

# CONTENTS

<b>Chapters</b>	<b>Page No.</b>
1. Visual Basic Environment and Overview	1-98
2. Fundamentals of Visual Basic	99-154
3. Procedures and Projects	155-256
4. Printers and Functions	257-340

# **SYLLABUS**

## **VISUAL BASIC**

### **SECTION-A**

#### **Visual basic environment and overview**

Overview of main screen, menu bar, tool bar, tool box using menus, customizing a form, building user control. command buttons text boxes, labels images controls.

### **SECTION-B**

Statements in visual basic, writing codes, dialog box. variable, type of variable string numbers.

### **SECTION-C**

Writing procedures, VB programs structure, projects. forms, modules, and frames, project with multiple forms displaying information on form, picture boxes, textboxes.

### **SECTION-D**

Printer object controlling program flow. built in function user defined function and procedures. arrays, grids & records. object oriented programming, creating object, building classes.

# VISUAL BASIC ENVIRONMENT AND OVERVIEW

---

## LEARNING OBJECTIVES

After going through this chapter, you should appreciate the following:

- Overview of Main Screen
- Menu Bar
- Tool Bar
- Tool Box
- Using Menus
- Customizing a Form
- Building user Control Command Buttons
- Text boxe
- Labels
- Image Controls

## OVERVIEW OF MAIN SCREEN

NOTES

Visual Basic is a development tool with multi facet advantages. One of them is that being Visual it is easy to learn and use. The other is that being a direct descendant of the original BASIC, which many of us remember as the original language of PCs. For a broader definition you can Visual Basic as a developing language which allows you to write and run programs under Windows environment.

### Editions of Visual Basic

Various editions of Visual Basic are:

Working Model	It lacks important functionality and is available with some books on Visual Basic.
Learning Model	It was earlier known as Standard Edition. It is good if just trying to learn the software and not use it for advanced purposes.
Professional Model	It features all that which is not available in Learning model. In fact it is the complete model.
Enterprise Model	It includes the features of Professional Edition in the context of enterprise and remote development.

### Visual Basic is part of Visual Studio

Microsoft has combined the various development tools in a package called Visual Studio. This package contains: Visual C++, Visual J++, and Visual FoxPro besides Visual Basic. Most of the applications in Visual Studio have a common interface design.

### Starting Visual Basic

You can start Visual Basic as you would start any of the programs in the Windows environment, i.e., from Start\Programs\Visual Studio\Visual Basic or from Visual Basic icon which you may have created.

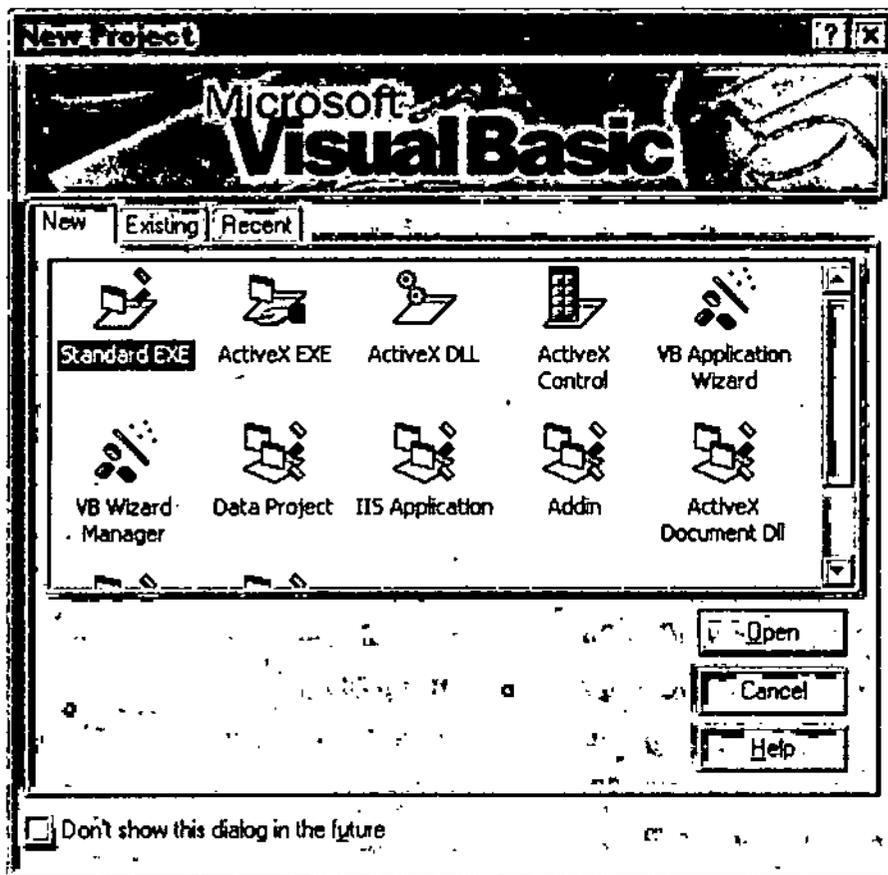
Once started it would give rise to an option box which asks you whether you want to open a new project or an existing project. You even have a panel for opening the recent ones, as shown below. If you have to start a new project, you must ensure that you have Standard.EXE highlighted.

Once clicked OK, you will get a blank desktop of Visual Basic as shown next.

The various options on the Visual Basic desktop are shown next.

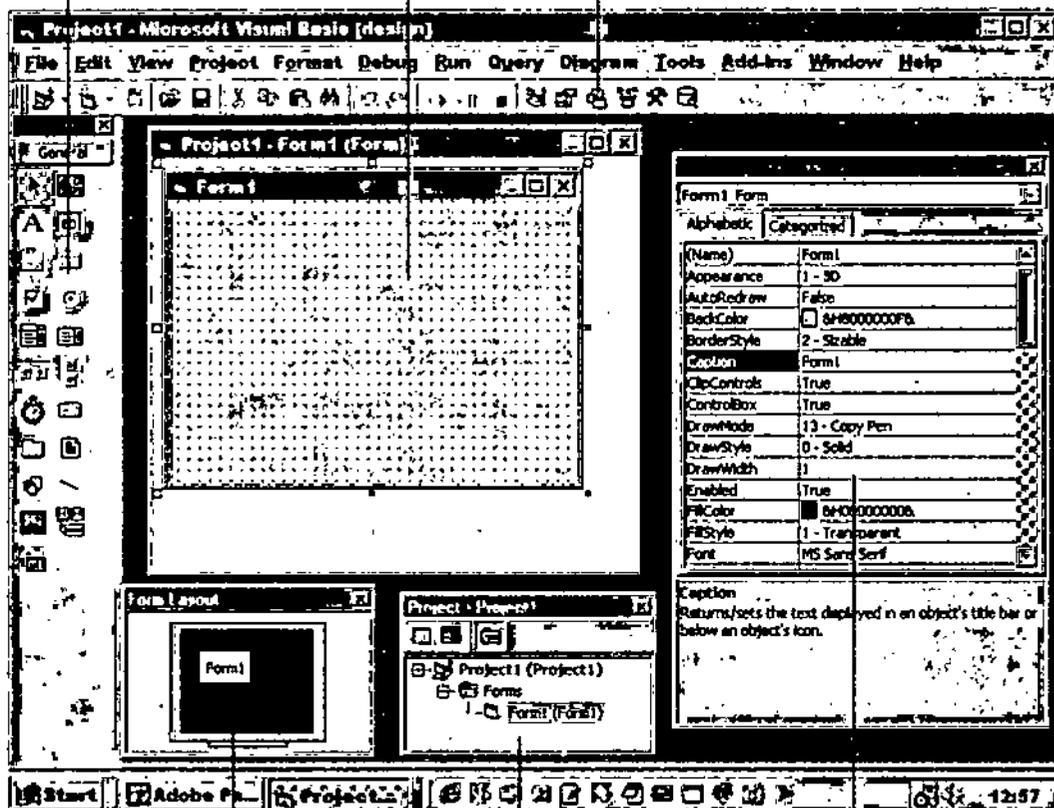
The toolbar options are shown on the next page.

Also on the next page, see the various objects on the Project Explorer. Detailed working of each one of these tools will be explained later when they are used.



NOTES

Toolbox Form Designer Toolbar

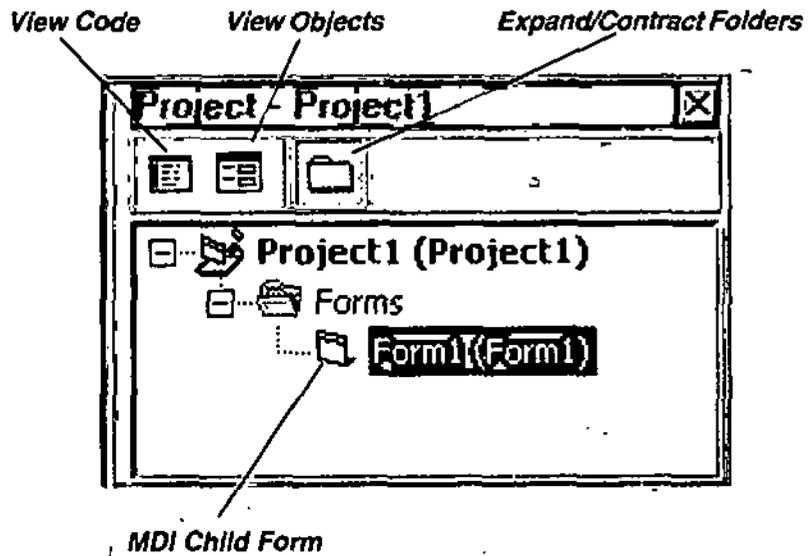
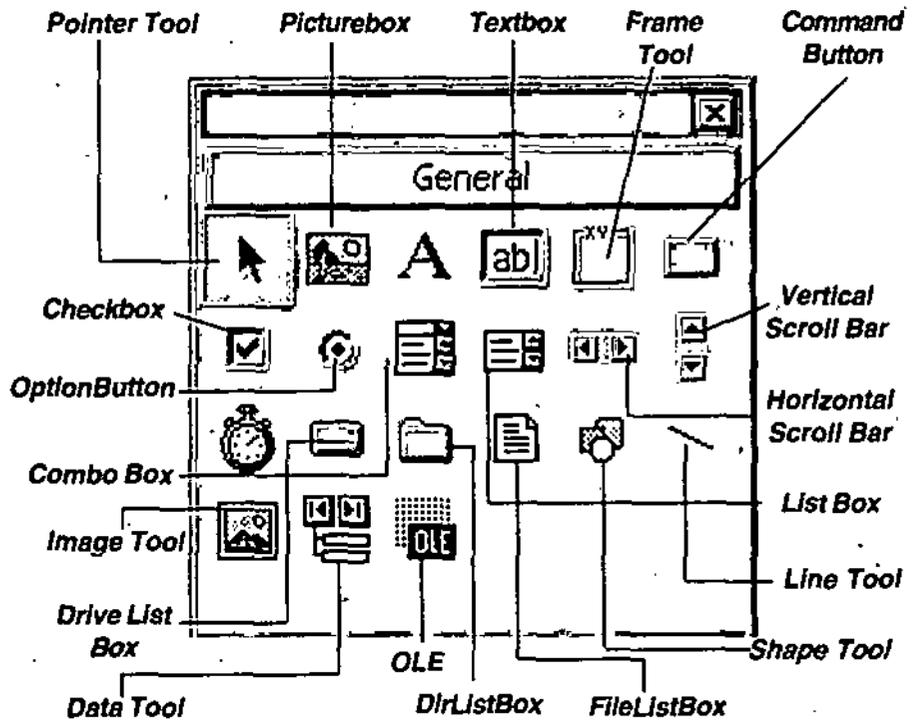


Form Layout

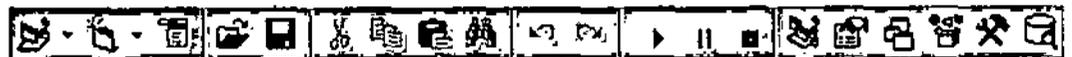
Project Explorer

Properties Window

NOTES



Next I tell you the various tools on the toolbar. For this also see the page next to next. Otherwise the toolbar looks like this:





**Add Project**



**Add Form**



**Menu Editor**



**Open Project**



**Save Project**



**Cut**



**Copy**



**Paste**



**Find**



**Undo**



**Redo**



**Start (Run)**



**Break**



**End (Stop)**



**Open Project Explorer**



**Open Properties Window**

NOTES

NOTES



Form Layout Window



Object Browser



View Toolbox



Data View Window

## MENU BAR

As in most of the software, Visual Basic too has a set of menu commands. The most common ones being File, Edit, View and Format ones. These are explained next, in short.

### File Menu

Various options available under the File menu are:

#### *New Project*

Displays the New Project dialog box where you choose the type of project you want to create. If there is currently another project open when you create a new project, you will be prompted to save your work.

#### *Open Project*

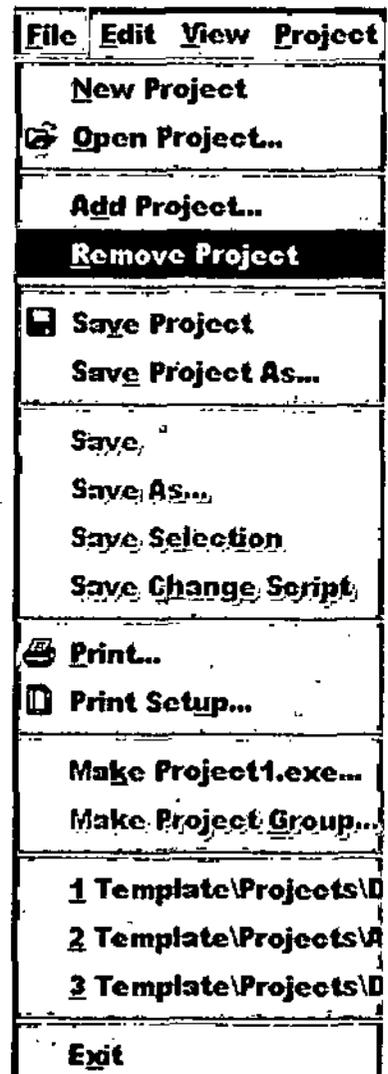
Closes the current project or group project, if one is loaded, and opens an existing project or group of projects. You can open as many projects as your system resources permit.

#### *Add Project*

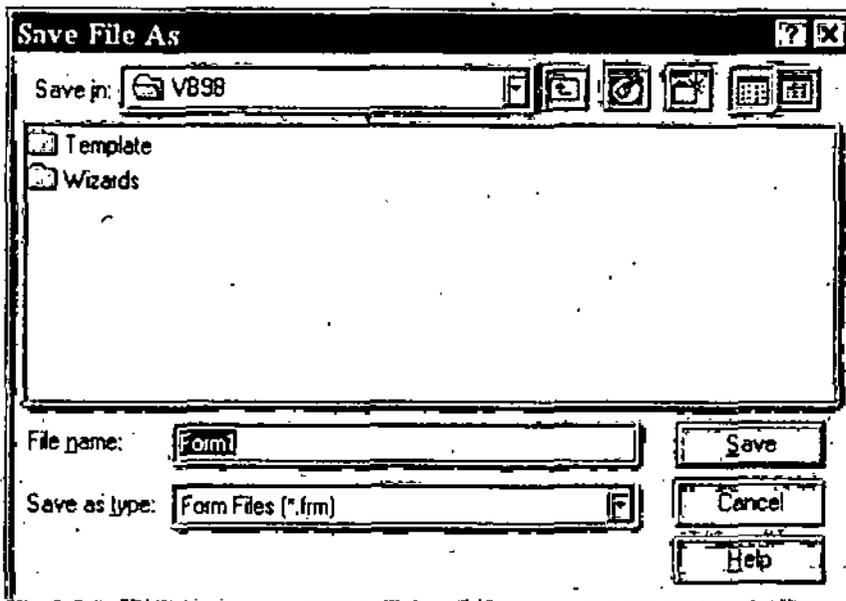
Displays the Add Project dialog box so that you can add a new or existing project to the currently open project group. If you have only one project open, Visual Basic adds the project and creates a project group. A project group exists only if there is more than one project.

#### *Remove Project*

Removes the selected project from the currently open project group. If there are pending changes to the project, you will be prompted to save them and then the project will be closed and removed from the project group. If you remove a project with a reference to any of the EXEs or other projects in the group, Visual Basic automatically switches the reference to the binary



NOTES



version of the component, if one is available. If a binary version is not available, the reference is marked as missing.

### ***Save Project/Save Project Group***

Saves the current project and all its components. The Save Project command changes to Save Project Group if you add a project to the project group.

### ***Save Project As/Save Project Group As***

The Save Project command displays the Save Project As dialog box if this is the first time the project is being saved. The Save Project As command displays the Save Project Group As dialog box if this is the first time the project group is being saved.

The Save File dialog box shown here has the following options:

#### ***SaveIn***

Select the location where you want to store the project file.

#### ***Up One Level***

Shows a list of folders or drives one level above the current folder.

#### ***Create New Folder***

Creates a new folder.

#### ***List***

Shows the folders or documents in a list format that includes the folder or document icon and its name.

#### ***Details***

Shows the folder or documents in a list that includes the folder or document icon and name, its size (documents only), type, and the date and time it was last modified.

#### ***List***

Shows the list of folders or documents.

## NOTES

**Filename**

Give your project a name. To save a project with a new name, or in a different location, type a new filename. To save a project with an existing filename, select the name in a list or type the current name. When you choose Save, Visual Basic asks if you want to overwrite the existing file.

**Save as type**

Select a file type from the list; the default is Project (\*.vbp). Files of the selected type will appear in the File Name list box.

**Save**

Saves the project group under the specified name.

**Cancel**

Closes the dialog box without saving the project.

**Print**

Prints forms and code to the printer specified in the Microsoft Windows Control Panel.

The Print dialog box, shown here has the following options.

**Printer**

Identifies the printer to which you are printing.

**Range**

Determines the range you print:

- Selection — Prints the currently selected code.
- Current Module — Prints the forms and/or code for the currently selected module.
- Current Project — Prints the forms and/or code for the entire project.

**Print What**

Determines what you print. You can select as many options as you like, depending on what you selected as the Range.

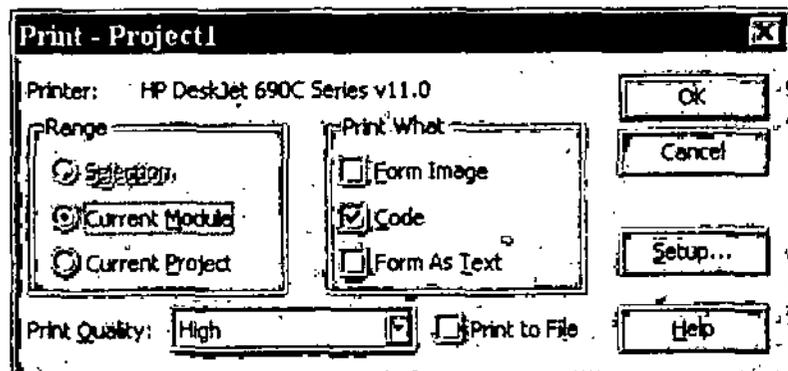
- Form Image — Prints the form images.
- Code — Prints the code for the selected range.

**Print Quality**

Determines whether you print high, medium, low, or draft output quality.

**Print to File**

If selected, print is sent to the file specified in the Print To File dialog box. This dialog box appears after you choose OK in the Print dialog box.



OK

Prints your selection.

Cancel

Closes the dialog box without printing.

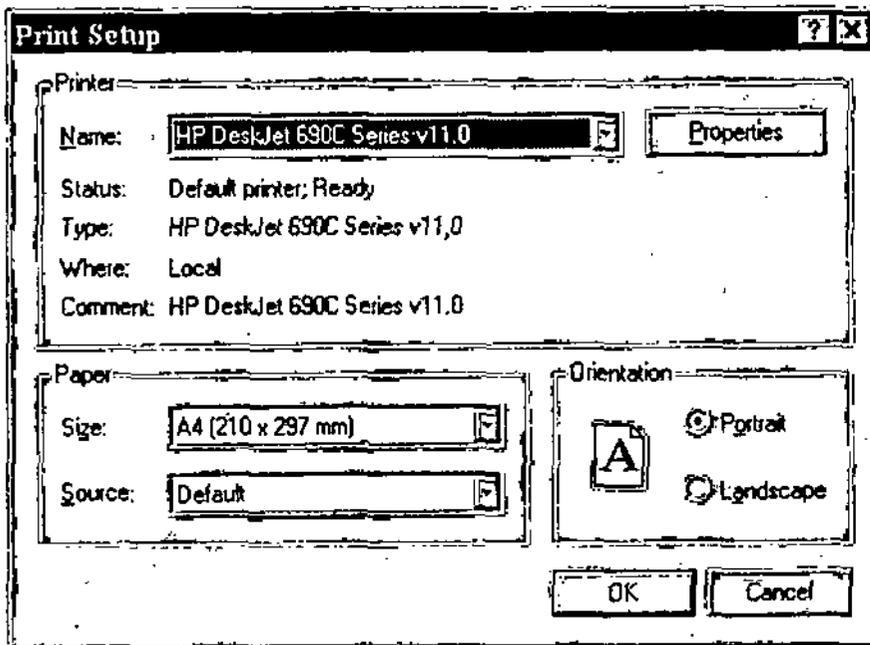
Setup

Displays the standard Print Setup dialog box.

Print Setup

Displays the standard Print Setup dialog box with options to specify the printer, page orientation, paper size, and paper source, as well as other printing options.

NOTES

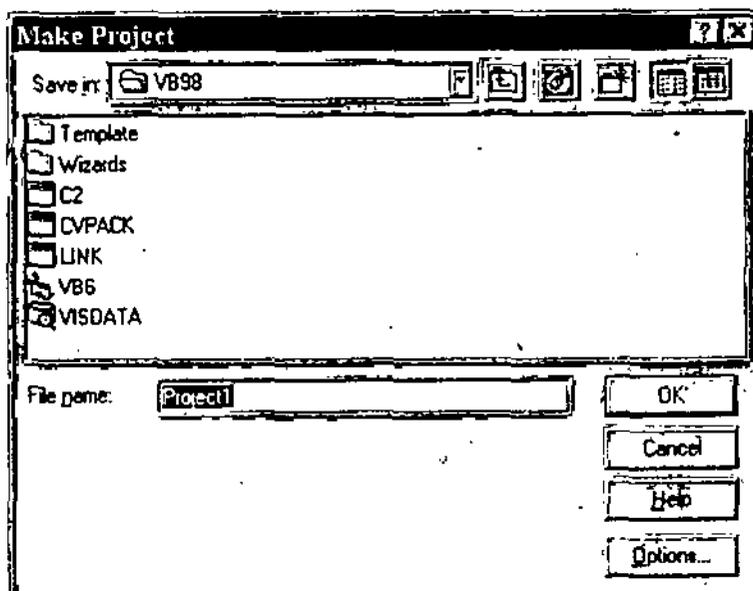


Make <Project>

Opens the Make Project dialog box so that you can build one or more projects contained in the project group into an executable file - EXE, DLL, or OCX.

Make Project Group

Creates a separate executable file for each project in the group you select. This is the same as using the /make flag in the command line and specifying a .vbg file.



**File 1, 2, 3, 4**

Lists the four most recently used projects (.vbp) or project groups (.vbg).

**Exit**

Closes the current project and quits Visual Basic. If you try to quit Visual Basic before saving changes to your work, you'll be prompted to save your work first.

## NOTES

**Edit Menu**

Various options here are similar to the ones which you would find under various other software.

**Undo**

Reverses the last editing action, such as typing text in the Code window or deleting controls. When you delete one or more controls, you can use the Undo command to restore the controls and all their properties. For forms, Undo is unavailable if the form has been edited since the last control deletion.

**Redo**

Restores the last text editing if no other actions have occurred since the last Undo.

**Cut**

Removes the selected control or text and places it on the Clipboard. You must select at least one character or control for this command to be available. You can undo the Cut command only in the Code window.

**Copy**

Copies the selected control or text onto the Clipboard. You must select at least one character or control for this command to be available. You cannot undo the Copy command in the Code window.

**Paste**

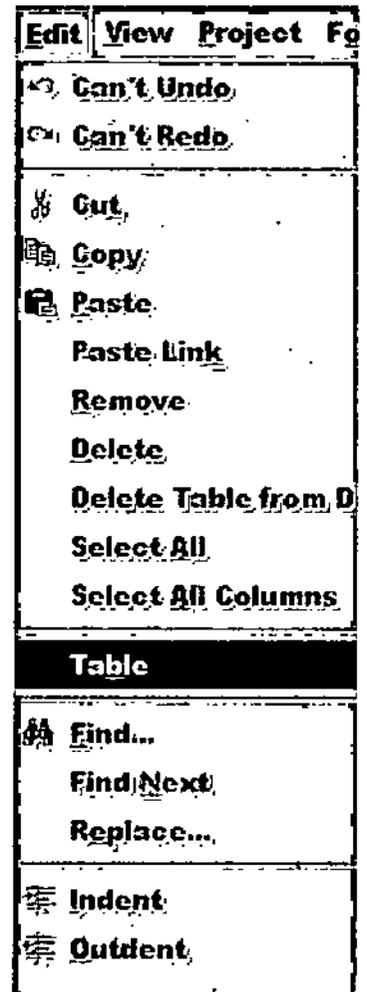
Inserts the contents of the Clipboard at the current location. Text is placed at the insertion point.

**Paste Link**

Supports pasting a link to a valid DDE source. Active only when the Clipboard contains a valid DDE source, and the selected control is a valid DDE link. You can also use Paste Link to link data from another application using the OLE container control. Once you copy valid data onto the Clipboard (a paragraph from a word processor or a range of cells from a spreadsheet), you can select the OLE container control on your form, and then choose Paste Link to link the data.

**Delete**

Deletes the currently selected control, text, or watch expression. You can undo the Delete command only in the Code window.



### **Delete Table from Database**

Deletes a table from a database. A table marked for deletion is permanently deleted from the database when you save your diagram. A reference to that table exists in memory until you save your diagram. If you close the diagram without saving it, the table will continue to exist in your database and appear in your diagram and every other diagram in which it appeared before you marked it for deletion. Used in the Database designer.

### **Remove Table from Diagram**

Removes a table from your diagram. Removing a table does not alter your server database; the table and its relationships to other tables continue to exist in the database and appear in Data View. Used in the Database designer.

### **Select All**

Selects all of the code in the active Code window.

### **Select All Columns**

Selects all the columns in an input source window in the Diagram pane so that you can perform an action on all of them at once. Used in the Query designer.

### **Table**

Displays a submenu that provides table commands. Used in the Table and Database designers.

### **Set Primary Key**

Defines a primary key to enforce uniqueness for values entered in specific columns that do not allow nulls.

### **Insert Column**

Adds a new column definition to the end of a table. Use additional columns to capture additional data that isn't already stored in an existing column.

### **Delete Column**

Deletes a column from a table. The columns, any constraints attached to them, any relationships they participate in, and the data that's stored in them are deleted from the database when you save the table or diagram.

### **Find**

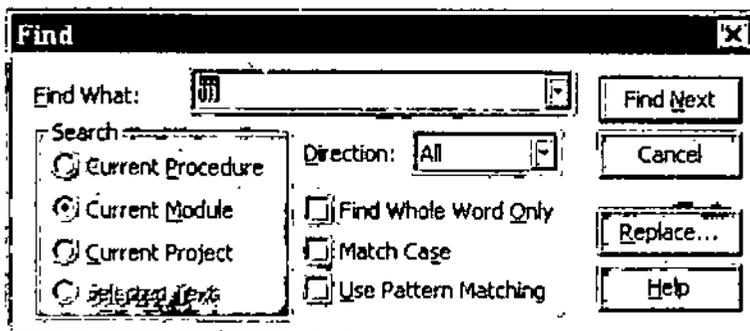
Searches for the specified text in a search range specified in the Find dialog box.

### **Find Next**

Finds and selects the next occurrence of the text specified in the Find What box of the Find dialog box.

### **Replace**

Searches code in the project for the specified text and replaces it with the new text specified in the Replace dialog box.



NOTES

***Indent***

Shifts all lines in the selection to the next tab stop. If you place the cursor anywhere in a line and choose the Indent command, the entire line is shifted to the next tab stop. All lines in the selection are moved the same number of spaces to retain the same relative indentation within the selected block.

***Outdent***

Shifts all lines in the selection to the previous tab stop. If you place the cursor anywhere in a line and choose the Outdent command, the entire line is shifted to the previous tab stop. All lines in the selection are moved the same number of spaces to retain the same relative indentation within the selected block.

***Insert File***

Opens the Insert File dialog box so that you can insert text from an existing file at the current pointer position in the Code window.

***List Properties/Methods***

Opens a dropdown list box in the Code window that contains the properties and methods available for the object that precedes the period (.). The List Properties/Methods command also displays a list of the globally available methods when the pointer is on a blank space. To have the list box automatically open as you type your code, select Auto List Members on the Editor tab in the Options dialog box.

***List Constants***

Opens a dropdown list box in the Code window that contains the valid constants for a property that you typed, and that preceded the equal sign (=). The List Constants command also works for functions with arguments that are constants. To have the list box automatically open as you type your code, select Auto List Members on the Editor tab in the Options dialog box.

***Quick Info***

Provides the syntax for a variable, function, statement, method, or procedure selected in the Code window.

Quick Info shows the syntax for the item and highlights the current parameter. For functions and procedures with parameters, the parameter appears bold as you type it, until you type the comma used to delineate it from the next parameter.

***Parameter Info***

Shows a popup in the Code window that contains information about the parameters of the initial function or statement. If you have a function or statement that contains functions as its parameters, choosing Parameter Info provides information about the first function. Quick Info provides information about each embedded function.

***Complete Word***

Fills in the rest of the word you are typing once you have entered enough characters for Visual Basic to identify the word you want.

***Go To Row***

Displays a submenu from which you can choose which Row in the Results pane to

go to. The options on the submenu are available only when a result set is available in the Results pane. Used in the Query and View designers. The following options are available on the submenu:

- First**  
Moves to the first row in the query results.
- Last**  
Moves to the last row in the query results.
- Next**  
Moves to the next row in the query results.
- Previous**  
Moves to the previous row in the query results.
- Row**  
Displays the Go To Row dialog box and allows you to move to a specified row in the query results.
- New**  
Moves to the end of the query results and appends a blank row to the grid to allow you to enter a new data row.

**Bookmarks**

Displays a menu that you can use to create or remove placeholders in the Code window, move to the next or preceding bookmark, or clear all of the bookmarks. Bookmarks mark lines of code so that you can easily return to them at a later time.

**View Menu**

The commands here allow you to view the various aspects of your Visual Basic projects.

Various commands here are:

**Code**

Displays or activates the Code window for a currently selected object.

**Object**

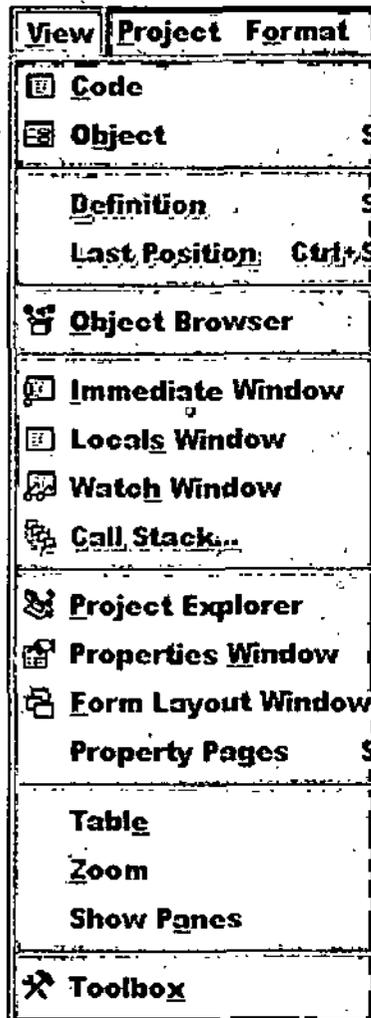
Displays the active item.

**Definition**

Displays the location in the Code window where the variable or procedure under the pointer is defined. If the definition is in a referenced library, it is displayed in the Object Browser.

**Last Position**

Allows you to quickly navigate to a previous location



NOTES

in your code. Enabled only if you edited code or made a Definition command call and only when the Code window is displayed. Visual Basic only keeps track of the last 8 lines that were accessed or edited.

### ***Object Browser***

Displays the Object Browser, which lists the object libraries, the type libraries, classes, methods, properties, events, and constants you can use in code, as well as the modules and procedures you defined for your project.

### ***Immediate Window***

Displays the Immediate window and displays information resulting from debugging statements in your code or from commands typed directly into the window.

### ***Locals Window***

Displays the Locals window and automatically displays all of the variables in the current stack and their values. The Locals window is automatically updated every time you change from run time to break mode and every time the stack context changes.

### ***Watch Window***

Displays the Watch window and displays the current watch expressions. The Watch window appears automatically if watch expressions are defined in the project. If the context of the expression isn't in scope when going to break mode, the current value isn't displayed.

### ***Call Stack***

Displays the Calls dialog box, which lists the procedures calls in the application that have started but are not completed. Available only in break mode. When Visual Basic is executing the code in a procedure, that procedure is added to a list of active procedure calls. If that procedure then calls another procedure, there are two procedures on the list of active procedure calls. Each time a procedure calls another Sub, Function, or Property procedure, it is added to the list. Each procedure is removed from the list as execution is returned to the calling procedure. Procedures called from the Immediate window are also added to the calls list.

### ***Project Explorer***

Displays the Project Explorer, which displays a hierarchical list of the currently open projects and their contents. The Project Explorer is a navigational and management tool only. You cannot build an application from the Project Explorer. You can close a project by choosing the Remove Project command from the shortcut menu.

### ***Properties Window***

Displays the Properties window, which lists the design-time properties for a selected form, control, class, user control, property page, user document, or menu.

### ***Form Layout Window***

Displays the Form Layout window where you can preview and position your form as it will appear in your application. Use the Resolution Guide command on the shortcut menu to preview it using different resolutions.

## Property Pages

Displays the property pages for a user control which to change a control's properties at design time.

## Table

Displays a submenu that enables you to collapse and expand the selected window in the Diagram pane. Used in the Database, Query, and View designers.

## Zoom

Displays a submenu that enables you to zoom the diagram by a fixed percentage: 10%, 25%, 50%, 75%, 100%, 150% or 200%. By default, database diagrams appear at 100%. Tables can be edited at 50% or higher. Relationships can be created at 25% or higher. Selection centers the selected table in the diagram window. Used in the Database designer.

## Show Panes

Displays a submenu that enables you to hide and show panes. Choose the name of the pane to hide or show. If only one pane is visible, you cannot hide it. Used in the Query and View designers.

## Toolbox

Displays the standard Visual Basic controls plus any ActiveX controls and insertable objects you have added to your project.

## Color Palette

Displays or activates the Color palette, which enables you to change a form or control's colors and set up a custom color scheme.

## Toolbars

Lists the toolbars that are built into Visual Basic and the Customize command.

## Project Menu

This menu contains commands related to projects. Various commands are:

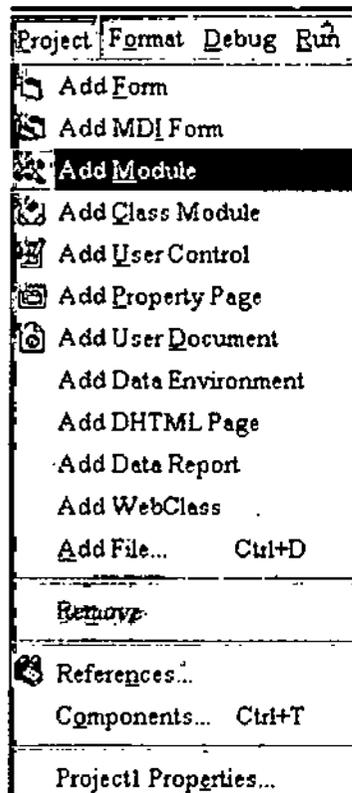
### Add Form

Displays the Add Form dialog box so you can insert a new or existing form into your active project.

### Add MDI Form

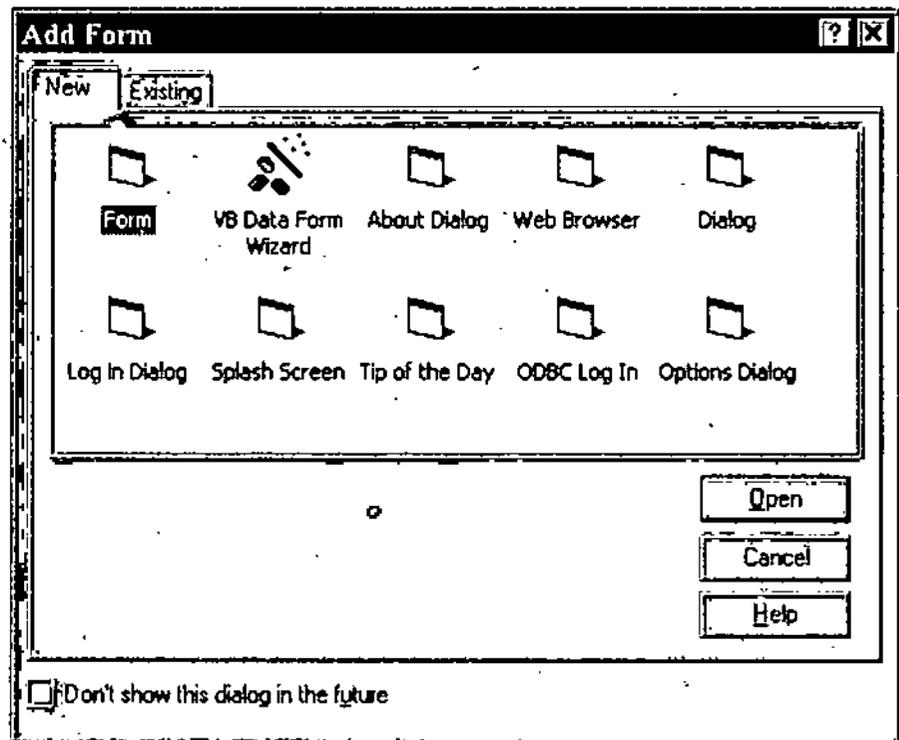
Displays the Add MDI Form dialog box so you can insert a new or existing MDI form into your active project. A project can have only one MDI form. This command is unavailable if a project already has an MDI form.

The MDI form command is also unavailable for ActiveX .exes and ActiveX .dlls if the threading is set to Thread per Object on the General tab on the Project Properties dialog box.



NOTES

NOTES



**Add Module**

Displays the Add Module or Add Class Module dialog box so you can insert a new or existing module or class module into your active project.

**Add Class Module**

Displays the Add Module or Add Class Module dialog box so you can insert a new or existing module or class module into your active project.

**Add User Control**

Displays the Add User Control dialog box so you can insert a new or existing user control into your active project.

**Add Property Page**

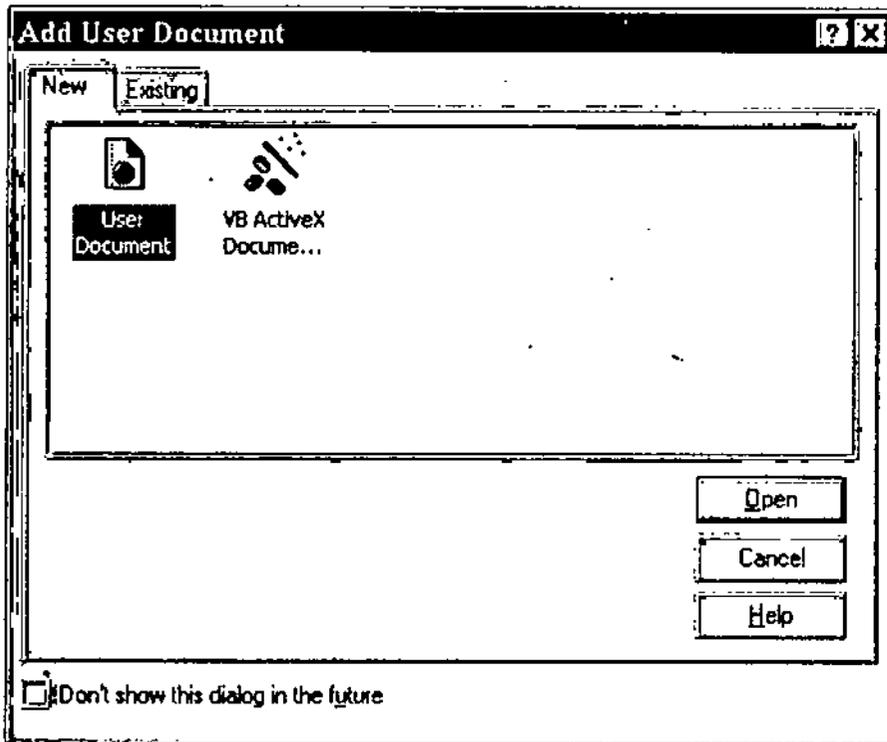
Displays the Add Property Page dialog box so you can insert new or existing property pages into your active project. Use the Property Page Wizard to build new property pages.

**Add User Document**

Displays the Add User Document dialog box so you can insert a new or existing User Document into your active project.

**Add AddIn Class/DHTML Page/Data Environment/Data Report/Web Class/Microsoft. UserConnection/ActiveX Designers**

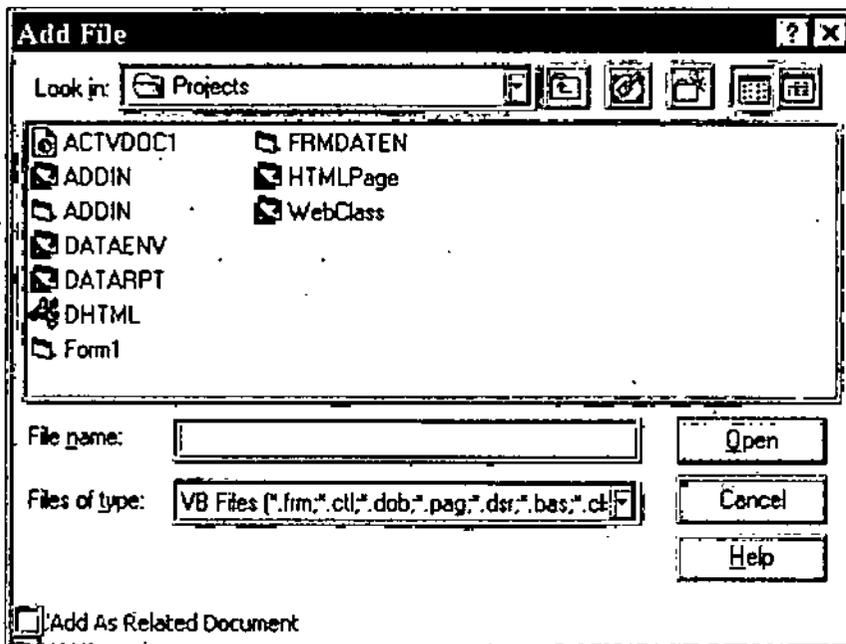
Displays the first four kinds of ActiveX designers that are loaded for a project. If more than four designers are loaded, the later ones will be displayed from the More ActiveX Designers submenu on the Project menu.



NOTES

### Add File

Adds an existing file to the current project. You can share files between projects. If you add a file to a project, you are simply including a reference to the existing file in the project; you are not adding a copy of the file to the project. Therefore, if you make changes to the file and save it, your changes will affect any project that includes the file.



