

```
(General)                    (Declarations)
Option Explicit
    Dim InRec As PhoneRec

Private Sub Form_Load()

    'Make sure all TextBoxes are blank

    Dim intCnt As Integer
    For intCnt = 0 To 7
        txt_field(intCnt).Text = ""
    Next intCnt

    Open "C:\VBApps\PhoneBook.txt" For Random As #1
End Sub
```

```
Private Sub cb_read_Click()
    On Error GoTo errRtn

    Dim intCnt As Integer
    Dim intCnt2 As Integer
    Dim position
    position = txt_field(0).Text

    Get #1, position, InRec

    txt_field(1).Text = InRec.Fname
    txt_field(2).Text = InRec.Lname
    txt_field(3).Text = InRec.Street
    txt_field(4).Text = InRec.City
    txt_field(5).Text = InRec.State
    txt_field(6).Text = InRec.Zip
    txt_field(7).Text = InRec.Phone

errRtn:
    Resume Next      'continue processing at
                     'next statement - ignore error
End Sub
```

Working with a database

Creating the database

In the final 2 lessons in this tutorial, we will be developing a VB Project on "Project Management". The actual database we will be using has already been modelled and created in the tutorial on **Database Design and SQL, Lesson 2**.

The Project Management example

ezConsulting Inc. is a company doing IT systems consulting work with a large number of clients. At any given time there are dozens of projects on the go, each employing several employees. In a given period (in this case, weekly) an employee could work on several different projects. In order to track costs and revenues for each project, each employee will submit a timesheet every week showing the number of hours spent on each project. And, since all employees are attached to only one department, costs and revenues can be calculated by department.

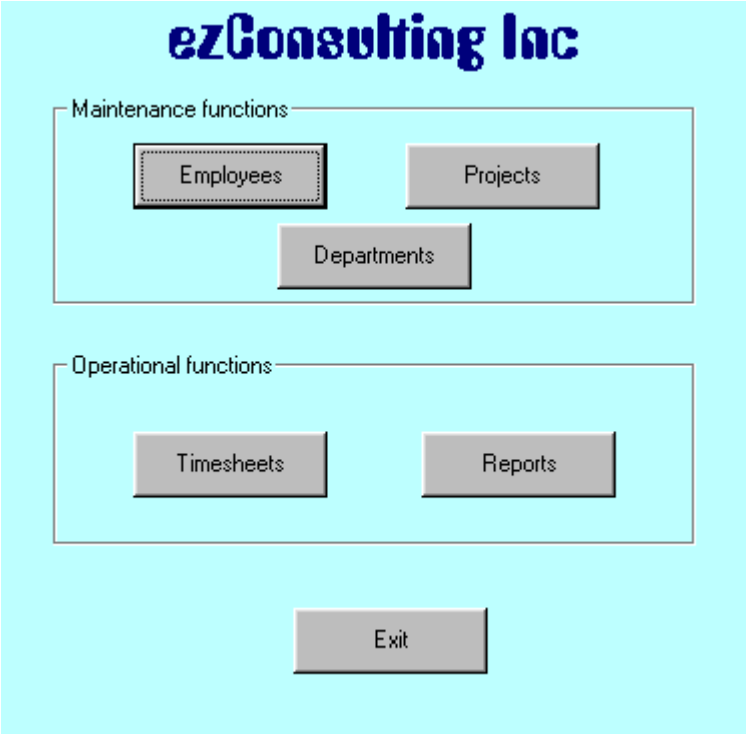
It has already been determined that the ProjectMgt database will consist of the following tables:

- **Employees**: details on every employee - ID, name, address, telephone, date hired, salary, chargeout rate, department
- **Projects**: details of every project - project number, title, budget, start date, end date
- **Departments**: lookup table of departments - number, name, head
- **Timesheets (Master/Detail)**: tables to store time spent on projects - date, employee, project, number of hours

The first task to be developed in the application consists of **table maintenance**. For each of the main tables, Employees, Projects and Departments, there have to be ways to add new records, remove records that are no longer needed and change records when appropriate. For example, new employees are hired and other employees leave, the name of a department is changed or a new project is started. Each of these maintenance operations will require a separate form.

Once the maintenance functions are in place, and they have to be (remember: **referential integrity** dictates that you can't accept a timesheet for a non-existant employee or non-existant project), we can start working on the **operational functions**, entering timesheets and producing reports. There will be forms for these tasks also.

To make it easier to access the different forms, we will create an **Application Menu** like we did in the previous lesson. The layout of the Menu form is standard and the code consists of a series of Load and Show statements for the various forms.



