UNIT – I : Visual Basic.

INTRODUCTION TO VISUAL BASIC

- Visual basic is an ideal medium for developing Windows based application. It is an event driven programming language.
- Visual basic is considered as the fastest and easiest way to create applications for Microsoft Windows. It provides with a complete set of tools to simplify rapid application development.
- The “Visual” part refers to the method used to create the Graphical user interface (GUI). Instead of writing lines of codes to describe the appearance and location of elements, we simply drag and drop pre-built objects into place on the screen.
- The “Basic” part refers to the BASIC language, a language used by more programmers than any other language in the history of computing. Visual Basic has evolved from the original BASIC language and it contains several hundred statements, functions, and keywords, which relate to the windows GUI.

FEATURES OF VISUAL BASIC

- Data access features allow creating database and front-end applications for most popular database formats, including Microsoft SQL Server and other enterprises-level databases.
- ActiveX technologies allow using the functionality provided by other applications, such as Microsoft Word, Microsoft Excel spreadsheet, and other Windows applications.
- Internet capabilities makes it easy to provide access to documents and applications across the Internet form within your application.(ActiveX Documents).
- Your finished applications is a true .exe file that uses a run-time dynamic-link library (DLL) that you can freely distribute.(Application Setup Wizard).

VISUAL BASIC EDITIONS

- The Visual Basic **Learning Edition,** allows the programmers to easily create powerful applications for Microsoft Windows 95 and Windows NT(r). It includes all intrinsic controls, grid, tab, and data-bound controls.
- The **Professional edition** provides computer professionals with a full-featured set of tools for developing solutions for others. It includes all the features of the Learning edition, plus additional ActiveX controls, including Internet controls and the Crystal Report Writer. Documentation provided with the Professional edition includes the Programmer's Guide, online Help, the Component Tools Guide and the Crystal Reports for Visual Basic User's Manual.
- The **Enterprise Edition** allows professionals to create robust distributed applications in a team setting. It includes all the features of the Professional edition, plus the Automation manager, Component manager, Database management tools, the Microsoft Visual SourceSafe™ project-oriented version control system and more.

CONTROLS

Any applications must consist of a number of programs or procedures that perform various activities. People behind Visual Basic decided to create special routines that would perform a specific task. Words like routines, procedures, programs were called as controls. For each purpose there have a separate control like to accept data, to display pictures, and to draw lines.

PROPERTIES

The controls are given some attributes that are clearly defined. A control is supposed to do only such and such activity. For example, a text box control can accept multiline text or Word-wrap. All these attributes are called properties therefore each control has certain properties.

EVENTS

There are number of things that can happen to a control. For example, a Command

button control can be clicked, as you click the start button in windows. These are all some of the events that can take place on the application. Visual Basic allows writing code to respond to such activities. Therefore we need to write code only for those events. For example if your program has to respond when the user clicks a command button, we need to write code only for click event.

METHODS

The action taken when the event occurs is the method. A method is a piece of code that accomplishes a task. So in an event driven program, there are 'Controls', which have 'Properties'. When an 'Event' occurs to the 'Control', some 'Methods' are invoked. Unless an event occurs no method will be invoked.

PROGRAMMING LANGUAGE

The following steps are needed to build an application.

1. Design the User Interface
2. Write code to respond to User Input/Events.

DESIGNING THE USER INTERFACE

The User Interface is built using the controls and setting the properties for the controls. For example the location of the Text Box, where the user will enter the Customer Id, or the location where the current date is to be displayed etc.

WRITE CODE TO RESPOND TO USER INPUT/EVENTS

- The code invokes the methods associated with the controls. If the user clicks on the control that displays the next record from the database, or the user selects a particular option, or wants to 'find' the details of a client, etc. All such events have to be acted upon.
- There are number of built-in keywords, associated with the controls that accomplish the given task. Visual basic provides a Form, which is another control. It also has properties, events occur on it and it has Methods associated with it.
- Visual Basic Integrated Development Environment also known as Visual Basic

IDE, has all the tools and fixtures to make the work easy. It has tool bars and menu bars.

CREATING AN APPLICATION

Creating an application in Visual Basic is working with projects. A project is the collection of files that is used to build an application. A project consists of

- One project file that keeps track of all components (.vbp).
- The .frm file. One file for each form(.frm). It contains the description of the properties of the form and the controls on it.
- One binary data file for each form containing data for properties of controls on the form (.frx). These files are not editable and are automatically generated for any .frm file that contains binary properties, such as picture or icon.
- The (.cls) file for each class module. This file is optional. The class file is created when the own objects are created.
- The standard (.bas) file . This is also optional, one file for each standard module. It contains module level declarations, procedures.
- The ActiveX Control (.ocx) file , becomes a part of the project file only if optional controls are added in the program.
- The resource (.res) file, contains bitmaps, text strings that are used in the program. We can have only one resource file.

The project file is simply a list of all the files and objects associated with the project as well as information on the environment options. This information is updated every time the project is saved. All these files and objects can be shared by other projects as well. When all the files for a project is completed we can convert the project into an executable file. We can also create other type of executable files such as .ocx, .dll files etc.

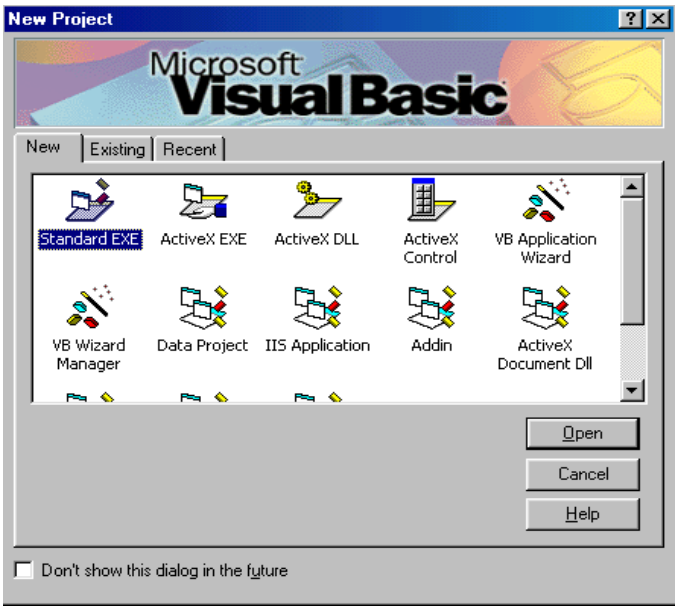
Example:

Visual Basic 6.0 (VB6) can be invoked by two ways:

- ➤ Double clicking on the shortcut path

➤ Going through the pull-up menu from start.

VB6 will first show a screen with the name of the owner of this copy of the software as shown below.

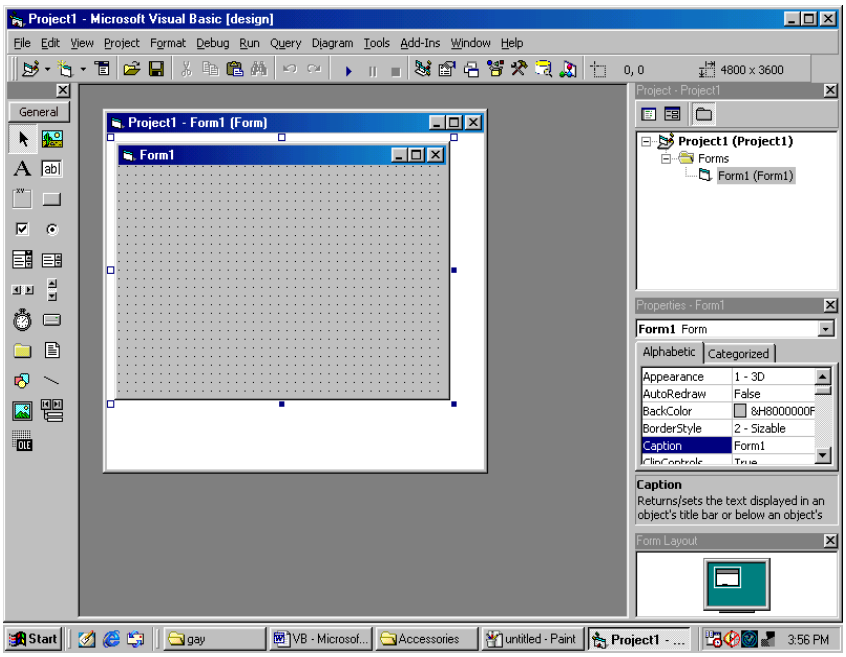


Here a number of icons are displayed along with the type of projects that each will start. The New Project Screen has three tabs:

- New
- Existing
- Recent

- **New:** It is the current tab used to open the new project file.
- **Existing:** It is used to display the existing projects on your screen.
- **Recent:** It is used to display the projects on which you have recently worked.

First choose the Standard.Exe Icon by clicking on it. Then the following screen will be displayed.



In all cases the types of files listed are the following

- .vbp : Visual basic Project File
- .vbg : Visual Basic Group File
- .mak : Project files built with earlier versions of Visual Basic.

Right on top we see “Project1-Microsoft Visual Basic [design]. This is the title of the working project. The title is displayed on the title bar and it tells that we are currently working on Project1 and we are in ‘design’ stage or design mode. There are two other modes. They are ‘run’ mode when the programs are running and ‘break’, while programs are being debugged.