### Remove <Item>

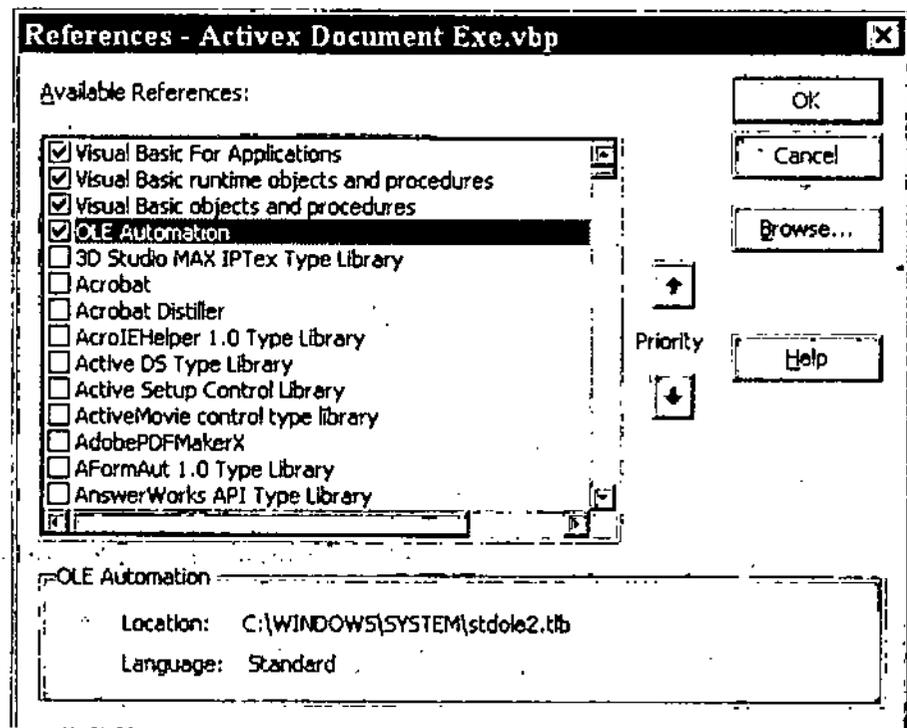
Removes the currently selected item from your project. If you made changes to it since last saving it, you are asked whether you want to save the changes first.

The file remains in all the other projects to which it has been added. Files that have been removed from a project remain stored on your hard disk until you delete them, following the standard procedures of your operating system.

### References

#### NOTES

Displays the References dialog box. This dialog box allows you to add an object library or type library, or project reference to your project. This makes another application's objects available in your code. Once a reference is set, the referenced objects are displayed in the Object Browser. You can also add references to other loaded and saved projects. If a project has not been saved, it appears as "UNSAVED: <ProjectName>" and you will be unable to make a reference to it.



### Components

Displays the Components dialog box from which you can add controls, designers, or insertable objects (such as a Microsoft Word Document) to the Toolbox. You can also reference loaded control projects.

### <Project Name> Properties

Displays the Project Properties dialog box where you view the project properties for the selected project.

### Format Menu

Various commands under this menu are:

#### Align

Aligns selected objects with each other using the last selected object, the one with the solid color grab handles, as the reference. The color of the grab handles is based on the color you set from Selected Items on the Appearance tab of the Display Properties dialog box in the Control Panel.

### *Lefts*

Aligns the horizontal position of the selected objects, putting the left-most edges in line with the last selected object, the one with the solid color grab handles.

### *Centers*

Aligns the horizontal position of the selected objects, putting the centers in line with the last selected object, the one with the solid color grab handles.

### *Rights*

Aligns the horizontal position of the selected objects, putting the right-most edges in line with the last selected object, the one with the solid color grab handles.

### *Tops*

Aligns the vertical position of the selected objects, putting the tops in line with the last selected object, the one with the solid color handles.

### *Middles*

Aligns the vertical position of selected objects, putting the middles in line with the last selected object, the one with the solid color grab handles.

### *Bottoms*

Aligns the vertical position of the selected objects, putting the bottoms in line with the last selected object, the one with the solid color grab handles.

### *To Grid*

Aligns the top left of the selected objects to the closest grid. The object is not resized.

### *Make Same Size*

Using the last object selected, the object with the solid color grab handles, makes the selected objects the same size in the dimension you select. The color of the grab handles is based on the color you set from Selected Items on the Appearance tab of the Display Properties dialog box in the Control Panel.

### *Size to Grid*

Adjusts the height and width of the selected object to fit the nearest gridlines in the form. You can change the size of your grid on the General tab of the Options dialog box.

### *Horizontal Spacing*

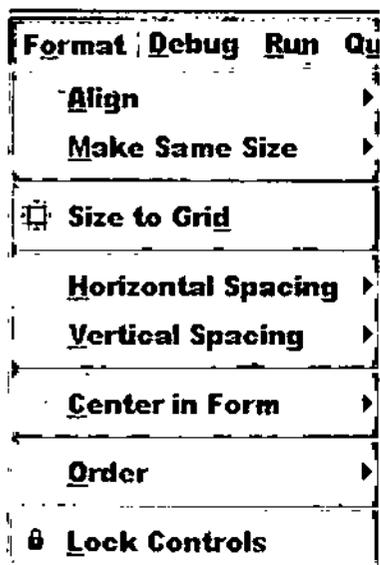
Changes the horizontal space between selected objects.

### *Make Equal*

Moves the selected objects so that there is equal space between them using the outermost objects as endpoints. The outermost objects do not move.

### *Increase*

Increases horizontal spacing by one grid unit based on the object with focus. When



NOTES

an object has focus, black handles appear on its borders. You can change the size of your grid units in the General tab of the Options dialog box.

*Decrease*

Decreases horizontal spacing by one grid unit based on the object with focus. When an object has focus, black handles appear on its borders. You can change the size of your grid units in the General tab of the Options dialog box.

*Remove*

Removes the horizontal space so that the objects are aligned with their edges touching based on the object with focus. When an object has focus, black handles appear on its borders.

**Vertical Spacing**

Changes the vertical space between the selected objects, based on the object with focus. When an object has focus, black handles appear on its borders.

*Make Equal*

Moves the selected objects so that there is equal space between them using the top and bottom objects as the endpoints.

*Increase*

Increases the vertical spacing by one grid based on the object with focus. You can change the size of your grid units in the General tab of the Options dialog box.

*Decrease*

Decreases the vertical spacing by one grid based on the object with focus. You can change the size of your grid units in the General tab of the Options dialog box.

*Remove*

Removes the vertical spacing so that the object's borders are touching, based on the object with focus

**Center in Form**

Centers selected objects on the central axes of the form.

*Horizontally*

Aligns the middles of the selected objects to a horizontal line in the middle of the form.

*Vertically*

Aligns the centers of the selected objects to a vertical line in the center of the form.

**Order**

Changes the order of the selected objects on a form.

*Bring To Front*

Moves the selected objects to the front of all other objects on a form.

*Send To Back*

Moves the selected objects behind all other objects on a form.

NOTES

### Bring Forward

Moves the selected object one step higher in the z-order.

### Send Backward

Moves the selected object one step lower in the z-order.

### Lock Controls

Locks all controls on the form in their current positions so you don't inadvertently move them once they are in the desired location. Because the Lock Controls command works on a form-by-form basis, it locks controls only on the selected form; controls on other forms are untouched. Visual Basic keeps track of which forms have controls locked in position and which don't. When the controls on the form are locked, the Lock Controls command toolbar button appears dimmed.

### Debug Menu

Various options for debugging the Visual Basic program are here. The options are:

#### Step Into

Step Into executes the statement at the current execution point. If the statement is a call to a procedure, the next statement displayed is the first statement in the procedure.

At design time, this menu item begins execution and enters break mode before the first line of code is executed. Not available at run time.

#### Step Over

Similar to Step Into. The difference in use occurs when the current statement contains a call to a procedure.

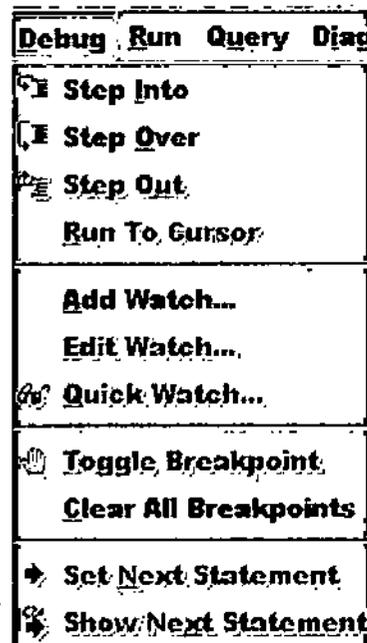
Step Over executes the procedure as a unit, and then steps to the next statement in the current procedure. Therefore, the next statement displayed is the next statement in the current procedure regardless of whether the current statement is a call to another procedure. Available in break mode only.

#### Step Out

Executes the remaining lines of a function in which the current execution point lies. The next statement displayed is the statement following the procedure call. All of the code is executed between the current and the final execution points. Available in break mode only.

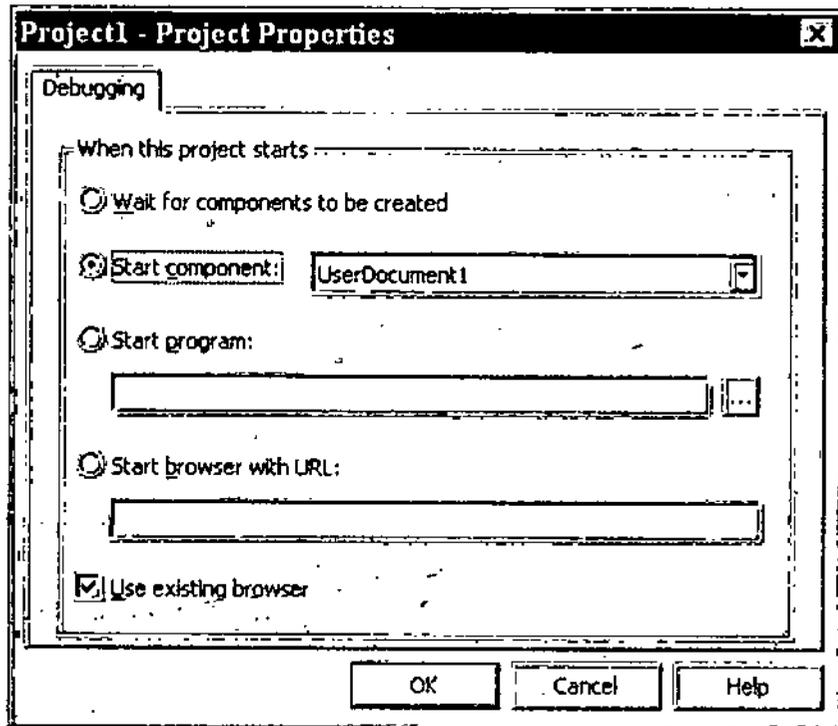
#### Run To Cursor

When your application is in break mode, use Run To Cursor to select a statement further down in your code where you want execution to stop. Your application will run from the current statement to the selected statement and the current line of execution margin indicator, appears in the Margin Indicator bar.



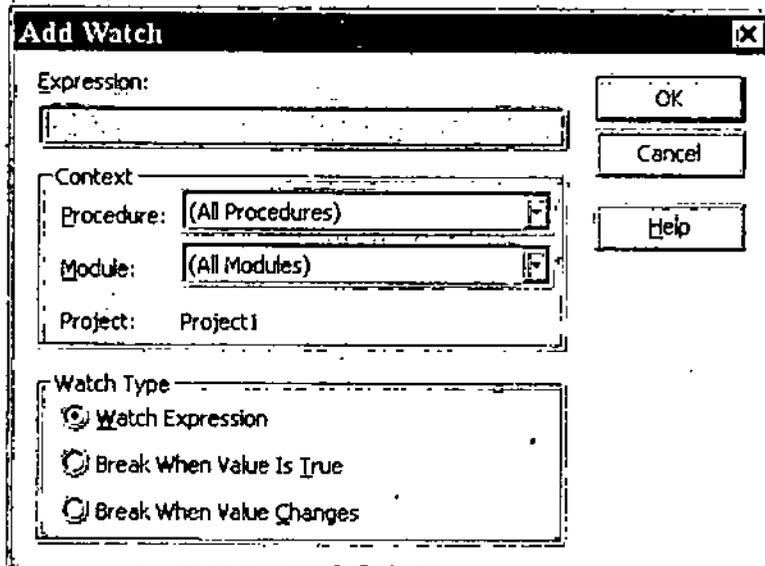
NOTES

NOTES



**Add Watch**

At design time or in break mode, this command displays the Add Watch dialog box in which you enter a watch expression. The expression can be any valid Basic expression. Watch expressions are updated in the Watch window each time you enter break mode.



**Edit Watch**

Displays the Edit Watch dialog box in which you can edit or delete a watch expression. Available when the watch is set even if the Watch window is hidden. Not available at run time.

**Quick Watch**

Displays the Quick Watch dialog box with the current value of the selected expression.

Available only in break mode. Use this command to check the current value of a variable, property, or other expression for which you have not defined a watch expression.

Select the expression from either the Code window or the Immediate window, and then choose the Quick Watch command. To add a watch expression based on the expression in the Quick Watch dialog box, choose the Add button.

### ***Toggle Breakpoint***

Sets or removes a breakpoint at the current line. You can't set a breakpoint on lines containing nonexecutable code such as comments, declaration statements, or blank lines. A line of code in which a breakpoint is set appears in the colors specified in the Editor Format tab of the Options dialog box.

### ***Clear All Breakpoints***

Removes all breakpoints in your project. Your application may still interrupt execution, however, if you have set a watch expression or selected the Break on All Errors option in the General tab of the Options dialog box. You cannot undo the Clear All Breakpoints command. Not available at run time.

### ***Set Next Statement***

Sets the execution point to the line of code you choose. You can set a different line of code to execute after the currently selected statement by selecting the line of code you want to execute and choosing the Set Next Statement command or by dragging the Current Execution Line margin indicator to the line of code you want to execute.

Using Set Next Statement, you can choose a line of code located before or after the currently selected statement. When you run the code, any intervening code isn't executed. Use this command when you want to rerun a statement within the current procedure or to skip over statements you don't want to execute. You can't use Set Next Statement for statements in different procedures.

### ***Show Next Statement***

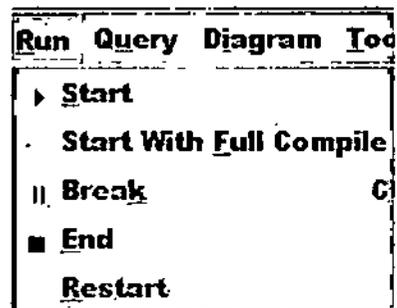
Highlights the next statement to be executed. Use the Show Next Statement command to place the cursor on the line that will execute next.

## **Run Menu**

For running your Visual Basic program you would be needing these commands. The commands are:

### ***Start***

Runs the application marked as the Start Up project in the Project Explorer. All forms being designed are closed, variables are initialized, and the startup form (if any) is loaded. You cannot run control projects at design time. DLL project types can be run by another EXE project creating an instance of them, and group projects cannot be run unless an EXE project exists in the group.



The first EXE project added to a project group is automatically set as the Start Up

NOTES

project unless you change it using the Set as Start Up command on the shortcut menu.

**Start with Full Compile**

Performs a full compile on the project.

NOTES

The application is fully compiled before starting, regardless of the current setting of the Compile on Demand and Background Compile options on the General tab of the Options dialog box. Choosing Start With Full Compile does not alter the Compile on Demand or Background compile settings.

**Break**

Stops execution of a program while it's running and switches to break mode. Any statement being executed when you choose this command is displayed in the Code window with in the left margin if you checked Margin Indicator Bar in the Editor Format tab of the Options dialog box. If the application is waiting for events in the idle loop (no statement is being executed), no statement is highlighted until an event occurs.

**End**

Stops running the program and returns to design time. This command is available at run time and in break mode.

**Restart**

Restarts your application after any kind of interruption. An interruption can be caused by run-time errors, a Stop statement, a breakpoint in your code, choosing the Break command, and a Break expression changing or becoming true.

**Query Menu**

Various options under this menu are:

**Run**

Executes the current query, stored procedure, or other SQL statement.

If you are creating a Select query, the results of the query appear in the Results pane of the Query Designer. If you are creating an Update, Insert, Insert Values query, Delete, or Make Table query, the Query Designer displays a message indicating how many rows were affected by the query.

**Clear Results**

Clears the results of the Results pane in the Query Designer. If a query is being executed when you clear the Results pane, the Query Designer stops the query.

Query	Diagram	Tools	Add-Ins	W
<b>Run</b>				
<b>Clear Results</b>				
<b>Verify SQL Syntax</b>				
<b>Group By</b>				
<b>Change Type</b>				
<b>Add To Output</b>				
<b>Sort Ascending</b>				
<b>Sort Decending</b>				
<b>Remove Filter</b>				
<b>Select All Rows From &lt;Table A&gt;</b>				
<b>Select All Rows From &lt;Table B&gt;</b>				

## **Verify SQL Syntax**

Checks the statement in the SQL pane against the database to determine whether the statement uses correct syntax. This command does not check that the command is valid (that is, that it will return correct results), only that the elements in the statement have been entered correctly for the database you are querying.

## **Group By**

Changes the current query to an aggregate query that summarizes and groups data; if the current query is already an aggregate query, reverts to a non-aggregate query. When you are creating an aggregate query, the Query Designer displays a Group By column in the Grid pane to allow you to specify the data columns to group.

## **Change Type**

Displays a submenu that enables you to change the current query to another type: Select, Insert, Insert Values, Update, Delete, or Make Table. The Query Designer preserves as much information as possible from the current query, such as search conditions.

## **Add To Output**

Adds the selected column or columns in the input source window of the Diagram pane to the list of columns to appear in the result set of a Select query.

## **Sort Ascending**

Adds the selected column or columns in the input source window of the Diagram pane to the list of columns to sort by and specifies Ascending in the Sort By column of the Grid pane. This option is available only when you have selected a column name in an input source window.

## **Sort Descending**

Adds the selected column or columns in the input source window of the Diagram pane to the list of columns to sort by and specifies Descending in the Sort By column of the Grid pane. This option is available only when you have selected a column name in an input source window.

## **Remove Filter**

Removes search conditions for the selected column or columns in the input source window of the Diagram pane. This command is available only when you have selected a column in the input source window that has search conditions defined. This option is available only when you have selected a column name in an input source window.

## **Select All Rows From <Table A>**

Specifies an outer join that includes rows from the first table even if they do not have related rows in the second table.

## **Select All Rows From <Table B>**

Specifies an outer join that includes rows from the second table even if they do not have related rows in the first table.

## **Diagram Menu**

Various commands under this menu are:

NOTES

### New Text Annotation

You can put notes into your diagram to describe the diagram and its objects.

#### Set Text Font

You can set the text font of the above diagram.

#### Add Related Tables

You can add a related table in the database using this option.

#### Show Relationship Labels

This command allows you to see the various relationship labels of the database.

#### Modify Custom View

It allows you to change the custom view of the database. Custom view shows only the column properties identified by the user. This dialog box appears when you right-click a table and then choose this command.

#### View Page Breaks

Equivalent to choosing the Print Preview command from the File menu. Previews the pages and page breaks of the active workbook on the screen before printing.

#### Recalculate Page Breaks

Printing your database diagrams gives you a useful picture of your database structure to refer to or distribute. You can preview the page breaks in the database diagram. You can also further customize the way your printed database diagram will look by changing the layout of the database diagram, and by specifying the table views in the diagram.

#### Arrange Selection

This command allows you to arrange the selections of the database on the screen.

#### Arrange Tables

It allows you to arrange the various tables in the database on the screen.

#### Autosize Selected Tables

On screen if you have more than one table, you can autosize them using this command.

### Tools Menu

Various commands under this menu are:

#### Add Procedure

Inserts a new Sub, Function, Property, or Event procedure into the active module.

#### Procedure Attributes

Opens the Procedure Attributes dialog box where you

Diagram	Tools	Add-Ins	Win
<b>New Text Annotation</b>			
<b>Set Text Font</b>			
<b>Add Related Tables</b>			
<b>Show Relationship Labels</b>			
<b>Modify Custom View</b>			
<b>View Page Breaks</b>			
<b>Recalculate Page Breaks</b>			
<b>Arrange Selection</b>			
<b>Arrange Tables</b>			
<b>Autosize Selected Tables</b>			

Tools	Add-Ins	Window
* <b>Add Procedure...</b>		
<b>Procedure Attributes</b>		
	<b>Menu Editor...</b>	<b>Ctrl+</b>
<b>Options...</b>		

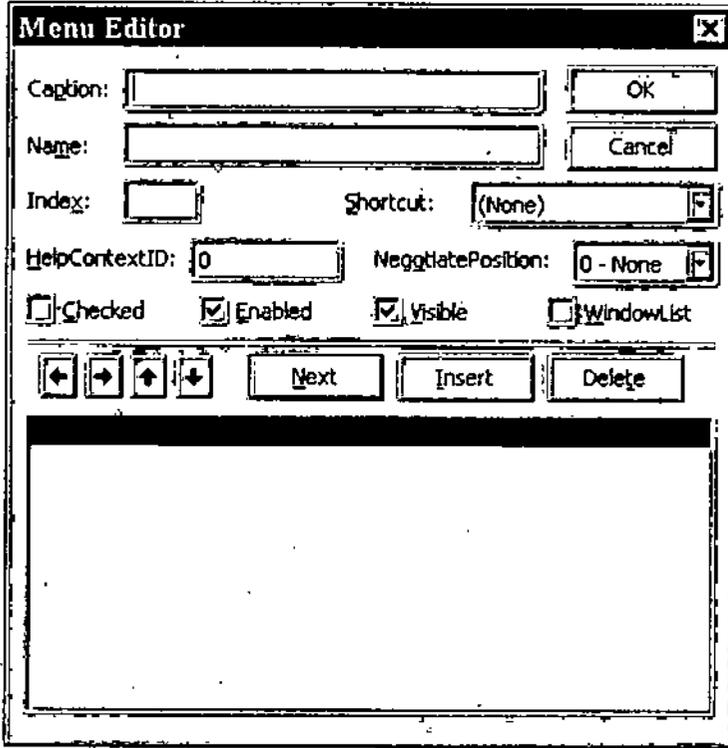
NOTES

can set the attributes for each property and method specified for an item. You can also use this command when you want to set the Value property for classes.

### Menu Editor

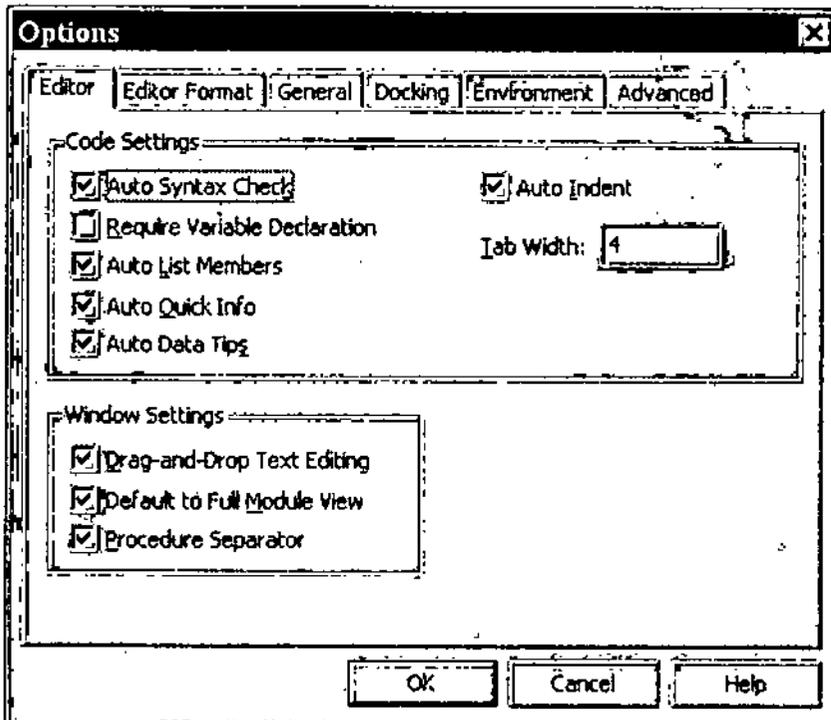
Displays the Menu Editor dialog box. Use the Menu Editor command to create custom menus for your application, and to define some of their properties. Available only at design time.

NOTES



### Options

Displays the Options dialog box, from which you can choose a tab to set attributes of the Visual Basic programming environment.



## Add-Ins Menu

Various commands under this menu are:

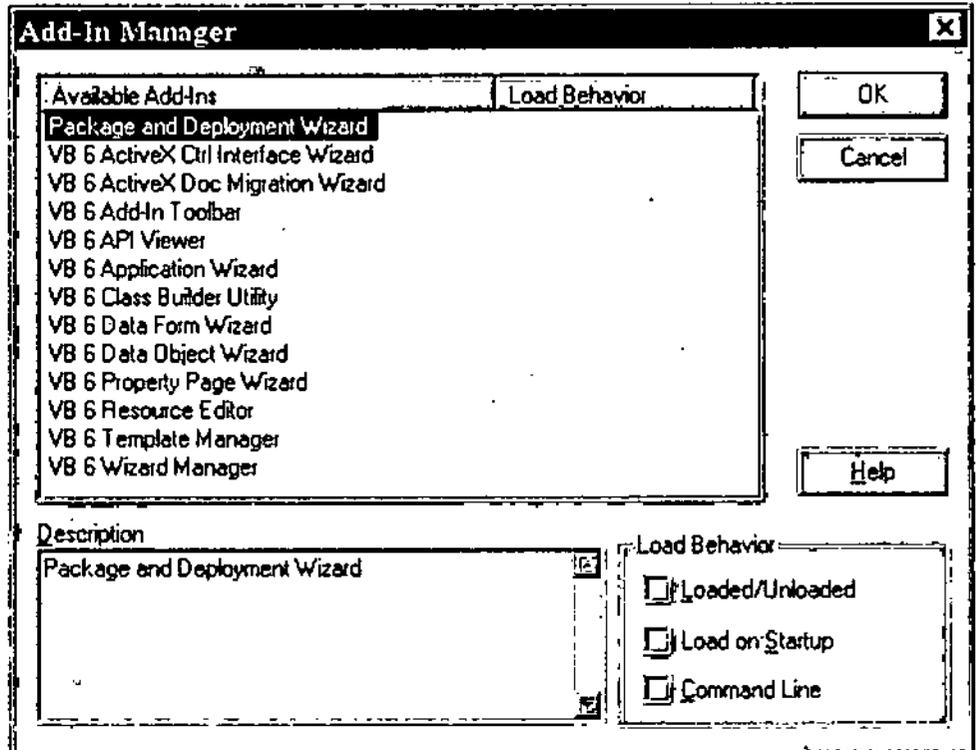
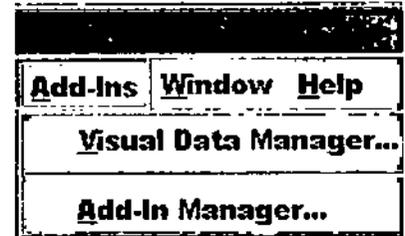
### Visual Data Manager...

It is an add to the Visual Basic which is used to manage the data in the database.

### Add-In Manager...

Allows you to register an add-in, load or unload it, and set its load behavior. To open the Add-In Manager, choose Add-In Manager from the Tools menu.

NOTES



## Window Menu

Various commands here are:

### Tile Horizontally

Tiles the Object and Code windows and the Object Browser horizontally when you are in MDI mode.

### Tile Vertically

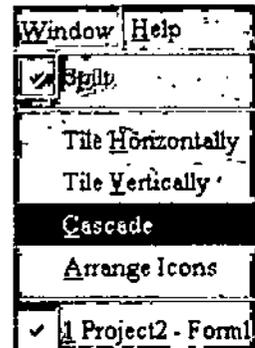
Tiles the Object and Code windows and the Object Browser vertically when you are in MDI mode.

### Cascade

Rearranges the Object and Code windows and the Object Browser in your project so they overlap in a cascade. Available only in MDI mode.

### Arrange Icons

Arranges the icons of the windows you have minimized, neatly at the bottom left of the window.



## Help Menu

Various help related commands are here:

### *Contents*

Using this command you can open the contents of the Help command.

### *Index*

You get the index of the Help command.

### *Search*

You can search for any command option of Visual Basic.

### *Technical Support*

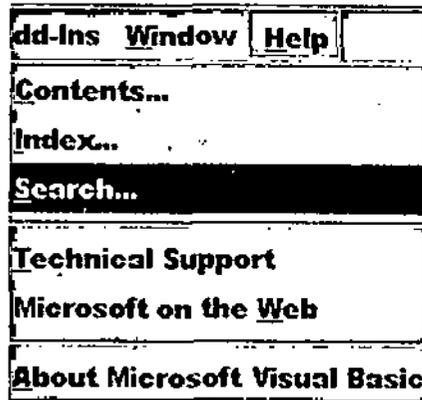
This section provides technical support, troubleshooting, and accessibility information for people with disabilities.

### *Microsoft on the Web*

You straightaway reach the web site of Microsoft and ask your query.

### *About Microsoft Visual Basic*

It displays the copyright page of Visual Basic.



NOTES

---

## TOOL BAR

Various items on the toolbar have been discussed earlier.

---

## TOOL BOX

Earlier, we had shown you the various tools in the toolbox. Now is the time to explain the working of each one of them.



Pointer

The only item in the Toolbox that doesn't draw a control. When you select the pointer, you can only resize or move a control that has already been drawn on a form.



PictureBox

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



Label

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



TextBox

Holds text that the user can either enter or change.



Frame

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





NOTES

	CommandButton	Creates a button the user can choose to carry out a command.
	CheckBox	Creates 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.
	OptionButton	Allows you to display multiple choices from which the user can choose only one.
	ComboBox	Allows you to draw a combination list box and text box. The user can either choose an item from the list or enter a value in the text box.
	ListBox	Used 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.
	Horizontal Scroll bar	Provides a graphical tool for quickly navigating through a long list of items or a large amount of information, for indicating the current position on a scale, or as an input device or indicator of speed or quantity.
	Vertical Scroll bar	Provides a graphical tool for quickly navigating through a long list of items or a large amount of information, for indicating the current position on a scale, or as an input device or indicator of speed or quantity.
	Timer	Generates timer events at set intervals. This control is invisible at run time.
	DriveListBox	Displays valid disk drives.
	DirListBox (directory list box)	Displays directories and paths.
	FileListBox	Displays a list of files.
	Shape	Allows you 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.
	Line	Used to draw a variety of line styles on your form at design time.
	Image	Displays a graphical image from a bitmap, icon, or metafile on your form. Images displayed in an Image control can only be decorative and use fewer resources than a PictureBox.
	Data	Provides access to data in databases through bound controls on your form.



---

## USING MENUS

---

You must have seen the various menus Visual Basic has. They were illustrated earlier. But, have you ever wondered how these menus are created. If you have to create similar menus in your application, you can do it by using the features given in this chapter.

Some menu items perform an action directly; for example, the Exit menu item on the File menu closes the application. Other menu items display a dialog box — a window that requires the user to supply information needed by the application to perform the action. These menu items should be followed by an ellipsis (...). For example, when you choose Save As... from the File menu, the Save File As dialog box appears.

A menu control is an object; like other objects it has properties that can be used to define its appearance and behavior. You can set the Caption property, the Enabled and Visible properties, the Checked property, and others at design time or at run time. Menu controls contain only one event, the Click event, which is invoked when the menu control is selected with the mouse or using the keyboard.

### *Pop-up Menus*

A pop-up menu is a floating menu that is displayed over a form, independent of the menu bar. The items displayed on the pop-up menu depend on the location of the pointer when the right mouse button is pressed; therefore, pop-up menus are also called context menus. You should use pop-up menus to provide an efficient method for accessing common, contextual commands.

Any menu that has at least one menu item can be displayed at run time as a pop-up menu. To display a pop-up menu, use the `PopupMenu` method.

### *Shortcut Menu*

It appears when the user right-clicks a selection, toolbar, or taskbar button, for example. The shortcut menu lists commands pertaining to that screen region or selection only.

## Designing Menu System

For this you have to open a menu editor, which is available under the Tools menu in Visual Basic.

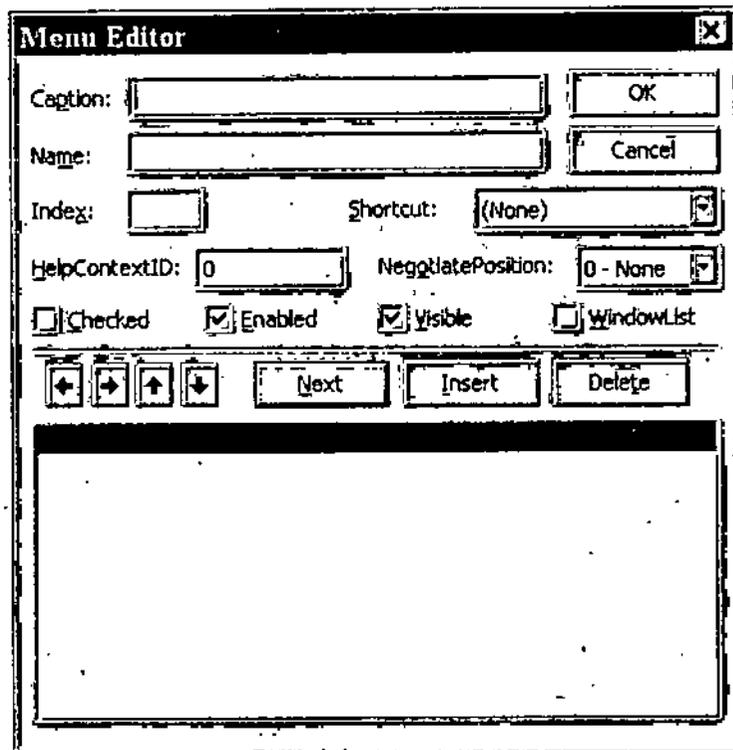
### *Menu Editor Dialog Box*

Various options in this Menu Editor are:

- Caption** This is the actual name of the menu. You can specify an access key, which allows the users to select the menu item with the keyboard instead of a mouse, by placing an ampersand (&) before the appropriate letter of the caption. For example, if a particular menu item's Caption property is set to `F&ormat`, the "o" is underlined, and the users can select that item by pressing `O` when the menu is active. The Caption property is required for all menu items.

NOTES

NOTES



**Name** This property is used to identify the menu item in code. The Name property is required for all menu items. Like all other controls, each Menu control must have name that will be used to refer to it in code. With these controls, unlike other controls, however, Visual Basic does not give menu items a default name; therefore, you must set the Name property of all menu items before leaving the Menu Editor.

**Index** If the menu item is part of control array, the Index property uniquely identifies the particular control array element.

**Shortcut** With this property, you can define shortcut key combinations that allow your users to select a menu item with one keystroke, bypassing the menu system entirely. For example, many programs use Ctrl+P as a shortcut key, eliminating the need to choose File, Print from the menu system.

**HelpContextID** This property sets a context ID that can be used in conjunction with a custom help file to provide context-sensitive help for your application.

**NegotiatePosition** If your application has linked or embedded objects, this property determines if and how this menu item is displayed while the linked or embedded object is active.

**Checked** If this property is True, a check mark appears to the left of the menu item's caption to indicate, for example, that a user has selected a particular option. If this property is False, no check mark appears.

**Enabled** This property can be set to False if its associated action is not appropriate at a particular time. For example, if no text is selected,

an Edit menu's Copy command can be disabled by setting its Enabled property to False.

**Visible** This property determines whether the menu item can be seen. Your application may have menu items that should be seen at certain times; for example, you may not want the users to be able to see the Window menu item if no windows are open.

**WindowList** If your menu is on an MDI (Multiple Document Interface) form, setting this property for a top-level menu control appends a dynamically-updated list of any active MDI child windows to the menu automatically.

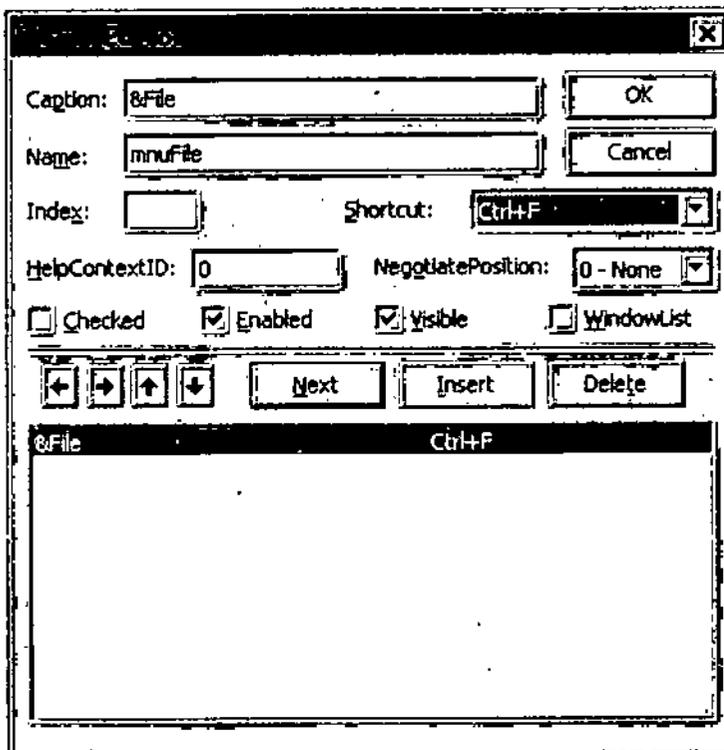
## NOTES

## Creating Menus

Visual Basic allows you to create various type of menus for your applications. Now you know that Menu Editor will help us in doing so but, let us how we can do that and create various type of menus.

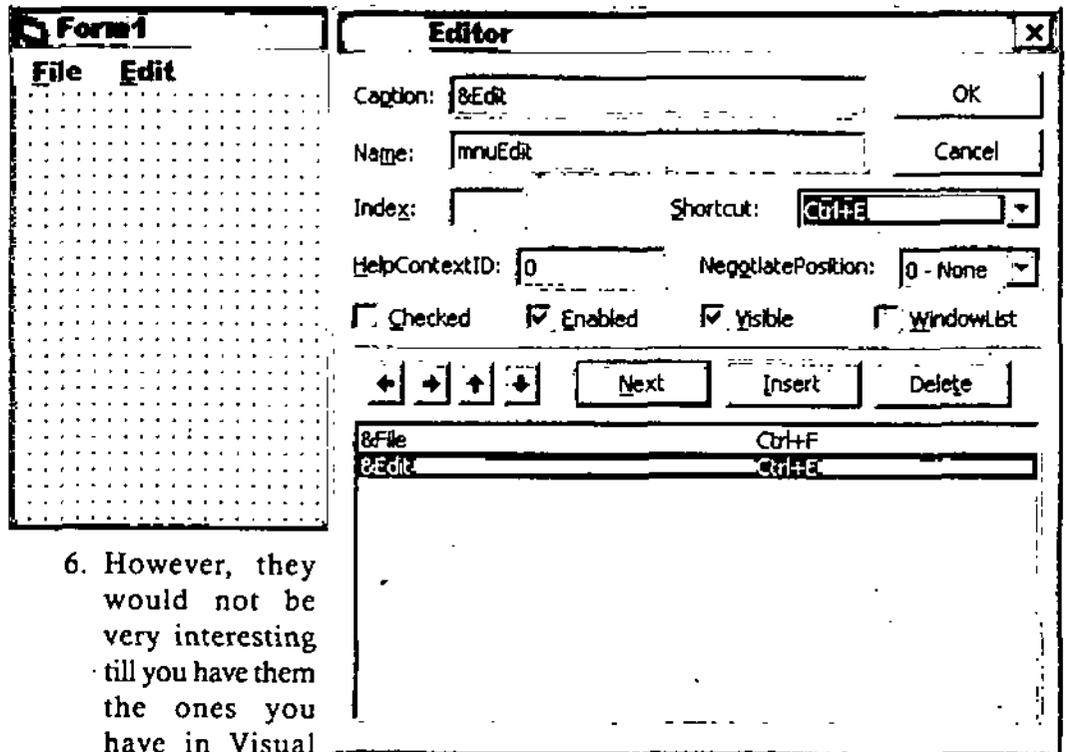
Let us create two menus on a form with the name of File and Edit. For this do the following:

1. With the Menu Editor open, type a caption for the first menu in the Caption box, for example, &File.



2. Type a name for the menu in the Name box, for example, mnuFile.
3. Click Next to move to the next menu.
4. Repeat the process to add a Edit menu.
5. Click Ok to see the menus on the form.

NOTES



6. However, they would not be very interesting till you have them the ones you have in Visual Basic.

But there are some conventions about naming which you have to follow before starting to create menu..

**Menu Naming Conventions**

The name of a menu or menu item is not the menu's caption which appears on the form—for example, File or Edit. The menu's name is, instead, a programmatic label for the menu object that users never see. Its conventional to use the prefix mnu for menus and menu items. The suffix for a menu should be the menu's caption—for example, mnuFile and mnuEdit.

Menu items should start with the menu that they are part of and append their caption. The New menu item in the File menu should be named mnuFileNew for example and the Open menu item in the File menu should be named mnuFileOpen. Because the Objects list in the Code Editor is alphabetical, this practice serves the purpose of grouping all the menu items on a menu.

**Setting the Properties**

The design time properties, which are given below, are exposed in the Menu Editor rather than in the Properties window.

Appearance	Caption	Checked
	Enabled	HelpContextID
Index	Name	NegotiatePosition
	Parent	ShortCut
Tag	Visible	WindowList

The following explains some of the more important properties to remember.

1. The Caption property determines the text you see in the menu. Using an ampersand (&) character gives an access key alternative to the mouse.
2. The Checked property places (or removes) a checkmark next to the menu item. This is handy for toggle items, and you can turn the checkmark off and on at run time by setting the Checked property appropriately.
3. The Enabled property is sometimes set to False when the menu item is not relevant. For example, you may want to disable a Save item until the user has entered some data. Once again, you can reset the Enabled property at run time.
4. A variation of the previous item is to use the Visible property and hide the item when it is not required—though it is less confusing for the user if you disable rather than hide items.
5. The Name property is, as always, the first one to define—by convention, menu controls begin with the mnu items.
6. The Shortcut property determines the keyboard alternative to the menu item. This is not quite the same as an access key (which is determined with the ampersand character). Usually, a shortcut is accessed through the Ctrl and Alt keys used in conjunction with another key, such as F1..

NOTES

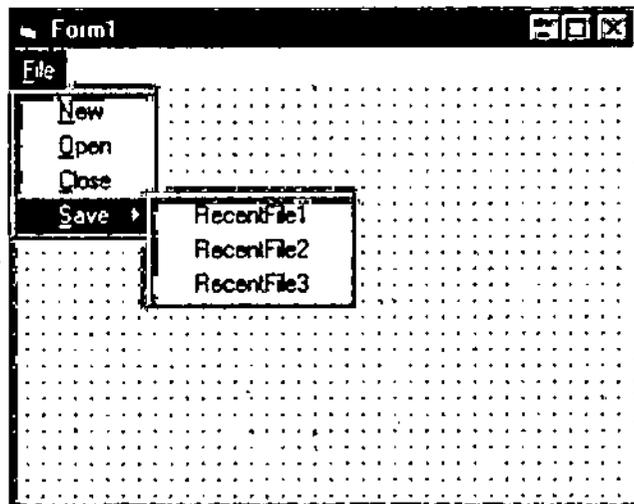
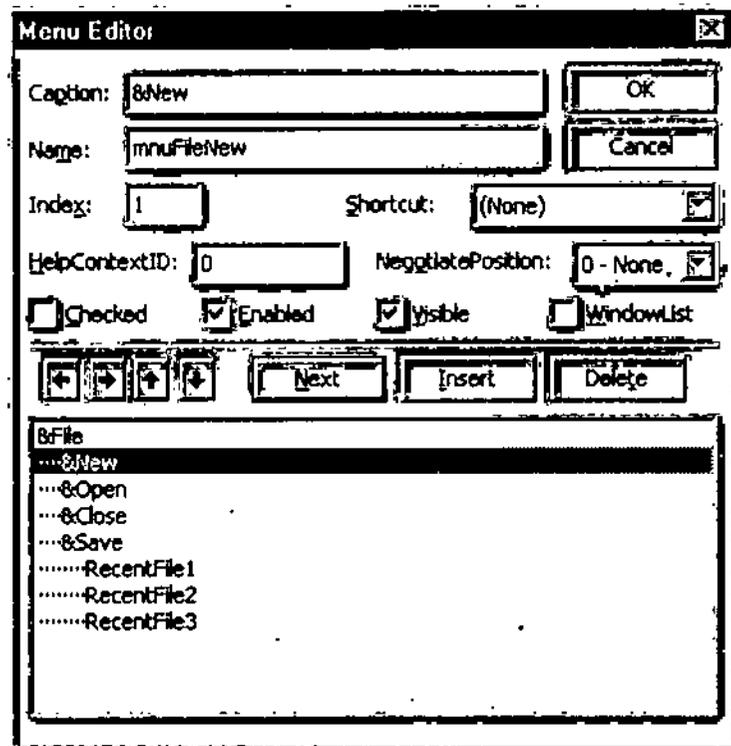
### ***Creating a Menu Control Array***

A menu control array is a set of menu items on the same menu that share the same name and even procedures. Each menu control array element is identified by a unique index value, indicated in the Index property box on the Menu Editor. When a member of a control array recognizes an event, Visual Basic passes its Index property value to the event procedure as an additional argument. Your event procedure must include code to check the value of the index property, so you can determine which control you are using.

