Erode Arts and Science College (Autonomous)

Department of Computer Science (SF)

VISUAL PROGRAMMING

Staff Name: M.DIVYA, M.Sc., M.Phil., **Course : III B.Sc (CS)**

Semester : V

Syllabus

UNIT- I

Getting started with Visual Basic 6.0: Introduction- Visual Basic 6.0 Programming Environment-Working with forms- Developing an application-Variables, Data type and Modules-Procedures and control structure- arrays in Visual Basic.

UNIT- II

Working with controls: Introduction-Creating and using controls-Classification of controls- Tab Index Property of controls- Using Textbox Controls- Using a Label control- Using a Command Button- Using Option Button Control- Using List Box and Combo box Control- -Using the Scrollbar Control- working with control Arrays: Creating a control arrays at design time-Adding a control array at runtime-Control Array applications.

UNIT- III

Menu, Mouse event and dialog boxes: Introduction-Menu interface-using the menu editor-Pop-Up Menus, Mouse Events: Graphical Mouse Application-Dragging and dropping. Dialog boxes: Model and modeless dialog boxes –predefined dialog boxes.

UNIT- IV

Graphics, MDI and flex Grid: Introduction- Graphics for application- Fundamentals of graphics – Using Graphical Control – Using Graphics Methods. Multiple document Interface (MDI)- Creating an MDI Application- Adjusting the Text box – Creating a tool Bar- Displaying a Status bar -The Flex Grid Control-Simple Programs.

UNIT- V

ODBC And Data Access Object: Evaluation of computing Architecture-Data Access option. ODBC using Data Access object and remote data Objects: Open Database Connectivity (ODBC) –Remote Data Object. Data Environment and data Report: Data Environment Designer-Data Report.

TEXT BOOK:

Content development Group, "Visual Basic 6.0 Programming", TATA McGraw-Hill Publications, 2007.

Unit –I : Chapter-1, **Unit – II :** Chapter : 2, **Unit – III :** Chapter : 3, **Unit – IV :** Chapter : 4, **Unit –V :** Chapter : 5,6,7

REFERENCE BOOK:

Gary Cornell, "Visual Basic 6 from the Ground Up", TATA McGraw-Hill Pub 2008.

UNIT – I

Getting started with Visual Basic 6.0:

Introduction:

- **VISUAL BASIC** is an extremely flexible programming product designed for a variety of applications. Also it is used to build powerful commercial applications and corporate productivity tools.
- **VISUAL BASIC** is a **high level programming language**; it was evolved from the earlier DOS version called BASIC.
- **BASIC** means **B**eginners' **A**ll-purpose **S**ymbolic **I**nstruction **C**ode.
- Different version of **BASIC**, such as **Microsoft QBASIC**, **QUICKBASIC**, **GWBASIC**, **IBM BASICA** and so on.
- Now people only use **Microsoft Visual Basic** today, because it is a well developed programming language and supporting resources are available everywhere.
- BASIC, programming is done in a **text-only environment** and the program is executed sequentially.
- In VISUAL BASIC, programming is done in a **graphical environment.**
- In the old BASIC, you to have to **write program codes for each graphical object** you wish to display it on screen, including its position and its color.
- In Visual Basic, you just need to **drag and drop any graphical object anywhere on the form**, and you can change its **color any time using the properties windows.**

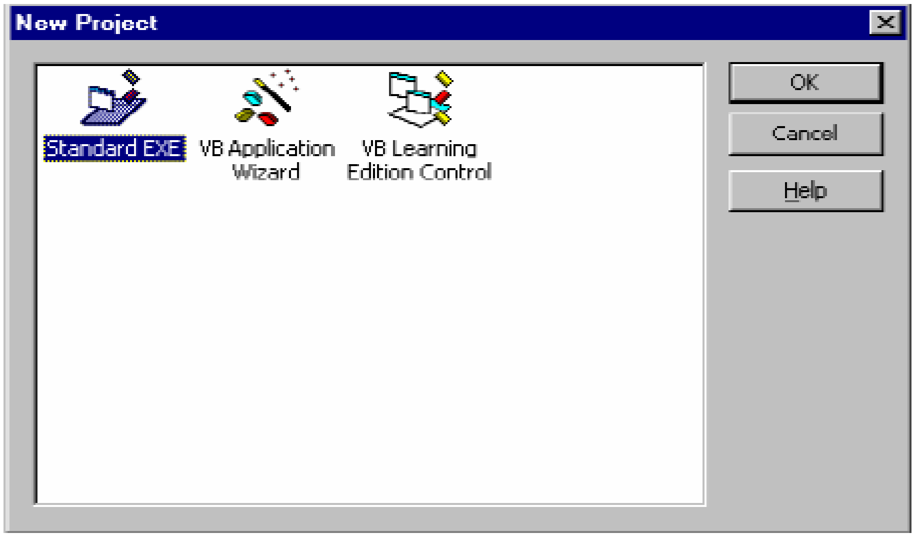
Visual Basic 6.0 Programming Environment

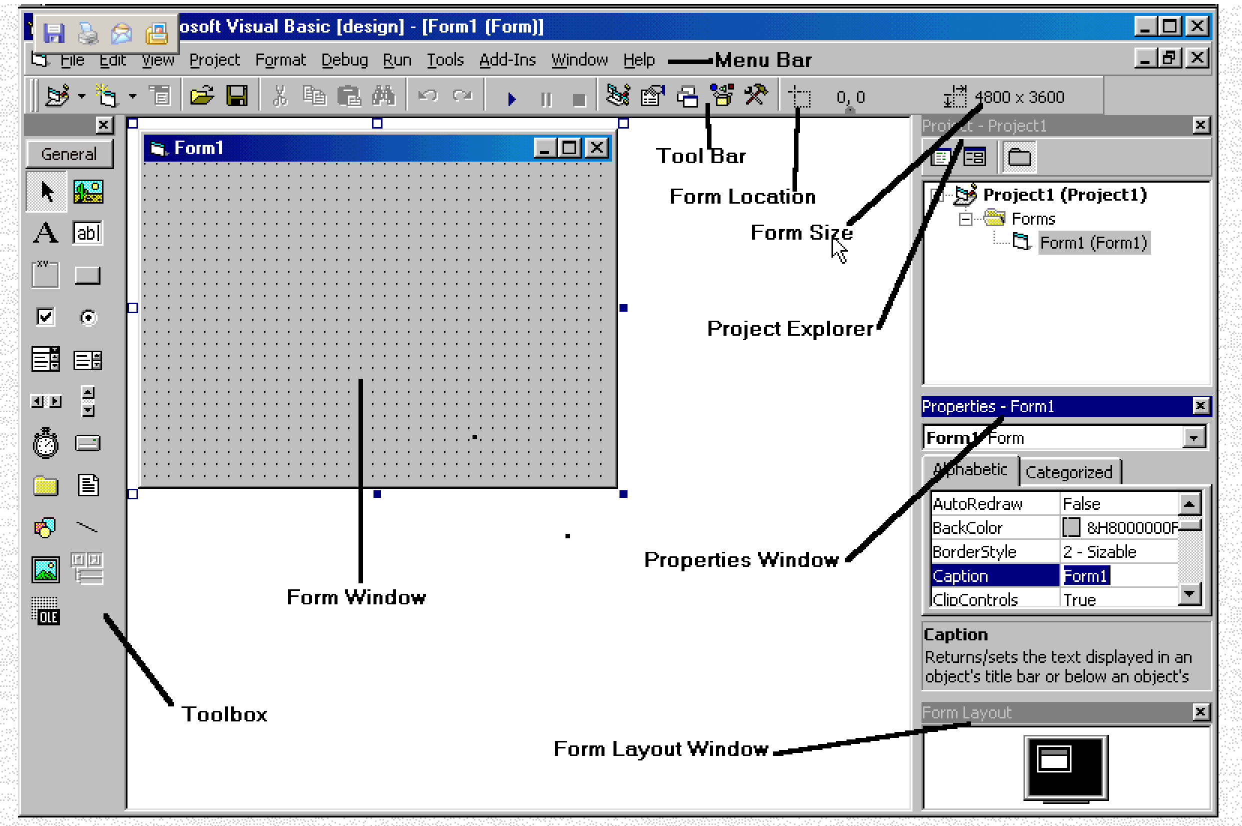
- Visual Basic 6.0 is the **Integrated Development Environment (IDE).** IDE is a term commonly used in the programming world to describe the **interface and environment that we use to create our applications.**
- It is called integrated because we can access virtually all of the development tools that we need from one screen called an interface.
- The IDE is also commonly referred to as the design environment, or the program.

STARTING VISUAL BASIC

- In Windows, click **Start**, point to **Programs**, and point to the **Microsoft Visual Basic 6.0** folder. The icons in the folder appear in a list.
- Click the **Microsoft Visual Basic 6.0** program icon. The **New Project** dialog box appears. This dialog box prompts you for the type of programming project you want to create. See the picture of New Project window below

.





AutoRedraw	False
BackColor	&H8000000F
BorderStyle	2 - Sizable
Caption	Form1
ClipControls	True

Caption
Returns/sets the text displayed in an object's title bar or below an object's

- The Visual Basic development environment contains **these programming tools and windows,** with you construct your **Visual Basic programs**

  - Menu bar
  - Toolbars
  - Visual Basic toolbox
  - Form window
  - Properties window
  - Project Explorer
  - Immediate window
  - Form Layout window

Menu Bar

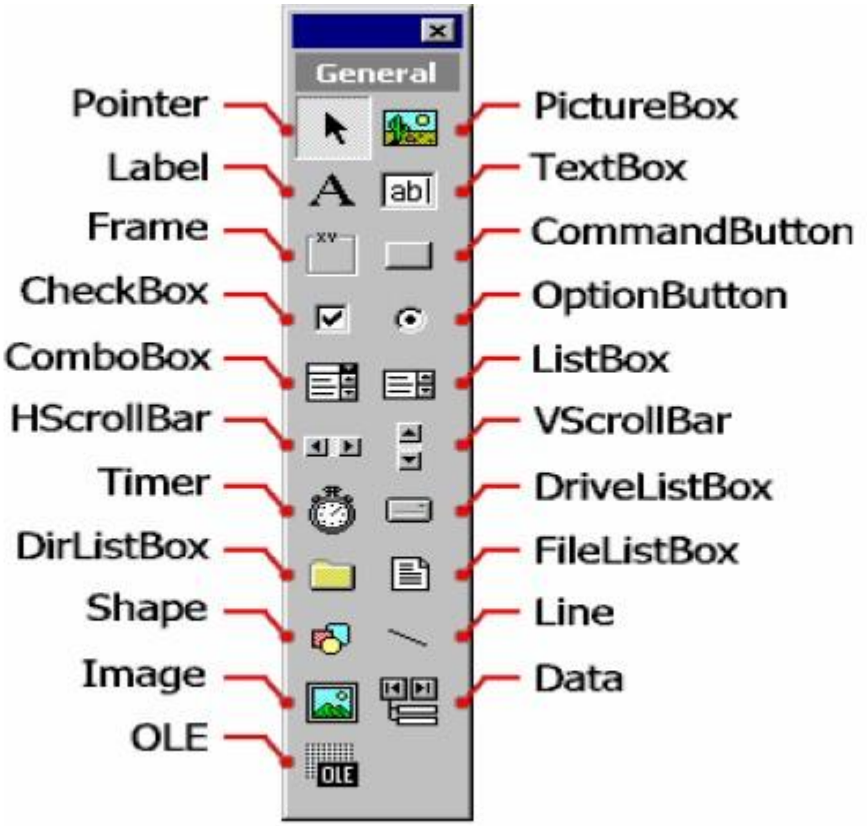
- Displays the commands you use to work with Visual Basic. Besides the standard **File**, **Edit**, **View**, **Window**, and **Help menu**, menus are provided to access functions specific to
- programming such as Project, Format, or Debug. The various options available in the **File Menu**, **Edit Menu**, **View Menu**, **Project Menu**, **Format Menu**, **Debug Menu**, and **Window Menu** can be selected as pull-down menu in the Menu Bar.

  - **File Menu**: The items in the File Menu are useful only when developing own applications by the user. The options available in File Menu are : **New Project, Open Project, Add/Remove Project, Save Project, Save As Project, Save Form, Save As Form, Print, Print Setup, Make .Exe, Make Project Group**, The Most Recently Used **List, Exit.**

  - **Edit Menu**: It contains more than 20 **items**. The most useful ones are : **Undo, Redo, Cut, Copy, Paste, PasteLink, Delete, SelectAll, Find, Find Next, Replace, Indent, Outdent, Inset File, List Properities / Methods,….,Complete Word and Book Mark.**

  - **View Menu**: The view menu lets you **display or hide features of Visual Basic environment.** The items on this menu are: **Code, Object, Definition, Last Position, Object Browser, Immediate Window, Local Window, Watch Window, Call Stack, Project Explorer, Project Window, Form Layout Window, Prosperities Pages, Toolbox, Color Palette, and Toolbars.**

TOOLBOX CONTROLS



The Toolbox, can be seen in probably the Window that you will become familiar with the quickest, as it provides access to all of the standard Controls that reside within the Visual Basic runtime itself.

- **Pointer:** The pointer is the only item on the Toolbox that isn't a Control. You can use it to select Controls that have already been placed on a Form.

- **Picture Box:** You use the Picture Box Control to display images in several different graphics formats such as BMP, GIF, and JPEG among others.

- **Label:** The Label Control is used to display text information that does not have a need to be edited by an end user. It's often displayed next to additional Controls such as Text Boxes to label their use.

- **Textbox:** You use Text Box Controls for user input. It may be the most widely used Control.

- **Frame:** A Frame Control is typically used for containing other Controls and for dividing the GUI. Controls placed within a Frame cannot be displayed outside of it, and if the Frame is moved on the Form, the Controls are moved with it.

- **Command Button:** Much like the Text Box Control, Command Button Controls are used for input on almost every Form. They are used as standard buttons for input like OK or Cancel.

- **Checkbox:** If you need the ability to select True/False or Yes/No, the Check Box Control is the correct Control.

- **Option Button:** The Option Button Control is similar to the Check Box Control in that it offers the ability to select an option. Option Button Control is most often used when a group of options exists and only one item can be selected. All additional items are deselected when a choice is made.

- **List Box:** The List Box Control contains a list of items, allowing an end user to select one or more items.

- **Combo Box:** Combo Box Controls are similar to List Box Controls, but they only provide support for a single selection.

- **Scrollbars:** The Scrollbar and Scrollbar Controls let you create scroll bars but are used infrequently because many Controls provide the ability to display their own Scroll Bars.

- **Timer:** The Timer Control is an oddity when it is compared to other Controls, in that it isn't displayed at runtime. It's used to provide timed functions for certain events.

- **DriveListBox, DirListBox, FileListBox:** These Controls can be used individually, but many times are used together to provide dialog boxes (also known as windows in this book) that display the contents of Drives, Directories, and Files. Shape, Line: The Shape and Line Controls are simply used to display lines, rectangles, circles, and ovals on Forms.

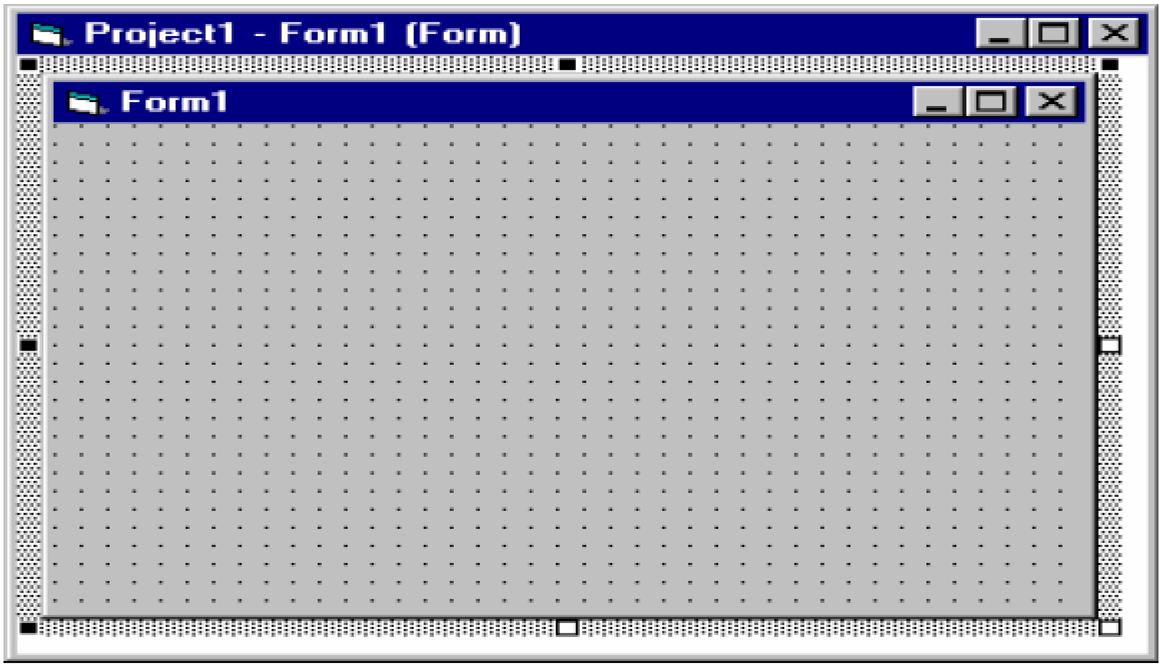
- **Image:** You can think of the Image Control as a lighter version of the Picture Box Control, and although it doesn't provide all of the functionality that the Picture Box Control does, it consumes fewer resources. As a result, you should use the Image Control whenever possible.