PROCEDURAL

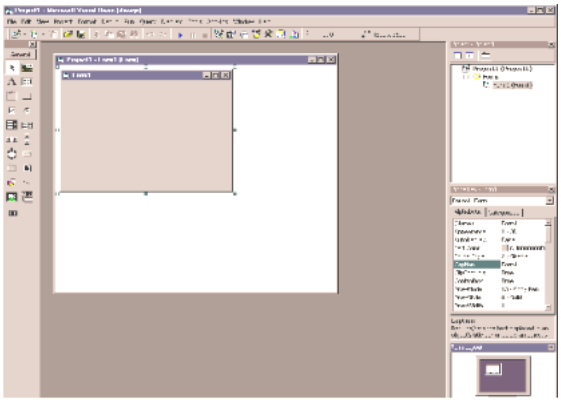
```
Private Sub add_Click()
        text3.Text = text1.Text + text2.Text
end Sub
```

IDE, FORMS AND CONTROLS

- **The Integrated Development Environment (IDE)** consists of the following

elements:

- Title bar
- Menu bar
- Toolbars
- Form window
- Toolbox
- Project Explorer window
- Properties window
- Form Layout window
- Code Editor window



VB ENVIRONMENT:

- **Title bar:** The title is displayed on the title bar. [“Project1-Microsoft Visual Basic design]”].
- **The Menu bar has the following Menus**:

File Menu : To open and save a new or existing project, to print and to make a project file.

Edit Menu : For all editing requirement **Cut, Paste, Find, Undo**, etc.

View Menu : To view the various parts of the project, and the Visual Basic environment.

Project Menu : Inserting or Removing forms, or objects to the project.

Format Menu : For spacing, placing and appearance of controls in the form.

Debug Menu : To remove the errors that have crept in.

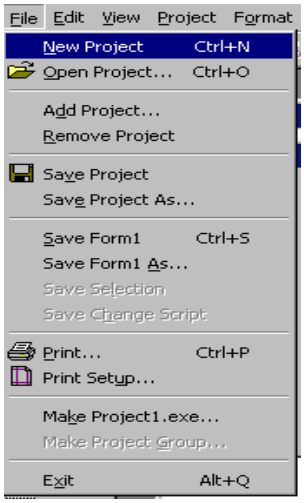
Run Menu : To compile, start and stop a program.

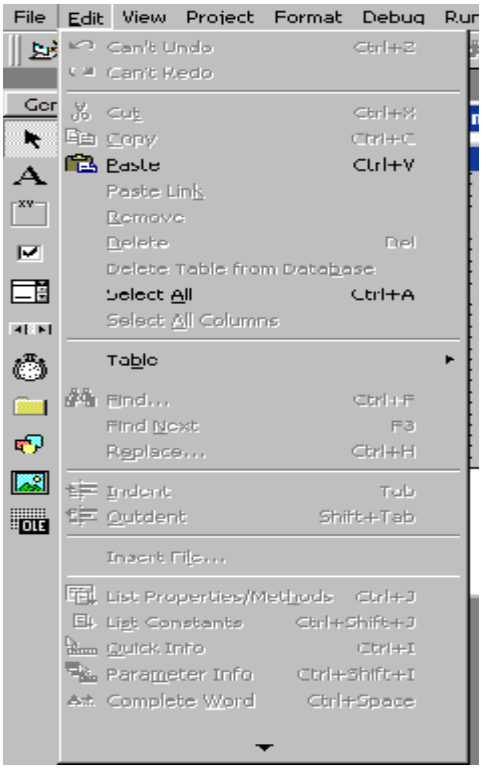
Tools Menu : To add procedure and to customize the environment for the project.

Add-Ins Menu : To add tools like **Data Manager**, other **Wizards**, etc.

Window Menu : Arranging appearance of various windows on the desktop.

Help Menu : For the on-line help that every programmer needs to refer to.

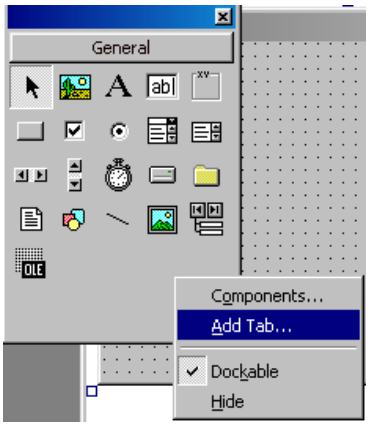




- **Toolbars:** It provides quick access to commonly used commands in the

programming environment. By default the standard toolbar is displayed when you start visual basic.

- **The Tool Box:** To the left of the Form1 window is the toolbox. It provides set of tools for design time to place controls on a form. If the toolbox is not open, display it by using the Toolbox command on the View menu.

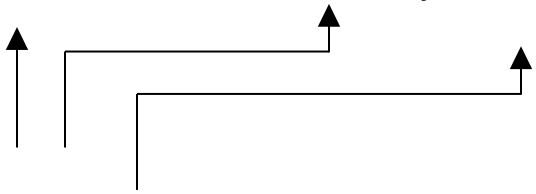


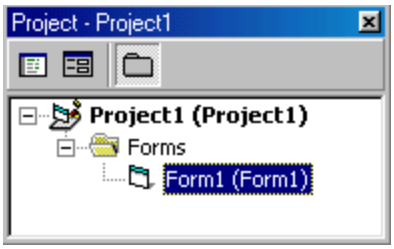
- **Form Window:** The blank window in the upper-left corner of the center window, which has a grid of dots. This is the form used to customize by adding controls.
- **The Project Explorer:** The Project Explorer is located on the right side . It organizes the application as one project. All the code, and controls that are used in the applications are stored in separate files. It is called a 'Project Explorer' because it has an interface like the Explorer and it deals with the working project.

The Project Explorer has three icons on its tool bar. Each icon represent a function.

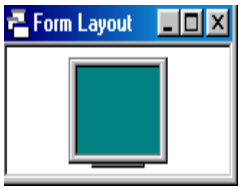
1. To view the code
2. To view the controls
3. To show or hide the forms

Icon to View code to view object to toggle folders

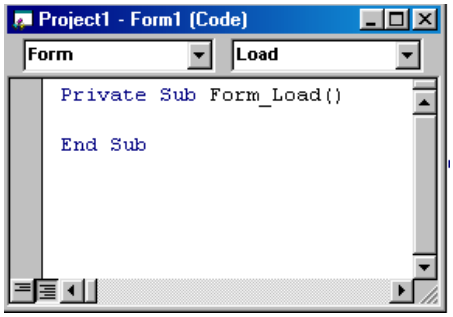




- **Form Layout Window:** This window allows you to position the forms in your application.



- **The Properties Window:** This Window lists all the properties for an object or control used in Visual Basic. Currently the only object is the Form1. This is the big blank window that is sitting in the middle of the screen. The caption in the properties window reads 'Form1'.And that is the name on the title bar of the Form1 window. We can change the caption, the height, the width, etc. Each object has a number of properties that can be changed as the need dictates.
- **Code Editor Window:**
  - It serves as an editor for entering application code.
  - A separate code editor window is created for each form or code module in the application.
  - The programmer can change the font size of the code.



SAVING THE PROJECT:

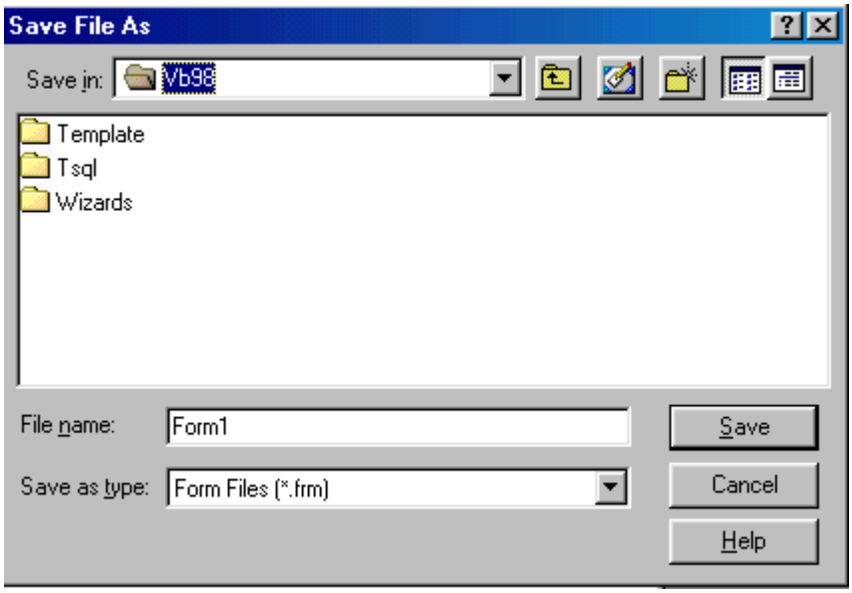
Go to file menu and choose the Save project option.

Then the name of the Form file will be asked. This is the name of the file and is

different from Form caption. Then the project name will be asked and the user will give it.

Note:

While saving the project on each time, Visual basic updates the project file(.vbp).A project file contains the same list of files that appears in the Project Explorer window.



There are **Four** options available with regard to **opening** an **saving** projects. They are

- New Project
- Open Project
- Save Project
- Save Project As

New Project:

- Closes the current project, prompting you to save any files that have changed.
- We can select a type of project from the New Project dialog.
- Visual Basic creates a new project with a single new file.

Open Project:

Closes the current project, prompting you to save any changes. Visual Basic the

opens an existing project, including the forms, modules, and ActiveX controls listed in its project (.vbp) file.

Save Project:

Updates the project file of the current project and all of its form, standards, and class modules.

Save Project As:

Updates the project file of the current project, saving the project file under a file name that is to be specified.

Restart VB:

In the New Project Screen, press the recent tab. The project will be listed on the top.

Types of Project:

Following are the options available with Visual Basic:

Standard:

This project type must be chosen for the small or large standalone application is developed.

ActiveX EXE:

This option is chosen for the creation of an executable component. An ActiveX executable component can be executed from other application. This will be a program that can provide functionality to a number of other applications.

ActiveX Control:

This helps to create a custom ActiveX control that can be used in other application.

ActiveX DLL:

Like the ActiveX EXE it provides added functionality to the application, but will work 'in-process' with an application.

Data Project:

Choose this option to create a project with the database components.

IIS:

This helps to create an Internet application.

ActiveX Document:

Creates a component that can take over the application at runtime. It creates an Internet application that can be executed from a browser.

DHTML Application:

Creates an application that can be executed from a web browser only.