Top

Version problems

VB 6 and Access 2000 have compatibility problems. Because VB 6 was released before Access 2000, it does not normally recognize the Access 2000 format. In the example that follows, look at the **Connect** property of the **Data control**. If you don't have Access 2000 in the choices when you open "Connect", you have an older version of VB. If you try to connect to an Access 2000 database, you will get a message saying that you have an "**Unrecognized database format"**. If you have an older version of VB6, you will have to get the fix for it. You may be aware that Microsoft regularly publish upgrades to their software products (not that they admit that there are problems with them!). Those upgrades are called **Service Packs**. Right now, Visual Studio (which includes Visual Basic) is at Service Pack 5. By the time you read this that may have changed. So, to fix your compatibility problem you will have to download the latest Visual Studio Service Pack from Microsoft.

There is a quick fix to the problem, which is what we've done here to save you the trouble of having to download. You can convert your Access 2000 database to Access 97 and use your old VB.

To do that in Access 2000, go to **Tools -->Database utilities -->Convert** and that will do the trick until you have the time to upgrade VB.

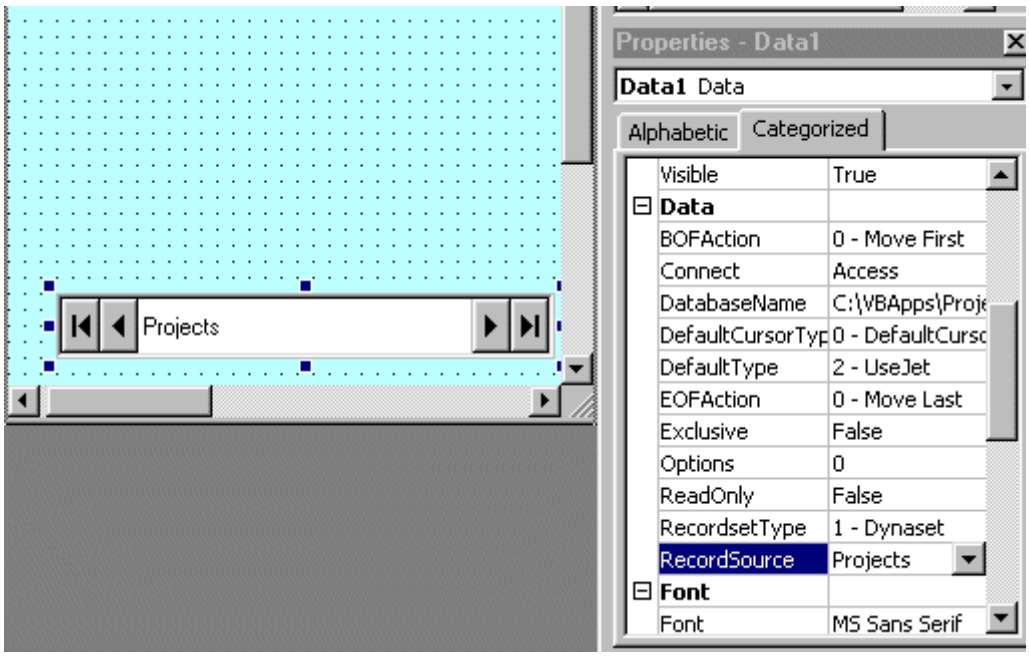
This will also come in handy later when we look at a VB Add-in called **Visual Data Manager**. Unfortunately, that Add-in does not work at all with Access 2000, even with the VB Service Pack. If you want to use it you will have to convert the database.

The Data Control

To begin the application, we will first create a new form for Projects maintenance: **ProjMaint**.

The first control we will place on the form, once we've set the basic form properties and saved it, is called the **Data Control**. It is the object which links a form to a database and allows you to access the fields in the tables making up the database. It's called **Data** in the Toolbox.

VB provides other objects that allow you to link to databases. **ADO (ActiveX Data Objects)** are certainly more powerful and more efficient than the Data Control. However, they do require a lot more coding and are more difficult to implement. Also, they are not available in the Standard Edition of VB, only in the Professional and Enterprise Editions. In simple applications, the Data Control, slow as it is, gives you a tool that is easy to implement and will provide most of the functionality you need.



The **arrow buttons** on the control are used to navigate through the database records:

First record and **Previous**

Next and **Last record**

The buttons correspond to 4 **methods** of the DC which you can use when you have to navigate using code. They are:

MoveFirst
MovePrevious
MoveNext
MoveLast

Let's look at the important properties of the Data Control:

- **Name:** the name to use in code - Data1 is default - eventually we'll have several data controls on the form - we'll call this one dta_proj.
- **Connect:** the kind of database - in this case it's Access - could be Foxpro, dBaseIV, etc.
- **DatabaseName:** the name and path of the database the control is connected to.
- **RecordSource:** the name of the database table being used.