- **Data**: The Data Control is a component that allows you to connect one or more Controls on a Form to fields in a database.

- **OLE: The OLE (Object Linking and Embedding)** Control can host Windows belonging to other executable programs. For instance, you can use it to display a spreadsheet generated by Microsoft Excel or a Word document.

Working with forms

- A default form (Form1) with a standard grid (a window consisting of regularly spaced dots) appears in a pane called the Form window. You can use the Form window grid to create the user interface and to line up interface elements



Controlling Form Placement

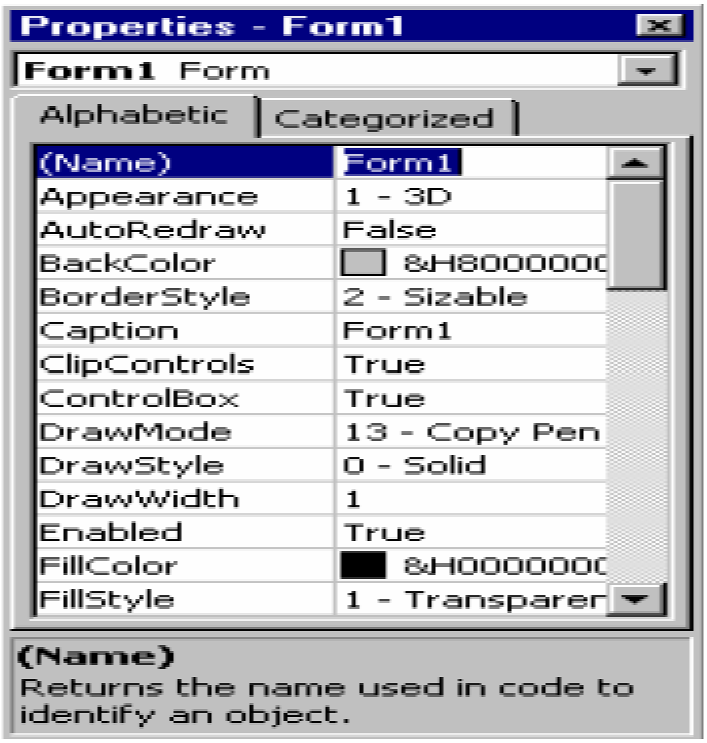
- To control the placement of the form when you run the program, adjust the placement of the form in the Form Layout window.

PROPERTIES WINDOW

- With the Properties window, you change the characteristics (property settings) of the user interface elements on a form. A property setting is a characteristic of a user interface object. For example, you can change the text displayed by a text box control to a different font, point size, or alignment. (With Visual Basic)

Displaying the Properties Window

- To display the Properties window, click the Properties Window button on the toolbar. If the window is currently docked, you can enlarge it by double-clicking the title bar. To redock properties window, double-click its title bar again.



(Name)	Form1
Appearance	1 - 3D
AutoRedraw	False
BackColor	□ &H8000000
BorderStyle	2 - Sizable
Caption	Form1
ClipControls	True
ControlBox	True
DrawMode	13 - Copy Pen
DrawStyle	0 - Solid
DrawWidth	1
Enabled	True
FillColor	■ &H0000000
FillStyle	1 - Transparer

(Name)
Returns the name used in code to identify an object.

Properties Window Elements

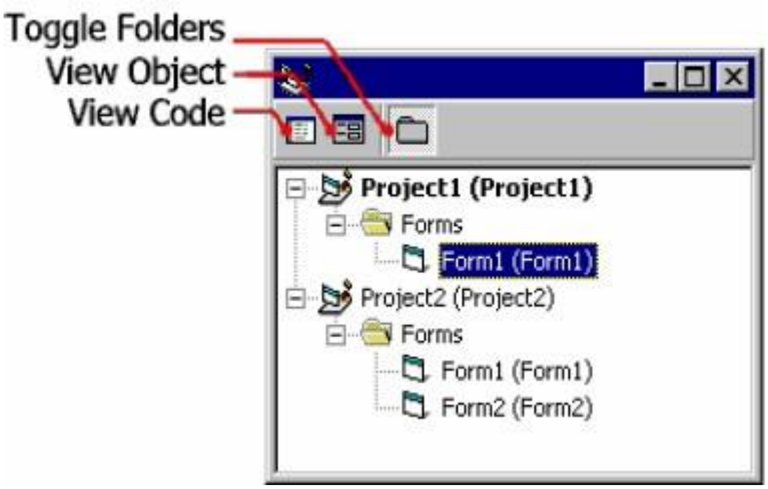
- ❖ The Properties window contains the following elements:
- ❖ A drop-down list box at the top of the window, from which you select the object whose properties you want to view or set.
- ❖ Two tabs, which list the properties either alphabetically or by category.
- ❖ A description pane that shows the name of the selected property and a short description of it.

Changing Property Settings

- You can change property settings by using the Properties window while you design the user interface or by using program code to make changes while the program runs.

PROJECT WINDOW

- A Visual Basic program consists of several files that are linked together to make the program run. The Visual Basic 6.0 development environment includes a Project window to help you switch back and forth between these components as you work on a project.



Project Window Components

- The Project window lists all the files used in the programming process and provides access to them with two special buttons: **View Code and View Object.**

Displaying the Project Window

- To display the Project window, click the Project Explorer button on the Visual Basic toolbar. If the window is currently docked, you can enlarge it by **double clicking** the title bar. To re-dock the Project window, double-click its title bar again.

Adding and Removing Files

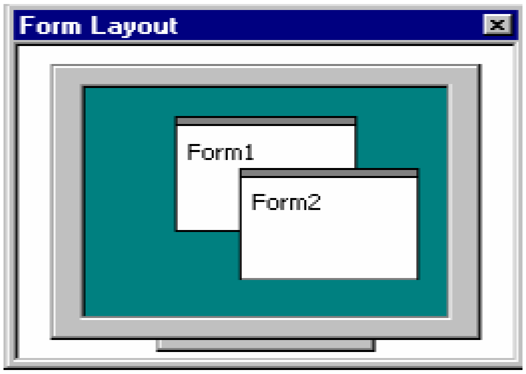
- The project file maintains a list of all the supporting files in a Visual Basic programming project. You can recognize project files by their **.vbp file name extension**. You can add individual files to and remove them from a project by using commands on the Project menu. The changes that you make will be reflected in the Project window.

Adding Projects

- If you load additional projects into Visual Basic with the File menu Add Project command, outlining symbols appear in the Project window help you to organize and switch between projects.

FORM LAYOUT WINDOW

The Form Layout window is a visual design tool. With it, you can control the placement of the forms in the Windows environment when they are executed. When you have more than one form in your program, the Form Layout window is especially useful.



Developing an application

A Visual Basic program is a Windows-based application that you create in the Visual Basic development environment. This section includes the following topics:

- Planning the Program
- Building the Program
- Testing, Compiling, and Distributing the Program.

Planning the Program

Identify your Objectives

- You should spend some time thinking about the programming problem you are trying to solve. Up-front planning will save you development time down the road, and you'll probably be much happier with the result.
- One part of the planning process might be creating an ordered list of programming steps, called an **algorithm.**

Building the Program

Building a Windows-based application with Visual Basic involves three programming steps:

- Creating the User Interface
- Setting the Properties
- Writing Program Code
- Saving a Project

Creating the User Interface

- The complete set of forms and controls used in a program is called the **program user interface.** The **user interface includes** all the menus, dialog boxes, buttons, objects, and pictures that users see when they operate the program.
- You can create all the components of a Windowsbased application quickly and efficiently.

Setting the Properties

- Properties are programmable characteristics associated with forms and their controls. You can set these properties either as you design your program **(at design time) or while you run it (at run time).**

- You change properties at design time by selecting an object, clicking the Properties window, and changing one or more of the property settings. To set properties at run time.

Writing the Program Code

- You finish building your program by typing program code for one or more user interface elements. Writing program code gives you more control over how your program works than you can get by just setting properties of user interface elements at design time.
- By using program code, you completely express your thoughts about how your application:
    - Processes data
    - Tests for conditions
    - Changes the order in which Visual Basic carries out instructions.

Saving a Project

- After you complete a program or find a good stopping point, you should save the project to disk with the Save Project As command on the File menu.

Testing, Compiling, and Distributing the Program

- Distribute your program, you also need to compile it into an executable program (a stand-alone Windows-based program) and give it to your users.
- If you decide to revise the program later, you repeat the process, beginning with planning and an analysis of any new goals. These steps complete the software development life cycle.

Testing and Debugging

- Testing a program involves checking it against a variety of real-life operating conditions to determine whether it works correctly. A problem that stops the program from running or from producing the expected results is called a software defect (bug).

Compiling

- You’ve finished creating and testing your program, you can compile it into an **executable (.exe)** file that will run under **Windows or Windows NT**.
- Creating an executable file with Visual Basic is as simple as clicking the **Make Project1.exe** command on the File menu.
- If you plan to run your new Visual Basic application only on your own system, creating the executable file will be your final step.

<u>Variables, Data type and Modules</u>

- Visual Basic uses building blocks such as **Variables, Data Types, Procedures, Functions and Control Structures** in its programming environment. This section concentrates on the programming fundamentals of Visual Basic with the blocks specified.

Modules

- Code in Visual Basic is stored in the form of modules. The three kinds of modules are **Form Modules, Standard Modules and Class Modules.**
- A simple application may contain a single Form, and the code resides in that **Form module** itself.
- As the application grows, additional Forms are added and there may be a common code to be executed in several Forms. To avoid the duplication of code, a separate module containing a procedure is created that implements the common code. This is a **standard Module**.
- **Class module** (.CLS filename extension) are the foundation of the object oriented programming in Visual Basic. New objects can be created by writing code in class modules. Each module can contain:

Declarations : May include constant, type, variable and DLL procedure declarations.

Procedures : A sub function, or property procedure that contain pieces of code that can be executed as a unit.

Variables

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 somebody 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 First-name use First_name or FirstName.

Data types in Visual Basic 6

- A variable is declared, a data type is supplied for it that determines the kind of data they can store. The fundamental data types in Visual Basic including variant are **integer, long, single, double, string, currency, byte and boolean.**
- Visual Basic supports a vast array of data types. Each data type has limits to the kind of information and the minimum and maximum values it can hold.

1. Numeric

Data Type	**Description**
Byte	**Integer values** in the range of 0 - 255
Integer	**Integer values** in the range of (-32,768) - (+ 32,767)
Long	**Integer values** in the range of (- 2,147,483,468) - (+ 2,147,483,468)
Single	**Floating point** value in the range of (-3.4x10-38) - (+ 3.4x1038)
Double	**Large floating value** which exceeding the single data type value

Currency	**Monetary** values. It supports 4 digits to the right of decimal point and 15 digits to the left

2. String

- ❖ To store alphanumeric values. A variable length string can store approximately **4 billion characters**

3. Date

- ❖ To store date and time values. A variable declared as date type can store both date and time values and it can store date values **01/01/0100 up to 12/31/9999**

4. Boolean

- ❖ Boolean data types hold either a **true or false** value. These are not stored as numeric values and cannot be used as such. Values are internally stored as -1 (True) and 0 (False) and any non-zero value is considered as true.

5. Variant

- ❖ Stores any type of data and is the default Visual Basic data type. In Visual Basic if we declare a variable without any data type by default the data type is assigned as default.

Creating a Variables

- ❖ Variables are declared using the Visual Basic **Dim** keyword. The **syntax** for a simple declaration of a variable is as follows:
  - ▪ **Dim variableName As variableType**
- ❖ **Dim** is the keyword which indicates to Visual Basic that a variable is being declared. variableName is the name assigned to the variable.

Example:

- o **Dim** intInterestRate As **Integer**
- o **Dim** intInterestRate, intExchangeRate As **Integer**
- o **Dim** strCustomerName As **String**, intInterestRate, intExchangeRate As **Integer**

Initializing Visual Basic Variables

- Visual Basic variables may be **initialized** either during the declaration, or after the declaration.

Itialization is performed using the Visual Basic **assignment operator** (=). To initialize a single variable when it is declared:

- **Dim** intInterestRate As **Integer** = 5
- **Dim** strCustomerName As **String** = "Fred", intInterestRate = 5 As **Integer**, intExchangeRate As **Integer** = 10

Procedures and control structure

Procedures in Visual Basic

- A ***procedure*** is a block of Visual Basic statements enclosed by a declaration statement **(Function, Sub, Operator, Get, Set)** and a matching **End** declaration.
- All executable statements in Visual Basic must be within some procedure.
- Visual Basic programs can be broken into smaller logical components called **Procedures.**

The benefits of using procedures in visual basic

- It is easier to debug a program with procedures, breaks a program into discrete logical limits.
- Procedures used in one program can act as building blocks for other program with slight modifications

Types of Procedures

- **Sub Procedures** perform actions but do not return a value to the calling code.
- Event-handling procedures are Sub procedures that execute in response to an event raised by user action or by an occurrence in a program.

Syntax:

```
[Access modifiers] Sub Procedure Name [(parameterList)]
‘Statements of the Sub procedure.
End Sub
```

Access modifiers: Private, public, static

Ex:

```
Private Sub Command1_Click()
    Msgbox “Hello World”
End Sub
```

- **Function Procedures** return a value to the calling code. They can perform other actions before returning.

  Syntax:

  ```
  [Access Modifiers] Function FunctionName [(ParameterList)] As ReturnType
    [Statements]
  End Function
  ```

  Ex:

  ```
  Function Hypotenuse(side1 As Double, side2 As Double) As Double
      Return Math.Sqrt((side1 ^ 2) + (side2 ^ 2))
  End Function
  ```

- **Property Procedures** return and assign values of properties on objects or modules.
- **Operator Procedures** define the behavior of a standard operator when one or both of the operands is a newly-defined class or structure.

Control Structure in Visual Basic

- Control Statements are used to control the flow of program's execution.
- Visual Basic supports control structures such as **if... Then, if...Then ...Else, Select...Case,** and Loop structures such **as Do While...Loop, While...Wend, For...Next** etc method.

If...Then selection structure

- The **If...Then** selection structure performs an indicated action only when the condition is True; otherwise the action is skipped.

Syntax of the If...Then selection

```
If <condition> Then

   True block statements

End If
```

e.g.:

```
If average>75 Then

    txtGrade.Text = "A"

End If
```

If...Then...Else selection structure

- The **If...Then...Else** selection structure allows the programmer to specify that a different action is to be performed when the condition is True than when the condition is False.

Syntax of the If...Then...Else selection

```
If <condition > Then

   True block statements

Else
```

```
  Flase block statements

End If
```

e.g.:

```
If average>50 Then

    txtGrade.Text = "Pass"

Else

   txtGrade.Text = "Fail"

End If
```

<u>Nested If...Then...Else selection structure</u>

- **Nested If...Then...Else** selection structures test for multiple cases by placing If...Then...Else selection structures inside If...Then...Else structures.

Syntax of the Nested If...Then...Else selection structure

You can use Nested If either of the methods as shown above

Method 1

```
If < condition 1 > Then

     True block statements - 1

   ElseIf < condition 2 > Then

True block statements - 2

ElseIf < condition 3 > Then

True block statements - 3

Else
```

```
False block Statement

End If
```

Method 2

```
If < condition 1 > Then

True block statements - 1

Else

If < condition 2 > Then

   True block statements - 2

Else

If < condition 3 > Then

   True block statements - 3

Else

  False block Statement

End If

End If

EndIf
```

e.g.: Assume you have to find the grade using nested if and display in a text box

```
If average > 75 Then

txtGrade.Text = "A"

ElseIf average > 65 Then

txtGrade.Text = "B"
```

```
ElseIf average > 55 Then

txtGrade.text = "C"

ElseIf average > 45 Then

txtGrade.Text = "S"

Else

txtGrade.Text = "F"

End If
```

Select...Case selection structure

- **Select...Case structure** is an alternative to If...Then...ElseIf for selectively executing a single block of statements from among multiple block of statements.
- Select...case is more convenient to use than the If...Else...End If. The following program block illustrate the working of Select...Case.

Syntax of the Select...Case selection structure

```
Select Case Index

Case 0

 Block Statements - 1

Case 1

Block Statements-2

End Select
```

e.g.: Assume you have to find the grade using select...case and display in the text box

```
Dim average as Integer
average = txtAverage.Text
Select Case average
Case 100 To 75
txtGrade.Text ="A"
Case 74 To 65
txtGrade.Text ="B"
Case 64 To 55
txtGrade.Text ="C"
Case 54 To 45
txtGrade.Text ="S"
Case 44 To 0
txtGrade.Text ="F"
Case Else
MsgBox "Invalid average marks"
End Select
```

Loop structures

- Visual Basic loop structures allow you to run one or more lines of code repetitively.
- You can repeat the statements in a loop structure until a condition is **True**, until a condition is **False**, a specified number of times, or once for each element in a collection.

- Do While...Loop
- While...Wend
- For...Next

Do While...Loop

- The **Do While...Loop** is used to execute statements until a certain condition is met. The following Do Loop counts from 1 to 100.

