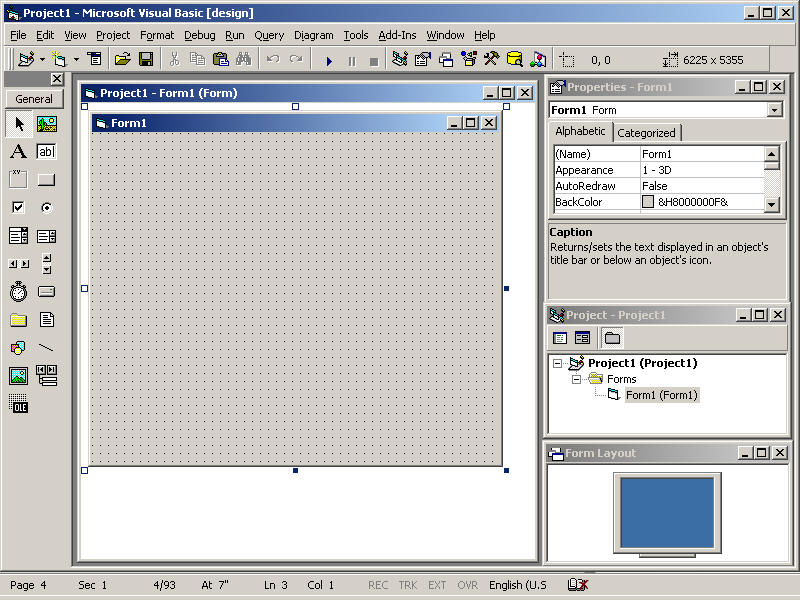
## Chapter: 1 Introduction to Visual Basic

Tool Bar

Menu Bar

Title Bar



Project Explorer

Tool Box

Form Layout Window

Properties

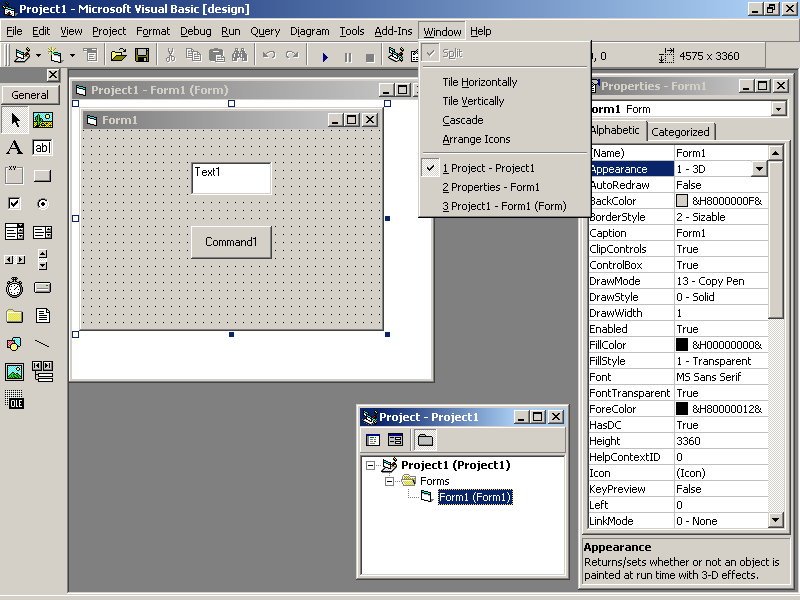
Windows

Initial Form Windows

**The different parts in the Visual Basic Environment**

**The Windows Menu**

The Windows menu is common to most Windows applications. It lets you control how the windows in your screen appear.



You can arrange your windows horizontally, vertically or in a cascade order. The window menu also displays the list of currently opened files. The Split option lets you split the Code window in two halves. This facilitates working easier, when debugging complex programs.

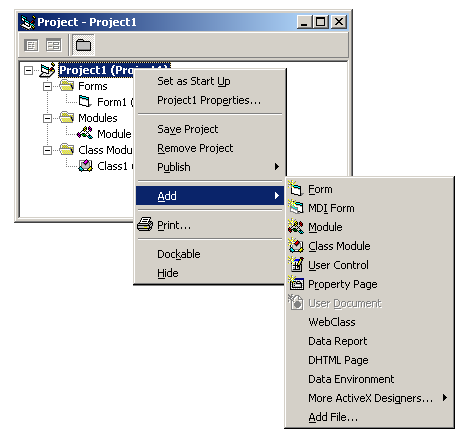
**The Project Explorer**

The Project explorer window is the window you use to manage, the files for any project. The Window is displayed in the upper right corner of the main working area of the VB IDE. It displays all the Forms, Modules, user controls and property pages that are currently included in the project.

**The Project Explorer Window Layout**

The three icons in the upper left corner of the Project Explorer Window are the view code icon, View object icon, and the Toggle Folders icon. Clicking the View Code icon displays the code in the code window for the highlighted object. Clicking the View object icon displays the Form associated with the object in the Form Window. The View object icon is available to those objects that have Form associated with them.

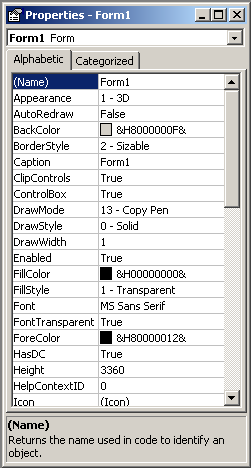
The Project Explorer Window defaults to the folder tree View, which displays all the objects Contained in the current Project in the tree view.



The name of the Project is followed by the form folder, the Modules folder, the class Modules folder, the user controls, and the Property Pages folder. Each folder contains a fist of all the appropriate objects contained in the Project, in alphabetical order. You can change the view from tree to a list of all the objects in alphabetical order regardless of their type by clicking the toggle Folder's icon, You can avail various options provided by Visual Basic like Save Form As, Adding Forms, Projects, Printing etc. by Right Clicking the Project explorer.

**The Properties Window**

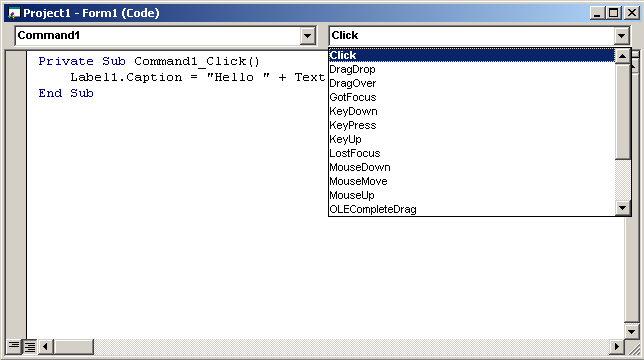
The properties window displays all the properties you can set at design time for any given Forms or Control. You can resize this window according to your preferences. The drop down Combo Box below the title Bar of the Properties Window contains a list of all the controls available on the Form.



The two tab groups below the Combo Box display the properties in Alphabetic or categorized order. The dynamic Text-box that displays the property name provides help on particular property on clicking the text box or pressing FI Key. Short-cut to the property Windows is F4 Key.

**The Code Window**

The code window on the left side of the screen, displays the code associated with the current form or module. This is an important window where you can add or modify the code. The code window contains two drop down combo Boxes in the Upper- left and right comer of the code window. The two combo boxes are called object dropdown combo box and the procedure combo box. OBJECT DROP DOWN COMBO BOX contains all the objects available on the form.



EVENT

PROCEDURE

COMBOBOX

OBJECT COMBO BOX

PROCEDURE COMBO BOX contains the name of all the event procedures available to the selected object in the object combo box. As soon as you select an object and an event Procedure,

Visual Basic automatically inserts the name of the procedure as.

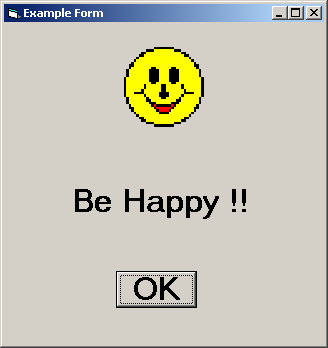
Private Sub CommaNd1\_c1ick ()

End Sub

Code Window has one more section, the general section where you can write general procedures and functions. Code window also contains the horizontal and the vertical scroll bars which let you view the code more easily.

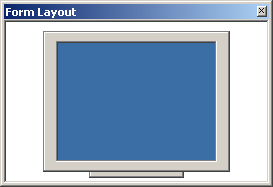
**The Form Window**

The form window is displayed whenever you click the view Object Button of the Project Explorer Window with a form selected. This window displays the form as well as the controls that you've placed on the Form. You can resize, move, cut, copy, delete or place controls on the Form within this window.



**The Form Layout Window**

The Form Layout window is designed to give you more control over the screen positioning of Form.



The Form Layout window is displayed on the right side of the screen. An icon representing the Computer Screen is shown in the main portion of the Form Layout window. You can select the Form you want to move by clicking the Form with the left mouse button, by holding the button down, you can move the form wherever you want on screen.

**The Debug Windows**

Debugging programs in Visual Basic is very easy. Because Visual Basic Integrated developed Environment provides three Debug windows:

1. Immediate Debug Window

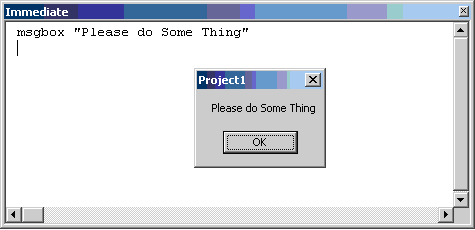
2. Local Debug Window

3. Watch Debug Window

Using these windows will be discussed in later chapters.

**The Immediate Debug Window**

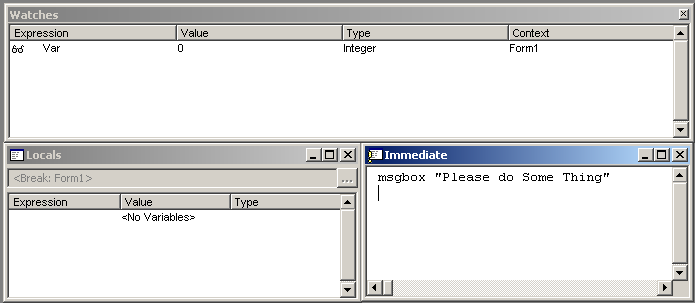
The Immediate Debug Window automatically opens empty in break mode. In this mode you can type and execute a line or code, but can not save.



The above illustration shows the Message box code executed from Immediate Debug window.

**The Local Debug Window**

The Local debug window automatically displays all the current declared variables and their values in the current procedure. When the Local Window is visible, it automatically displays the change. The first component of the Locals window is the calls stack text box and push button displayed below, the title bar.



The text displayed shows the current procedure name. The calls stack push Button displays a dialog box, which lists the procedures in the calls stack. You can use this window to go backward through the calls stack and set break points to aid in the project debugging. Below the calls stack Text Box and push Button are the column headers Expression value and Type. Global variables and variables in other projects are not accessible from the Local window.

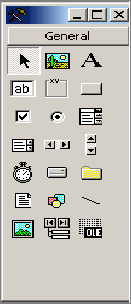
**The Watch Debug Window**

The watch Debug Window displays all the variables that have a watch set on then. It contains a list box with the column Headers expression. You can also drag a selected variable to the immediate window. The expression column displays the watch expression with the watch icon on the left. The value column displays the value of the expression at the time of transition to break mode. You can edit a value and press enter and the value is highlighted. The Edit field remains active if the value is illegal. The type column displays the type of element such as integer. The context column displays the context of the watch expression.

**The Toolbox Window**

The Toolbox contains the tools you use to draw controls on your forms. It displays all the standard, Visual Basic controls plus any custom controls and insert able objects you want to add to your project with the Custom Controls dialog box.

You can display ToolTips for the Toolbox button by selecting the Show ToolTips option in the Environment tab of the Options dialog box. To open the Toolbox, choose Toolbox from the View menu. To close the Toolbox, double-click the Control-menu box.



There are 21 basic tool in the Toolbox:

1.  Pointer:

This is the only item in the Toolbox that doesn't draw a control. Use it to resize or move a control after it's been drawn on a form .When you select the pointer, you can use it only to move or resize controls. When you select any other tool, you can use it only to draw new control.

1.  PictureBox:

Used to display graphical images (either decorative or active) as a container that receives output from graphics methods, or as container for other controls.

1.  Label:

Used for text that you don't want the user to change, such as caption under a graphic.

1.  TextBox :

Used to hold text that the user can either enter or change.

1. Frame:

Used to create a graphical or functional grouping *for* controls. To group controls, draw the Frame first, and then draw controls inside the frame.

1. CommandButton :

Used to create a button the user can choose to carry out a command.

1. CheckBox:

Used to create a box that the user can easily choose to indicate if something is true or false, or to display multiple choices when the user can choose more than one.

1.  OptionButton :

Use in a group of option buttons to display multiple choices from which the user can choose only one.

1. ComboBox:

Use to draw a combination list box and test box. The user can either choose an item from the list or enter a value in the text box.

1.  ListBox:

Use to display a list of items from which the user can choose one. The list can be scrolled if it has more items than can be displayed at one time.

1.  HScrollBar (Horizontal Scroll Bar) :.

Use to provide a graphical tool for quickly navigating through along 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.

1. VScrollBar(VerticalScrollBar):

Used to provide 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 indicate or of speed or quantity.

1.  Timer:

Used to generate timer events at set intervals. This control is invisible at run time.

1.  DriveListBox:

Used to display valid disk drives.

1.  DirListBox (Directory List box) :

Used to display directories and paths.

1. FileListBox:

Used to display a list of files.

1. Shape:

Used to draw a variety of shapes on your, Form at design time. You can choose a rectangle, rounded rectangle, square, rounded square, oval, or circle.

1.  Line:

Used to draw a variety of line styles Qn your form at design time.

1.  Image:

Used to display a graphical image frpm a bitmap, icon, or metafile on your form. Images displayed in any image control can only be decorative and use fewer resources than a Picture Box

1.  Data:

Used to link an embedded object from other applications in your Visual Basic application.

1. OLE Container:

Used to link and embed objects from other application in your Visual Basic application.

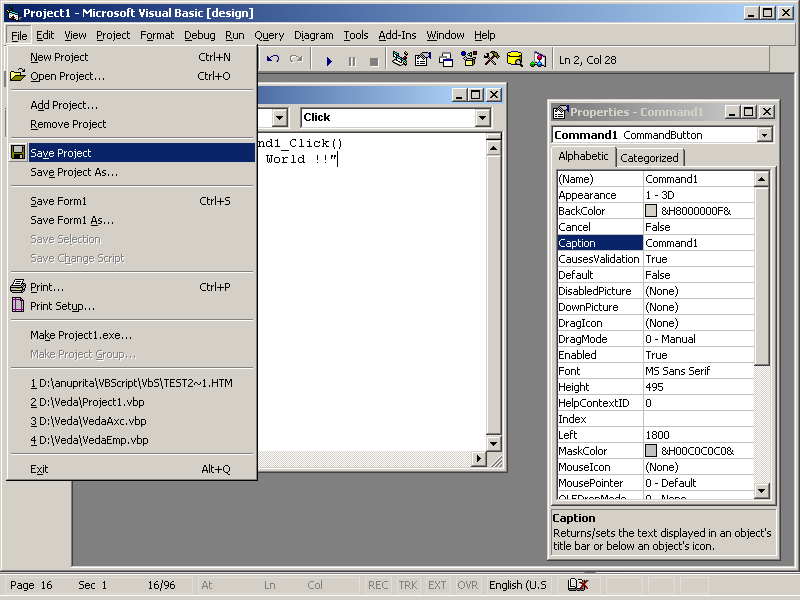
**Saving the Project**

The following screens depict the saving of a project in the default Visual Basic directory 'VB98',

However, you have to save the project in your current working directory. To save the project, carry out the following steps:

1. Select the main menu option 'File'.

This is shown in Figure



2. Select the sub option 'SaveProject'.

This brings up the 'Save File As' dialog as shown in Figure

Visual Basic first prompts us to save the form. Refer Figure

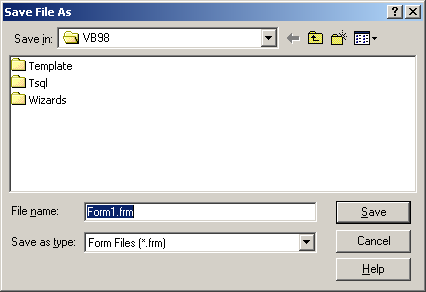


Figure: Save Dialog box

The list of items displayed in the 'Save File As' dialog may vary on different machines.

3. Specify *first\_frm* as the name to save the form against the label 'File Name:' in the Current working directory, The default extension used for forms is 'frm'.

4. Click on the button 'Save'.

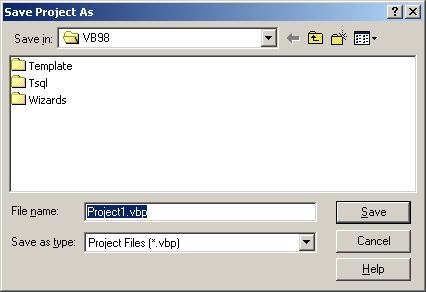
This brings up the 'SaveProjectAs' dialog.

5. Specify *firstprj* as the name for the project in the current working directory.

The default extension for project files is 'vbp'. Refer to Figure.

6. Click on the button 'Save',

Save the Project and form in your current login directory.



In Visual Basic the forms and Projects are stored as separate files.

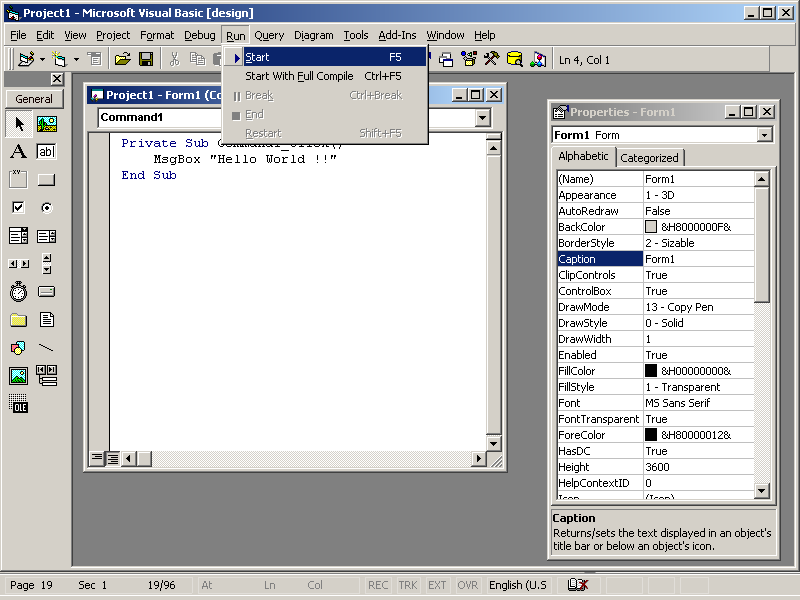
The project has two files. The 'Project file' with the '.vbp' extension contains the information Visual Basic uses for building the Project. The 'Form file' with the '.frm' extension contains information about the form.

**Running the Project**

Let's run the application to see your code in action.