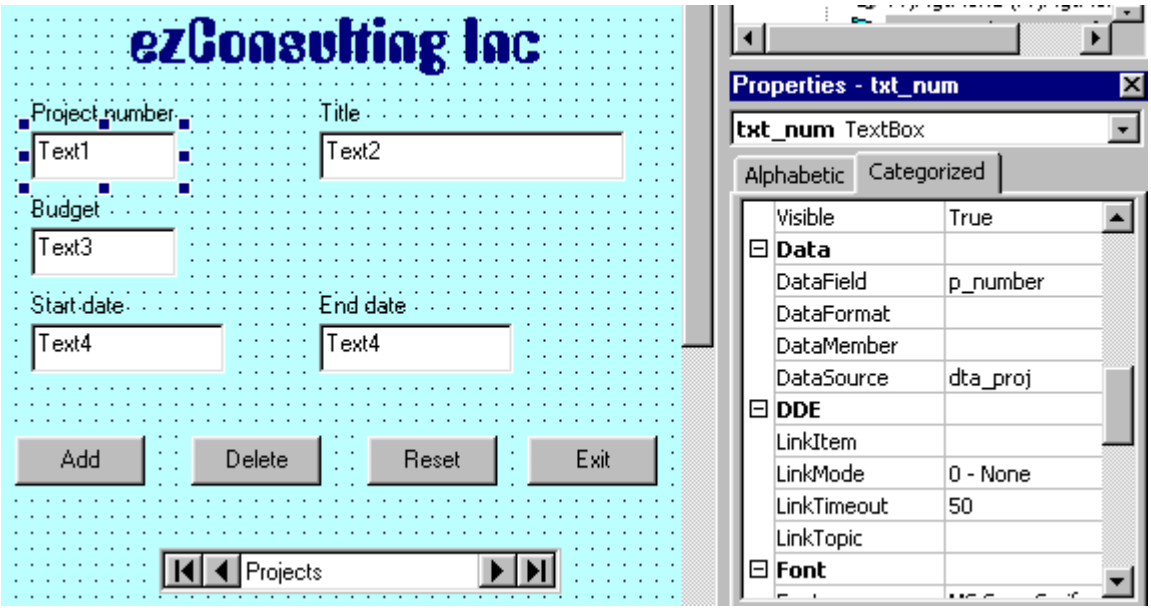
- **BOFAction** and **EOFAction**: action to take when trying to read before the beginning of file or past the end of file - we'll look at those later.

- **Recordset:** this is a run time property, and it's an **important one** - it represents the result of the query executed to the database - it contains all the records required by this Data Control - when you navigate through the database, you are actually navigating through the recordset, which is then mapped back to the database - that is why the methods of the Data Control refer to the Recordset property.

Next we add the controls needed to look at the fields in the records. In many instances we will need to make changes to the data. Therefore, we'll use a TextBox for each of the fields so that we can both diaplay and enter data as needed. Each TextBox will be a **bound control**, meaning that it is bound or tied to a specific field from the database. When we navigate through the database using the arrow buttons the content of each TextBox will always reflect the content of the current field. To bind the control to the database field we use its **Data properties**:

- **DataSource** is the name of the Data Control - remember that the DC specifies the name of the database to use and the name of the table to access - tip: enter this one before the DataField.

- **DataField** is the name of the field to bind - that field is selected from the content of the table.



Top

Notice that we've also added several buttons to the form. These buttons represent the most common actions we have to perform on the records.

Important note: when you work with bound controls you have to remember that every time you move off a record, the record is automatically modified in the database - every change you make to a TextBox is reflected immediately in the table. That is why there is no Update button - the function is done automatically.

The **Reset** button allows you to cancel any changes you may have made to the fields - if you modified information and then change your mind about updating it, the Reset will skip the automatic update and return the fields to their original state. This is a method of the Data Control object and is written as:
dta_proj.UpdateControls

There are 2 ways to **Add** new records to the table:

- in the Data Control, dta_proj, set the **EOFAction property = 2** - this will allow the user to go to Last Record and do a Next, which will add a blank record to the table;

- use the AddNew method of the Data Control, as in:

  dta_proj.Recordset.AddNew

To **Delete** the current record, you must use the **Delete** method followed by a **MoveNext** to move off the deleted record:
dta_proj.Recordset.Delete
dta_proj.Recordset.MoveNext

```
cb_add                                  Click
Private Sub cb_add_Click()
    dta_proj.Recordset.AddNew
    txt_budget.Text = ""
    txt_num.SetFocus
End Sub

Private Sub cb_del_Click()
    dta_proj.Recordset.Delete
    dta_proj.Recordset.MoveNext
End Sub

Private Sub cb_exit_Click()
    Unload Me
End Sub

Private Sub cb_reset_Click()
    dta_proj.UpdateControls
End Sub
```

Top

Validating data

Before the data you are entering get permanently transfered to the database, you often want to make sure they are correct. That is called **data validation**. We look here at two simple ways of validating data.