To create a menu control array in the Menu Editor:

1. Select the form.
2. From the Tools menu, choose Menu Editor.
3. In the Caption text box, type the text for the first menu title that you want to appear on the menu bar. The menu title text is displayed in the menu control list box.
4. In the Name text box, type the name that you will use to refer to the menu control in code. Leave the Index box empty.
5. At the next indentation level, create the menu item that will become the first element in the array by setting its Caption (RecentFile1) and Name (mnuRecentFile).
6. Set the menu separator Index for the first element in the array to 0.
7. Create a second menu item at the same level of indentation as first.
8. Set the name of the second element to the same as the first element and set its Index to 1.
9. Repeat steps 5 to 8 for subsequent elements of the control menu array.

NOTES

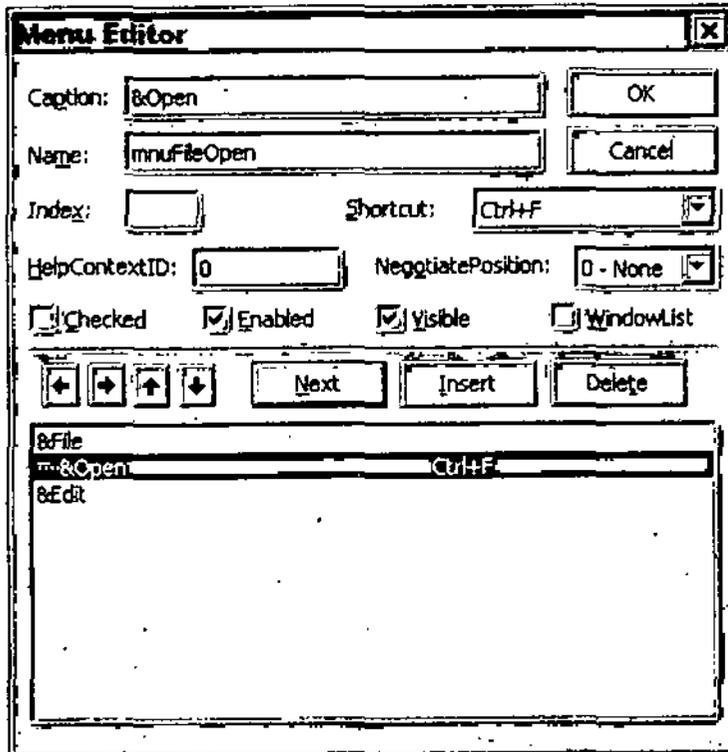


**Creating Sub Menus**

Sub menus as the name sounds are children of main menus. So have a child, you must have a father first. Here the father is the main menu and sub menu is the child. For this we will create menu items first. You can assign shortcut keys to the items which you cannot do to the main menus.

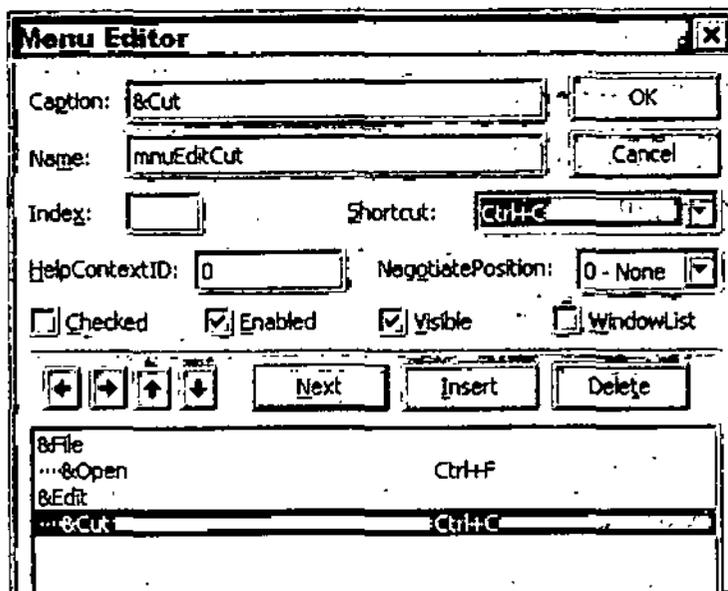
Follow the following steps to insert menu items in the menus.

1. In the Menu Editor's menu list, select the menu to which you want to add menu items.
2. Click the Insert button to add a space below the menu.
3. Click the right-arrow button to indent the menu one level in the hierarchy.
4. An ellipsis appears, representing the indentation.



NOTES

5. Type a caption for the menu item in the Caption box.
6. Type a name for the menu item in the Name box.
7. Be sure to include the menu name as a prefix of the menu item's name.
8. Use the drop-down menu to assign a shortcut key to the menu item.
9. Click Ok.
10. You now see the menu item with its shortcut key indicated as part of the window's menu system.
11. Repeat the process to add more menu items.



12. Click Ok.
13. The menu items appear as part of the window's menu system.

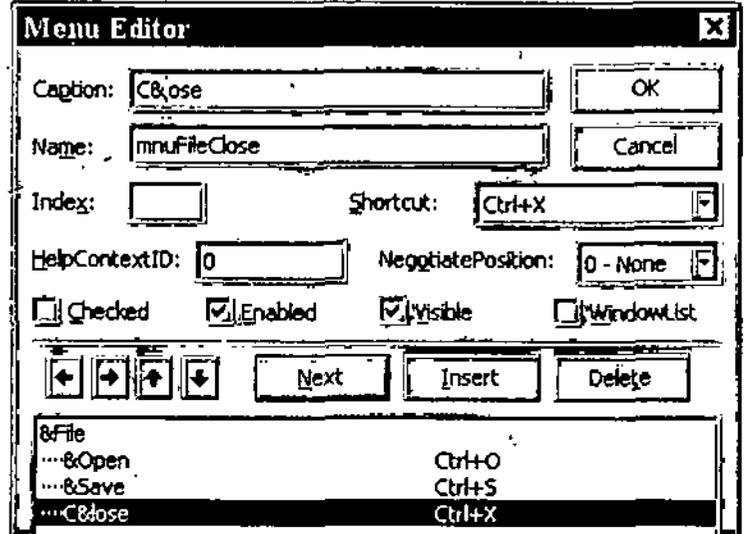


NOTES

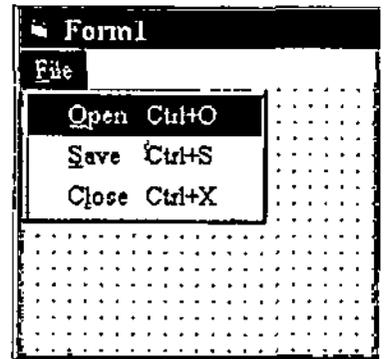
**Assigning Access Keys and Shortcut Keys**

Access keys are used to navigate a menu hierarchy in conjunction with the Alt keys. If the access key for the File menu is F and the access key for the New item in the File menu is N, pressing Alt + F followed by Alt + N has the same effect as clicking the New menu item — that is, it invokes the code in the New menu item's Click event. It is a principal of interface usability that every menu item can be reached via a combination of access keys.

To create an access key, use the Editor to place an ampersand (&) in the caption of the menu or menu item before the letter that will be used for access. It is important that access keys be unique at each menu level. If you don't follow this technique, the results will be unpredictable. The upper-level menu access keys must not be duplicated, for example, and all access keys for menu items in the File menu must be unique.



The Save menu item in the File menu and the Select All menu item in the Edit menu, however, could both use S as their access key. Although access keys are often the first letter of the caption, to avoid duplication, you can use a subsequent letter instead—for example, E&xit, making Alt + X the access key for the Exit menu item.



**Shortcut keys**

These keys differ from access keys in that they provide immediate invocation of the functionality of menu item. To reach a menu item using access keys, you must navigate the menu hierarchy.

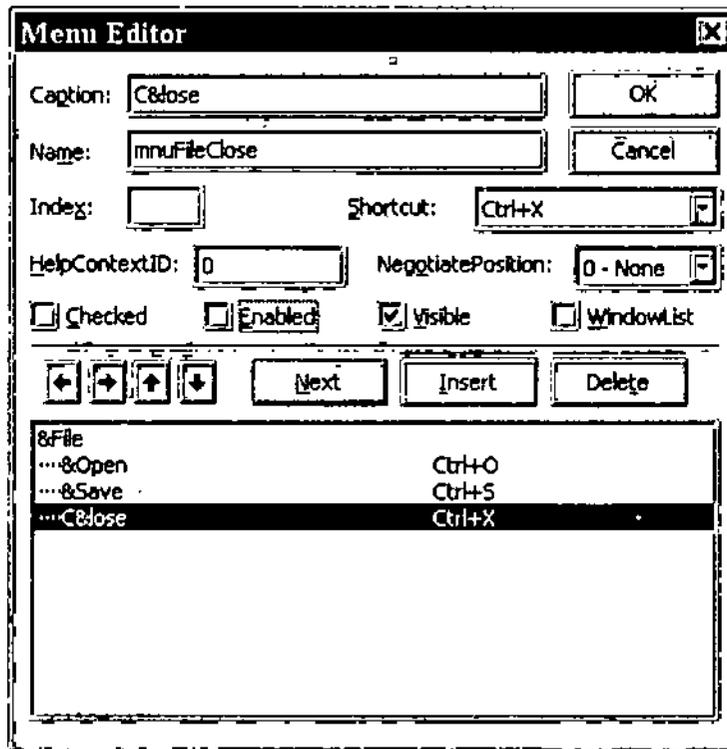
Shortcut keys are assigned to menu items (not menus) via a drop-down menu in the Menu Editor. Shortcut keys—actually keyboard combination, such as Ctrl + O—must be unique across an entire application.

Follow the following steps to add access keys and shortcut keys.

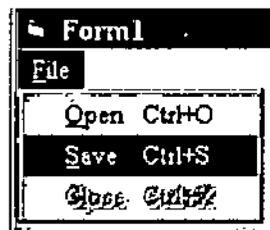
1. Open the Menu Editor.
2. Create a Caption with File.

3. Click at Insert to go the the next line.
4. Press right arrow to move to create ellipse.
5. Create a new entry for Open with putting & before the character O, so that it can be opened with Alt + F + O.
6. Add keyboard shortcut to it in the form of Ctrl + O
7. As above create 2 more entries with Save and Close. Let them open with S and C.

NOTES



8. Similar to above add keyboard shortcuts as Ctrl + S and Ctrl + X.
9. Click Ok.
10. You will see the menu as shown here.



**Runtime Enabling and Disabling of Menu Commands**

You must have seen that sometimes some menu items are dimmed are not available for use. This is what is enabling and disabling. You can disable a menu item by the following method.

1. With the Menu Editor open, select the menu item you want to disable. In this case the Close menu item.
2. Uncheck the Enabled checkbox.
3. Click Ok.
4. Click at File menu of your form to see that the Close menu item is dimmed.

**Displaying a Check Mark on a Menu Control**

Using this command you can check a menu item. It will be marked as Tick as if it has been checked. You can do this by the following method:

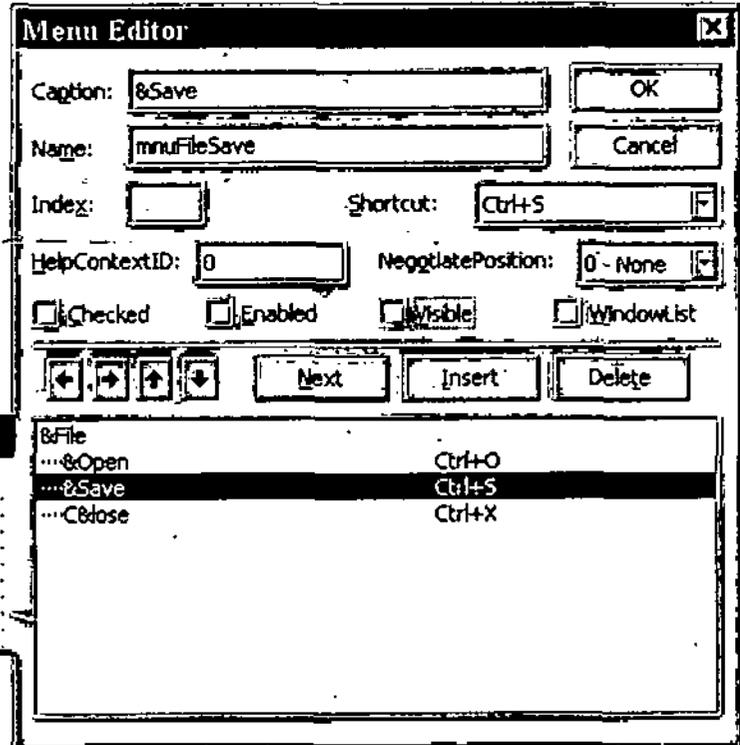
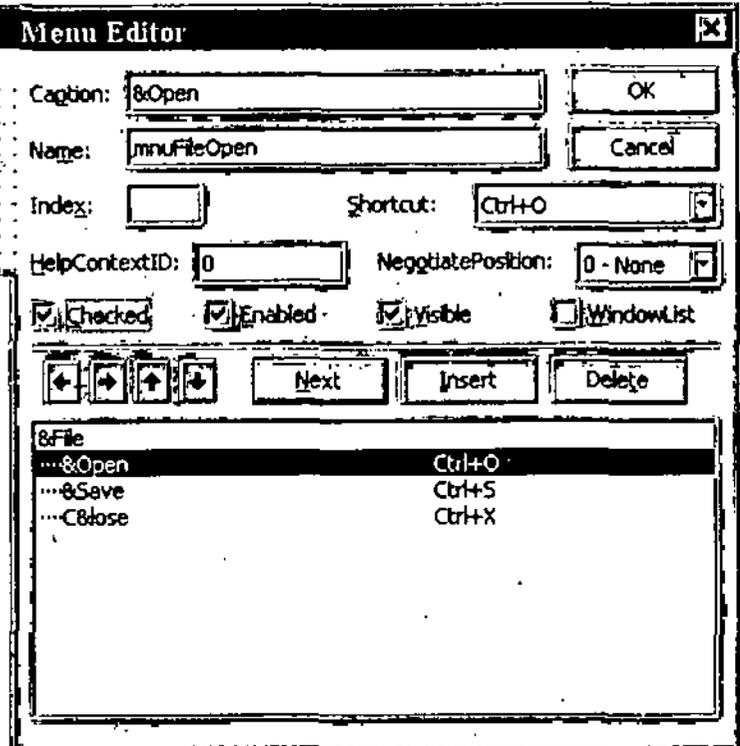
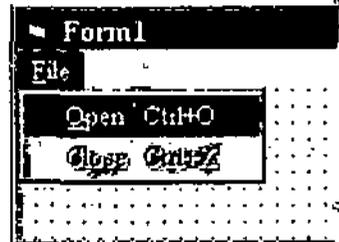
NOTES

1. With the Menu Editor open, select the menu item you want to checked. In this case the Open menu item.
2. Click at Checked checkbox.
3. Click Ok.
4. Click at File menu of your form to see that the Open menu item is clicked.

**Making a Menu Control Invisible**

You can make a menu control item invisible if you want to. You can do this by the following method:

1. With the Menu Editor open, select the menu item you want to hide. In this case the Save menu item.
2. Uncheck the Visible checkbox.
3. Click Ok.
4. Click at File menu of your form to see that the Save menu item is no longer visible.

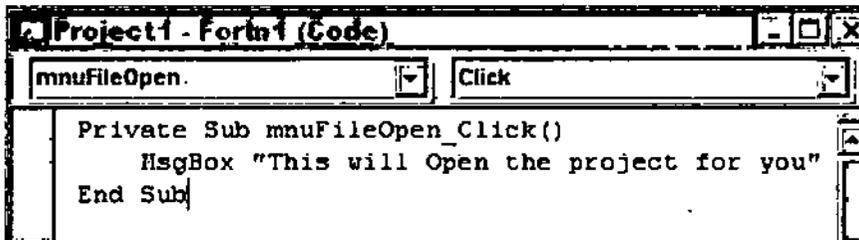


## Adding Menu Control at Runtime

What is the use of having items on the menu which do not perform? So, let us add some activity to the items. In the above case of the File menu, we will add a statement "This will Open the project for you", to the Open menu item.

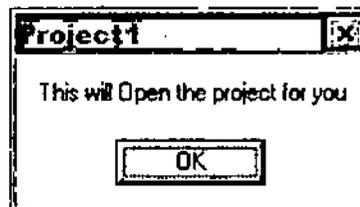
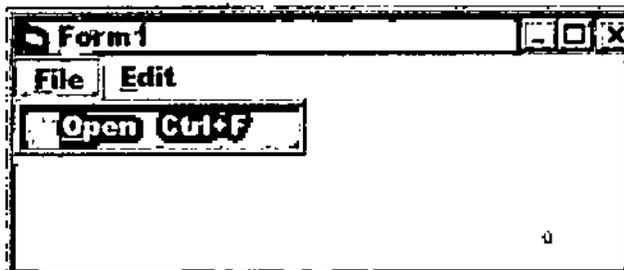
Let us see how this is done.

1. Open the Code view editor.
2. From the Objects list, choose select mnuFileOpen, as shown here.



```
Project1 - Form1 (Code)
mnuFileOpen. Click
Private Sub mnuFileOpen_Click()
    MsgBox "This will Open the project for you"
End Sub
```

3. Add code to display a message, as shown above.
4. Run the project.
5. You will get the form with two menus called File and Edit.
6. Click at File menu to get Open menu item.
7. Click at it.
8. You will get the response immediately, as shown below.



## Adding Separator Lines

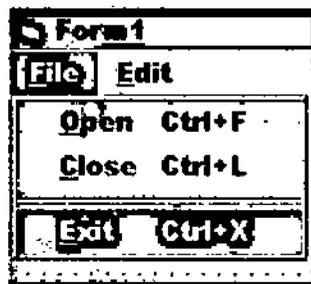
If you open your Visual Basic, you would see that in File menu, there is a separator line after New Project & Open Project and after Add Project & Remove Project. Let us now see how this separator line is created.

Follow the following steps to add separators

1. Open the Menu Editor.
2. Add more items called Close and Exit.
3. Now assume that we have to add a line after Close or between Close and Exit.
4. Click at Insert to add space after Close.
5. Press right arrow to move to create ellipse.

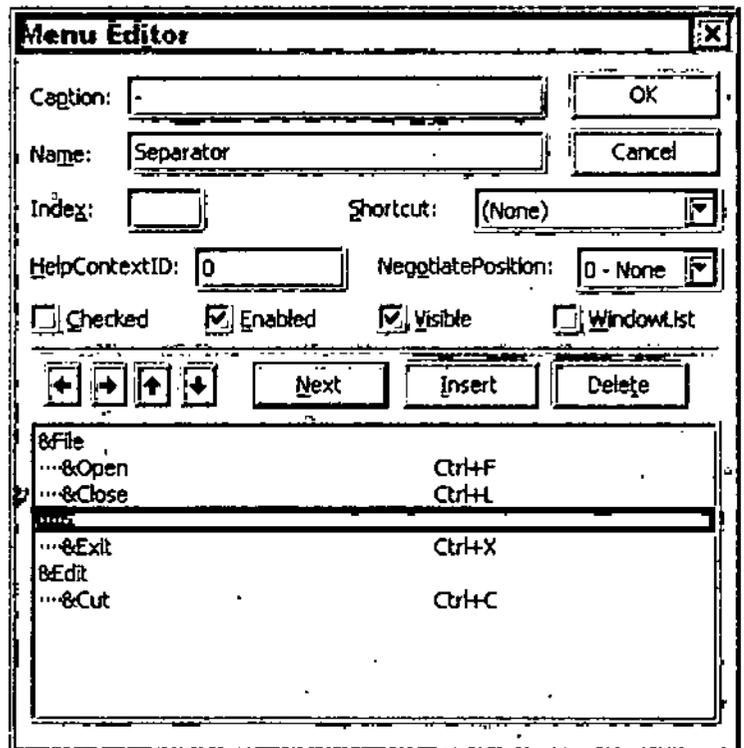
## NOTES

NOTES



6. Now type a hyphen (-) in the caption box.
7. Click Ok.
8. You will see the separator line after Close in the File menu of your form.

You must have wondered why we have not used the other buttons called Checked, Enabled, Visible and WindowList. Well let me explain them now. Let me start with Checked one.

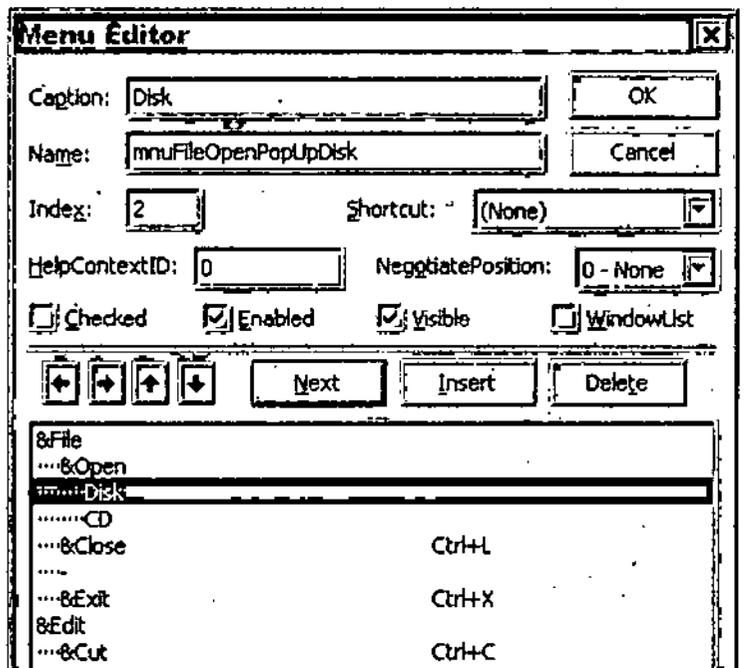


**Pop-Up Menus**

I am quite sure that you must have seen these type of menu which come on screen only when you press a menu item. You can call them as submenus of the main menu items.

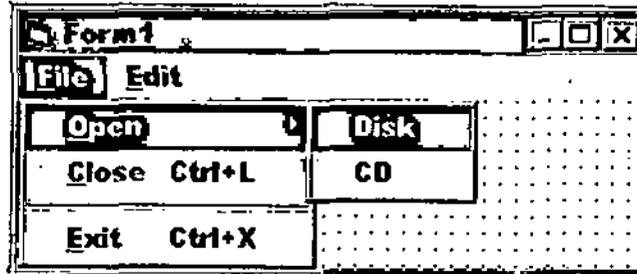
This is how they are created.

1. With the Menu Editor open, open the menu item with which you want to add the pop-up items.
2. In this add PopUp with the name of the item.
3. Create a new item with the use of Insert.
4. With the help of right arrow take this item to the second indent.



5. Type it as Disk as shown on the next page making its name as mnuFileOpenPopUpDisk.

6. Add another menu item as above and name it CD.
7. Now both Disk and CD are part of the Open menu item.
8. Click Ok.



9. You have the sub menu Disk and CD part of the Open menu item of the file menu, as shown here.

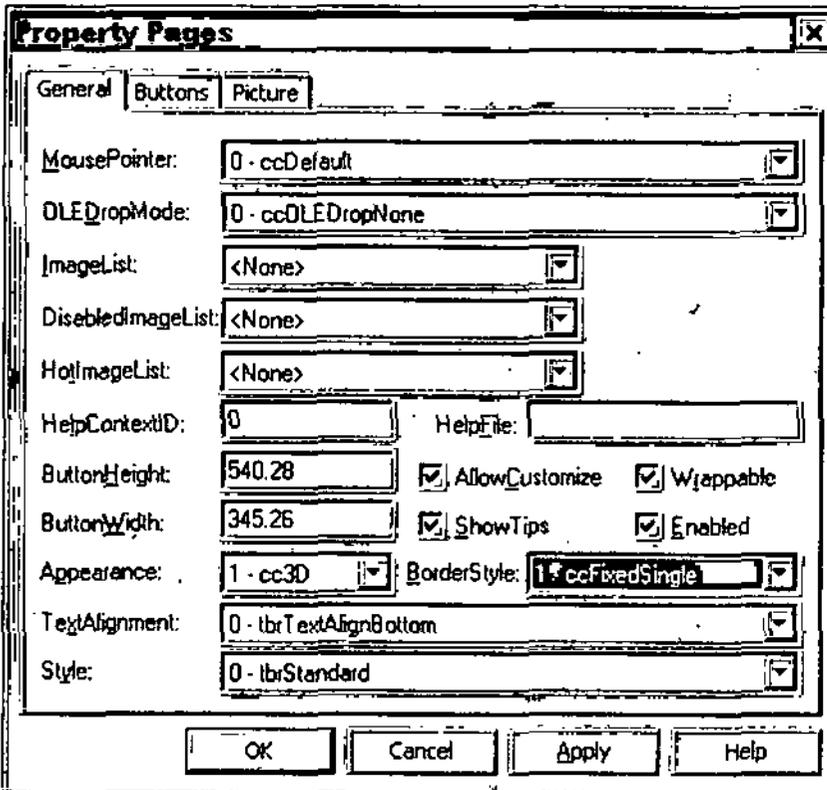
NOTES

## General Controls/Toolbars

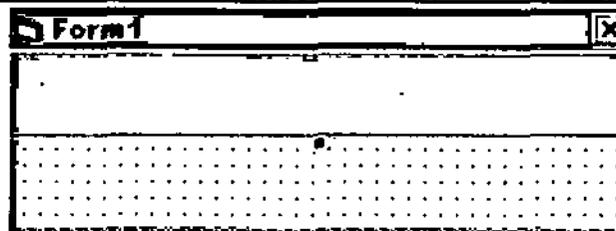
They are used to provide buttons to the user which he can click. To add buttons to a toolbar, you need to use it in conjunction with a linked ImageList control which functions as a kind of Image library.

Follow the following steps to add a toolbar to your form.

1. Toolbar control is usually there in the Toolbox. If not follow the earlier procedures to add it from the Components.



2. At this stage you must make sure that the form is open.
3. To toolbar to the form, double click the Toolbar control in Toolbox.
4. You can name it according to your liking in the Properties window.
5. Run the project. You will see the toolbar aligned at the top of the form.



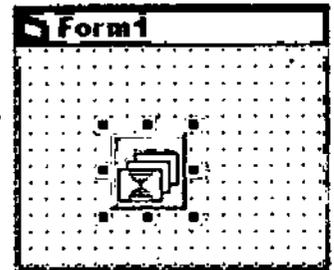
## Adding an ImageList

1. Double click the ImageList control in the Toolbox.

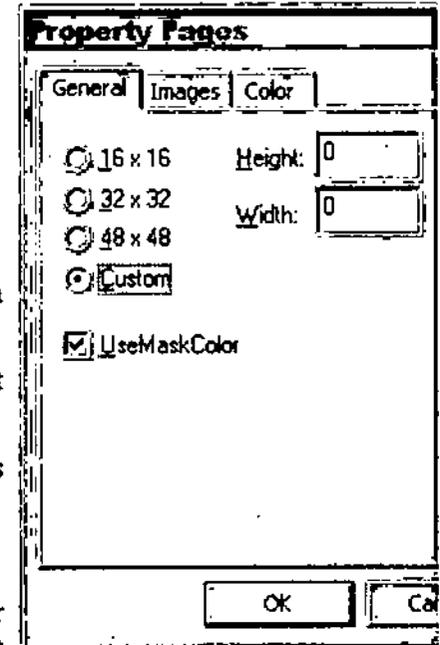
2. An image list is added to the form.

### Adding an ImageList with Images

1. Use the Properties window to change the name of the ImageList to imgDemo.
2. Open the property pages for the ImageList.
3. Click at the General tab.
4. Click the images tab.
5. Click Insert Picture.
6. The Select Picture dialog box opens.
7. Select the picture.
8. Click open to use the picture.
9. A small icon of the picture will be seen in the Image tab.
10. If you have to add more pictures repeat the process.
11. Click when you have all the pictures required.

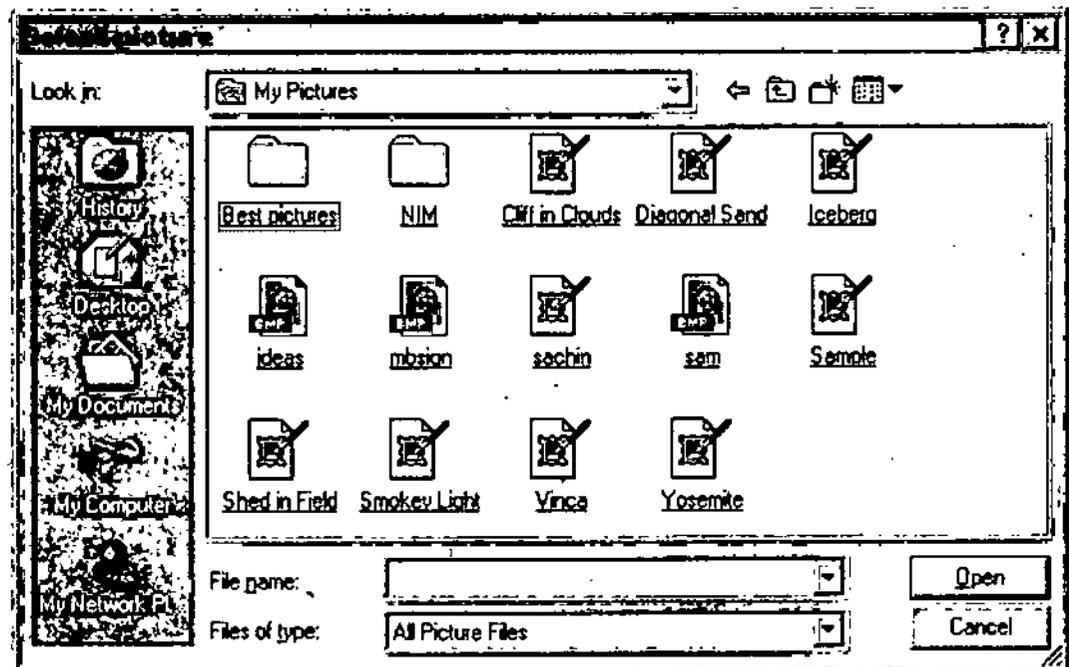


NOTES

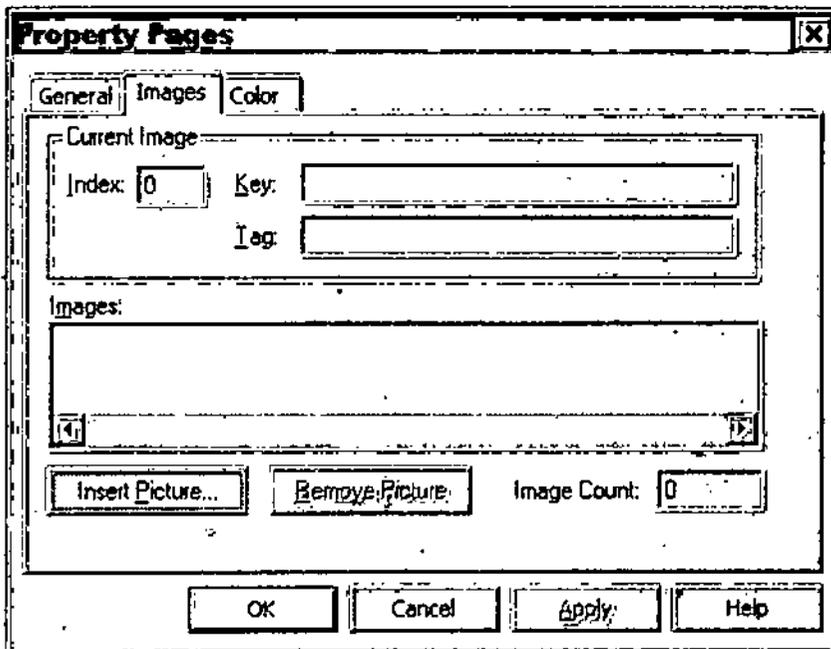


### Adding Buttons to the Toolbar

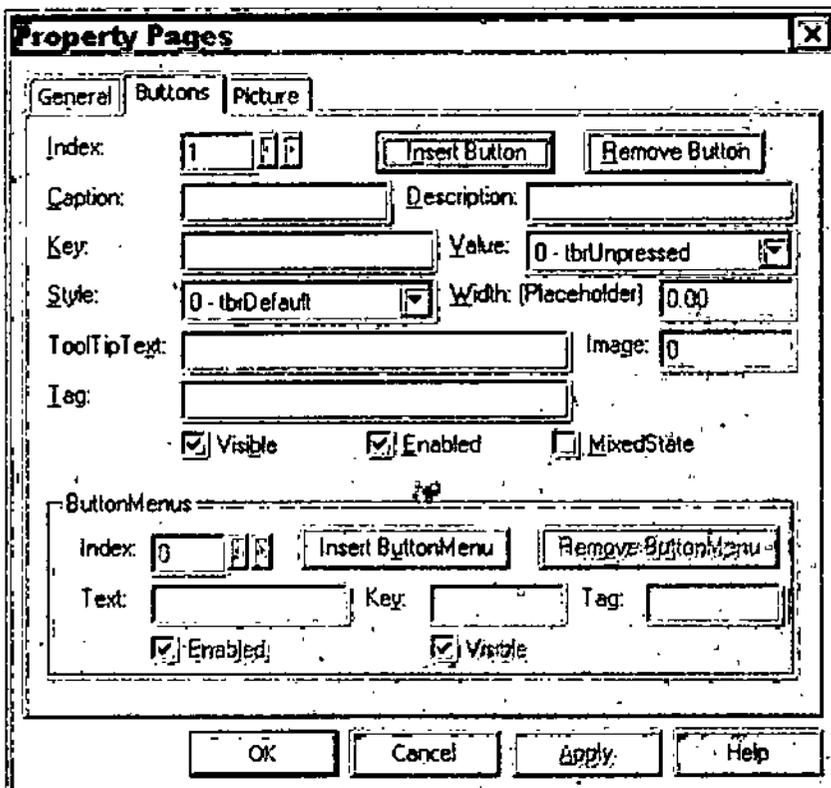
1. With the property pages for the toolbar open, select the Buttons tab. Notice that the Index field is grayed.
2. Click Insert Picture. The Index field is now active.
3. In the Image field, put the number of the picture in the ImageList.



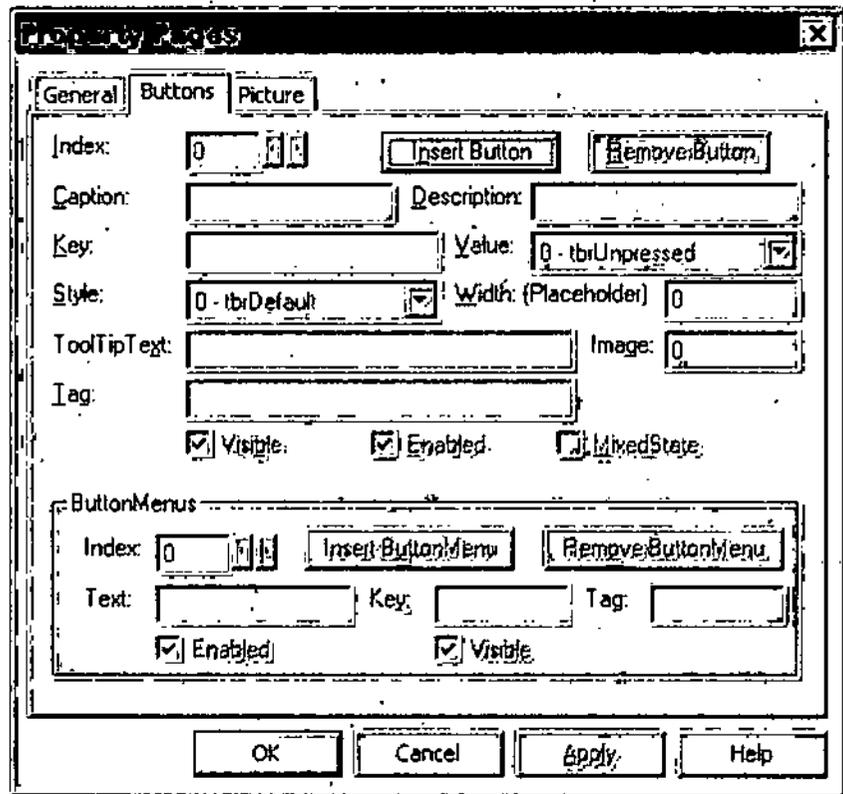
NOTES



4. Tab away from the field. A button with the first ImageList picture appears in the toolbar.
5. Add a caption if you want text to appear on the button.
6. Add a name to the Key field that will be used to reference the button in code.
7. Add ToolTip text is you want.
8. Click Apply.
9. Click Insert Picture, and repeat the process for the second button.



NOTES



10. Repeat the process for each of the buttons that will be in the toolbar.
11. Run the project.

Your toolbar, with all the buttons will appear along the top of the form.

### Adding Status Bars

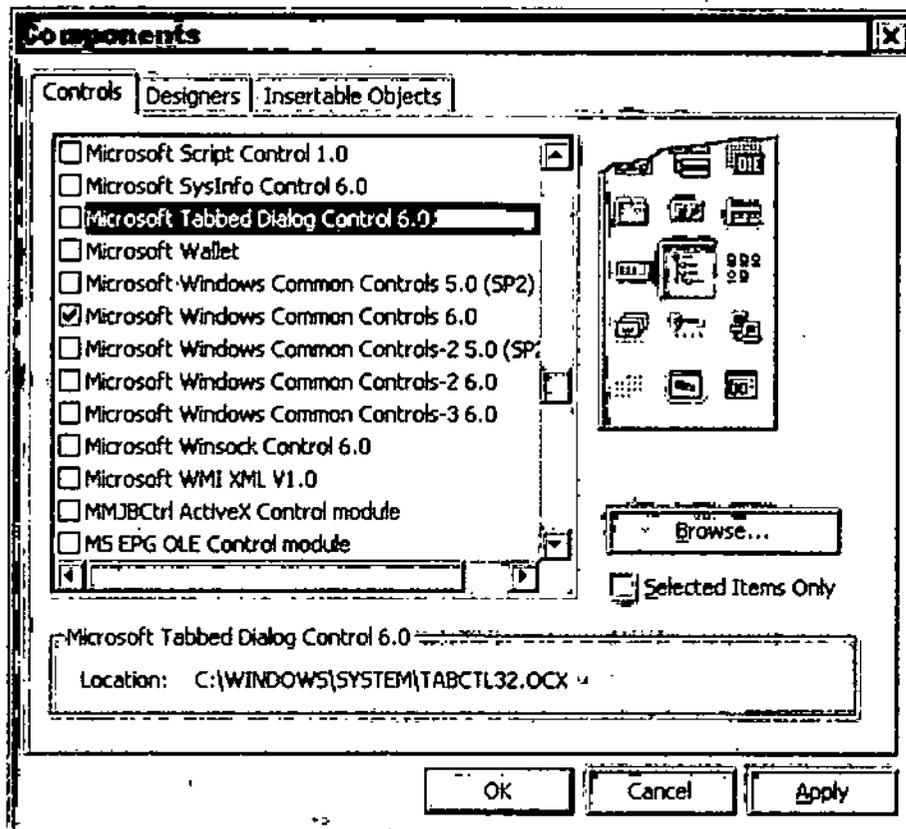
These are used to give your application a good look. It helps users of your application to keep track of program options they have selected. But, let us add this in the ToolBox.

Follow the following steps to add StatusBar to the ToolBox.

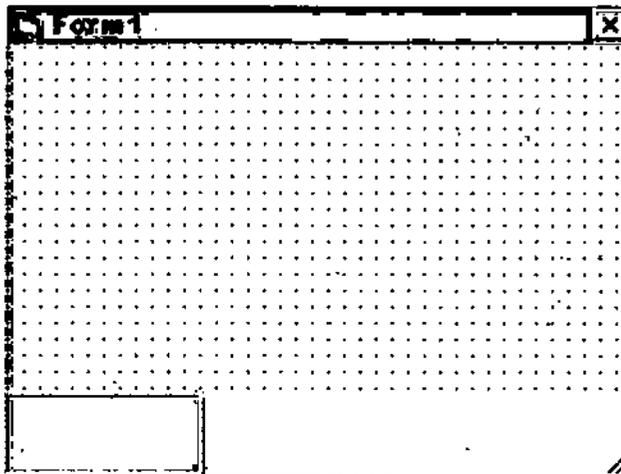
1. With the form selected click at Project menu and Components.
2. From here you choose Microsoft Windows Common Controls 6.0.
3. Click at the box next to it.
4. Click at Ok to close the dialog box.
5. The StatusBar is now available in the ToolBox.

#### Adding StatusBar to the form:

1. From the Properties window set the BorderStyle of your form to 1-Fixed Single.
2. On the form double-click the StatusBar control in the ToolBox to add it to the form.
3. You can name the StatusBar from the Properties window.



NOTES



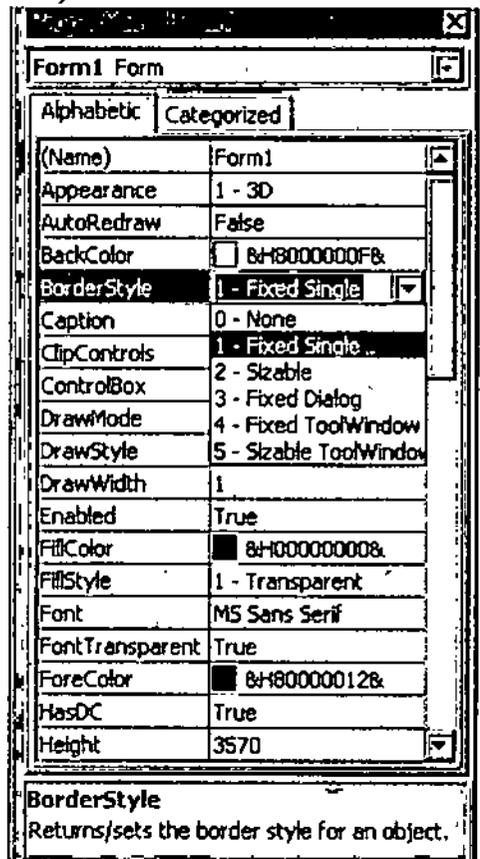
**Adjusting the First Panel:**

1. Right click on the StatusBar to get the Property Pages dialog box.
2. Click at Panels tab. The default panel Style property setting is 0-sbrText, meaning that the panel displays text. Various other options are:

<i>Value and Constants</i>	<i>Description</i>
0 - sbrText	Default; panel displays text
1 - sbrCaps	Displays CAPS in bold when Caps Lock is enabled and grayed when disabled.
2 - sbrNum	Displays NUM in bold when Num Lock is enabled and grayed when disabled.