Syntax:

```
Do { While | Until } condition
    [ statements ]
    [ Continue Do ]
    [ statements ]
    [ Exit Do ]
    [ statements ]
Loop
```

Example:

```
Dim number As Integer
number = 1
Do While number <= 100
number = number + 1
Loop
```

- A variable number is initialized to 1 and then the Do While Loop starts.
- 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.

While...Wend

- A While...Wend statement behaves like the Do While...Loop statement. The following While...Wend counts from 1 to 100

Syntax:

```
While condition
    [ statements ]
    [ Continue Whi e ]
    [ statements ]
    [ Exit While ]
    [ statements ]
Wend
```

Example:

```
Dim number As Integer

number = 1

While number <=100

number = number + 1

Wend
```

For...Next

- The For...Next Loop is another way to make loops in Visual Basic. For...Next repetition structure handles all the details of counter-controlled repetition.
- The following loop counts the numbers from 1 to 100:

Syntax:

```
For counter [ As datatype ] = start To end [ Step step ]
    [ statements ]
    [ Continue For ]
    [ statements ]
    [ Exit For ]
    [ statements ]
Next [ counter ]
```

Example:

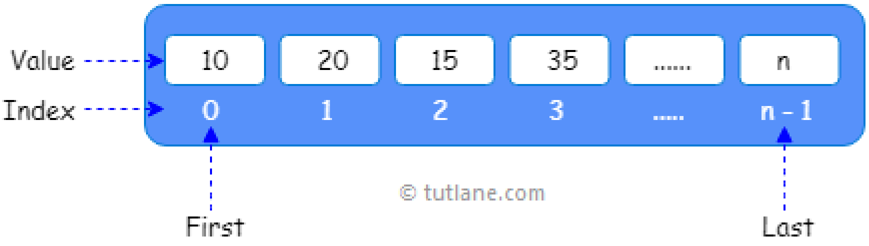
```
Dim x As Integer

For x = 1 To 50

Print x

Next
```

In order to count the numbers from 1 yo 50 in steps of 2, the following loop can be used

```
For x = 1 To 50 Step 2

Print x

Next
```

Arrays in Visual Basic

- **Arrays** are useful to store multiple elements of the **same data type** at contiguous memory locations.
- **Arrays** will allow us to store the fixed number of elements sequentially based on the predefined number of items.
- An array can start storing the values from index 0. Suppose if we have an array with n elements, then it will start storing the elements from index 0 to n-1.



There are two types of array

- Fixed-size array – The size of array always remains the same.
- Dynamic array – The size can be changed Run-time.

Fixed-size array

- Arrays can be declared by specifying the type of elements followed by the brackets () like as shown below.

Syntax is:

```
    Dim array_name As [Data_Type]();
```

```
' Store only int values
Dim numbers As Integer()

' Store only string values
Dim names As String()

' Store only double values

Dim ranges As Double()
```

Example:

```
' Declaring and Initializing an array with size of 5
Dim array As Integer() = New Integer(4) {}
' Defining and assigning an elements at the same time
Dim array2 As Integer() = New Integer(4) {1, 2, 3, 4, 5}
' Initialize with 5 elements will indicates the size of an array
Dim array3 As Integer() = New Integer() {1, 2, 3, 4, 5}
' Another way to initialize an array without size
Dim array4 As Integer() = {1, 2, 3, 4, 5}
' Declare an array without initialization
Dim array5 As Integer()

array5 = New Integer() {1, 2, 3, 4, 5}
```

Multidimensional Array

- **Multidimensional Arrays** can be declared by specifying the data type of an elements followed by the brackets () with comma (,) separator.
- Following are the examples of creating two or three-dimensional arrays in visual basic programming language.

Syntax:

```
' Two Dimensional Array
Dim arr As Integer(,) = New Integer(3, 1) {}

' Three Dimensional Array
Dim arr1 As Integer(,,) = New Integer(3, 1, 2) {}
```

Example:

```
Dim intarr As Integer(,) = New Integer(2, 1) {{4, 5}, {5, 0}, {3, 1}}
' Two Dimensional Integer Array without Dimensions
Dim intarr1 As Integer(,) = New Integer(,) {{4, 5}, {5, 0}, {3, 1}}
' Three Dimensional Array
Dim array3D As Integer(,,) = New Integer(1, 1, 2) {{{1, 2, 3}, {4, 5, 6}},
{{7, 8, 9}, {10, 11, 12}}}
' Three Dimensional Array without Dimensions
Dim array3D1 As Integer(,,) = New Integer(,,) {{{1, 2, 3}, {4, 5, 6}}, {{7,
8, 9}, {10, 11, 12}}}
```

Dynamic array

A dynamic array can be initially declared and can add elements, when we needed instead of declaring the size of the array at design time.

Example:

Dim NewArray()

The actual number of elements can be allocated using a ReDim statement.

Example:

ReDim NewArray(Y+1)

Here, allocates the number of elements in the array based on the value of the variable Y.

Erode Arts and Science College (Autonomous)

Department of Computer Science (SF)

VISUAL PROGRAMMING

UNIT – II

Working with controls

Introduction:

- Visual Basic controls and the ways of creating and implementing.
- It also helps us to understand the concept of Control Arrays. Controls are used to receive user input and display output and has its own set of properties, methods and events.

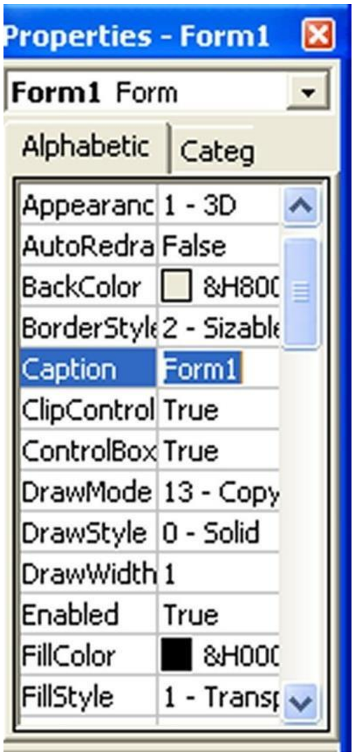
Creating and using controls

- Visual Basic controls are broadly classified as standard controls, **ActiveX controls** and insertable objects.
- **ActiveX controls** exist as separate files with either **.VBX or .OCX** extension.
- **Standard controls** such as CommandButton, Label and Frame controls are contained inside **.EXE** file and are always included in the ToolBox which cannot be removed.
- They include specialized controls such as:
- MSChart control
- The Communications control
- The Animation control
- The ListView control
- An ImageList control
- The Multimedia control
- The Internet Transfer control
- The WinSock control
- The TreeView control
- The SysInfo control

- The Picture Clip control

Tab Index Property of controls

- **Properties window** for a form. In the properties window, the item appears at the top part is the object currently selected.
- At the bottom part, the items listed in the left column represent the names of various properties associated with the selected object while the items listed in the right column represent the states of the properties.
- Properties can be set by highlighting the items in the right column then change them by typing or selecting the options available.
- You can also change the properties at runtime to give special effects such as change of color, shape, animation effect and so on.



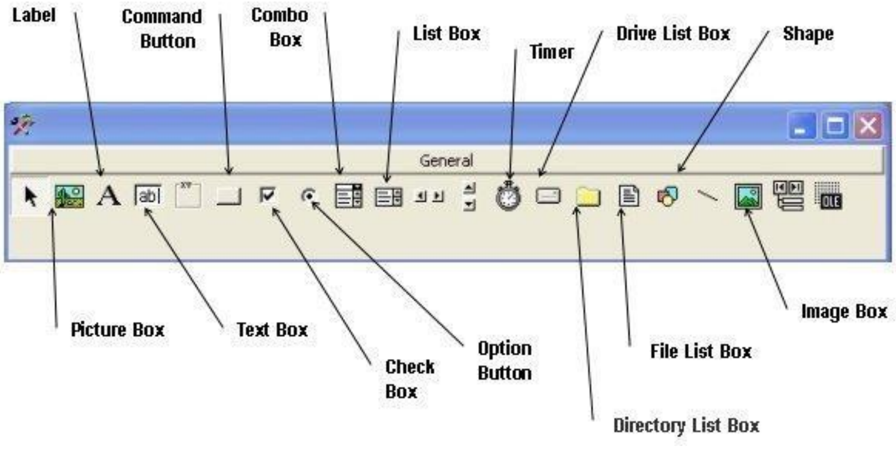
Properties - Form1

Form1 Form

Alphabetic | Categ

Appearanc	1 - 3D
AutoRedra	False
BackColor	□ &H80C
BorderStyl	2 - Sizable
Caption	Form1
ClipControl	True
ControlBox	True
DrawMode	13 - Copy
DrawStyle	0 - Solid
DrawWidth	1
Enabled	True
FillColor	■ &H00C
FillStyle	1 - Transp

Handling some of the common Controls



Textbox Controls

- The **Text Box** is the standard control for **accepting input** from the user as well as to **display the output.**
- It can handle **string (text) and numeric data** but not images or pictures.
- A string entered into a text box can be converted to a numeric data by using the function **Val (text).**

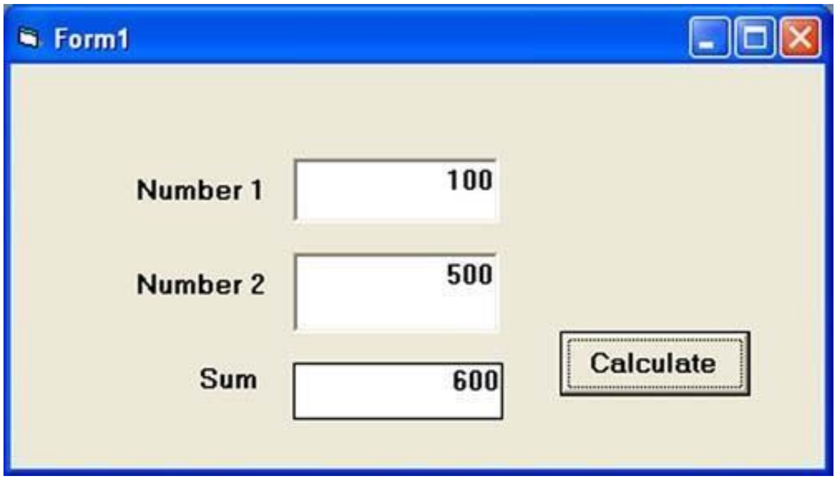
Example:

- In this program, **two text boxes** are inserted into the form together with a few **labels**.
- The two text boxes are used to **accept inputs** from the user and one of the labels will be used to **display** the **sum of two numbers** that are entered into the two text boxes.

Coding:

```
Private Sub Command1_Click()
'To add the values in TextBox1 and TextBox2
  Sum = Val(Text1.Text) +Val(Text2.Text)
'To display the answer on label 1
  Label1.Caption = Sum
End Sub
```

Output:



Using a Label control

- The **label** is a very useful control for Visual Basic, as it is not only used to **provide instructions and guides to the users.**
- it can also be used to display outputs. One of its most **important properties is Caption.** Using the **syntax Label.Caption**, it can display text and numeric data.

Note above Example

Using a Command Button

- The **Command** button is one of the most important controls as it is used to execute commands.
- It displays an illusion that the button is pressed when the user click on it. The most common event associated with the command button is the **Click** event, and the syntax for the procedure is

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

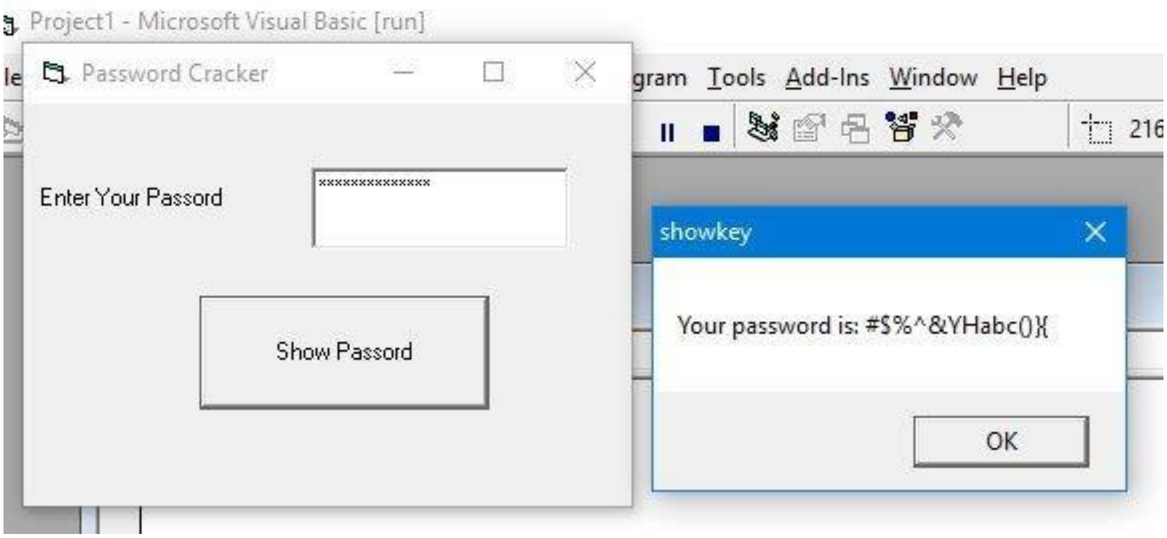
Example:

- In the design phase, insert a **command button** and change its name to **cmd_ShowPass**.
- Next, insert a **TextBox** and rename it as **TxtPassword** and delete Text1 from the Text property.
- Besides that, set its **PasswordChr** to *. Now, enter the following code in the code window.

Coding:

```
Private Sub cmd_ShowPass_Click()
 Dim yourpassword As String
 yourpassword = Txt_Password.Text
 MsgBox ("Your password is: " & yourpassword)
End Sub
```

Output:



Your password is: #$%^&YHabc()}{

Using Option Button Control

- The **OptionButton control** also lets the user selects **one of the choices**.
- Two or more Option buttons must work together because as **one of the option buttons is selected**, the **other Option button will be unselected.**
- **Only one Option Box can be selected at one time**. An option box is selected, its value is set to **"True"** and when it is unselected; its value is set to **"False".**

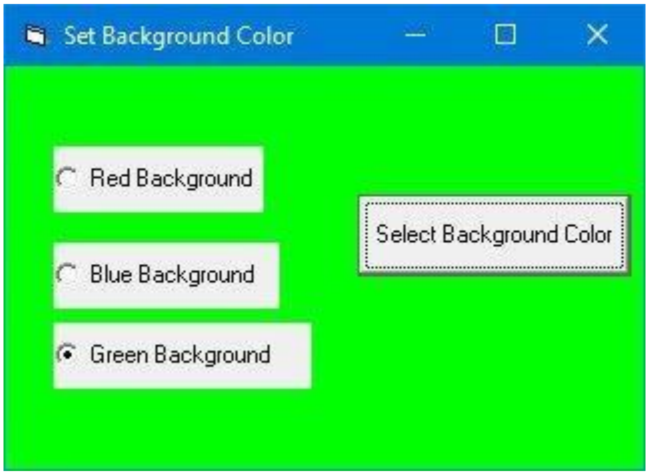
Example:

- You insert **three option buttons** and change their captions to **"Red Background", "Blue Background" and "Green Background"** respectively.
- Next, insert a **command button** and change its name to **cmd_SetColor** and its caption to **"Set Background Color".**
- Now, click on the command button and enter the following code in the code window:

Coding:

```
Private Sub cmd_SetColor_Click()
 If Option1.Value = True Then
 Form1.BackColor = vbRed
 ElseIf Option2.Value = True Then
 Form1.BackColor = vbBlue
 Else
 Form1.BackColor = vbGreen
 End If
End Sub
```

Output:



Using List Box and Combo box Control

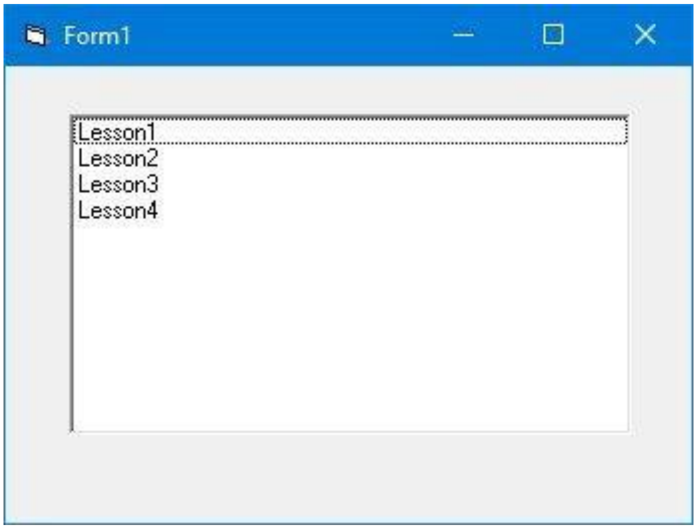
Listbox Control:

- The function of the **ListBox** is to present a list of items the **user can click and select the items from the list.**
- In order to add items to the list, we can use the **AddItem method.**

Example:

```
Private Sub Form_Load ( )
 List1.AddItem “Lesson1”
 List1.AddItem “Lesson2”
 List1.AddItem “Lesson3”
 List1.AddItem “Lesson4”
End Sub
```