1. Select the menu option 'Run'.

Refer Figure



Run Menu Option

2. Click on the sub option 'Start'.

This runs the application when the application is running on the screen it is as shown in Figure.

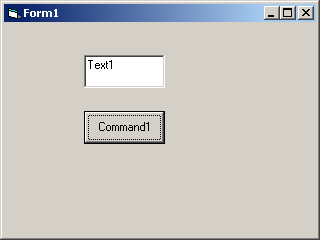


Figure: The Application at runtime

3. Click on the command button *OK.*

The text 'HelloWorld!' is displayed in the textbox. Refer Figure.



4. Click on the application 'Close' button to close the application.

This takes us back to the Visual Basic environment.

**Properties**

Associated with every interface component are certain characteristics. These characteristics are used to describe the component.

For example,

1 Name of a component

2. Height

3. Color

The characteristics of the component are referred to as its properties

The properties of the component are set to the default values when the component is created.

The look of the component is decided by the values assigned to its properties. Thus, the user customizes an interface component provided by the visual development environment using the properties.

For example, a button may have a default label of Command1.This label can be changed to more meaningful text such as Exit to indicate the action it provides.

**Events**

A major part of the interactions between people in everyday life is in the form of events and responses to events.

For example, when the telephone rings we pick up the phone. The ringing of the telephone is the Event and the action of our picking it up is the response to the event. Since computers seek to emulate everyday life, applications also respond to events. An event is any user action directed at the application.

Examples of events that affect an application are:

1. Pressing a key on the keyboard

2. Clicking with the mouse

The interface components have the ability to respond to events. For example. When we click on a button it is recognized as an event by the button. Similarly, a textbox recognizes events like a key press. The ability to recognize events is built into an interface component. The events that an interface

Component recognizes are different for each type of component.

1. An event driven application operates by responding to the user events.

The calculator is a typical example of an event driven application. The application starts up and then waits for the user to perform an action. When the user clicks on one of the buttons, the button recognizes the click and executes some code in response. For example, clicking on a number button will display the number in the text box. Similarly, Clicking on the = button will display the result of the current operation. Events like methods need Additional information in the form of arguments.

Some of the events like methods require additional information in the form of arguments. For example The Click event does not require an argument where as the DragDrop event requires arguments as shown below

Private Sub Command1\_Click ()

Statements

End Sub

Private Sub Command1\_DragDrop (Source As Control, X As Single, Y

As Single)

Statements

End Sub

**Methods**

A method is code that is built into the interface component and can be executed as required. For example, consider a list box. A list box component will provide a method called Add item.

This method is to be executed each time an item is to be added to the list box.

Add item is referred to as the Method. Similarly, most controls provide a method called Move that can be used to reposition the control during the course of the execution. This feature is required very often in applications that provide animation.

Each component can have several methods associated with it. Each method provides a distinct Functionality. In the case of a list box, the method Addltem is used to add an item to the list while Removeltem will delete an item from the list. Unlike properties, some of the methods require additional information in the form of arguments.

For example, the Set Focus method does not have arguments where as the Move method requires arguments to be specified. Consider an application with a command button and a text box placed on it. The fallowing code does not allow the user to leave the text box blank.

The code uses the SetFocus method as follows:

Private Sub Command1\_Click ()

If Text1.Text = " "

Text1.SetFocus

End If

End sub

And to move the button to the top left corner of the screen we use:

Private sub command1\_Click ()

Command1 .Move 0, 0

End Sub

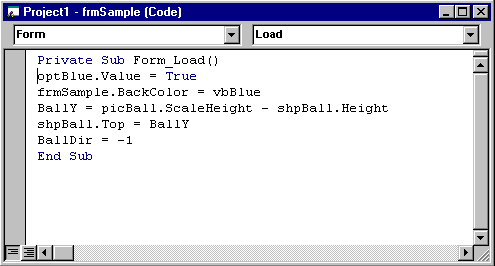
***Chapter*: *2* *Adding* *Code* *and* *Events***

**Code Window**

Let’s look at a new window. Recall Visual Basic is event-driven - when an event is detected, the project goes to the correct **event** **procedure**. Event procedures are used to tell the computer what to do in response to an event. They are where the actual computer programming (using the BASIC language) occurs. We view the event procedures in the **Code** **Window**. There are many ways to display the code window. One way is to use the appropriate button found in the project window. Another is to click **View** on the main menu, then **Code**. Or, as an alternate, press the **F7** function key. Find the code window for the **Sample** project.

Procedure List

Object List



**Naming Conventions**

The most important property for any control is its **Name**. Because of its importance, we address it separately. When we name a control, we want to specify two pieces of information: the **type** of control and the **purpose** of the control. Such naming will make our programming tasks much easier.

In the Visual Basic programming community, a rule has been developed for naming controls. The first three letters of the control name (called a **prefix**) specify the type of control. Some of these prefixes are (we will see more throughout the class):

**Control Prefix**

Form **frm**

Command Button **cmd**

Label **lbl**

Text Box **txt**

Check Box **chk**

Option Button **opt**

After the control name prefix, we choose a name (it usually starts with an upper case letter to show the prefix has ended) that indicates what the control does. The complete control name can have up to 40 characters. The name must start with a letter (this is taken care of by using prefixes) and can only contain letters (lower or upper case), numbers, and the underscore (\_) character. Even though you can have 40 character control names, keep the names as short as possible without letting them lose their meaning. This will save you lots of typing.

Let’s look at some example control names to give you an idea of how to choose names. These are names used in the **Sample** project looked at in Class 1 and Class 2. Examples:

**frmSample** - Form for the Sample project

**cmdBeep** - Command button that causes a beep

**lblPick** - Label showing number picked

**optBlue** - Option button that changes background color to Blue

**chkTruck** - Check box that displays or hides the truck image

This should give you an idea of how to pick control names. We can’t emphasize enough the importance of choosing proper names. It will make your work as a programmer much easier.

**Setting Properties in Run Mode**

To illustrate the importance of proper control names, let’s look at a common task in Visual Basic. We have seen one of the steps in developing a Visual Basic project is to establish control properties in design mode. You can also establish or change properties while your project is in run mode. For example, in the **Sample** project, when you clicked on an option button, the **BackColor** property of the form was changed. To change a property in run mode, we need to use a line of BASIC code (you’re about to learn your first line of BASIC!). The format for this code is:

**ControlName.PropertyName = PropertyValue**

That is, we type the control’s name, a dot (same as a period or decimal point), the name of the property we are changing (found in the properties window), an equal sign (called an assignment operator), and the new value. Such a format is referred to as **dot** **notation**.

The code used to change the **Sample** project form background color to blue is:

**frmSample.BackColor = vbBlue**

Notice proper control naming makes this line of code very understandable, even if you don’t know any BASIC. It says that background color of the Sample form has been set to blue.

**How Control Names are used in Event Procedures**

Another place the importance of proper control naming becomes apparent is when we write event procedures (discussed next). When you put a control on a form, all of the possible event procedures that control can have are added to your project. We have seen that these event procedures are viewed in the code window. The structure for these event procedures is:

Header line: **Private Sub ControlName\_EventName()**

[BASIC code goes here]

Footer line: **End Sub**

Note the header line uses the control name. So, with proper naming, we can easily identify each event procedure.

As an example, using **Sample** again, the **Click** event procedure for the **optBlue** control is:

**Private Sub optBlue\_Click()**

**frmSample.BackColor = vbBlue**

**End Sub**

**DATA TYPES**

Data types can be called as the structure in which data will be stored in the memory. Thus the compiler must know the exact amount of memory space a data will be using. Since data is stored in a variable, therefore the structure of the variable should be known by the compiler to allocate memory space for the variable. Most of the languages compel you to declare variables before they are used. Visual Basic too enforces you to declare variables before you use them, but on your choice i.e. if you want to be prompted for using all undeclared variable, add option Explicit statement in the genera1 section of your form. Otherwise, you can use on-the-fly variables.

**SUMMARY OF DATA TYPES**

Visual Basic Language has 14 basic Data Types, and even Visual Basic allows you to create user defined data-types. The following table shows the supported data types, including their storage sizes and ranges.

|  |  |  |
| --- | --- | --- |
| **Data type** | **Storage size** | **Range** |
| **Byte** | 1 byte | 0 to 255 |
| **Boolean** | 2 bytes | **True** or **False** |
| **Integer** | 2 bytes | -32,768 to 32,767 |
| **Long** (long integer) | 4 bytes | -2,147,483,648 to 2,147,483,647 |
| **Single** (single-precision floating-point) | 4 bytes | -3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values |
| **Double** (double-precision floating-point) | 8 bytes | -1.79769313486232E308 to  -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values |
| **Currency** (scaled integer) | 8 bytes | -922,337,203,685,477.5808 to 922,337,203,685,477.5807 |
| **Decimal** | 14 bytes | +/-79,228,162,514,264,337,593,543,950,335 with no decimal point;  +/-7.9228162514264337593543950335 with 28 places to the right of the decimal; smallest non-zero number is  +/-0.0000000000000000000000000001 |
| **Date** | 8 bytes | January 1, 100 to December 31, 9999 |
| **Object** | 4 bytes | Any **Object** reference |
| **String**  (variable-length) | 10 bytes + string length | 0 to approximately 2 billion |
| **String** (fixed-length) | Length of string | 1 to approximately 65,400 |
| **Variant** (with numbers) | 16 bytes | Any numeric value up to the range of a **Double** |
| **Variant** (with characters) | 22 bytes + string length | Same range as for variable-length **String** |
| User-defined (using **Type**) | Number required by elements | The range of each element is the same as the range of its data type. |

**VARIABLES**

Variables in general, can be called as the temporary storage location, where the information is stored and can be over written. In Visual Basic Variable name must always start with an alphabet and call include any combination of letters, numbers, underscores and hyphen sign. Variable name can be up to 255 characters long and can include - (underscore). Variable in Visual basic are not case sensitive, Visual Basic by default changes the case of the variables as in declaration. If you are not using a declared variable, then visual Basic automatically changes the case at all the places where the variable has been used, according to the last case of the variable you have typed. This may help in debugging a misspelled variable name. Always try to make sure that the variable name you've declared is not a Visual Basic Keyword. Visual Basic provides basically 14 standard types of variables. Besides this variable type can also be define the user that is user-defined variables. User defined variables belong to user defined data types. Declaring a user-defined data type will be discussed later.

Declaring Variables

**Dim Statements**

Use Dim Statement to declare the data type of a variable at module or procedure level.

Syntax:

Dim VARIABLENAME [(Subscript)] AS [NEW] [TYPE]

Example:

**Dim** x As **Integer:** Declares an integer variable x and assigns 0 by default.

**Dim** x As **String:** Will declare an empty string [character type variable] variable x.

**Dim** x As **Form:** Declares x as object variable which will hold an instance of a Form. If new keyword is not added the x will be an empty variable, if included then an instance of the object exists initially.

**Dim** x As **New Form**: If no data-type is mentioned then the variable is declared as the variant type. You can now assign value of any data-type to this variable.

**Public** X As **Integer:** Declares a variable x of integer type and sets its scope for the entire Module.

**Classification of Variables and Their Scope**

On the basis of their scope, variables in Visual Basic can be classified into two groups.

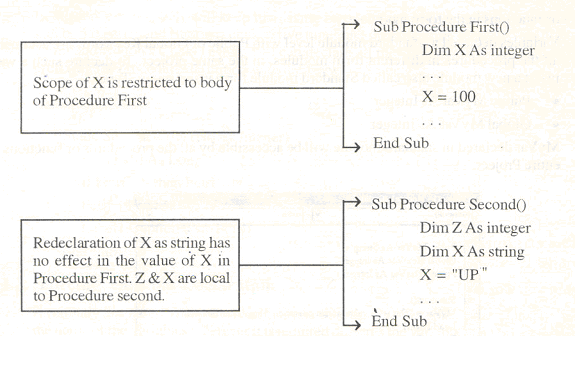
* Procedure Level Variables
* Module Level Variables

Procedure level variable are the variables declared within procedure and are private to a procedure only, their values are-not accessible by other procedures you can read or write value from and to a procedure level variable.

Module level variables are the variables declared in a module and are accessible by all the users within a module. These variables can be used by all procedures in the same module or other modules depending upon the way they have been declared. .Module level variables are of two types, public and private. Variables which are declared as public are public to all other modules, but the scope of private variables limits to the module where they have been declared.

**Private Variables**

Visual Basic isolates variables within procedures unless they are not declared otherwise. Variable declared within the body of the procedure or the function will be available only within the body of that procedure or function. -



Similarly variable declared at form level with Dim or Private statements are local to that particular module only. But such variable are accessible by different procedures in the same form module. Such variables are not accessible by other form in order to declare a variable accessible by all the forms and modules. You must declare global variable Private variables lose their existence as soon as the procedure is over. Private Variables are declared by simply using.

* Private keyword
* Dim Statement.

Syntax:

Private Variable name ([(subscript)]) As Variables declared at any event procedure or function with Dim statement is local to that procedure or function only. Variables declared in the General Section of the form module with Private or Dim Statement is local to that form module e.g., Private my\_variable As Integer. Such a variable can be called as a Public variable and is accessible by all the procedures with in the form. Variables declared in the code modules with Private Statement are local to them only.

**Public Variable**

Variables in a form, accessible by all the procedures and functions can be called as Public Variables. Variables are declared public when you need a variable required by all the procedures. In order to declare a public Variable, move to the General Declaration of the form and declare a Variable using Public Keyword.

Syntax:

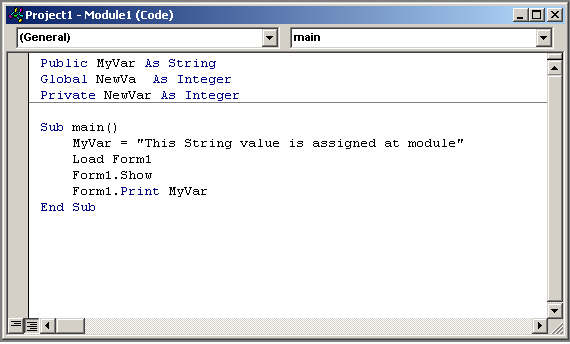
Public Variable name [[(subscript)]As [NEW]type]

E.g. Public MyVar As integer.

In the above example MyVar will be accessible by all the event procedures and. general procedures or functions in the form module. Variables declared at standard module level with Public or Global Keywords are accessible by all the procedures in different form modules, in the same project. To declare such a variable insert a new module also called Standard module form insert menu.

* Public MyVar As Integer
* Global MyVar As Integer

MyVar declared in standard module will be accessible by all the procedures or functions in the entire Project.



**Static Variables**

Variables in Visual Basic are always initialized with 0 or empty values. Life of variable is limited to the execution of the body of the procedure *or f*unction where they are declared. Thus each time the execution of the procedure's or function's body ends, variables loose their values and are reinitialized on the recall of the Procedure or function (with value 0 or empty). Sometimes, you require variables which can keep their values safe even after the procedure is over. In order to retain the values of the variables, variables are declared static. Static variables retain their previous values and remain in the memory till the execution of the Program.

Syntax:

STATIC Variablename [[subscript)] As [New] Type]

Example:

Private Sub Increase

Static X

X = X + 1

Print X

End sub.

In the above example X will keep increasing by one as it has been declared static. Value of X will be 5 if command Button is clicked five times and the procedure Increase is called within the command Button's Click event. You can declare all the variables in a procedure as static by prefixing the procedure definition with the keyword Static.

*Example:*

Static Sub Factorial (Number1 As Integer)

Dim Fctrl As Long

If Fctrl =0 then Fctrl =Number1

Fctrl =Fctrl \* (Number1 - 1)

Label1.Caption = Str (Fctrl)

End Sub

**Forcing Variable Declaration**

As already discussed Visual Basic allows you "On-the-fly" variable declarations; i.e. just typing the name of the Variables wherever it is required declares the variable of a Variant type data type.

*Example:*

MyVariable = 50 + 50.

Without format declaration.

This causes sometimes difficult bugs in the program. Thus declaration of variables is forced by using OPTION EXPLICIT Statement in the general section of the form.

After incorporating this statement to your modules, Visual Basic will not let you use an undeclared variable. If you want Option Explicit statement to be automatically added in your projects, choose the check box "Require Variable Declaration" from Options dialog box of your Visual Basic IDE.

**Assigning Values to Variables**

Values to Variables in Visual Basic can be assigned by using '=' operator known as assignment operator. Values are always assigned from Right to left like X =10. Values to object Variables are assigned by using Set Statement

Syntax:

Set ObjectVar = {[New] object Expression | Notation}

Example:

Dim MyForm as Forml

Set Myform = New Forml

**User Defined Data Types**

User defined data-types are the structures defined by user. This is done by using Type-End Type Statements. Used at Module level to define a user-defined data type containing one or more elements. User D.rl!!1eddata types are also known as Data Structure enables you to store different types of data or multiple elements of the same type of data in a single variable. A user defined data type is created with the **TYPE** statement in General declaration section of Modules.

Syntax:

Type <Name>

VARIABLE DECLARATION

--------------

--------------

--------------

End Type

Example:

Let's declare a data type which will contain the information of Employees of an organization.

Type Emplnfo

Name As String

Id As String

Dept As String

DOB As Date

JobTitle As String

Basic As CUlTency

End Type

After declaring the data-type, you should declare a variable or array of this data type wherever you need to use this structure.

Dim Emp As EmpInfo

The above declaration makes Emp such a variable which can store four string types, one Date and one integer type values. Value to such a variable is assigned as follows.

VARIABLENAME.StructureMember = Value

Example:

Emp.Name = "ASHISH'

Emp.Id = "UP-001"

Emp.Dept = "Education"

Emp.DOB = #11-12-1976 #

Emp.Jobtitle = "Systems Officer"

Emp.Basic = "6000"

**CONSTANTS**

Visual Basic's named constant feature allows you to use mnemonic names for values that never change. Rules to name the constant are same as for the variables. Constants Call is local, module level and global. Visual basic has some in-built constants generally with a prefix VB e.g. VBYesNo, VBOKOnly, VBRed etc.

Constants can be declared by using Const Keyword e.g., const Pi = 3.14. The value of Pi call be used any where in the program e.g.

r.= 5

Area = Pi \* r\*I

where Area and r are the variables.

## Chapter: 3 Visual Basic Controls

**Label Control**

A **label** is a control that displays information the user cannot edit directly. It is often used to provide titles for other controls. Or, it is used to display the results of some computer operation. The label control is selected from the toolbox. It appears as:

**In Toolbox**: **On Form (default properties)**:

**Properties**

A few useful properties for the label are:

**Property Description**

**Name** Name used to identify label. Three letter prefix for label names is **lbl**.

**Caption** Text (string type) that appears in the label.

**Font** Sets style, size, and type of Caption text.

**Alignment** Sets whether Caption text is left-justified, right-justified, or centered in label.

**BackColor** Sets label background color.

**ForeColor** Sets color of Caption text.

**BorderStyle** Determines type of label border.

**Left** Distance from left side of form to left side of label.

**Top** Distance from top side of form to top side of label.

**Width** Width of the label in twips.

**Height** Height of label in twips.

**Visible** Determines whether the label appears on the form (in run mode).

**Example**

Make sure Visual Basic is running and start a new project. Put a label on the form. Resize it and move it, if desired. Set the Caption property. Try different Fonts. Try different values of the Alignment property. Notice Alignment only centers the Caption horizontally - there is no vertical alignment. See the difference between the two BorderStyle possibilities. Notice the default value (**0-None**) makes the label fit into the form, where the other value (**1-Fixed Single**) gives the label a three-dimensional inset look. Change the BackColor and ForeColor properties. You may find certain color combinations that don’t do a very good job of displaying the Caption when in color. Make sure you are aware of combinations that do and don’t work. You want your user to be able to read what is displayed.