NOTES

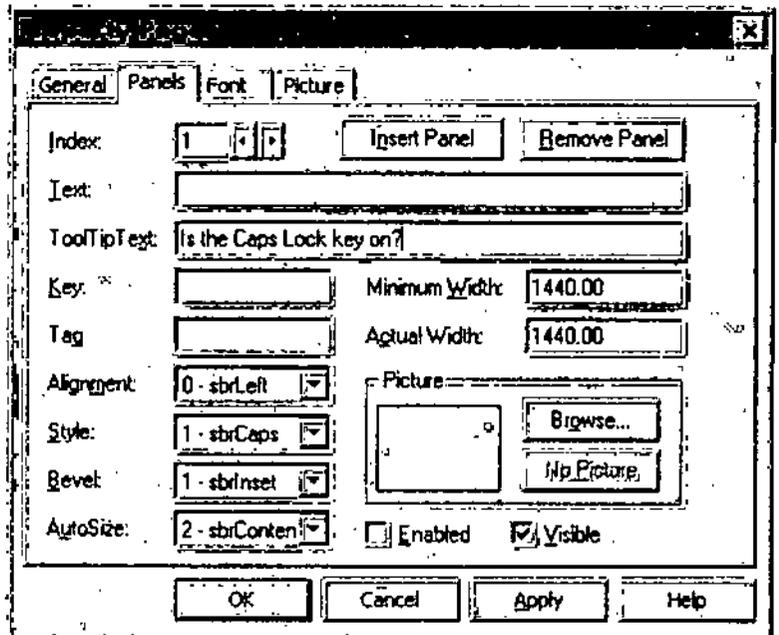
- 3 - `sbrIns` Displays **INS** in bold when Insert mode is enabled and grayed when disabled.
- 4 - `sbrScri` Displays **SCRL** in bold when Scroll Lock is enabled and grayed when disabled.
- 5 - `sbrTime` Displays the current time, using the system format.
- 6 - `sbrDate` Displays the current date, using the system format.
- 7 - `sbrKana` Displays **KANA** in bold when Scroll Lock is enabled and grayed when disabled.



3. Here I have set the Style property to 1-`sbrCaps`.

4. The Text property is ignored unless the Style property is set to 0 - `sbrText`, so leave the Text field empty.

5. In the ToolTip Text field, enter Is the Caps Lock key on? This text is displayed when the user runs the mouse across the panel.



6. Click Apply or Ok.

7. Run the project.

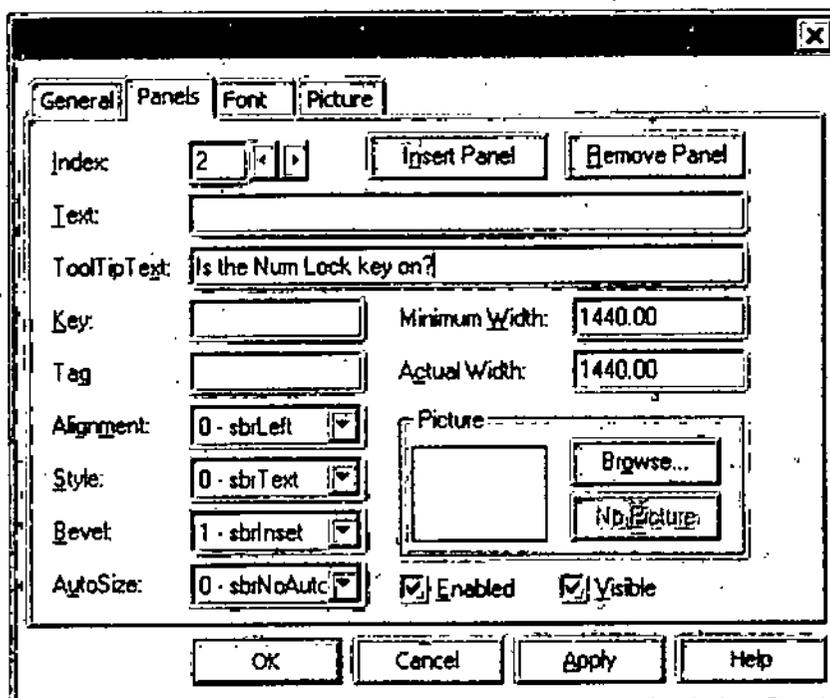
The panel displays CAPS in bold if Caps Lock is on; otherwise, CAPS is grayed. In addition, the ToolTip text you specified is displayed.

### Adding StatusBar Panels:

1. With Pages Property panel open by right clicking.
2. Now click at the Insert panel.
3. Click the right arrow to the right of the Index box to move to the second panel.
4. Index is now 2.
- 5: Set Style property of the second panel to 2 - sbrNum.
- .6. Set the ToolTip Text field to Is the Num Lock key on?
7. Click at Ok or Apply.

NOTES

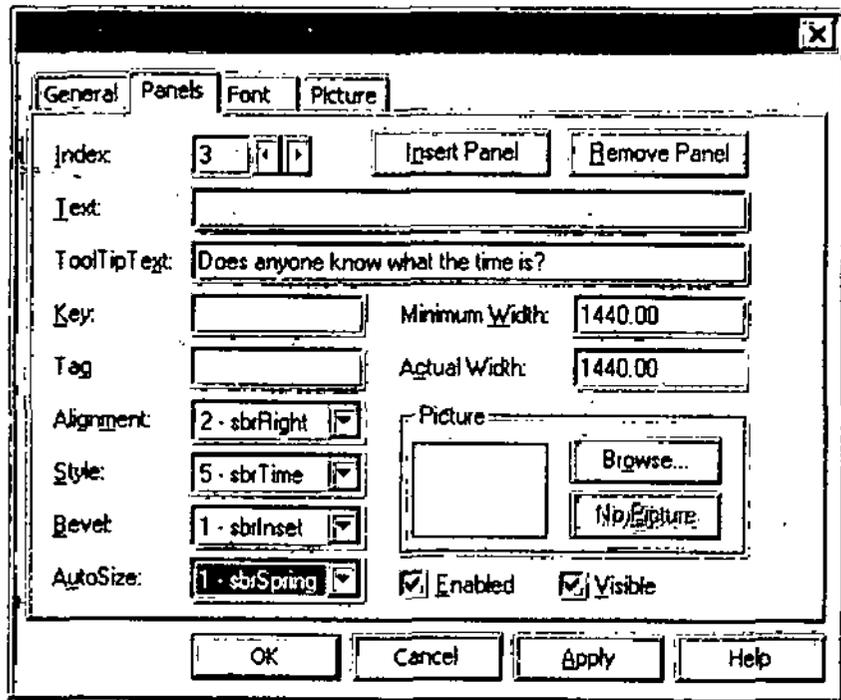
When you run the project, the second panel displays NUM in bold when Num Lock is enabled and NUM in gray if it is not.



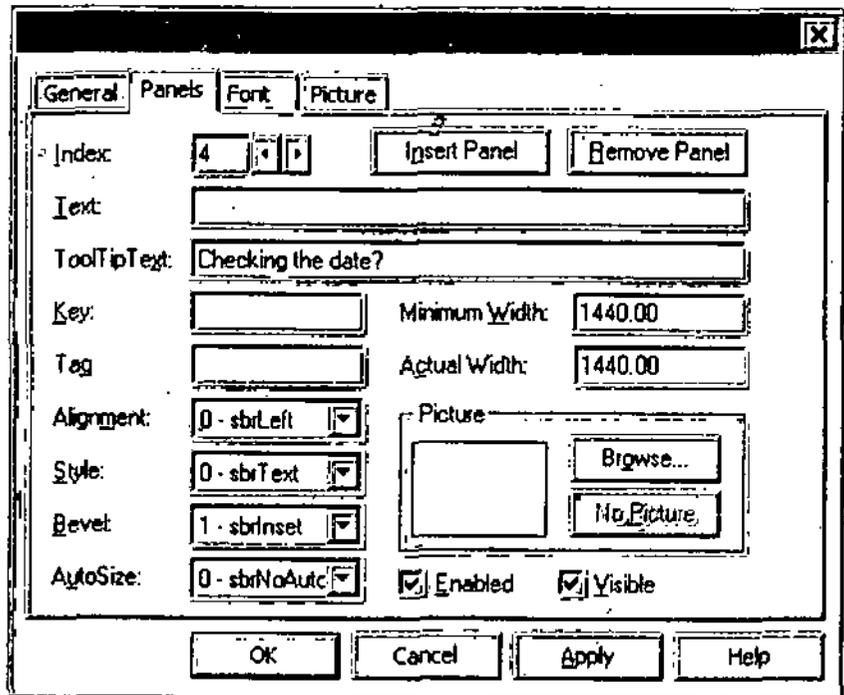
### Adding Time and Date Panels

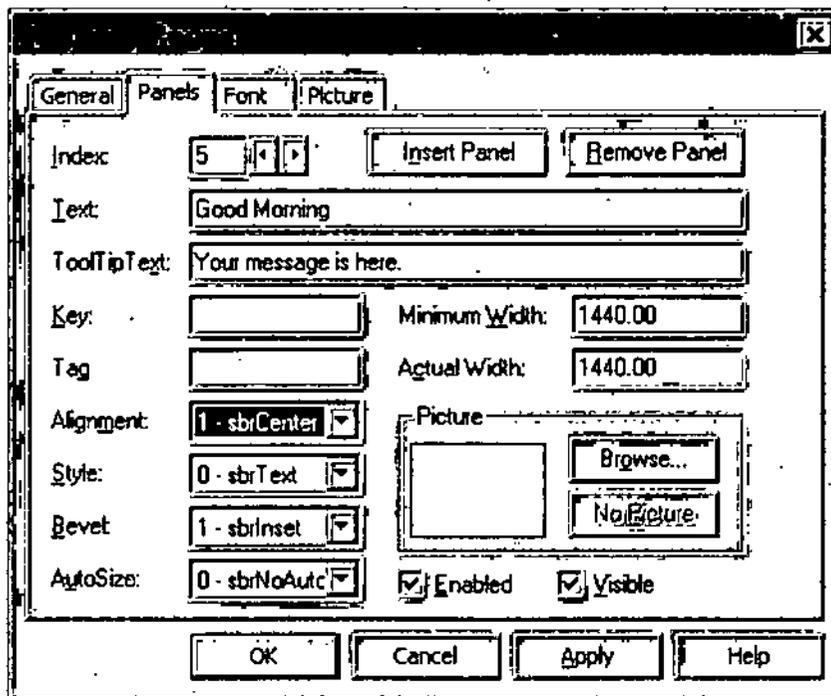
1. Click at the Insert Panel button to add another panel.
2. Click the arrow button to the right of the Index box to move to the fourth panel.
3. Change the fourth panel's Style property to 5 - sbrTime.
4. Set the panel Alignment property to 2 - sbrRight. This aligns the time in the right of the panel.
5. Add ToopTip text for the panel.
6. Click Apply.
7. Click the Insert Panel button to add another panel.

NOTES

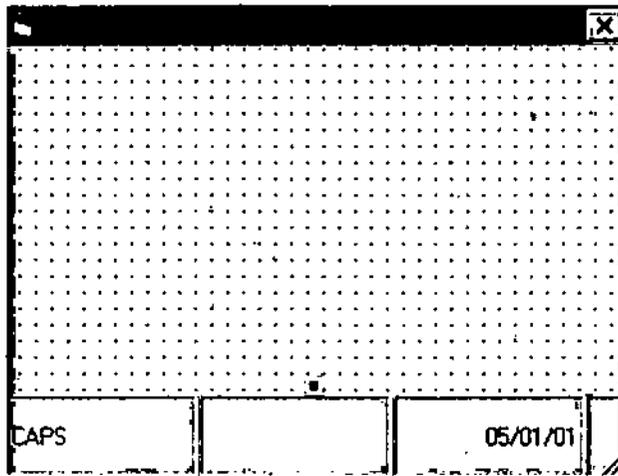


8. Click the arrow button to the right of the Index box to move to the fifth panel.
9. Change the fourth panel's Style property to 6 - sbrDate.
10. Set the panel Alignment property to 2 - sbrRight. This aligns the time in the right of the panel.
11. Add ToolTip text for the panel.
12. Click Ok.
13. Run the project.





## NOTES



The Status Bar displays all five panels.

### **Adding a Text Panel**

1. Click the Insert Panel button to add another panel.
2. Click the arrow button to the right of the Index box to move to the third panel.
3. Leave the third panel's Style property set to the default 0 - sbrText.
4. Set the panel Alignment property to 1 - sbrCenter. This aligns the text in the middle of the panel.
5. Add text to display in the Panel's Text property, for example, Good Morning.
6. Add ToolTip text for the panel—for example, Write your message here.
7. Click Apply or Ok.

8. Run the project. The text you specified is displayed in the panel.
9. Pause your mouse over the panel. The ToolTip text you specified appears.

---

## CUSTOMIZING A FORM

---

### NOTES

When a Visual Basic project is run these forms become the Windows of the programs. This means that forms are the basis of the user interface in any program written in Visual Basic.

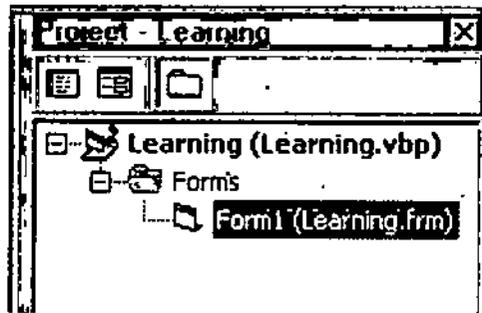
The new project usually has a form attached to it. However, you can add a new form to a project easily. This is how you do it.

---

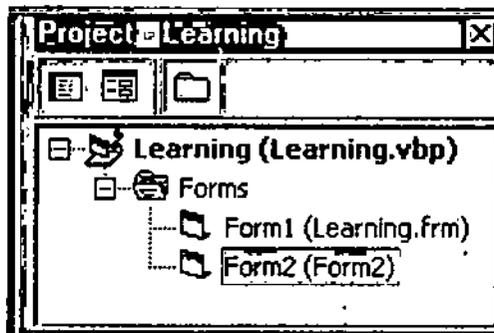
## ADDING A FORM TO A PROJECT

---

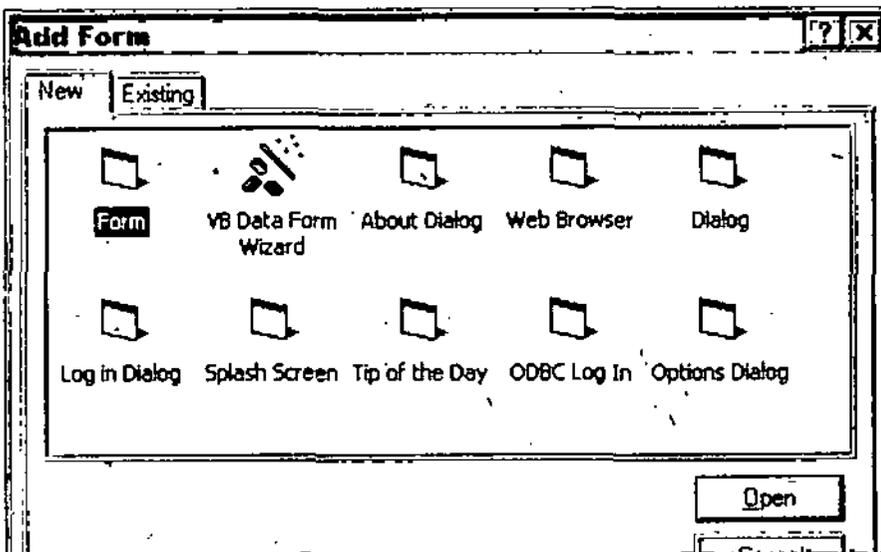
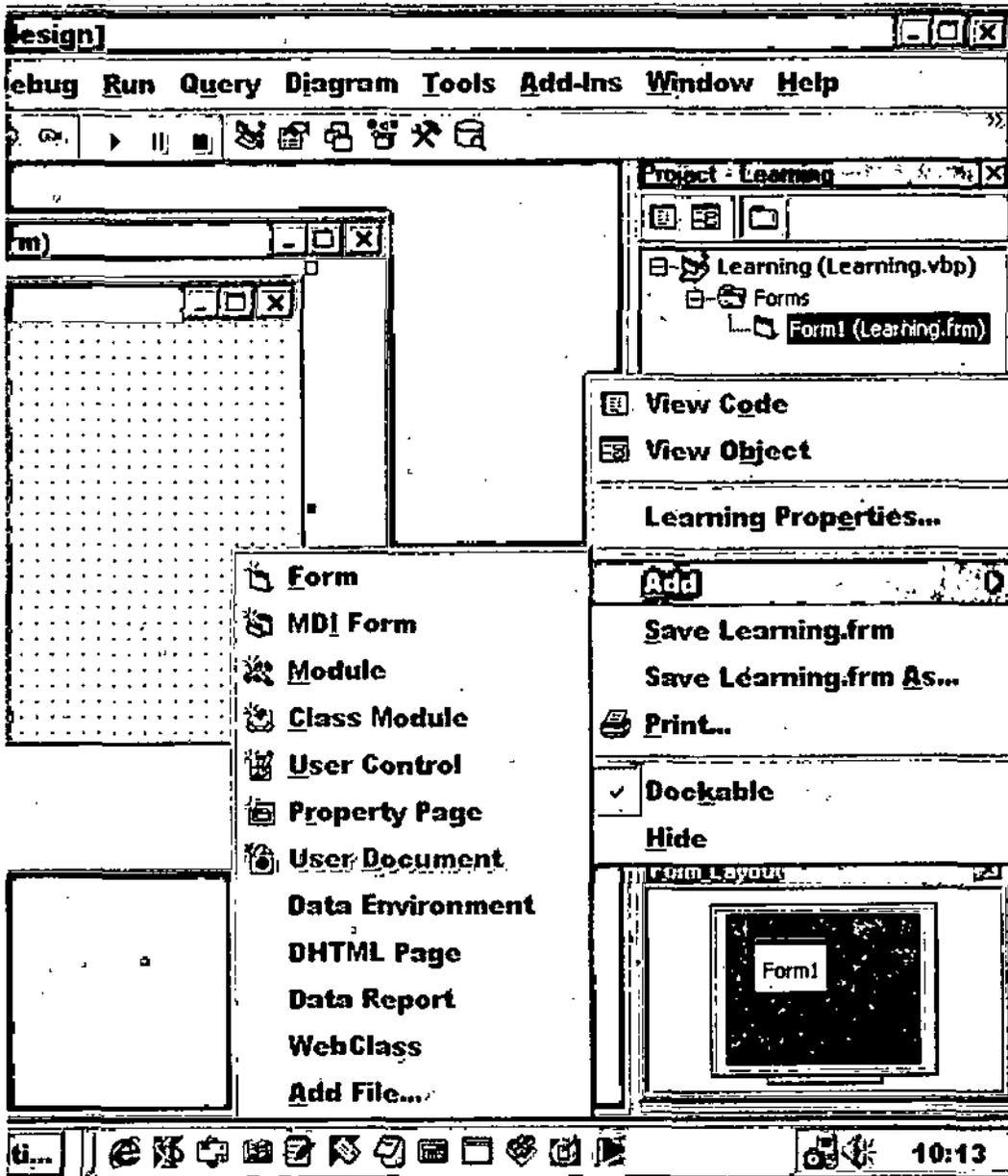
1. If the project explorer, as shown below, is not open on the screen, you can open it by clicking at Project Explorer from the View menu.



2. Right click within the project explorer to get the pop-up menu, as shown on the next page.
3. Click at Add to get list of items which can be added.
4. Choose Form to get the Add Form dialog box, shown on next page.
5. Click at New panel to choose Form.
6. Click at Open to add a new form to the project.
7. A new form is added in the project explorer dialog box, as shown next.



NOTES

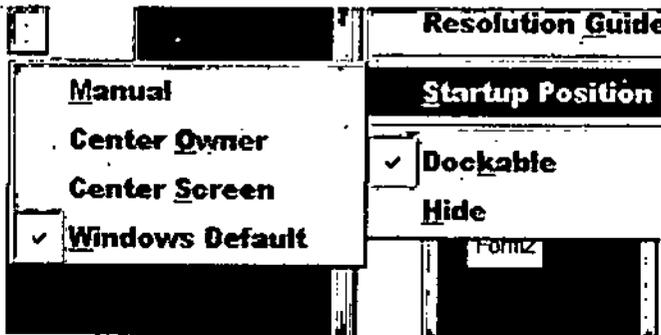


## Changing Position of the form

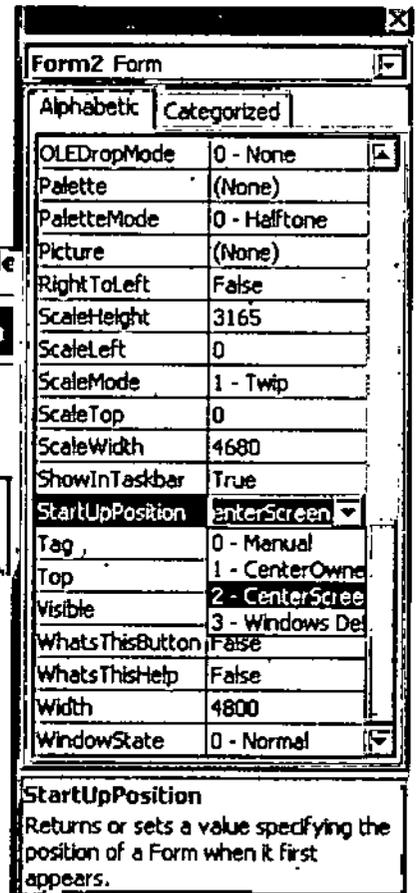
You can change the position of the form in the window by using the following steps.

1. In the form layout right click to get the pop-up menu, as shown below.

NOTES



2. Click at Startup Position to get another pop-up menu.
3. From here you can make your choice of making the form in the Centre of the Screen or any of the 3 other options, i.e., Manual, Center Owner, Windows Default.
4. You can make the same changes in the Form position from the Properties menu, as shown on the right.



## BUILDING USER CONTROL COMMAND BUTTONS

The common dialog control provides a standard set of dialog boxes for following operations:

- a. opening and saving files
- b. setting print options
- c. selecting colors and fonts

The control also has the ability to display Help by running the Windows Help engine.

The common dialog control provides an interface between Visual Basic and the procedures in the Microsoft Windows dynamic-link library `Commdlg.dll`.

*To create a dialog box using this control, `Commdlg.dll` must be in your Microsoft Windows\System directory.*

You can use the common dialog control in your applications by adding it to a form and setting its properties. The dialog displayed by the control is determined by the methods of the control. At run time, a dialog box is displayed or the Help engine is executed when the appropriate method is invoked. At design time, the common dialog control is displayed as an icon on a form. This icon can not be resized and is not available at run time.

The common dialog control allows you to display the following commonly used dialog boxes:

- Open
- Save As
- Color
- Font
- Print

NOTES

To use the common dialog control, do this:

1. Add the common dialog control to the toolbox by selecting Components from the Project menu. Locate and select Microsoft Common Dialog Control 6.0 in the Controls tabbed dialog, then click the OK button.
2. On the toolbox, click the CommonDialog control and draw it on a form.

*When you draw a common dialog control on a form, it automatically resizes itself. Common dialog control is invisible at run time.*

3. At run time, use the appropriate method, as listed in the following table, to display the desired dialog.

<i>Method</i>	<i>Dialog displayed</i>
ShowOpen	Open
ShowSave	Save As
ShowColor	Color
ShowFont	Font
ShowPrinter	Print
ShowHelp	Invokes Windows Help

### File Open and Save As Dialog Box

The Open dialog box allows the user to specify a drive, a directory, a file name extension and a file name.

The Save As dialog box is identical to the Open dialog box in appearance, except for the dialog's caption and file names appearing dimmed out. At run time, when the user chooses a file and closes the dialog box, the FileName property is used to get the selected file name.

#### Properties for File Open and Save As Dialog Box

<i>Property</i>	<i>Determines</i>
DefaultExt	Sets the default extension for the files in the dialog box.
FileName	Returns the path and full name of the file selected.
FileTitle	Returns the file name only.

NOTES

Filter	Filters the kind of file according to the extensions given. Multiple extensions can be given using   (pipe) symbol.
FilterIndex	Used to specify look and behaviour of the dialog box.
InitDir	Specifies the initial directory.
MaxFileSize	Sets the maximum size of the file name including all the path information.

**Flags for the File Dialog Boxes**

<i>Flags</i>	<i>Characteristics</i>
CdIOFNAllowMultiSelect	Used to allow the users multiple selection of files from the box.
CdIOFNCreateprompt	Pops up a message box that inquires to create a new file, when a new name is given by the user in the file name.
CdIOFNFileMustExist	Pops up a message box if a user tries to enter as file or path that does not exist.
CdIOFNPathMustExist	
CdIOFNHideReadOnly	The first flag hides the Read only check box in the dialog box.
CdIOFNReadOnly	The second flag makes the Read only check box checked when the dialog first appears.
CdIOFNOverViewPrompt	If you use Show Save and file already exists, Visual Basic will pop up a message box asking the user to confirm that he or she wants to overwrite the File.
CdIOFNShareAware	If this flag is set, windows will ignore sharing violation errors.
CdIOFNHelpButton	Displays help Button in the dialog Box.

**Color Dialog Box**

The Color dialog box allows the user to select a colour from a palette or to create and select a custom colour. At run time, when the user chooses a colour and closes the dialog box, then you use the Color property to get the selected colour.

To display the Color dialog box, do this:

1. Set the Flags property for the common dialog control to the Visual Basic constant `cdICCRGBInit`.
2. Use the ShowColor method to display the dialog box.

**Font Dialog Box**

The Font dialog box allows the user to select a font by its size, color and style. Once you make selections in the Font dialog box, you can use the properties containing information given in the following table.

<i>Property</i>	<i>Determines</i>
Color	The selected color. To use this property, you must first set the Flags property to <code>cdlCFEffects</code> .
FontBold	Whether bold was selected.
FontItalic	Whether italic was selected.
FontStrikethru	Whether strikethrough was selected
FontUnderline	Whether underline was selected
FontName	The selected font name.
FontSize	The selected font size.

NOTES

### Flags Associated with Font Dialog Box

<i>Flags</i>	<i>Characteristics</i>
<code>cdlCFHelpButton</code>	Determines whether the dialog box displays a Help button.
<code>cdlCFEffects</code>	Determines whether you want to allow strikeout, underline and color effects.
<code>cdlCFBoth</code>	Shows both screen and printer fonts.
<code>cdlCFApply</code>	Determines whether the dialog box enables the Apply button.
<code>cdlCFANSIOOnly</code>	Determines whether the dialog box displays only the fonts that include the Windows character set.
<code>cdlCFNoVectorFonts</code>	Determines whether the dialog box should not display vector-based fonts.
<code>cdlCFNoSimulations</code>	Determines whether the dialog box will not allow graphic device interface (GDI) font simulations.
<code>cdlCFLimitsize</code>	Determines whether the dialog box should show only font sizes between those specified by the Max and Min properties.
<code>cdlCFFixedPitchonly</code>	Determines whether the dialog box should display only fixed-pitch fonts.
<code>cdlCFWYSIWYG</code>	Determines whether the dialog box should show only fonts to both screen and printer.
<code>CdlCFForceFontExit</code>	Determines whether a message box pops up if a user selects a font style that does not exist.
<code>CdlCFScalableonly</code>	Determines whether the dialog box will allow the user to only select scalable fonts, such as True type fonts.
<code>cCdlCFTTOnly</code>	Specifies that the dialog box should allow the user to select only True Type fonts.
<code>cCdlCFNoFaceSel</code>	This is returned if no font name is selected.

cdlCFNoStyleSel This is returned if no font style is selected.

cdlCFNoSizeSel This is returned if no font size is selected.

### Print Dialog Box

NOTES

The Print dialog box allows the user to specify how output should be printed. The user can specify a range of pages to be printed, a print quality, a number of copies and so on. This dialog box also shows information about the currently installed printer and allows the user to configure or reinstall a new default printer.

#### Properties for Print Dialog Box

<i>Property</i>	<i>Determines</i>
Copies	The number of copies to print.
FromPage	The page to start printing.
ToPage	The page to stop printing.
hDc	The device context for the selected printer.
Orientation	The page's orientation (portrait or landscape).

#### Flags Associated with Print Dialog Box

Flags associated with the Print Dialog box are shown in the following table.

*This dialog box does not actually send data to a printer. It allows users to sepecify how they want data printed. You must write code to print the data in the format they select.*

<i>Flags</i>	<i>Characteristics</i>
cdlPDAllPages	Returns the value or set the All Pages option button.
cdlPDCollate	Returns the value or set the Collate check box.
cdlPDDisablePrintToFile	Disables the Print to File check box.
cdlPDHidePrintToFile	Hides the Print to File check box.
cdlPDNoPageNums	Returns the value or sets the page option button.
cdlPDNoSelection	Disables the Selection option button.
cdlPDNoWarning	Prevents Visual Basic from issuing a warning message when there is no default printer.
cdlPDPageNums	Returns the value or sets the pages option button.
cdlPDPrintSetup	Displays Print Setup dialog box rather than the printer dialog box.
cdlPDPrintToFile	Returns the value or sets the Print to the File check box.
cdlPDReturnDC	Returns a device context for the printer selection from the HDC property of the dialog box.

cdIPDRReturnDefault	Returns the default printer name.
cdIPDRReturnI	Returns an information context for the printer selection value from the HOC property of the dialog box.
cdIPDSelection	Returns the value or sets the Selection option button.
cdIPDHelpButton	Determines whether the dialog box displays the Help button.
cdIPDUseDevModelCopies	Sets support for multiple copies.

NOTES

## Help Dialog Box

The ShowHelp method of the common dialog control allows you to display a Help file.

To display a Help file using the ShowHelp method

1. Set the HelpCommand and HelpFile properties.
2. Use the ShowHelp method to display the specified Help file.

### Example

The following application implements all the above properties that are listed in the above tables. Create a form with Common Dialog control, six Command buttons and TextBox. The following table lists the property settings for the objects in the application.

<i>Object</i>	<i>Property</i>	<i>Settings</i>
CommonDialog1	Name	CommonDialog1
Command1	Name	cmdOpen
	Caption	Open
Command1	Name	cmdSave
	Caption	Save
Command1	Name	cmdColor
	Caption	Color
Command1	Name	cmdFont
	Caption	Font
Command1	Name	cmdPrinter
	Caption	Printer
Command1	Name	cmdHelp
	Caption	Help
Text111	Name	txtFont
	Text	CommonDialog





**Events In Common Dialog Control**

Add the code shown in the following program to the cmdColor\_Click event procedure. This event changes the background color of the form when user selects particular color from Color dialog box.

NOTES

```

Private Sub cmdColor_Click()
  'Set Cancel to True.
  CommonDialog1.CancelError = True
  On Error GoTo ErrHandler
  'Set the Flags property. CommonDialog1.Flags =
    cdLCCRGBInit
  'Display the Color dialog box.
  CommonDialog1.ShowColor
  'Set the form's background color to the selected
  'color.
  frmDialog.BackColor = CommonDialog1.Color
Exit Sub
ErrHandler:
  'User pressed Cancel button.
  Exit Sub
End Sub

```

Add the code shown in the following figure to the cmdFont\_Click event procedure. This event will change the style of the font when user selects the desired options from the Font dialog box.

```

Private Sub cmdFont_Click()
  'Set Cancel to True.
  CommonDialog1.CancelError = True
  On Error GoTo ErrHandler
  ' Set the Flags property.
  CommonDialog1.Flags = cdLCFBoth Or cdLCFEffects
  ' Display the Font dialog box.
  CommonDialog1.ShowFont
  ' Set text properties according to user's selections.
  txtFont.Font.Name = CommonDialog1.FontName
  txtFont.Font.Size = CommonDialog1.FontSize
  txtFont.Font.Bold = CommonDialog1.FontBold
  txtFont.Font.Italic = CommonDialog1.FontItalic
  txtFont.Font.Underline = CommonDialog1.FontUnderline
  txtFont.Font.Strikethru = CommonDialog1.FontStrikethru
  txtFont.ForeColor = CommonDialog1.Color
Exit Sub
ErrHandler:

```

```
' User pressed Cancel button.
```

```
Exit Sub
```

```
End Sub
```

Add the code shown in the following figure to the cmdHelp\_Click event procedure.

```
Private Sub cmdHelp_Click()  
  ' Set Cancel to True.  
  CommonDialog1.CancelError = True  
  On Error GoTo ErrHandler  
  ' Set the HelpCommand Property  
  CommonDialog1.HelpCommand = cdlHelpForceFile  
  ' Specify the Help file.  
  CommonDialog1.HelpFile = "c:\Windows\CommonDialog.hlp"  
  ' Display the Windows Help engine.  
  CommonDialog1.ShowHelp  
Exit Sub  
ErrHandler:  
  ' User pressed Cancel button.  
  Exit Sub  
End Sub
```

Add the code shown in the following figure to the cmdOpen\_click event procedure

```
Private Sub cmdOpen_click()  
  ' CancelError is True.  
  On Error GoTo ErrHandler  
  'Set filters.  
  CommonDialog1.Filter = A"ll files (*.*) *.*|Text Files  
    (*.txt) |*.txt|Batch Files (*.bat) |*.bat"  
  ' Specify default filter.  
  CommonDialog1.FilterIndex = 2  
  ' Display the Open dialog box.  
  CommonDialog1.ShowOpen  
Exit Sub  
ErrHandler:  
  ' User pressed Cancel button.  
  Exit Sub  
End Sub
```

Add the code shown in the following figure to the cmdPrinter\_Click event procedure.

```
Private Sub cmdPrinter_Click()  
  Dim BeginPage, EndPage, NumCopies, Orientation, i  
  ' Set Cancel to True.  
  CommonDialog1.CancelError = True  
  On Error GoTo ErrHandler
```

NOTES

## NOTES

```

    \ Display the Print dialog box.
    CommonDialog1.ShowPrinter
    \ Get user-selected values from the dialog box.
    BeginPage = CommonDialog1.FromPage
    EndPage = commonDialog1.ToPage
    NumCopies = CommonDialog1.Copies
    Orientation = CommonDialog1.Orientation
    For i = 1 To NumCopies
        \ Put code here to send data to your printer.
    Next
    Exit Sub
ErrorHandler:
    \ User pressed Cancel button.
    Exit Sub
End Sub

```

Add the code shown in the following figure to the cmdSave\_Click event procedure.

```

Private Sub cmdSave_Click()
    \ CancelError is True.
    On Error GoTo ErrorHandler
    \ Set filters.
    CommonDialog1.Filter = "All Files (*.*) |*. *IText Files
        (*.txt) | *.txtIBatch Files (*.bat) |*.bat"
    \ I Specify default filter.
    CommonDialog1.FilterIndex = 2
    \ Display the Save dialog box.
    CommonDialog1.ShowSave
    Exit Sub
ErrorHandler:
    \ User pressed Cancel button.
    Exit Sub
End Sub

```

## Mouse Controls

Application can be made powerful by using mouse events, depending on which mouse button is used or whether the Shift, Ctrl or ALT key is pressed. To provide these options, you use the arguments button and Shift with the MouseDown, MouseUp and MouseMove event procedures.

The MouseDown, MouseUp and MouseMove events use the button argument to determine which mouse button or buttons are pressed. The button argument is one bit-field argument – a value in which each bit represents a state or condition. These values are expressed as integers. The three least-significant (lowest) bits represent the left, right and middle mouse buttons. The default value of each bit is 0 (False). If no buttons are pressed, the binary value of the three bits is 000. If you press the left button, the binary value or pattern, changes to 001. The left button bit-value changes from 0 (False) to 1 (True).

The button argument uses either a decimal value or a constant to represent these binary patterns. The following table lists the binary value of the bits, the decimal equivalent and the Visual Basic constant:

<i>Binary Value</i>	<i>Decimal Value</i>	<i>Constant</i>	<i>Meaning</i>
001	1	vbLeftButton	The left button is pressed.
010	2	vbRightButton	The right button is pressed.
100	4	vbMiddleButton	The middle button is pressed.

NOTES

### The MouseDown Event

MouseDown is the most frequently used event out of the three mouse events. It can be used to reposition controls on a form at run time or to create graphical effects. The MouseDown event is triggered when a mouse button is pressed.

The MouseDown event is combined with the Move method to move a command button to a different location on a form. The new location is determined by the position of the mouse pointer. When the user clicks anywhere on the form (except on the control), the control moves to the cursor location.

A single procedure, Form-MouseDown, performs this action:

```
Private Sub Form-MouseDown (Button As Integer, shift As
    Integer, X As Single, Y As Single)
    Command1.Move X, Y
End Sub
```

Move method places the command button control's upper-left corner at the location of the mouse pointer, indicated by the x and y arguments. You can revise this procedure to place the center of the control at the mouse location as given below:

```
Private Sub Form-MouseDown (Button As Integer, Shift As
    Integer, X As Single, Y As Single)
    Command1.Move (X - Command1.Width / 2) ,
    Y - Command1.Height / 2)
End Sub
```

### MouseMove Event

The Move event occurs when the mouse pointer is moved across the screen. Both forms and controls recognize the Mouse - Move event while the mouse pointer is within their borders. Syntax for this is as follows

```
Private Sub Form_MouseMove(button As Integer, shift As
    Integer, x As Single, y As single)
```

### MouseUp Event

The MouseUp event occurs when the user releases the mouse button MouseUp is a useful companion to the MouseDown and Move events.

Syntax for this is as follows:

```
Private Sub Form_MouseUp(button As Integer, shift As
    Integer, x As Single, y As single)
```

The following example demonstrates a simple paint application. The MouseDown event procedure works with a related MouseMove event procedure to enable painting when any mouse button is pressed and dragged. The MouseUp event procedure disables painting size.

Add the code shown below in Form Module.

NOTES

```

Dim PaintNow As Boolean
Private Sub Form_MouseDown(Button As Integer, Shift As
    Integer, X As Single, Y As Single)
    PaintNow = True ' Enable painting.
End Sub
Private Sub Form_MouseUp(Button As Integer, Shift As
    Integer, X As Single, Y As Single)
    PaintNow = False ' Disable painting.
End Sub
Private Sub Form_MouseMove(Button As Integer, Shift As
    Integer, X As Single, Y As Single)
    If PaintNow Then
        PSet (X, Y) , Draw a point.
    End If
End Sub
Private Sub Form_Load()
    DrawWidth.10 ' Use wider brush.
End Sub

```

### Detecting Button and Release Events

Detecting Button argument is used with MouseDown to determine which button is being pressed and with MouseUp to determine which button has been released. Because only one bit is set for event, you can not test for whether two or more buttons are being used at the same time. In other words, MouseDown and MouseUp only recognize one button press at a time.

The procedure shown in following program prints a message when button is released.

```

Private Sub Form_MouseDown (Button As Integer, Shift As
    Integer, X As Single, Y As Single)
    If Button = 1 Then Print "You pressed the left button."
    If Button = 2 Then Print "You pressed the right button."
    "
    If Button = 4 Then Print "You pressed the middle button."
End Sub

```

The following program prints a message when a pressed button is released

```

Private Sub Form_MouseUp(Button As Integer, Shift As
    Integer, X As Single, y As Single)
    If Button = 1 Then Print "You released the left button."
    If Button = 2 Then Print "You released the right button."
    "

```

```

    If Button = 4 Then Print "You released the middle
        button. "

```

```
End Sub
```

## Dragging List Items

During the designing phase of Visual Basic application, you often drag controls on the form. The drag-and-drop features in Visual Basic allow you to extend this ability to the user at run time. The action of holding a mouse button down and moving a control is called dragging and the action of releasing the button is called dropping.

## Drag-and-Drop Properties

<i>Property</i>	<i>Description</i>
DragMode	Enables automatic or manual dragging of a control.
DragIcon	Specifies what icon is displayed when the control is dragged.

## Drag-and-Drop Events

<i>Event</i>	<i>Description</i>
DragDrop	Enables automatic or manual dragging of a control.
DragOver	Specifies what icon is displayed when the control is dragged.

## Drag-and-Drop Method

<i>Drag</i>	<i>Description</i>
Drag	Starts or stops manual dragging.

## Dragging and Dropping Operations

By default, Visual Basic has a manual setting for the DragMode property. The Manual setting gives more control as compared to Automatic setting. The Manual setting allows you to specify when a control can and cannot be dragged. In order to drag manually i.e. from the code leave DragMode in its default setting (0-Manual). Then use the Drag method whenever you want to begin or stop dragging an object.

Following are the Visual Basic constants to specify the action of the Drag argument.

<i>Constant</i>	<i>Value</i>	<i>Description</i>
vbCancel	0	Cancel drag operation
vbBeginDrag	1	Begin drag operation
vbEndDrag	2	End drag operation

Syntax:

```
[object.]Drag action
```

### Example

This example performs following drag and drop operation.

NOTES

- MouseDown event - When user starts dragging the source.
- DragDrop event - When the user releases the mouse button after dragging a control, Visual Basic generates a Drag Drop event.

Set the Image1 DragMode property to 0-Manual and then add the procedure shown below:

## NOTES

```
Private Sub Image1-MouseDown(Button As Integer, Shift As
    Integer, X As Single, Y As Single)
    Image1.Drag vbBeginDrag
    Set Image1.DragIcon=LoadPicture("c:\Program
        Files\Microsoft Visual Studio \Common\ Graphics\
        Icons\ Dragdrop\ Dragfldr.ico" )
End Sub
```

```
Private Sub Image2_DragDrop(Source As Control, X As Single,
    Y As Single)
    Source.Visible = False
    Image2.Picture = LoadPicture("c:\Program Files\Microsoft
        Visual Studio\common\Graphics\Icons\Office\
        Files03a.ico")
End Sub
```

The procedure shown in above two programs illustrates how the source and target interact. The source is an Image control with its Picture property set to load a sample icon file representing a few file folders. Its DragMode property has been set to 1 - Automatic and its DragIcon property to a sample drag and drop icon file. The target, also an image control, contains a picture of an open file cabinet.

Dragging and dropping image1 onto image2 causes image1 to vanish and image2 to change its picture to that of a closed file cabinet. Using the source argument, the Visible property of image1 was changed to False.

Adding a third image control to the form demonstrates canceling a drop operation. In this example, the image3 Picture property contains an icon of a trashcan. Using the DragOver event and the source argument, dragging the files over image3 cancels the drag operation.

```
Private Sub Image3_DragOver (Source As Control, X As Single,
    Y As Single, State As Integer)
    Source.Drag vbCancel
End Sub
```

### Connecting Menus to Event Procedures

This topic determines how you can work with menus and event procedures. In this example we see a menu item called Font, under which it has sub - menu items like Bold, Italic, Strikethru and Underline. When user selects bold menu - item the text entered in the textbox change to bold text.

Add the following code in different event procedures:

```
Private Sub mnuBold_click()
    Text1.FontBold = True
```

```
End Sub
Private Sub mnuItalic_click()
Text1.FontItalic = True
End Sub
Private Sub mnuStrikethru_click()
Text1.FontStrikethru = True
End Sub
Private Sub mnuUnderline_click()
Text1.FontUnderline = True
End Sub
```

NOTES

## Programming Controls

Visual Basic has a number of controls. Some of them like labels or list boxes, give users feedback, while others, like command buttons and text boxes, elicit responses. Other controls sit quietly, invisible to the user and perform some of the grunt work that makes your application useful. The timer control is one example of an invisible control. Controls are easy to use and when used properly, can add significant functionality to your programs. To add a control to a form you simply double-click the control you want to add, or you can "draw" the controls on a form by clicking the control and then dragging the mouse around the area on the form where you want the control to be. After you add some controls, you can set most of their properties in the Properties window. You simply click the control to make it active and change the appropriate properties in the Properties window.

The Toolbox is the containing window that holds the custom controls for your applications. In this chapter, we are going to take a look at some of the basic controls you can use to get your applications up and running.

There are also several advanced controls that come with Visual Basic 6 many of which have been enhanced since the previous version. Some controls work with multimedia, such as the Multimedia control and others Winsock and Internet Transfer controls – utilize the Internet. There are also several new data-aware controls that help you work with database. In addition, you can develop your own ActiveX custom controls and add them to your Toolbox or you can distribute your custom controls to other developers through a variety of methods.

Let us learn about these buttons one by one.

### Using Command Buttons

This button is one of the most common one. You can use a command button to elicit simple responses from the user or to invoke special functions on forms. Every time you click the OK button on a dialog box you click a command button. Let us learn more about its properties, methods and events attached to it.

### Properties

Various properties of the command button are given below. More information about these properties can be obtained from the Properties window.

Appearance

BackColor

Cancel

NOTES

Caption	CausesValidation	Container
Default	DisabledPicture	DownPicture
DragIcon	DragMode	Enabled
Font	FontBold	FontItalic
FontName	FontSize	FontStrikethru
FontUnderline	ForeColor	Height
HelpContextID	Hwnd	Index
Index	Left	MaskColor
MouseIcon	MousePointer	Name
OLEDropMode	Parent	Picture
RightToLeft	Style	TabIndex
TabStop	Tag	ToolTipText
Top	UseMaskColor	Value
Visible	WhatsThisHelpID	Width

The two most important properties of command buttons are Name and Caption. The Name property is used to give the control its own identity. This name is used by your code and Visual Basic to distinguish it from the rest of the controls. The Caption property determines the text that appears on the command button. Placing an ampersand character (&) in the caption gives a keyboard-across key alternative (called a hotkey) to a mouse-click. You access these controls by holding the Alt key down while you press the underlined letter of the control you wish to access. The user could also tab to the command button and press the spacebar to simulate a mouse-click on the button. Two other useful properties are Cancel and Default. Setting the Default property to True means the user can simulate a click on the button by pressing Enter. Setting Cancel to True means the user can close a form by pressing Esc.