TOOL BOX:

A toolbox contains number of controls. These controls are used to create a simple application. It contains 21 controls. They are

Text Box	Picture Box	Label Box	Option Button
Frame Control	List Box	Combo Box	Data Control
H Scroll Bar	V Scroll Bar	Command Button	Check Box
Drive List Box.	Dir Control	File List Box	Line Control
Shape Control	Image Control	OLE	Timer

Control & Pointer

Text Box 

Text box control is a versatile control. It can be used for getting the input from the user and also display the result. It is called as Edit field or Edit control.

Picture Box 

It can display a graphic from a bitmap, icon or metafile as well as enhanced metafile, JPEG or GIF files.

Label Box 

It allows you to display text that you don't want the user to change, such as caption under a graphics.

Option Button 

It allows you to display the multiple choices from which the user can choose only one option.

Frame Control 

The frame control allows you to create graphical or functional grouping controls. To group controls, draw the frame first and then draw controls inside the frame.

List Box 

List box displays a list of items. It can occupy some space on the form.

Combo Box 

Combo box is also used to display the items. But it can display a single item at a time. It can occupy a less space on the screen.

Data Control 

Data control provides access to database through controls on your form.

H Scroll Bar (Horizontal Scroll Bar) 

Provides a graphical tool for quickly navigating through a long list of items or a large amount of information for indicating the current position on a scale, or as an input device or indicator of speed or quantity.

V Scroll Bar (Vertical Scroll Bar) 

Provides a graphical tool for quickly navigating through a long list of items or a large amount of information for indicating the current position on a scale, or as an input device or indicator of speed or quantity.

The Pointer: 

The first item on the toolbox is not a control but is used to manipulate controls after you create them. Click the pointer when you want to select, resize, or move an existing control. The Pointer is automatically activated after you place a control on a form.

Command Button 

Creates a button that the user can choose to carry out a command. The user will click on this button and the computer will perform the task associated with the button.

Check Box 

It allows you to display the multiple choices from which the user can choose any

number of items.

Drive List Box, Directory List Box & File List Box

These controls are used to display available Drives, Directories and Files. The user can select a valid Drive on his system. The user can see a hierarchical structure of directories and files.

Line control and Shape control

These controls are used to draw lines, square, circles, rectangles and etc.

Image control

It is very similar to the picture box control. Stretch property is available in Image control. So we can easily resize the images.

OLE (Object Linking & Embedding)

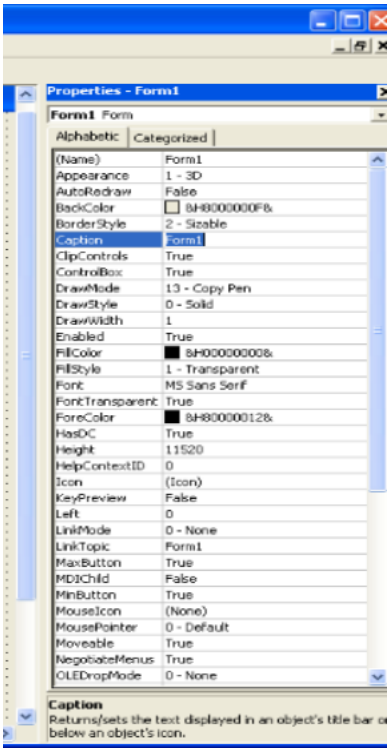
This control allows you to link your program to another object or program like as Ms Word, Internet Explorer, and Ms Word etc.

Timer control

It is a very important control. It is used for time-related processing. It is mainly used in the auto save option.

PROPERTIES WINDOW:

- It is located above the Form Layout window on the right-hand side of the VB environment. If the properties window is hidden make it visible by pressing F4, or choose View→|Properties.
- The title bar of the properties window indicates the properties of control. The line below the title bar indicates what object the user working with.
- It consists of two columns. The first column indicates the properties.
- The second column indicates the current setting of the property. It is also called as settings box.



- The right-hand column is working like an ordinary text box, the method for changing the setting for a property by :
  - ➤ Move the mouse until the mouse pointer is in the right column of the correct line in the properties window.
  - ➤ Click at the location where the text to be inserted.
  - ➤ Enter the text.
- The properties window consists of two Tabs. The first default Tab is alphabetical. The properties to be listed by functionality by using the second Tab categorized at the top of the properties window.

Moving Through the Properties Window:

- To quickly move through the properties in the properties window use Shift+CTRL+letter key. This moves to the first property that begins with that letter. Subsequent uses of this combination move to succeeding properties that begin with that letter. (OR) Use the arrow keys or the mouse to scroll through the properties window.

  Example:
- Whenever a property has a fixed number of options the arrow in the line of the properties window indicates that a drop-down list box is available.
- The Max Button property consists of two values. The default value is True. To change this property to False by using three ways:

1. Once you have highlighted this line in the properties window, the simplest way to do this is just to press F. the Max Button property changes from True to False.
2. Double-click in the right-hand column.
3. To click the arrow immediately to the right of the settings box where True appears. To select the False option by pressing DOWN ARROW and then pressing ENTER, or by clicking the word False.

Working with the Properties Window:

- Display the properties window by pressing F4 if it is not visible.
- Move to the properties window and select an item from the properties in the list box.

- Enter the new setting for the property.
- Press ENTER to accept new string.
- Keyboard Shortcuts:

Key board shortcuts for manipulating the properties window

Key	Action
SHIFT+CTRL+Letterkey	Moves to the first item beginning with that letter
Down arrow	Moves to the next item in the properties window
UP Arrow	Moves to the previous item in the properties list box
Page Down / END	Moves to the last item displayed in the properties list box or to the last item
Page UP / HOME	Moves to the first item displayed in the properties list box or to the first item
F4	Brings up the properties window

COMMON FORM PROPERTIES:

- The Form consists of many properties. Some of the commonly used properties are:
- **Caption:** This property sets the title of the form. It can be changed at runtime. It must be meaningful and informative to the user.
- **Name:** It is used only in code. It gives the name by which the Form is referred to in your code. The name of a form cannot be changed at runtime.
- **Appearance:** This property determines whether the form will have a three-dimensional look. The default value of 1 indicates the form will appear three-dimensional. If it changes to 0, the form will appear flat.
- **Border Style:** This property determines the type of window that the user will view at runtime. It consists of five values. The default value, 2-Sizable, allows the user to size and shape the form via the hot spots located on the boundary of the form.

  The other values are:

  1. 1-Fixed Single
  2. 3-Fixed Double
  3. 4-Fixed Tool Window

4. 5-Sizable Tool Window
5. 0-None

- Set the Border style value to 0-None, the application does not consists of no border. The form created without a border cannot be moved, resized, or reshaped.
- The value 1-FixedSingle, the user no longer be able to resize the window. The users minimize and maximize the form window.
- The third setting 3-FixedDouble not used for ordinary forms, but it is commonly used for dialog boxes. It gives a nonsizable (it has no hot spots).
- The 4-Fixed Tool Window setting not used very often. Under windows 95/98, this displays the form with a close button. The fifth setting 5-Sizable ToolWindow like a FixedToolWindow setting.
- **ControlBox:** The value of this property is True or False. If it is set to True, the Control Box is visible on the top left-hand corner of the form. The minimize and maximize button are displayed on the title bar of the Form.
- The value of this property is set to False the user not access the minimize and maximize commands.
- **Enabled:** This property consists of two values. The default value is True. If it is set to False, the form cannot respond to any events such as the user clicking on the form.
- **Font:** To access the Font property press Ctrl+Shift+F. It includes the following:
  1. Font Name: Name of the font.
  2. Font Bold: If set to true, the text will be displayed in bold.
  3. Font Size: Set the size of the text in points.
- **Height, Width**: This property indicates the height and width of the form. The user can change their values directly via the properties window.
- To perform this users enter the value in the appropriate line in the right-hand column of the properties window.
- **Icon:** This property determines the icon your application will display when it is minimized on the toolbar. To choose an icon for user applications go to properties window and select the Icon property.

- Click the box containing three dots a dialog box will appear.
- From this dialog box we can choose the icon for the application Left, Top:
- These properties determine the distance between the left or top of the form and the screen. To control these properties by using the Form Layout window.
- **MousePointer, MouseIcon**: The MousePointer is a useful property it set the shape of the mouse pointer. The default value is 0, but as the pull-down list indicates, there are 17 other values. Set the MosuePointer property to a value of 99, the user able to use a custom icon.
- **StartUpPosition:** It is another way to decide on the initial position of the form at run time. It is more precise than using the Form Layout window.
- **Visible:** This property consists of two values. The default value is True. Set the value of this property to False, the form will no longer be visible.
- **WindowState:** This property determines how the form will look at run time. There are three possible settings. A setting of 1 reduces the form to an icon, and a setting of 2 maximizes the form. A setting of 0 is the normal default setting.

SCALE PROPERTIES:

- The scale properties are need to position objects or text in a form accurately. The following properties that affect the scale used in a form:
  1. ScaleMode
  2. ScaleHeight, ScaleWidth
  3. ScaleLeft, ScaleTop

ScaleMode:

- It allows you to change the units used in the forms internal coordinate system. It consists of seven possibilities. The default value is 1.

ScaleHeight, ScaleWidth:

- Use this property when you set up your own scale for the height and width of the form.

ScaleLeft, ScaleTop:

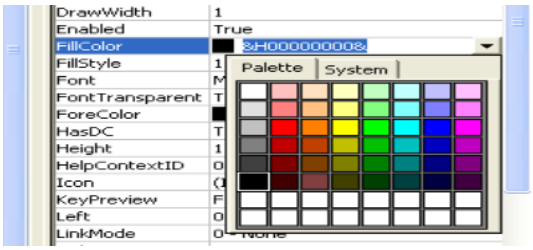
- These properties describe what value visual basic uses for the left or top corner

of the form. The original (default) value for each of these properties is 0.

- These properties are most useful when you are working with graphics like ScaleHeight and ScaleWidth.

COLOR PROPERTIES:

- Using Color properties specify the background color (BackColor) and the foreground color (Forecolor) for text and graphics in the form.