The most used label property is Caption. It holds the information that is displayed in the label control. There are two things you need to be aware of. First, make sure your label is big enough to hold any Caption you might provide for it. Second, note the Caption is a string type property. It can only hold string values. When setting the Caption property in run mode, the Caption information must be in quotes. For example, if you have a label control named **lblExample** and you want to set the **Caption** property to **My Label Box**, you would use the BASIC code (note the dot notation):

**lblExample.Caption = “My Label Box”**

You don’t have to worry about the quotes when setting the Caption in design mode. Visual Basic knows this is a string value.

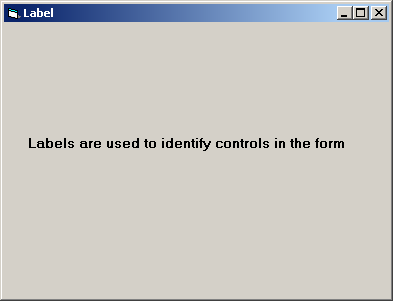
**Events**

There is only one label event of interest:

**Event Description**

**Click** Event executed when user clicks on the label with the mouse.

With this event, you could allow your user to choose among a set of displayed label boxes. Why would you want to do this? Example applications include multiple choice answers in a test or color choices.



**Text Box Control**

The **text** **box** control is used to display information entered in design mode, by a user in run mode, or assigned within an event procedure. Just think of a text box as a label whose contents your user can change. The text box is selected from the Visual Basic toolbox. It appears as:

**In Toolbox**: **On Form (default properties)**:

**Properties**

The text box has a wealth of useful properties:

**Property Description**

**Name** Name used to identify text box. Three letter prefix for text box names is **txt**.

**Text** Text (string value) that appears in text box.

**Font** Sets style, size, and type of Text.

**Alignment** Sets whether Text is left-justified, right-justified, or centered in text box (only works when **MultiLine** property is **True**).

**MultiLine** Specifies whether text box displays one line or multiple lines.

**ScrollBars** Specifies type of displayed scroll bar(s).

**MaxLength** Maximum length of displayed Text. If **0**, length is unlimited.

**BackColor** Sets text box background color.

**ForeColor** Sets color of Text.

**BorderStyle** Determines type of text box border.

**Left** Distance from left side of form to left side of text box.

**Top** Distance from top side of form to top side of text box.

**Width** Width of the text box in twips.

**Height** Height of text box in twips.

**Locked** If **True**, user can’t change contents of text box (run mode only).

**Visible** Determines whether the text box appears on the form (in run mode).

**Example**

Start a new Visual Basic project. Put a text box on the form. Resize it and move it, if desired. Set the Text property. Try different Fonts. Try different values of the Alignment property. Notice you can’t center or right justify text unless the MultiLine property is **True**. See the difference between the two BorderStyle possibilities. The label box used **None** as default, the text box uses **Fixed** **Single**. Change the BackColor and ForeColor properties. Set MultiLine to **True** and try different ScrollBars values. I think you can see the text box is very flexible in how it appears on your form.

Like the Caption property of the label control, the Text property of a text box is a string value. So, when setting the Text property in run mode, we must enclose the value in quotes (“) to provide a proper assignment. Setting the Text property in design mode does not require (and you shouldn’t use) quotes.

**Events**

The most important property of the text box is the Text property. As a programmer, you need to know when this property has changed in order to make use of the new value. There are two events you can use to do this:

**Event Description**

**Change** Event executed whenever **Text** changes.

**LostFocus** Event executed when the user leaves the text box and causes an event on another control.

The **Change** event is executed a lot - every time a user presses a key while typing in the text box, the Change event procedure is called. Looking at the Text property in this event procedure will give you its current value.

The **LostFocus** event is the more useful event for examining Text. Remember in placing controls on the form in design mode, you can make one control ‘active’ by clicking on it. There is a similar concept while an application is in run mode. A user can have interaction with only one control at a time. The control the user is interacting with (causing events) is said to have **focus**. While a user is typing in a text box, that box has focus. The LostFocus event is executed when another control gets focus. At that point, we know the user is done typing in the text box and is done changing the Text property. That’s why this event procedure is a good place to find the value of the Text property.

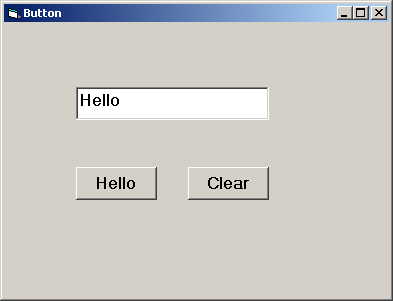


**Command Button Control**

The easiest way to allow the user to interact with an application is to provide a button to click. Command buttons the plain buttons that you simply click and release are the most common type of buttons. These are the buttons you see everywhere in Visual Basic applications. They are usually just rounded, rectangular, gray buttons with a caption.

Visual Basic allows us to create two types of buttons: command buttons and image buttons.

**Command buttons** are also called push buttons and allow the user to click them to perform the required action. **Image buttons** display a picture on the screen.



**The Button Properties**

Some of the properties of buttons are:

**Caption**: Sets the text on the command button.

**Picture**: Sets the graphic to be displayed on the image button.

**Enabled**: Sets a value that determines whether a control can respond to user-generated events.

**The Button Methods**

One of the methods used often with the command and the image buttons is:

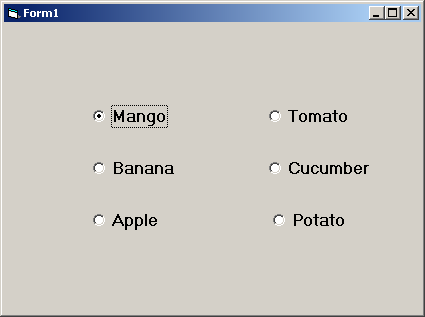
**Move**

**The Button Events**

The commonly used event is:

**Click Event**: Occurs when the user clicks on the button.

**Frame Control**

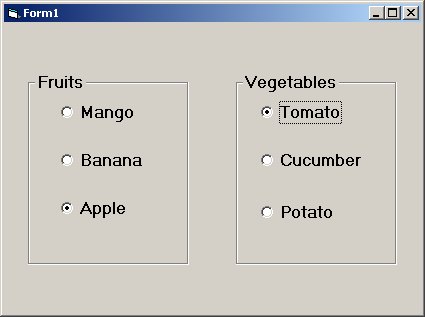


In the above figure, the option buttons on the left represent animals while those on the right represent countries. Theuser must be able to select one animal and one country. However, when the option buttons are placed as in figure we can select one of them.

Thus, even if two sets of option buttons have nothing in common with each other but they appear on the same form, only one of the option buttons can be chosen. This is because when option buttons are placed directly in a form they are said to form a 'group'. Option buttons in a group are assumed to be related and so only one of them can be selected.

Thus, the need of the hour is to separate the two sets of option buttons in the above figure into separate groups. This is achieved using the 'Frame' control. The frame control can be used to separate option buttons into groups. Once this is done, it is possible to select one option button in each group. Thus, all the option buttons inside any given frame constitute a group. When such separate groups are to be created, the frame should always be drawn first and then the option buttons placed in it.

The following figure shows the two sets of option. Buttons are separated into two groups using the frame controls.



**The Frame Properties**

The frame is essentially meant to be a container for other controls. Thus, it does not have any unique properties. One property of the group box that is important is:

**Caption**: The text that appears on the frame.

**The Frame Methods**

One of the methods associated with the frame control is:

**Move**: Moves the frame control along with the controls placed within it.

**Drag**: Begins, ends or cancels a drag operation of the control.

**The Frame Events**

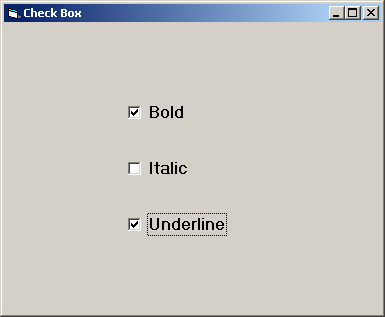
One of the commonly used events of the frame control is:

**Click**: Occurs when the user clicks on the frame control.

## The Check Box

A checkbox indicates whether a particular condition is on or off. Checkboxes are used in an application to give users true/false or yes/no options. Since checkboxes work independently of each other, a user can select any number of check boxes at the same time. Check box controls are used to display multiple choices from which the user can select one or more.

The following figure shows a checkbox control as displayed on the Visual Basic toolbox.



# The Check Box Properties

The checkbox properties are similar to those of the radio button.

**Caption** - The text that appears next to the checkbox.

**Value** - This property specifies whether the check box is selected.

# The Check Box Methods

Typically, there are no special methods associated with check boxes. Among the methods associated with check boxes is:

# Move

# The CheckBox Events

Among the events supported by the check box is:

**Click**- Occurs when the user clicks on the checkbox.

**The Radio Button or Option Button**

The radio buttons, also referred to as Option buttons, are used when the user can select one and only one of the multiple options. The following figure shows a radio button as represented in the Visual Basic tool box. It shows a screen of Visual Basic with the option buttons placed on it. The user can select only one of these option buttons.



The Option Button Properties

Among the often used properties of the option buttons are:

Caption: The text that appears next to the option button.

Value: This property specifies whether the option button is selected.

Enabled: Sets a value that determines whether a form or control can respond to user-generated events.

The Option Button Methods

Typically, there are no special methods associated with option buttons. A method associated with option buttons is:

Move: Moves a control on the form.

The Option Button Events

While option buttons support several events, the most commonly used is:

Click - Occurs when the user clicks on the option button.

**The List Box**

List boxes display long lists of options from which users can choose. The following figure shows a list box control as displayed in the toolbox of Visual Basic. To place the list box in the form, click on the symbol shown in figure and then click-and-drag in the form where the control is to be placed.

List boxes do just what their name implies: display a list of items. The user can make a selection from that list, and Visual Basic will inform our program what’s going on. Because list boxes can use scroll bars if a list gets too long, these controls are very useful to present long lists of items in a way that doesn’t take up too much space.



The advantage of using a list box lies in the user not having to remember all possible options. In addition, it also precludes the possibility of an invalid option.

# The List Box Properties

Some properties unique to a list box are:

**List Index**: Is a number used to access individual elements in the list box. The list index starts with 0 for the first item in the list box and takes on consecutive numbers for the following elements. Thus the list index identifies each item in the list box. Items in a sorted order. If this option is not set then the items are in the order in which they have been a Me()'to" Multi Select: - Used to specify if the user can select multiple items in the list.

**List Count**: How do you keep track of the total number of items in a list box? You use the **ListCount** property; that is, if you loop over the **List** array, you’ll use **ListCount** as the maximum value to loop to. Used to return the number of items in a list box.

**Selected**: Sets the selection status of an item in a List Box control. You get the index of the selected item in a list box with the **ListIndex** property. If no item

is selected, **ListIndex** will be \_1.

**Sorted:** Sorting a list box can change the indexes of the items in that list box (unless they were already in alphabetical order). After the sorting is finished, the first item in the newly sorted list has index 0, the next index 1, and so on. If you want to change the indexes of the items back to their original values, you can set their **Index** properties.

**Multiselect :** A multiselect list box allows the user to select a number of items at one time. You make a list box into a multiselect list box with the **MultiSelect** property. The user can then select multiple items using the Shift and Ctrl keys. Here are the possible settings for **MultiSelect**:

* *0* Multiple selection isn’t allowed (this is the default).
* *1* Simple multiple selection. A mouse click or pressing the

spacebar selects or deselects an item in the list. (Arrow keys

move the focus.)

* *2* Extended multiple selection. Pressing the Shift key and

clicking the mouse or pressing the Shift key and one of the

arrow keys extends the selection from the previously selected item to the current item.

Pressing the Ctrl key and clicking the mouse selects or deselects an item in the list. The **DblClick** event isn’t very useful with multiselect list boxes, because when

you click the list box a second time, every item but the one you’ve clicked is deselected. In addition, a **Click** event is generated each time the user selects a new item, and you might want to wait until all selections are made before taking action. This is why you often use a command button to initiate action after a user selects items in a multiselect list box. Take a look at the following example to see how this works.

**The List Box Methods**

List box methods are used to manipulate the list box and/or its items at run time. Here are some methods that are commonly used:

**Add item**: This method adds the specified item to the list box. Then new item is appended at the end of the list.

**Remove Item**: Used to delete an item from the list.

**Set Focus**: Used to make the list box the current active element.

You can add items to a list box at either design time or at runtime. At design time, you can use the **List** property, which is a very handy array of the items in the list box; and at runtime, you can use both the **List** property and the **AddItem()** method. Here’s how you use the **List** property in code (keep in mind that you can get or set items in the list box with the **List** array):

 ListBox.List(index) [= string]

At design time, you can add items directly to your list box by typing them into the

**List** property in the Properties window. Selecting the **List** property displays a

drop-down list (which is appropriate considering you’re filling a list box), and you can type item after item into the list box that way.

At runtime, you can either use the indexed **List** property as detailed previously, or the **AddItem()** method this way:

Private Sub Form\_Load()

List1.AddItem (\_Item 1\_)

    List1.AddItem (\_Item 2\_)

    List1.AddItem (\_Item 3\_)

    List1.AddItem (\_Item 4\_)

End Sub

When you add items to a list box, each item is given an *index*, and you can refer to the item in the list box using this index (for example, you can get the item’s text by using the **List** property: **List (*index*)**). The first item added to a list box gets the index 0, the next index 1, and so on.

List (0) = "Item 0"

List (1) = "Item 1"

List (2) = "Item 2"

List (3) = "Item 3"

Now we can refer to the items in the list box by index using the **List** property as **List (0)**, **List (1)**, and so on. When the user clicks the list, causing a **Click** event, we can display the item number the user clicked with the **ListIndex** property, which holds the index of the currently selected item:

Private Sub List1\_Click()

    MsgBox "You clicked item " & Str(List1.ListIndex)

End Sub

You can remove items from a list box at design time simply by deleting them in the **List** property. At runtime, you use the **RemoveItem()** method. Here’s an example; in this case, we add four items, Items 0 through 3 to a list box:

Private Sub Form\_Load ()

    List1.AddItem ("Item 0")

    List1.AddItem ("Item 1")

    List1.AddItem ("Item 2")

    List1.AddItem ("Item 3")

End Sub

Item 0 has index 0 in the list box, Item 1 has index 1, and so on. To remove, say, Item 1 when the user clicks a command button, we can use **RemoveItem** and pass it the items index:

Private Sub Command1\_Click()

    List1.RemoveItem 1

End Sub

The List Box Events

Some events commonly required of a list box events are:

**Click**: Occurs each time the user clicks on the list box. You use the **Click** event just as you use the **Click** event in a button, with a **Click** event handler. Here, we display the item in the list box the user has clicked, using the **ListIndex** property (you can get the selected items text with **List1.List(ListIndex)** or with **List1.Text**)

List boxes can also be multiselect list boxes (see \_Using Multiselect List Boxes\_ later in this chapter), which means the user can select a number of items in the list box. If your list box is a multiselect box, you can determine which items the user has selected by using the **Selected** property this way:

For intLoopIndex = 0 To List1.ListCount - 1

    If List1.Selected(intLoopIndex) Then

...

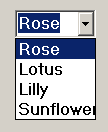
    End If

Next intLoopIndex

**The Combo Box or Dropdown List Box**

A combo box control combines the features of a text box and a list box. This control allows the user to select an item either by typing text into the combo box, or by selecting it from the list. The figure below shows a combo box control as displayed in the Visual Basic tool box. Combo boxes are list boxes combined with text boxes. With combo boxes, you can give users the option of selecting from a list (usually a drop-down list activated when users click the downwards-pointing arrow at right in a combo box) or typing their selections directly into the text box part of the combo box.

Combo boxes present a list of choices to the user such that only the selected item is displayed while the rest of the list is hidden. The user can see the list box by clicking on the arrow on the right hand corner of the combo box's text box portion. If the number of items exceeds what is box portion of the combo box, a vertical scroll bar will automatically appear on the control. The user can then scroll up and down through the list.



The Combo Box Properties

Typically, a combo box has all the properties exhibited by the text box and the list box.

Properties unique to the combo box are:

**Style**: The value assigned to this property decides the "look" of the combo box. There are three combo box styles:

*i)* **Simple Combo Box**: Specifies a simple combo box in which the list is displayed at all times. The user can also enter text.

*ii)* **Drop down Combo Box**: The user can either enter text directly (as in a text box) or click the detached arrow at the right of the combo box to open a list of choices.

*iii)* **Drop down List Combo Box**: Displays a list box when the user clicks on the arrow.

The user can only select an item from the list and cannot enter text in the box.

Creating Simple Combo Boxes, Drop-Down Combo Boxes, And Drop-Down List Combo Boxes Combo boxes are those controls that usually display a text box and a drop-down list. In fact, you might think there is only one kind of combo box, but there are really *three* types, and you select which type you want with the combo box’s Style property. The default type of combo box is probably what you think of when you think of combo boxes, because, as mentioned, it is made up of a text box and a drop-down list. However, you can also have combo boxes where the list doesn’t drop down (the list is always open, and you have to make sure to provide space for it when you add the combo box to your form) and combo boxes where the user can only select from the list. Here are the settings for the combo box Style property.

0 drop-down combo, includes a drop-down list and a text box. The user can select from the list or type in the text box. (This is default.)

1 Simple Combo, simple combo box, includes a text box and a list, which doesn’t drop down. The user can select from the list or type in the text box. The size of a simple combo box includes both the edit and list portions. By default, a simple combo box is sized so that none of the list is displayed. Increase the Height property to display more of the list.

2 DropDownList, drop-down list. This style allows a selection only from the drop-down list. This is a good one to keep in mind when you want to restrict. The user\_s input; however, if you want to use this one, you should also consider simple list boxes. The selected item appears in the (read-only) text box.

If you want to restrict the user’s input to items from the combo box’s list, set the combo box’s **Style** property to **VbComboDrop-DownList**, a predefined Visual Basic constant whose value is 2. In this style of combo boxes, the user cannot type into the text part of the control.

**Locked**: Used to specify whether the user can enter a value in the text box section of the control. You can *lock* a combo box by setting its **Locked** property to True. When locked, the user cannot enter text in the combo’s text box and cannot make selections from the combo’s list (although if the list is drop-down, it will still open). However, when programmers think of locking a combo box, it’s not usually the **Locked** property that they want.