By setting the Style property, you can make the button contain text only or you can add a picture to the button. If you want the button in its normal, unpressed state to have a picture, you can specify the picture's filename in the Picture property. You can also place other graphics on the button by setting them using the DisabledPicture and DownPicture properties.

Two properties can stop the user from accessing a command button Enabled and Visible. If either is set to False, the command button will be unusable. Disabling a command button is a handy technique if you want to force the end user to complete certain actions (such as filling in text boxes) before clicking the next button in the process.

If the user moves around the form with the Tab key, you can determine the order in which they visit controls by specifying the TabIndex property. A control with a TabIndex of 0 (zero) is the first to receive the focus on a form, provided it's not disabled or invisible. If you alter the TabIndex of one control, the other controls' order adjust to accommodate the new order. When you want to prevent a user from tabbing to a control, set its TabStop property to False. This does not prevent a mouse-click on a control - to stop that, use the Enabled or Visible properties described previously.

## Events

The most frequently coded event procedure for a command button corresponds to the Click() event; the other events for a command button are listed here:

Click	DragDrop	DragOver
GotFocus	KeyDown	KeyPress
KeyUp	LostFocus	MouseDown
MouseMove	MouseUp	OLECompleteDrag
OLEDragDrop	OLEDragOver	OLEGiveFeedback
OLESetData	OLEStartDrag	

NOTES

In your first programs, the Click() event is probably the only event in which you'd be interested. It's the most commonly used event on a command button. You won't use many of the other events until you become more proficient in Visual Basic. However, you can also use the MouseUp() even in place of the Click() event. Many Windows 95/98 applications use this event because it gives the user a chance to back out without firing a Click() event.

## Methods

Listed Below are the methods for the command button. The most commonly used method is Set Focus.

Drag	Move	OLEDrag
Refresh	SetFocus	ShowWhatThis
ZOrder		

The SetFocus method is sometimes used to place the focus on a particular button. This comes in handy if you want the user to return to a default button after editing a text box on a form. If that were so, the code for the focus button looks like this:

```
cmdMyButton.SetFocus
```

And it might be placed in the Change() event procedure for a text box.

---

## TEXT BOXES

---

Nearly every Visual Basic project involves at least one text box control. Text boxes are commonly used for accepting user input or for entering data. Their properties are, or course specifically designed for these purposes. If you only want the simplest of user responses, you might consider using an InputBox instead. The InputBox displays a dialog box and prompts the user to enter something and returns this to the application.

## Properties

Here is the list of properties for the text box control.

Alignment	Font	LinkItem	RightToLeft
Appearance	FontBold	LinkMode	ScrollBars
BackColor	FontItalic	LinkTimeout	SellLength

## NOTES

BorderStyle	FontName	LinkTopic	SelStart
CausesValidation	FontSize	Locked	SelText
Container	FontStrikethru	MaxLength	TabIndex
DataChanged	FontUnderline	MouseIcon	TabStop
DataField	ForeColor	MousePointer	Tag
DataFormat	Height	MultiLine	Text
DataMember	HelpContextID	Name	ToolTipText
DataSource	HideSelection	OLEDragMode	Top
DragIcon	HWND	OLEDropMode	Visible
DragMode	Index	Parent	
WhatsThisHelpID			
Enabled	Left	PasswordChar	Width

As always, the property you set first is the Name property. By convention this begins with the text prefix. Notice that there is no Caption for a text box. Instead the text shown in the text box is determined by the Text property. You can provide a default entry in the text box by setting the Text property accordingly. It's possible you don't want any value in the text box – you want the user to enter something from scratch blank. The MaxLength property is handy for limiting the user to a specified number of characters. This is often used in conjunction with the PasswordChar property. The letter is valuable for showing a default character (the asterisk character \*- is the best choice) when the user is entering a password. MaxLength and PasswordChar properties are often employed for a text box on a logon form.

The MultiLine property lets the user type more than one line of text into the text box. If MultiLine is used with the ScrollBars property, you can make a simple text editor with no coding-though you would need a couple of code to save the user's typing.

The SetLength, SelStart and SelText properties are useful for dealing with text appropriately. For example, the SelText property returns the text in the text box that the user selected with the mouse or arrow keys. From there it's easy to copy or cut the selected text to the Clipboard.

Note that the ReadOnly property from previous versions has been replaced by the Locked property. Setting the Locked property to True will cause the text box to display data, but will permit no editing. You may have noticed this type of text box on license agreement dialog boxes that appear during program installations. You can select and copy text, but you cannot type or delete text from the box.

To change the order in which the user tabs around the text boxes (and other controls) on a form, change the TabIndex setting. If you don't want the user to tab into a text box, set its TabStop property to False. To prevent a user from clicking in the text box with the mouse, set the Enabled property to False. There may be some situations where you would want to prevent the user from accessing a text box. For example, the user is not allowed to enter a message in an e-mail program until an address has been entered in the address text box. You will discover other examples of why you would want to use this feature as you become more proficient with Visual Basic.

## Events

The text box control supports a few events that are listed in the table here:

Change	KeyDown	LinkOpen	OLEDragDrop
Click	KeyPress	LostFocus	OLEDragOver
DbtClick	KeyUp	MouseDown	
OLEGiveFeedback			
DragDrop	LinkClose	MouseMove	OLESetData
DragOver	LinkError	MouseMove	OLEStartDrag
GotFocus	LinkNotify	OLECompleteDrag	Validate

NOTES

The Change() event occurs every time the user inserts, replaces or deletes a character in a text box. You can perform some elementary validation of user entry in the Change() event. You can even use it to restrict entry to certain characters only. However, you may find that the Masked Edit control or one of the Key events is more suitable for this purpose. The Microsoft Masked Edit control lets you specify entry templates or masks. It's a custom control that you need to add to the Toolbox if you want to use it. There's a full reference for all the custom control bundled with Visual Basic in the Microsoft Developer Network, included on your Visual Basic CD.

## Methods

Here is the list of methods supported by the text box control:

Drag	LinkRequest	OLEDrag	
ShowWhatsThis			
LinkExecutive	LinkSend	Refresh	ZOrder
LinkPoke	Move	SetFocus	

Most of the methods here are not used very frequently, though the Link methods are necessary if your text box is involved in a DDE (Dynamic Data Exchange) conversation. DDE allows one application to communicate with another. As the user interacts with one application, it sends data automatically to the other application. Unfortunately, it is beyond the scope of this book to cover DDE in detail. If you are interested in pursuing it further, check the online help.

The SetFocus method, though is a boon in data-entry applications. When the user clicks a command button (say, an Update button) the focus remains on that button. If the last statement in the Click() event for the command button is a SetFocus method, you can force the focus back to the data-entry text boxes. This saves the user from an extra mouse-click or excessive use of the Tab key just to get back into position. The syntax is:

```
txtMyTextBox.SetFocus
```

---

## LABELS

---

A label control is similar to a text box control is that both display text. The main difference however is that a label displays read-only text as far as the user is concerned though you can alter the caption as a run-time property.

## NOTES

The property of interest in a label control is the Caption property, as opposed to a text box's Text property. Labels are often used for providing information to the user. This can be in the form of a message shown on the form itself or a prompt. The prompt is usually combined with a text box, list box, or other control. It gives the user an idea or the nature of the referenced control. For example, if you had a text box that allowed the user to enter the data for a customer's name, then you might place a label to the left or above the text box with its Caption set to Customer Name.

### Properties

You have already worked some of the properties of a label control. Here is the complete list of properties for this control:

Alignment	DataSource	Height	Parent
Appearance	DragIcon	Index	RightToLeft
AutoSize	DragMode	Left	TabIndex
BackColor	Enabled	LinkItem	Tag
BackStyle	Font	LinkMode	ToolTipText
BorderStyle	FontBold	LinkNotify	Top
Caption	FontItalic	LinkTimeout	UseMnemonic
Container	FontName	LinkTopic	Visible
DataChanged	FontSize	MouseIcon	WhatsThisHelpID
DataField	FontStrikethru	MousePointer	Width
DataFormat	FontUnderline	Name	WorldWrap
DataMember	ForeColor	OLEDropMode	

As a remainder, the most important property at the outset is – once again the Name property. For labels the prefix is normally lbl. The Caption property determines the text shown in the label. If you incorporate an ampersand (&) character in the caption, an access key is defined. This raises an interesting question: What happens if you want to show an actual ampersand in the caption? An ampersand does not display; instead, it remains hidden and causes the subsequent character to appear underlined.

You define the size of the label at design time. At run time you might wish to alter the Caption property, only to find it's too big to fit within the label control. You could calculate the length of the caption and adjust the label size accordingly, but this is messy and there's danger of an enlarged label obscuring other controls. To simplify matters, use the AutoSize and WordWrap properties, either by themselves or in conjunction. That way the caption will fit and you can control whether the label expands vertically rather than horizontally.

One more interesting label control property is BorderStyle. This is not related to the form property of the same name – there are only two choices. But by setting BorderStyle to 1 – Fixed Single and the BackColor to white (or whatever), the label looks exactly like a text box, except that it's read-only. Labels were often used in this fashion in prior versions of Visual Basic to show data for browsing purposes only.

## Events

The label control has many of the same events as any other controls:

Change	LinkClose	MouseMove	OLEGiveFeedback
Click	LinkError	MouseUp	OLESetData
DbClick	LinkNotify	OLECompleteDrag	OLEStartDrag
DragDrop	LinkOpen	OLEDragDrop	
DragOver	MouseDown	OLEDragOver	

NOTES

Most of the standard events are supported. But note the absence of any Key() events. This is consistent with a label not being able to receive the focus. The Mouse() events are there, because there's nothing to stop you from clicking a label at run time. The ability to click a control does not indicate it must necessarily receive the focus. The Link() events are not shared by many other controls (with the exception of text boxes and picture controls). These events are concerned with DDE (Dynamic Data Exchange) conversations.

## Methods

The label control also has methods, but you will probably not use them very often in your applications. The following shows the methods supported by the label control:

Drag	LinkRequest	OLEDrag	Zorder
LinkExecute	LinkSend	Refresh	
LinkPoke	Move	ShowWhatThis	

The label methods are not particularly useful, although the LinkRequest method is sometimes used to update a nonautomatic DDE link.

## Using Option Buttons

Option button controls, also called radio buttons are used to allow the user to select one and only one, option from a group of options. Usually option buttons are grouped together within a frame control but they can also be grouped on a plain form, if there is to be only one group of option buttons. Thus, if you had a frame specifying a delivery method, you might have one button for UPS (United Parcel Service) and another four Courier delivery. Products can only be shipped by one of these methods (not both-and not more). In contrast, option buttons representing, say, bold and italic settings for text would not make sense. Text can be both bold and italic or neither (none).

## Properties

The option button supports many properties, are shown in the table below:

Alignment	FontSize	Picture
Appearance	FontStrikethru	RightToLeft
BackColor	FontColor	TabIndex
CausesValidation	Height	TabStop
Container	HelpContextID	Tag

NOTES

DisabledPicture	HWnd	ToolTipText
DownPicture	Index	Top
DragIcon	Left	UseMaskColor
DragMode	MaskColor	Value
Enabled	MouseIcon	Visible
Font	MousePointer	WhatsThisHelpID
FontBold	Name	Width
FontItalic	OLEDropMode	
FontName	Parent	

Once again the Name property is the one to set first; option buttons have an opt prefix by convention. The Caption property helps the user determine the purpose of an action button. The other popular property is Value. This is invaluable at both design time and run time. At run time you test the Value property to see if the user has turned on (or off) the option button. The property has two settings, true and False. At design time you can set the Value property to True for one of the buttons if you wish – the default setting is False. This means that the option button (and only that option in a group) is pre-selected when the form opens. If you try to make Value for another button in the group True, then the previous the previous one reverts to a False setting. A new property added in version 6 is the Style property. The default setting of 0 – Standard will draw a normal option button, as shown at the beginning of this section. However, by setting this property to 1 – Graphical, you can make the option button look just like a command button, but it allows only one selection to be made within a group. It works much like the old memory preset buttons on the old stock car radios.

**Events**

The option button control has a few events, but only the Click() event is really used:

Click	KeyDown	MouseMove	OLEGiveFeedback
DbtClick	KeyPress	MouseUp	OLESetData
DragDrop	KeyUp	OLECompleteDrag	OLEStartDrag
DragOver	LostFocus	OLEDragDrop	Validate
GotFocus	MouseDown	OLEDragOver	

The typical way of dealing with option buttons is to test the Value property at run time to see if they're selected. Your code then initiates actions accordingly. It's common to test for the Value property in the Click() event procedure for a command button that's clicked after the user has selected the option button of interest. This allows you to check for a condition before the next procedure is called. You test the Value property in an If ...End If or Select Case...End Select construct. But there may be occasions when you want to initiate an action immediately after the user makes a choice. Then you may want to trap the option button's Click() event.

**Methods**

The methods for the option button are of little use in the Visual Basic environment:

Drag                      OLEDrag                      SetFocus                      Zorder  
 Move                      Refresh                      ShowWhatsThis

Open and start the Controls sample application. Click the Option Buttons button to bring up the Options dialog box. The five option buttons are actually in two groups. The option labeled 486, Pentium and Pentium Pro are in their own group directly on the form. The Windows 95/98 and Windows NT options are in a separate group on the Operating System frame. This frame then rests on the form and separates the two groups of option buttons. If you click an option button, you will notice the other option buttons in the same group become deselected. Only one option can be selected in any particular group at one time.

NOTES

**Using Check Boxes**

A check box control is rather similar to an option button, which was described in the last section. Both often partake in group and the Value property is tested to see if a check box is on or off. But there are two fundamental differences between check boxes and option buttons: Check boxes are valid as single controls – a single option button is probably counter-intuitive. Check boxes (even when in a group) are not mutually exclusive. Finally, check boxes have three possible settings for the Value property.

An option button is either on or it's off. Therefore, Value can be either True or False. Check boxes can be in one of three states on, or grayed. Grayed check box is neither on nor off, though the user can change its setting. If the check box were disabled the user wouldn't be able to turn it on or off. A grayed check box are selected. If you have installed Windows 95/98, then you understand what are grayed check box means.

**Properties**

The following table lists the properties for the check box control:

Alignment	DownPicture	Height	RightToLeft
Appearance	DragIcon	HelpContextID	Style
BackColor	DragMode	hWnd	TabIndex
Caption	Enabled	Index	TabStop
CausesValidation	Font	Left	Tag
Container	FontBold	MaskColor	ToolTipText
DataChanged	FontItalic	MouseIcon	Top
DataField	FontSize	Name	Value
DataMember	FontStrikethru	OLEDropMode	Visible
DataSource	FontUnderline	Parent	WhatsThisHelpID
DisabledPicture	ForeColor	Picture	Width

Again, as with option buttons, the three most popular properties are Name, Caption and Value. When setting the Name property it's conventional to use a chk prefix.

## Events

The following table shows you that the check box has similar events to the option button control:

NOTES

Click	KeyPress	MouseUp	OLESetData
DragDrop	KeyUp	OLECompleteDrag	OLEStartDrag
DragOver	LostFocus	OLEDragDrop	Validate
GotFocus	MouseDown	OLEDragOver	
KeyDown	MouseMove	OLEGiveFeedback	

To carry out further processing as soon as the user has clicked the check box, use the Click() event. Normally, though, if you do not put code directly within the click() event, you will have another procedure that will ascertain the value of a check box by retrieving its Value property.

## Methods

Like the events of the check box control, the methods are similar to those for the option button control:

Drag	OLEDrag	SetFocus	ZOrder
Move	Refresh	ShowWhatsThis	

## Using Frame Controls

When used by itself, the frame control is not particularly useful. The controls normally placed in a frame are option buttons and check boxes. This has the effect of grouping them together so that when the frame is moved, the other controls move too. For this to work you can't double-click a control (say, an option button) to add it to the form and then drag it into position within the frame. Instead, you must single-click the control in Toolbox and drag a location for it inside the frame. Then all the controls move together. In addition, the option buttons function as a group - that is, if you select one at run time, the others become deselected. If you simply scatter option buttons randomly on a form, then they all function as one large group. To create separate groupings of options buttons, you place them in frames. The button items within each frame act as a self-contained group and have no effect on the option buttons in other frame groups.

Although a frame is often used as a container for check box groups too, each check box is completely independent. Thus the setting for one check box has no effect on the setting for the others in the same group. This is the behavior you would expect of check boxes. Check boxes are not mutually exclusive. This contrasts with option buttons, where the buttons within a single group should be mutually exclusive. The reasons then for placing check boxes in a frame is to enable you to move the group as a whole, when you reposition the frame at design time. The frame also serves as a visual grouping for the check boxes. For example, the check boxes relating to a particular feature can be in one frame and those pertinent to another feature in another frame.

A frame is usually given the prefix fra. You place the frame on the form before you place the controls it's going to contain.

## Properties

The frame control has several properties, listed below:

Appearance	Enabled	Height	Parent
BackColor	Font	HelpContextID	RightToLeft
BorderStyle	FontBold	hWnd	TabIndex
Caption	FontItalic	Index	Tag
Container	FontName	Left	ToolTipNext
ClipControls	FontSize	MouseIcon	Top
Container	FontStrikethru	MousePointer	Visible
DragIcon	FontUnderline	Name	WhatsThisHelpID
DragMode	FreeColor	OLEDropMode	Width

NOTES

After the Name property, perhaps the single most important property is Caption. You use this to give a meaningful title to the frame on the form. Then it's clear to the end user which feature the option buttons (or check boxes) in the frame refer to. To provide a clue as to how each option button affects the features, you use the Caption property of the buttons. For example, in an order dispatch system you might have a frame with the caption Delivery. And within that frame you might have two option buttons, with the captions Normal and Express.

## Events

The frame control supports a few events:

Click	MouseMove	OLEGiveFeedBack
Db1Click	MouseUp	OLESetData
DragDrop	OLECompleteDrag	OLEStartDrag
DragOver	OLEDragDrop	
MouseDown	OLEDragOver	

The frame control events are only rarely used. In an application that use drag-and-drop, however, the DragDrop() event is sometimes used to initiate actions when the user drops an object into a frame area.

## Methods

A frame object supports only a few methods. None are very helpful and they're hardly ever seen in Visual Basic projects:

Drag	OLEDrag	ShowWhatsThis
Move	Refresh	Zorder

## Using List Boxes

If you're a regular user of Windows, then you're familiar with list box controls. A list box is an ideal way of representing users with a list of data. Users can browse the data in the list box or select one or more items as the basis for further processing. The

user can't edit the data in a list box directly – one way around this is to use a combo box instead; combo boxes are discussed next. When the list of data is too long for the list box, Visual Basic will add a vertical scrollbar. Let's examine most of the important list box control properties, events and methods.

NOTES

**Properties**

Many of the list box properties are shared by a combo box control and some of them are essential for getting the best from the control:

Appearance	FontBold	List	Style
BackColor	FontItalic	ListCount	TabIndex
CausesValidation	FontName	ListIndex	TabStop
Columns	FontSize	MouseIcon	Tag
Container	FontStrikethru	MousePointer	Text
DataChanged	FontUnderline	MultiSelect	ToolTipText
DataField	ForeColor	Name	Top
DataFormat	Height	NewIndex	ToolIndex
DataMember	HelpContextID	OLEDragMode	Visible
DataSource	hWnd	Parent	WhatsThisHelpID
DragIcon	Index	RightToLeft	Width
DragMode	IntegralHeight	Selcount	
Enabled	ItemData	Selected	
Font	Left	Sorted	

The Columns property lets you create a multicolumn list box. Unfortunately, the columns are of the snaking or newspaper, type. There's no direct support for the multiple columns of an Access-style list box where different data items are displayed in separate columns. Instead Visual Basic wraps the same type of data items from column to column.

The List property sets or returns the value of an item in the list. You use the index of the item with the List property. The index position of items in a list box start at 0 (zero) and run to 1 less than the total number of items in a list. Thus, if you had 10 items in the list box, the index positions run from 0 to 9.

You use the List property to get the value of any item in the list. For example, to return the value of the third item in the list, use the following listList1.List(2)

To get the value of the currently selected item in the list, simply use the Text property of the list box. The ListIndex property sets or returns the index position of the currently selected item – if no item is selected, the ListIndex property is -1.

You can pick up the index position of the last item added to a list with the NewIndex property. The ListCount property returns the number of items in a list box. Confusingly, this is always 1 greater than the value returned by the NewIndex property – this is because the index positions count from 0 (zero) while the ListCount property starts counting from 1 for the first item. ListCount returns 0 if the list box is empty.

The MultiSelect property determines whether the user can select one item or whether they can select more than one. List boxes support both a simple and an extended multiple selection. A simple multiple selection allows the user to select contiguous items – usually accomplished with the mouse and the Shift key. An extended multiple selection lets the user select contiguous and noncontiguous items – usually done by mouse clicks in conjunction with the Ctrl and/or Shift keys.

The Selected property is a Boolean property and is a run-time property only. A Boolean property is one that can take only a True or False setting. The following line preselects the third item in a list box:

```
lstList1.Selected(2) = True
```

note the use of index position (2) to reference the third item in the list.

The final property that we're going to consider is the Sorted property. This is one of those properties that you can set only at design time. You can read it or return it, at run time (that is, see if it's true or False) but you can't set it (that is, change an unsorted list into a sorted one or vice versa). When you set the Sorted property to True at design time, any items you add to a list box (typically, with the AddItem method) are sorted in alphabetical order. The sort can only be in ascending order and it's not case-sensitive.

## Events

The list box control supports a few events, shown here:

Click	KeyDown	MouseUp	OLEStartDrag
DbtClick	KeyPress	OLECompleteDrag	Scroll
DragDrop	KeyUp	OLEDragDrop	Validate
DragOver	LostFocus	OLEDragOver	
GotFocus	MouseDown	OLEGiveFeedback	
ItemCheck	MouseMove	OLESetData	

Perhaps the most commonly used event for a list box is DbtClick(). This coincides with the normal operation of list boxes in Windows applications. The first thing to do with a list box is usually to fill it with items for the list. The AddItem method can be used to do this. You can then, if you want, preselect one of the items in the list by setting the Selected property to True. The user either accepts the default selection or chooses another item with a single-click. Then clicking an OK command button carries out a process using the Text property which returns the value of the selected item. However, a popular shortcut is to double-click the item in the list – that way, the user can both select an item and initiate a process based on that item in a single action. Many Windows applications adopt this technique to copy one item from a list box into another list box.

## Methods

The list box has many of its own methods, as well as some common to the other controls discussed so far:

AddItem	Move	SetFocus
Clear	Refresh	ShowWhatsThis

## NOTES

There are three methods here worthy of note – AddItem, Clear and RemoveItem. AddItem, as already indicated is for adding items to a list box control. The RemoveItem method, as you might expect, removes items from a list box. To remove all the items in one fell swoop, use the Clear method.

An example of this simplest syntax for the AddItem method is:

```
LstList1.AddItem "Hello"
```

This adds the word "Hello" to the list box. Often you employ a number of AddItem methods, one after the other, to populate (fill in) a list box. Many developers place the AddItem methods in a Form\_Load() event procedure so the list box is filling as the form loads. You can specify the position that an item will take in a list by specifying the index position:

```
lstList1.AddItem "hello", 3
```

This places the text "Hello" at the fourth position in the list. If you omit the index position, the item is added to the end of the list-or if the Sorted property is set to True, the item is placed in the correct sorted order.

### Using Combo Boxes

The name combo box comes from "combination box". The idea is that a combo box combines the features of both a text box and a list box. A potential problem with list boxes – in some situations anyway – is that you're stuck with the entries displayed. You can't directly edit an item in the list or select an entry that's not already there. Of course, if you want to restrict the user, than a list box is fine in this respect. A combo box control (at least in two of its styles available in Visual Basic) allows you to select a predefined item from a list or to enter a new item not in the list. A combo box can also incorporate a drop-down section – that means it takes less room on a form than a normal list box. In all, there are three types of combo boxes to choose from at design time: a drop-down combo, a simple combo and a drop-down list. You can specify the type by setting the Style property.

Apart from the Style property, the properties, events and methods of combo boxes are very similar to those of list boxes, described shortly. However, the Text property is different. With a list box, the Text property returns the text of the currently selected item at run time. With a combo box, on the other hand, you can assign a value to the Text property at run time – in effect, you can set the text, even if the item is not already in the list.

### Properties

These are the properties for the combo box control:

Appearance	FontItalic	ListCount	Style
BackColor	FontName	ListIndex	TabIndex
CausesValidation	FontSize	Locked	TabStop
Container	FontStrikethru	MouseIcon	Tag
DataChanged	FontUnderline	MousePointer	Text
DataField	ForeColor	Name	ToolTipText

DataFormat	Height	NewIndex	Top
DataMember	HelpContextID	OLEDragMode	TopIndex
DataSource	hWnd	Parent	Visible
DragIcon	Index	RightToLeft	WhatsThisHelpID
DragMode	IntegralHeight	SelLength	Width
Enabled	ItemData	SelStart	
Font	Left	SelText	
FontBold	List	Sorted	

NOTES

The List, ListCount, ListIndex, NewIndex and Sorted properties are identical to those for a list box. The property that's different and that consists of one of the fundamentals of designing combo boxes, is the Style property. There are three settings for Style, and the behavior and appearance of the combo box are determined by the setting you choose.

The styles are numbered from 0 to 2 and represent, in order, a drop-down combo, a simple combo and drop-down list.

- The drop-down combo looks like a standard text box with a drop-down arrow to the right. Clicking the arrow opens a list beneath the text box. The user has the choice of selecting an item from the list, which places it in the text box or of entering their own text in the text box. In other words, this is the true combo box.
- The simple combo is a variation of this theme – the only difference being that the list is permanently displayed. This one's an option if your form is not too crowded.
- The final style, the drop-down list is really a type of list box rather than a combo box. It looks identical to a drop-down combo but, as the name suggests, the user is confined to selecting a predefined item from the list. The advantage of this latter style is that it takes up less space than a conventional list box.

## Events

Here are the events of the combo box:

Change	DropDown	LostFocus	OLESetData
Click	GotFocus	OLECompleteDrag	OLEStartDrag
DbClick	KeyDown	OLEDragDrop	Scroll
DragDrop	KeyPress	OLEDragOver	Validate
DragOver	KeyUp	OLEGiveFeedback	

Most of the events for a combo box control are the standard ones. However, the DropDown() event is specific to combo boxes – though not supported by the simple combo box, which is already “dropped-down.”

The Change() event is not supported by the drop-down list because the user can't change the entry in the text box section. To see if that type of combo box has been accessed by the user, try the Click() or the DropDown() event procedures.

## NOTES

The DblClick() event is only relevant to the simple combo box for it's the only one where the user can see the full list by default. Usually the DblClick() event procedure calls the Click() event for a command button. This means the user can simply double-click an item in the list, rather than using a single-click to select followed by a click on a command button to carry out some processing based on the user selection.

### Methods

The methods you can apply to a combo box control are the same as those for a list box:

AddItem	Move	RemoveItem	Zorder
Clear	OLEDrag	SetFocus	
Drag	Refresh	ShowWhatsThis	

Again, the important methods are AddItem, Clear and RemoveItem. And, just as with a list box control, it's common practice to populate a combo box with a series of AddItem methods in the Load event of a form. Incidentally, especially if you've worked with the database application Microsoft Access, you may be wondering about the flexibility of list and combo boxes. What do you do if the predetermined items in the list continually change? Do you continually have to re-code EDE you generate from your project? Another concern of yours could be regarding the actual tedium of typing long series of AddItem methods.

All of these questions are easily resolved if you exploit the RowSource and ListField properties of a data-bound list or data-bound combo box. Even more flexibility is provided by these data-bound versions of those two controls (DBList and DBCombo). Often you want the user first to select an item from a list box and then to click a command button. The button initiates an action using the selected item. An accepted alternative is for the user to simply double-click the item in the list. This both selects the item and carries out the action.

---

## IMAGE CONTROLS

---

The image control (its prefix is often img) is a lightweight equivalent of the picture box control, which is described in a later section. But unlike the picture control, the image control can't act as a container for other objects. In some of its other properties it's not as versatile, but it's a good choice if you simply want to display a picture on a form. Image controls consume far less memory than picture controls. The image control that comes with Visual Basic can now display bitmap (.BMP), icon (.ICO), metafile (.WMF), JPEG (.JPG) and GIF (.GIF) files. This makes it easier to display graphics from the World Wide Web as well as graphics from other popular graphics programs.

### Properties

The image control utilizes several properties, but fewer than the picture box, discussed later.

Appearance	DragIcon	MousePointer	Tag
BorderStyle	DragMode	Name	Top

Container	Enabled	OLEDragMode	Visible
DataField	Height	OLEDropMode	WhatsThis HelpID
DataFormat	Index	Parent	Width
DataMember	Left	Picture	
Datasource	MouseIcon	Stretch	

NOTES

As with most other graphics controls, you add the graphic by setting the Picture property. Perhaps the most interesting property here is Stretch. This is a Boolean property – meaning it takes only the values True or False. When Stretch is set to False (the default), the control resizes to the size of the picture placed inside it. If you later resize the control, then the loaded picture either is cropped or has empty space showing around it or both, depending on the relative directions of horizontal and vertical resizing. But if you set Stretch to true, the picture resizes with the control. Thus you can make the enclosed picture larger or smaller or fatter or thinner, by resizing the control after the picture is loaded. A picture control has no Stretch property. Its nearest equivalent is the AutoSize property. When AutoSize is set to True for a picture box, then the control adapts to the size of the loaded picture. However, unlike an image control with Stretch turned on, if the picture box is resized the enclosed picture remains the same – the picture does not “stretch” with the picture box control.

**Events**

The image control doesn't use many events and of those, you may only use a few, if any.

Click	MouseDown	OLEDragDrop	OLEStartDrag
DbClick	MouseMove	OLEDragOver	
DragDrop	MouseUp	OLEGiveFeedBack	
DragOver	OLECompleteDrag	OLESetData	

Image controls are sometimes handy as drag-and-drop destinations. This is because you can have a picture inside that gives an indication of the results of dropping onto the control.

**Image Methods**

The following are properties of an image control:

Drag	OLEDrag	ShowWhatsThis
Move	Refresh	ZOrder

You will most likely not use any of these methods in your applications.

**Picture Boxes**

As you might expect, picture boxes often display graphics (for example, bitmaps, icons, JPEGs and GIFs). In this role, picture boxes are similar to image controls. However, picture boxes and images have slightly different properties and therefore behave differently. If you just want to show a picture, then an image control is usually a better choice than a picture box. An image control takes up less memory and is a lightweight version of the picture box control. However, if you want to move the

graphic around the form, a picture box produces a smoother display. In addition, you can create text and use graphics methods in a picture box at run time. The graphics methods enable you to draw lines, circles and rectangles at run time. But, most importantly for this application, picture boxes can act as containers for other controls. Thus, you can place a command button within a picture box. In this respect, picture boxes function as "forms within forms".

## NOTES

### Properties

The table below lists the properties for the picture box control. Notice that there are many more properties for this control than there are for the image control.

Align	FillStyle	MousePointer
Appearance	Font	Name
AutoRedraw	FontBold	OLEDragMode
AutoSize	FontItalic	OLEDropMode
BackColor	FontName	Parent
BorderStyle	FontSize	Picture
CausesValidation	FontStrikethru	RightToLeft
ClipControls	FontTransparent	ScaleHeight
Container	FontUnderline	ScaleLeft
CurrentX	ForeColor	ScaleMode
CurrentY	HasDC	ScaleTop
DataChanged	hDC	ScaleWidth
DataField	Height	TabIndex
DataFormat	HelpContextID	TabStop
DataMember	hWnd	Tag
DataSource	Image	ToolTipText
DragIcon	Index	Top
DragMode	Left	Visible
DrawMode	LinkItem	WhatsThisHelpID
DrawStyle	LinkMode	Width
DrawWidth	LinkTimeout	
Enabled	LinkTopic	
FillColor	MouseIcon	

Quite a lot of properties this time! When you put a picture into a picture box, it appears as its normal size. If it's too big for the picture box, the graphic is clipped. Setting the AutoSize property to True causes the picture box to resize to match the size of the graphic. The graphic displayed in the picture box is determined by the Picture property you can change this property at both design time and run time. There's a similar-sounding property – the Image property. This one's only available at

run time and it's used to make a copy from one picture box to another. The syntax for doing this is:

```
Picture2.Picture = Picture1.Image
```

You can place this line of code in any event where it is relevant. For example, you may want to change the picture in a picture box when the user selects a different record in a database.

The line above places a copy (image) of the picture in the first picture box into the second picture box (using its Picture property). You can even change the picture directly at run time. The syntax is:

```
Picture1.Picture = LoadPicture ("filename")
```

To empty a picture box, use the Visual Basic LoadPicture function with no parameter.

```
Picture1.Picture = LoadPicture()
```

## Events

The picture box events are listed in the following table:

Change	KeyPress	MouseDown	OLESetData
Click	KeyUp	MouseMove	OLEStartDrag
DbClick	LinkClose	MouseUp	Paint
DragDrop	LinkError	OLECompleteDrag	Resize
DragOver	LinkNotify	OLEDragDrop	Validate
GotFocus	LinkOpen	OLEDragOver	
KeyDown	LostFocus	OLEGiveFeedback	

Two of the popular events for picture boxes are the Click() and DragDrop() events.

## Methods

The picture box controls supports more methods than its counterpart, the image box. The most important ones are listed in boldface in the following table:

Circle	LinkRequest	PSet	TextHeight
Cls	LinkSend	Refresh	TextWidth
Drag	Move	ScaleX	Zorder
Line	OLEDrag	ScaleY	
LinkExecute	PaintPicture	SetFocus	
LinkPoke	Point	ShowWhatsThis	

The Circle, Cls, Line, PaintPicture and PSet methods are all used when drawing graphics or text in the picture at run time – Cls (like the old DOD command for “clear screen”) is actually used to erase entries. The Zorder method is the run-time equivalent of Format – Order – Bring to Front or Format – Order – Send to Back. You can use Zorder to determine which controls overlap other controls. However, you should be aware that there are three layers on a form – Zorder only works within the layer that contains the control. All the nongraphical controls except labels (for example, command buttons) belong to the top layer. Picture boxes and graphical controls (as

NOTES

## NOTES

well as labels) belong to the middle layer. The bottom layer contains the result of the graphics methods – for instance, a circle drawn with the Circle method is on the bottom layer. This contrasts with a circle drawn with the Shape control, which is in the middle layer. What all this means is that you can't position a picture box over a command buttons with ZOrder – the picture box over a command button with ZOrder – the picture box is permanently relegated to the layer behind. The ZOrder method is for rearranging objects within one layer.

### Timers

The timer control is one of the few controls always hidden at run time. This means you don't have to find room for it on a form – it can go anywhere, even on top of existing controls. The timer basically does just one thing: It checks the system clock and acts accordingly.

### Properties

The timer control does not have many properties, as you can see in this table:

Enabled	Left	Tag
Index	Name	Top
Interval	Parent	

Apart from the Name Property (a tmr prefix is recommended), there are only two important properties for the time control – the Enabled property and the Interval property. Indeed, you have to set these properties to get the timer to do anything at all (assuming the Enabled property is at its default, true). The Left and Top properties are virtually superfluous – it makes little difference where you put a timer on a form.

The Interval property is measured in milliseconds. This means if you want to count seconds, you have to multiply the number of seconds by 1,000. Once an interval has elapsed (provided the timer is enabled), the timer generates its own Timer event. It does this by checking the system clock at frequent intervals.

### Event

The timer control has only one event called, appropriately, a Timer() event. As already stated, this event takes place every time an interval elapses. The interval is determined by the Interval property. To stop the Timer() event from occurring, you can set the timer's Enabled property to False at run time.

### Methods

The timer control does not support any methods at all.

---

## SUMMARY

---

### NOTES

1. You can start Visual Basic as you would start any of the programs in the Windows environment, i.e., from Start\Programs\Visual Studio\Visual Basic.
2. Various items on the toolbar are: Add Project, Add Form, Menu Editor, Open Project, Save Project, Cut, Copy, Paste, Find, Undo, Redo, Start (Run), Break, End (Stop), Break, End (Stop), Open Project Explorer, Open Properties Window, Form Layout Window, Object Browser, View Toolbox and Data View Window.
3. File menu has: New Project, Open Project, Add Project, Remove Project, Save Project/Save Project Group, Save Project As/Save Project Group As, Print, Print Setup, Make <Project>, Make Project Group, File 1, 2, 3, 4 and Exit.
4. Edit menu has: Undo, Redo, Cut, Copy, Paste, Paste Link, Delete, Delete Table from Database, Remove Table from Diagram, Select All, Select All Columns, Table, Find, Find Next, Replace, Indent, Outdent, Insert File, List Properties/Methods, List Constants, Quick Info, Parameter Info, Complete Word, Go To Row and Bookmarks.
5. View menu has : Code, Object, Definition, Last Position, Object Browser, Immediate Window, Locals Window, Watch Window, Call Stack, Project Explorer, Properties Window, Form Layout Window, Property Pages, Table, Zoom, Show Panes, Toolbox, Color Palette and Toolbars.
6. Project menu has: Add Form, Add MDI Form, Add Module, Add Class Module, Add User Control, Add Property Page, Add User Document, Add AddIn Class/DHTML Page/Data Environment/Data Report/Web Class/Microsoft UserConnection/ActiveX Designers, Add File, Remove <Item>, References, Components and <Project Name> Properties.
7. Format menu has the following options: Align, Make Same Size, Size to Grid, Horizontal Spacing, Vertical Spacing, Center in Form, Order and Lock Controls.
8. Debug menu has the following commands: Step Into, Step Over, Step Out, Run To Cursor, Add Watch, Edit Watch, Quick Watch, Toggle Breakpoint, Clear All Breakpoints, Set Next Statement and Show Next Statement.
9. Run menu has the following commands: Start, Start With Full Compile, Break, End and Restart.
10. Query menu has the following commands: Run, Clear Results, Verify SQL Syntax, Group By, Change Type, Add To Output, Sort Ascending, Sort Decending, Remove Filter, Select All Rows From <Table A> and Select All Rows From <Table B>.
11. Diagram Menu has the following commands: New Text Annotation, Set Text Font, Add Related Tables, Show Relationship Labels, Modify Custom View, View Page Breaks, Recalculate Page Breaks, Arrange Selection, Arrange Tables and Autosize Selected Tables.
12. Tools menu has the following commands: Add Procedure, Procedure Attributes, Menu Editor and Options.
13. Add Ins menu has the following commands: Visual Data Manager and Add-In Manager.
14. Window menu has the following commands: Tile Horizontally, Tile Vertically, Cascade and Arrange Icons.
15. Help menu has the following commands: Contents, Index, Search, Technical Support, Microsoft on the Web and About Microsoft Visual Basic.
16. Menu controls contain only one event, the Click event, which is invoked when the menu control is selected with the mouse or using the keyboard.
17. A pop-up menu is a floating menu that is displayed over a form, independent of the menu bar.
18. Shortcut menu appears when the user right-clicks a selection, toolbar, or taskbar button, for example.
19. Various options of the Menu Editor dialog box are: Caption, Name, Index, Shortcut, HelpContextID, Negotiate Position, Checked, Enabled, Visible, WindowList.

NOTES

20. Visual Basic allows you to create various type of menus for your applications.
21. Menu items should start with the menu that they are part of and append their caption.
22. Sub menus as the name sounds are children of main menus.
23. Access keys are used to navigate a menu hierarchy in conjunction with the Alt keys.
24. Shortcut keys provide immediate invocation of the functionality of menu item.
25. You can set a menu item to enable or disable.
26. You can make a menu item invisible too.
27. You can add activity to a menu item.
28. You can add a separator line to a menu item.
29. Visual Basic has a number of controls. Some of them like labels or list boxes, give users feedback, while others, like command buttons and text boxes, elicit responses.
30. You can use a command button to elicit simple responses from the user or to invoke special functions on forms.
31. Text boxes are commonly used for accepting user input or for entering data.
32. A label control is similar to a text box control is that both display text.

---

### SELFASSESSMENT QUESTIONS

---

1. How would you start Visual Basic?
2. Describe the various options of the opening screen of Visual Basic.
3. Describe the various commands on the following menus:

File menu	Edit menu
View menu	Project menu
Format menu	Debug menu
Run menu	Query menu
Diagram menu	Tools menu
Add Ins menu	Window menu
Help menu	
4. What all is there on the Properties Window?
5. How would you use the Form Layout Window?
6. What are Forms?
7. How would you control various windows in Visual Basic?
8. Describe the various options on the toolbox.
9. What type of Events are there in Visual Basic?
10. What is Immediate Window?
11. What are menus?
12. Describe the various items on the Menu Editor.
13. How would you create a menu item?
14. Describe the method of creating menu with access keys.
15. How would you add keyboard shortcuts to the menu items.
16. What are pop up menus?
17. Describe the process of making a menu item enabled.
18. How would you make a menu item checked?
19. How would you make a menu item invisible?
20. What sort of text boxes can be added to the menus?
21. How would you put a line in between the menu items?
22. What are Programming Controls?
23. How is Text Box used?

24. How is Label button used?
25. What are Option Buttons? How are they used?
26. What are Check Boxes? How are they used?
27. What are Frame Controls?
28. How do you use List Boxes?
29. What are Combo Boxes?
30. How would you use Image Objects button?
31. What are Picture Boxes? How are they used?
32. What are Timers?
33. How would you use Timers?

### Multiple Choice Questions

1. Save project command is there on :  
(a) File menu                      (b) Edit menu                      (c) Project menu
2. Make project group command is there on :  
(a) File menu                      (b) Edit menu                      (c) Project menu
3. Select All command is there on :  
(a) File menu                      (b) Edit menu                      (c) Query menu
4. Insert File command is there on :  
(a) File menu                      (b) Edit menu                      (c) Windows menu
5. Call Stack command is there on :  
(a) File menu                      (b) Edit menu                      (c) View menu
6. Show Panes command is there on :  
(a) File menu                      (b) Edit menu                      (c) View menu
7. Add Property Page command is there on :  
(a) Project menu                      (b) Edit menu                      (c) File menu
8. References command is there on :  
(a) File menu                      (b) Edit menu                      (c) Project menu
9. Align command is there on :  
(a) File menu                      (b) Format menu                      (c) Project menu
10. Vertical Spacing command is there on :  
(a) File menu                      (b) Edit menu                      (c) Format menu
11. Quick watch command is there on :  
(a) Debug menu                      (b) Edit menu                      (c) Project menu
12. Step over command is there on :  
(a) File menu                      (b) Debug menu                      (c) Project menu
13. Add to Output command is there on :  
(a) File menu                      (b) Edit menu                      (c) Query menu
14. Clear results command is there on :  
(a) Query menu                      (b) Edit menu                      (c) Project menu
15. Set Text Font command is there on :  
(a) File menu                      (b) Diagram menu                      (c) Project menu
16. Tile Vertically command is there on :  
(a) File menu                      (b) Window menu                      (c) Project menu
17. Menus are created using :  
(a) Menu Editor                      (b) Menu Creator                      (c) Menu Tools
18. Short cut menu appear on :  
(a) Left click of mouse                      (b) Right click of mouse                      (c) Clicking at menu

NOTES

NOTES

19. Shortcut keys are there for :
  - (a) Show
  - (b) Nothing
  - (c) Immediate action
20. Access keys can be used using which key :
  - (a) Ctrl
  - (b) Shift
  - (c) Alt
21. Two type of dialog boxes used in Visual Basic are Message box and :
  - (a) New box
  - (b) Input box
  - (c) In box
22. Buttons are provided for :
  - (a) General controls
  - (b) Pop-menus
  - (c) Nothing
23. For single controls you use :
  - (a) Text box
  - (b) Combo box
  - (c) Check box
24. For list data you use :
  - (a) Text box
  - (b) List box
  - (c) Check box
25. Combo box combines the features of a text box and :
  - (a) list box
  - (b) check box
  - (c) combo box
26. This control is always hidden at the time of running:
  - (a) Command
  - (b) Timer
  - (c) Date

**True/False Questions**

1. Visual Basic can be started from the Start button.
2. Copy, Paste are there on the toolbar.
3. View Toolbox is there on the toolbar.
4. Delete command is there on the File menu.
5. Print command is there on the File menu.
6. Undo command is there on the Edit menu.
7. Watch Window command is there on the View menu.
8. Exit command is there on the View menu.
9. Zoom command is there on the View menu.
10. Add form is there on the Project menu.
11. Last Position is there on the Project menu.
12. Save Project command is part of File menu.
13. Select All command is part of Edit menu.
14. Object command is part of View menu.
15. Add User Document command is part of Project menu.
16. Add Data Environment is part of Project menu.
17. Align is part of Debug menu.
18. New Text Annotation is part of Diagram menu.
19. Remove Filter is part of Query menu.
20. Menu editor is part of Diagram menu.
21. Modify Custom View command is part of Tools menu.
22. Tile Horizontally is part of Window menu.
23. Procedure attribute command is part of Tools menu.
24. Properties Window has the following options: Object Box, Properties List Tabs and Description Pane.
25. Immediate window displays information resulting from debugging statements in your code or from commands typed directly into the window.
26. You cannot tile all document windows in Visual Basic.
27. You can show or hide document windows and display project documents when you open a project.