The BackColor and ForeColor Properties via the Color Palette:

- To set the Backcolor property, open the properties window and select BackColor (represented by hexadecimal code-base 16). To set colors is to choose one of the color properties and click the down arrow in the Settings box. This opens up a tabbed dialog box with two tabs.
- The System tab on this dialog box gives the colors currently used by windows for its various elements. To click on the Palette tab, the color grid will appear. Select any color from the Palette and the Color code for that color is placed in the settings box. The background color of the form will automatically show the user changes to the BackGround property.

Working with the Color Palette:

- To create the own colors by working with the color palette directly. Open the color palette by view menu choosing the color palette. (ALT+V, L).
- To the left of the palette, a dark box enclosed in a lighter box. The inner box displays the current foreground color, and the outer box displays the current background color.
- To change the foreground color by clicking the inner box and the clicking any of the colored boxes displayed.
- To change the background color, click the outer box and then click any of the colored boxes displayed.

- To go back to the default colors specified in the Windows control panel, click the Default command button at the right.
- To create own colors for the color palette consists of the following steps:
    1. Click one of these blank boxes, and then click the Define Colors command button. This opens the Define Color dialog box.
    2. Change the amount of color to suit the user needs by adjusting the controls in the dialog box.
    3. Press the Add Color button to create the custom color or the Close button to cancel.

OBJECT ORIENTED AND EVENT DRIVEN

- Move to the event drop-down list box on the right and click the down arrow.
- Move through the box until the user get to the click item.
- Select Form_Click event and click on it.

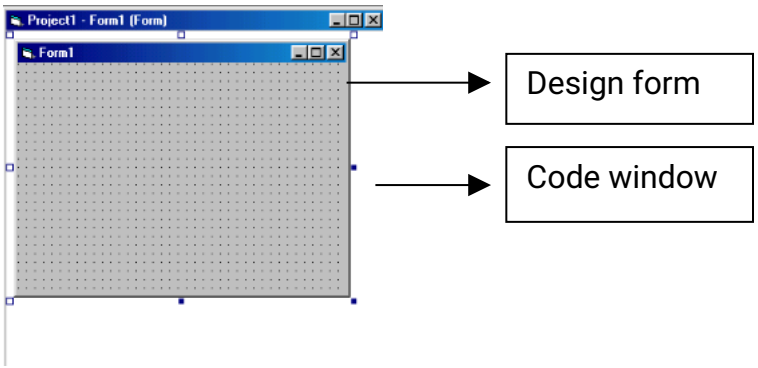
Then Visual Basic does the following:

- Gives the user a new event procedure template for the Form-Click event procedure
- Adds a dotted line between the Form_Load event and the click event.
- Moves the cursor to the blank line before the End Sub line in the click event procedure.

Output

When the user run the form and click once above the form it will show the following

output



FORM EVENTS:

1. Click Event: To respond to a click

2. DBl Click: To respond to a Double click

3. Resize: To respond when the user resizes the form.

4. Initialize Event:

As the name suggests, all the variables associated with this form are initialized.

5. Activate Event:

This event occurs when the form gets user input. This event also occurs when the show method or set focus method of the form is called.

6. Load Event:

During the load event, the form with all its properties and variables is loaded in memory. The load event occurs whenever the "Show" method is executed or a form property is referenced.

7. Deactivate Event:

This event occurs when an another form gets the focus.

8. Unload Event:

When the user closes the form, the form is closed from the memory. Another event called the Query Unloaded event occurs before the unload event fired. The query unload event allows the option to abort the unload event.

9. Terminate Event:

All the memory that was held for the form variable is released.

Example Program for Form Events:

1. Open the new project window.
2. Add Form2 to the new project through **Project→ Add Form**
3. Add command button control to form1.
4. Type the following code to the Form1.

```
Private Sub Command1_Click()
        Form2.Show
End Sub
Private Sub Form_Activate()
        MsgBox "Form Activated"
End Sub
Private Sub Form_Deactivate()
```

```
Private Sub Form_Load()
        MsgBox "Form Loaded"
End Sub
Private Sub Form_Terminate()
        MsgBox "Form Terminated"
End Sub
Private Sub Form_Unload(Cancel As
```

Print Form –A Single command used for getting an image of the form, including whatever is currently displayed on the form to the printer.

The Print Form method tries to send to the printer connected a dot-for-dot image of the entire form.

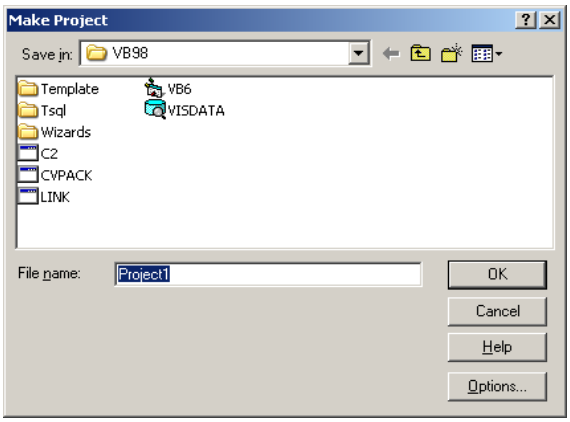
- **Example:**

  Form1.Cls

  Form1.print “WELCOME College”

CREATING STAND-ALONE WINDOWS PROGRAMS:

- One of the most exciting features of Visual Basic is the ability to change your projects into stand-alone Microsoft windows programs.
- To make a stand-alone VB application,



simply go to the File menu and choose the Make Project EXE File option (ALT+F, K). This opens a dialog box like as follows:

- The default name for the .exe version of your file is the project name.
- For the stand-alone program, the Windows desktop uses the same icon that Visual Basic uses for the executable version of the Project.

EXIT FROM VB APPLICATION:

- The three ways are available to run a Visual Basic application:
  1. Select the Start option from the toolbar by clicking the forward arrow.
  2. Select the Start option from Run menu by using the mouse or by pressing ALT+R, S.
  3. Press F5.
- To return to developing an application click on the exit(x) button on the form, or open the Run menu and click the End option, or use the End tool.

WRITING FIRST PROJECT:

There are two methods of creating the Control on the Form;

First Method:

- To put a control on the Form, Click with the left mouse button on the control.
- Next move the mouse pointer to the location on the Toolbar where you want the control on the Form.
- Notice that the pointer has changed to a crosshair.
- Hold down the left mouse button and drag in any direction.
- Release the left mouse button.
- Then the control is appeared on the Form.

Second Method:

- Double click the desired control on the ToolBox.
- That the control will appear at the center.
- Now we can drag the control and place it at any location.

CONTROLS IN VB

Working with Control:

- In the properties of the control, we can change resize the control .
- All the windows application have a uniform size for their button.
- Select a control by simply moving the mouse pointer inside the control and clicking the left mouse button, the sizing handles will appear.
- The four handles on four corners are to increase or decrease the length and breadth of the control.
- The two sizing handles on the horizontal edges are used to increase or decrease the height of the control.
- The two sizing handles on the vertical edges are used to increase or decrease the width of the control.
- To move a control to another location, click on the control, hold down the left mouse button and drag the control to the desired location.

More work on a control:

Click the right mouse button on the control, a pop up menu will be displayed. It will help to control, view the code written for it, and view its properties, also remove the button from the form, make a copy of it.

This menu has another option that is useful when number of controls is used and when they are overlapped. if the currently visible control is hiding another control, this option will get enabled. The options used **are Bring to Front** and **Send to Back**.

Select the control and click on Format on the menu bar. The menu will be displayed as follows

The Lock controls option is used stop the movement of the control from one place to another until you unlock the control.

CODE CONTROL IN WINDOW:

There are several options are used to open the code window.

i) Select the control, press the right mouse button, pop-up menu will appear, choose the view code option.

(Or)

Press **F7**, after selecting the control.

(Or)

Click on **View** in the Menu bar, and then click the **View Code** option.

(Or)

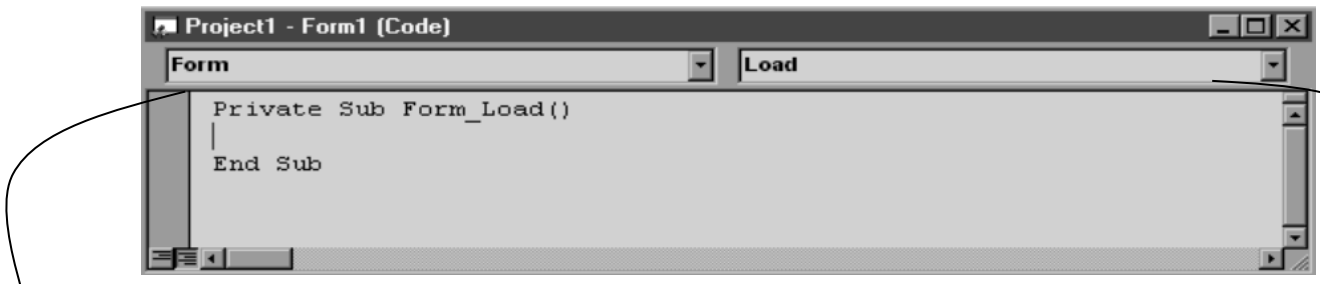
Use the View Code icon on the Project Explorer window.

(Or)

Directly double click the control.

Anatomy of the Code Window:

The title bar of the code window contains the name of the Project and Form name. Next there are two combo boxes. One holds the text "Form" and the other holds the text "Load".



This means that form is the name of the object and load is the event that you want to

write the code for. Click the down arrow button and the Drop Down list box will display all the controls that have been placed in the form.

If you intend to write the Command1, then select it from the list by clicking on it. Now click on the Down arrow button on the other Drop down control. This will list all the events related to the selected object. The lines given below provide a framework within which you can enter the code.

Private Sub Command1_Click ()

End Sub

The Private means that the variables declared and the code used here can be used only by this function. Then we have the word Sub. This is short for Sub-routine or Function. Command1_Click() is the name of the function which is self explanatory. End Sub means end of this subroutine.