Let's say that the specs for the Projects maintenance part of the application call for three verifications:

- a budget amount must be entered;
- the budget amount must not exceed $1,000,000
- the project end-date cannot be earlier than the start-date.

For the first two we'll use the **Validate event** of the control. This event is triggered when the **CausesValidation property** in the TextBox is set to True. Before losing focus, the validation is done. The parameter assigned to the event when it is generated automatically (it's called Cancel) represents the KeepFocus property. If you set it to true in code when you encounter a validation problem, it keeps focus on the current control until the problem is corrected.

The second technique is to use the **LostFocus event**. When focus moves off the control, you do the

validation. If there is an error, you evoke the **SetFocus method** to put focus back to the control with the error.

```
'Validate event triggered by
'CausesValidation property of control
Private Sub txt_budget_Validate(Cancel As Boolean)
    'Error if budget not entered or
    'budget > 1,000,000 (arbitrary)
    If txt_budget.Text = "" Then
        MsgBox ("Must enter budget amount")
        Cancel = True   'KeepFocus on current control
    ElseIf txt_budget.Text > 1000000 Then
        MsgBox ("Budget amount too big")
        Cancel = True
    End If
End Sub

Private Sub txt_end_LostFocus()
    'Error if end date is before start date
    If txt_end.Text < txt_start.Text Then
        MsgBox ("Incorrect end date")
        txt_end.SetFocus
    End If
End Sub
```

Finding a specific record

When you navigate with the arrow buttons or the Move... methods you are necessarily moving one record at a time. Very often there is a need to access a specific record in the database. For example, it might be to change the ending-date for the project called "XYZ Corp. Payroll System".

In this example we assume that the search will be conducted on Project title. It could be on Number or End-date if necessary and it would just involve minor changes to the code. We also assume that the user does not want to enter the full project title and will only input the first few characters; we will therefore make use of the "Like" operator to match the recordset to the search string.

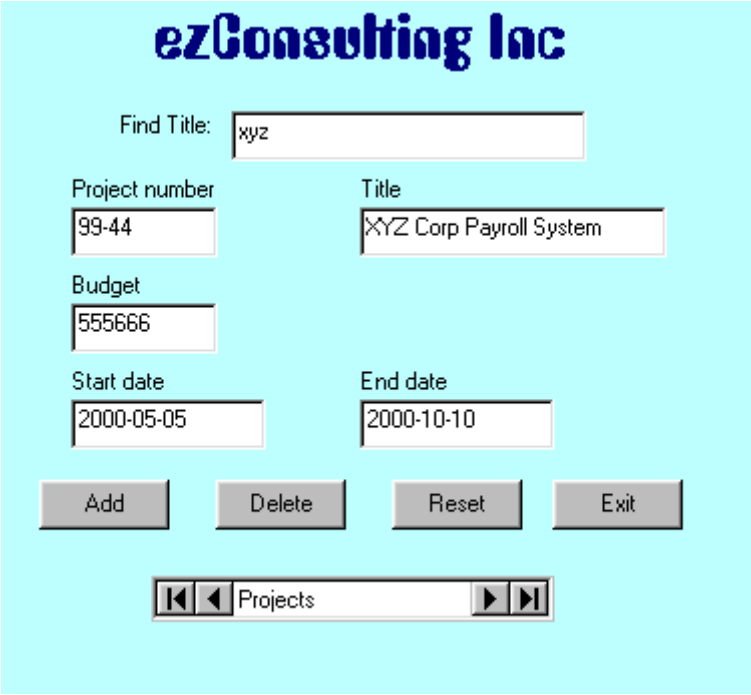
First, we create a **new TextBox**, called **txt_findTitle**, to enter the search string. We will give this TextBox the **TabIndex 0** because we want it to be the first control read when we look at a record. As soon as we move off the TextBox, the **LostFocus event** is triggered and checks whether the user has entered a search string or not. If there is no input into the search string, the user can work with the current record in the form. If there is a search string specified, the appropriate record will be loaded into the form.

The **FindFirst method** of the DC will locate the first occurence in the recordset matching the "content" parameter. If there are more than one records that match, the user then navigates forward using the arrows. The format of the FindFirst method is:

DataControl.Recordset.FindFirst "fieldname = 'searchstring'"

If the fieldname contains a string value, you have to use single quotes to name the searchstring; you can use the other comparison operators in place of the =.