28. You cannot expand or contract a docking window in its docking area.
29. You can change a docked window to a floating window, dock a floating window, and position a floating window over a dock (without docking it).
30. Check box is part of Visual Basic toolbox.
31. Label is not there on Visual Basic toolbox.
32. Horizontal scroll bar is part of Visual Basic toolbox.
33. Timer is not there on the Visual Basic toolbox.
34. OLE is part of Visual Basic toolbox.
35. There is no naming convention in Visual Basic.
36. In Visual Basic, menus are created using Menu Editor.
37. A pop-up menu cannot be created in Visual Basic.
38. Shortcut menu can be called using right button of the mouse.
39. Menu items should start with the menu that they are part of and append their caption.
40. Sub menus as the name sounds are children of main menus.
41. Access keys can be activated using Shift keys.
42. You can set a menu item to enable or disable.
43. You cannot make a menu item invisible.
44. You can add activity to a menu item.
45. You can add a separator line to a menu item.
46. There are seven methods of the CodeModule class that you can use to modify code.
47. You can use a label button to elicit simple responses from the user or to invoke special functions on forms.
48. Text boxes are commonly used for accepting user input or for entering data.
49. A label control is similar to a text box control is that both display text.
50. Option button controls, also called radio buttons are used to allow the user to select one and only one, option from a group of options.
51. List boxes are valid as single controls – a single option button is probably counter-intuitive.
52. A list box is an ideal way of representing users with a list of data.
53. A combo box combines the features of both a text box and a list box.
54. The timer control is one of the few controls always hidden at run time.
55. A scroll bar control on a form is not a scroll bar on a large text box or list box.

## NOTES

### Short Questions with Answers

1. Which are the various editions of Visual Basic?

**Ans.** Various editions of Visual Basic are:

**Working Model:** It lacks important functionality and is available with some books on Visual Basic.

**Learning Model:** It was earlier known as Standard Edition. It is good if just trying to learn the software and not use it for advanced purposes.

**Professional Model:** It features all that which is not available in Learning model. In fact it is the complete model.

**Enterprise Model:** It includes the features of Professional Edition in the context of enterprise and remote development.

2. What is a Pop-up menu?

**Ans.** A pop-up menu is a floating menu that is displayed over a form, independent of the menu bar. The items displayed on the pop-up menu depend on the location of the pointer when the right mouse button is pressed; therefore, pop-up menus are also called context menus. You should use pop-up menus to provide an efficient method for accessing common, contextual commands.

NOTES

3. What convention you have to follow while naming the menus?

**Ans.** Menu items should start with the menu that they are part of and append their caption. The New menu item in the File menu should be named `mnuFileNew` for example and the Open menu item in the File menu should be named `mnuFileOpen`. Because the Objects list in the Code Editor is alphabetical, this practice serves the purpose of grouping all the menu items on a menu.

4. What are Access keys and Shortcut keys?

**Ans.** Access keys are used to navigate a menu hierarchy in conjunction with the Alt keys. If the access key for the File menu is F and the access key for the New item in the File menu is N, pressing Alt + F followed by Alt + N has the same effect as clicking the New menu item — that is, it invokes the code in the New menu item's Click event. It is a principle of interface usability that every menu item can be reached via a combination of access keys.

Shortcut keys differ from access keys in that they provide immediate invocation of the functionality of menu item. To reach a menu item using access keys, you must navigate the menu hierarchy. Shortcut keys are assigned to menu items (not menus) via a drop-down menu in the Menu Editor. Shortcut keys—actually keyboard combination, such as Ctrl+O—must be unique across an entire application.

5. What are common dialog boxes?

**Ans.** The common dialog control provides an interface between Visual Basic and the procedures in the Microsoft Windows dynamic-link library `Commdlg.dll`.

6. What are Color Dialog Box and Font Dialog Box?

**Ans.** The Color dialog box allows the user to select a colour from a palette or to create and select a custom colour. At run time, when the user chooses a colour and closes the dialog box, then you use the Color property to get the selected colour.

The Font dialog box allows the user to select a font by its size, color and style. Once you make selections in the Font dialog box, you can use the properties containing information given in the following table.

7. Which are the various mouse controls?

**Ans.** The `MouseDown`, `MouseUp` and `MouseMove` events use the button argument to determine which mouse button or buttons are pressed. The button argument is one bit-field argument — a value in which each bit represents a state or condition. These values are expressed as integers. The three least-significant (lowest) bits represent the left, right and middle mouse buttons. The default value of each bit is 0 (False). If no buttons are pressed, the binary value of the three bits is 000. If you press the left button, the binary value or pattern, changes to 001. The left button bit-value changes from 0 (False) to 1 (True).

8. Which are the various controls in Visual Basic?

**Ans.** Visual Basic has a number of controls. Some of them like labels or list boxes, give users feedback, while others, like command buttons and text boxes, elicit responses. Other controls sit quietly, invisible to the user and perform some of the grunt work that makes your application useful. The timer control is one example of an invisible control.

9. What is Text box?

**Ans.** Nearly every Visual Basic project involves at least one text box control. Text boxes are commonly used for accepting user input or for entering data. Their properties are, of course specifically designed for these purposes. If you only want the simplest of user responses, you might consider using an `InputBox` instead. The `InputBox` displays a dialog box and prompts the user to enter something and returns this to the application.

10. What is a Label control?

**Ans.** A label control is similar to a text box control in that both display text. The main difference however is that a label displays read-only text as far as the user is concerned though you can alter the caption as a run-time property.

11. What is a check box?

**Ans.** A check box control is rather similar to an option button, which was described in the last

section. Both often partake in group and the Value property is tested to see if a check box is on or off. But there are two fundamental differences between check boxes and option buttons: Check boxes are valid as single controls – a single option button is probably counter-intuitive. Check boxes (even when in a group) are not mutually exclusive. Finally, check boxes have three possible settings for the Value property.

12. What are image controls?

**Ans.** The image control (its prefix is often img) is a lightweight equivalent of the picture box control, which is described in a later section. But unlike the picture control, the image control can't act as a container for other objects. In some of its other properties it's not as versatile, but it's a good choice if you simply want to display a picture on a form. Image controls consume far less memory than picture controls. The image control that comes with Visual Basic can now display bitmap (.BMP), icon (.ICO), metafile (.WMF), JPEG (.JPG) and GIF (.GIF) files. This makes it easier to display graphics from the World Wide Web as well as graphics from other popular graphics programs.

13. What is the difference between image control and picture boxes?

**Ans.** As you might expect, picture boxes often display graphics (for example, bitmaps, icons, JPEGs and GIFs). In this role, picture boxes are similar to image controls. However, picture boxes and images have slightly different properties and therefore behave differently. If you just want to show a picture, then an image control is usually a better choice than a picture box. An image control takes up less memory and is a lightweight version of the picture box control.

14. What is a menu control?

**Ans.** A menu control is an object; like other objects it has properties that can be used to define its appearance and behavior. You can set the Caption property, the Enabled and Visible properties, the Checked property, and others at design time or at run time. Menu controls contain only one event, the Click event, which is invoked when the menu control is selected with the mouse or using the keyboard.

15. Which are the menus of Visual Basic?

**Ans.** Various menus of Visual Basic are:

File menu	Edit menu
View menu	Project menu
Format menu	Debug menu
Run menu	Query menu
Diagram menu	Tools menu
Add-Ins menu	Window menu
Help menu	

16. What is a combo box?

**Ans.** A combo box control (at least in two of its styles available in Visual Basic) allows you to select a predefined item from a list or to enter a new item not in the list. A combo box can also incorporate a drop-down section – that means it takes less room on a form than a normal list box. In all, there are three types of combo boxes to choose from at design time: a drop-down combo, a simple combo and a drop-down list. You can specify the type by setting the Style property.

17. What is an option button?

**Ans.** Option button controls, also called radio buttons are used to allow the user to select one and only one, option from a group of options. Usually option buttons are grouped together within a frame control but they can also be grouped on a plain form, if there is to be only one group of option buttons. Thus, if you had a frame specifying a delivery method, you might have one button for UPS (United Parcel Service) and another for Courier delivery. Products can only be shipped by one of these methods (not both-and not more). In contrast, option buttons representing, say, bold and italic settings for text would not make sense. Text can be both bold and italic or neither (none).

NOTES

NOTES

**Multiple Choice Questions**

- |       |       |       |       |
|-------|-------|-------|-------|
| 1. a  | 2. a  | 3. b  | 4. b  |
| 5. c  | 6. c  | 7. a  | 8. c  |
| 9. b  | 10. c | 11. a | 12. b |
| 13. c | 14. c | 15. b | 16. b |
| 17. a | 18. b | 18. c | 20. c |
| 21. b | 22. a | 22. c | 24. b |
| 25. a | 26. b |       |       |

**True False Questions**

- |       |       |       |       |
|-------|-------|-------|-------|
| 1. T  | 2. T  | 3. T  | 4. F  |
| 5. T  | 6. T  | 7. T  | 8. F  |
| 9. T  | 10. T | 11. F | 12. T |
| 13. T | 14. T | 15. T | 16. T |
| 17. F | 18. T | 19. T | 20. F |
| 21. F | 22. T | 23. T | 24. T |
| 25. T | 26. F | 27. T | 28. F |
| 29. T | 30. T | 31. F | 32. T |
| 33. F | 34. T | 35. F | 36. T |
| 37. F | 38. T | 39. T | 40. T |
| 41. F | 42. T | 43. F | 44. T |
| 45. T | 46. T | 47. F | 48. T |
| 49. T | 50. T | 51. F | 52. T |
| 53. T | 54. T | 55. F |       |

# FUNDAMENTALS OF VISUAL BASIC

---

### LEARNING OBJECTIVES

After going through this chapter, you should appreciate the following:

- Statements in Visual Basic
- Writing Codes
- Dialog Box
- Variables
- Type of Variable String Numbers.

---

## STATEMENTS IN VISUAL BASIC

---

## NOTES

In Visual Basic you would often need to store values temporarily when performing calculations. For example, you might want to calculate several values, compare them, and perform different operations on them, depending on the result of the comparison. You need to retain the values if you want to compare them, but you don't need to store them.

Visual Basic, like most programming languages, uses **variables** for storing values. Variables have a name (the word you use to refer to the value the variable contains), and a **data type** (which determines the kind of data the variable can store). **Arrays** can be used to store indexed collections of related variables.

**Constants** also store values, but as the name implies, those values remain constant throughout the execution of an application. Using constants can make your code more readable by providing meaningful names instead of numbers. There are a number of built-in constants in Visual Basic, but you can also create your own.

**Data types** control the internal storage of data in Visual Basic. By default, Visual Basic uses the Variant data type. There are a number of other available data types that allow you to optimize your code for speed and size when you don't need the flexibility that Variant provides.

In Visual Basic each data type has its own characteristics, as is shown below:

<i>Data Type</i>	<i>Purpose</i>
Integer	A numeric variable, holds values in the range of -32,768 to 32,768
Long	A numeric variable with a wider range than integer
Single	A numeric variable, holds numbers with decimal places
Double	A numeric variable with a wider range than single
Currency	For holding monetary values
String	For holding text or string values
Byte	A numeric variable with a range of 0 to 255, even less than integer
Boolean	For holding True or False values
Date	For holding date values
Object	For holding references to objects in Visual Basic and other applications
Variant	A general-purpose variable that can hold most other types of variable values

### Using Integer

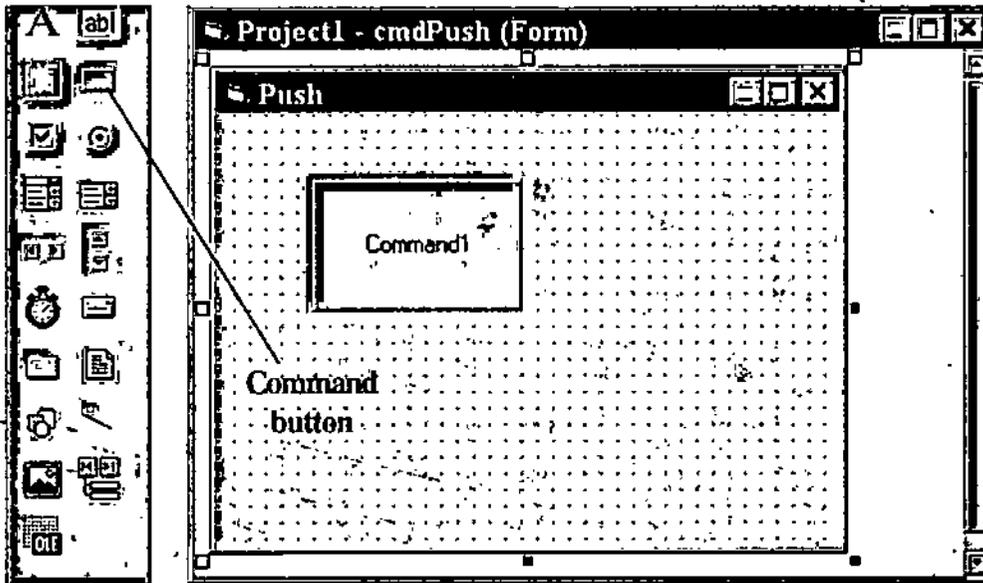
An integer is a numeric data type much like a byte data type but it can be signed, i.e., it may have positive or negative value. It can hold the value of a minimum of -32,768 to a maximum of 32,767. These are useful for simple mathematics where

you know that values are not going to exceed the integer's range. Integers are also useful when used as counters.

Let us create a small project and see how integer data type is used.

1. Start a new project.
2. Add a command button to the form.

NOTES

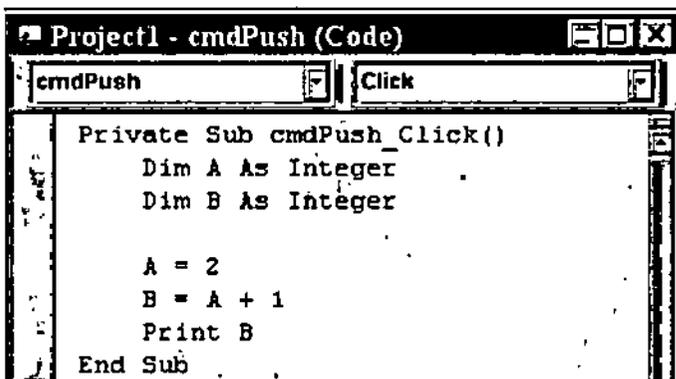
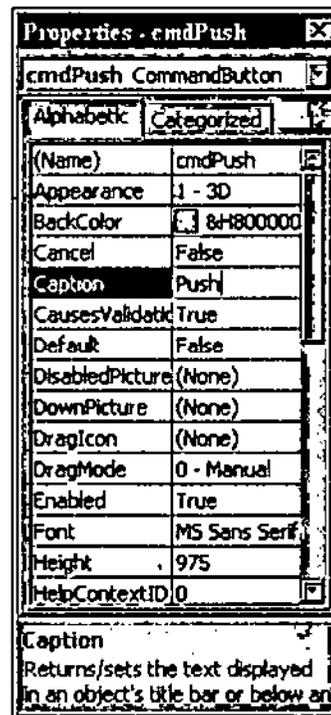


3. Set its Name property to cmdPush. Set its Caption to Push.
4. Double-click the command button to open the Code window.
5. Add the following code to the command button's Click() event:

```
DIM A As Integer  
DIM B As Integer
```

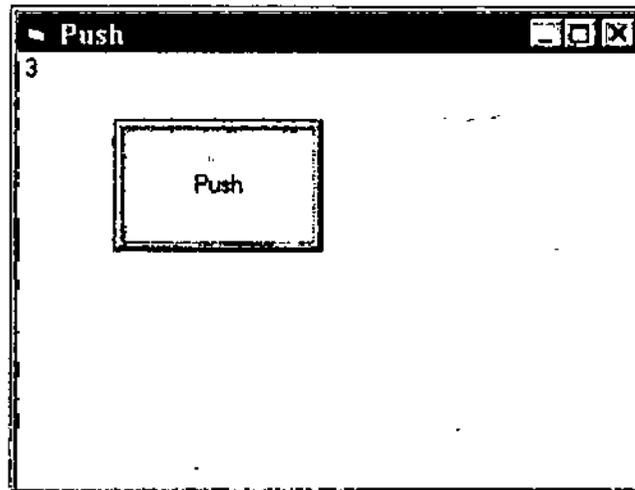
```
A = 2  
B = A + 1  
Print B
```

6. Run the project and click the button.



7. Since  $2 + 1$  is 3. So everytime you click the push button you will see a new 3 appearing on the screen, as shown below.

NOTES



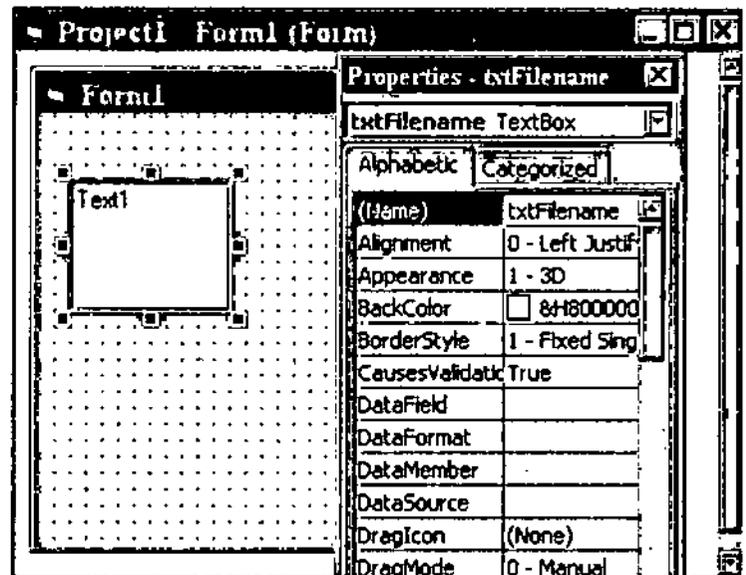
Let us now see how the Boolean Variable can be used.

### Using Boolean Variable

As mentioned earlier this variable of the type which gives the values of True or False. You can set the return code to True if the function was successful, or to False if the function failed. Let us look at it with an example.

1. Start a new project.
2. In this case select Standard EXE from the Project Wizard.

3. Add a Text Box control to the Form1 and set its name as txtFilename.

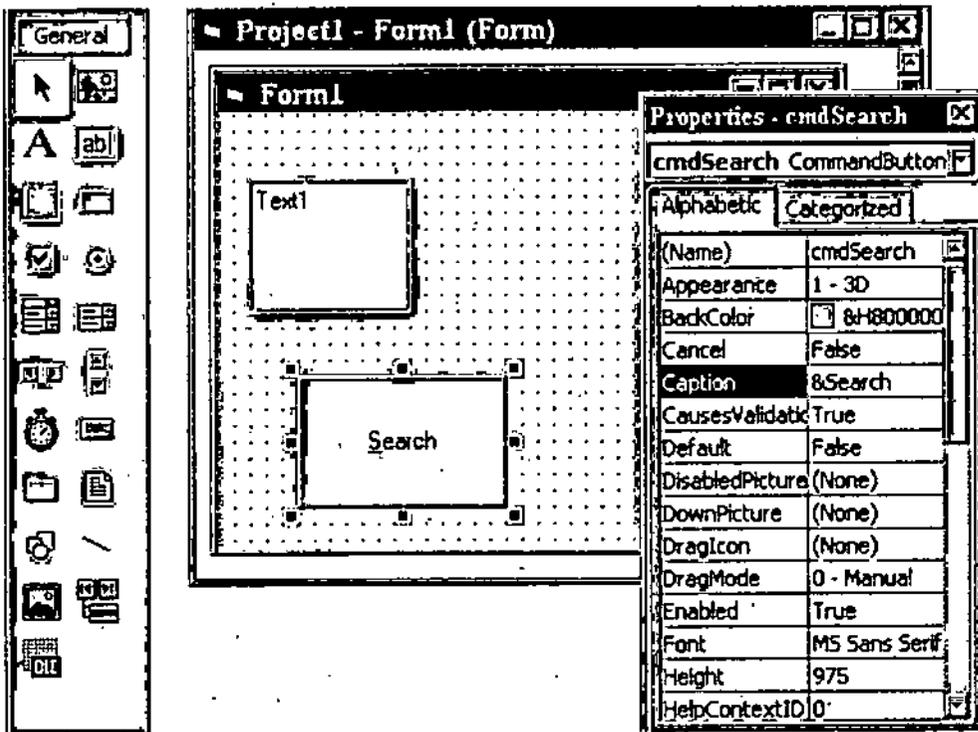


4. Blank the Caption property of the Text Box by double clicking the Caption property in the Properties window and deleting Text1.

5. Add a command button to the form. Using the Properties window, set its name property to cmdSearch and its caption to &Search.

6. Double click Form1 to open the code window.

7. Type the following code in the Code window to create a wrapper function called IsFile().



NOTES

```

Private Function IsFile(filename As String) As Boolean
    If Len(Dir$(filename)) > 0 Then
        IsFile = True
    Else
        IsFile = False
    End If
End Function

```

8. Select cmdSearch from the object drop-down list (on the top left of the code window). The event will set itself to Click().
9. Add the following code to the Click() event of cmdSearch:

```

Private Sub cmdSearch_Click()
    Dim filename As String
    Dim rc As Boolean

    filename = txtFilename.Text
    rc = IsFile(filename)
    If rc = True Then
        MsgBox "File exists!"
    Else
        MsgBox "File not found!"
    End If
End Sub

```

10. Run the project.

NOTES

```

Project1 - Form1 (Code)
cmdSearch Click

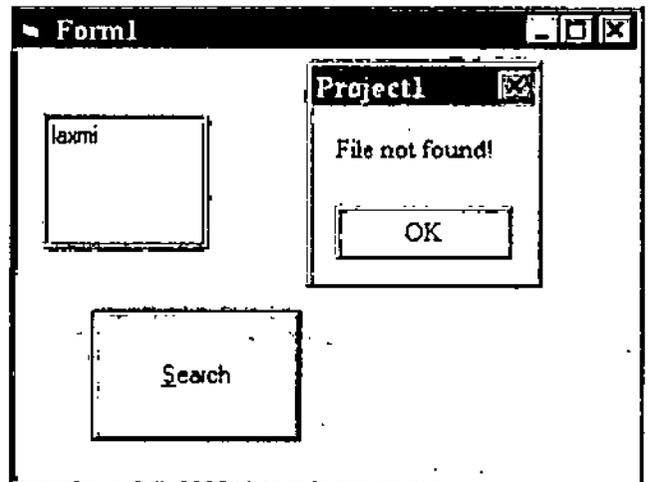
Private Function IsFile(filename As String) As Boolean
    If Len(Dir$(filename)) > 0 Then
        IsFile = True
    Else
        IsFile = False
    End If
End Function

Private Sub cmdSearch_Click()
    Dim filename As Variant
    Dim rc As Boolean

    filename = txtFilename.Text
    rc = IsFile(filename)
    If rc = True Then
        MsgBox "File exists!"
    Else
        MsgBox "File not found!"
    End If
End Sub
    
```

As you can see from here, I had given the name `laxmi` in the file name box and pressed Search button to find out whether the file name exists or not. The result has been given to me as `File not found!`

So now with these example, I hope you would have understood how these various Data Types are used in Visual Basic.



### Long Data Type

Long (long integer) variables are stored as signed 32-bit (4-byte) numbers ranging in value from -2,147,433,648 to 2,147,483,647. The type-declaration character for Long is the ampersand (&).

### Single Data Type

Single (single-precision floating-point) variables are stored as IEEE 32-bit (4-byte) floating-point numbers, ranging in value from -3.402823E38 to -1.401298E-45 for negative values and from 1.401298E-45 to 3.402823E38 for positive values. The type-declaration character for Single is the exclamation point

### Double Data Type

Double (double-precision floating-point) variables are stored as IEEE 64-bit (8-byte) floating-point numbers ranging in value from -1.79769313486232E308 to -4.94065645841247E-324 for negative values and from 4.94065645841247E-324 to

1.79769313486232E308 for positive values. The type-declaration character for Double is the number sign (#).

## The Currency Data Type

Use the Currency data type instead of Numeric for monetary values. If you specify more than four decimal places in a currency expression, Visual Basic rounds to four places before evaluating the expression.

To assign the Currency data type, use the dollar sign:

```
money = $50.33
moremoney = $675.43886
```

## The String Data Type

If you have a variable that will always contain a string and never a numeric value, you can declare it to be of type String:

```
Private S As String
```

You can then assign strings to this variable and manipulate it using string functions:

```
S = "Database"
S = Left(S, 4)
```

By default, a string variable or argument is a variable-length string; the string grows or shrinks as you assign new data to it. You can also declare strings that have a fixed length. You specify a fixed-length string with this syntax:

```
String * size
```

For example, to declare a string that is always 50 characters long, use code like this:

```
Dim EmpName As String * 50
```

If you assign a string of fewer than 50 characters, EmpName is padded with enough trailing spaces to total 50 characters. If you assign a string that is too long for the fixed-length string, Visual Basic simply truncates the characters.

## The Byte Data Type

If the variable contains binary data, declare it as an array of the Byte data type. Using Byte variables to store binary data preserves it during format conversions. When String variables are converted between ANSI and Unicode formats, any binary data in the variable is corrupted.

All operators that work on integers work with the Byte data type except unary minus. Since Byte is an unsigned type with the range 0-255, it cannot represent a negative number. So for unary minus, Visual Basic assigns the Byte to a signed integer first. All numeric variables can be assigned to each other and to variables of the Variant type. Visual Basic rounds off rather than truncates the fractional part of a floating-point number before assigning it to an integer.

## The Date Data Type

Date and time values can be contained both in the specific Date data type and in Variant variables. The same general characteristics apply to dates in both types.

When other numeric data types are converted to Date, values to the left of the decimal

NOTES

represent date information, while values to the right of the decimal represent time. Midnight is 0, and midday is 0.5. Negative whole numbers represent dates before December 30, 1899.

## The Object Data Type

### NOTES

Object variables are stored as 32-bit (4-byte) addresses that refer to objects within an application or within some other application. A variable declared as Object is one that can subsequently be assigned (using the Set statement) to refer to any actual object recognized by the application.

```
Dim objDb As Object
Set objDb = OpenDatabase("c:\Vb5\Biblio.mdb")
```

When declaring object variables, try to use specific classes (such as TextBox instead of Control or, in the preceding case, Database instead of Object) rather than the generic Object. Visual Basic can resolve references to the properties and methods of objects with specific types before you run an application. This allows the application to perform faster at run time. When working with other applications' objects, instead of using a Variant or the generic Object, declare objects as they are listed in the Classes list in the Object Browser. This ensures that Visual Basic recognizes the specific type of object you're referencing, allowing the reference to be resolved at run time.

## The Variant Data Type

A Variant variable is capable of storing all system-defined types of data. You don't have to convert between these types of data if you assign them to a Variant variable; Visual Basic automatically performs any necessary conversion. For example:

```
Dim SomeValue ` Variant by default.
SomeValue = "17" ` SomeValue contains "17" (a two-
                  ` character string).
SomeValue = SomeValue - 15 ` SomeValue now contains
                          ` the numeric value 2.
SomeValue = "U" & SomeValue ` SomeValue now contains
                          ` "U2" (a two- character string).
```

While you can perform operations on Variant variables without much concern for the kind of data they contain, there are some traps you must avoid.

1. If you perform arithmetic operations or functions on a Variant, the Variant must contain something that is a number.
2. If you are concatenating strings, use the & operator instead of the + operator.

---

## WRITING CODES

---

Initially when the operating system was DOS, then the programs were written keeping in mind the operating system which had no support at all for graphics. Event coloured outputs were rare. With the introduction of Windows as the operating system, more and more use of graphics started. Even the programs which were written now supported colours and graphics. Most of the DOS based programs were modified to support graphics. In quite a number of cases, the software were added prefix Visual to make

them look that they support the visual effects too. One such example is Visual Basic. But before going into Visual Basic as such, let us learn about the few programming concepts.

## Modular Programming

It is a method in which long programs are divided into smaller programs or modules that can be designed, coded and debugging separately with a minimum amount of interaction. Let us refresh our memory and remember what we studied about modular programming in the last chapter.

In modular programming instead of running a big program altogether, the program is divided into small parts and they are run independently. These parts are self sufficient and can be run alone. These parts are called subroutines or modules.

A module can be further divided into sub modules. All these small modules join together and can be called as superior modules. A superior module should be capable of calling sub-modules in programming with the help of calling statement. Modular Programming does not use GOTO statement for the purpose of calling a sub-module by a module. This is so since after executing the modules, the control has to come back whereas in the case of GOTO the control goes out of hand.

There are different ways of using the modular programming. They are:

1. Sequence structure
2. Decision/Selection structure
3. Loop structure

Read about them in the last chapter.

## Object Oriented Programming

Visual Basic is an object-based programming language. The mere mention of objects may cause undue anxiety in many programmers. Don't worry; whether you realize it or not, you've been dealing with objects most of your life. Once you understand a few basic concepts, objects actually help to make programming easier than ever before.

If you've programmed in other languages, much of the material covered in this chapter will seem familiar. While most of the constructs are similar to other languages, the event-driven nature of Visual Basic introduces some subtle differences. Try and approach this material with an open mind; once you understand the differences you can use them to your advantage.

If you're new to programming, the material in this chapter will serve as an introduction to the basic building blocks for writing code. Once you understand the basics, you will be able to create powerful applications using Visual Basic.

## Event Driven Programming

In event driven programming each application is event-driven, meaning that the flow of program execution is controlled by the events that occur as the program is running. As you will read later most of these events are the direct result of actions initiated by the users; however, some events are invoked by other objects.

In traditional or "procedural" applications, the application itself controls which portions of code execute and in what sequence. Execution starts with the first line of

NOTES

NOTES

code and follows a predefined path through the application, calling procedures as needed.

In an event-driven application, the code doesn't follow a predetermined path — it executes different code sections in response to events. Events can be triggered by the user's actions, by messages from the system or other applications, or even from the application itself. The sequence of these events determines the sequence in which the code executes, thus the path through the application's code differs each time the program runs.

Because you can't predict the sequence of events, your code must make certain assumptions about the "state of the world" when it executes. When you make assumptions (for example, that an entry field must contain a value before running a procedure to process that value), you should structure your application in such a way as to make sure that the assumption will always be valid (for example, disabling the command button that starts the procedure until the entry field contains a value).

Your code can also trigger events during execution. For example, programmatically changing the text in a text box cause the text box's Change event to occur. This would cause the code (if any) contained in the Change event to execute. If you assumed that this event would only be triggered by user interaction, you might see unexpected results. It is for this reason that it is important to understand the event-driven model and keep it in mind when designing your application.

### **Introduction to Visual Basic Programming**

Visual Basic, is the fastest and easiest way to create applications for Microsoft Windows. Whether you are an experienced professional or brand new to Windows programming, Visual Basic provides you with a complete set of tools to simplify rapid application development.

Let us first see what Visual Basic is. The **Visual** part refers to the method used to create the Graphical User Interface (GUI). Rather than writing numerous lines of code to describe the appearance and location of interface elements, you simply add prebuilt objects into place on screen. If you've ever used a drawing program such as Paint, you already have most of the skills necessary to create an effective user interface.

The **Basic** part refers to the BASIC (Beginners All-Purpose Symbolic Instruction Code) language, a language used by more programmers than any other language in the history of computing. Visual Basic has evolved from the original BASIC language and now contains several hundred statements, functions, and keywords, many of which relate directly to the Windows GUI. Beginners can create useful applications by learning just a few of the keywords, yet the power of the language allows professionals to accomplish anything that can be accomplished using any other Windows programming language.

The Visual Basic programming language is not unique to Visual Basic. The Visual Basic programming system, Applications Edition included in Microsoft Excel, Microsoft Access, and many other Windows applications uses the same language. The Visual Basic Scripting Edition (VBScript) is a widely used scripting language and a subset of the Visual Basic language. The investment you make in learning Visual Basic will carry over to these other areas. Whether your goal is to create a small utility for yourself or your work group, a large enterprise-wide system, or even distributed applications spanning the globe via the Internet, Visual Basic has the tools you need.

- Data access features allow you to create databases, front-end applications, and scalable server-side components for most popular database formats, including Microsoft SQL Server and other enterprise-level databases.
- ActiveX™ technologies allow you to use the functionality provided by other applications, such as Microsoft Word word processor, Microsoft Excel spreadsheet, and other Windows applications. You can even automate applications and objects created using the Professional or Enterprise editions of Visual Basic.
- Internet capabilities make it easy to provide access to documents and applications across the Internet or intranet from within your application, or to create Internet server applications.
- Your finished application is a true .exe file that uses a Visual Basic Virtual Machine that you can freely distribute.

NOTES

### Rapid Application Development using Visual Basic

In Visual Basic an application can be created for each of the job which you want to perform. This option is shown later in the chapter with the help of various screens using the Application Wizard. But, before that you must know various other options of the software, e.g., projects.

### Concept of Project in Visual Basic

In Visual Basic a project is a group of related files, typically all the files required to develop a software component. Files can be grouped within a project to create subprojects. Projects can be defined in any way meaningful to the user(s). For example, one project per version, or one project per language. Projects tend to be organized in the same way as file directories.

### Working with Projects

As you develop an application, you work with a project to manage all the different files that make up the application. A project consists of:

- One project file that keeps track of all the components (.vbp).
- One file for each form (.frm).
- One binary data file for each form containing data for properties of controls on the form (.frx). These files are not editable and are automatically generated for any .frm file that contains binary properties, such as Picture or Icon.
- Optionally, one file for each class module (.cls).
- Optionally, one file for each standard module (.bas).
- Optionally, one or more files containing ActiveX controls (.ocx).
- Optionally, a single resource file (.res).

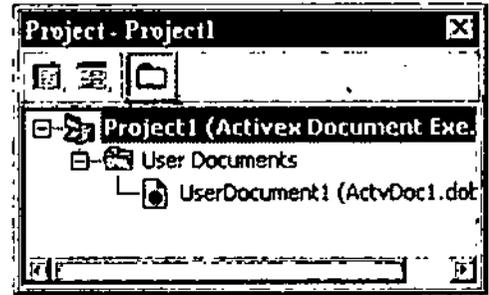
The *project file* is simply a list of all the files and objects associated with the project, as well as information on the environment options you set. This information is updated every time you save the project. All of the files and objects can be shared by other projects as well.

When you have completed all the files for a project, you can convert the project into

an executable file (.exe): From the File menu, choose the Make *project.exe* command.

### The Project Explorer

As you create, add, or remove editable files from a project, Visual Basic reflects your changes in the Project Explorer window, which contains a current list of the files in the project. The Project Explorer window shown here has some of the types of files you can include in a Visual Basic project.



NOTES

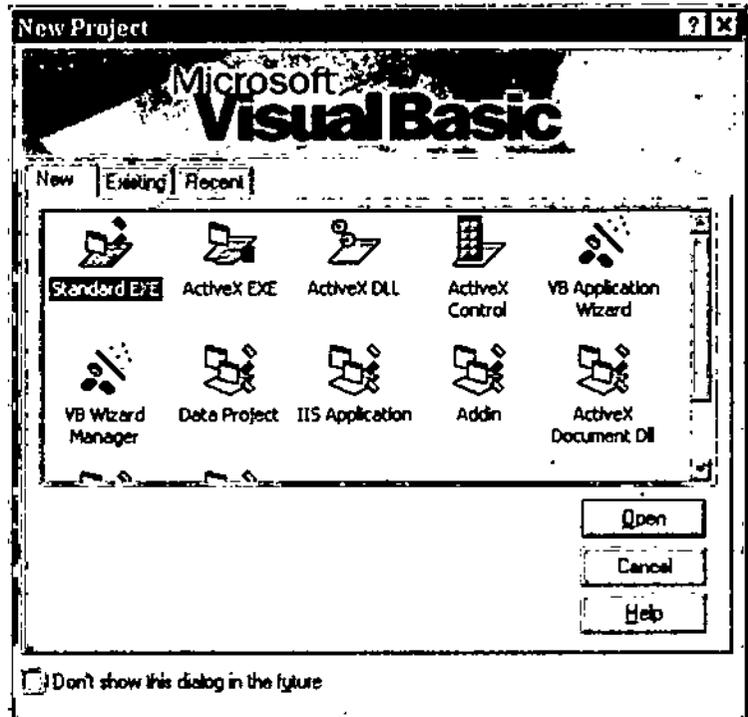
### The Project File

Each time you save a project, Visual Basic updates the project file (.vbp). A project file contains the same list of files that appears in the Project Explorer window, as well as references to the ActiveX controls and insertable objects that are used in the project.

You can open an existing project file by double-clicking its icon, by choosing the Open Project command from the File menu, or by dragging the file and dropping it on the Project Explorer window.

### VB Project Options

As you click at New in the Visual Basic, you get an option to create various types of projects as shown here. This dialog box allows you to select the type of project you want to create. If there is currently another project or project group open, you will be asked to save any changes before the new project and group are created. If you want to add projects to a project group, use the Add Project command on the File menu.



The dialog box has the following options:

#### Standard EXE

Creates a standard executable file.

#### ActiveX EXE

Creates an ActiveX executable file.

#### ActiveX DLL

Creates an ActiveX DLL file.

**ActiveX Control**

Creates an ActiveX control.

**Data Project**

Creates a Data Project.

**IIS Application**

Creates an IIS application.

**Add-In**

The Add-In object provides information about an Add-In to other Add-In objects. The Add-Ins collection contains one or more Add-In objects.

**VB Application Wizard**

Generates a new, fully functional application from which you build a more complex application.

Any project with a .vbp (project file), .vbz (wizard file), or .vbg (project group) extension that is in the vb\template\projects directory.

When an Application Wizard creates a new project, it creates an application wizard object called cApplication, subclassed from the wzApplication object in the Appwiz.vcx library. When the application runs, the main program (projname.prg) creates an instance of the application wizard object cApplication, then uses the object's properties calls its methods to control the application.

In addition, the wizard creates a form called QckStart, which is subclassed from the Wzquickstartform form in the Appwiz.vcx library.

**The Role of ActiveX**

ActiveX leverages Internet technology for several reasons:

- It provides a familiar client/server infrastructure to run your applications.
- Stand-alone Visual Basic applications can be ported over to ActiveX documents so they can then be downloaded via Internet Explorer.
- You can use this same technology to update ActiveX programs on clients' computers: As the application runs from the browser, the document can be configured to request updates when they are available. Then the control can be automatically installed on the client's computers.
- And, it's all done without disks and without installation programs.

**Creating and Using ActiveX Documents**

The ActiveX document allows you to bring the power of Visual Basic to the Internet or your own intranet. An ActiveX document is a Visual Basic application that utilizes Microsoft Internet Explorer as an application container. ActiveX documents allow you to create portable versions of your application that can be used on laptops, at remote offices, or even from your home. Everything runs from your browser. However, an ActiveX document is not a Web page, it is its own application. In addition, users can navigate between ActiveX documents and Web pages seamlessly through their browser.

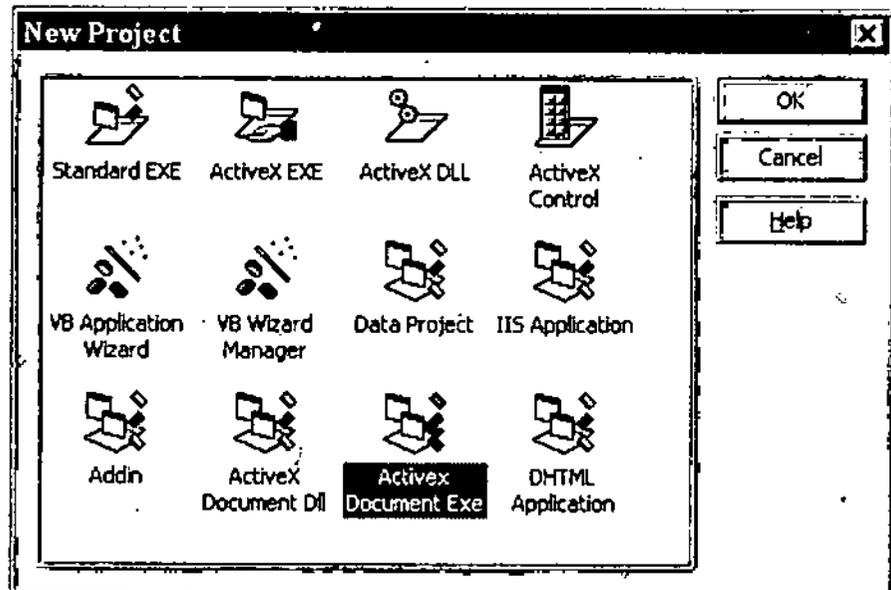
NOTES

When you design your application, you can use ActiveX documents on the front-end to serve as the interface and ActiveX DLLs on the back end to do the processing.

To make the ActiveX document shown in the beginning of the skill, follow the following steps:

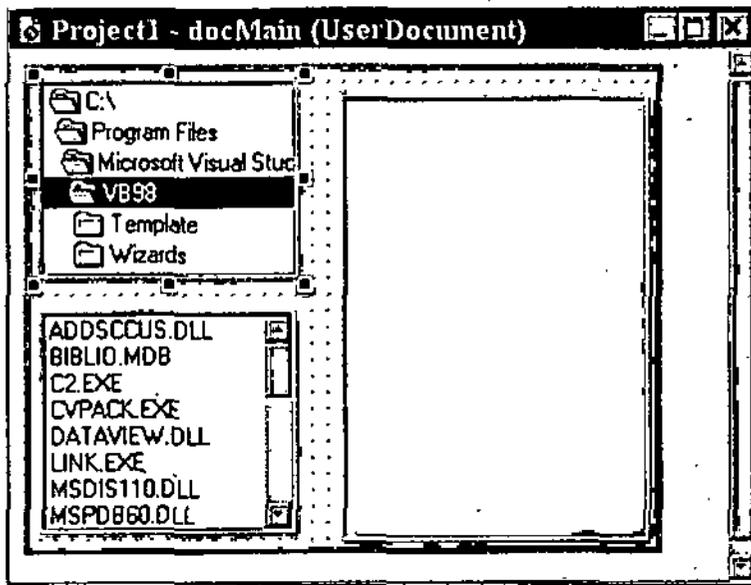
## NOTES

1. Start a new project by selecting File>New Project. Select ActiveX Document EXE from the Project Wizard.
2. In the Properties window, set the Name property of UserDocument1 to docMain.
3. Add a drive control to the upper-left corner of docMain. Place its upper-left corner on a grid line away from the top and left sides. Stretch the control so it is about 2 inches wide.
4. Add a directory list control to docMain, one grid line below the drive control. Stretch it so it is the same width as the drive control. Stretch it vertically so its bottom edge is about half-way down the document.
5. Add a file list control below the directory list control. Stretch it the same width as the directory list and stretch it down to one grid line above the bottom edge of docMain.