Output:



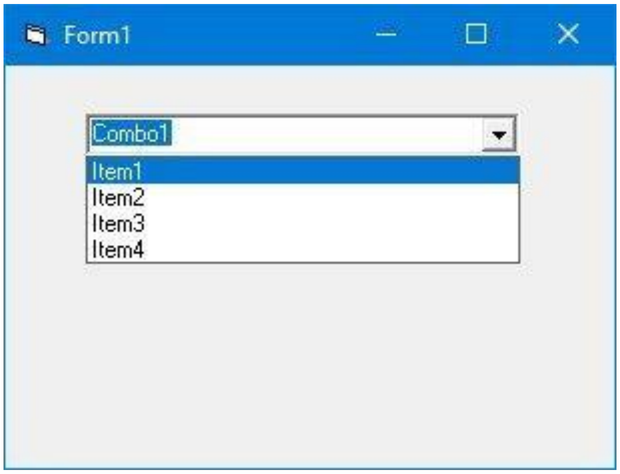
Lesson1
Lesson2
Lesson3
Lesson4

Combo box Control

- The function of the **Combo Box** is also to present a list of items where the user can click and select the items from the list.
- The user needs to click on the small arrowhead on the right of the combo box to see the items which are presented in a **drop-down list**.
- In order to add items to the list, you can also use the **AddItem method.**

Example:

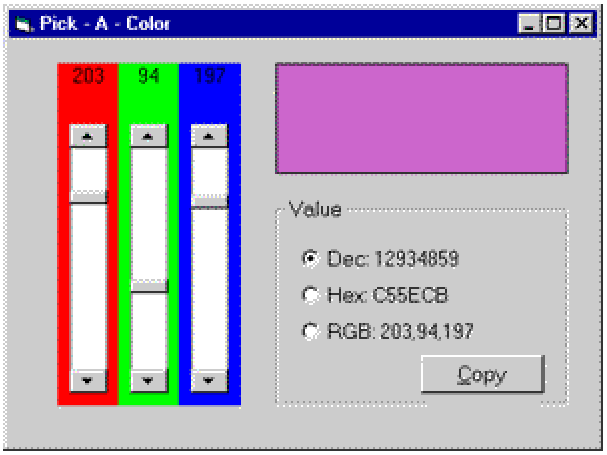
```
Private Sub Form_Load ( )
Combo1.AddItem "Item1"
Combo1.AddItem "Item2"
Combo1.AddItem "Item3"
Combo1.AddItem "Item4"
End Sub
```



Using the Scrollbar Control

- The **ScrollBar** is a commonly used control, it enables the user to select a value by positioning it at the desired location. It represents a set of values.
- The Min and Max property represents the minimum and maximum value. The value property of the ScrollBar represents its current value, that may be any integer between minimum and maximum values assigned.
- The **HScrollBar and the VScrollBar controls** are perfectly identical, apart from their different orientation. After you place an instance of such a control on a form,
- **Few properties**:
  - **Min and Max represent** the valid range of values, **SmallChange** is the variation in value you get when clicking on the scroll bar's arrows, and **LargeChange** is the variation you get when you click on either side of the scroll bar indicator.

Example:



● Dec: 12934859
○ Hex: C55ECB
○ RGB: 203,94,197

More control see this web site:

https://www.vbtutor.net/lesson3.html#google_vignette

Working with control Arrays

- **A Control Array is a group of controls that share the same name type and the same event procedures.** Adding controls with control arrays uses fewer resources than adding multiple control of same type at design time.
- A control array can be created only at design time, and at the very minimum at least one control must belong to it. You create a control array following one of these three methods:
- You create a control and then assign **a numeric, non-negative value to its Index property;** you have thus created a control array with just one element.
- You create two controls of the same class and assign them an identical Name property. Visual Basic shows a dialog box warning you that there's already a control with that name and asks whether you want to create a control array. Click on the Yes button.

Creating a control arrays at design time

Control arrays can be created at run time using the statements

- Load object (Index %)
- Unload object (Index %)

- Object is the name of the control to add or delete from the control array. **Index %** is the value of the index in the array.
- The control array to be added must be an element of the existing array created at design time with an **index value of 0**. When a new element of a control array is loaded, most of the property settings are copied from the lowest existing element in the array.

Following example illustrates the use of the control array.

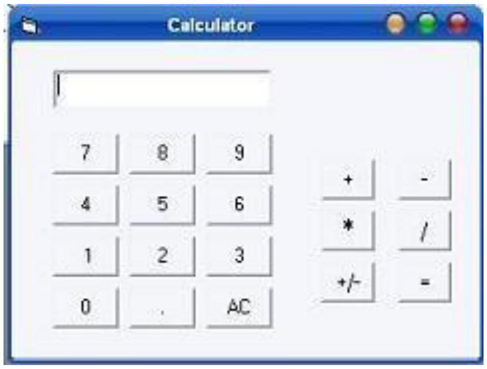
* Open a **Standard EXE project** and save the Form as Calculator.frm and save the Project as **Calculater.vbp.**

* Design the form as shown below.

Object	Property	Setting
Form	Caption	Calculator
	Name	frmCalculator
CommandButton	Caption	1
	Name	cmd
	Index	0
CommandButton	Caption	2
	Name	cmd
	Index	1
CommandButton	Caption	3
	Name	cmd
	Index	2

CommandButton	Caption	4
	Name	cmd
	Index	3
CommandButton	Caption	5
	Name	cmd
	Index	4
CommandButton	Caption	6
	Name	cmd
	Index	5
CommandButton	Caption	7
	Name	cmd
	Index	6
CommandButton	Caption	8
	Name	cmd
	Index	7
CommandButton	Caption	9
	Name	cmd
	Index	8
CommandButton	Caption	0
	Name	cmd
	Index	10
CommandButton	Caption	.
	Name	cmd
	Index	11
CommandButton	Caption	AC
	Name	cmdAC
CommandButton	Caption	+
	Name	cmdPlus

CommandButton	Caption Name	- cmdMinus
CommandButton	Caption Name	* cmdMultiply
CommandButton	Caption Name	/ cmdDivide
CommandButton	Caption Name	+/- cmdNeg
TextBox	Name Text	txtDisplay (empty)
CommandButton	Caption Name	= cmdEqual



The following variables are declared inside the general declaration

```
Dim Current As Double
Dim Previous As Double
Dim Choice As String
Dim Result As Double
```

The following code is entered in the cmd_Click( ) (Control Array) event procedure

```
Private Sub cmd_Click(Index As Integer)
txtDisplay.Text = txtDisplay.Text & cmd(Index).Caption
'&is the concatenation operator
Current = Val(txtDisplay.Text)
End Sub
```

The following code is entered in the cmdAC_Click ( ) event procedure

```
Private Sub cmdAC_Click()
Current = Previous = 0
txtDisplay.Text = ""
End Sub
```

The below code is entered in the cmdNeg_Click( ) procedure

```
Private Sub cmdNeg_Click()
Current = -Current
txtDisplay.Text = Current
End Sub
```

The following code is entered in the click events of the cmdPlus, cmdMinus, cmdMultiply, cmdDevide controls respectively.

```
Private Sub cmdDevide_Click()
txtDisplay.Text = ""
Previous = Current
Current = 0
Choice = "/"
End Sub

Private Sub cmdMinus_Click()
txtDisplay.Text = ""
Previous = Current
Current = 0
Choice = "-"
End Sub

Private Sub cmdMultiply_Click()
txtDisplay.Text = ""
Previous = Current
Current = 0
Choice = "*"
End Sub

Private Sub cmdPlus_Click()
txtDisplay.Text = ""
Previous = Current
Current = 0
Choice = "+"
End Sub
```

To print the result on the text box, the following code is entered in the cmdEqual_Click ( ) event procedure.

```
Private Sub cmdEqual_Click()

Select Case Choice

Case "+"
Result = Previous + Current
txtDisplay.Text = Result
Case "-"
```

```
Result = Previous - Current
txtDisplay.Text = Result
Case "*"
Result = Previous * Current
txtDisplay.Text = Result
Case "/"
Result = Previous / Current
txtDisplay.Text = Result
End Select

Current = Result

End Sub
```

Save and run the project. On clicking digits of user's choice and an operator button, the output appears.

Adding a control array at runtime

You must follow **several rules to be able to create and remove controls with control arrays**.

- First you must have a control of the type that you will be adding placed on the desired form at runtime.
- If you will be adding TextBox controls to a form,
- **For example**, you must have at least one text box drawn on that form at design time. That control can be invisible, and there are no restrictions for the size or placement of that control, but it must be on the form at design time.
- The second requirement for using dynamic control arrays is that the template object that you draw at design time must be part of a control array.
- Usually it is the only control of its type with its Index property set to a value of 0. Continuing with the text box example, you can have a form with only one text box as a template, and that text box must have its Index property set to some integer value (typically 0 or 1). If your application required it, you might have additional text boxes **with Index values of 1, 2, 3, and so on.**

Control Array applications

- Arrays of variables, you can group a set of controls together as an array. The following facts apply to control arrays:

- The set of controls that form a control array must be all of the same **type (all textboxes, all labels, all option buttons, etc.)**

- You set up a **control array by naming one or more controls of the same type the same name** and **set the Index property of each control in the array to a non-negative value** (i.e., the controls in the control array are usually indexed from 0 to one less than the number of controls in the array).

To refer to a member of a control array, **the syntax is**:

ControlName(Index)[.Property]

For example, to refer to the Text property of the first element of an array of textboxes called **txtField**, you would use:

txtField(0).Text

- All the members of the control array share the same event procedure.
- **For example,** if you have a **control array of 10 textboxes call txtField, indexed 0 to 9, you will not have 10 different GotFocus events – you will just have one that is shared amongst the 10 members.**
- To differentiate which member of the control array is being acted upon, VB will automatically pass an Index parameter to the event procedure. For example, the GotFocus event procedure for the txtField control array might look like this:

```
Private Sub txtField_GotFocus(Index As Integer)

   txtField(Index).SelStart = 0

   txtField(Index).SelLength = Len(txtField(Index).Text)

End Sub
```

Erode Arts and Science College (Autonomous)

Department of Computer Science (SF)

VISUAL PROGRAMMING

Staff Name: M.Nagaraj, M.C.A., M.Phil., **Course : III B.Sc (CS)**

Semester : V

UNIT- III

Menu, Mouse event and dialog boxes

Menu Editor – Introduction:

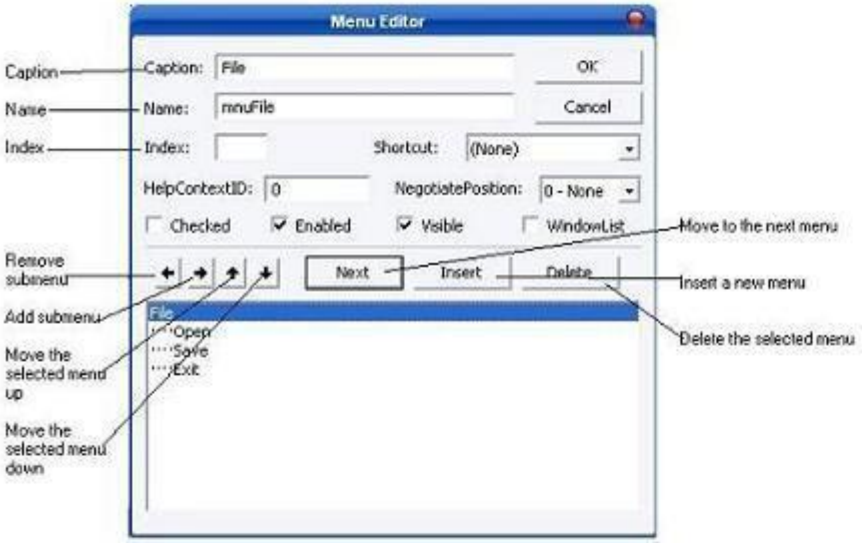
- Visual Basic provides an easy way to create menus with the modal **Menu Editor dialog.**
- The **Menu Editor command** is grayed unless the form is visible. And also you can display the Menu Editor window by right clicking on the Form and selecting Menu Editor.
- Each menu item has a Caption property (possibly with an embedded & character to create an access key) and a Name. Each item also exposes **three Boolean properties**, Enabled, Visible, and Checked, **you can set both at design time and at run time.**

Menu Interface

- Menus are located on the menu bar of a form, contain a list of related commands. You click a menu title in a Windows-based program, a list of menu commands should always appear in a well-organized list.
- Most menu commands run immediately after they are clicked.
- **For example,** the user clicks the Edit menu Copy command, Windows immediately copies information to the Clipboard. However, if ellipsis points (…) follow the menu command, Visual Basic displays a dialog box that requests more information before the command is carried out.

This section includes the following topics:

- Using the Menu Editor
- Adding Access and Shortcut Keys
- Processing Menu Choices



Menu Editor

Caption: File

Name: mnuFile

Index:

Shortcut: (None)

HelpContextID: 0

NegotiatePosition: 0 - None

□ Checked ☑ Enabled ☑ Visible □ WindowList

File
····Open
····Save
····Exit

Using The Menu Editor

The Menu Editor is a Visual Basic dialog box that manages menus in your programs. With the Menu Editor, you can:

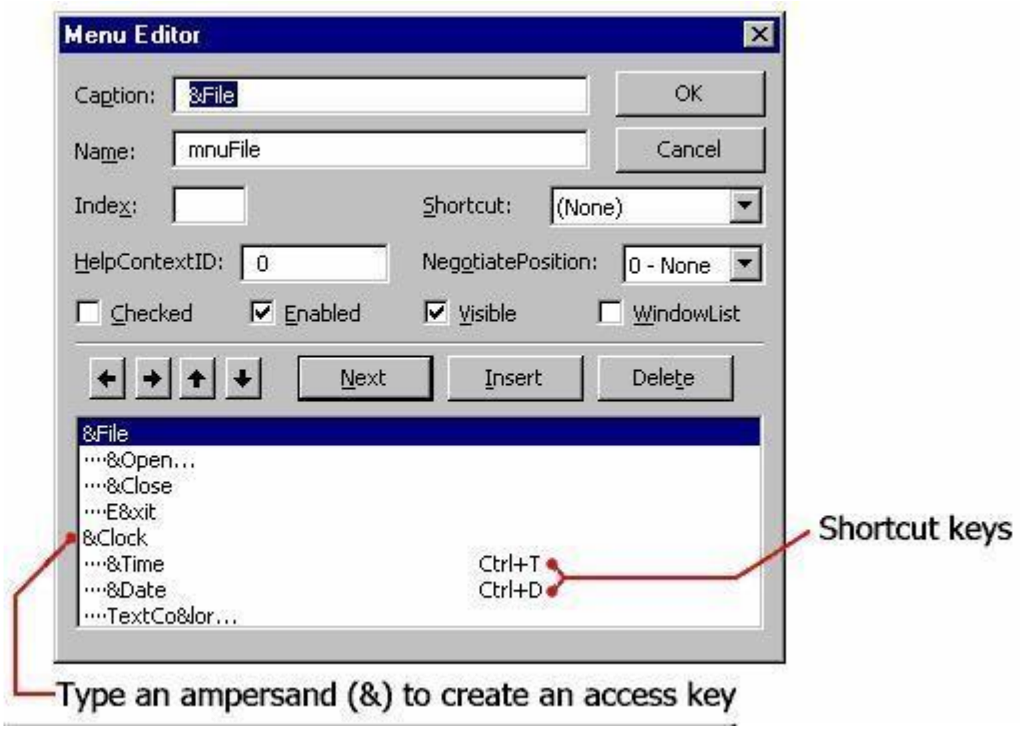
- Add new menus
- Modify and reorder existing menus
- Delete old menus
- Add special effects to your menus, such as access keys, check marks, and keyboard shortcuts.

The Menu Editor Window In the Menu Editor Window you can specify the structure of your menu by adding one command at a time. Each menu command has two compulsory properties.

• Caption : This is the name that user sees on the application's menu bar

• Name : This is the name of the menu command. This property doesn't

appear on the screen, but this name is used program the menu command.



Menu Editor

Caption: &File

Name: mnuFile

Index:

Shortcut: (None)

HelpContextID: 0

NegotiatePosition: 0 - None

□ Checked ☑ Enabled ☑ Visible □ WindowList

```
&File
····&Open...
····&Close
····E&xit
&Clock
····&Time                Ctrl+T
····&Date                Ctrl+D
····TextCo&lor...
```

Shortcut keys

Type an ampersand (&) to create an access key

Pop-Up Menus

- **Popup menus** are a useful metaphor. Popup menus are sometimes referred to as **speed menus, right-click menus, or context menus**.
- Popup menus are generally **invoked by right-clicking the mouse button.** Based on the hover location of the mouse, most applications will display a specific menu.
- The users-focus, or context, determines which menu is shown hence the name context menu.

- The code below displays a pop-up menu at the cursor location when the user clicks the right mouse button over a form.

- To try this **example**, create a form with a menu named “**mnuFile**” (“mnuFile” must have at least one submenu). Copy the code into the Declarations section of the form then run the application.

```
  Private Sub Form_MouseDown (Button As Integer, Shift As Integer, X As
Single, Y As Single)

    If Button = 2 Then PopupMenu mnuFile

  End Sub
```

Mouse Events:

- Visual Basic responds to various mouse events are recognized by most of the controls.
- **The main events are MouseDown, MouseUp and MouseMove.**
- **MouseDown** occurs when the user presses any mouse button and **MouseUp** occurs when the user releases any mouse button.
- These events use the arguments button, Shift, X, Y and they contain information about the mouse's condition when the button is clicked.