THE NAME (CONTROL NAME) PROPERTY:

- This property determines the name by which it is referred to in the program. This property is more important the user avoid this name use Me keyword in your code because VB knows the form that the code is attached to.

  Example: Height = 5000 or Me.Height = 5000
- To change the height of a command button whose Name property is MyCommandButton, represented as

  MyCommandButton.Height=500
- To Double-Click the MyCommandButton the event procedure will look as follows:

```
Private Sub MyCommandButton_Click()
End Sub
```

- The limits on a control name are the same as for form names:
- The name must begin with a letter.
- Use any combination of letters, digits, and underscores.
- It cannot be longer than 40 characters.

PROPERTIES OF COMMAND BUTTONS:

Command Button properties

Property	Description
Name	The name of the object so you can call it at runtime
BackColor	This specifies the command button's background color. Click the BackColor's palette down arrow to see a list of common Windows control colours, you must change this to the style property from 0 - standard to 1 - graphical
Cancel	Determines whether the command button gets a Click event if the user presses escape
Caption	Holds the text that appears on the command button.
Default	Determins if the command button responds to an enter keypress even if another control has the focus
Enable	Determines whether the command button is active. Often, you'll change the enable property at runtime with code to prevent the user pressing the button
Font	Produces a Font dialog box in which you can set the caption's font name , style and size.
Height	Positions the height of the object - can be used for down
Left	Positions the left control - can be used for right

MousePointer	If selected to an icon can change the picture of the mouse pointer over that object
Picture	Hold's the name of an icon graphic image so that it appears as a picture instead of a **Button** for this option to work the graphical tag must be set to 1
Style	This determines if the **Command Button** appears as a standard windows dialog box or a graphical image
Tab index	Specifies the order of the command button in tab order
Tab Stop	Whether the object can be tabbed to (this can be used in labels which have no other function)
Tool Tip Text	If the mouse is held over the object a brief description can be displayed (for example hold your mouse over one of the above pictures to see this happening
Visible	If you want the user to see the button/label select true other wise just press false
Width	Show the width of the object

SIMPLE EVENT PROCEDURES FOR COMMAND BUTTONS:

Double click a control or press F7 to go to code window after that select the required event for the command button and write the procedure for it. For example the command button shown below has the name property as CmdClickMe and the code written for it is (print “WELCOME College”) and the event is Click.

OTHER EVENTS FOR COMMAND BUTTONS:

Events

GotFocus

Occurs when the CommandButton gets focus either from the Tab key, when it is clicked on by the user or by using the SetFocus method.

KeyDown, KeyUp

KeyCode As Integer, Shift As Integer

When the user presses a keyboard key down or releases a key, these events occur.

```
Private Sub Command1_KeyDown(KeyCode As Integer, Shift As Integer)
If KeyCode = vbKeyBack Then
   MsgBox "You pressed the backspace key!"
End If
End Sub
```

The Shift argument contains the value of the Shift, Control and Alt keys. This is useful for performing a different action when one of these keys is pressed in combination with another e.g. pressing 'A' would be different to pressing 'Ctrl+A'.

Constant	Value	Description
vbShiftMask	1	Shift key
vbCtrlMask	2	Ctrl key
vbAltMask	4	Alt key

If a combination of these keys are pressed, the Shift argument will contain the sum of all keys pressed e.g. Ctrl+Alt would be 6.

```
Private Sub Command1_KeyDown(KeyCode As Integer, Shift As Integer)
If Shift = vbCtrlMask + vbShiftMask Then
   If KeyCode = vbKeyA Then
      MsgBox "You pressed the Ctrl+Shift+A key combination!"
  EndIf
End If
End Sub
```

KeyPress

KeyAscii As Integer

This event works differently to the KeyDown and KeyUp events in that it detects the actual character (as the ANSI keycode) that is pressed rather than the keyboard key. E.g.:

Key Pressed	Result
s	115

Shift	Nothing
Space	32

You can use the Chr function to convert the code into a letter e.g. Chr(KeyAscii). If KeyAscii is set to 0 in this event, this makes the program act as if the key was never pressed.

LostFocus

Occurs when the CommandButton loses focus.

MouseDown, MouseUp

Button As Integer, Shift As Integer, X As Single, Y As Single

This event occurs when a mouse button is pressed down or released over a CommandButton.

```
Private Sub Command1_MouseUp(Button As Integer, Shift As Integer, X
As Single, Y As Single)
If Button = vbMiddleButton Then
   MsgBox "The middle button was pressed on the CommandButton"
End If
End Sub
```

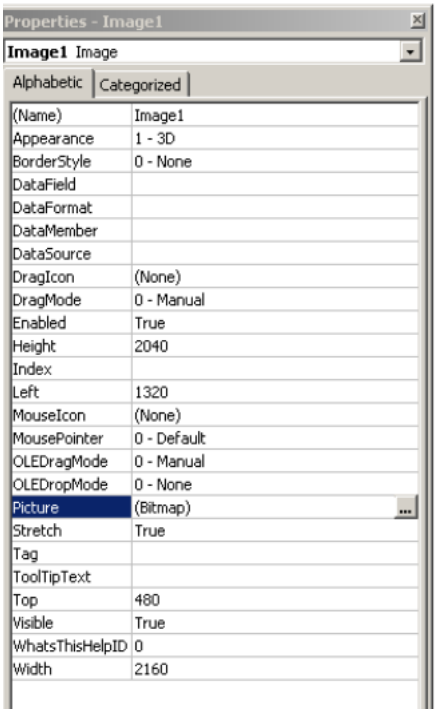
The Button argument returns one of the following possible values:

Constant	Value	Description
vbLeftButton	1	Left mouse button
vbRightButton	2	Right mouse button
vbMiddleButton	4	Middle mouse button

The Shift argument works the in same the same way as in the KeyDown and KeyUp events. The X and Y arguments return the position of the cursor in relation to the CommandButton i.e. the coordinates of the very top-left of the CommandButton are 0, 0.

The MouseDown event occurs before the Click event and the MouseUp event occurs after the Click event.

THE IMAGE & PICTURE BOX CONTROL:



The PictureBox Control:

PictureBox controls are among the most powerful and complex items in the Visual Basic Toolbox window.

These controls are more similar to forms than to other controls.

PictureBox controls support all the properties related to graphic output, including AutoRedraw, ClipControls, HasDC, FontTransparent, CurrentX, CurrentY, and all the Drawxxxx, Fillxxxx, and Scalexxxx properties. PictureBox controls also support all graphic methods, such as Cls, PSet, Point, Line, and Circle and conversion methods, such as ScaleX, ScaleY, TextWidth, and TextHeight.

The Image Control:

Image controls hold pictures.

They can also be used for creating toolboxes

Image controls can be used to display icons or pictures created with the program such as Microsoft paintbrush. They can also hold windows metafiles, JREGs or Gif files.

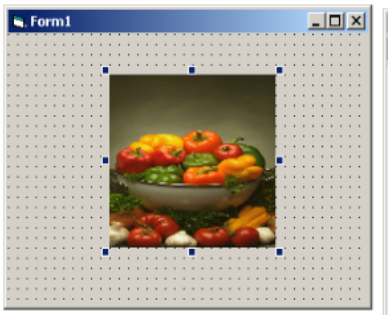


Image controls are far less complex than Picture Box controls. They don't support graphical methods or the AutoRedraw and the Clip Controls properties, and they can't work as containers, just to hint at their biggest limitations. Nevertheless, you should always strive to use Image controls instead of Picture Box controls because they load faster and consume less memory and system resources. Remember that Image controls are windowless objects that are actually managed by Visual Basic without creating a Windows object. Image controls can load bitmaps and JPEG and GIF images.

It is used to display a graphic. An Image control can display a graphic from a bitmap, Icon, or metafile, as well as enhanced metafile, JPEG, or GIF files.

Property	Description	Data Value
Left, Top	This property determines the distance between the Image Control and the left edge and top of the container (form), respectively.	
BorderStyle	Returns/sets the border style of the object	• No border (setting =0) • (Default) Fixed Single border (Setting=1)
Stretch	Returns or sets a value indicating whether a graphic resizes to fit the size of an Image control	• True-The graphic resizes to fit the control. • False-(Default) The control resizes to fit the graphic.
Picture	Returns or sets a graphic to be displayed in a control.	Picture(Bitmap, metafile, Icon etc)

TEXTBOXES:

- The text boxes are the primary method for accepting input and displaying output in Visual Basic.

Standard Properties of Text Boxes:

- It consists of 50 properties. The standard properties are:
  - ➤ Name
  - ➤ Font
  - ➤ Enabled
  - ➤ Visible
  - ➤ ForeColor and BackColor

- **Name:** This property indicates the name of the text box is used only for the code. (The Microsoft's Prefix for the Name property of a text box is txt).
- **Font:** To set the font properties via the Font dialog box available from the Properties window, but the user can only use one font per text box.
- **Enabled:** This property affects whether the textbox will respond to events. If the text box is displayed, the user cannot enter text inside it. It also grayed. It consists of two settings. The default value is True.
- **Visible:** This property consists of two values. The default value is True. If it is change to False the text box will disappear.
- **ForeColor and BackColor:** It ForeColor affects the color of the text that is displayed. The BackColor affects the rest of the text box. Both of these can be set independently of the surrounding container. It is easiest to set them using the color palette from the Properties window.

Some Special Properties for Text Boxes:

- **Text:** The default value for this property is set to Text1, Text2 and so on. If you want a Text box to be empty when the application starts, select the Text property and blank out the original setting.
- **Alignment:** This property controls how text is displayed. The default value is 0, which indicates the text is left-aligned. Use a value of 1 and text is right-aligned. Use a value of 2 and text is centered.
- **MultiLine**: This property determines whether a text box can accept more than one line of text when the user runs the application, and it is usually combined with resetting the value of the Scrollbars property.
- VB automatically word-wraps when a user types more than one line of information into a multiline text box. The multiline text boxes are the usual method for displaying large amounts of text in Visual Basic. The limit for a multiline text box is approximately 32,000 characters.
- **ScrollBars:** This property determines whether a text box has horizontal or vertical scrollbars. Without scrollbars, it becomes much harder for the user to move

through the data contained in the text box, thus making editing the information that much more difficult.