This technique can be adapted to search any field in the recordset for a specific record. There are also **FindNext, FindPrevious** and **FindLast** methods for the Data Control recordset.




```
Private Sub txt_findTitle_LostFocus()
    Dim content
    'Build search string;
    'append * to the end in order to use
    'the "like" operator;
    'user only has to enter first characters
    content = Trim(txt_findTitle.Text) & "*"
    content = "p_title like '" & content & "'"
    'If search string was entered, FindFirst
    'will locate first occurence of it
    If txt_findTitle.Text <> "" Then
        dta_proj.Recordset.FindFirst content
    End If
End Sub
```

LESSON 10 - Working with a database ...part 2
Tuesday, August 02, 2011

Using multiple tables

Our ProjectMgt application contains an Employee table and a Department table linked through the employee's department number.

Employee : Table

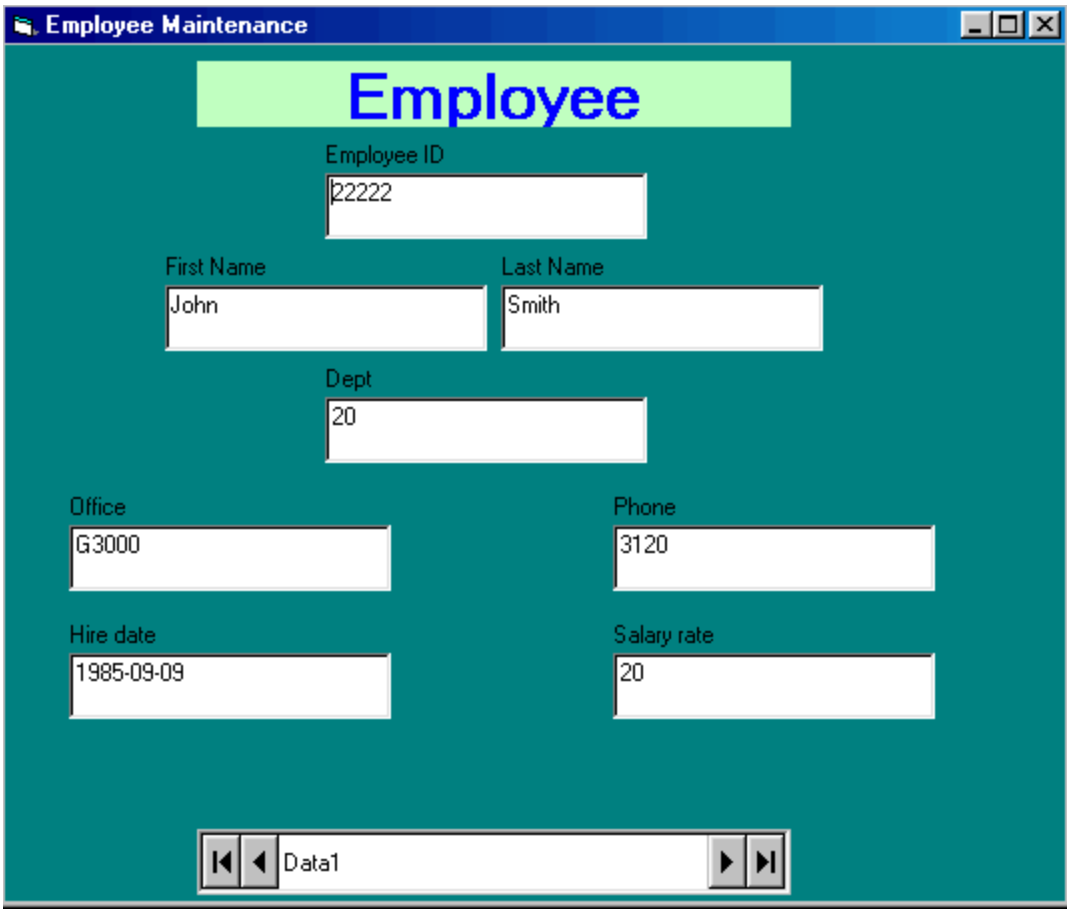
e_Id	e_Lname	e_Fname	e-Dept	e_Office	e_Phone	e_HireDate	e_HourlyRate
11111	Latreille	Michael	10	F2290	2134	1990-10-10	$45.00
22222	Smith	John	20	G3000	3120	1985-09-09	$20.00
33333	Bourget	Louise	20	G3200	3330	1989-12-12	$50.00
44444	Bashir	Mohamed	10	F2350	2150	1988-01-11	$35.00
55555	Nguyen	Han	30	B2200	2200	1992-03-03	$30.00
77777	Nguyen	Luu	20	B2200	2200	1992-03-03	$20.00
			0				$0.00

Department : Table

d_Number	d_Name	d_Head
10	Systems Analysis	Lucy Tremblay
20	Programming	Joseph Mubala
30	Network support	Arnold Schmit
0		