Positioning a control

- **MouseDown** is the commonly used event and it is combined with the move method to move an Image control to different locations in a Form.
- The following application illustrates the movement of objects responding to move events. it makes use of **two OptionButton Controls**, two image controls and a **CommandButton.** The application is designed in such a way that when an OptionButton is selected, the corresponding image control is placed anywhere in the form whenever it is clicked.

  - The **first argument** is an integer called **Button**. The value of the argument indicates whether the **left, right or middle mouse button was clicked.**
  - The **second argument** in an integer called **shift**. The value of this argument indicates whether the mouse button was clicked simultaneously with the Shift key, Ctrl key or Alt key.

- The **third and fourth arguments X and Y** are the coordinates of the mouse location at the time the mouse button was clicked. As the Form_MouseDown( ) is executed automatically whenever the mouse button is clicked inside the Form's area the X, Y co-ordinates are referenced to the form.

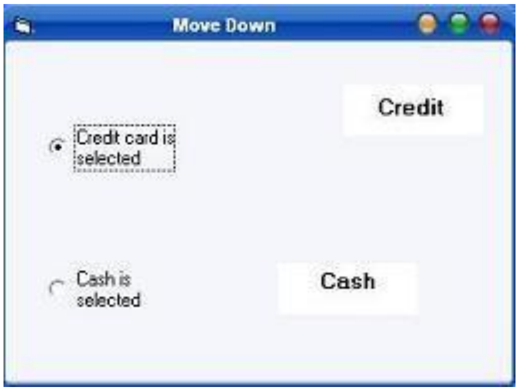
Object	Property	Setting
Form	Caption Name	MouseDown frmMouseDown
OptionButton	Caption Name Value	Credit card is selected optCredit True
OptionButton	Caption Name	Cash is selected optCash
Image	Name Picture	imgCredit c:/credit.jpg
Image	Name Picture	imgCash c:/cash.jpg

The follwoing code is entered in the general declarations section of the Form.

```
Option Explicit
```

The following code is entered in the Form_MouseDown( ) event

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single,
Y As Single)
If optCredit = True Then
     imgCredit.Move X, Y
  Else
     imgCash.Move X, Y
End If
End Sub
```



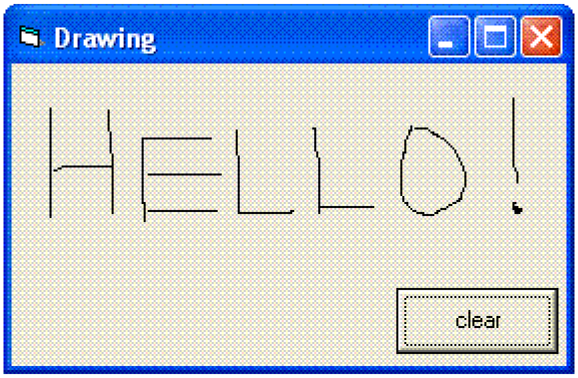
Graphical Mouse Application

- The mouse events can be combined with graphics methods and any number of customized drawing or paint applications can be created.
- The following application combines **MouseMove and MouseDown events**, and illustrates a drawing program.

Open a new **Standard EXE project** and save the Form as Draw.frm and save the Project as Draw.vbp. Name the caption of the as Drawing. Add command button control and name the caption of it as Clear

Enter the following code in the **Form_MouseDown ( ) procedure, Form_MouseMove ( ) procedure and cmdClear_Click ( ) procedures** respectively.

```
Private Sub cmdClear_Click()
frmDraw.Cls
End Sub

Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single,
Y As Single)
frmDraw.CurrentX = X
frmDraw.CurrentY = Y
End Sub

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single,
Y As Single)
If Button = 1 Then
Line (frmDraw.CurrentX, frmDraw.CurrentY)-(X, Y)
End If
End Sub
```



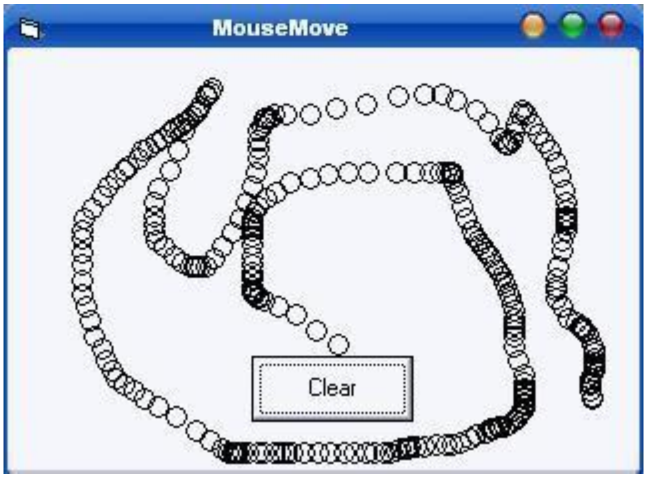
MouseMove application

- Visual Basic does not generate a **MouseMove event** for every pixel the mouse moves over and a limited number of mouse messages are generated per second by the operating environment.
- The following application illustrates how often the Form_MouseMove ( ) event is executed.

Open a new standard EXE project and save the form as MouseMove.frm and save the Project as MouseMOve.vbp. Place a `CommandButton` control and name the caption as Clear and set the name as `cmdClear`.

The following code is entered in the `cmdClear_Click ( )` and `Form_MouseMove ( )` events respectively.

```
Private Sub cmdClear_Click()
frmMouseMove.Cls
End Sub

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single,
Y As Single)
Circle (X, Y), 70
End Sub
```



Dragging and dropping

- The drag-and-drop features in Visual Basic allow you to incorporate this functionality in the programs you develop.
- The action of holding a mouse button down and moving a control is called **dragging**, and the action of releasing the button is called **dropping**.

- Visual Basic supports two drag-and-drop modes, automatic or manual. In automatic mode, you just have to set a property at design time or at run time and let Visual Basic do everything.

- Conversely, in manual mode you have to respond to a number of events that occur while dragging is in progress, but in return you get better control over the process.

Properties

- The two properties involved are **DragMode**, which specifies whether Automatic or Manual dragging will be used, and **DragIcon**, which specifies what icon is displayed when the control is dragged.

Events

- The two events involved are **DragDrop**, which occurs when a control is dropped onto the object, and **DragOver**, which occurs when a control is dragged over the object.

Method

- The Drag method starts or stops manual dragging.
- All controls except menus, timers, lines, and shapes support the **DragMode** and **DragIcon** properties and the Drag method.
- Forms recognize the **DragDrop** and **DragOver** events, but they don't support the **DragMode** and **DragIcon** properties or the Drag method.

Dialog boxes: Model and modeless dialog boxes

You can use class Common Dialog to manage two kinds of dialog boxes:

- **Modal dialog boxes** which require the user to respond before continuing the program
- **Modeless dialog boxes** which stay on the screen and are available for use at any time but permit other user activities

The resource editing and procedures for creating a dialog template are the same for modal and modeless dialog boxes.

Creating a dialog box for your program requires the following steps:

- Use the dialog editor to design the dialog box and create its dialog-template resource.
- Create a dialog class.
- Connect the dialog resource's controls to message handlers in the dialog class.
- Add data members associated with the dialog box's controls and to specify dialog data exchange and dialog data validations for the controls.

Predefined dialog boxes

Program On Predefined Dialogue Box in VB

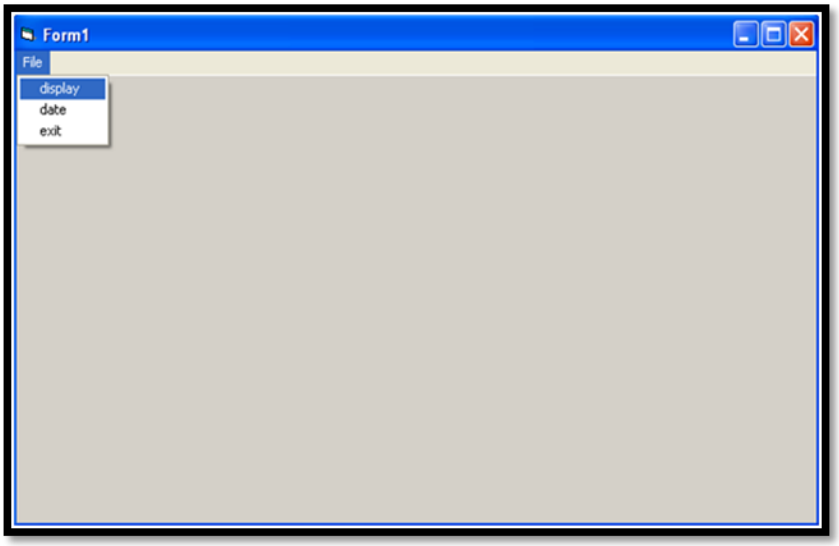
This is a VB 6.0 Application program for predefined Dialog box.

Program On Predefined Dialogue Box

In properties window change the control caption as follows:

Caption	Name
File	Mnufile
Display	Mnudisplay
Date	Mnudate
Exit	Mnuexit

DESIGN VIEW:

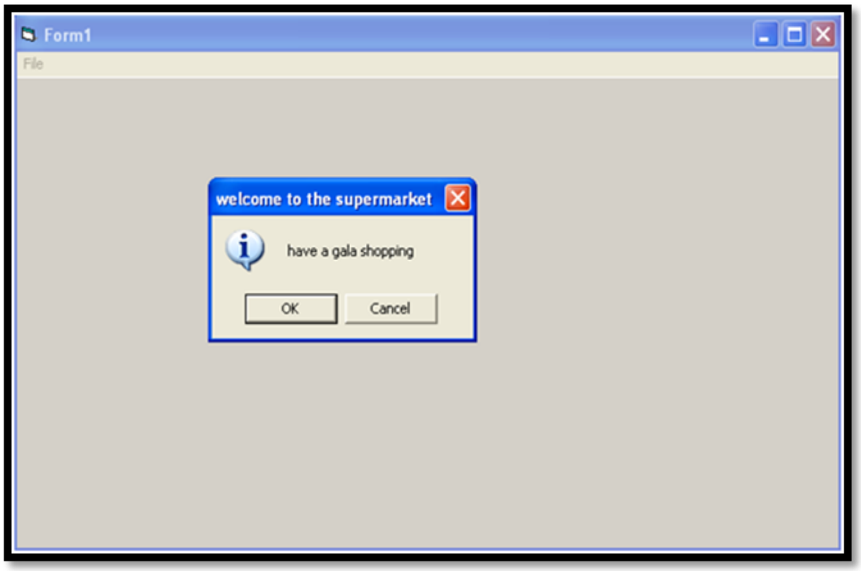


CODING:

Private Sub mnudate_Click()

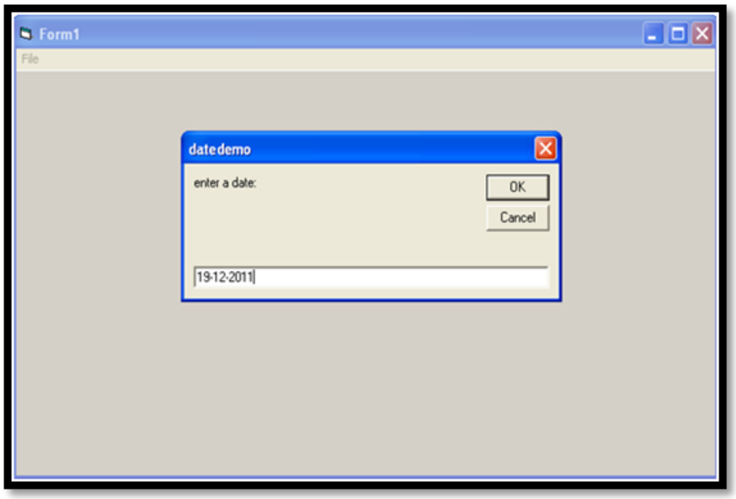
```
        Dim i, day, msg

        i = InputBox("enter a date:", "datedemo")

        If Not IsDate(i) Then

        MsgBox "invalid date"

        Exit Sub

        End If
        day = Format(i, "ddd")

        msg = "day of this date:" + day

        MsgBox msg

End Sub

Private Sub mnudisplay_Click()

        Dim message As String

        Dim dialogtype As Integer

        Dim title As String

        message = "have a gala shopping"

        dialogtype = vbOK + vbInformation

        title = "welcome to the supermarket"

        MsgBox message, dialogtype, title

End Sub

Private Sub mnuexit_Click()

        Dim message As String

        Dim title As String

        Dim dialogtype As Integer

        Dim response As Integer

        message = "thank you,visit again"

        dialogtype = vbYesNo + vbInformation

        title = "goodbye"
        reponse = MsgBox(message, dialogtype, title)

        If response = vbYes Then

        End If

End Sub
```

OUTPUT 1:



have a gala shopping

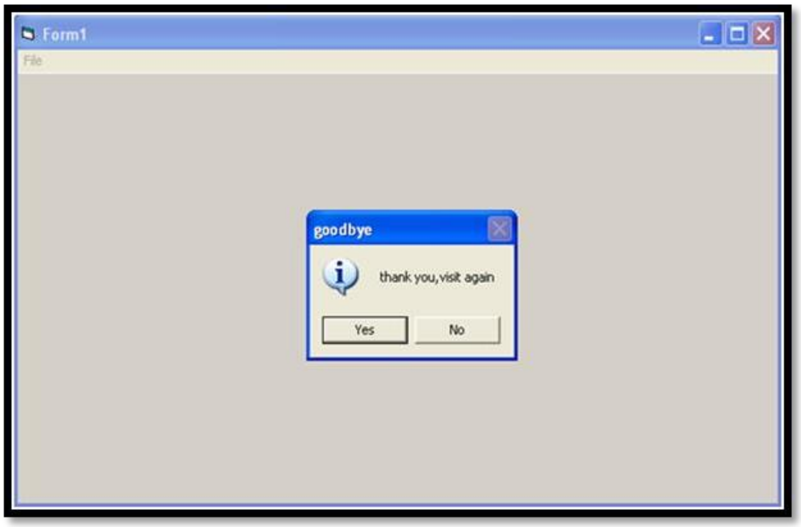
OUTPUT 2:



enter a date:

19-12-2011

OUTPUT 3:



goodbye

thank you,visit again

Erode Arts and Science College (Autonomous)

Department of Computer Science (SF)

VISUAL PROGRAMMING

Staff Name: M.Nagaraj, M.C.A., M.Phil., **Course : III B.Sc (CS)**

Semester : V

UNIT- IV

Graphics, MDI and flex Grid

Introduction

- The graphics method in Visual Basic allows us to control each do known as a pixel or picture elements. Graphics is an interesting aspects of Visual Basic that uses various methods and controls.

Graphics for Application

- VB6 gives you the flexibility and power to make graphical applications in easy steps. It is rich in graphics related features.
- The built-in methods, properties and events allow you to use these features most effectively.
- There are 2 approaches of creating graphics for an applicatios
  - Graphics Controls
  - Graphics Methods

Fundamentals of graphics

- You will learn about various graphic methods, properties and techniques. The graphic methods and properties let you perform some graphical operations.
- More specifically, they allow you to draw **points, lines, circles, rectangles, ellipses and other shapes.** You will also learn how to display text and images. Finally, this tutorial introduces you to the Paint event.

- A colour is represented by a long integer and there are four ways of specifying it t run-time. They are specified using RGB function, QBColor function, and using one of the intrinsic constants listed in the Object Browser and by entering a color value directly.
- The RGB( ) function enables to specify colours. RGB stands for Red, Green, Blue combining one or more of these three primary colours can produce any visible colour.
- **The maximum value of each argument is 255 and minimum is 0.**
- Example:
    - RGB (255,0,0)
    - RGB (255,255,0)