Just as in the case of the list box, the combo box has the following properties:

**1. Index**

**2. List Count**

**3. Sorted**

**4. ListIndex**

You can also get the currently selected item’s index in the combo box’s list using the **ListIndex** property. If no selection is made (for instance, when the form first loads and the combo’s text box is empty), this property will return -1. If the user has altered the selection by typing into the text box (in other words, so the selected item no longer matches the item the combo box’s list), **ListIndex** will also be \_1. And if the user opens the combo box’s list and then clicks outside that list without making a selection, **ListIndex** is set to -1.

Unlike a list box, a combo box does not allow multiple selections.

**The Combo Box Methods**

The methods of the combo box are, typically, the same as those of the list box.

1. **Add Item** - This method will add the specified item to the combo box control.

You can add items to a Combo box at either design time or at runtime. At design time, you can use the List property, which is a very handy array of the items in the Combobox; and at runtime, you can use both the List property and the AddItem() method. Here’s how you use the List property in code (keep in mind that you can get or set items in the Combo box with the List array):

*Combo1*.List(*index*) [= *string*]

2. **Remove Item** – Used to delete an item from the combo box control.

Just as with list boxes, you can remove items from combo boxes using the RemoveItem() method. You just pass the index of the item you want to remove from the combo box’s list to RemoveItem().

You should note that removing an item from a combo box changes the indexes of the remaining items. After you remove Item 1 in the preceding example, Item 2 now gets

Index 1 and Item 3 get Index 2. If you want to change those indexes back to their original values, set the items Index properties.

**The Combo Box Events**

The combo box supports all the events that a list box has such as:

1. **Click** - You can also get Click events when the user makes a selection in the list box using the mouse. You can determine which item the user clicked using the combo’s ListIndex property (which holds the index of the clicked item) or get that item text using the Text property, because when you click an item, it is made the new selected item in the text box.

2. **Scroll** - Occurs when a user scrolls through the list in the list box.

Similarly, the combo box also has events in common with the text box:

3. **Change** - Occurs when the user changes the text in the text box portion of the combo box, just as it does when the user types in a standard text box. You can read the new text in the text box with the Text property. Here’s a fact that takes many programmers by surprise: no Change event occurs when you use the mouse to select an item in a combo box’s list, even if doing so changes the text in the combo’s text box.

In addition, the combo box has events that are unique to it. One such event is:

4.**Drop Down** - Occurs when the user clicks on the arrow to display the list.

## Chapter: 4 Working with Functions

Function can be called as the wealth of any programming language .Visual Basic has many in-built routines, some of them return some values, take some parameters and some of them just help you behind the scene without taking any parameters or return values.

This chapter covers a summary of most frequently used functions and statements in visual basic. Although, visual basic editor keep helping you by prompting for the full and correct syntax of the function, but few of them which are frequently used must be in your memory.

**Common built-in numeric functions.**

|  |  |
| --- | --- |
| ***Function*** | ***Description*** |
| Abs() | Returns the argument's absolute value. The absolute value is the positive equivalent of the argument, so the absolute value of both -87 and 87 is 87. Use absolute values for distance calculations and weight differences because such values must always be positive. |
| Atn() | Returns the argument's arc tangent, expressed in radians. To compute the arc tangent in degrees (or any other trigonometric function), multiply the argument by pi (approximately 3.14159) and then divide by 180. |
| Cos() | Returns the argument's cosine value, expressed in radians. |
| Exp() | Returns the argument's natural logarithm base. |
| Len() | Returns the number of memory characters required to hold the argument. |
| Log() | Returns the argument's natural logarithm. |
| Sin() | Returns the argument's sine value, expressed in radians. |
| Tan() | Returns the argument's tangent value, expressed in radians. |

**Common built-in string functions**

|  |  |
| --- | --- |
| ***Function*** | ***Description*** |
| Chr(int) | Returns the ASCII character that matches the numeric argument. |
| LCase(str) | Returns the argument in all lowercase letters. If any character in the argument is already lowercase, no change takes place for that character. |
| Left(str, int) | Returns the leftmost int characters from the string argument. |
| Len(str) | Returns the number of characters in the string. (Notice that Len() works on numeric arguments as well.) Also, Len() does not return a string even though Len() works with string arguments. |
| LTrim(str) | Returns the string argument, with any leading spaces trimmed off. |
| Mid(str, intStart[, intLen]) | Returns a substring of the string argument, starting with the character at intStart and continuing until the entire rest of the string is extracted or until the optional intLen characters have been extracted. Mid() is called the midstring function because it can return the middle portion of a string. |
| Right(str, int) | Returns the rightmost int characters from the string argument. |
| RTrim(str) | Returns the string argument, with any trailing spaces trimmed off. |
| Str() | Converts its numeric argument to a string with the numeric digits in the string. |
| UCase(str) | Returns the argument in all uppercase letters. If any character in the argument is already uppercase, no change takes place for that character. |

**Date and time functions.**

|  |  |
| --- | --- |
| ***Function*** | ***Description*** |
| Date | Returns the current date. |
| DateSerial(intYr, intMo, intDay) | Returns an internal date value for the three arguments. |
| DateAdd(strIntrvl, intN, dteDate) | Adds the intN value to the date specified by dtrDate for the given strIntrvl. |
| DateDiff(strIntrvl,dte1,dte2) | Returns the number of time intervals (specified by strIntrvl) between the two dates. |
| DatePart(strIntrvl, dteDate) | Returns the strIntrvl portion of the dtrDate. |
| Now | Returns the current date and time in the date format. |
| Time | Returns the current time. |
| Timer | Returns the number of seconds since midnight. |
| TimeSerial(hour, min, sec) | Returns the current date and time in the internal date format for the time specified. |

**The VarType() return values**

|  |  |  |
| --- | --- | --- |
| ***Return*** | ***Named Literal*** | ***Describes*** |
| 0 | vbEmpty | Empty and not initialized argument. |
| 1 | vbNull | Invalid data or a null string argument. |
| 2 | vbInteger | Integer argument. |
| 3 | vbLong | Long argument. |
| 4 | vbSingle | Single argument. |
| 5 | vbDouble | Double argument. |
| 6 | vbCurrency | Currency argument. |
| 7 | vbDate | Date argument. |
| 8 | vbString | String argument. |
| 9 | vbObject | Object argument. |
| 10 | vbError | Error argument. |
| 11 | vbBoolean | Boolean argument. |
| 12 | vbVariant | Variant argument. |
| 13 | vbDataObject | Data Access Object (DAO) argument. A Data Access Object is an advanced database value such as a field or record. |
| 14 | vbDecimal | Decimal argument. |
| 17 | vbByte | Byte argument. |

**Most Frequently Used Function With Description**

**1. Abs Function**

General Syntax:

Abs (num Expression)

Description**:**

The absoluter value of a number can be returned by this function. The num expression argument is a numeric expression of which you want to return the absolute value.

**2. Asc Function**

General Syntax:

Asc (str Expression)

Description:

The Asc function return the numeric code of a character. Str Expression is a string for which you want to return the numeric code of the first character.

**3. Chr Function**

General Syntax:

Chr (int ANSI Code)

Description:

The Chr function return the character that corresponds to the supplied code. The int ANSI code argument is an integer for which you want to return the corresponding character.

**4. Cos Function**

General Syntax:

Cos (dbl Angle)

Description:

The Cos function return the cosine of an angle. The argument for the dbl Angle is a numerical expression for which you want to return the cosine.

**5. Date Value Function**

General Syntax:

DateValue (date string)

Description:

This function helps to convert a date in the form of a string into a Visual Basic date data type. This function change differently formatted dates to a universal numeric form.

**6. Date, Date$ Function**

General Syntax:

Date $

Date

Description:

The Date function returns the current system date

**7. Time and Time$ Function**

General Syntax:

Time

Time $

Description:

The Time function returns the current system as a date type whereas the Time $ function returns your computer’s current system time as a string argument.

**8. UCase Function**

General Syntax:

Week day Name (weekday; [abbreviate], [first day of week])

Description:

The day of the week is based on the number passed can be returned by the weekday Name function. Weekday Name argument specifies the numeric value for the day of the week.

**9. Val Function**

General Syntax:

val (str Expression)

Description:

The Val function returns the numeric value of the supplied string expression. The argument under str Expression is the string for which you want to return the numeric value.

**10. Sqr Function**

General Syntax:

Sqr (dbl Expression)

Description:

The Sqr function is used to return the square root of a number, the argument of dbl Expression is a Numeric Expression whose square root you want to return.

**11. Rnd Function**

General Syntax:

Rnd (lng Expression)

Description:

It returns a single precision random number between 0 and 1. The argument for lng Expression is a value that determines which number in the pseudorandom sequence to return.

**12. Str Function**

General Syntax:

Str (number)

Description:

It converts numeric data into an unformatted string. The number argument is a type of numeric data to convert to a string.

**13. RTrim Function**

General Syntax:

RTrim (String)

Description:

The RTrim function helps to return a copy of a string with any trailing spaces removed. The String argument the string to be modified.

**14. LTrim Function**

General Syntax:

LTrim (String)

Description:

LTrim removes leading spaces from a string or byte array. The string argument in which the string being terminated.

**15. Len Function**

General Syntax:

Len (Variable Name)

Description:

The Len functions return the storage length of a variable. The are most commonly used to find the length of a string. The variable name argument is the name of the VB variable for which you want to determine the storage length.

## Chapter: 5 Control Statements

**Control Statements**

**Decision Making Statements**

**If-Then-Else**

Visual Basic provides if then else statements for decision making.

e.g. *If* condition True the statements *else* other statements.

If the test fails, and else part is not mentioned then the processing Skips to the next statement.

**Syntax:**

**If** condition **Then**

[*statements*]

[**ElseIf** condition-n **Then**

[*statements*]]...

**Else**

[*elsestatements*]]

**End If**

Example: if x =0 then Msgbox "Zero" else Msgbox "Non-Zero"

More often multiple statements’ are to be processed if a certain condition is true or false.

All such statements are thus blocked in If...End If statements.

**If** x =0 **Then**

Msgbox "Zero"

X=x+3

X=X\*4

**Else**

Msgbox "Non-Zero"

n=x-3

x = x/4

**End If**

Multiple Conditions can be checked in one if statement using ‘And’ keyword.

e.g.,

**If** x *>=0* ***And*** *x <=10* **Then**

Msgbox "Between 0 and 10"

**Else**

Msgbox "Below 0 or greater than 10"

**End If**

One out of many conditions can be checked for True using OR keyword.

Example:

**If** x >= 0 **Or** x <= 10 **Then**

Msgbox "Between 0 and 10"

**else**

Msgbox "Below 0 or greater than 10"

**End If**

You can also use nested If structures i.e. If-then Block within If-then Blocks. Nested if statements can be replaced by using If-ElseIf-EndIf structure.

Example:

**If** Div >= 60 **Then**

Result = "First"

**ElseIf** Div >= 45 **Then**

Result = "Second"

**ElseIf** Div >= 33 **Then**

Result = "Third"

**ElseIf** Div >= 30 **Then**

Result = "PassedWith Grace"

**Else**

Result="Fail"

**End If**

**Immediate If**

The Iff0 function provides immediate if for quick decisions. The Iff 0 function returns one part out of two, after evaluating the gi ven expression. The entire expression consists of two parts the if part and else part.

Example:

Result =**Iff** (Div >= 60, "First", "Second")

**Select Case Statement**

Executes one out of several groups of Statements, depending upon the value of an expression.

**Syntax:**

**Select Case** <Test Expression>

**Case** Expression list

[ Statements]

**Case** Expression list

[ Statements]

---------------------------------------------

---------------------------------------------

**Case Else**

Statements

**End Select**

The Case Else clause is used to indicate the statements to be executed if no match is found between the test expression and in expression List in any of the other case selections. Execution continues at the statement following End Select. Case Else statement should be added in the select case Block, in order to handle unforeseen test expression value. Comparison can be done using logical operators while testing the expression in each clause.

Example:

Case Is > 1000

Case Is ="Robert"

Multiple expression or ranges can be used in each Clause.

Example:

1.

**Select Case** Month(Date)

**Case** 1

MsgBox "Month is January"

**Case** 2

MsgBox "Month is Feb"

**Case** 6

MsgBox "Month is June"

**Case** 12

MsgBox "Month is December"

**Case Else**

MsgBox "Neither Jan nGr Feb nor June nor Dec"

**End Select**

2.

**Select Case** Month(Date)

**Case** 1,3,5, 7,8, 10, 12

MsgBox "Month has 31 Days"

**Case** 2

MsgBox "Month has 28 or 29 Days"

**Case Else**

MsgBox "Month has 30 Days"

**End Select**

**Iterations: Loop Structure**

Visual Basic Provides loop structures to perform iterative operations. Repeating actions can be implemented in two ways:

**1. Determinate** Loops**:** Repeats the operation a fixed number of times.

**2. Indeterminate Loops:** Continues until your each a specific predetermined goal, or continue until certain initial conditions have changed.

**For ...Next**

Using this loop structure enables the program to execute a block of statements a fixed number of times. At the beginning of the loop, specify the starting value and ending value the counter. The program executes till the counter reaches the ending value.

E. g.

**For** x =1 to 10

Text1.Text =X%

**Next** x

If you want to count by 2's, 3's, fractions or backwards, use

Step keyword.

E. g.

**For** 1%=10 to 1 **Step** - 1

Print "Its moving backwards" + SIT(1%)

**Next** 1%

Print "REACHED"

**Do…..Loops**

This loop structure enables the program to execute a block of statements an indefinite number of times based on a condition. Each time the code loops, the condition is tested again. When the condition finally becomes *false,* the loop terminates.

**Do- While Loop**

This executes a code indefinitely till some condition is True or False.

**Syntax:**

**Do while** the condition is true

Execute these statements

**Loop**

E. g.

VAR =10

**Do while** VAR<>O

Print "It is raining"

Var = Var 1

**Loop**

**Do.. Until Loop**

Instead of using *Do While* is test for a false condition, use the *Do-*Until.

Temp = 45

VAR= 0

**Do until** VAR

**If** temp <-25

Print "Its cold"

Var = 1

**Else**

Print "Still too hot!"

Temp = 20

**End if**

**Loop**

The loop executes till VAR is equal to 0, when VAR becomes 1, the loop terminates.

Sub Form\_Load()

**Do**

X = lnputBox ("Enter password")

**Loop** until X = "Vanilla Orange"

End Sub

It is an example code for password protection. The exact password, with proper cases should be typed. To stop an infinite loop, use *Ctrl-Break* or *Select Run» End* from *Menu Bar.*

**A while...Wend Statements**

Executes a series of statements as long as a given condition is True.

**Syntax**:

**while** condition

[Statements]

**Wend**

Example:

x= 10

**while** x > 10

x = x-I

Print x

**Wend**

**For Each...Next Loops**

For Each...Next loops are the loops that iterate through all the elements of an object's collection. A collection is a group of objects. Each object has its own identity in the collection. For instance, you might want to hide all the control's on a form then you can use the following code.

Dim Control As Object

**For Each** Control In Controls

Control. Visible =False

**Next** Control

In the above example the collection controls contains all the controls placed on a form. If there are no objects in the collection, For Each...Next loop is automatically terminated.

**WITH….END WITH STATEMENTS**

With-End With statements are used to avoid writing the name of the object again and again while referencing more than one time. Instead of referring repeatedly to the same object, you can identify the object and then use a with-End With statement to perform a series of actions on it.

Example:

All the Properties of TextBox can be referenced as

**With**Text1

.Text ="Howard"

.Height = 14000

.Width =18000

.ForeColor =RGB (0, 256, 0)

**End with**

This can also be used as convenient method for quickly accessing the members of user defined data types.

Example:

**Type** Emplnfo

Name As String

Id As Integer

Basic As hlteger

**End Type**

Dim Emp As EmpInfo

**With** Emp

.Name = "DivyaVishen"

.Id = 001

.Basic = 5456

**End With**

## Chapter: 6 Dialog Boxes

**A MsgBox() and InputBox() Overview**

You use input boxes and message boxes when you need to ask the user questions or display error messages and advice to the user. As stated earlier, the form's controls don't often work well for such user dialogs. For example, suppose the user is to enter a sales code of A, B, or C to indicate a discount to be used in a total calculation. Users don't always know what's expected of them, so a message box can pop up when the user enters a bad value and the message box can explain that the user needs to enter only A, B, or C. If the user enters an invalid code, your program could display an error message such as the one shown in Figure.

A message box is a dialog box you display to give the user information.

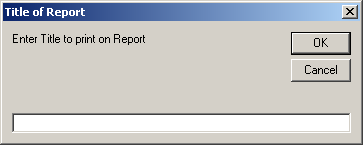




An input box is a dialog box you display to ask the user questions.

*A message box can tell the user what to expect.*

The Text Box controls that you've seen are great for getting values from the user. Other controls that you'll learn as you progress through this book also accept the user's input from the keyboard or mouse. Nevertheless, Visual Basic's controls just aren't enough to handle all the input that your program will need. Input boxes are great to use when the user must respond to certain kinds of questions. Text boxes and other controls are fine for getting fixed input from the user, such as data values with which the program will compute. Input boxes are great for asking the user questions that arise only under certain conditions. Input boxes always give the user a place to respond with an answer. In Figure the input box is asking the user for a title that will go at the top of a printed report listing.  
  
*Input boxes get user information.*

**

**Examining MsgBox()**

Always assign a MsgBox() function to an integer variable. The variable will hold the return value, and that value will indicate the button the user clicked (message boxes can display multiple buttons such as OK and Cancel).

Here is the format of the MsgBox() function:

anIntVariable = MsgBox( strMsg [, [intType] [, strTitle]])

strMsg is a string (either a variable or a string constant enclosed in quotation marks) and forms the text of the message displayed in the message box. Int Type is an optional numeric value or expression that describes the options you want in the message box.

Table 6.1, Table 6.2, and Table 6.3 contain all the possible values you can use for the type of message box you want displayed. (Visual Basic displays no icon if you don't specify an intType value.) If you want to use a value from two or more of the tables, you'll add the values together. Although you can use the integer value, if you use the built-in Visual Basic named literal, you'll more easily understand the message box's style if you ever have to change the message box in the future. strTitle is an optional string that represents the text in the message box's title bar. If you omit strTitle, Visual Basic uses the project's name for the message box's title bar text.

**Table 6.1a. The buttons displayed in a message box.**

|  |  |  |
| --- | --- | --- |
| ***Named Literal*** | ***Value*** | ***Description*** |
| vbOKOnly | 0 | Displays the OK button. |
| vbOKCancel | 1 | Displays the OK and Cancel buttons. |
| vbAbortRetryIgnore | 2 | Displays the Abort, Retry, and Ignore buttons. |
| vbYesNoCancel | 3 | Displays the Yes, No, and Cancel buttons. |
| vbYesNo | 4 | Displays the Yes and No buttons. |
| vbRetryCancel | 5 | Displays the Retry and Cancel buttons. |

**Table 6.1b. The icons displayed in a message box.**

|  |  |  |
| --- | --- | --- |
| ***Named Literal*** | ***Value*** | ***Description*** |
| vbCritical | 16 | Displays Critical Message icon. |
| vbQuestion | 32 | Displays Warning Query icon. |
| vbExclamation | 48 | Displays Warning Message icon. |
| vbInformation | 64 | Displays Information Message icon. |

**Table 6.1c. The default buttons displayed in a message box.**

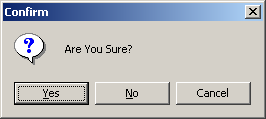
|  |  |  |
| --- | --- | --- |
| ***Named Literal*** | ***Value*** | ***Description*** |
| vbDefaultButton1 | 0 | The first button is the default. |
| vbDefaultButton2 | 256 | The second button is the default. |
| vbDefaultButton3 | 512 | The third button is the default. |

The options that you select, using the intType value in the MsgBox() function, determine whether the message box displays an icon and controls the modality of the message box. The modality determines whether a message box is application specific or system specific. If it is application specific, the user must respond to the message box before the user can do anything else in the application. If the message box is system specific, the user must respond to the message box before doing anything else on the system.