Relationship between Department and Employee is on dept_number

Now, if we create a form for Employee maintenance using the same technique we used in the previous lesson, we can access all fields in the Employee table using one data control, **Data1**. This is what the basic form will look like, before we get to put in the usual improvements:

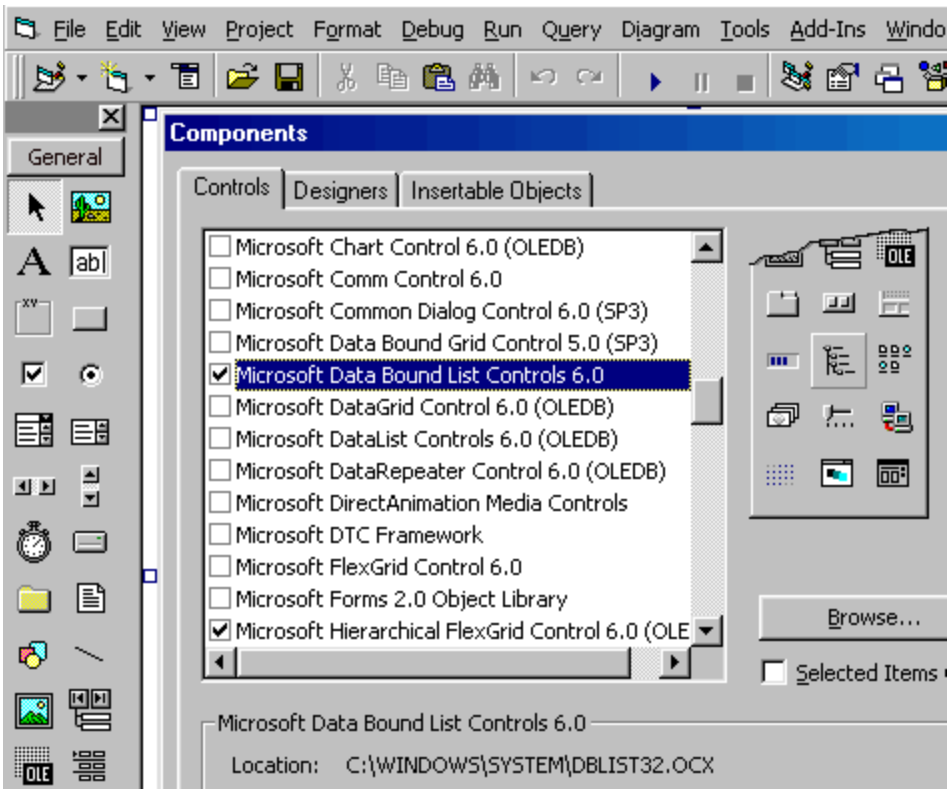


But suppose I want to select the employee's department from a list rather than keying in the department number.

For this we need a new control - it's called the **DBList control**. It's not in the standard toolbox, we have to get it.

For that: go to the menu bar -->Project -->Components -->Microsoft Data Bound List Controls 6.0 and put a check in the box then click OK.

Once you've done that two new controls appear in the toolbox.