```
Form1.BackColor = RGB (120, 87, 55)
```

Using Graphical Control

Line Control:

The Line method draws a line. Using the Line method, you can also draw other geometric shapes such as rectangle, triangle etc.

Shape Control

- The Shape control is an extension of the **Line control**.
- It can display six basic shapes: **Rectangle, Square, Oval, Circle, Rounded Rectangle, and Rounded Square.**
- It supports all the Line control's properties and a few more: BorderStyle (0-Transparent, 1-Solid), FillColor, and FillStyle (the same as a form's properties with the same names). The same performance considerations I pointed out for the Line control apply to the Shape control.

Using Graphics Methods

- The graphic methods allow you to draw on the form and the PictureBox control. In Visual Basic 6, graphic methods are only supported by the form object and the PictureBox control.

The common graphic methods are explained below.

Print:

Print is the simplest graphic method in Visual Basic 6. **This method has been used throughout the earlier versions of the language.** It prints some text on the form or on the PictureBox control. It displays texts.

Cls:

The **Cls method is another simple graphic method that is used to clear the surface of the form or the PictureBox control.** If some texts are present, you can use the Cls method to remove the texts. It clears any drawing created by the graphic methods.

Point:

The Point method returns the color value from an image for a pixel at a particular point. This method is generally used to retrieve color values from bitmaps.

Refresh:

The refresh method redraws a control or object. In other words, it refreshes the control. Generally, controls are refreshed automatically most of the times.

PSet:

The PSet method sets the color of a single pixel on the form. This method is used to draw points.

Line:

The Line method draws a line. Using the Line method, you can also draw other geometric shapes such as rectangle, triangle etc.

Circle:

The Circle method draws a circle. Using the Circle method, you can also draw other geometric shapes such ellipses, arcs etc.

PaintPicture:

The PaintPicture method displays an image on the form at run-time.

TextHeight:

The TextHeight method returns the height of a string on the form at run-time.

TextWidth:

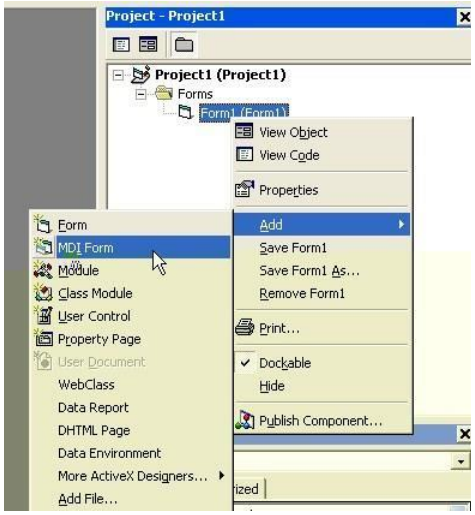
The TextWidth method returns the width of a string on the form at run-time.

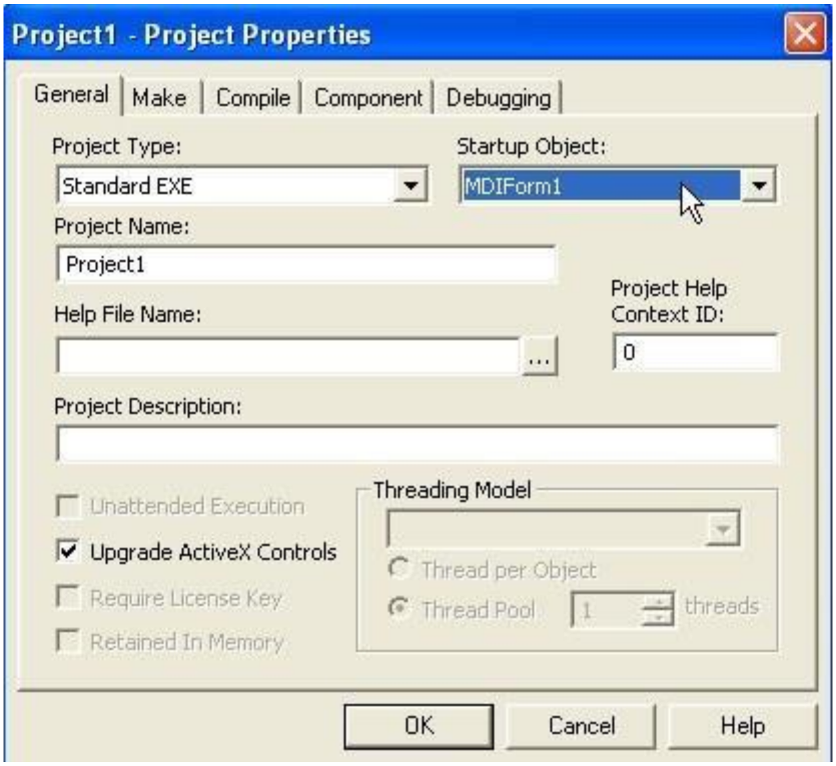
Multiple document Interface (MDI)

- The **Multiple Document Interface (MDI)** was designed to simplify the exchange of information among documents, all under the same roof.
- With the main application, you can maintain multiple open windows, but not multiple copies of the application. Data exchange is easier when you can view and compare many documents simultaneously.
- Each document is displayed in its own window, and all document windows have the same behavior. **The main Form, or MDI Form, isn't duplicated**, **but it acts as a container for all the windows, and it is called the parent window**. The windows in which the individual documents are displayed are called **Child windows.**

Creating an MDI Application

- To create an MDI application, follow these steps:
- Start a **new project** and then choose Project >>> **Add MDI Form** to add the parent Form.
- Set the Form's caption to MDI Window
- Choose Project >>> **Add Form to add a SDI Form.**
- Make this Form as child of MDI Form by setting the MDI Child property of the SDI Form to True. Set the caption property to MDI Child window.





Project Type: Standard EXE

Startup Object: MDIForm1

Project Name: Project1

Help File Name:

Project Help Context ID: 0

Project Description:

□ Unattended Execution

☑ Upgrade ActiveX Controls

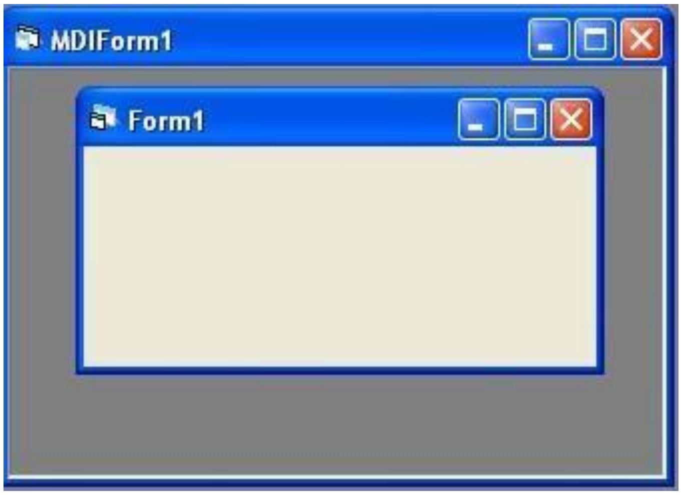
□ Require License Key

□ Retained In Memory

Threading Model

○ Thread per Object

● Thread Pool 1 threads



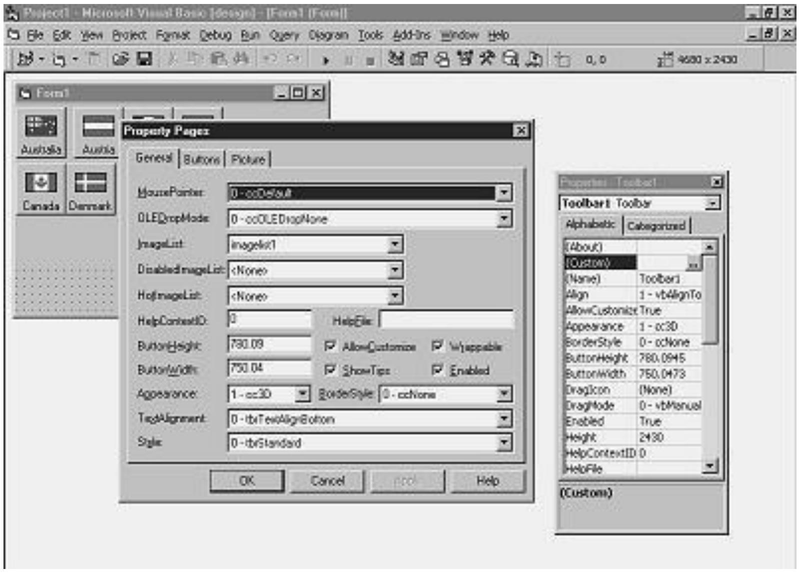
Adjusting the Text box

Creating a tool Bar

- A **toolbar** is becoming a standard feature of Windows applications. Toolbars provide functionality to a user through an **easily accessible, graphical interface**.
- For common functions the user does not need to navigate through a menu or remember shortcut keys to use an application. Applications can expose their most common features through buttons on a toolbar.

Setting Custom Properties

- The ToolBar control is available through the **Windows Common Controls**, along with the **TreeView, ListView, and ImageList**.
- After you have drawn a toolbar on a form, you will usually start by setting properties through the Property Pages dialog box for the control.
- Like the **TreeView** and the **ListView** controls, the toolbar gets images from an **ImageList** control. If you are building a toolbar that will have graphics on its buttons.
- You will first need to add an ImageList control to the form and then bind that ImageList to the toolbar through the Property Pages dialog box.



MousePointer: 0 - ccDefault
OLEDropMode: 0 - ccOLEDropNone
ImageList: imagelist1
DisabledImageList: <None>
HotImageList: <None>
HelpContextID: 0
HelpFile:
ButtonHeight: 780.09
ButtonWidth: 750.04
☑ AllowCustomize
☑ Wrappable
☑ ShowTips
☑ Enabled
Appearance: 1 - cc3D
BorderStyle: 0 - ccNone
TextAlignment: 0 - tbrTextAlignBottom
Style: 0 - tbrStandard

Toolbar1 Toolbar

(About)	
(Custom)	
(Name)	Toolbar1
Align	1 - vbAlignTo
AllowCustomize	True
Appearance	1 - cc3D
BorderStyle	0 - ccNone
ButtonHeight	780.0945
ButtonWidth	750.0473
DragIcon	(None)
DragMode	0 - vbManual
Enabled	True
Height	2430
HelpContextID	0
HelpFile	

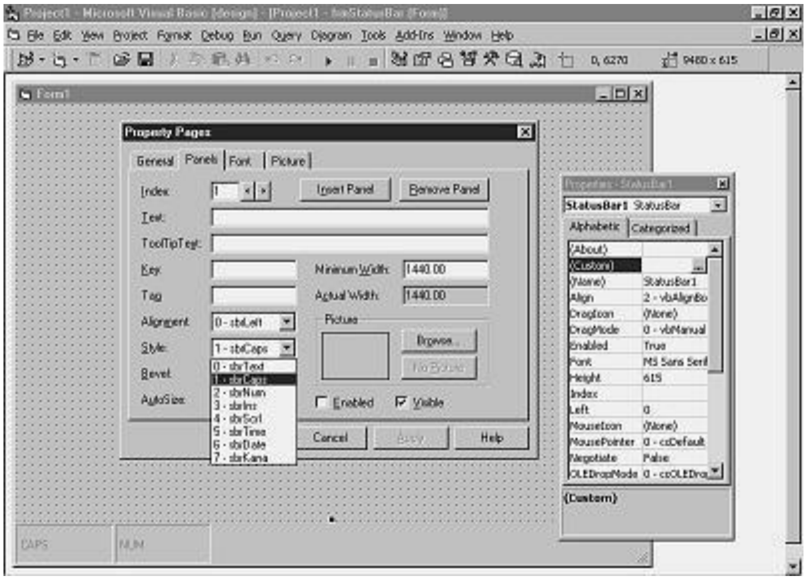
(Custom)

Displaying a Status bar

- The **StatusBar** control is another **ActiveX Control** available through the Windows Common Controls components. With the StatusBar.
- You can easily **display information about the date, time, and other details** about the application and the environment to the user.

Panel Object and Panels Collection

- A StatusBar is made up of Panel objects, each of which displays different types of information. All the Panel objects are contained in the Panels Collection of the StatusBar.
- The appearance and purpose of each Panel is determined by the Style property of that Panel.
- The following styles are available:
  - **sbrText (0)**: The Panel will display text information. The information displayed is in the Text property of the Panel.
  - **sbrCaps (1)**: The Panel will contain the string "Caps". It will be highlighted if Caps Lock is turned on or disabled if Caps Lock is turned off.
  - **sbrNum (2)**: The Panel will contain the string "Num". It will be highlighted if Num Lock is turned on or disabled if Num Lock is turned off.
  - **sbrIns (3):** The Panel will contain the string "Ins". It will be highlighted if Insert mode is turned on or disabled if Insert mode is turned off.
  - **sbrScrl (4)**: The Panel will contain the string "Scrl". It will be highlighted if Scroll Lock is turned on or disabled if Scroll Lock is turned off.
  - **sbrTime (5)**: The Panel will display the current time in the system format.
  - **sbrDate (6)**: The Panel will display the current date in the system format.
  - **sbrKana (7)**: The Panel will contain the string "Kana". It will be highlighted if Kana (a special character set only available in the Japanese PC market) is enabled or disabled if Kana is disabled.



Index: 1

Text:

ToolTipText:

Key:

Tag:

Minimum Width: 1440.00

Actual Width: 1440.00

Alignment: 0 - sbrLeft

Style: 1 - sbrCaps

Bevel:

AutoSize:

0 - sbrText
1 - sbrCaps
2 - sbrNum
3 - sbrIns
4 - sbrScrl
5 - sbrTime
6 - sbrDate
7 - sbrKana

Picture

□ Enabled ☑ Visible

StatusBar1 StatusBar

(About)	
(Custom)	...
(Name)	StatusBar1
Align	2 - vbAlignBo
DragIcon	(None)
DragMode	0 - vbManual
Enabled	True
Font	MS Sans Serif
Height	615
Index	
Left	0
MouseIcon	(None)
MousePointer	0 - ccDefault
Negotiate	False
OLEDropMode	0 - ccOLEDro

(Custom)

The Flex Grid Control

You can create utilities to display, filter, edit, validate and update your data. For example, such utilities could include:

- data entry & validation
- high level reports
- ported spreadsheet macro applications retaining cell layout & format

❖ VB6, are several grid-orientated controls aimed at managing rows and columns of data, however one of the most versatile, hence its name, is the FlexGrid.

❖ Most of the other grid objects are specifically designed for data binding, whereas the FlexGrid has many collections of properties, methods and events that lend themselves to several environments in addition to just data-binding.

❖ One of the most powerful applications of the FlexGrid is in managing data from database tables in an unbound state. That is, to populate & edit the FlexGrid and write the contents back to the database in code.

❖ This may seem a pointless exercise as objects such Microsoft's Data Bound Grid Control are written specifically for this purpose, except these grids simply display the contents of a recordset, record by record, field by field. Whereas the FlexGid can populate the grid in any manner,

Examples of which are:

✓ separating grouped **data with blank rows/columns**,

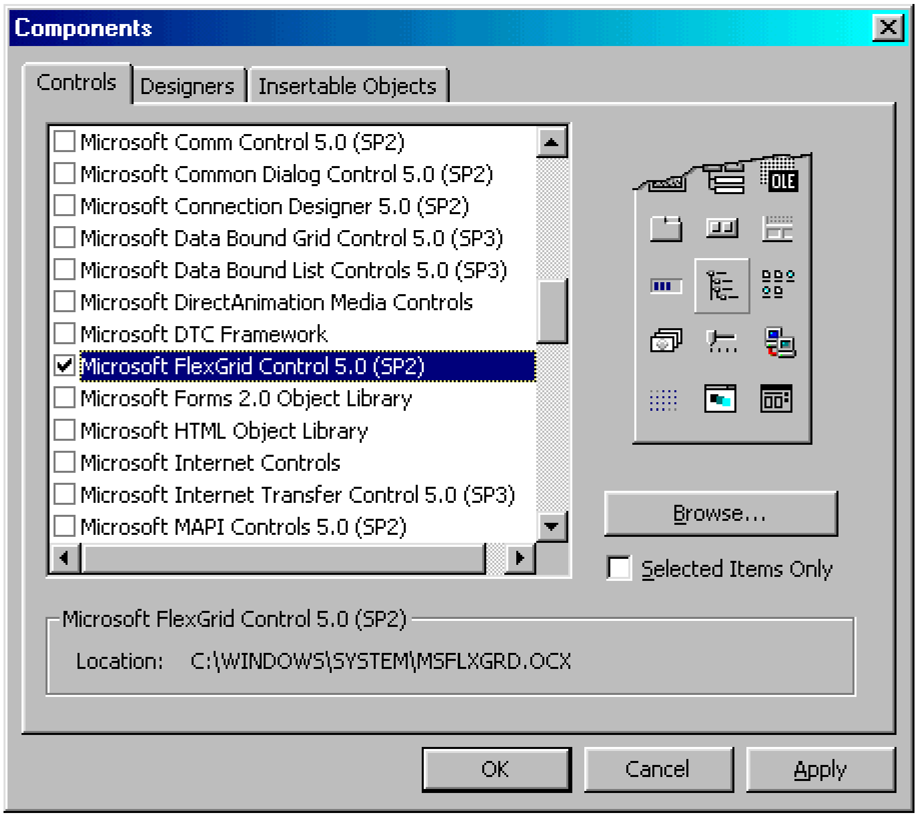
✓ adding **sub/grand-total rows/columns**,

✓ changing the colour of the cell background or text **in individual or multiple cells,**

✓ reacting to the state of the data, e.g. **highlighting negative values**,

✓ validating entered data e.g., numeric values, positive values, permitted date ranges etc.

✓ These are just a few of the possibilities available with FlexGrids, we'll take a look at more later on.



□ Microsoft Comm Control 5.0 (SP2)
□ Microsoft Common Dialog Control 5.0 (SP2)
□ Microsoft Connection Designer 5.0 (SP2)
□ Microsoft Data Bound Grid Control 5.0 (SP3)
□ Microsoft Data Bound List Controls 5.0 (SP3)
□ Microsoft DirectAnimation Media Controls
□ Microsoft DTC Framework
☑ Microsoft FlexGrid Control 5.0 (SP2)
□ Microsoft Forms 2.0 Object Library
□ Microsoft HTML Object Library
□ Microsoft Internet Controls
□ Microsoft Internet Transfer Control 5.0 (SP3)
□ Microsoft MAPI Controls 5.0 (SP2)

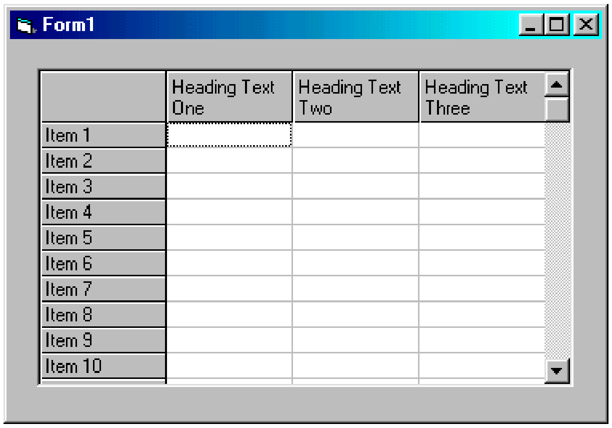
□ Selected Items Only

Microsoft FlexGrid Control 5.0 (SP2)

Location: C:\WINDOWS\SYSTEM\MSFLXGRD.OCX

An example of grid initialisation code would be as follows