6. Now, add a picture box control to the document. Set its Name property to picGraphic.
7. Place picGraphic one grid line to the right of the edge of the drive control. Stretch it so its right edge is one grid line away from the right edge of the document. Drag the bottom of the picture box so its bottom edge is even with the bottom edge of the file list control. Your document should look similar to the figure shown here.
8. Double-click the drive control to open the Code window.
9. Add the following code to the Change() even of Drive1.  

```
Private Sub Drive1_Change()
```



## NOTES

```
'Synchronize directory with drive control
Dir1.Path = Drive1.Path
```

```
End Sub
```

- 9 Add the following code to the Change() event of Dir1:

```
Private Sub Dir1_Change()
```

```
'Synchronize files with directory control
File1.Path = Dir1.Path
```

```
End Sub
```

10. Add this code to the Click() event of File1:

```
Private Sub File1_Click()
```

```
'Show the graphic
picGraphic.Picture = LoadPicture(Dir1.Path & _ "\ " &
File1.filename)
```

```
End Sub
```

11. Unlike forms, you initialize ActiveX documents in the Initialize() event. You can place code in this event to prepare the document before it is displayed.

12. We only want to view bitmap graphics for now. Add the following code to the Initialize() event of the UserDocument object(docMain):

```
Private Sub UserDocument_Initialize()
```

```
'Show only BMP files
File1.Pattern = "*.bmp"
```

```
End Sub
```

13. Add the last bit of code, listed below, to the Resize() event of the docMain:

```
Private Sub UserDocument_Resize()
```

```
'Resize File List
File1.Height = (ScaleHeight - File1.Top)
```

```

`Resize Graphic Window
picGraphic.Height = ScaleHeight
picGraphic.Width = (ScaleWidth - picGraphic.Left)
End Sub

```

## NOTES

The previous code is responsible for making everything on the document look neat. Whenever you resize the browser, the document will resize itself. The code in the `Resize()` event will stretch the picture box control to fit the document. In addition, it will vertically stretch the file list control to be flush with the bottom of the document.

14. Run the project by selecting `Run>Start` or by pressing `F5`.
15. You will be presented with a Debugging tab on a Project Properties dialog box. Notice that the Start Component field is set to `docMain`. Click `OK` to launch the document.

Notice that Internet Explorer automatically launches and displays the document for you. In the previous version of Visual Basic, you needed to manually run Internet Explorer and find the file yourself.

You have now completed your first ActiveX document. This program is a simple graphics viewer that runs in the browser. Now let's see the document work.

1. Using the drive and directory list controls, open the `\Common\Graphics\Bitmaps\Assorted` directory.
2. Click any of the files in the file list box to view it in the picture box.

That's it for your first ActiveX document control. You can enhance this application or create your own. As time passes, you will see more and more ActiveX documents appearing on the Internet as well as intranets around the world. The possibilities are endless.

### Creating and Using ActiveX Controls

ActiveX control is a custom control you create that can be added to the Toolbox and used in your applications. This type of control can then be used in other Visual Basic projects as well as other ActiveX-compliant programs such as Microsoft Excel. These controls can also be embedded and distributed through HTML Web pages.

To create an ActiveX control, start a new project and select ActiveX Control as the project type.

Next you must create the interface. After you do this, you can use the ActiveX Control Interface Wizard to help you finish the job. Let us build a simple button control so you can see how its done:

1. Start a new project. Select ActiveX Control as the project type.
2. Set the Name property of the project to `ActiveXButton`.
3. Set the Name property of the User Control to `ctlExit`.
4. From the Toolbar, add a Command Button to the control designer. Position it so it is flush with the top-left corner of the control container.
5. Set the button's Name property to `cmdExit`. Set its Caption property to `E&xit`.

```

Project1 - docMain (Code)
Dir1 Change
Private Sub Dir1_Change()
    'Synchronize directory with drive control
    Dir1.Path = Drive1.Drive
End Sub

Private Sub Dir1_Change()
    'Synchronize files with directory control
    File1.Path = Dir1.Path
End Sub

Private Sub File1_Click()
    'Show the graphic
    picGraphic.Picture = LoadPicture(c \ windows \ clouds.bmp &
End Sub

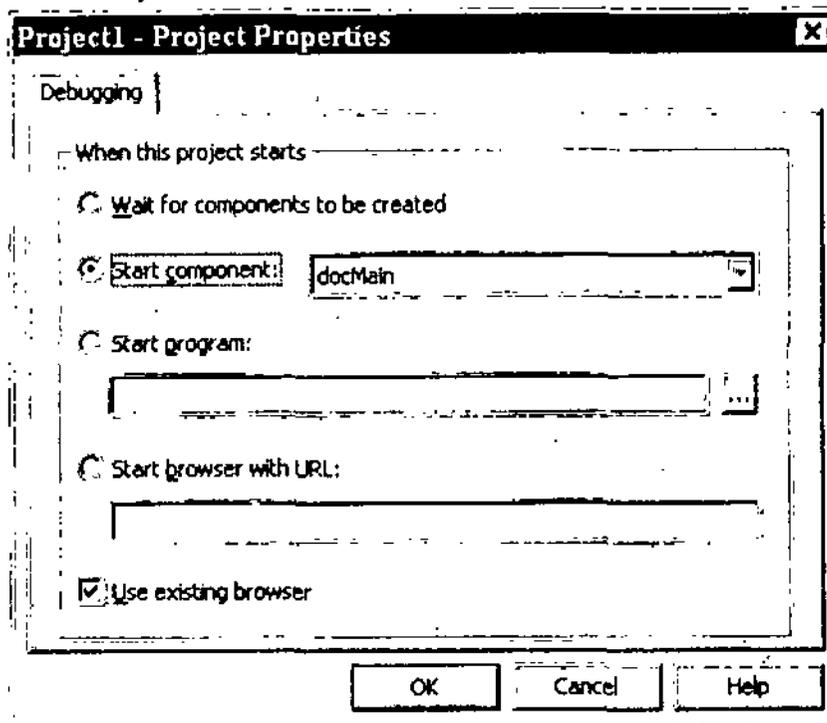
Private Sub UserDocument_Initialize()
    'Show only BHP files
    File1.Pattern = " *.bmp"
End Sub

Private Sub UserDocument_Resize()
    'Resize File List
    File1.Height = (ScaleHeight - File1.Top)
    'Resize Graphic Window

```

NOTES

6. Set the Style property to 1 - Graphical.
7. Now, to add a graphic to the button, set the button's Picture property to the MsgBox01.Ico icon found in the \Common\Graphics\Icons\Computer subdirectory.

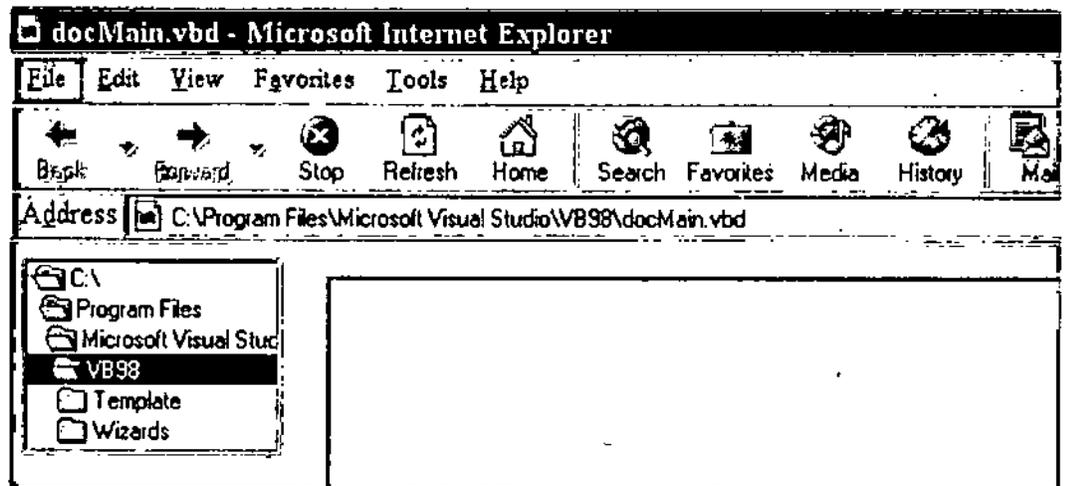






8. Resize the container control (the gray flat form-like ara that the button is sitting on) so it is the same size as the button.

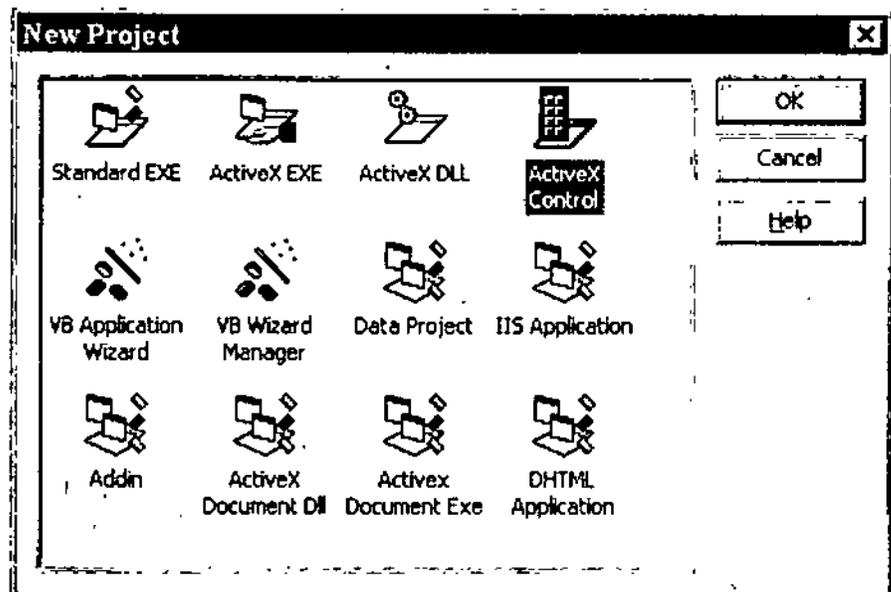
NOTES

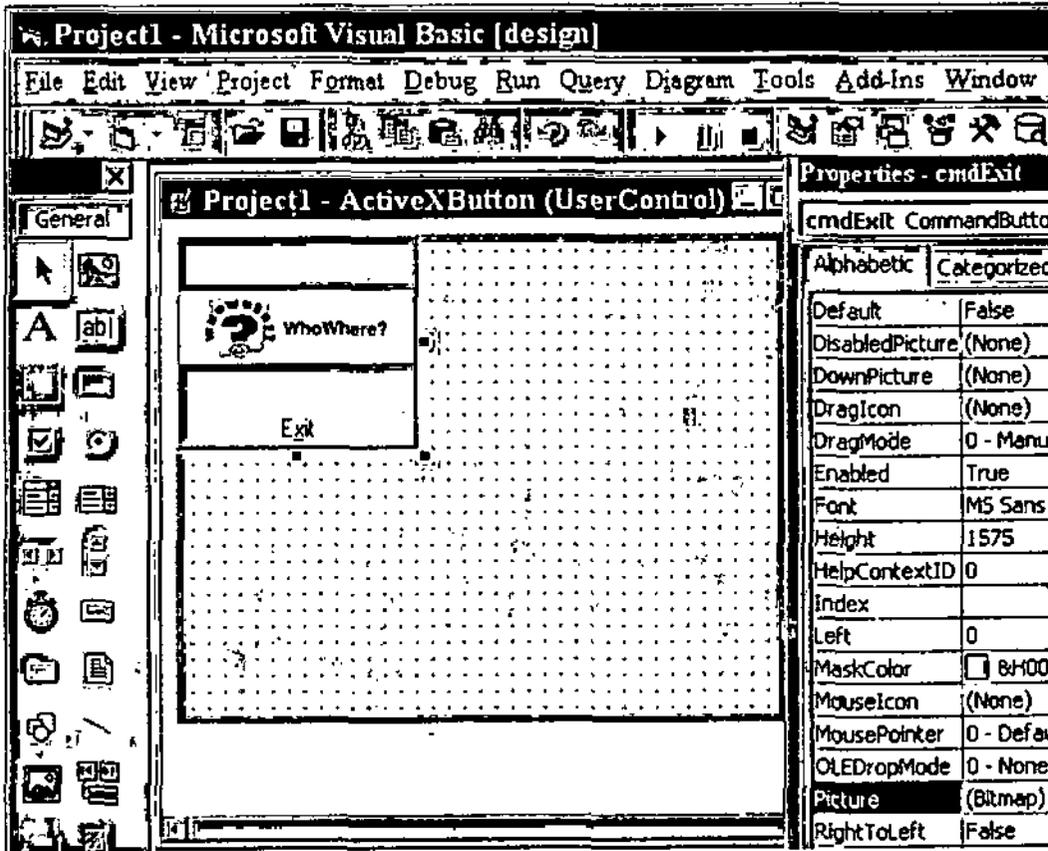


### Adding the Code

Now you have finished with the interface to the control. The next step is to put some code behind it. You will use the ActiveX Control Interface Wizard to accomplish this:

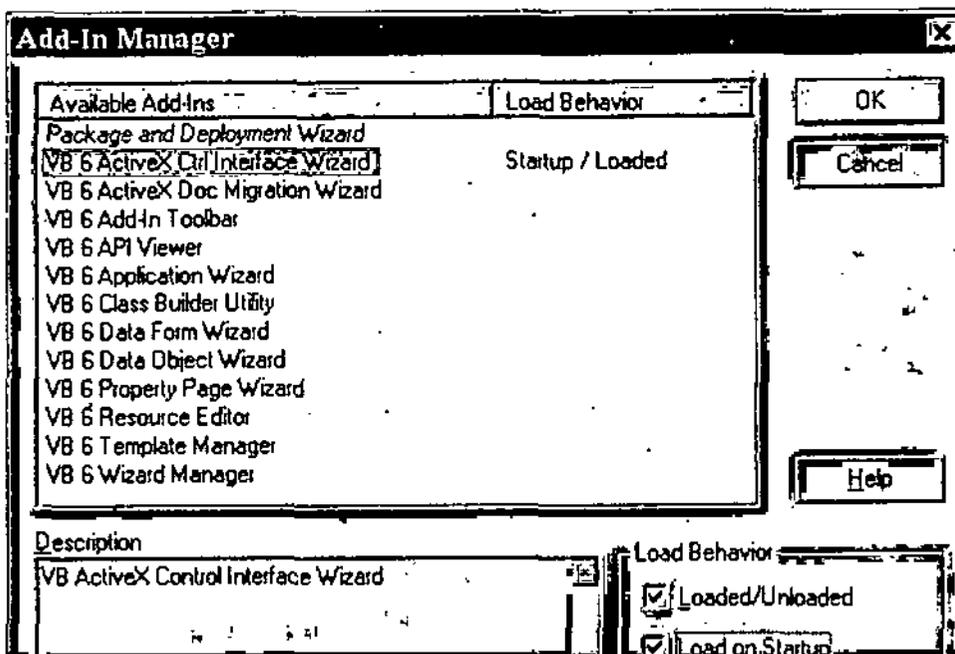
1. Select Add-Ins > Add-In Manager from the menu. This will bring up a dialog box similar to the Components dialog box in the previous exercise.
2. Highlight VB6 ActiveX Ctrl Interface Wizard and check the boxes labeled Loaded/Unloaded, and Load on StartUp. Click the Ok button to close the dialog box. The wizard has now been added to the IDE.
3. Next, select Add-Ins > ActiveX Control Interface Wizard from the menu to start the wizard.
4. When the Introduction dialog box appears, read it and click the Next button.
5. In the next step, you determine which properties, methods, and events your control will need. If you think of your control as any other control and picture





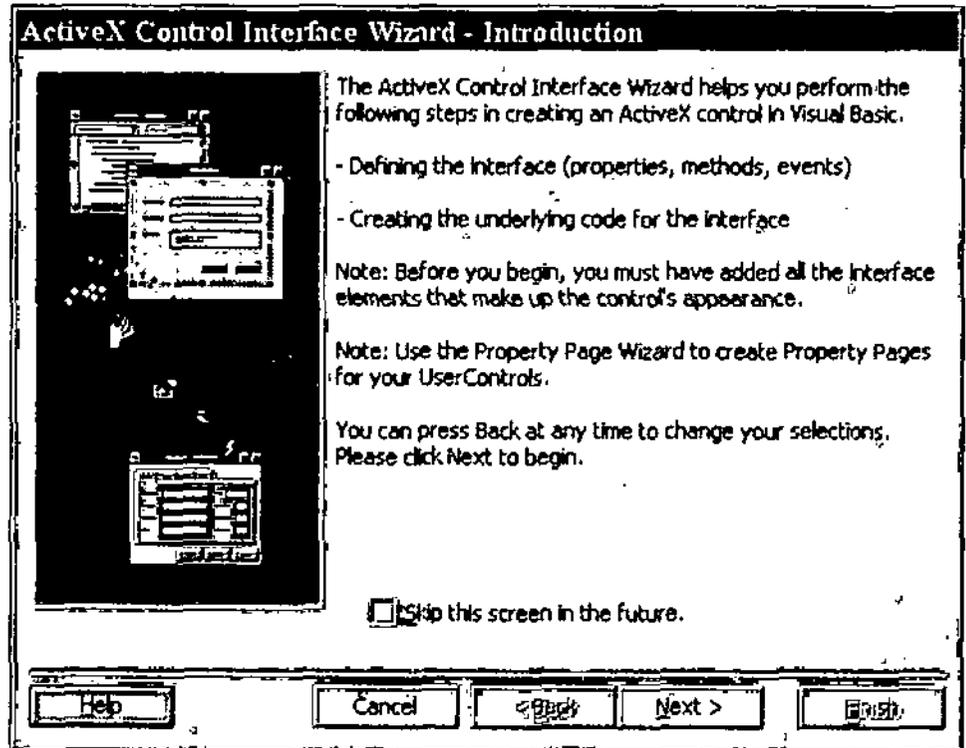
NOTES

- its properties in the Properties window, you can get an idea of what your control will need. This button is very simple, so you only need a few properties and only one event.
- 6. Remove all of the items from Selected Names list except for those listed here. Your object should have Caption, Enabled, Font, and one Click event. You will need to add the Caption property from the Available Names list. When all looks good, click the Next button.

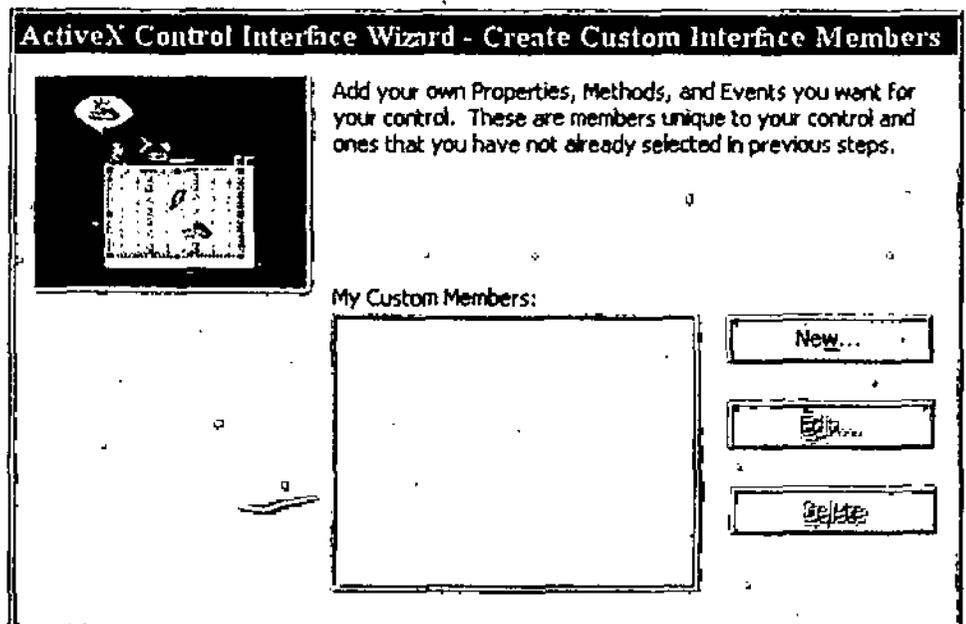


NOTES

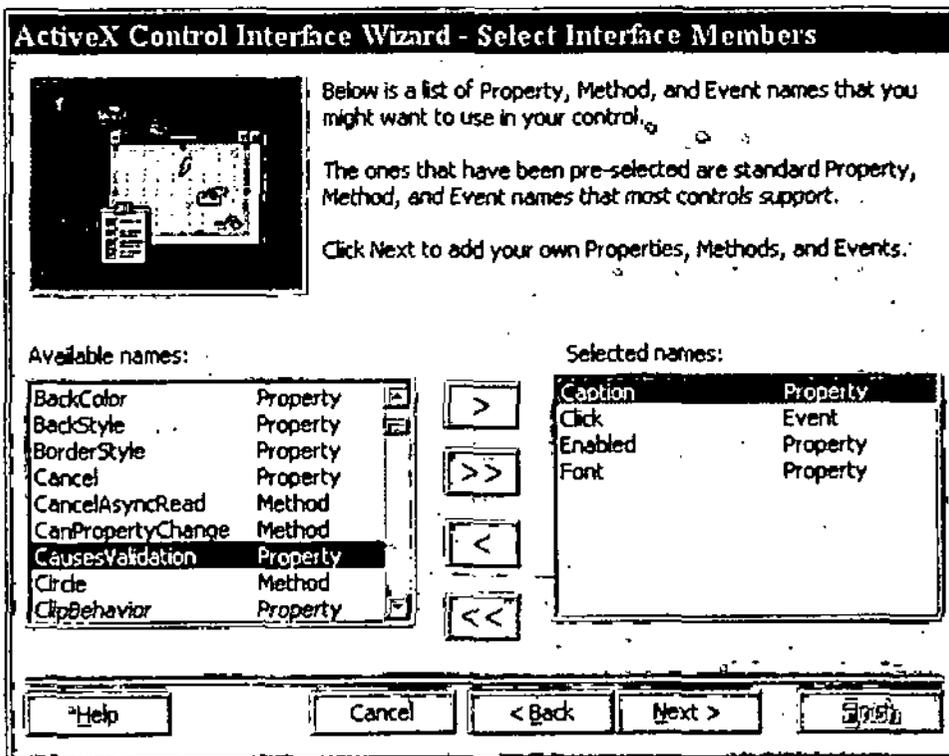
7. The next dialog box will ask to create custom members. Since we have none, just click the Next button to move to the next step.
8. Now you need to map out the control's properties to the properties of its components: Click the Caption property and map it to the Exit button by selecting cmdExit from the Control drop-down list. The Member property should change to Caption.



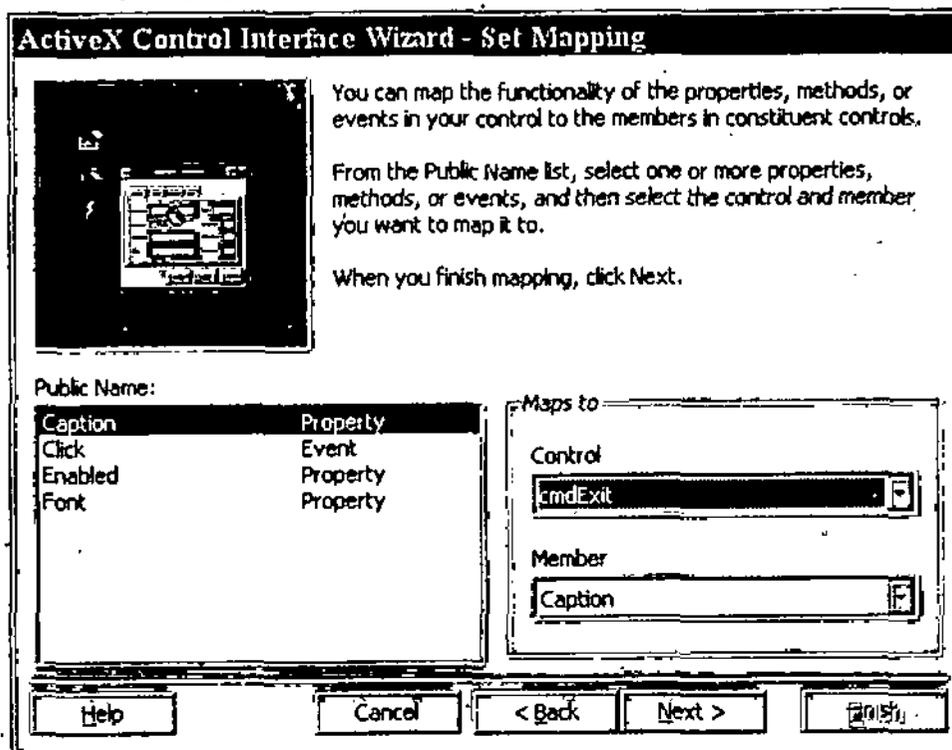
9. Click the Click() event and map it to the Exit button's Click() event.
10. Map the Font property to the Exit button's Font property.



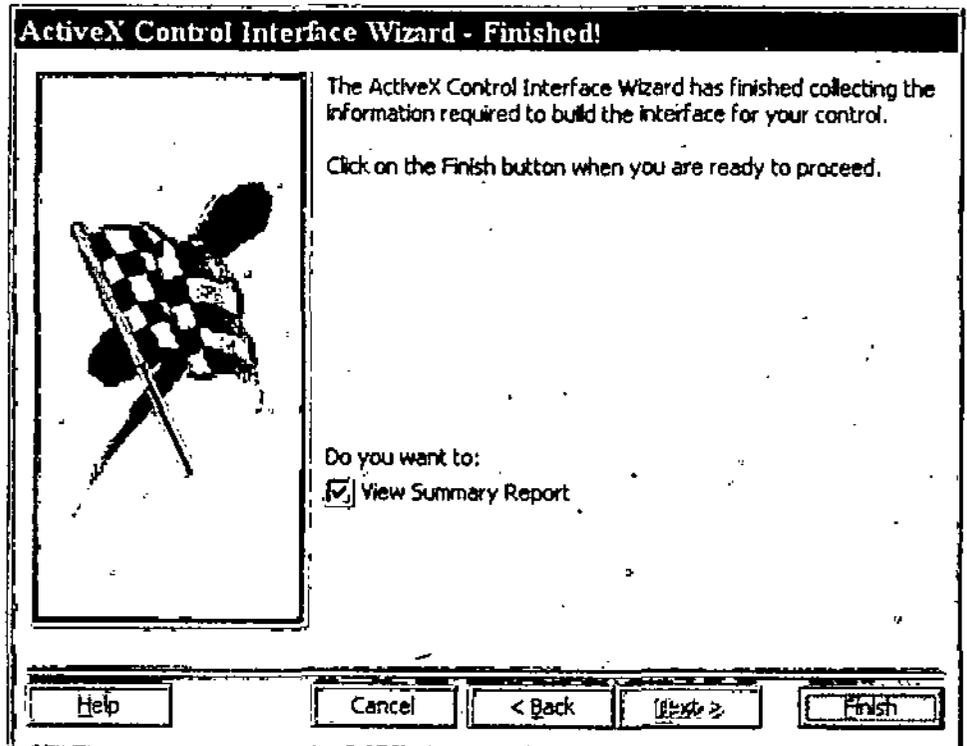
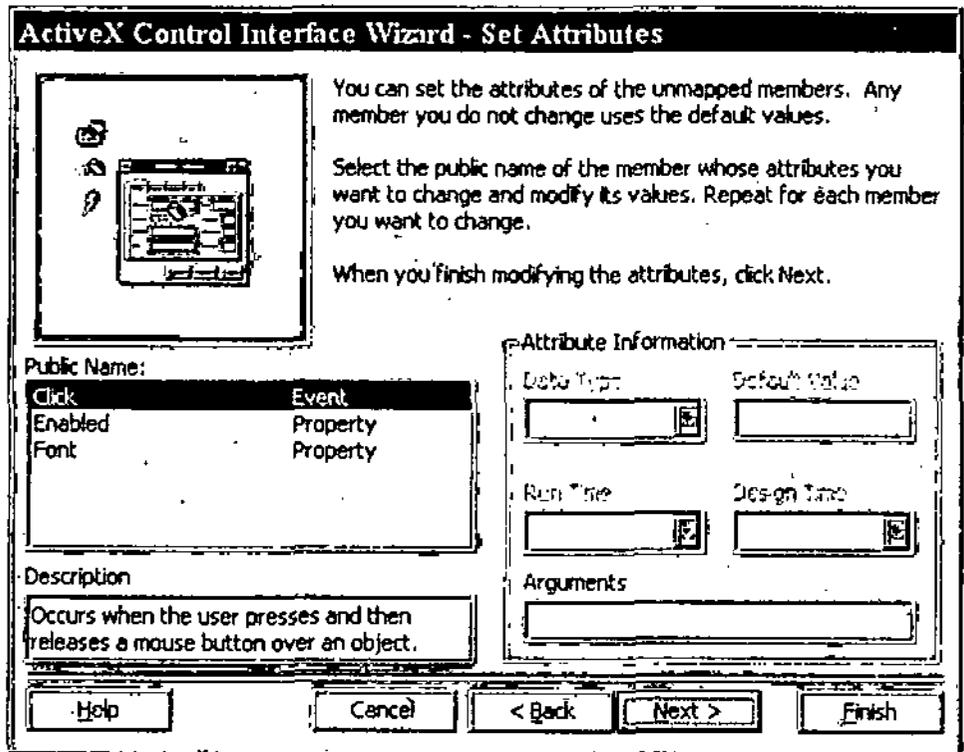
NOTES



11. The next frame of the wizard will ask you if you want to view a summary report. This report contains important information on how to utilize your control when it is finished. If you want to view the report, check the check box. If you do not want to view it, clear the check box.
12. Click the Finish button.
13. Save your project.



NOTES



Congratulations! you have just create your first ActiveX control. Although it seemed like a lot of steps for such a simple control, you will get used to it, and with practice you will be quickly creating ActiveX controls of your own.

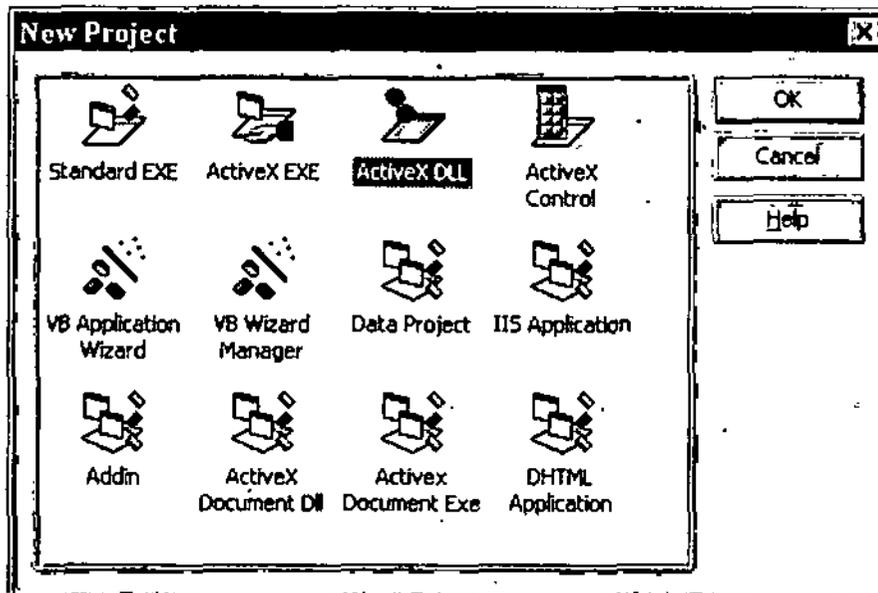
### Creating and using ActiveX DLLs

These components are especially useful when they are linked to databases and run on

servers. These DLLs can be called from browser or your custom Visual Basic front-end. Here we will create a dialog box using ActiveX DLL.

Follow the following steps to create dialog box with ActiveX DLL.

1. Start a new project by selecting File > New project.



NOTES

2. Select ActiveX DLL as the project type, and press Ok.
3. Add a code module to your project and set its Name property to modDialogs.
4. Open the Code window for modDialogs and add the following procedure:

```
Sub Main()  
    'No code is required here. However,  
    'this sub is required for  
    'the DLL to start.  
End Sub
```

5. Double click Class1 in the Project Explorer to make it active.

6. Set the following properties for the class:

```
Name: clsDialogs  
Instancing: 5-Multiuse
```

7. In the first dialog box you are going to add is simple Yes/No. This type is handy for "Are you sure?" type of questions. All you have to do is to create a title and a message and pass them to the YNBox function. Then just check to see if the return code is vbYes or 6.

8. Open the Code window for clsDialogs and add the following function:

```
Public Function YNBox(title As String, msg As String)  
    As Integer  
    Dim rc As Integer  
    Dim DlgDef As Long
```

```

    DlgDef = vbYesNo + vbQuestion
    rc = MsgBox(msg, DlgDef, title)
    YnBox = rc
End Function

```

## NOTES

- The next subroutine we are going to add is an error message dialog box. You can call this from your error-trapping routines so your error messages have a similar look and feel. Create the following sub:

```

Public Sub ErrMsg(title As String, msg As String)
    Dim rc As Integer
    Dim DlgDef As Long
    DlgDef = vbOKCancel + vbCritical
    rc = MsgBox(msg, DlgDef, title)
End Sub

```

To use this subroutine you just pass a title and a message, and the procedure will do the rest of the work for you. The last function we are going to add is a login dialog box. This dialog box's only purpose is to prompt you to enter your UserID. Let us see how this is done.

- Add the following code.

```

Public Function LoginBox(title As String, msg As String,
    -
    default As String) As String
    Dim rc As String
    rc = InputBox(msg, title)
    LoginBox = rc
End Function

```

- Save your project as dialogs.vbp.
- Open the Project Properties dialog box, and type Dialogs in the Project Name field. Then, type A Sample Dialog Class in the Project Description field.
- Select the Make tab and type Dialogs in the Application Title field. Click Ok to close the dialog box.
- Now that we have added the code to our dialog class, we need to compile it so we can use it in our application. Select File > Make Dialog.Dll. Click the Ok button.

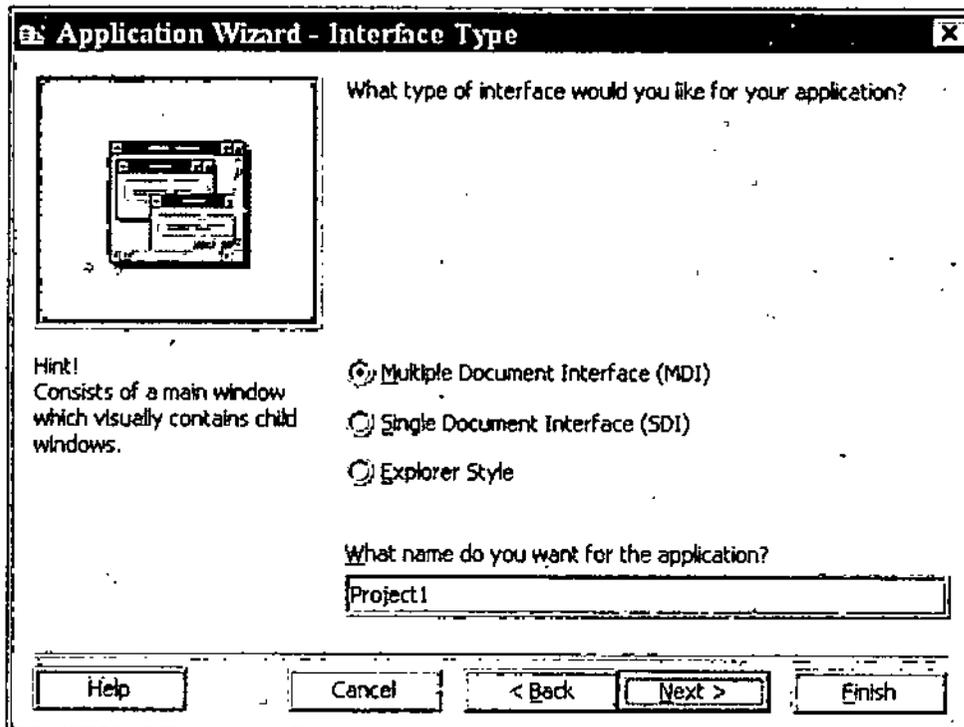
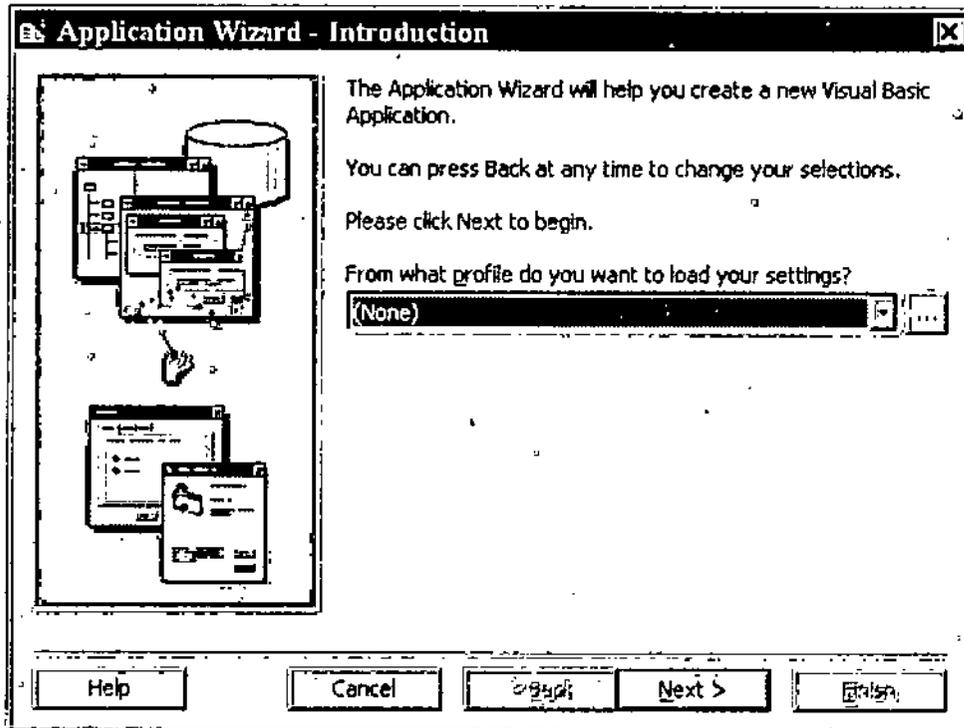
After a short pause, you will have a fully compiled DLL that you can use in other programs.

### Creating Application using Wizard

Now, we will try to generate a new project using the option of Application Wizard from the dialog box, shown on the last page. This application wizard will give you various screens, asking you various options which each one of the screens has. You change the options which are shown there or leave them as it is. For the trial run, I suggest you leave them as it and try it to create a project just for practice. All the options of the screens are shown here.

You may not be familiar with most of these options but as you study Visual Basic you will learn more about them. Once all the screens are done with, you would be shown on the screen that the application has been created and would be available to you on the screen in few seconds.

Next you will see the various options of the application on the screen. You can run this application by choosing Run option from the Run Menu. You would be asked for the password. Since you have not yet provided the password, you can run it by just pressing Escape key.

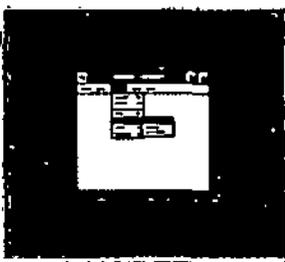


NOTES

### Application Wizard - Menus

Select the Menus and Sub Menu you would like in your application?

You can always use the Menu Editor to modify the menus after the application is created.



Menu	Sub Menu
<input checked="" type="checkbox"/> &File	<input checked="" type="checkbox"/> &New
<input checked="" type="checkbox"/> &Edit	<input checked="" type="checkbox"/> &Open...
<input checked="" type="checkbox"/> &View	<input checked="" type="checkbox"/> &Close
<input type="checkbox"/> &Tools	<input type="checkbox"/> [Separator]
<input checked="" type="checkbox"/> &Window	<input checked="" type="checkbox"/> &Save
<input checked="" type="checkbox"/> &Help	<input checked="" type="checkbox"/> Save &As...
	<input checked="" type="checkbox"/> Save A&ll
	<input checked="" type="checkbox"/> [Separator]

Reset

### Application Wizard - Customize Toolbar

Customize the toolbar by moving the desired buttons to the list on the right. Change the order with the up/down arrows and add external images with the image button. You may also drag/drop from list to list.



Available Buttons	Selected Buttons
[Separator]	[New]
Arc	Open
Back	Save
Button	[Separator]
Camera	Print
Delete	[Separator]
Disconnect Net Drive	Cut
Double Underline	Copy

Reset

### Application Wizard - Resources

Resource files make it easy to distribute your product in multiple languages and can increase performance.

The resource file will be stored in memory by the Resource Editor Add-In until the project has been saved. Once saved, the .res file will appear in the project list under the Related Documents category.

Would you like to use a Resource file for the strings in your application?

Yes  No

NOTES

**Application Wizard - Internet Connectivity**



The wizard can provide you with a custom web Browser and can add a jump to your home page!

Note: These features require that your users already have an Internet provider!

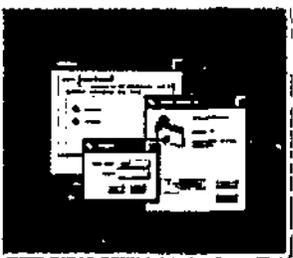
Do you want your users to be able to access the Internet from your application?

Yes       No

To include these features, you must specify a default startup URL for the application.

http://www.microsoft.com

**Application Wizard - Standard Forms**



Would you like to include any of these standard forms in your application?

Splash screen at application start up

Login dialog to accept an ID and Password

Options dialog for custom settings

About Box

To include any custom form templates, click here.

Form Templates...

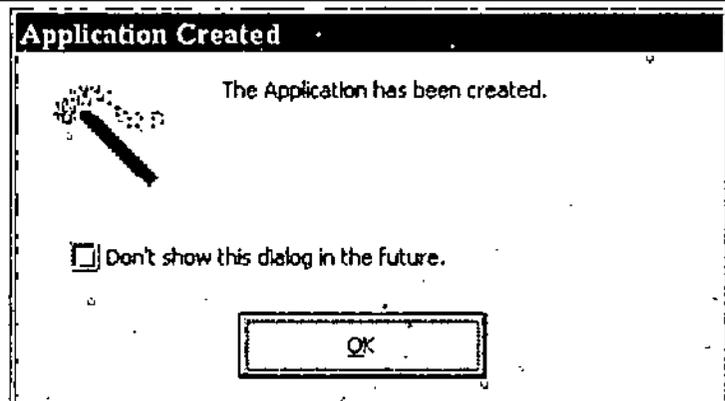
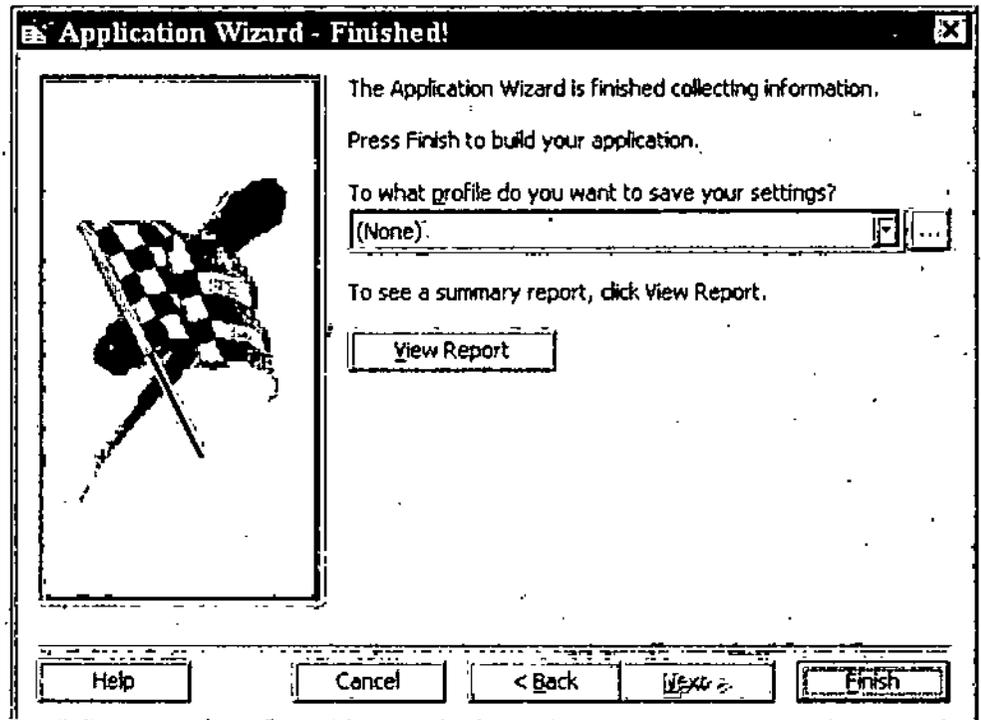
**Application Wizard - Data Access Forms**



Would you like to generate forms based upon tables and queries from your database(s)?