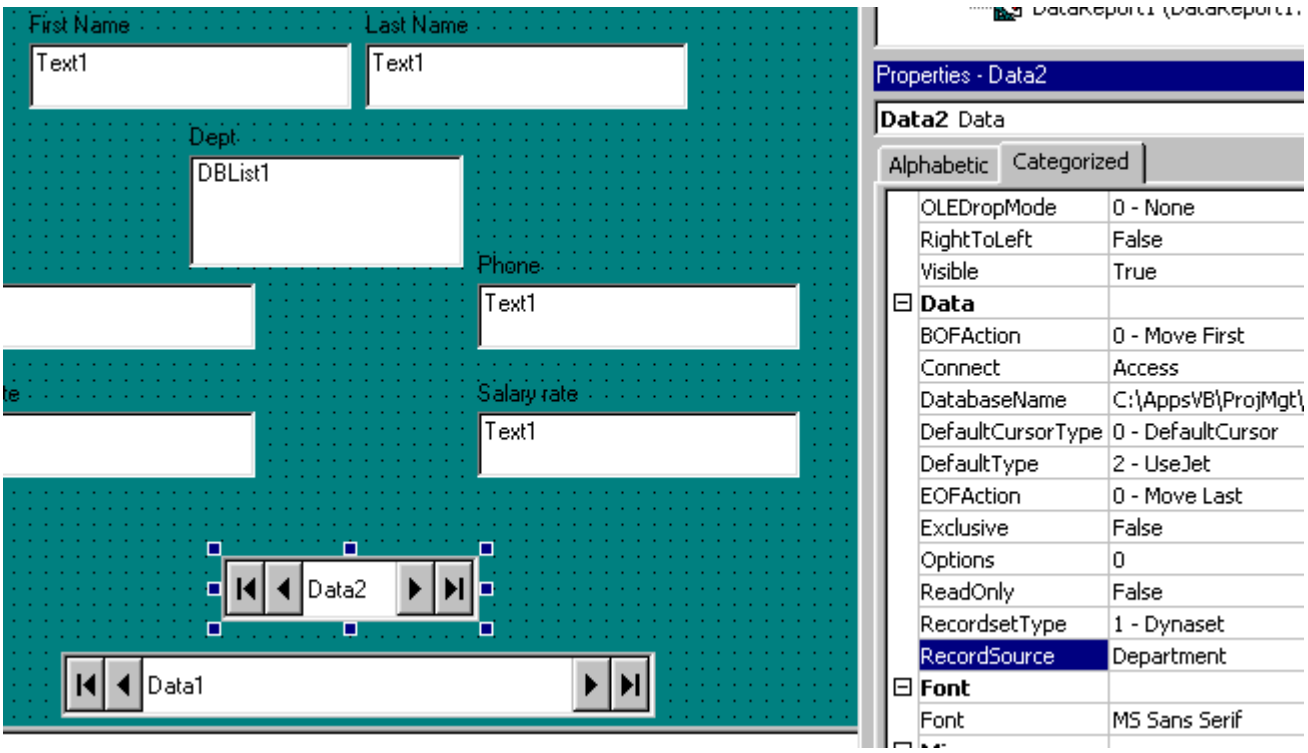
Now to get the department names to appear in the list.

That means I will have to access the Department table, in addition to the Employee table.

Remember: you need one data control for every table you want to access.

So first, create a second data control, **Data2**, on the form. It doesn't matter where we put it, we're going to make it invisible anyway.

Data2 has to have the same Connect property and the same DatabaseName as Data1 but, the RecordSource must specify: **Department**.



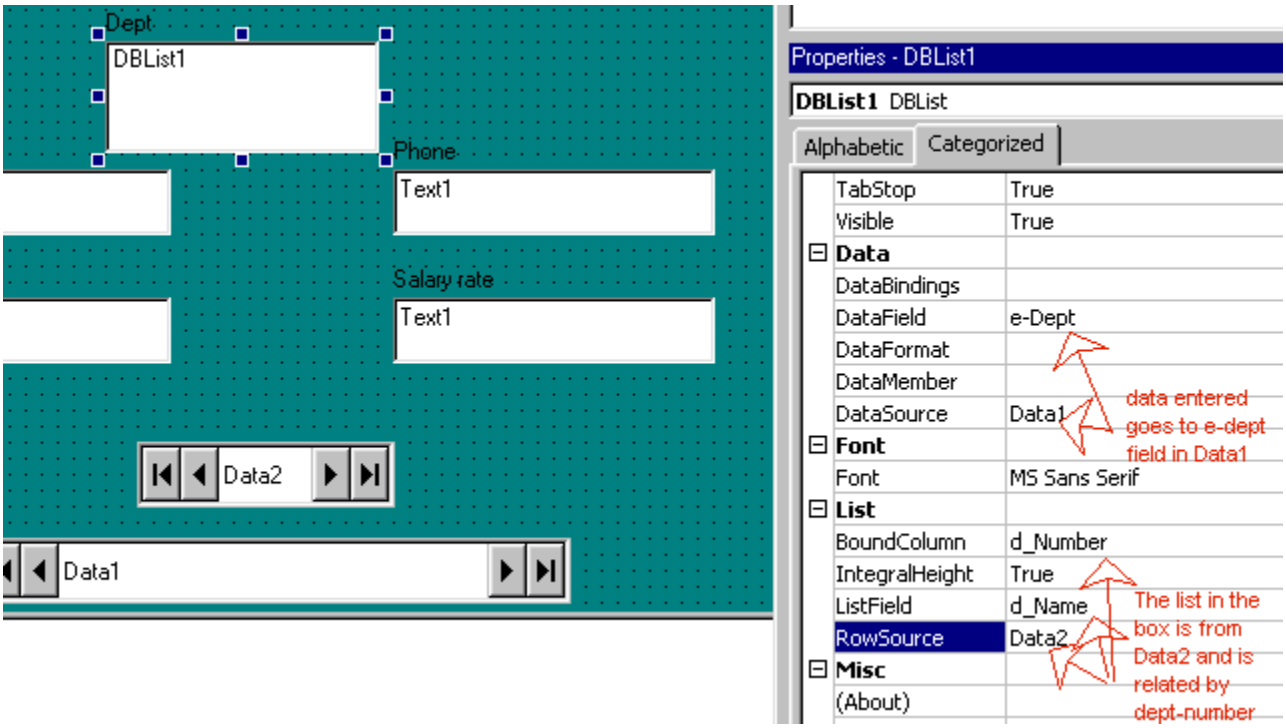
Now to get the list right.