Modality determines how the system handles a dialog box.

The modality often causes confusion. If you don't specify a system-modal intType value of 4096 (or if you don't use the named literal vbSystemModal to specify the system's modal mode), the user's application will not continue until the user closes the message box, but the user can switch to another Windows program by pressing Alt + Tab or by switching to another program using the application's control menu. If, however, you do specify that the message box is system modal, the user will not be able to switch to another Windows program until the user responds to the message box because the message box will have full control of the system. Reserve the system-modal message boxes for serious error messages that you want the user to read and respond to before continuing the program.

The following MsgBox() function produces the message box shown in Figure



intPress = MsgBox("Are you Sure?", vbQuestion + vbYesNoCancel, "Confirm")

MsgBox()s Return Value

The reason that you assign MsgBox() functions to variables is so you can tell which button the user presses. Suppose that the user pressed the Yes button in Figure . The program could then print the report. If, however, the user pressed the No button, the program could describe what the user needed to do to get ready for the report (load paper, turn on the printer, and so on). If the user pressed the Cancel button, the program would know that the user didn't want the report at all.

Table 6.4 lists the seven possible MsgBox() return values. You can test either for the integer or the named literal return value.

**Table 6.1d. MsgBox() return values.**

|  |  |  |
| --- | --- | --- |
| ***Named Constant*** | ***Value*** | ***Description*** |
| vbOK | 1 | The user clicked the OK button. |
| vbCancel | 2 | The user clicked the Cancel button. |
| vbAbort | 3 | The user clicked the Abort button. |
| vbRetry | 4 | The user clicked the Retry button. |
| vbIgnore | 5 | The user clicked the Ignore button. |
| vbYes | 6 | The user clicked the Yes button. |
| vbNo | 7 | The user clicked the No button |

**Examining InputBox()**

You'll find that the InputBox() function is easy because it acts a lot like the MsgBox() function. The InputBox() function receives answers that are more complete than the MsgBox() function can get. Whereas MsgBox() returns one of seven values that indicate the user's command button press, the InputBox() function returns a string data value that holds the answer typed by the user.

Here is the format of the InputBox() function:

strVariable = InputBox( strprompt [, [strTitle] [, strDefault]

A[, intXpos, intYpos]]])

strPrompt works a lot like the strmsg value in a MsgBox() function. The user sees strPrompt inside the input box displayed on the screen. strTitle is the title inside the input box's title bar. strDefault is a default string value that Visual Basic displays for a default answer, and the user can accept the default answer or change the default answer.

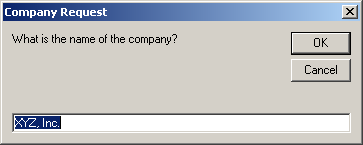
The intXpos and intYpos positions indicate the exact location where you want the input box to appear on the form. The intXpos value holds the number of twips from the left edge of the Form window to the left edge of the input box. The intYpos value holds the number of twips from the top edge of the Form window to the top edge of the input box. If you omit the intXpos and intYpos values, Visual Basic centers the message box on the form.

The following statement displays an input box that asks the user for a company name. The user either enters a response to the prompt or clicks the Cancel command button to indicate that no answer is coming.

strCompName = InputBox ("What is the name of the company?", \_

"Company Request", "XYZ, Inc.")

Figure 6.4 contains the message box displayed from this InputBox() function.  
 *Asking the user a question and getting the answer with InputBox().*

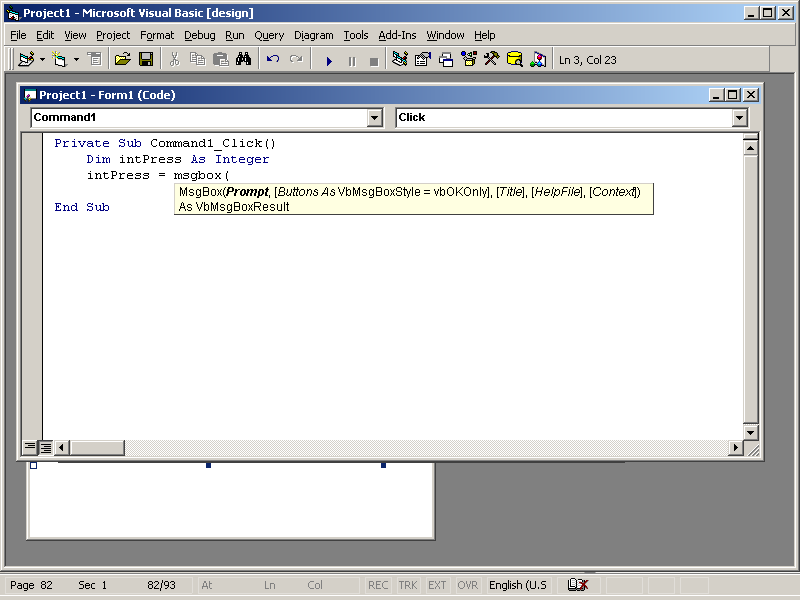


**Visual Basics Code Window Help**

Can you remember the named literals in this lesson's tables? How can you remember that the named literal value to display three buttons--Yes, No, and Cancel--is the vbYesNoCancel named literal?

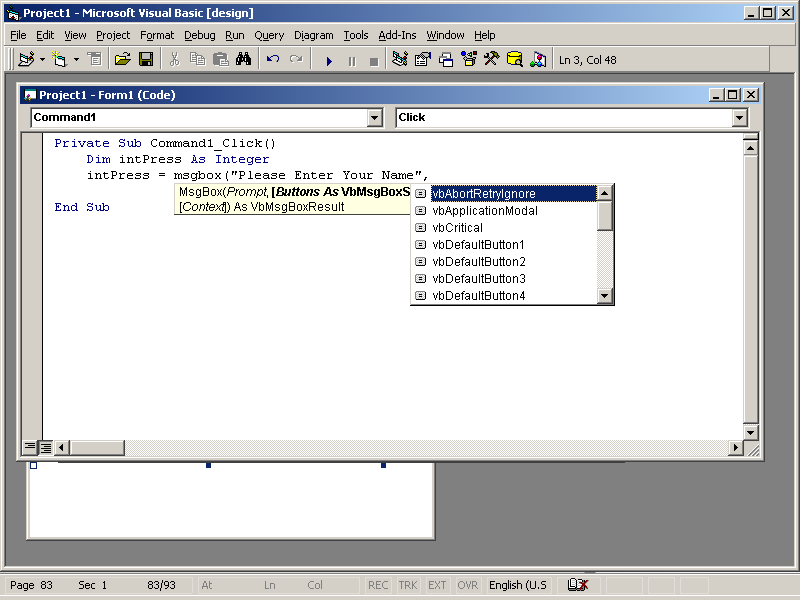
Visual Basic now supplies you with all the help you need. As soon as VB's Code window editor recognizes that you're entering a function, the editor immediately displays pop-up help that displays the function's format, as shown in Figure 6.5.

*Visual Basic displays the function's format for you.*

**

Visual Basic give you help not only with a function's format, but also with the functions named literals. When you get to any function argument that requires one of the named literals, Visual Basic displays a drop-down list box such as the one in Figure, from which you can select a named literal. To accept the selected named literal, press Enter, type a comma, or press the Spacebar to continue with the program.

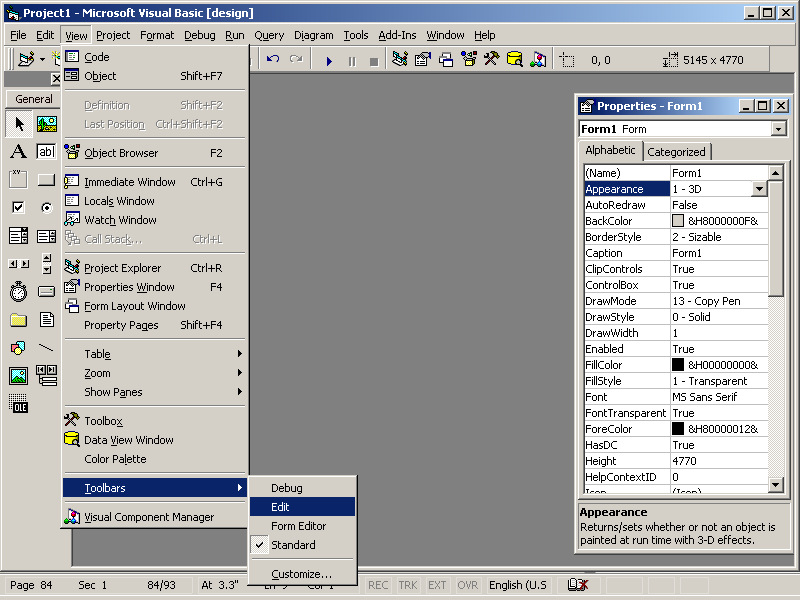
*Visual Basic displays the function's named literals.*



## Chapter: 7 Menus in VB Application

Before looking at menu creation, take a moment to familiarize yourself with Figure menu components. The rest of this lesson discusses the various components that make up most Windows menus. In working with Visual Basic, you've already seen these menu components.

Figure

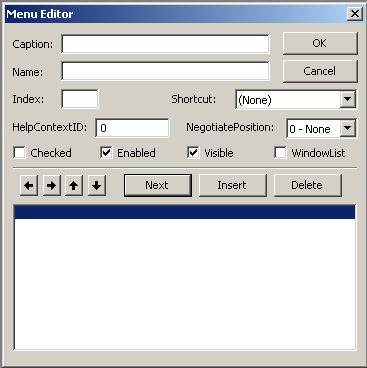


A menu is another control object just like a command button or a text box. Once you add a menu bar to an application, the menu bar and its options are all controls that you can manage from the Properties window. Even though the menu items are regular controls with properties you can set, the programming gurus don't often know that because they use a better resource than the Properties window for creating their menus. Whereas the Properties window is great for setting normal toolbox control properties, the Menu Editor makes for a better menu-creation tool.

The Menu Editor lets you quickly and easily place menu bar items into your application by pushing command buttons and typing a few property values. The Menu Editor contains menu description tools that let you create the application's menu bar, menu commands, and shortcut access keys to all of your applications.

The Menu Editor is a dialog box that you access from the Form window by pressing Ctrl + E or by selecting Tools | Menu Editor from Visual Basic's own menu bar. Figure shows the Menu Editor Dialog box.

*Creating a menu with the Menu Editor*

**

**Adding an Applications Menu Bar**

An application's menu bar is one of the easiest parts of the menu system to add. This section walks you through the steps necessary to add a menu bar. Subsequent sections show you how to add pull-down menu options to each of the menu bar commands.

The Menu Editor makes adding a menu bar to any application simple. Create a new project so you can practice creating a menu. The menu bar you create will contain the following options:

File

Edit

View

Help

This tutorial could go into a lot of detail, explaining all the nuances of the Menu Editor. Luckily, you don't need all that preliminary detailed description. The Menu Editor is most easily mastered by jumping in and building a menu from scratch. You don't need a bunch of theory to use the Menu Editor.

Every option on a menu bar, as well as the menu options, submenus, and separator bars that appear when you display a pull-down menu, has properties just as the other controls do. The Menu Editor acts like a dialog box that helps you set menu property values. The Properties window is perfect for the other controls, but as you'll see, menus require a few extra kinds of property choices that the other controls don't need. That's why using the customized Menu Editor is simpler than modifying an application's menu through the Properties window.

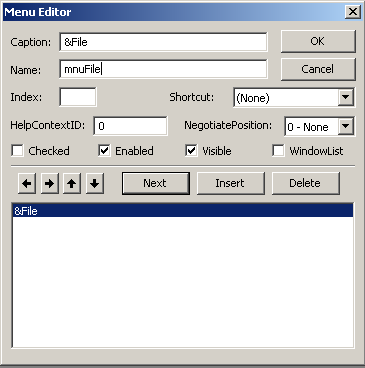
Perform the following steps to add a menu bar to your new project:

1. Press Ctrl+E to display the Menu Editor. Each menu bar command requires a caption (specified by the Caption property) and a name (specified by the Name property). The other Menu Editor items are optional.  
   The additional Menu Editor properties, such as the Enabled property that determines whether the menu item is grayed out and unavailable for certain procedures, as well as a Visible property, which determines when the user can see the menu bar command, are not needed for every option. You'll rarely change these extra property values from their default values for menu barcommands.  
   **2.** At the Caption prompt, type &File. The ampersand, as with the other controls' Caption properties, indicates an accelerator keystroke of Alt + F for the File menu item. As you type the Caption value, notice that Visual Basic adds the caption in the Menu Editor's lower section. The Menu Editor's lower half displays the menu bar and the pull-down options as you add them to the menu. The Menu Editor's top half contains a description of individual

items in the menu.  
 **3.** Press Tab to move the focus to the Name text box, and type mnuFile. The application will refer to the File menu bar item by the name mnuFile as needed. In other words, just as a command button might be named cmdPressMe, the menu bar option can be named mnuFile. The three-letter prefix indicates that the mnuFile object is a menu item and not some other kind of control. Your Menu Editor's window should look something like the one in Figure.  
  
The only accelerator keystroke available for menu bar options is the underlined Alt + keystroke that occurs as the result of the Caption property's underlined letter. Don't attempt to select Ctrl + keystroke from the Shortcut drop-down list box for the menu bar options. Ctrl + keystroke shortcut combinations are available only for pull-down menu options.

Don't press Enter or click the OK button to close the Menu Editor just yet, because you've got to add the additional menu bar options before closing the Menu Editor's window.

*The menu bar now has a defined File option*



**Adding Pull-Down Menu Options**

Each menu bar command opens a pull-down menu that consists of a series of options, separator bars, access keystrokes, and submenus. The Menu Editor's four arrow command buttons let you indent the pull-down menu commands from their matching menu bar commands to show which items go with which menu bar commands.

Now that you've added the menu bar, you can add the individual options to the pull-down menus. You didn't have to complete the menu bar before completing each pull-down menu. You could have added the File option to the menu bar and then completed the File option's pull-down menu before adding the View option to the menu bar. The order in which you add menu items doesn't matter at all. It is where you place them and how you indent them that determines the order in which the menu items appear.

The File pull-down menu will contain the following items:

The New command

The Open command with a shortcut access keystroke of Ctrl+O

The Close command

A separator bar

The Exit command

After you add these submenu items, you can hook up the menu commands to Click() event procedures that you write, as explained in the next section.

Adding pull-down items requires that you follow the same steps you followed when you added the menu bar items in the previous section. The difference is that the Menu Editor options that the previous section ignored, such as the Shortcut option, become more important because you'll apply some of these options to the pull-down menu items. Table 1.1a explains the remaining Menu Editor properties.

**Table the Menu Editor's remaining properties.**

|  |  |
| --- | --- |
| ***Property*** | ***Description*** |
| **Checked** | Indicates whether a menu item has a checkmark next to it. Generally, you'll add checkmarks to menu options that perform on or off actions, such as a View menu that contains a Highlighted command. The checkmark appears when you, at design time or through code, set the menu item's Checked property to True. The checkmark goes away (indicating that the item is no longer active or selected) when you set the Checked property to False. |
| **HelpContextID** | This is a code that matches a help file description if and when you add help files to your application. |
| **Index** | If you create a menu control array rather than name individual menu items separately, this Index property specifies the menu item's subscript within the control array. |
| **Shortcut** | This is a drop-down list of Ctrl+keystroke access keys that you can add to any pull-down menu item. |
| **Window List** | Specifies whether the menu item applies to an advanced application's MDI (multiple-document interface) document. The menus that you create for this book don't require the use of MDI features. |

Perhaps the most important command keys on the Menu Editor, when you add pull-down menu items, are the four arrow command buttons. The left and right arrow command buttons indicate which items go with which menu bar option. In other words, if four items in the lower window are indented to the right and appear directly beneath the File menu bar item, those four indented items will appear on File's pull-down menu. The left arrow removes an indentation level and the right arrow adds an indentation level. The up- and down-arrow keys move menu items up and down the list of menu items, rearranging the order if you need to do so.

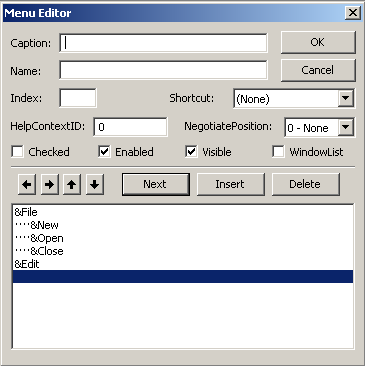
The arrow keys make a lot of sense when you see them used. Follow these steps to create the File pull-down menu bar's submenu:

**1.** Move the lower window's highlight line to the &Edit menu bar item. Click the Insert command button. You always insert before an item, so to add items to the File menu, you must insert before the Edit menu bar item in the lower window.

**2.** Click the right-arrow command button. Visual Basic adds four dots (similar to an ellipsis), showing that the newly inserted item will be indented under the File option.

**3.** Move the focus to the caption prompt and type &New.  
 **4.** Press Tab to move the focus to the name prompt and type mnuFileNew.  
 **5.** Click Next and then Insert, and press the right arrow command button to insert another item beneath the New item. Your Menu Editor should look like the one in

Figure. Notice that the File menu bar option now has a pull-down menu; you know this because of the indentation of the New option right below &File.



**Menu Extras**

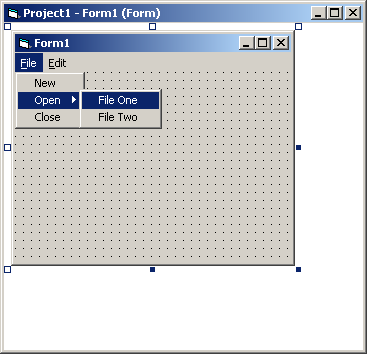
You don't need to complete all the menu bar options. You already know how to add routine options. If you need to add additional menu elements, however, such as a submenu or a checked item, the mechanics of those additions are about as simple as the items that you added in the previous sections.

To practice adding a checked object, add one checked item to the View pull-down menu bar item. Add an indented option that uses Highlighted for the Caption item and mnuViewHighlighted for the Name. Click the Checked check box. The View | Highlighted option will initially be checked when the user displays the View pull-down menu. Your code can check and uncheck the item by changing the mnuViewHighlighted object's checked property to True and False.

If you want to add a submenu from a pull-down menu item, add an additional level of indentation. For example, to add a two-option submenu off the File | Open option that gave the user an additional choice of Binary or Text (binary and text are two possible kinds of files), insert a place for the first item right beneath Open and click the right-arrow command button to add a second ellipsis. Type &Binary for the Caption property and mnuFileOpenBinary for the Name property. Insert an additional item beneath that, indented at the same level, and type &Text for the Caption property and mnuFileOpenText for the Name property.