```
Dim lngWidth As Long
Dim intLoopCount As Integer
Const SCROLL_BAR_WIDTH = 320

With MSFlexGrid

    lngWidth = .Width - SCROLL_BAR_WIDTH
    .Cols = 4
    .FixedCols = 1
    .Rows = 0
    .AddItem vbTab & "Heading Text One" & vbTab & _
      "Heading Text Two" & vbTab & "Heading Text Three" & _
      vbTab & "Heading Text Four"
    .Rows = 12
    .FixedRows = 1
    .WordWrap = True
    .RowHeight(0) = .RowHeight(0) * 2

    .ColWidth(0) = lngWidth / 4
    .ColWidth(1) = lngWidth / 4
    .ColWidth(2) = lngWidth / 4
    .ColWidth(3) = lngWidth / 4

    For intLoopCount = 1 To (.Rows - 1)
        .TextMatrix(intLoopCount, 0) = "Item " & intLoopCount
    Next intLoopCount
End With
```



	Heading Text One	Heading Text Two	Heading Text Three
Item 1			
Item 2			
Item 3			
Item 4			
Item 5			
Item 6			
Item 7			
Item 8			
Item 9			
Item 10			

Erode Arts and Science College (Autonomous)

Department of Computer Science (SF)

VISUAL PROGRAMMING

Staff Name: M.Nagaraj, M.C.A., M.Phil., **Course : III B.Sc (CS)**

Semester : V

UNIT- V

ODBC And Data Access Object

Introduction:

- **Open Database Connectivity (ODBC)** provides universal database connectivity **Application Programming Interface (API)** that enables applications to access data in a wide range of proprietary databases.
- Based on the X/Open **SQL Access Group's Call Level Interface (CLI)** specification, ODBC is an open, vendor-neutral way to uniformly access data stored in different formats and database engines.
- ODBC is the most widely used interface to relational data. It's also quite fast, but you pay for the fast access with complex application code.

The general characteristics of ODBC are:

- ✓ Highly efficient performance.
- ✓ Coding difficulty.
- ✓ Reasonable memory requirements.
- ✓ Compatibility with existing database technologies.
- ✓ Portability across many operating system platforms.
- ✓ A connection model that allows for different networks, security systems, and database options.

Evaluation of computing Architecture

Centralized System Architecture

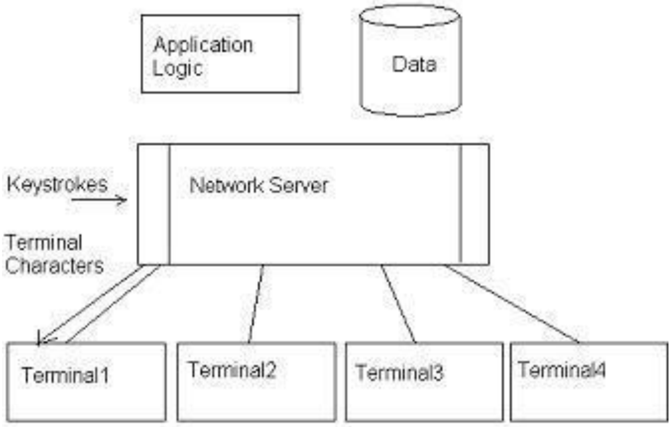
- Companies that needed real computing power turned to the mainframe computer, which is **centralized system architecture**.
- Marshalling is the process of packaging interface elements and sending them across process boundaries. Thus in a centralized system, keystrokes are marshaled from the terminal (client) to the host.
- The centralized system architecture is illustrated in the figure shown below. This is very different from the currently available client-server architecture. The pros and cons of using this type of architecture are listed below.

Advantages of Centralized System Architecture

- Excellent Security
- Centralized administration as both application logic and data reside on the same machine

Disadvantages of Centralized System Architecture

- Mainframe computers are very expensive to buy, lease, maintain and use
- The limitation is that both the application and the database live within the same machine process thereby offering no way to partition the application logic beyond the physical limitations of the mainframe.



Data Access option

What is Data Access?

- **Data access** is a feature of Visual Basic that allows you to access and manipulate any database from Visual Basic. The database may be either **MS-Access database or FoxPro** database or it may also be any of the relational databases such as **Oracle**.
- You can develop applications in Visual Basic that can access data in a database. The database may be on the same machine as the application or it may be on **database server** that is far away from Visual Basic application.

The following are the various ways of access database:

- Data Access Objects (DAO)
- Remote Data Objects (RDO)
- ActiveX Data Objects (ADO)

Data Access Objects (DAO)

- **Data Access Objects (DAO)** can be used either with the Microsoft Jet database engine or, using the ODBCDirect option, without it.
- This chapter discusses design and implementation issues that arise when using the **Data Access Objects (DAO) to access remote databases.**

The general characteristics of DAO are:

- ✓ Difficulty in coding.
- ✓ Flexibility, with facilities to access many different data sources.
- ✓ Adequate-to-slow performance.
- ✓ Dynamic Data Language (DDL) functionality.
- ✓ Support for complex cursors.

Remote Data Objects (RDO)

- **Remote Data Objects (RDO)** is specifically designed to access **remote ODBC relational data sources,** and makes it easier to use ODBC without complex application code.

- ❖ RDO is a primary means of accessing **SQL Server, Oracle, or any relational database that is exposed with an ODBC driver.**

The general characteristics of RDO are:

- ✓ Simplicity (when compared to the ODBC API).
- ✓ High performance against remote ODBC data sources.
- ✓ Programmatic control of cursors.
- ✓ Complex cursors, including batch.
- ✓ Ability to return multiple result sets from a single query.
- ✓ Synchronous, asynchronous, or event-driven query execution.
- ✓ Reusable, property-changeable objects.
- ✓ Ability to expose underlying ODBC handles (for those ODBC functions that are not handled by RDO).
- ✓ Excellent error trapping.

ActiveX Data Objects (ADO)

- ➢ **ActiveX Data Objects (ADO)** is designed to be an easy-to-use **application-level interface to any OLE DB data provider**, including **relational and non-relational databases, e-mail and file systems, text and graphics,** and custom business objects, as well as existing ODBC data sources.
- ➢ **ADO** is easy to use, **language-independent, implemented with a small footprint, uses minimal network traffic**, and has few layers between the client application and the data source — **all to provide lightweight, high-performance data access.**

The general characteristics of ADO are:

- ✓ Ease of use.
- ✓ High performance.
- ✓ Programmatic control of cursors.
- ✓ Complex cursor types, including batch and server- and client-side cursors.
- ✓ Ability to return multiple result sets from a single query.
- ✓ Synchronous, asynchronous, or event-driven query execution.

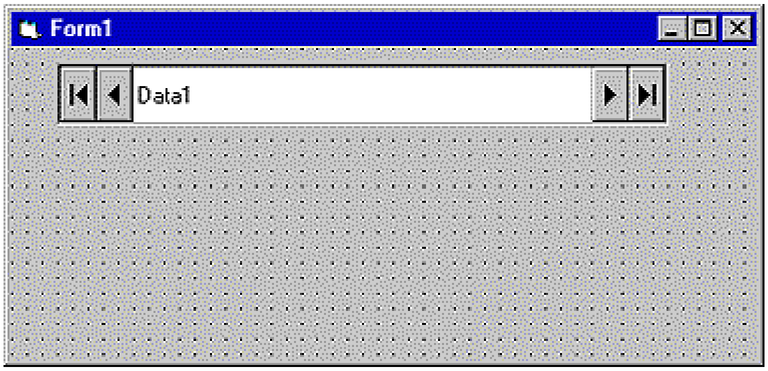
- ✓ Reusable, property-changeable objects.
- ✓ Advanced recordset cache management.
- ✓ Flexibility — it works with existing database technologies and all OLE DB providers.
- ✓ Excellent error trapping.

ODBC using Data Access object and remote data Objects:

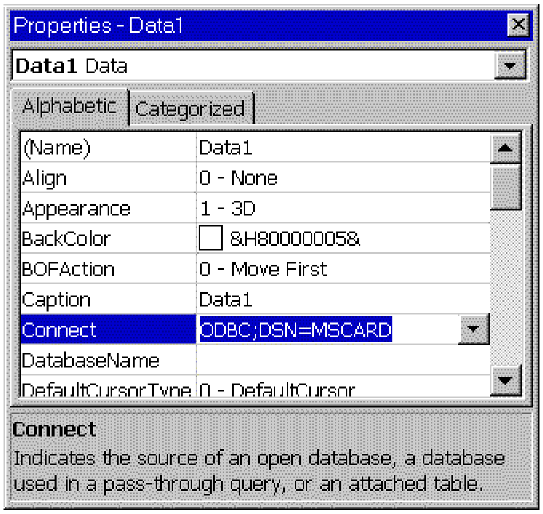
ODBC using Data Access object:

- ➢ Using Data Access Objects (Dao) Ms Visual Basic With Minisoft's Odbc Driver
- ➢ The following steps to be follow:

Step1: Create a new blank project and Place a 'Data' object on your form.odbc



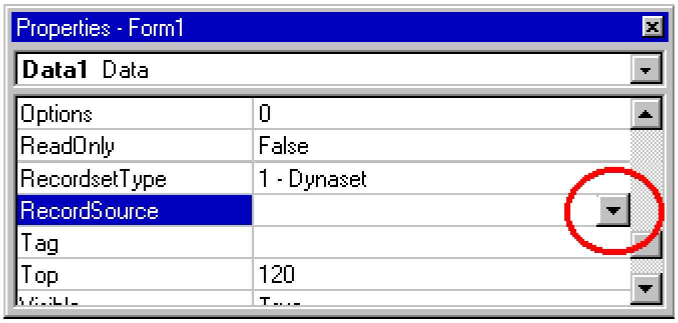
Step2: In the 'Connect' property, type the string "ODBC;DSN=datasource" (without quotes). Replace datasource with the datasource name you entered for the connection created in the 32bit ODBC Administrator.



(Name)	Data1
Align	0 - None
Appearance	1 - 3D
BackColor	□ &H80000005&
BOFAction	0 - Move First
Caption	Data1
Connect	ODBC;DSN=MSCARD
DatabaseName	
DefaultCursorType	0 - DefaultCursor

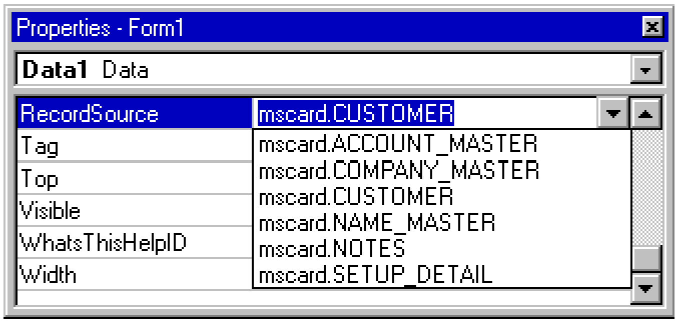
Connect
Indicates the source of an open database, a database used in a pass-through query, or an attached table.

Step3: Go to the ‘RecordSource’ property and click on the dropdown arrow.ODBC_f03



Options	0
ReadOnly	False
RecordsetType	1 - Dynaset
RecordSource	
Tag	
Top	120

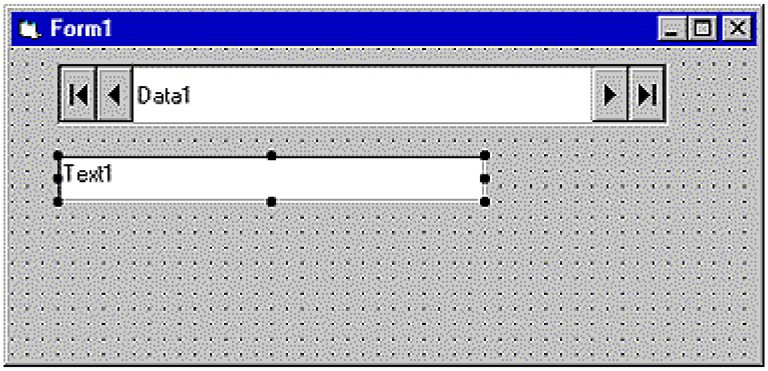
Step4: After a few moments, a list of datasets will appear. Select one.ODBC_f04



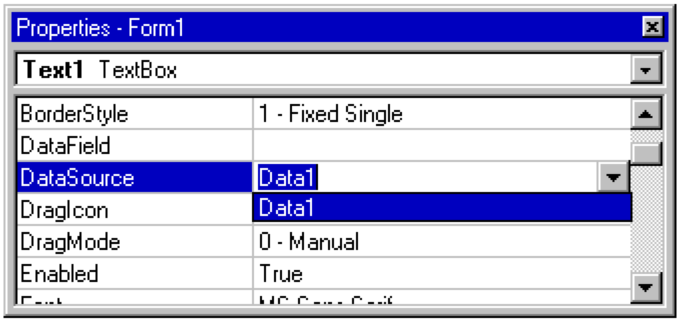