- The four possible settings for the ScrollBars property are:
  The default value is 0.
  0→ The text box lacks both vertical and horizontal scrollbars.
  1→The text box has horizontal scrollbars only.
  2→The text box has vertical scrollbars only.
  3→The text box has both horizontal and vertical scroll bars.
- **BorderStyle:** There are only two possible settings for the BorderStyle property for a text box. The default value is 1, which gives the single-width border, called a fixed single. If you change the value of this property to 0, the border disappears.
- **MaxLength:** This property determines the maximum number of characters the text box will accept. The default value is 0. Any setting other than 0 will limit the user's ability to enter data into that text box to that number of characters.
- **PasswordChar:** The asterisk (*) symbol used as a password character. This feature is combined with the MaxLength property to add a password feature to your programs.Locked:
- This True/False property prevents users from changing the contents of the text box. Users can scroll highlight text, but won't be able to change it.

Event Procedures for Text Boxes:

- The text boxes can recognize 23 events. VB monitors the text box and calls the Change event procedure whenever a user makes any changes in the text box. One of the most common uses of the Change event procedure is to warn people that they should not be entering data in specific text box at this moment.

LABELS:

It is used to display information about the controls present near by. It is used to identify a textbox or other control by describing its content.

Properties for Labels

1. Caption Property:

To display text on a label control, set its **Caption** property.

2. BackColor Property

Sets the back color of the label

3. ForeColor Property

Sets the fore color (color of the text that is displayed) of the label

4. Visible Property

Makes the labels appear and disappear by setting the value true or false.

5. Alignment

Return/sets the alignment of a control's text. By setting the values as

- 0-(default) left-aligned
- 1-right-aligned
- 2-center

6. BorderStyle

Returns/sets the border style of the object. By setting the values as

- No border (setting =0)
- (Default) Fixed Single border (setting =1)

7. BackStyle

Determines whether the label is transparent or opaque.

8. Autosize

Determines whether a control is automatically resized to display its entire contents. By setting the values as

- True- resized according to the contents
- False (default)-does not resizes.

9. WordWrap

Returns/sets a value that determines whether a control expands to fit the text in its caption. By setting the values as

- True- wraps the contents

- False (default)-does not wraps the contents.

10. Usemnemonic

Returns/sets a value that specifies whether an ampersand (&) included in the text of the caption property of the label control defines an access key. By setting the values as

- True-(Default) Any ampersand appearing in the text of the caption property causes the character following the ampersand to become an access key. The ampersand itself is not displayed in the interface of the label control
- False-Any ampersand appearing in the text of the caption property is displayed as an ampersand in the interface of the Label control.

Some Event Procedures for Labels:

Click event- it occurs on clicking the label.

Change event- it occurs when the contents of a control have changed.

DblClick event- it occurs on double clicking the label.

MouseDown-it occurs when the user presses the mouse button.

MouseUp events-it occurs when the user releases the mouse button.

MouseMove event-it occurs when the mouse is moved towards the label.

MESSAGE BOXES:

- The message boxes display information in a dialog box superimposed on the form. They wait for the user to choose a button before returning to the application.
- The users cannot switch to another form in your application as long as VB is displaying a message box.
- It should be used for short messages or to give transient feedback. It can hold a maximum of 1,024 characters, and VB automatically breaks the lines at the right side of the dialog box.
- Syntax for message box

```
MsgBox  MessageInBox,TypeOfBox,TitleOfBox
```

- Example:

The application display a message box when the user moves the focus away from a text box before placing information inside it.

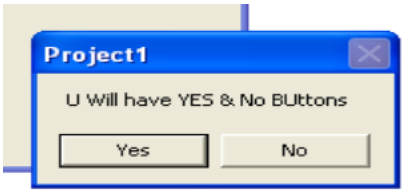
```
Private Sub Text1_LostFocus()
    MsgBox "Enter the Value"
End Sub
```

- The three different groups of built-in integer constants to specify the kind of message box.

Symbolic Constant	Value	Meaning
vbOKOnly	0	Display OK Button Only
vbOkCancel	1	Display OK and Cancel buttons
vbAbortRetryIgnore	2	Display Abort, Retry, and Ignore buttons
vbYesNoCancel	3	Display Yes, No, and Cancel buttons
vbYesNo	4	Display Yes and No buttons
vbRetryCancel	5	Display Retry and Cancel buttons
vbCritical	16	Display Critical Message icon
vbQuestion	32	Display Warning Query icon
vbExclamation	48	Display Warning Message icon
vbInformation	64	Display Information Message icon

- Consider the following statement

```
MsgBox  " U Will have Yes and No Buttons",vbYesNo
```



- The next group of number controls which button is the default button for the box.

Symbolic Constant	Value	Meaning
vbDefaultButton1	0	First button is default
vbDefaultButton2	256	Second button is default
vbDefaultButton3	512	Third button is default

Example:

```
MsgBox "Welcome to WELCOME
```

college",vbOkCancel+vbExclamation+vbDefaultButton2,

"Test Message Box"

- This box contains an exclamation mark icon with Ok and Cancel buttons, and the second button, Cancel, would be the default button for this form. The title bar of the message box would show the title.

DIALOGBOXES:

A dialog box is special window displayed by the system or application to request a response from or provide information to the user.

Types of Dialog boxes

1. **Pre-defined dialog boxes**-Created using Inputbox () and Msgbox () function.
2. **Customized dialog boxes**-Created using a standard form or by customizing an existing dialog box.
3. **Standard dialog box**-file Open dialog box, Print dialog box created using the Common dialog.

Unit II

VARIABLES IN VISUAL BASIC

Variable:

The various values used during computation are stored in 'variables'.

For Example:

TotalBillAmount-it holds the total invoice value

ItemRate- it holds the rate of the item.

The code will look like this:

TotalBillAmount = TotalBillAmount + ItemRate

An application is a piece of code acting on data and data is manipulated by transferring

it into variables. We can manipulate the data in the table without explicitly using variables.

Declaring Variables:

To write programs, we need some variables. Before the variable is used , we must inform the program that this particular word is a variable. The process or method of providing this information is called declaring a variable. Variables are used for storing values. It is declared using DIM statement.

Syntax:

Dim variablename[As type]

Example: Dim TotalBillAmount as interger

A variable has to have a name. There are some naming conventions or rules. The name of the variable:

- Must begin with an alphabet,
- Must not have an embedded period or a special character.
- Must not exceed 255 characters.
- Must be unique within the same scope.

DATA TYPES:

Variables have a name and a data type. The data type of a variable determines the bits/bytes representing those values are stored in the computer's memory. While declaring a variable a data type also assigned for it. All variables have a data type that determines the kind of data they can store.

Visual basic has a number of variable types to deal with various programming requirements. One needs to use different types of variables for different requirements in order to optimize speed, and memory requirements.

Data types apply to other things besides variables. When we assign a value to a property, that value has a data type, arguments to functions to functions also have data types. Anything in visual basic that involves data also involves data types. We can also declare arrays of any of the fundamental types.

Variables and their Purposes:

Integer : A numeric variable holds numeric values –32,768 to 32,767.

Long : A numeric variable holds a wider range of integers than integer.

(Long Integer) -2,147,483,648 to 2,147,483,647

Single : A numeric variable which holds numbers with decimal places.

-3.402823E38 to -1.401298E-45 For negative values.

1.401298E to 3.402823E38 For positive values.

Double : A numeric variable with a wider range than single.

1.79769313486232E308 to –4.9406564584124E-324 for negative values

4.9406564584124E-324 to 1.79769313486232E308 for positive values.

Currency :For holding monetary values.

-922,337,203,685,477.5808 to 922,337,203,685,477.5807.

String : For holding text or string values.

0 to approximately 2 billion for variable length.

1 to approximately 65,400 for fixed length.

Byte :A numeric variable, holding less than the value 255, 0 to 255.

Boolean : Fro holding True or False values.

Date : Fro holding Date values inclusive of and between January 1, 100 to December 31,9999.

Object : For holding references to objects in Visual Basic and other applications or any Object reference.

User-defined : Number required by elements. The range of each element is the same as the

(using Type) range of its data type.

Variant : A general-purpose variable that can hold most other types of variables values (with number). Any numeric value up to the range of a double. With Character values, it has the same range as for a variable-length string.

Integer:

- This data type is used to store whole numbers and cannot be used in calculations where decimals or fractions are involved. They can store large numbers.

- It occupies only **two bytes** of memory and is quite fast when used in calculations.

Long:

- It is bigger than Integer data type and can hold much larger values.
- It occupies twice as much space as the Integer.
- It must be used in large calculations and it is slower then Integer.

Single:

- It is equivalent to Floating-point numbers. It can store fractions and provide precision to high level.
- It occupies **4 bytes** of memory space.

Double:

- This solves the problem of precision that the single data type lacks.
- It occupies **8 bytes** of memory space and used in very high precision in need.
- It is slower than integer data type and used where accuracy is needed in calculations.

Currency:

- It is used for holding values related to item rates, payroll details and other financial functions. This data type should not be used for the need of extreme accuracy beyond the fourth decimal point.

Example: Foreign exchange interest rates for very very large values.

Boolean:

- This data type accepts only **True** or **False** values.
- Since the default value for all numeric data types is zero, the default value for a Boolean data type is also zero.
- Zero value is interpreted as False and a non-zero value is interpreted as True.
- The VB keywords True and False can be used to assign values to the Boolean data type.

Date:

- This variable holds **date** and **time** data.
- It can hold **date** from **January 1 100 to December 31 9999,** and **time** from **00.00.00(midnight) to 23.59.59**(one second before midnight) in one second increments.