Now that you've completed the menu (as far as we're taking it here), click the OK command button. When the Menu Editor disappears, you'll see the application's Form window with the menu bar across the top of the screen. Open the File menu and then select Open to see the submenu like the one shown in Figure 17.8. Notice the right arrow next to Open, which indicates that an additional submenu will appear for that option.

The File menu is now complete.



## Chapter: 8 Accessing Data

**Database Management**

A database can be called as set of interrelated data. Database management deals with the storage, retrieval and editing of the data within the database. One of the most important functions of Database management is the Database security and database validations.

Visual Basic makes it simple by supplying a variety of tools and approaches for data management. For significant database applications, utilizing database functionality that already exists becomes necessary. Visual Basic 6.0 provides full-fledged database engine called Jet. This engine contains virtually all the database functions that applications may need. The simplest and easiest way to access the Jet database engine is to through bound data control.

Data Access Object (DAO) has been the standard database access method for Visual Basic programmers. DAO was the first database interface included with Visual Basic in its third version. Remote Data Object (RDO) was then introduced to make it more edible and functional in VB4. The most recent release by Microsoft with Visual Basic is the ActiveX data object, which is in a short while going to take over all the other methods of data access.

**Data Access Object**

Data Access Object enables you to use a programming language to access and manipulate data in local or remote databases and to manage databases, their objects and their structure. DAO utilizes Microsoft's jet database engine and can be used to access a wide variety of database formats, including, ISAM (Indexed Sequential Access Method) Text.

In addition to this, DAO can also make use of Open Database Connectivity (ODBC) databases through its OBDC Direct Object Model.

Generally ODBC is a special technique to access a Database residing on a separate Server such as Informix, Oracle etc.

The DAO object model has 17tyyes of objects like Databases, Recordsets, and Fields etc. Databases are accessed by opening an existing Database object or creating a new one. The individual fields in a-table make up a Recordset Field object and table of a database is held in a Recordset.

**Object**

Data Access object is implemented through an ActiveX Control called the Data Control. This allows a large amount of DAO's power to be utilized without having to add a single line of code. When used in conjunction with Data aware controls such as Textbox, DBList, DBCombo, DBGrid etc, the Data Control becomes very powerful. Information from a database table is fetched by the DAO and then supplied to various Data aware controls like TextBox, DBListBox etc.

**Data Binding**

In order to display the values of the columns accessed by the Data Control, some controls are required which support the data control. These controls are called Data Bound controls and are simpler to use e.g. TextBox, Label DBGrid, DBList etc.

These controls have basically two standard properties other than their regular properties. The DataSource property enables you to specify the name of the data Control to which the client Control wi1l bind. The name of the Data Control(s) available on your form is already present in the dropped down ComboBox of this property in the property window.

The Data field property, a string property of the Client Control, specifies which of the Database field names available from the data control the client will bind.

After the Data Source and Data field properties are set the Data Control is properly initialized, the Client Control automatically displays the contents of the bound field as the data controls is used to navigate though the recordset. You call then use an appropriate editable control, such as the textbox, to change the Data Contained in the Bound field. DBCombo and DBlist enable you to use the data control to provide a list of valid choice for a field, as well as to bind the client control to a specific field.

**Data Control and the Data Bound Controls**

Visual Basic provides many sources to Connect to a data source. Data Control is one of them.

DataControl was the first medium to connect the applications to a database in the third version of Visual Basic.

The Data control is an excellent tool for rapidly developing database information retrieval system. With very little code you can get a useful view into stored information. This is practically a drop-in-place process where by you can place a data Control, a DBGrid and a Command

Button on a form and be able to access your database in a matter of minutes. After you set Properties your application is set to go.

**Data Control**

Data Control is one of the easiest modes to access databases. You can perform most data access operations using the Data control without writing any code at all. Data-aware controls bound to a Data control automatically display data from one or more fields for the current record or, in some cases, for a set of records on either side of the current record. The Data control performs all operations on the current record.

If the Data control is instructed to move to a different record, all bound controls automatically pass any changes to the Data control to be saved in the database. The Data control then moves to the requested record and passes back data from the current record to the bound controls when it's displayed.

The Data control is itself smart enough to handle a number of contingencies including empty recordsets, adding new records, editing and updating existing records, and handling some types of errors. However, in more sophisticated applications, you need to trap some error conditions that the Data control can't handle. For example, if the Microsoft Jet database engine has a problem accessing the database file, doesn't have permission, or can't execute the query as coded, a trappable error results. If the error occurs before your application procedures start or due to some internal errors, the Errors event is triggered.

**Data Control in Action**

Once the application begins, Visual Basic uses data control properties to open the selected database, create a Database object and create a Recordset object. The Data control's Database and Recordset properties refer to the newly created Database and Recordset 0bj etc which may be manipulated independently of the Data control - with or without bound controls. 1J.le Data control is initialized before the initial Form Load event for the form on which it is laced. If any errors occur during this initialization step, a non-trappable error results.

When Visual Basic uses the Jet database engine to create a Recordset, no other Visual Basic operations or events can occur until the operation is complete. However, other Windows based applications are permitted to continue executing while the Recordset is being created. You can use CTRL-Break to terminate the operation while the jet engine is building the Recordset for the specified table. The Recordset property of the Data control is set to nothing.

When you use a Data control to create a Recordset object or when you create a Recordset object in code and assign it to the Data control, The Microsoft Jet database engine automatically populates the Recordset object.

The Data control can be manipulated with the mouse, moving from record to record or to the beginning or end of the Recordset. The EOFAction and BOFAction properties determine what happens when the user moves to the beginning or end of a Recordset with the mouse. You can't set focus to the Data control using the SetFocus method.

Being a developer you must always be conscious that only valid data must be stored in the tables, otherwise it may result in unexpected results while processing the Data. You can use the Validate event and the DataChanged property to perform last minute checks on the recordset being written to the database.

**End of File and Beginning of File Actions**

The Data control can also manage what happens when you encounter a Recordset with no records. By changing the EOFAction property, you can program the Data control to enter Addnew mode automatically. These properties have been discussed later.

Alignment you can program the Data control to automatically snap to the Top or bottom of its parent form by using the Align property. In either case, the Data control is resized horizontally to fill the width of its parent forms whenever the parent form is resized. This property allows a Data control to be placed on an MDI form without requiring an enclosing Picture control.

**Data Binding**

In order to display the values of the columns accessed by the Data Control, some controls are required which support the data control. These controls are called Data Bound controls and are simpler to use e.g. TextBox Label DBGrid, DBList etc.

These controls have basically two standard properties out of their regular properties. The DataSource property enables you to specify the~ name of the Data Control to which the client control will bind. The name of the Data Control(s) available on your form is already present in the dropped down Combo Box of this property in the property window.

The Datafield property, a string property of the Client Control, specifies which of the Database field names available from the data control, the client will bind.

After the DataSource and Datafield properties are set, the Data Control is properly initialized, the Client Control automatically displays the contents of the bound fields as the data control is used to navigate though the recordset you can use an appropriate editable control, such as the Textbox, to change the Data contained in the Bound field. DBCombo and DBList enable you to use the data control to provide a list of valid choice for a field, as well as to bind the client control to a specific field.

**Properties of the Data Control**

BOF Action and EOF Action Properties

As discussed earlier the BOFAction and EOFAction properties return or set a value indicating what action the data control takes when the BOF or EOF is encountered by Data control.

Syntax:

DataControlName.RecordsetBOFAction =value

DataControlName.RecordsetEOFAction =value

Different settings for the properties are :

|  |  |  |
| --- | --- | --- |
| **Constant** | **Value** | **Description** |
| VbBOFActionMoveFirst | 0 | Moves the record pointer back to the first valid record. |
| VbBOFActionBOF | 1 | Causes a validation event on the first record repositions to the first record. |

Above given values are for the BOFAction property, values for EOFAction property are:

|  |  |  |
| --- | --- | --- |
| **Constant** | **Value** | **Description** |
| VbEOFActionMoveLast | 0 | Moves the record pointer back to the last record. |
| VbEOFActionEOF | 1 | Causes a validation event on the last record, repositions to the last record. |
| VbEOFActionAddNew | 2 | After moving past the last record, triggers an AddNew event after Validation event. |

**Connect Property**

The Connect Property sets or returns a Value that provides information about the source of an open connection, an open database, a database used in a pass through query, or a linked table. This property is read/write for database objects, new coill1ectionobjects linked tables and tabledef objects. This property is read only for Querydef objects and base tables.

Syntax:

Object.Connect = databasetype.parameters

Database type specifies the type of database. This parameter can be excluded for Microsoft Jet Databases.

**Database Property**

This property returns a reference to a database connected through a data control.

Syntax:

Object.Database

The Database object created by the Data control is based on the control’s Database Name, Exclusive Read Only and connects properties. Database objects have properties and methods you can use to manage your data. You can use any method of a Database object with the Database property of a Data control such as close and execute. You can also examine the interval structure of database by using its TableDefs collection and in turn, the Fields, Indexes Collections of individual TableDef objects.

**DatabaseName Property**

The DatabaseName Property reaps or sets the name and location of the database. Setting the Value in the data control's dabaseName property while your program is running, changes the Constant Value database to which the control is connected. You must use the Refresh method to reopen the control after changing databases for a data control.

Syntax:

DataControlName.DatabaseName =PathName

The following example closes a database refined by DBName variable after confirming from the user.

Private Sub OpenDatabaseNew (DBName as String)

aDS=Msgbox (IClose"+Data1.DatabaseName+"Now", VBYesNo,

"Confirm")

If aDS=VBYes Then Data1.DatabaseName =DBName

Datal.Refresh

End Sub

**Exclusive Property**

The Exclusive property reads or sets whether a database is opened for single user or multi-user access. The data control exclusively reads or writes the database at hath design time an runtime, if set to True

Syntax:

ObjectExclusive =True / False

**Read-Only Property**

Read-Only Property returns or sets a value that determines whether the control's Database is opened for read-only access.

Syntax:

objectReadOnly [=boolean]

The object expression evaluates to a data object and a Boolean expression determines read/write access.

The control's Database object is opened with read-only access when set to True. Changes to data aren't allowed. The control's Database is opened with read/write access to when set False.

You can use the Read-Only property with a Data control to specify whether data in the underlying Database can be changed. For example, you might create an application that only displays data.

Accessing a Database using read-only results in faster access.

For a Data control, this property is used only the first time a database is opened by your application. If your application subsequently opens other instances of the database, the property is ignored. For a change in this property to take effect, your must close all instances of the database and then use the Refresh method.

**Recordset Property**

The Recordset property sets or returns a Recordset object defined by a Data control's properties or by an existing Recordset object.

You can use the Recordset property like any other Recordset object. For example, you can use any of the Recordset methods or properties to examine the structure of the Recordset objects underlying schema.

You can also request the type of Recordset to be created by setting the Data controls Recordset Type property. If you don't request a specific type, a dynaset -type Recordset is created.

Using the RecordsetType property, you can request to create either a Table-, Snapshot or dynaset type Recordset.

The syntax for the RecordSet property is as follows:

Set Object.RecordSet =value

Object refers to a valid Data control name and value is any object variable containing a reference to a RecordSet object.

**RecordSource Property**

The RecordSource Property specifies the source of the records accessible through bound controls on your form.

Syntax:

Object.RecordSource = [SQL Query| Table Name | Procedure Name]

The object expression evaluates to a Data control. Values can be the name of any table in the database specified by the DatabaseName property of the Data control or any table defined in the database object's TableDefs collection. Value can be also a string variable or a string containing a SQL query. It can also be the name of any stored procedure in the Database. If you set the RecordSource property to the name of an existing table in the database, all of the fields in that table are visible to the bound controls attached to the Data control. For table-type recordsets (RecordsetType = vbRSTypeTable), the order of the records retrieved is set by the Index object that you select using the Index property of the Recordset. For dynaset-type and snapshot-type Recordset objects, you can order the records by using a SQL statement with an Order By clause in the RecordSource property of the Data control. Otherwise, the data is returned in no particular order.

If you set the RecordSource property to the name of an existing Querydef in the database, all fields returned by the QueryDef are visible to the bound controls attached to the Data control.

The order of the records retrieved is set by the QueryDef object's query. For example, the QueryDef may include an OJSPER BY clause to change the order of the records returned by the Recordset created by the Data control or a WHERE clause to filter the records.

You must use the Refresh method on a Data control to reopen the database if the RecordSo.urce, property settings have changed and rebuild the dynaset in the control’s Recordset property.

Example:

Data1.RecordSource ="Select \* from Emp where EMPID >= 1001"

Data1.Refresh

**The Refresh Method**

The *refresh* method when applied to a data control opens or reopens a database specified in the DatabaseName property of Data control.

Syntax:

Object.Refresh

This *refresh* method also resets the current record back to the first record or (row) in the table.

**Important Events of the Data Control**

**Error Event**

This event occurs only as the result of a data access error that takes place when no Visual Basic code is being executed. The general form of this event procedure is

Private Sub object\_Error ([Index As Integer,] dataerr As Integer, response As

Integer)

End Sub

Index identities the control if it's in a control array, contains the index number of the Data control.

Dataerr returns the error number and response is a number corresponding to the response you want to take, as described below:

|  |  |  |
| --- | --- | --- |
| **Constant** | **value** | **Description** |
| DataErrContinue | 0 | Ignores the error and continues the execution of the program. |
| VbDataErrDisplay | 1 | Displays the Error message. This is by default |

Run-time errors can occur when none of your code is running, as when:

A user clicks a Data control button.

The Data control automatically opens a database and loads a Recordset object after the Form\_Load event.

A custom control performs an operation such as the MoveNext method, the AddNew method or the Delete method.

If an error results from one of these actions, the Error event occurs.

If an event procedure has not been written for the Error event, Visual Basic itself displays the message associated with the error.

**Reposition Event**

Reposition event occurs after a record becomes the current record. When a Data control is loaded, the first record in the Recordset object becomes the current record. Whenever a user clicks any button on the Data control, moving from record to record, or if you use one of the Move methods, such as MoveNext, the Find methods, such as FindFirst, or any other property or method that changes the current record, the reposition event occurs after, when each record becomes current. Delete, Unload, Update are few actions causing this event. The general form of this event is:

Private Sub Object. Reposition ([lndex As Integer])

End Sub

Index specifies the index number of the control if it is a part of control array. In contrast, the Validate event occurs before moving to a different record. You can use this event to perform calculations based on data in the current record.

**Validate Event**

The Validate event occurs before the reposition event (or a different record becomes the current record). Validate event procedure has two arguments Index number as integer for the Data.

If it is a part of a Data control array, Save a Boolean expression specifying whether bound Data has changed or not and Action an integer specifying operation that causes this event. Values for action are:

|  |  |  |
| --- | --- | --- |
| **Constant** | **Value** | **Description** |
| VbOataActionCancel | 0 | Cancels the operation when the sub exits. |
| VbOataActionMoveNext | 3 | Caused due to MoveNext method. |
| YbDataActionMoveLast | 4 | Caused due to MoveLast method. |
| VbDataActionAddNew | 5 | Caused due to AddNew method. |
| VbDataActionUpdate | 6 | Caused due to Update method. |
| VbDataActiOlillelete | 7 | Caused due to Delete method. |
| VbDataActionFind | 8 | Caused due to Find method. |
| VbOataActionUnload | 11 | Caused due to FOTInbeing unloaded. |

This event occurs even if no changes have been made to data in bound controls and even if no bound controls exist. You can use this event to change values and update data. You can also choose to save data to stop whatever action is causing the event to occur and substitute a different action.

You can change the action argument to convert one action into another. you can change the various Move methods and the AddNew method, which can be freely exchanged. Any action can be stopped by setting action to 0.

You can check the data in each bound control where DataChanged is True. You can then set DataChanged to False to avoid saving that data in the database. If it is invalid or corrupted, you can't use any methods such as MoveNext on the underlying Recordset object during this event.

**Using Objects**

Visual Basic allows you to create objects to represent a database, Recordsets, Fields etc. This makes the job easier in programming a Database application. You can then easily manipulate the objects or change the DataSource according to your requirement in run-time. Data Access Object (DAO) has been the simple Database access methods for Visual Basic programmers. You can use the Database and RecordSet objects. The RecordSet and Database objects, each has property and methods of their own. You can either use a Database, RecordSet object created by a Data control or create an instance of your own, to manipulate the Data.

**Database Object**

A Database object represents a Database. A Database object contains RecordSet, QueryDef, and Procedures. A Database object is declared within a Workspace object, which is a part of Work spaces collection.

A database object can be declared using the following code:

Dim MyDb As Database

Dim MyWrk As WorkSpace

Set MyWrk =WorkSpaces(O)

Set MyDb =MyWrk.QpenDatabase ("C:\ProgramFiles\MicrosoftOffice\Office\Samples\-

Northwind.MDB ").

The opendatabase method of the Workspace object opens a database and returns its reference to a valid Database type object variable.

Important Methods of a Database Object Are

Close Closes an open Database Access object.

Execute Runs an Action Query or executes an SQL statement

ExecuteSQL Executes an Action Query as an SQL statement on an

ODBC Database.

Open Recordset Creates a new Recordset object and applies it to the

Recordsets collection.

Refresh Refresh method closes or rebuilds the Recordset object.

**Important Properties of the Database Object**

Connect Specifies the source of an open database. The default

source for ODBC and certain ISAM drivers in Jet databases.

Count Returns the number of objects in a collection.

Name Returns or sets a user defined name for data access objects.

**RecordSet Object**

A RecordSet object represents the entire set of records from a base table or the set of records resulted by an executed query. At any time, RecordSet object refers only to a single record within the set as the current record.

A RecordSet object can be used to manipulate, and access the data. In order to navigate records in the Database you can use various methods of the RecordSet object.

The number of records retu111edby the Recordset can be determined by moving to the last record in the Recordset or by examining the Recordset object's Record Count property. Before you move to the last record, the value retu111edby the property to display the number of records in a Data control's RecordSet:

Data1.RecordsetMoveLast

It MsgBox "Records:" & Datal.RecordsetRecordCount

If you create a Recordset object using either code for, another Data control, you can set the Recordset' property of the Data control to tells new Recordset Any existing Recordset in the Data control and the Database object associated with it are released when a new Recordset is assigned to the Recordset property.

The five types of RecordSet object are:

* TabletypeRecordSet
* Dynaset type RecordSet
* Snapshot type RecordSet
* Forward only type RecordSet
* Dynamic type RecordSet

All the above mentioned RecordSet objects represent a set of records. Let us discuss these different types of RecordSets.

Table Type RecordSet Object

The first type of RecordSet object is the Table-type RecordSet. A table type object can be created from a table in the Microsoft Access database, but not from an Open Database Connectivity (ODBC) or a linked table. When you create a table-type the Jet database engine opens the actual table and your subsequent data manipulations operate directly on base table data. A table-type can be opened on only one table; it Validate opened on a union query or a select query with a join.

The best part of this type of object is that you can use index created for the underlying table. Tells allows much faster sorting and searching than the other types.