RecordSource	mscard.CUSTOMER
Tag	
Top	
Visible	
WhatsThisHelpID	
Width	

mscard.ACCOUNT_MASTER
mscard.COMPANY_MASTER
mscard.CUSTOMER
mscard.NAME_MASTER
mscard.NOTES
mscard.SETUP_DETAIL

Step5: Add a TextBox to your form.ODBC_f05



Step6: The DataSource dropdown list has the name of the Data object you added earlier. Select it.ODBC_f06

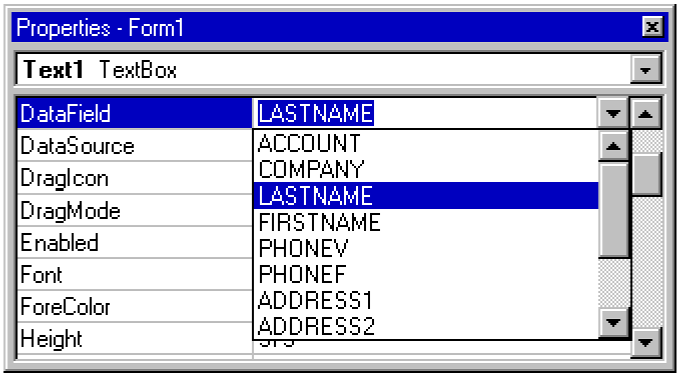


Properties - Form1

Text1 TextBox

BorderStyle	1 - Fixed Single
DataField	
DataSource	Data1
DragIcon	Data1
DragMode	0 - Manual
Enabled	True
Font	MS Sans Serif

Step7: The DataField dropdown box will now have the fields available in the dataset you selected. Select one.ODBC_f07



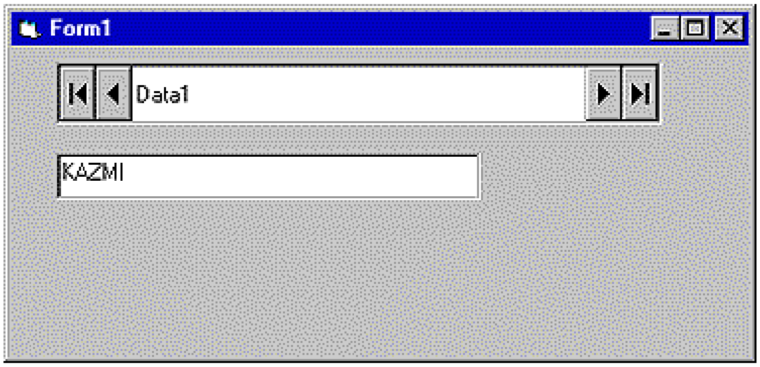
Properties - Form1

Text1 TextBox

DataField	LASTNAME
DataSource	ACCOUNT
DragIcon	COMPANY LASTNAME
DragMode	FIRSTNAME
Enabled	PHONEV
Font	PHONEF
ForeColor	ADDRESS1
Height	ADDRESS2

Step8: Run the application.

Step9: The TextBox contains the value of the field you selected from the first record in the dataset. Use the left and right arrows to scroll through the data. You can add other fields as additional text boxes.ODBC_f09



Form1

Data1

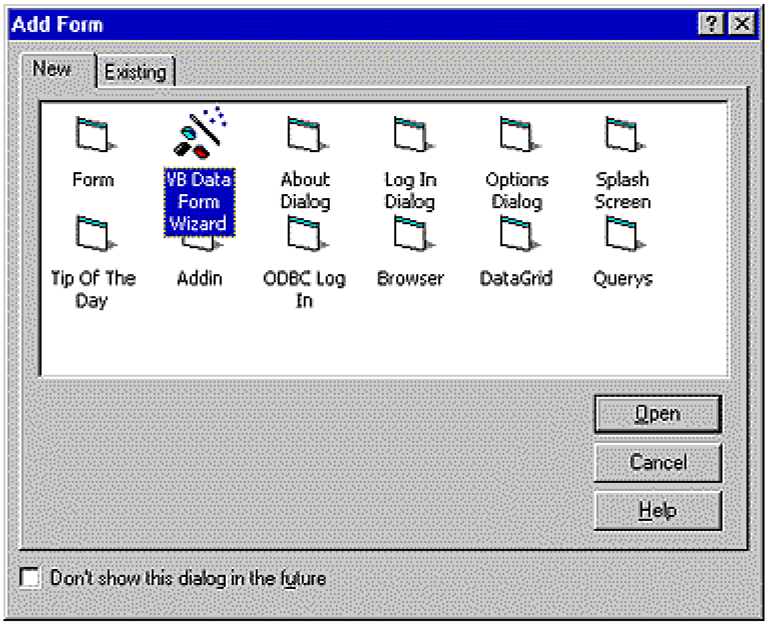
KAZMI

ODBC using remote data Objects:

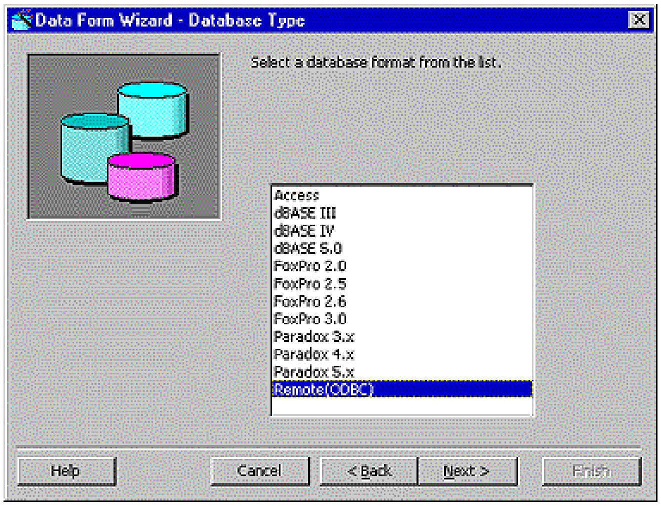
The following are the first steps in using RDO for accessing your HP e3000 data. Use RDO in place of DAO where you wish to have more control over how the data is handled.

Step1: Create a new blank project.

- Add a new form using the VB Data Form Wizard. To create a new form select Project from the menu bar, then click Add Form. The Add Form dialog box will then appear.odbc_u01



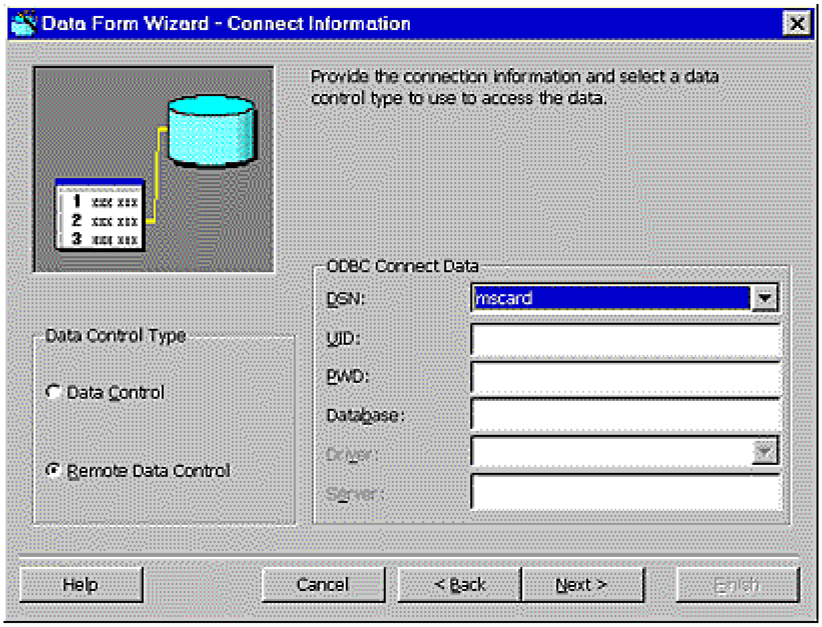
Step2: Select Remote(ODBC) as the Database type. Then click Next.odbc_u02



Select a database format from the list.

Access
dBASE III
dBASE IV
dBASE 5.0
FoxPro 2.0
FoxPro 2.5
FoxPro 2.6
FoxPro 3.0
Paradox 3.x
Paradox 4.x
Paradox 5.x
Remote(ODBC)

Step3: Under Data Control Type choose “Remote Data Control”. Under ODBC Connect Data select the DSN you created using the HP3000 Data Access Driver. Then click Next.msvb5



Data Form Wizard - Connect Information

Provide the connection information and select a data control type to use to access the data.

Data Control Type

○ Data Control

● Remote Data Control

ODBC Connect Data

DSN: mscard

UID:

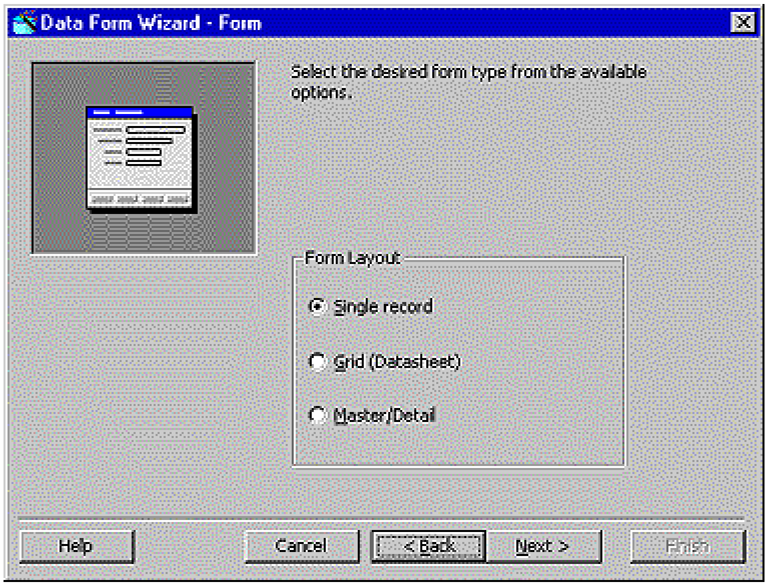
PWD:

Database:

Driver:

Server:

Step4: Under Form Layout select Single record, then click Next.odbc_u04



Data Form Wizard - Form

Select the desired form type from the available options.

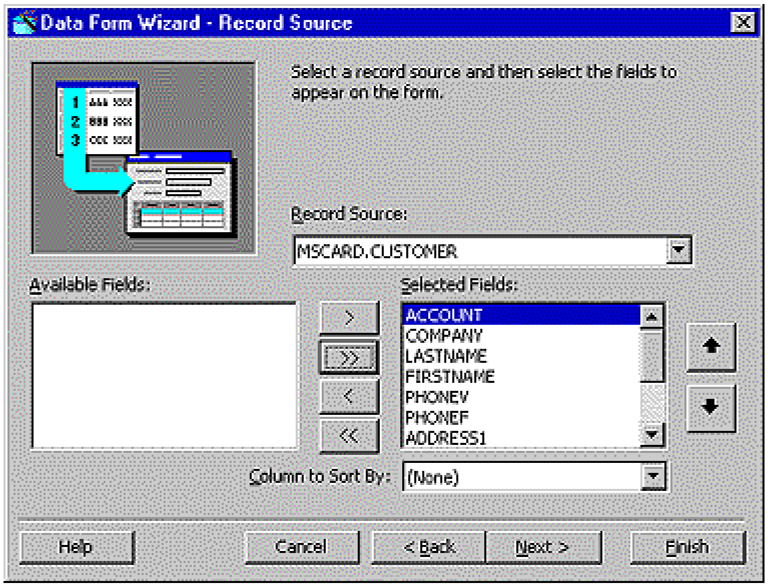
Form Layout

● Single record

○ Grid (Datasheet)

○ Master/Detail

Step5: Select the Dataset or File you wish to view as the Record Source. Add Available Fields to your form. Then click Next.odbc_u05



Select a record source and then select the fields to appear on the form.

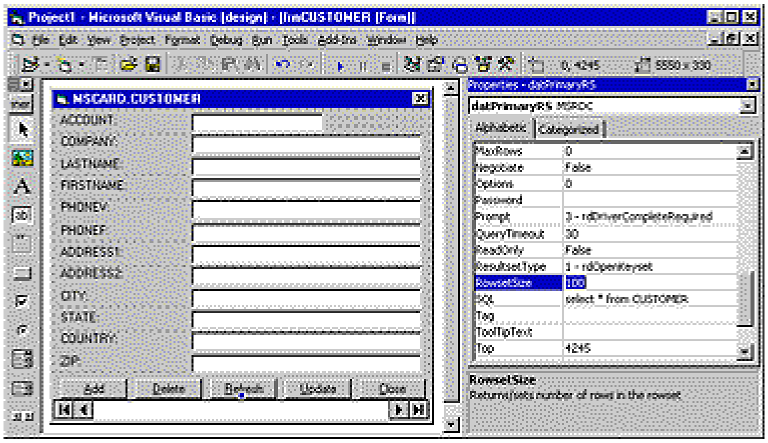
Record Source: MSCARD.CUSTOMER

Selected Fields:

ACCOUNT
COMPANY
LASTNAME
FIRSTNAME
PHONEV
PHONEF
ADDRESS1

Column to Sort By: (None)

Step6: Change the MSRDC properties to optimze the use of the selection. For speedier browsing, choose a smaller value for RowsetSize. To limit the result set, enter a WHERE clause in the SQL statement odbc_u06



MaxRows	0
Negotiate	False
Options	0
Password	
Prompt	3 - rdDriverCompleteRequired
QueryTimeout	30
ReadOnly	False
ResultsetType	1 - rdOpenKeyset
RowsetSize	100
SQL	select * from CUSTOMER
Tag	
ToolTipText	
Top	4245

RowsetSize
Returns/sets number of rows in the rowset

Data Environment and data Report:

Introduction:

- **There are two steps to creating a data report.**
- **First**, **we need to create a Data Environment**. This is designed within Visual Basic and is used to tell the **data report what is in the database**.

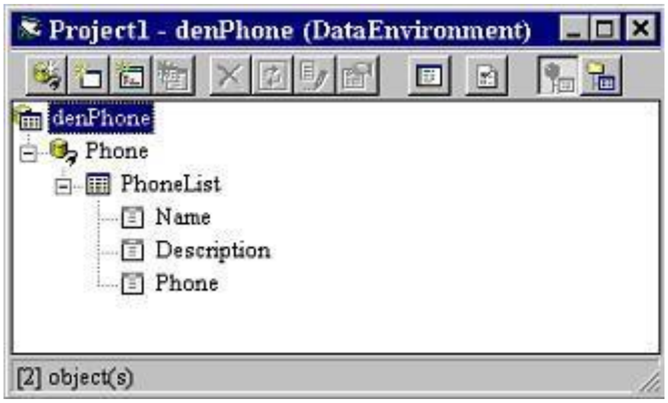
- **Second**, **we create the Data Report itself**. This, too, is done within Visual Basic. The **Data Environment and Data Report files then become part of the Visual Basic project developed as a database management system.**.

Example - Phone Directory - Building a Data Report

- We will build a data report that **lists all the names and phone numbers in our phone database**.
- We will do this by first creating a **Data Environment, then a Data Report.** We will then reopen the **phone database management project and add data reporting capabilities.**

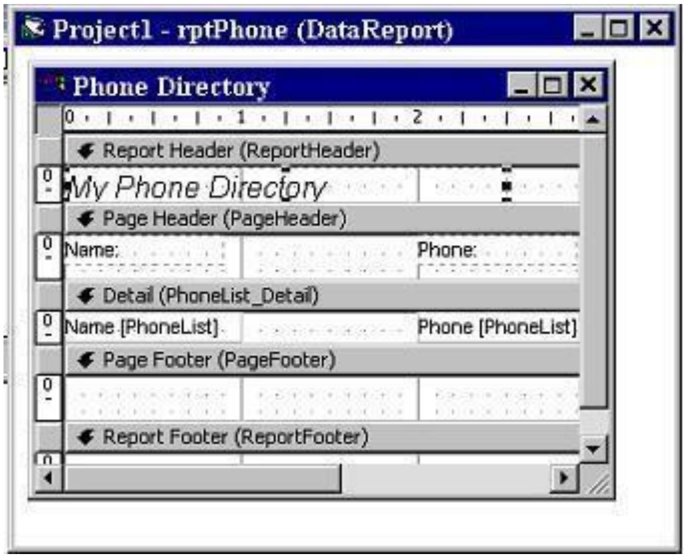
Creating a Data Environment

- ✓ Start a **new Standard EXE** project.
- ✓ On the Project menu, click **Add Data Environment**. If this item is not on the menu, click Components. Click the Designers tab, and **choose Data Environment and click OK to add the designer to your menu.**
- ✓ In the Data Environment window, **right-click the Connection1** tab and select **Properties**. In the Data Link Properties dialog box, **choose Microsoft Jet 3.51 OLE DB Provider. Click Next** to get to the Connection tab. **Click the ellipsis button. Find your phone database (mdb) file**. Click OK to close the dialog box.
- ✓ **Right-click the Connection1 tab and click Rename**. **Change the name of the tab to Phone.** Right-click this newly named tab and click **Add Command to create a Command1 tab.** Right-click this tab and choose Properties. **Assign the following properties:**
    - Command Name – PhoneList
    - Connection – Phone
    - DataBase Objec t- Table
    - ObjectName – PhoneList
- ✓ Click OK. All this was needed just to connect the environment to our database.
- ✓ Display the properties window and give the **data environment a name property of denPhone**. Click File and Save denPhone As. Save the environment in an appropriate folder. We will eventually add this file to our **phone database management system.**



Creating a Data Report

- ❖ Once the Data Environment has been created, we can create a Data Report.
- ❖ We will drag things out of the Data Environment onto a form created for the Data Report, so make sure your Data Environment window is still available.
- ✓ **click Add Data Report and one will be added to your project.** If this item is not on the menu, click Components. **Click the Designers tab, and choose Data Report and click OK to add the designer to your menu.**
- ✓ **Set the following properties for the report:**
  - Name - rptPhone
  - Caption - Phone Directory
  - DataSource - denPhone (your phone data environment - choose, don’t type)
  - DataMember - PhoneList (the table name - choose don’t type)
- ✓ Right-click the **Data Report** and click **Retrieve Structure**. This establishes a report format based on the **Data Environment.**

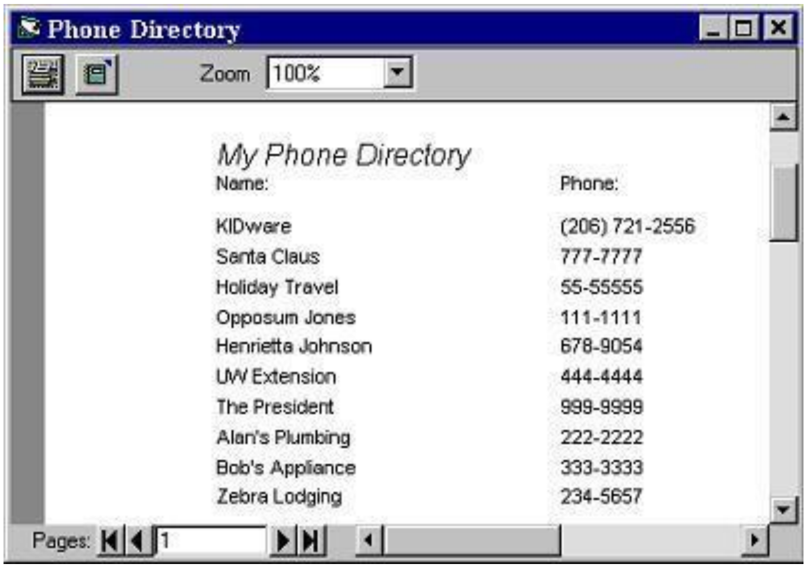


<u>Accessing the Data Report</u>

- Reopen the phone directory project. Add a command button named **cmdReport** and give it a Caption of **Show Report.**
- We will now add the **data environment** and **data report** files to the project. Click the Project menu item, then click Add File. Choose **denPhone** and click OK.
- Also add **rptPhone**. Look at your Project Window. Those files should be listed under Designers.
- Use this code in **cmdReport_Click:**

```
Private Sub cmdReport_Click()
rptPhone.Show
End Sub
```

- This uses the **Show method to display the data report.**
- **Save the application and run it.** Click the Show **Report button and this should appear**.



My Phone Directory

Name:	Phone:
KIDware	(206) 721-2556
Santa Claus	777-7777
Holiday Travel	55-55555
Opposum Jones	111-1111
Henrietta Johnson	678-9054
UW Extension	444-4444
The President	999-9999
Alan's Plumbing	222-2222
Bob's Appliance	333-3333
Zebra Lodging	234-5657