- It occupies **8 bytes** of memory. The data is displayed as per the settings in the computer.
- The date can be stored in British Format, American Format, or any other format that is available or the Regional Settings on the control panel.
- When other numeric data types are converted to date, values to the left of the decimal represent date information, right of the decimal represent time. Midnight is 0, midday is 0.5. Negative whole numbers represent dates before December 30, 1899.

The String Data Type:

- The most commonly used data type is the String. Every application has details like Name, Address, zip code, Phone number etc. All these are strings, although some of them will consists of numeric data.
- Declare the variable with this data type it will always contain a string and never a numeric value.
- The variable can be declared as **'variable-length'** string or a **'fixed-length'** string.
- By **default**, a string variable is **variable-length** string, the string grows or shrinks by assign new data to it.

Example

```
Dim ItemName As String * 30  ' a fixed-length string'
Dim ItemDescription As String   ' a variable-length string'
```

- The string ItemName will always be 30 characters long. If we assign a string of fewer than 30 characters, ItemName will be padded with enough trailing spaces to total 30 characters. If the string is too long for a fixed-length string, it truncates the characters.
- To remove trailing spaces while working with fixed-length strings, the functions like Trim and Rtrim are used.

Object Data Type:

- In Visual Basic, forms, controls, procedures and recordset, are all considered as Objects.
- All programming activity resolves around these objects. Since VB is very much an object base programming language, it is very natural to use Object data types.

- An object variable refers to an object within the application or in some other application. This object can be a Textbox or a Form or a Database.
- A variable declared as an Object is one that can subsequently be assigned to refer to any actual object recognized by the application.
- **Example:**
    - Dim objDb As database
    - Set objDb = OpenDatabase("c:\SISI\EIS.mdb")
- A variable declared as an object occupies **4 bytes** of storage.

The Variant Data Type:

A variant data type is a variable that can change its type freely. It can accept text, numeric data or byte data easily. The variable is given the Variant data type by default. When the variable is assigned variant data type, the conversion of data type takes place in visual basic automatically.

Example:

```
Dim VarValue                    ' Variant by default.
VarValue = "100"                ' VarValue contains "100" (a string)
VarValue = VarValue  - 70       'VarValue now contains numeric value 30.
                                'the conversion is done automatically
VarValue = VarValue & "+"       ' Varvalue now contains the string "30+"
```

The variant data type is used where the data type of files are not known. While performing calculations, make sure that the value contained in the variant is a number. Otherwise it will throw an error.

Example: we cannot perform any mathematical operations or a function on a variant that does not contain a number even though it contains a numeric character like 30+.

It is a good idea to determine if a variant variable contain a value that can be used as a number. The **IsNumeric** function performs this task. For the concatenation operation make sure that the values in the variants are strings. It is better to use the **"&"** operator rather than **"+"** operator.

If both of the variants contains numbers, the + operator performs addition. If both of the variants contains strings, then the + operator performs string concatenation.

If one of the values is numeric and the other is a string, then it will create a problem. Visual basic first tempts to convert the string into a number. If the conversion is successful, the + operator adds the two values, if unsuccessful, it generates a Type mismatch error.

Special Features of Variant Data Type:

It contains special values that other variables cannot contain. These values are

- The Null Value
- The Empty Value
- The Error Value

The Null Value

Null is commonly used in database applications to indicate unknown or missing data. Assigning Null to a variant variable does not cause an error, otherwise it will cause an error.

The Null keyword is used for assigning the variable. **Example:** Z = Null.

The **IsNull** function is used to test if a Variant variable contains Null.

Example:
```
If IsNull(x) Then
    Debug.prin
End If
```

We can return Null from any function procedure with a Variant return value. Variables are not set to Null unless we explicitly assign Null to them.

The Error Value

In a variant, Error is a special value used to indicate that an error condition has occurred in a procedure. An error value is created by converting a real number using the **CVErr** function. However, unlike other kinds of errors, normal application–level error handling does not occur.

The Empty Value

A variant has the Empty value before it is assigned a value. The Empty value is a special value different from 0, a zero-length string (“”), or the Null value. We can test for the Empty value with the **IsEmpty** function.

Example: If IsEmpty(x) Then statements.

A variant can be assigned the Empty value using the Empty keyword. When a

variant contains the Empty value, we can use it in expression, where it is treated as either 0 or a zero-length string, depending on the expression.

The Empty value disappears as soon as any value is assigned to a variant. A variant always takes up 16 bytes, regardless of the type of data stored in it. Objects, strings, and arrays are not physically stored in the variant.

Four bytes of the variant are used to hold either an object reference or a pointer to the string or array. The actual data is stored elsewhere.

The Scope of a Variable:

Visual Basic is event driven. Code is written for each control and individually for each of that control, while variables are declared for each event or procedure. In order to avoid locking up the computer's memory and causing confusion, variables are limited with scope. The scope of a variable is the range from which the variable can be referenced – a procedure, a form, and so on.

The value of a variable in one procedure cannot be accessed from another procedure. The value of a variable is local to that procedure. Variables declared with the Dim statement within a procedure exist only as long as the procedure is executing.

When the procedure finishes, the value of the variable disappears. These characteristics allow using the same variable names in different procedures without confusion. The scope of the variable is determined by the way it has been declared. It can be **procedure-level** variable or a **module-level** variable.

Example:

MODULE1

Public x as Integer

FORM1

Public y as Integer

Private Sub Command_click()

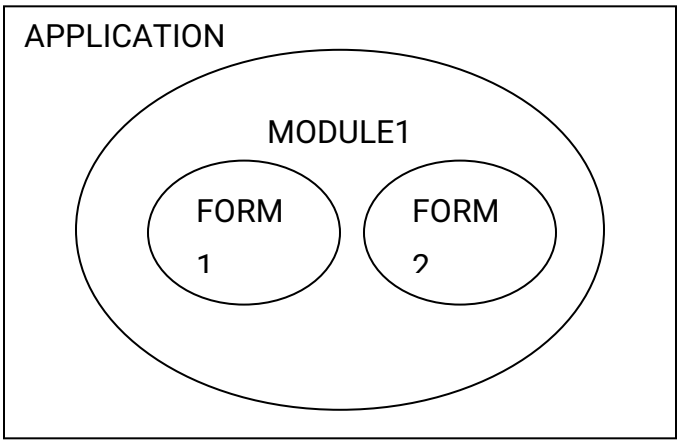
Dim x as Integer

X = 30

Module1.x = 10

Form1.y = 20

X = 30



```
    Msgbox "Module1.x = " & str(Module1.x)
    Msgbox "Form1.y = "& str(Form1.y)
    Msgbox "X = " & str(x)
    Load Form2
End Sub
Private Sub Form_Load( )
    Msgbox "Module1.x = " & str(Module1.x)
    Msgbox "Form1.y = "& str(Form1.y)
    Msgbox "X = " & str(x)
End Sub
```

Procedure level variables are recognized only in the procedure in which they are declared. These are also known as local variables and also declared using the DIM keyword.

Example:

```
Dim Amount As Integer
```

Local variables are available only to that procedure where they have been declared and another procedure cannot alter or affect the value in this variable. Any change made by a procedure will be restricted to its variables, although two procedures can have variables with the same name. Variables declared using the Dim keyword exists only as long as the procedure is executing.

Static:

Values in local variables declared with Ststic exist the entire time the application is running. Variables are declared as static using the 'Ststic' keyword.

Example:

```
Static count As Integer
```

Variables declared using the **static** keyword exist as long as the application is running, and are usually used to update counters. Since all local variables cease to exist when the procedure terminates, the variables will be initialized when the procedure is called again.

For Example:

To count the total number of customers for a day, this counter is declared in the procedure that prints the invoice and if we exit this procedure for some reason, and call it again, the total number of customers for the day will start from 0. In order to avoid this problem, the counter can be declared as a Static variable. In this case the value of the counter will remain in the memory even if the procedure terminates. When the procedure is invoked again the last value will be available to it.

Advantage: This variable can be accessed only by this procedure and not by any other procedure.

Module Level Variables:

Variables declared as module-level variables will be available to all procedures within that module. They will not be available to procedures in other modules. A module level variable is declared using the **Private** keyword, in the declaration section of the module.

Example: Private intCount As Integer

The declaration must be in the declaration section of the module. In order to make available to all other modules, use the Public keyword. Public variables cannot be declared in a procedure. They can only on the declaration section of a module.

Public intTemp As Integer

Scope	Private	Public
Procedure-level	Variables are private to the procedure in which they appear.	Not applicable. You cannot declare public variables within a procedure.
Module-level	Variables are private to the module in which they appear.	Variables are available to all modules.

Public and Private Variables:

If the variables are local to their procedures there will be no problem. If we have public variable and a private variable with the same name then they will need to be referenced with some caution. This is very similar to the way filenames are handled in MSDOS and UNIX, using the relative pathname and the absolute pathname.

Example:

Public x As Integer

X=15 ‘ the assignment is done later, and in a procedure we have decalred

Dim x As Integer

X=10

Assigning the variable a in procedure will give the value “10”.

Accessing x outside the procedure will give the value “15”. If we have to access the variable in the Module-level from the above procedure we will have to refer to it as follows

Debug. print Module1.x

This will display the value “15”. In this case the public variable is referred to by its full name. The local variable will always be accessed in preference to the public variable. This is called **‘Shadowing’**.

Declaring Variable

The DIM statement is used to declare the variable and other methods are also available. By declaring a variable in the declaration section of a form, standard, or class module, rather than within a procedure, the variable will be available to all the procedures in the module.

By declaring a variable using the **Public** keyword, it will be available throughout the application. Declaring a procedure-level variable using the **static** keyword preserves its value even when a procedure ends.

We can use a variable without first declaration. Visual Basic automatically creates a variable with that name. This is called **Implicit Declaration.** But it gives some problems.