**DynaSet Type RecordSet Object**

Dynaset-type Recordset object represents a result of a query that can have updatable records. A dynaset-type object can be created from either a local or a linked table, or with a row-returning query that joins tables. It's actually a set of references to records in one or more tables. With a dynaset you can extract and update data from more than one table, including linked tables from other databases. Heterogeneous updatable joints are an unique feature of dynaset-they enable you to use updatable queries against tables in different type of databases. You can use the Seek methods rather than Find methods for a Dynaset-type Recordset object for better performance.

**SnapShot Type RecordSet Object**

A Snapshot-type object is a static copy of a set of records as it exists at the time the snapshot is created. A snapshot-type object can contain fields from one or more tables in a database. You can't update a snapshot. –The main advantage of asI; apshot is that it creates less processing overheads then the other types, so it can run queries under took data f<lster, especially when working with ODBC data sources.

**Forward Only**

A forward-only type object is identical to a SnapShot type object. A forward-only-type object, sometimes referred to as a forward-scrolling snapshot or a forward-only snapshot, provides a subset of the capabilities of a snapshot. With forward-only snapshots, you can move only in a forward direction though the records. Objects of the type cannot be cloned and only support the Move and MoveNext methods. Like Shan shots, you can't update a forward-only-type object.

**Dynamic Type RecordSet Object**

Dynamic type RecordSet object is a result of query from one or more base tables. You can add, change or delete records from a row-returning query. Besides this, records other users add, delete or edit in the base table are also present in the RecordSet.

Although you Call work with any of the above RecordSet object, you must always choose the correct RecordSet type that suits or satisfies your desires. For example you would like to work with the sorted data or a data where searching operations like FindFirst, Seek could be performed in a short span of time, then you will go for indexed table type RecordSet. Similarly, in order to scan all the Records you must use a Forward only type RecordSet for better performance.

You can use the type argument of the RecordSet object to choose the RecordSet type.

**Important Methods of the RecordSet Object**

Addnew

The *MS Access* engine maintains a copy buffer where it keeps the pending data that it will be writing to the database. The addnew method clears the copy buffer and moves the current record to the end. This is like adding a new row.

Syntax:

Datacontrolname. Recordset.Addnew.

The *Addnew* method does not actually add the information to the database. This is done by the *UPDATE* method.

**Update**

This method sends the contents of the copy buffer to the table or dynaset

Syntax:

Datacontrolname. Recordset.Update

Example:

Data.Recordset.Addnew.

Data.Recordset.Fields ("Author") = “Homes”

Datal.Recordset.Fields ("Titles") ="Iliad"

*Datal Recordset.Update*

This code adds a record to a table which has two fields *Authors* and *Title.*

Update controls. This method can be used to retain the original data in the bound controls and ignore the changes made to the data by a user.

e.g. This method can be used in the *Cancel* button to reset the contents of bound controls to their original value.

**Edit**

This method copies the current record into the copy buffer for editing.

Example:

Datal.RecordsetEdit

Authomame=Datal.RecordSetFields ("Author")

Call Changefield (Authomame)

Datal.Recordset.Fields ("Author") =Authomame

*Datal Recordset.Update*

This code uses a user defined procedure called *Changefield,* which changes the value of the field.

**Delete**

This method deletes the current record in the RecordSet.

Syntax:

Datacontrolname.Recordset. Delete

After delete you must move the record pointer away from the deleted record by a move method.

Example:

Datal. Recordset. Delete

*Datal. Recordset. Moveprevious*

**Close**

To close an open database attached to a data control or to close the specific RecordSet currently attached to the control, use the following:

Syntax:

ObjectName. Close

*Object Name* can be an open database, record set, dynaset, snapshot or table object. You must use the update method before you dose the database to save all the pending changes made to the database.

**UpdateRecord Method**

Saves the current values of bound controls. Doesn't support named arguments.

Syntax:

object.UpdateRecord

The object represents a RecordSet object.

Remarks:

Use this method to save the current content of bound controls to the database during the Validate event without triggering the Validate event again. Using this method avoids creating a cascading event. The UpdateRecord method has the same effect as executing the Edit method, changing a field and then executing the Update method, except that no events occur. You can use this method to avoid triggering the Validate event.

**Summary of the Methods of RecordSet Object**

|  |  |
| --- | --- |
| **Name** | **Description** |
| Properties | Contains all the property objects for a specific instance of the  Recordset object. |
| Fields | Contains all stored Field objects of the Recoredset object. |
| AddNew | Creates a new record for an updatable Recordset object |
| CancelUpdate | Cancels any changes made to the current record or to a new  record prior to calling the Update method. |
| Clone | Creates a duplicate Recordset object from an existing  Recordset object. |
| Close | Closes the open Recordset object and any dependent objects. |
| Delete | Deletes the current record or a group of records. |
| GetRows | Retrieves multiple records of a Recordset object and makes  that record the current record. |
| Move | Moves the position of the current record in a Recordset object. |
| MoveFirst | Moves to the first record in the Recordset object and makes  that record the current record/ |
| MoveLast | Moves to the last record in the Recordset object and makes  that record the current record. |
| MoveNext | Moves to the next record in the Recordset object and makes that record the current record. |
| MovePrevious | Moves to the previous record in the Recordset object and makes that record the current record. |
| NextRecordset | Advances through a series of commands by clearing the current Recordset object and returns the next RecordSet. |
| Open | Opens a cursor |
| Update | Saves any changes you make to the current record of a Recordset object. |
| UpdateBatch | Writes all pending batch updates to disk |

**Properties of the Recordset Object**

|  |  |
| --- | --- |
| **Name** | **Description** |
| AbsolutePage | Sets the "page" number on which the current record is located. uses the PageSize property to logically divide the RecordSet object into a series of the pages, Name Description each of which has the number of records equal to pageSize. The provider must support the appropriate functionality for this property to be available. |
| AbsolutePosition | Sets the ordinal position of a Recordset object's current record. |
| ActiveCOlmection | Shows to which Connection object the specified Recordset object Currently belongs. |
| BOF | Shows that whether the current record position is before the first record in a Recordset object, or not. Returns True or False. |
| Bookmark | Returns a bookmark that uniquely identifies the current record in a Recordset object. Sets the current record in a Recordset object or the record identified to be a valued bookmark. |
| CacheSize | Shows the number of records from a Recordset object that are cached locally in memory. |
| CursorLocation | Sets or retums the location of the cursor engine. |
| CursorType | Shows the type of cursor used in a Recordset object-for example, Read Only or Updatable. |
| EditMode | Shows the editing status of the current record. |
| EOF | Shows that whether the current record position is after the last record in a Recordset object, or not. Returns True or False. |
| Filter | Shows that the current record position is after the last record in a Recordset object. |
| LockType | Shows the type of locks placed on records during editing. |
| MarshalOption | Indicates which records are to be marshaled back to the server. |
| MaxRecords | Shows the maximum number of records to return to a Recordset from a query. |
| PageCount | Shows how many pages of data the Recordset object contains. |
| PageSize | Shows how many records constitute one "page" in the Recordset. |
| RecordCount | Shows the current number of records in a Recordset object. |
| State | Describes the current state of an object. |
| Status | Shows the status of the current record with respect or batch updates or other bulk operations. |

You can also declare a RecordSet type object variable using the following code:

Dim MyTable As RecordSet

A reference to a RecordSet can be given to MyTable using the following line.

Set MyTable =DB.OpenRecordSet ("Authors")

or your new RecordSet can be result of a new query.

Qry ="Select \* from Authors where Su\_ID >= 10"

Set MyTable = DB.OpenRecordSet (Qry, dbOpenDynaset)

**Field Object**

Field object represents columns of data with a common data type. The Fields collection contains all stored Field objects of a Recordset object. The Field object contains one collection; the Fields collection. The Field object contains the methods and properties given in the following table.

**Methods**

|  |  |
| --- | --- |
| **Name** | **Description** |
| AppendChunk | Appends data to a large Text or binary data field object. |
| GetChunk | Returns all or a portion of the contents of a large text or binary data Field object. |

**Property**

|  |  |
| --- | --- |
| **Name** | **Description** |
| ActualSize | Shows the actual length of a field’s value. |
| Attributes | Shows one or more characteristics. |
| DefinedSize | Shows the defined size of a field object. |
| Name | Shows the defined Name of a field object. |
| NumericScale | Returns a Byte value indicating the number of decimal places to  which the numeric value will be resolved. |
| OriginalValue | Shows the value of a Field that existed in the record before any  changes were made. |
| Precision | Sets or returns a Byte Value, indicating the maximum total number of digits used to represent values. |
| Type | Returns the Data type |
| UnderlyingValue | Shows a Field objects current value in the database. |
| Value | Shows the value assigned to a Field object. |

The Fields collection contains the count and Item properties which return the total number of fields and reference to a particular Field respectively.

**Remote Data Object**

Remote Data object commonly known as RDO is similar to DAO except it is faster than DAO and uses less workstation resources. It allows the programmer more control over collections to Client Server databases and lets him fully utilize the power of ODBC.

Since RDO is used primarily by professional programmer for building enterprise applications, it is only available with the Enterprise Edition of Visual Basic. The RDO model consists of total of nine different object types. A collection to a database is referenced by RDO Connection object. This can contain rdoTable object and rdoResultset objects. Other object types include rdoQuery and rdoColumn, which is similar to DAO's field object.

RDO uses the Remote Data Control like DAO uses the Data Control. Data Aware controls like TextBoxes, List boxes etc can be bound to it.

**Activex Data Object**

Microsoft's ActiveX Data object (ADO) is the application programming interface used to access information. This interface is made accessible through a vendor specific OLEDB driver, called provider, to allow for a smooth transition to this new data access strategy. Microsoft provides OLEDB provider for ODBC that enables you to access ODBC data sources programmatically.

ADO provides the means for you to perform the following action:

1. Specify a command to COlli1ectto a Data source, with variable parameters, or

optionally optimized for performance.

2. Execute the command.

3. Connect to a data source.

4. Stores the result of the command to evaluate, manipulate or change.

5. If appropriate, update the data source with changes from the cache of rows.

6. Provide a general procedure for error handling.

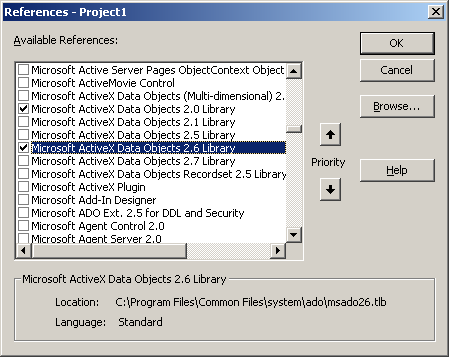
The ActiveX Data Object Libraries

The ADO data control is supplied in two separate Libraries with Visual Basic 6.0. These libraries are:

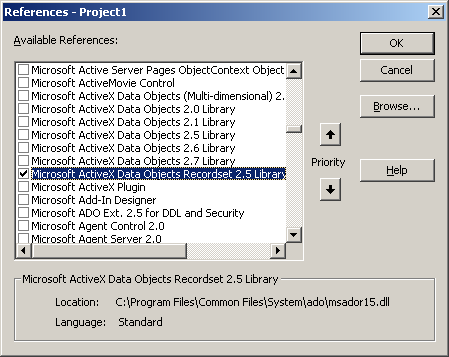
* ADODB
* ADOR

The ADODB library appears in the Reference dialog box with the name of the Microsoft ActiveX data object 2.0 Library. .

You must add a reference to these libraries using Reference dialog box from project menu before using them.



This library contains all of the ADO objects and will most likely be suitable for the majority of your projects. The ADOR Library also known as Microsoft ActiveX Data object Recordset 2.0 Library is used for those projects where a much less resource intensive ADO is required. This library contains support for only Recordsets contributing to a decrease in resource requirements.



In order to access and use all of the ADO objects, methods and properties and display them through the VB's object Browser, any of the two libraries should be selected from the reference dialog box.

**The ActiveX Data Object - Object Model**

The ADO Object Model defines a collection of programming objects that support the component object Model (COM) and OLE Automation to leverage the powerful partner technology called OLEDB.

The ADO object model, when compared with its counterpart the Data Access objects such as RDO or DAO is easier and simpler to use.

ADO concisely consists of seven objects, and four collections. ADO 2.0 introduces the concept events to the programming model. There are two families of events, Connections events and Recordset events. Connection events are issued when transactions on a collection begin, are committed *or* rolled back *or* when commands execute. RecordSet events are issued to report the progress of data retrieval when you navigate through the rows *of* a RecordSet and when any change is made within the RecordSet.

ADO's events are processed by event handler routines, which are called before *or* after certain operations. Some events are called before an operation starts, and are called as Will events and have the names in the form of WillAnEvent, example WillExecute which triggers before connection will start. Similarly events called after an operation concludes are called as complete events and have names of the form AnEvent complete, example ConnectComplete triggers when connection has started.

**Command Object**

A Command object is the definition of a specific command, such as SQL statement or stored procedure that you execute on a specific data source. The command object contains the collection, methods and properties as described in the table below.

**Collections of the Command Object**

|  |  |
| --- | --- |
| **Name** | **Description** |
| Parameters | The collection of parameter objects for the command. Parameter  Object represents a parameter or argument associated with a  Command object that is based on a parameterized query or stored procedure. |
| Properties | The set of properties for the Command object. |

**Methods of the Command Object**

|  |  |
| --- | --- |
| **Name** | **Description** |
| CreateParameter | Creates a new Parameter object. |
| Execute | Executes the query, SQL statement, or stored procedure specified in the CommandText property. |

**Properties of the Command Object**

|  |  |
| --- | --- |
| **Name** | **Description** |
| ActiveConnection | Indicates the Connection object to which the specified  Command object currently belongs. |
| CommandText | Contains the text of the SQL statement that you want to  issue on a specific provider. |
| CommandTimeout | Determines the time to wait when attempting to establish a  connection before timing out and generating an error. |
| CommandType | Contains the type of the command for example, a stored  procedure. |
| Name | Indicates the name of the object. |
| State | Describes the current state of the object. |

**Connection Object**

Maintains connection information such as cursor type, connect string, database etc. Connection objects can be created independently of any other previously defined object. The Connection object represents a specific unique session with a particular data source. The Collection object contains the collection, methods, and properties as described below.

**Collection of the Connection Object**

|  |  |
| --- | --- |
| **Name** | **Description** |
| Errors | Contains the errors generated on the connection. |
| Properties | The set of properties for the connection. |

**Methods of the Connection Object**

|  |  |
| --- | --- |
| **Name** | **Description** |
| BeginTrans | Starts a transaction if the database engine supports transaction processing. |
| Close | Closes the connection |
| CommitTrans | Commits and terminates the transaction. |
| Execute | Executes the specified SQL statement. |
| Open | Opens the connection to the data source. |
| RollbackTrans | Reverses a transaction. |
| OpenSchema | Returns a Recordset object that contains schema information. |

**Properties of the Connection Object**

|  |  |
| --- | --- |
| **Name** | **Description** |
| Attributes | Reads the XactAttributeEnum values for a Connection object. The value can be a sum of anyone or more of the XactAttributeEnum values. |
| CommandTimeout | Sets the time to wait for a command to execute before timing out and generating an error. |
| ConnectionString | Contains the information necessary to establish a collection, such as the DSN and password. You can even build a connection string using the property dialog box. |
| ConnectionTimeout | Determines the time to wait when attempting to establish a connection before time out and generating an error. |
| CursorLocation | Sets or returns the location of the cursor engine. |
| DefaultDatabase | Indicates the default database for the Connection object. |
| IsolationLevel | Reads or sets the isolation level of a transaction. |
| Mode | Indicates the permissions that are available in a connection for modifying data. |
| Provider | Indicates the OLEDB Data Provider for a Connection object. |
| Version | Indicates the ADO version number. |

**Error Object and Errors Collection**

An error object is generated each time an error is encountered in an ADO operation. The error object is part of an errors Collection. The Error object contains the properties as described below.

**The Error Object Contains Seven Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| Description | A string containing the description of the last error. |
| HelpContext | Returns a context ID, as a Long integer value, for a topic in a  Microsoft Windows Help file. |
| HelpFile | Returns a String that evaluates to a fully qualified path to a Help file. |
| NativeError | Indicates the provider-specific error code for a given Error object. |
| Number | Indicates the number that uniquely identifies an Error object. |
| Source | Indicates the name of the object that generated the error. |
| SQLState | Returns a five-character string that follows the ANSI SQL standard. |

The Errors collection has one method, the Clear method, which removes all of the Error objects in the collection. The Errors collection also has two properties: Count, which indicates the total number of Error objects in the collection, and Item, which returns a specific Error object.

**Field Object and Fields Collection**

The field object has been discussed earlier in the last chapter. Field object works in a similar manner as it works with the Recordset object.

The Fields collection contains one method, the Refresh method, which updates the objects in the Fields collection to reflect Field Objects available from, the provider.

The Fields collection contains the Count and Item properties.

**Parameter Object and Parameters Collection**

A Parameter object represents a parameter or argument associated with a Command object that is based on a parameterized query or stored procedure. The Parameter object contains one method, the AppendChunk method, which is used to append a large text or binary file to a parameter. The Parameter object contains the properties and methods as described below.

**Properties of the Parameter Object**

|  |  |
| --- | --- |
| **Name** | **Description** |
| Attributes | For the Parameter object, the value is the sum of anyone or more of the Parameter AttributesEnum values. |
| Direction | Shows whether the Parameter represents an input parameter, an output parameter, both, or a return value from a stored procedure. |
| Name | The name of a parameter that can be used in references to the parameter rather than its ordinal number in the collection. |
| NumericScale | Sets or returns the number of decimal places *for* numeric values. |
| Size | The maximum size, in bytes or characters, of a Parameter object. |
| Type | The data type of a Parameter object. |
| Value | The value assigned to a Parameter object. |

**Methods of the Parameter Object**

|  |  |
| --- | --- |
| **Name** | **Description** |
| Append | Appends a parameter object to the collection. |
| Delete | Removes a parameter object from the collection. |
| Refresh | Updates the parameter objects in the collection to show the object available from, and specific to, the provider. |

**Property Object**

A Property object represents a dynamic characteristic of an ADO object that is defined by the provider. ADO objects contain two types of properties: dynamic and built-in. Built-in properties are available to any object. They do not appear in the object's Properties collection preventing them from being modified or deleted.

Dynamic properties appear in the object's Properties collection. They can be referenced only through the collection. Dynamic property has four built-in properties derided as follows.

**Properties of the Property Object**

|  |  |
| --- | --- |
| **Name** | **Description** |
| Name | A String that identifies the property. |
| Type | An Integer that specifies the property data type. |
| Value | A Variant that contains the property setting. |
| Attributes | A Long that indicates characteristics of the property specific to the provider. |

**RecordSet Object**

You do not need to create a Connection object to create a Recordset object. This is accomplished by passing a connection string with the Open method. ADO creates a Connection object, but it isn't assigned to an object variable. If you are opening multiple Recordset objects, over the same collection, you must create and open a Connection object. This assigns the Connection object to an object variable. If you do not use this object variable when opening your Recordset objects.