First, we delete the department_number TextBox. Then we add a DBList.

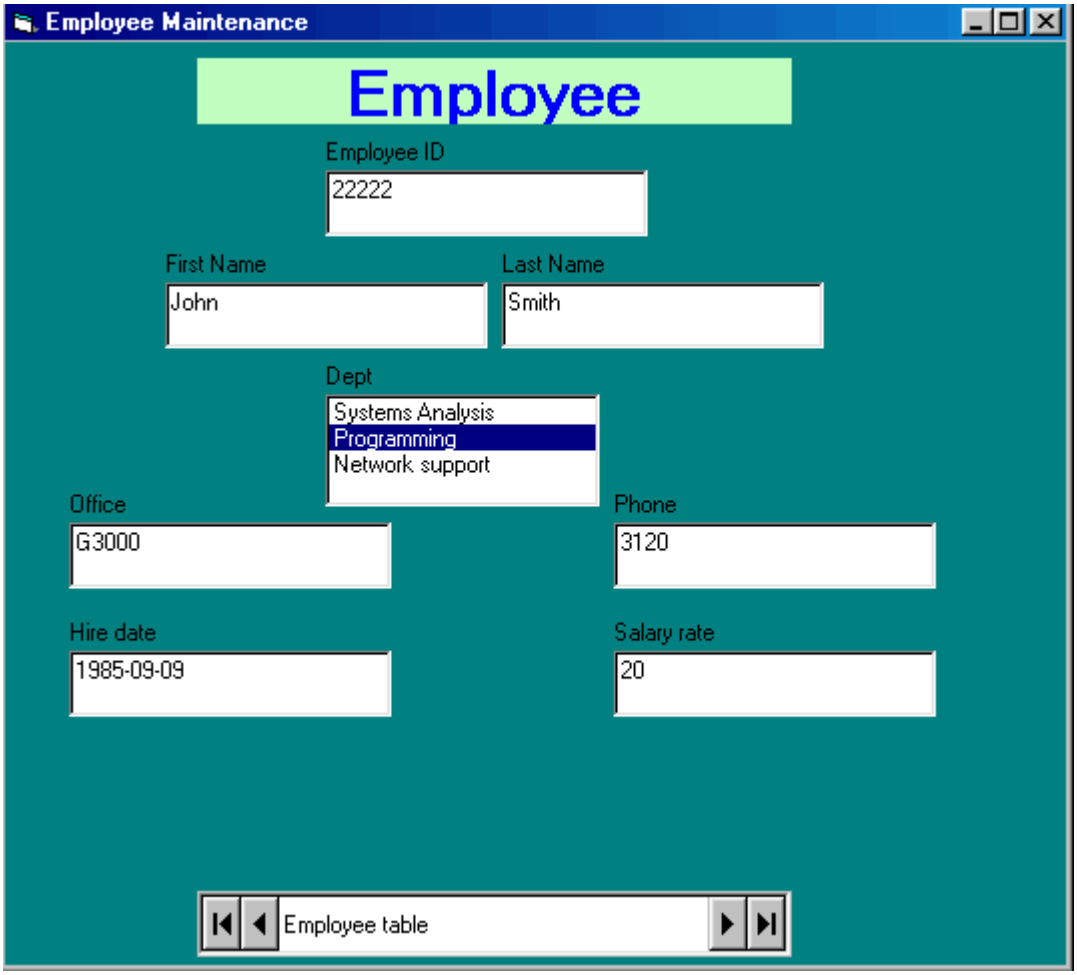
Now we specify the DBList1 properties. **Careful! This is where most people hit a snag!**

The **Data properties:** these specify where the data entered will be stored. We are in the Employee table. That's Data1. So, the data entered will go into **DataSource: Data1** and the field into which it is going is **DataField: e-Dept**.

The **List properties:** these tell the control where to get the information to show in the list. Since we want it from the Department table, we specify **RowSource: Data2**. What will appear in the list is the Department name so we choose **ListField: d_Name**. Finally, there has to be a link between Data2 and Data1. That is always the field which is the **primary key** in the list table and that is the **BoundColumn: d_Number**.



And once everything is cleaned-up, the Data2 control is hidden, we get the final result:


Employee Maintenance
Employee
Employee ID
22222
First Name
John
Last Name
Smith
Dept
Systems Analysis
Programming
Network support
Office
G3000
Phone
3120
Hire date
1985-09-09
Salary rate
20
Employee table