For example:

```
Private Sub Cmdsave_Click( )
        Intvoice = Intvoice +  Intsale
        Print Intvice
End Sub
```

The function above will not result in an error but the value will print in the form as 0. this is because when VB comes across “Intvice” it automatically creates a new variable with that name, as it does not understand the spelling mistake. Implicit declarations are not for serious programmers or applications. They are at best useful

for simple test routines.

One way of correcting this habit is to enforce variables declarations before they are used and this can be done using the **"Option Explicit"** statement. This will avoid the problem of misnaming variables and VB will warn you whenever it encounters a name not declared explicitly as a variable.

Place the **Option Explicit** statement in the declaration section of a Form, Module or Class.

The other method is : From the **Tools** menu, choose **Options**, click the **Editor** tab and check the **Require Variable Declaration** option. This will automatically insert the **Option Explicit** statement in any new module. It operates on a per-module basis and it must be placed in the declaration section of every form, standard and class module to enforce explicit variable declarations. It used to catch these kinds of errors.

Constants:

The value, which does not change during the execution of the program, is called constant. While performing calculations, we need to work with figures that are constant.

Example:

To find circumference of a circle 2*(3.14)*r where **pi** = 3.14 , r is radius of circle. The value of pi can be represented as **3.14**. This number can be assigned to a **Constant** and can be used in all calculations.

Constants store values like variables, but as the name implies, those values remain constant throughout the execution of an application. Using constants can make the code more readable by providing meaningful names instead of numbers. There are number of built-in constants in visual Basic, it can also created by own.

Creating own Constants

Syntax:

[Public | Private] Const constantname [As type] = expression

- constantname should be valid name.
- As type is the Data type.
- Expression is the numeric or string value that has to be assigned to the constant.

Rules:

- Must begin with an alphabet,

- Must not have an embedded period or a special character.
- Must not exceed 255 characters.
- Must be unique within the same scope.

Example:

```
Const conpi = 3.14
Dim IntRad = Val(Text1.Text)
Circum = 2 * conpi * IntRad
```

Other than user defined constants there are many intrinsic or system-defined constants provided by applications and controls. Visual Basic constants are listed in the Visual Basic (VB) and **Visual Basic for Applications (VBA)** object libraries in the **object browser.**

Scope of Constant

By declaring a Constant in the declarations section of a form, standard, or class module, rather than within a procedure, the Constant will available to all the procedures in the module. By declaring a constant using the **Public** keyword, it is available throughout the application. Declaring a constant in a procedure will be available to that procedure only.

Circular References

Constants can be defined with reference to other constants.

Example:

```
Public Const conA = conB * 1.414
Public Const conB = conA * 2
```

Since both the constants are available throughout the application, Visual Basic will generate an error. This method of defining constants where each is defined in terms of the other is called **Circular Reference.** Visual Basic will not proceed with the execution of the program till this circular reference is resolved.

A Word of Caution on Variables:

1. A variable in the module cannot have the same name as any procedures or types defined in the module.
2. A local variable can have the same name as public procedures, types, or

variables defined in other modules. If this variable is accessed from another module, it must be qualified with the module.

Converting Data Types:

Visual Basic provides functions to convert values into data types that are needed. They are as follows:

Conversion Function	Converts an Expression to
Cbool	Boolean
Cbyte	Byte
Ccur	Currency
Cdate	Date
CDbl	Double
Cint	Integer
CLng	Long
CSng	Single
CStr	String
Cvar	Variant
CVErr	Error

Note: Values passed to a conversion function must be valid for the destination data type or an error will occur. For example, to convert a Long to an Integer, the Long must be within the valid range for the Integer data type.

Arrays:

An Array is a set of similar items. All items in an array have the same name and are identified by an index. Arrays allows to refer to a series of variables by the same name and to use a number (an index) to tell them apart.

Example:

```
Dim num(10) as integer
Dim x(10 to 20) as integer
```

Dim x(5) as variant

Types of Array:

One Dimensional Array

Fixed-Size Array

Multidimensional Array

Dynamic Array

One Dimensional Array:

It consists of single dimension.

Syntax:

Dim Varname [([subscripts])] as [New] type [,varname…]

Example: 1

Dim num(10) as integer

In this case, 'num' is set of 11 integers. Num(0) is the first element of the array. Num(10) is the eleventh and the last element of the array.

Example: 2

To compute monthly sales for an organization. Sales figures for each month have to be calculated. This means that we need to have at least 12 variables. SaleJan, SaleFeb, etc. when it is don in one variable it is difficult processing. Therefore using an array, we can declare the variable as follows.

Dim Saleval(11) as Long

Since there are 12 months, Saleval(0) will hold the sales figures of the first month and Saleval(5) will hold the sales figures of the sixth month an so forth.

Fixed-size Arrays:

In the Fixed-size array, the total number of items must be already known.

Declaring Fixed-size Arrays:

Fixed-size Arrays are declared just like declaring the variables. The scope of the array will depend upon the method of declaration.

1. To create a local array, use the Private statement ina procedure to declare the array.

   Dim counters (10) As Integer

2. To create a module-level array, use the Private statement in the declaration

section of a module to declare the array.

Private counters(10) As Integer

3. To create a public array, use the Public statement in the declaration section of a Form.

Public counters(10) As Integer

In the case of fixed-size arrays it is compulsory to enter the upper bound of the array in the parenthesis. The upper bound is the upper limit for the size of the array. To specify the lower bound of an array, provide it explicitly using the **To** keyword.

Dim counter(1 To 10) As Integer

In the above statement, the index numbers of counters range from 1 to 10. The **Lbound** is a function that returns the lower bound of an array. The **Ubound** returns the upper bound of an array.

Example:

Dim sum(20)

X= Lbound(sum)

Debug.print X

This will display 0 in the debug window.

Multi-dimensional Arrays:

Arrays can have more than one dimension. A table of data will be represented by a multidimensional array.

Example:

To record the sales figures for twelve months for three of the departments in the organization, then the array can be declared as follows.

Dim Salecal(11,2) As Integer

Here the subscripts 11 indicates the months and the subscripts2 indicates the departments. This is a two-dimensional array. We can also have three-dimensional arrays. To record the sales of five products in three departments for twelve months then we need a three-dimensional arrays.

Dim Saleval(11,2,4) As Integer

Where the subscript 11 indicates months, the subscript2 indicates the three departments and subscript3 indicates the 5 products. Multidimensional array takes up a

lot of space.

Dynamic Arrays:

Dynamic arrays are used when the number of elements for an array is not known. For example, for reading a string into an array, we may want to have the capability of changing the size of the array at run time.

A dynamic array can be resized at any time and this helps to manage memory efficiently. For example, we can use a large array for a short time and then free memory to the system when we are no longer using the array. We can increase the size of the array after having declared a smaller array.

The **ReDim** is used in conjunction with the Dim statement while declaring these arrays. The alternative is to declare an array with the largest possible size and then ignore array elements that are not needed. This will result in the operating environment to run low on memory.

Declaring a Dynamic Array:

1. Declare the array aas dynamic by giving it an empty dimension list.

   Dim sum()

2. Use the ReDim ststement to allocate the actual number of elements.

   ReDim sum(11,4)

Note:

ReDim is an executable statement and can appear only in a procedure. Unlike the Dim and Static statements, it makes the application carry out an action at run time. We can use ReDim each time we want to change the upper or lower bounds of a dimension. We cannot however change the number of dimensions.

The rules of syntax for dynamic arrays are same as they are for fixed sized arrays. The scooping rules are also the same as they are for other variables.

Example:

Dim salval () As Integer

“this statement will create an open-ended array”

ReDim salval(12,6)

This will create a two-dimension array. The values for the subscripts of the array

can be passed using variables.

Example:

```
Dim salval() As Integer
Dim Months As Integer
Dim Depts As Integer
Months = 12
Depts = 6
ReDim salval(Months, Depts)
```

In this way, if the number of departments vary ;ater on the arrays and the program need not be modified.

The Preserve Keyword:

Whenever the ReDim statement is used the previous array and its contents are destroyed. Visual basic resets the values to the Empty value(for variant arrays), to zero(for numeric arrays), to a zero-length string(for string arrays), or nothing (for arrays of objects). This is useful to prepare the array for new data, or to shrink the size of the array to take up minimal memory.

To expand or increase the size of the array ReDim by itself is not good news. In order to allow the array to '**grow**', the **Preserve** keyword is used. The statement

```
ReDim Preserve Salval(12,9)
```

Will not destroy the data that has already been entered. It only add the value for the second dimension. In this case more columns are added to the table.

In the case of the single dimension dynamic array, we can enlarge an array by one element without losing the values of the existing elements using the **UBound** function to refer to the upper bound. The **UBound** function can be used to get the bound of the array.

```
ReDim Preserve DynSalval(Ubound(DynSalval)+1)
```

More on ReDim:

1. Only the upper bound of the last dimension in a multidimensional array can be changed while using the Preserve keyword, if any of the other dimension, or lower bound is changed, a run-time error will occur.

    Dim sum(11,2,4) As Integer

    It can be resized using the Preserve keyword as follows

    ReDim Preserve sum(11,2,6) As Integer

The following statement will return an error

    ReDim Preserve sum(15,2,6) As Integer

2. Date type of an array cannot be changed using the **ReDim** function except the **variant** data type. If the array is contained in the variant data type, the type of the element is changed using an **As** type clause unless you are using the **Preserve** keyword. The **preserve** keyword **does not allow** the data type to change.
3. if the size of the array is reduced, then the data in the eliminated element will be lost.
4. The **ReDim** statement acts as a declarative statement. If the variable it declares does not exist at the module or procedure level, a new variable will be created. If another variable with the same name is created later, ReDim will refer to the later variable and will not cause a compilation error, even if **Option Explicit** is in effect.
5. **ReDim** should not be used as a declarative statement, but simply for redimensioning arrays.

step 1 : vb enter

step 2 : form1 (save project, form)

step 3 : control, click, drag into the form

(text1

text2

text3

Button 1)

step 4: button1 double click

step 5 : enter code window

step 6 :

```
private sub buttonl_click()
        text3.text = (1/2 * Text1.text *Text2.text)
End sub
```