**CREATING REPORTS**

In order to create a report, we can use the Microsoft data report tool. The

Microsoft Data Report designer is a versatile data report generator that features the ability to created banded hierarchical reports. Used in conjunction with a data source such as the Data Environment designer, you can create reports from several different relational tables. In addition to creating printable reports, you can also export the report to HTML or text files. The Data Report designer has several features:

**1. Drag-and-Drop Functionality for Fields --** Drag fields from the Microsoft Data Environment designer to the Data Report designer. When you do this,

Visual Basic automatically creates a text box control on the data report and sets the DataMember and DataField properties of the dropped field. You can also drag a Command object from the Data Environment designer to the Data Report designer. In that case, for each of the fields contained by the Command object, a text box control will be created on the data report; the DataMember and DataField property for each text box will be set to the appropriate values.

**2. Toolbox Control --** The Data Report designer features its own set of controls. When a Data Report designer is added to a project, the controls are automatically created on a new Toolbox tab named Data Report. Most of the controls are functionally identical to Visual Basic intrinsic controls, and include a Label, Shape, Image, TextBox, and Line control. The sixth control, the Function control, automatically generates one of four kinds of information: Sum, Average, Minimum, or Maximum. For more information about the Function control, see "Adding a Function Control to the Data Report."

**3. Print Preview**-Preview the report by using the Show method. The data report is then generated and displayed in its own window.

Note: A printer must be installed on the computer to show the report in print preview mode.

**4. Print Report** --Print a report programmatically by calling the PrintReport method. When the data report is in preview mode, users can also print by clicking the printer icon on the toolbar.

Note: A printer must be installed on the computer to print a report.

**5. File Export**-Export the data report information using the ExportReport method. Formats for export include HTML and text.

**6. Export Templates**-You can create a collection of file templates to be used with the ExportReport method. This is useful for exporting reports in a variety of formats, each tailored to the report type.

**7. Asynchronous Operation**-The DataReport object's Print Report and ExportReport methods are asynchronous operations: Using the Processing Timeout event, you can monitor the state of these operations and cancel any that are taking too long.

**Parts of the Data Report**

The Data Report designer consists of the following objects:

**1. DataReport object**---Similar to a Visual Basic form, the DataReport object has both a visual designer and a code module. Use the designer to create the layout of a report. You can also add code to the designer's code module to programmatically format controls or sections contained by the designer.

**2. Section object**-Each section of the Data Report designer is represented by a Section object in a Sections collection. At design time, each section is represented by a header that you can click to select the section and the section's pane where you can place and position controls. Use the object and its properties to dynamically reconfigure a report before it is built.

**3 Data Report Controls**--Special controls that only work on the Data Report designer are included with it. (Note: you cannot use Visual Basic's intrinsic controls, or any ActiveX controls, on the Data Report designer). These controls are found in the Visual Basic Toolbox, but they are placed on; a separate tab named "DataReport."

**Sections of the Data Report Designer**

The default Data Report designer contains these Sections:

* Report Header-contains the text that appears at the very beginning of a report, such as the report title, author or database name. If you want the Report Header to be the first page in the report, set its ForcePageBreak property to rptPageBreakAfter.
* Page Header-contains information that goes at the top of every page, such as the report's title.
* Group Header / Footer-contains a "repeating" section of the data report. Each group header is matched with group tooter. The header and footer pair is associated with a single Command object in the Data Environment designer.
* Details-contains the innermost "repeating" part (the records) of the report. The details section is associated with the lowest-level Command object in a Data Environment hierarchy.
* Page Footer-contains the information that goes at the bottom of every page, such as the page number.
* Report Footer-contains the text that appears at the very end of the report, such as summary information, or an address or contact name. The Report

Footer appears between the last Page Header and Page Footer.

**Data Report Controls**

When a new Data Report designer is added to a project, the following controls are automatically placed in the Toolbox tab named DataReport:

* Text Box Control (RptTextBox)- allows you to format text, or assign a

DataFormat.

* Label Control (RptLabel) - allows you to place labels on the report to identify fields or sections.
* Image Control (Rptlmage}-enables you to place graphics on your report.

Note that this control cannot be bound to a data field.

* Line Control (RptLine)-lets you draw rules on the report to further distinguish sections.
* Shape Control (RptShape}-enables you to place rectangles, triangles, or circles (and ovals) on a report.
* Function Control (RptFunction)-a special text box that calculates values as the report is generated.

**Select Project new and select DataProject**

Two designers (Data Report designer and Data Environment designer) and a standard form is added to the project.

**Alternate Method** We can start the procedure by creating a new Standard EXE project also. In which case a form will be automatically added. To add the Data Environment designer and Data Report designer, you must select the project menu, click Add Data Environment. If the designer is not listed on the Project menu, click Components. Click on the Designers tab, and click Data Environment to add the designer to the menu. Similarly to add the Data Report designer, select Project menu, click Add Data Report, and Visual Basic will add it to your project. If the designer is not on .the Project menu, click Components. Click on the Designers tab, and click Data Report to add the designer to the menu.

Note: The first four kinds of ActiveX designers loaded for a project are listed on the Project menu. If more than four designers are loaded, the later ones will be available from the More ActiveX Designers submenu on the Project menu.

Select the Data Environment designer from the project explorer. The connection object is available as Connection1. Right-click Connection1 and select Properties. The Data link properties dialog is displayed.

1. On the Data link Properties dialog box, click Microsoft Jet 3.51 OLE DB Provider. This selects the correct OLE DB provider for accessing a Jet database.

2. Click on the Next button to get to the Connection tab.

3. Click on the ellipsis button (...) next to the first text box.

Use the Select Access Database dialog box to navigate to the

STUDENT.mdb database.

4. Click OK to close the dialog box.

5. Right-click on the Connection1 icon, and click Rename. Change the name of the connection to Student.

6. Right-click on the Student icon, and then click Add Command.

Command1 is added. Right-Click on the Command1 object and select properties to display the Command1 dialog box. In the dialog box, set the properties as shown below:

|  |  |
| --- | --- |
| **Property** | **Setting** |
| Command Name | Stdmast |
| Connection | Student |
| DataBase Object | Table |
| Object Name | Stdmast |

7. Click OK to close the dialog box and Save the project.

Once the Data Environment designer has been created, you can create a data report. To do so, Select the DataReport designer in the project explorer window and set the properties of the DataReport object according to the table below:

|  |  |
| --- | --- |
| **Property** | **Setting** |
| Name | rptStudent |
| Caption | Student Data Report |

1. On the Properties window, click DataSource and then DataEnvironment1.

2. Then click DataMember and click Stdmast:

3. Right-click on the Data Report designer, and click Retrieve Structure.

You have added a new group section to the designer. Each group section has a one-to-one correspondence to a Command object in the data. Environment; in this case, the new Group section corresponds to the Stdmast Command object. Notice also that the Group Header has a matching Group

**Footer section**

From the Data Environment designer, drag the fields to the Report section.

Delete the label control that is attached to the textbox control.

Resize the Data Report designer's sections height - Details section - to be as short as possible. Any extra space below or above the text boxes will result in unneeded space in the final report.

**Save the project**

Preview the Data Report Using the Show Method

Now that the data environment and the data report objects have been created, you are almost ready to run the project. One step remains to write code to show the data report.

**To show the data report at run time**

1. On the Project Explorer window, double-click on the Form1 icon to display the Form designer.

2. On Toolbox, click on the General tab.

When you add a Data Report designer to your project, its controls are added to the tab named DataReport. To use the standard Visual Basic controls, you must switch to the General tab.

3. Click on the Command Button icon and draw a Command Button on the form. Set the properties of the Command1 control according to the table below:

|  |  |
| --- | --- |
| **Property** | **Setting** |
| Name | cmdShow |
| Caption | Show Report |

4. In the button's Click event, paste the code below.

Private Sub cmdShow\_ClickO.

rptStudent. Show –

End Sub

5. Save and run the project.

6. Click Show Report to display the report in print preview mode.

Optional-Setting the Data Report as the Startup Object

**You can also display the data report with no code at all.**

1. On the Project menu, click Properties.

2. In the Startup Object box, select rptStudent.

3. Save and run the project.

Note if you use this method, you can remove the Form object from your project.

A calculated field is a field whose value is calculated as the report is generated.

For example, when estimating the tax on an order, you must multiply the total price by the local tax rate. But since local tax rates differ and is not stored in the database, the tax is generated as the report is created-a calculated field. A Total field that calculates the value of Quantity \* UnitPrice is a calculated field.

Let us add a calculated field to calculate the Registration fee of 5% of fee paid by the student. This involves the use of an SQL statement in the data environment's Command object to create the calculated field.

1. In the Data Environment designer, right-click on the command objects. Then click Properties to display the Properties dialog box.

2. On the General tab, click on the SQL Statement button, and add the

Following statement to the box:

Select st\_id, SLname, SCdob, SUees, SUees \* .05 As Registration from stdmast

3. The SQL statement multiplies the Fees value by 5% to create the Registration value--the calculated field. Also note that if the name of the table contains a space, then it must be enclosed by brackets- (stdmast].

4. Click OK to close the dialog box.

5. From the Data Environment designer, drag the fields on to the Detail section of the Data Report designer

6. On the Data Report designer, click on the Registration text box to select it.

On the Properties window, double-click DataFormat to display the Property

Pages dialog box.

7. In the Format Type box, click Currency. In the Symbol combo box, select the currency appropriate to your country.

8. Similarly, change the DataFormat property of the Fees text box to Currency.

9. Click on the Registration text box control to select it. On the Properties window, change the Alignment property to 1- rptJustifyRight.

10. Save and run the project.

**Grouping Information Based on a Field**

Information that is grouped can give the user a different perspective on data.

Working in tandem with the Data Environment designer, the Data Report designer gives you the ability to group data according to any field in a table. The marks data can be grouped according to studenUd or Subjectcode.

The Grouping feature of the Data Environment designer allows you to select a particular field in the table as the grouping field without having to create a new Command object. The steps to adding a grouping field include:

1. Creating a group field in the Data Environment designer.

2. Adding a Group Header/Footer to the Data Report designer to correspond to the new command.

3. Resetting the DataMember property of the data report to the new Grouping

Command object created in the data environment.

4. Dragging the group field from the data environment to the data report.

**Steps to add a grouping field to the Data Report designer**

1. In the Data Environment designer, right click on the Command object. Click

Properties to show the Properties dialog box.

2. Click on the Grouping tab.

3. Click Group Command Object.

4. In the Fields in Command box, double-click on the field to group on (e.g.

St\_id of stdmarks). Click OK to close the dialog box. An object stdmarks\_Grouping is added.

5. Right-click on the Data Report designer, and then click Insert Group

Header/Footer.

6. Click on the up arrow three times to insert the group header at the outer most, edge of the header/footer pairs. Click OK 10close the dialog box.

7. Click on the Data Report designer's title bar to select the entire data report.

On the Properties window, click DataMember and change the property from Stdmast 10 Stdmarks- Grouping.

When the grouping field was added to the data environment, the equivalent of a new Command object was also added to the data report. That virtual Command object is displayed in the drop-down list of data members as Stdmarks\_Grouplng.

8. In the Data Environment designer, open Summary Fields in stdmarks\_Grouping. Drag the ST\_10 field into the new section on the Data Report designer.

9. Delete the Label control that accompanies the SUd field. Place the new field at the leftmost edge of the designer.

10. Save and run the project.

**Adding a Function Control to the Data Report - to Print Sub-Totals and Grand Totals**

The Data Report designer features its own set of controls. The Function control displays data that is calculated at run time, using a built-in function, as the report is generated. The Function control can calculate values only after all other records in a group section have been processed. In contrast, a SQL statement calculates the values as part of the fields in a record as t:'ey are processed.

The steps to add a Function control to the Data Report designer are:

1. Draw a Function control in an appropriate Footer section of the Data Report designer.

2. Set the DataMember and DataField properties to appropriate values (a numeric field from a relevant data environment Command object.)

**Steps to add a function control to the data report**

1. Using the mouse pointer, select the group footer where the sub-total or grand total is to be printed. Drag it downward to create enough space.

2. On the Toolbox, click on the rptFunction control.

3. Draw the rptFunction control in the space between the two footers.

4. Set the properties of the rptFunction control according to the table below:

|  |  |
| --- | --- |
| **Property** | **Setting** |
| DataMember | Stdmarks |
| DataField | Marks |
| Name | FncTotal |
| Aligment | 1 - rptJustifyRight |
| DataFormat | Currency |

The FunctionType property of the control determines what operation will be performed with the data found in the DataField. By default, the property is set

To 0 - RptFuncSum, to sum the data. Other functions include: Average, Minimum, Maximum, Row Count, STDDev (Standard Deviation), and Value Count.

5. Set the Caption property of the label to Total.

6. Click on the fncTotal control to select it. On the Properties window, double click Font to display the Font dialog box. Change the Font style to Bold.

7. Save and run the project.

**Adding an Aggregate Field to the Data Report**

In the Data Environment designer you can also create aggregate fields—fields that summarize data from a section. An aggregate field is thus similar to the

Function control in that both are calculated as the report is generated. But there are a few differences: whereas the Function control can only be placed in a

Group Footer, an aggregate field can be placed in any section of the DataReport designer, except the Report Header / Footer and Page Header / Footer sections.

Another difference is found in how the two fields are created: while the Function control is a feature of the Data Report designer, the aggregate field is a feature of the Data Environment designer. The steps to adding an aggregate field to the report are:

1. Create an aggregate field in the Data Environment designer.

2. From the Data Environment designer, drag the aggregate field onto the Data Report designer.

For example, let us create an aggregate field that displays the number of Student.

**Data Report with Aggregate Field**

Using a Function control can be more efficient than an aggregate field. When the data report calculates the value for a Function control, it takes a certain amount of processing to create each value. When the data environment creates an aggregate field, it takes a similar amount of processing. However, you can save that: processing time by using the Function control because the Data Report designer must create the entire report in its own process.

**To Create an Aggregate field in the Data Environment**

I. On the Data Environment designer, open the Detail Fields in Student folder, and right-click Stdmast. Then click Properties.

2. On the STDMAST Properties dialog box, click on the Aggregates ta\:1.

3. Click on the Add button.

4. Click on the Name box and change Aggregate1 to StudentCount.

5. In the Function combo box select COUNT.

6. In the Aggregate combo box, select GrandTotal (it should be selected by default).

7. In the Field combo box, select SLid. Click OK to close the dialog box.

8. A new folder, GrandTotal fields in GrandTotal1 is added under STDMAST.

Now that you have created the aggregate field, you can place it on the Data

Reportdesigner.

**To Add the aggregate field to the data report**

1. Select the DataReport and change the DataMember property to,

0 GrandTotal1.

2. From the Data Environment designer, drag the Student Count aggregate field onto the appropriate group section. Change the Caption of the Label control to Student total.

3. Reposition and resize the TextBox control and the Label control.

4. Save and run the project.

**Forcing Page Breaks In the Data Report**

A page break helps to make a report more readable (and therefore understandable) by breaking the data into logical chunks of information. The

Data Report designer allows you to force page breaks before or after any group header or footer, or force a page break after the report header or before the report footer. (Forcing of page breaks on page headers and page footers is ignored because they already occur before and after a page break.)

**To Force a Page Break:**

1. Click on the group header to select it.

2. On the Properties window, click forcePageBreak, and then click on the arrow that appears .at the right of the box.

3. Click on the 1 - rptPageBreakBefore from the drop-down list. Run the project.

**Adding Date, Time, Page Number, and Title to the Data Report**

You can add Header and Footer data to each page of a data report by adding an appropriate Label control to the Page Header or Page Footer section. The Data

Report designer features the following header and footer sections:

* **Report Header**-occurs only once, at the very top of the report. Can be used for introductory material, such as the purpose of the report.
* **Report Footer**-Like the Report Header, occurs only once, at the very end' of the report. Can be used for summary material, such as conclusions, highlights of the report or grandtotals using the Function control.
* **Page Header**-occurs at the top of every page. Can be used for page number, date and time of the report, or title of the report.
* **Page Footer**-occurs at the bottom of every page. Can also be used for page number, date and time of the report, or other incidental information.

The Data Report designer contains several pre-configured controls that can be , used to quickly add the date, time, page number, or report title to any section 9ft the report. These controls are simply Label controls containing codes that display the following information:

|  |  |
| --- | --- |
| **Function** | **Code** |
| Current Page Number | %p |
| Total Number of Pages | %P |
| Current Data (Short Format) | %d |
| Current Data (Long Format) | %D |
| Current Time (Short Format) | %t |
| Current Time (Long Format) | %T |
| Report Title | %i |

To display a percent sign, use %%

**To add the title and page numbers to the page header**

1. Click on the Data Report designer's title bar to select it.

2. On the Properties window, click Title and type the title. This is the title that will display in the page header.

3. Right-click on the Data Report designer and click Show Page Header/Footer.

4. Right-click on the page header section and click Insert Control.

5. From the menu, click Report Title.

6. Reposition the new control in the middle of the section, extending it wide enough to display the entire report title.

7. Repeat step 4.

8. From the menu, click Current Page Number.

9. Position the l1awcontrol in the far right corner of the designer.

10. Set the new control's Alignment property to 1- Justify Right.

**Printing a Data Report**

Printing a data report can be accomplished in one of two ways. The user can click on the Print button that appears on the data report in Print Preview mode (using the Show method), or you can programmatically enable printing using the

Print Report method. If an error occurs during printing, trap it in the Error event.

Choosing to Display a Print Dialog Box

When printing a report programmatically, you have two choices: to print by

Displaying the Print dialog box, or by printing without displaying the dialog box.

**To display the Print dialog box**

1. Add a Command Button to a Form.

2. In the button's Click event, place the following code:

DataReport1.PrintReport True

The Print dialog box allows the user to select a printer, print to file, select a range of pages to print, and specify the number of copies to print.

Note Printers must be installed on the computer in order to present a choice of printers.

**Printing Without a Dialog Box**

In some cases, you may wish to print the report without user intervention. The

Print Report method also gives you the option of selecting a range of pages to print, either all, or a specified range.

**To print without displaying the dialog box**

1. Add a Command Button to a Form.

2. In the button's Click event, place the following code:

GataReport1.printReport False

Or, to specify a range of pages to print, use the code below:

DataReport1.PrintReport False, rptRangeFromTo,1,2

**Selecting Data for the Report Based on a User Input**

It may be necessary to generate ~report based on some input by the user at runtime. For example when you generate the student marks report, the report contains data of all students. You may need to generate a report of that student whose roll number is input by the user. In order to do this,

* Select the DataEnvironment designer from the project explorer and go to the DataEnvironment designer window.
* In the DataEnvironment designer, select the Connection object and double click to go the code window.
* Select the procedure. Will Execute of the connection object
* Private Sub Connection1\_WiIIExecut (Source As String, cursor Type As

ADODB.CursorType Enum, LockType As ADODB.LockTypeEnum, Options As

Long, adStatus As ADODB.EventStatusEnum, ByVal pCommand As

AOODB.Command, ByVaI pRerordset As ADODB.Recordset, ByVaI

pConnection As ADODB;Connection)

Dim rollNo

RoIINo=lnputBox("Enter roll number of student","marks")

Source="SELECT. FROM STDMARKS WHERE ST\_ID " & RollNo

End Sub