Data Forms:

NOTES



### IIS Application

Among the other tools in the Internet enabled applications is the IIS Application project template. This project template helps lay the framework for a DHTML styled application that runs on Microsoft's Internet Information Server. It utilizes a combination of DHTML, Webclasses, Active Server Pages, and a scripting language such as VBScript or JavaScript to build sophisticated server-side applications.

### Dynamic HTML

DHTML, or Dynamic Hypertext Markup Language, brings additional life to ordinary Web pages. Using Visual Basic you can develop your own DHTML applications. DHTML is based on the Document Object Model, which is a hierarchy of Web page elements. A page in DHTML is like a Form object in Visual Basic. You can write a Visual Basic code inside the events of DHTML elements, just like you do in your normal projects.

### Creating a DHTML Project.

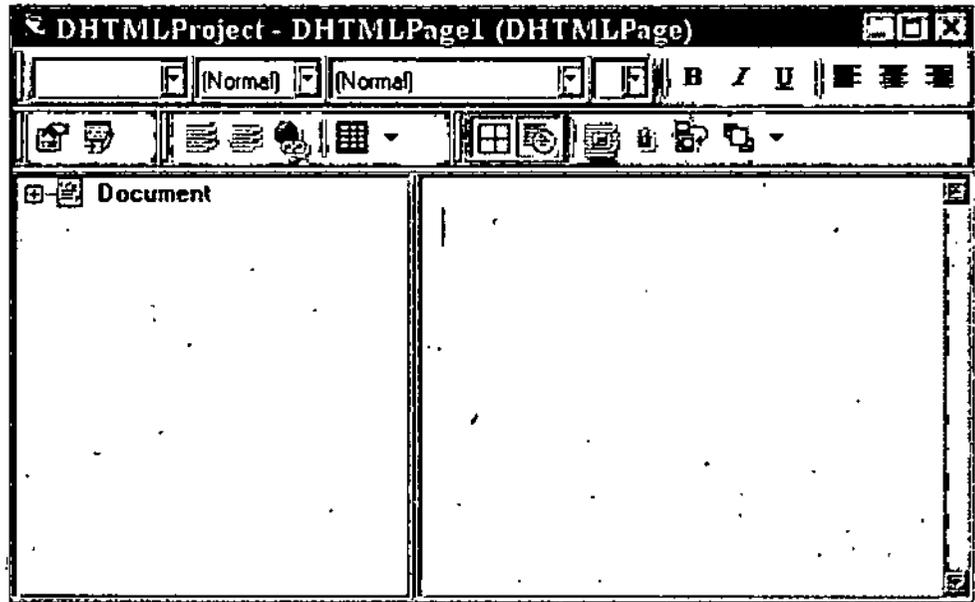
Now we will create a DHTML page that will act as a start page for the WebComm Client application.

1. Create a new project.
2. Select DHTML Application from the New Project dialog box and click ok to create the project.
3. Change the Name property of the Project to WebCommStart. This will be the WebComm Start page.
4. Double click the designer for DHTMLPage1 in the Project Explorer to open it in the Form Designer.
5. As you can see the designer is split into two panes. The pane on the left shows the list of components, called elements of DHTML. You can scroll down to view some of the properties of each element. The pane on the right represents the browser's representation of the page. You can type text or add more complex elements such as CommandButtons and TextBoxes. As you add elements to the page, you will see them appear on the left page, where you can select them so you can modify their properties.
6. Set the ID property of DHTMLPage1 to htmWebCommStart.
7. Click in the right pane and type Welcome to WebComm and press the Enter key.
8. Highlight Welcome to WebComm and change the font to Arial, Bold, 6 using the toolbar at the top of the designer. Set its ID property to pWelcome.
9. Type Brought to You by and double click the Hyperlink element in the toolbox. This will add a hyperlink after the text:
10. Type over the word Hyperlink1 and change it to Laxmi!
11. Highlight Brought to You by Laxmi! and change its font to Arial, Italic, 3. Set its ID property to pLaxmi.
12. Right click the Laxmi hyperlink and select properties from the Property Pages dialog box.
13. Type www.laxmi.com in the Link field. Type Look to Laxmi for all of your computer book needs! in the Pop-up text field. Click the Ok button to commit the changes.
14. In the Properties window, change the ID property to InkLaxmi.
15. Double click the Button element in the Toolbox to add it to the page.
16. Set the ID property to cmdStart, and its Value property to Click Here to Start.
17. Double click the page to open the Code window.
18. Add the following line of code to the (General)(Declaration) section:  

```
Private Const LNK_WEBCOMM_CLIENT = " "
```
19. Type the complete drive and path to the WebComm.vbd file that you created in the previous example. This the path that the page will look for when cmdStart is clicked.
20. Much like a Form object has a Load() event in Visual Basic, a DHTML application has an onLoad() event. You can place code in this event to configure the page before it is rendered in the browser.

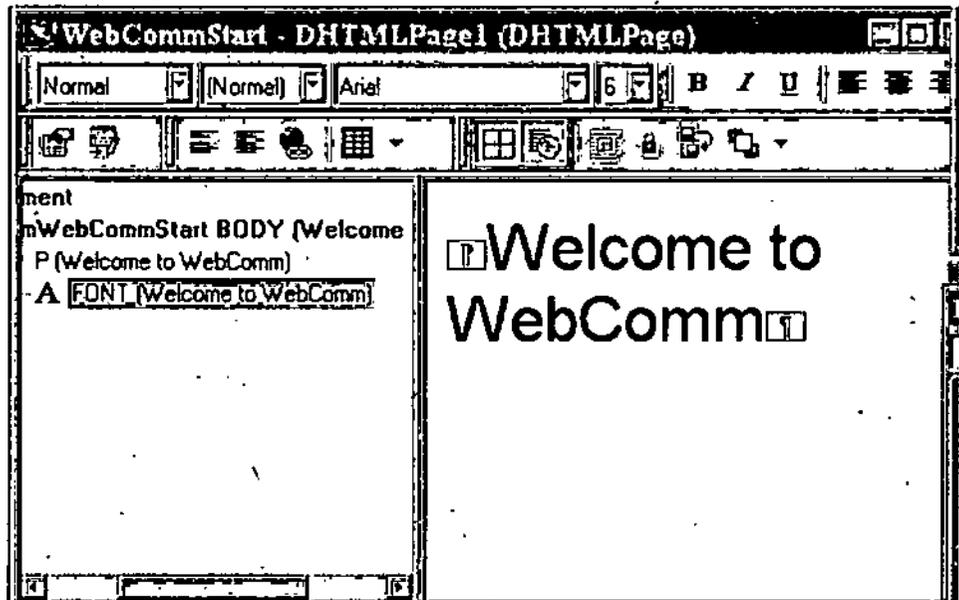
## NOTES

NOTES

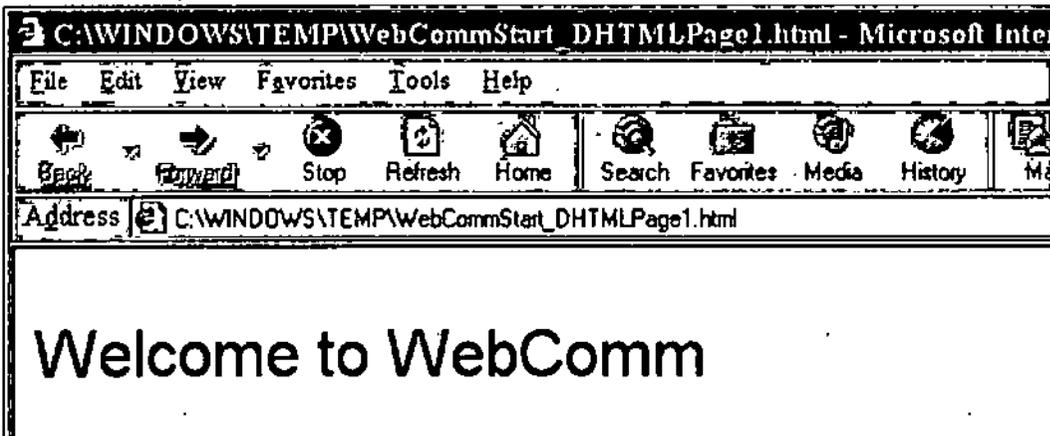


21. Add the following code to the onLoad() event of the BaseWindow object:

```
Private Sub BaseWindow_onLoad()  
    'Set the document properties  
    With Documents  
        .bgColor = "lightyellow"  
        .linkColor = "blue"  
        .vlinkColor = "blue"  
        .alinkColor = "blue"  
    End with  
    'Configure the Welcome paragraph  
    pWelcome.Style.Color = "blue"  
    pLaxmi.Style.Color = "black"  
End Sub
```



22. You may have noticed that instead of using intrinsic constants, the colour values were text. This is because variables in DHTML pages are Variants by default. In addition, HTML uses text instead of constants.



NOTES

23. You may also noticed that the code changes the colour of the Style property of the object. Since DHTML pages extensively utilize style sheets, you change the appearance, or style, of an object through its Style property.
24. In order to add some functionality to the Start button on the form, add the following code to the cmdStart\_onClick() event.

```
Private Function cmdStart_onClick() As Boolean
    'Start the WebComm client
    BaseWindow.navigate LNK_WEBCOMM_CLIENT
End Function
```

25. To add a little flair to the Laxmi link, lets make it change its colour to red when the mouse moves over it. you can do this through the link's onmouseover() event with the following code:

```
Private Sub lnkLaxmi_onmouseover()
    'Change the link to red
    lnkLaxmi.Style.Color = "red"
End Sub
```

26. And naturally, you want the link to reset its appearance when the mouse leaves the link, Do this by adding the following code to the onmouseout() event of lnkLaxmi:

```
Private Sub lnkLaxmi_onmouseout()
    'Change the link to blue
    lnkLaxmi.Style.Color = "blue"
End Sub
```

27. Save and run the project.

## Visual Basic and Windows

In order to understand the application development process, it is helpful to understand some of the key concepts upon which Visual Basic is built. Because Visual Basic is a Windows development language, some familiarity with the Windows environment is

necessary. If you are new to Windows programming, you need to be aware of some fundamental differences between programming for Windows versus other environments.

## How Windows Works: Windows, Events and Messages

### NOTES

Think of a window as simply a rectangular region with its own boundaries. You are probably already aware of several different types of windows: an Explorer window in Windows 98, a document window within your word processing program, or a dialog box that pops up to remind you of an appointment. While these are the most common examples, there are actually many other types of windows. A command button is a window. Icons, text boxes, option buttons and menu bars are all windows. The Microsoft Windows operating system manages all of these many windows by assigning each one a unique id number (window handle or hWnd). The system continually monitors each of these windows for signs of activity or events. Events can occur through user actions such as a mouse click or a key press, through programmatic control, or even as a result of another window's actions.

Each time an event occurs, it causes a message to be sent to the operating system. The system processes the message and broadcasts it to the other windows. Each window can then take the appropriate action based on its own instructions for dealing with that particular message (for example, repainting itself when it has been uncovered by another window). As you might imagine, dealing with all of the possible combinations of windows, events and messages could be mind-boggling. Fortunately, Visual Basic insulates you from having to deal with all of the low-level message handling. Many of the messages are handled automatically by Visual Basic; others are exposed as Event procedures for your convenience. This allows you to quickly create powerful applications without having to deal with unnecessary details.

## Interactive Development

The traditional application development process can be broken into three distinct steps: writing, compiling, and testing code. Unlike traditional languages, Visual Basic uses an interactive approach to development, blurring the distinction between the three steps.

1. With most languages, if you make a mistake in writing your code, the error is caught by the compiler when you start to compile your application. You must then find and fix the error and begin the compile cycle again, repeating the process for each error found. Visual Basic interprets your code as you enter it, catching and highlighting most syntax or spelling errors on the fly. It's almost like having an expert watching over your shoulder as you enter your code.
2. In addition to catching errors on the fly, Visual Basic also partially compiles the code as it is entered. When you are ready to run and test your application, there is only a brief delay to finish compiling. If the compiler finds an error, it is highlighted in your code. You can fix the error and continue compiling without having to start over.
3. Because of the interactive nature of Visual Basic, you'll find yourself running your application frequently as you develop it. This way you can test the effects of your code as you work rather than waiting to compile later.

## DIALOG BOX

Dialog box in Windows environment is used to make a selection, suggest various options, convey messages across, etc. They are there to seek the attention of the user and ask for the response from him. In Visual Basic you have few predefined dialog boxes and then you can create your own too as per your specifications.

### Predefined Dialog Boxes

Visual Basic uses two main type of dialog boxes, message box and input box. Luckily for you both of them are readily available in Visual Basic.

A message box can be displayed in Visual Basic using the following information:

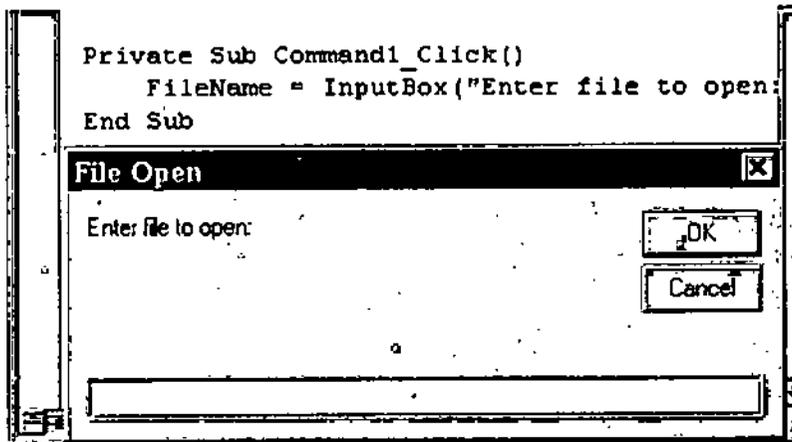
`MsgBox Prompt, DlgDef, Title`

where

**Prompt** It is the text that the user sees in the message box.

**Title** It is the caption in the message box's title bar.

**DlgDef** It is the parameter used to set the Dialog Definition.



Here the input box has been opened with the help of command

`FileName = InputBox("Enter file to open:)`

This has resulted in opening the dialog box, as shown here.

### Custom Dialog Box

As mentioned just before that you can create your own dialog box. So lets do that. Follow the following steps to do so.

1. Start a new project.
2. Select Standard EXE from the Project Wizard.
3. Add a command button to Form1.
4. In the Properties window, set the Name property fo the command button to cmdClickMe.
5. Set the Caption property of cmdClickMe to &ClickMe.

NOTES



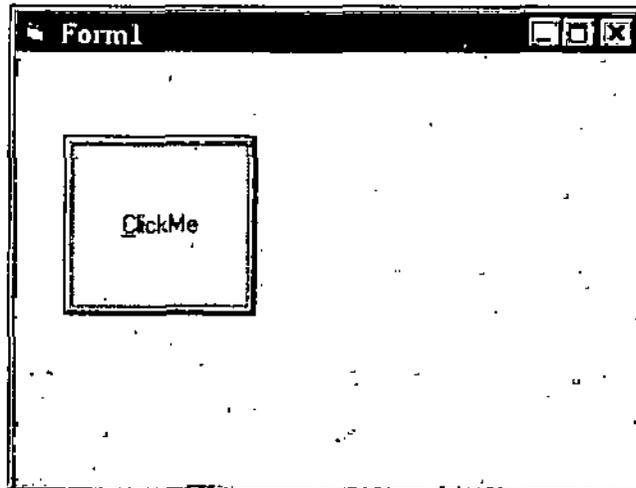


NOTES

```

Project1 - Form1 (Code)
cmdClickMe Click
Private Sub cmdClickMe_Click()
    Dim rc As String
    rc = InputBox("Enter your name here:")
    MsgBox "Hello, " & rc & "!"
End Sub
    
```

6. Double click cmdClickMe to open its Code window.



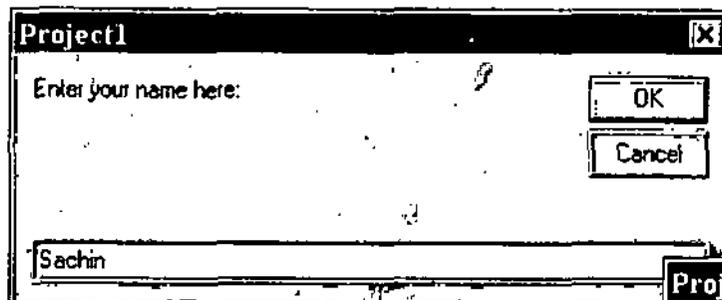
7. Add the following code to the Click() event of cmdClickMe:

```

Private Sub cmdClickMe_Click()
    Dim rc As String

    rc = InputBox("Enter your name here:")
    MsgBox " Hello, " & rc & "!"
End Sub
    
```

8. Run the project.



9. Click the Click Me button to test the InputBox dialog box.
10. When the InputBox appears, enter your name and click the Ok button to see what happens.



---

## VARIABLES AND STRING NUMBERS

---

In Visual Basic, you use variables to temporarily store values during the execution of an application. Variables have a name (the word you use to refer to the value the variable contains) and a data type (which determines the kind of data the variable can store).

You can think of a variable as a placeholder in memory for an unknown value. For example, imagine you are creating a program for a fruit stand to track the sales of apples. You don't know the price of an apple or the quantity sold until the sale actually occurs. You can use two variables to hold the unknown values — let's name them `ApplePrice` and `ApplesSold`. Each time the program is run, the user supplies the values for the two variables. To calculate the total sales and display it in a Textbox named `txtSales`, your code would look like this:

```
txtSales.txt = ApplePrice * ApplesSold
```

### Need to use Variables

In the above examples of Boolean and Integer, we have used variable with the help of Dim statements. Dim is in fact the short form of Dimension statement. It is used to declare a variable. But, this variable can be accessed in the same procedure.

In other words, you can't assign it a value or read its value from any other procedure, whether in the same form, or in another form, or in a standard code module. The variable is a module-level variable and can be accessed from any procedure in the module—its scope is larger than a procedure level variable. There we have the concept of Private and Global level of variables.

You can have a variable that is application wide in its scope. It is called the Global or Public level of variables. So in the above cases for all Global level of variables, we would use Public instead of Private while defining them.

### Declaring Variables

To declare a variable is to tell the program about it in advance. You declare a variable with the Dim statement, supplying a name for the variable:

```
Dim variablename [As type]
```

Variables declared with the Dim statement within a procedure exist only as long as the procedure is executing. When the procedure finishes, the value of the variable disappears. In addition, the value of a variable in a procedure is local to that procedure — that is, you can't access a variable in one procedure from another procedure. These characteristics allow you to use the same variable names in different procedures without worrying about conflicts or accidental changes.

A variable name:

- Must begin with a letter.
- Can't contain an embedded period or embedded type-declaration character.
- Must not exceed 255 characters.
- Must be unique within the same *scope*, which is the range from which the variable can be referenced — a procedure, a form, and so on.

NOTES

NOTES

The optional As type clause in the Dim statement allows you to define the data type or object type of the variable you are declaring. Data types define the type of information the variable stores. Some examples of data types include String, Integer, and Currency. Variables can also contain objects from Visual Basic or other applications.

**Variable Naming Conventions**

Variables in Visual Basic require well-formed naming conventions. Variables should always be defined with the smallest scope possible. Global (Public) variables can create enormously complex state machines and make the logic of an application extremely difficult to understand. Global variables also make the reuse and maintenance of your code much more difficult.

Variables in Visual Basic can have the following scope:

<i>Used in</i>	<i>Found in</i>
<b>Procedure level</b>	
'Private' in procedure, sub, or function	The procedure in which it is declared
<b>Module-level</b>	
'Private' in the declarations section of a form or code module (.frm, .bas)	Every procedure in the form or code module
<b>Global level</b>	
'Public' in the declarations section of a code module (.bas)	Everywhere in the application

In a Visual Basic application, global variables should be used only when there is no other convenient way to share data between forms. When global variables must be used, it is good practice to declare them all in a single module, grouped by function. Give the module a meaningful name that indicates its purpose, such as Public.bas.

It is good coding practice to write modular code whenever possible. For example, if your application displays a dialog box, put all the controls and code required to perform the dialog's task in a single form. This helps to keep the application's code organized into useful components and minimizes its run-time overhead.

With the exception of global variables (which should not be passed), procedures and functions should operate only on objects passed to them. Global variables that are used in procedures should be identified in the declaration section at the beginning of the procedure. In addition, you should pass arguments to subs and functions using ByVal, unless you explicitly need to change the value of the passed argument.

**Assigning Values to Variables**

As you start to isolate the possible cause of an error, you may want to test the effects of particular data values. In break mode, you can set values with statements like these in the Immediate window:

```

BackColor = 255
VScroll1.Value = 100
MaxRows = 50
    
```

The first statement alters a property of the currently active form, the second alters a property of VScroll1, and the third assigns a value to a variable.

After you set the values of one or more variables, you can continue execution to see the results.

## Data Type of Variables

In Visual Basic you declare variables in a number of ways. Most often, you use the Dim statement to declare a variable. If you do not specify the variable type when you use Dim, it creates a variant, which can operate as any variable type. You can specify the variable type using the As keyword like this:

```
Dim IntegerValue As Integer
```

## Scope of Variables

The scope of a variable defines which parts of your code are aware of its existence. When you declare a variable within a procedure, only code within that procedure can access or change the value of that variable; it has a scope that is local to that procedure. Sometimes, however, you need to use a variable with a broader scope, such as one whose value is available to all the procedures within the same module, or even to all the procedures in your entire application. Visual Basic allows you to specify the scope of a variable when you declare it.

### Scoping Variables

Depending on how it is declared, a variable is scoped as either a procedure-level (local) or module-level variable.

#### Procedure-level

Private	Variables are private to the procedure in which they appear.
Public	Not applicable. You cannot declare public variables within a procedure.

#### Module-level

Private	Variables are private to the module in which they appear.
Public	Variables are available to all modules.

### Variables used Within a Procedure

Procedure-level variables are recognized only in the procedure in which they're declared. These are also known as local variables. You declare them with the Dim or Static keywords. For example:

```
Dim intTemp As Integer
```

-or-

```
Static intPermanent As Integer
```

Values in local variables declared with Static exist the entire time your application is running while variables declared with Dim exist only as long as the procedure is executing.

Local variables are a good choice for any kind of temporary calculation. For example, you can create a dozen different procedures containing a variable called **intTemp**. As long as each **intTemp** is declared as a local variable, each procedure recognizes only

NOTES

its own version of `intTemp`. Any one procedure can alter the value in its local `intTemp` without affecting `intTemp` variables in other procedures.

### *Variables used Within a Module*

By default, a module-level variable is available to all the procedures in that module, but not to code in other modules. You create module-level variables by declaring them with the `Private` keyword in the Declarations section at the top of the module. For example:

```
Private intTemp As Integer
```

At the module level, there is no difference between `Private` and `Dim`, but `Private` is preferred because it readily contrasts with `Public` and makes your code easier to understand.

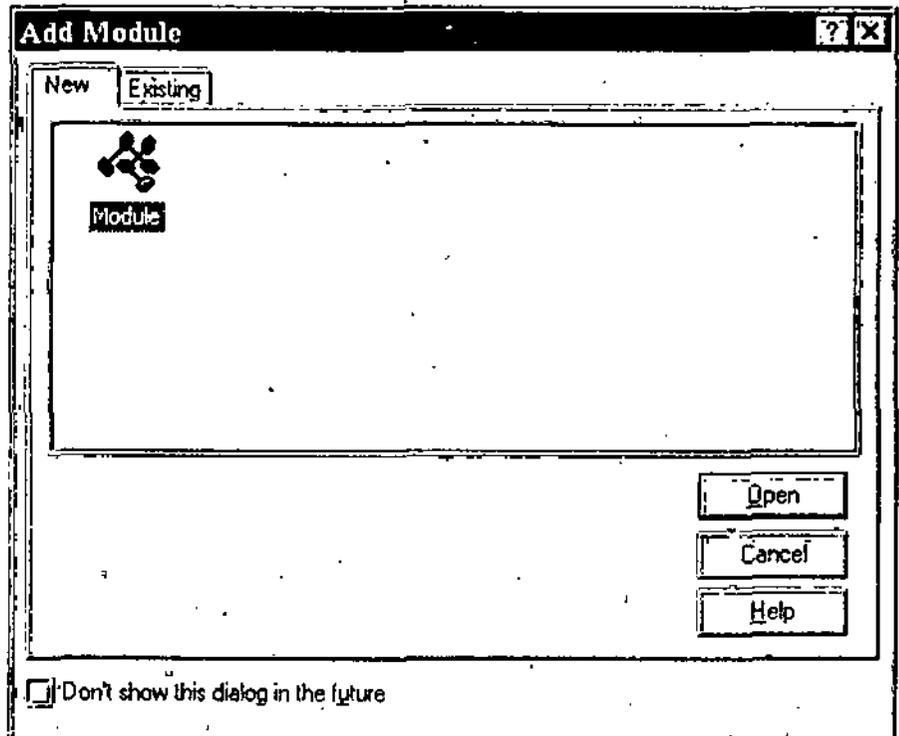
### *Variables used by All Modules*

To make a module-level variable available to other modules, use the `Public` keyword to declare the variable. The values in public variables are available to all procedures in your application. Like all module-level variables, public variables are declared in the Declarations section at the top of the module. For example:

```
Public intTemp As Integer
```

Let us now try to understand how these various scopes work by looking at an example, given below.

1. Create a new project.
2. Select Standard EXE from the Project Wizard.
3. Add a code module to the project by right-clicking in the Project Explorer and selecting Add>Module.



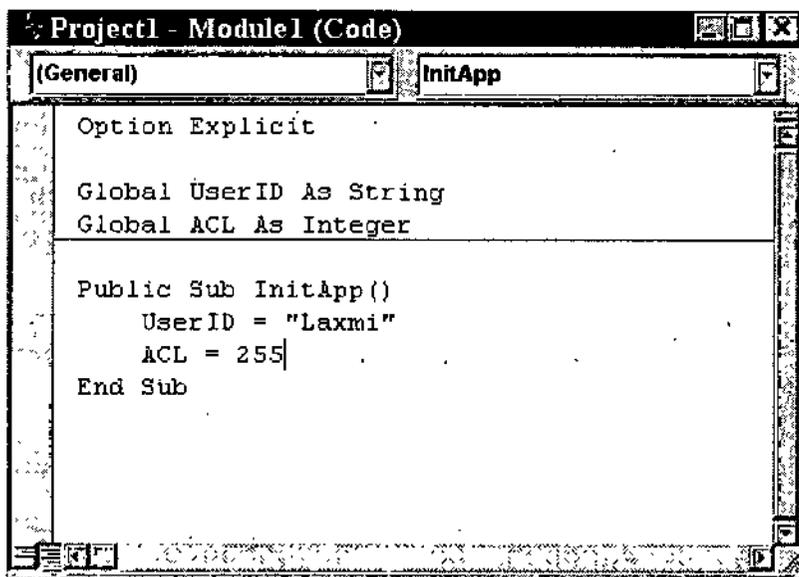
NOTES

4. Select Module from the Add Module dialog box. This will also open the Code window for Module1.
5. In the Code window, add the following statements:  
**Option Explicit**

```
Global UserID As String
Global ACL As Integer
```

6. Add the following procedure to Module1:

```
Public Sub InitApp()
    UserID = "Laxmi"
    ACL = 255
End Sub
```



Above all will create both UserID and ACL as Global, which means that they can be accessed from anywhere within the application. Because the function InitApp() resides in the code module and has the Public keyword in front of it, it becomes a global function that can be called from any procedure within the application. It will be called to initialize the application by setting the UserID variable to Laxmi and ACL to 255.

Now let us create the logon form.

1. Double click Form1 in the Project Explorer to make it the active control in the Form Designer.
2. In the Properties window, set the Name property of Form1 to frmLogon. Set the Caption property to User Logon.
3. Add a Label control to the form. Set its Name to lblUserID; and its Caption to UserID.
4. Add another Label control below lblUserID and set the Name property to lblPassword and the Caption to Password:.

NOTES

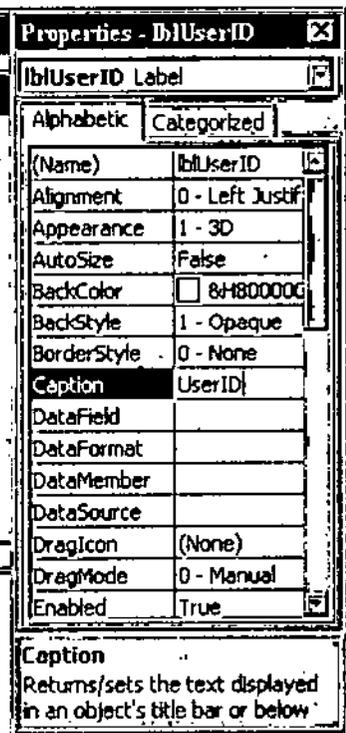
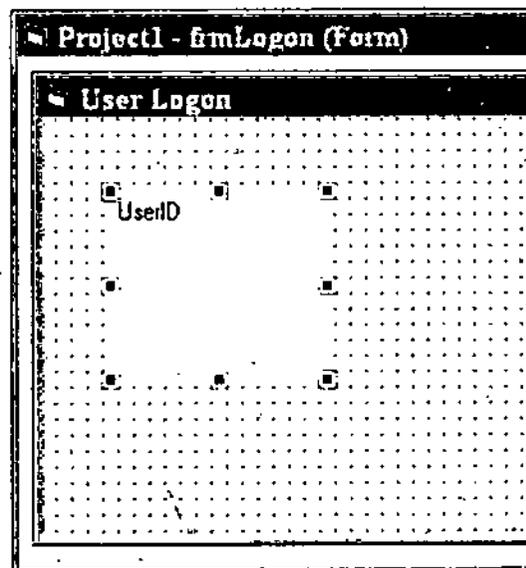
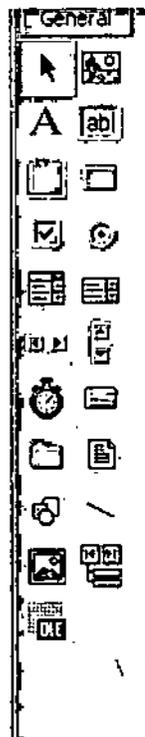
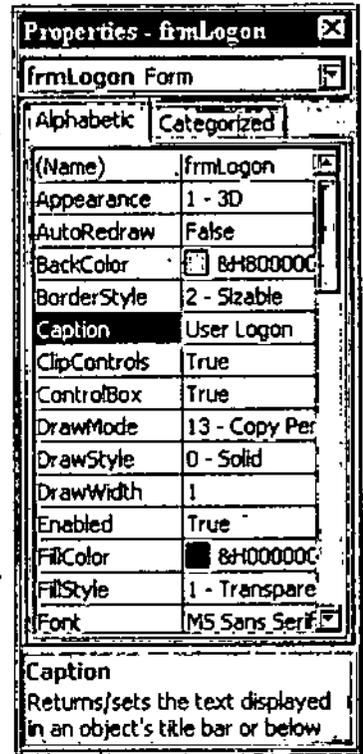
NOTES

5. Add a Text Box control to the right of lblPassword. Set the Name property to txtUserID.
6. Add another Text Box control to the right of lblPassword. In the Properties window set the Name property to txtPassword and the PasswordChar property to the asterisk character (\*).
7. Add a Command Button to the bottom center of the form. Set its Name property to cmdLogon and the Caption to &Logon.
8. Next, double click Form1 to open the Code window.
9. Position the cursor to the left of Option Explicit in the (General) (Declarations) section of Form1.
10. Press the Enter key twice to insert two lines in front of Option Explicit.
11. Type the following line of code above Option Explicit.

```
Private Password As String
```

12. Open the Load() event of the form and add the following code:

```
Private Sub Form_Load()  
    InitApp  
    txtUserID.Text = UserID
```



```
txtPassword.Text = " "
```

```
End Sub
```

13. In the Code window, open the Click() event for cmdLogon and add the following code:

```
Private Sub cmdLogon_Click()
    Dim msg As String

    UserID = txtUserID.Text
    Password = txtPassword.Text

    msg = "UserID: " & UserID & Chr$(13)
    msg = msg & "Password: " & Password & Chr$(13)
    msg = msg & "ACL: " & Str$(ACL)
    MsgBox msg

End Sub
```

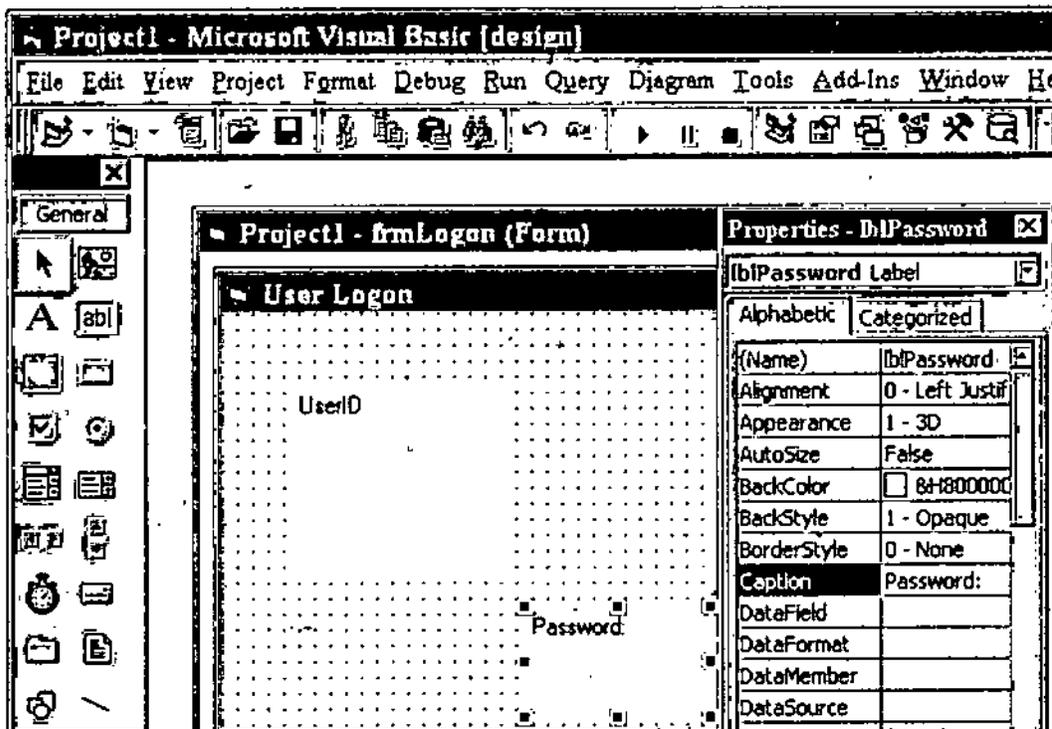
14. Run the project by clicking Run > Start.

Type anything in the password field and click the Logon button.

The code in the Click() event utilizes both module level and global variables to display your user information.

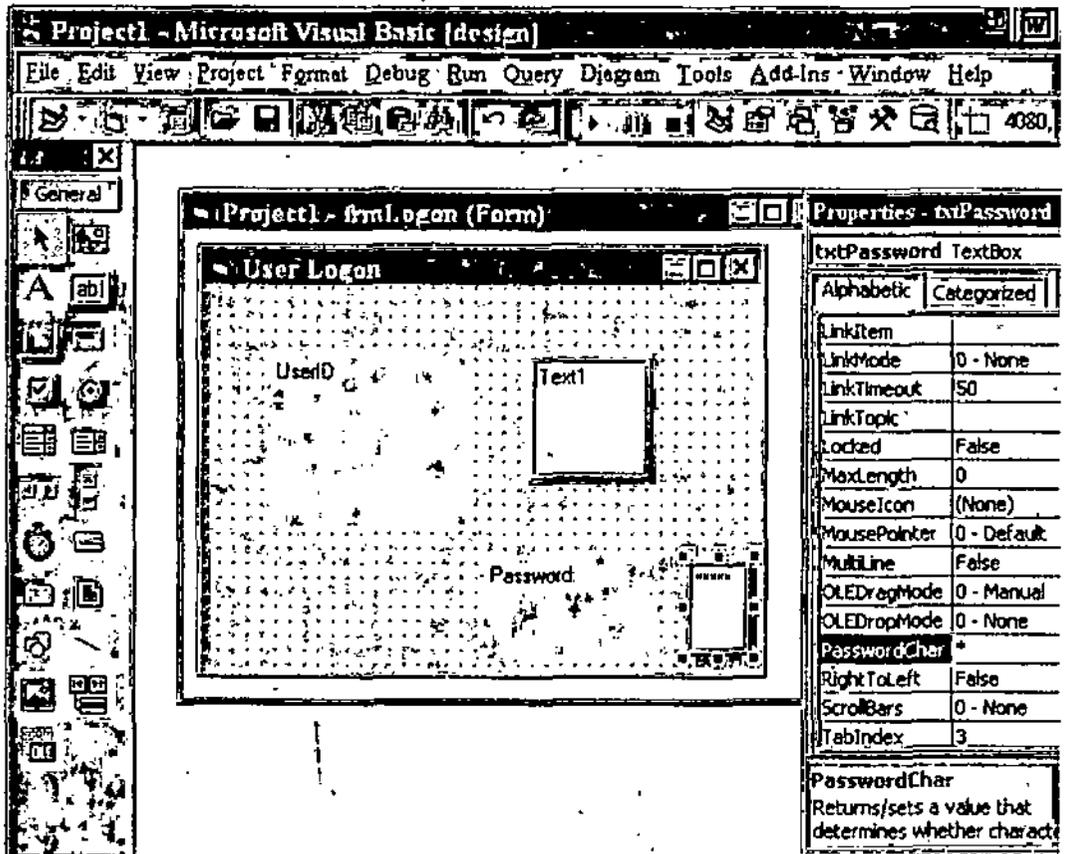
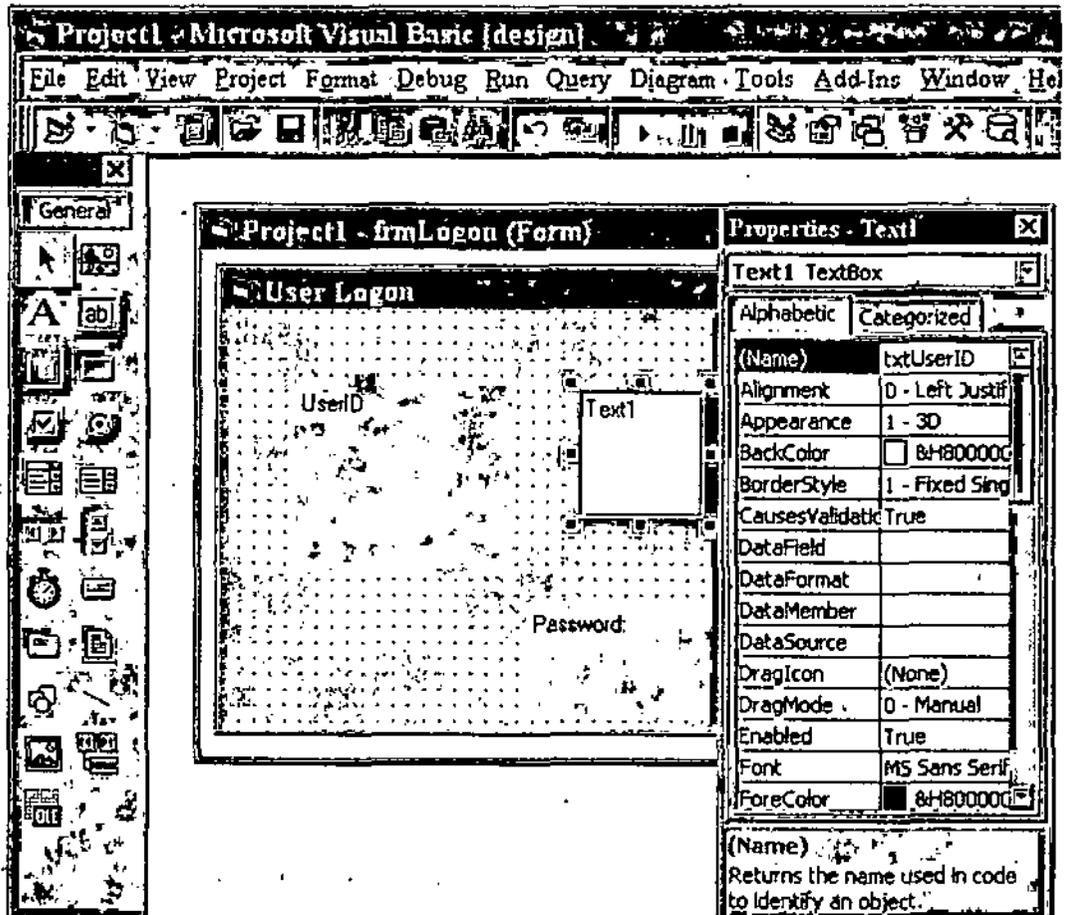
## Logical Operators

Logical operators are those which follow some logic, it could be in the form of less than or greater than or even not equal to. They can be listed as following:

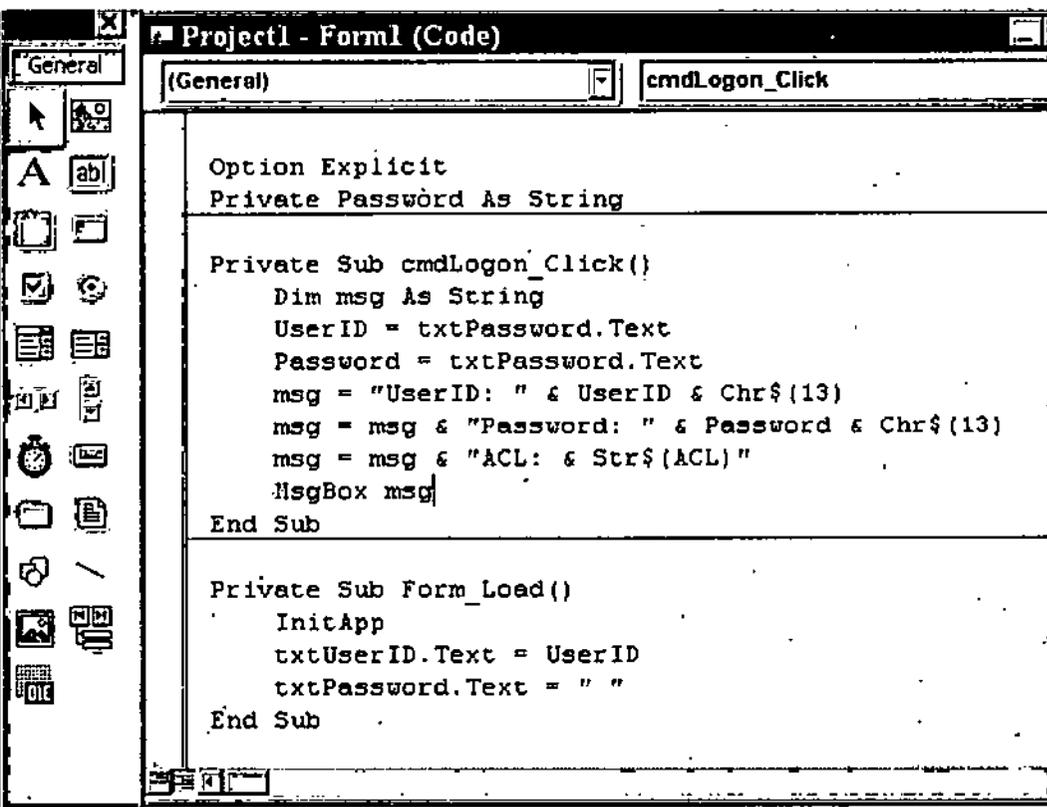
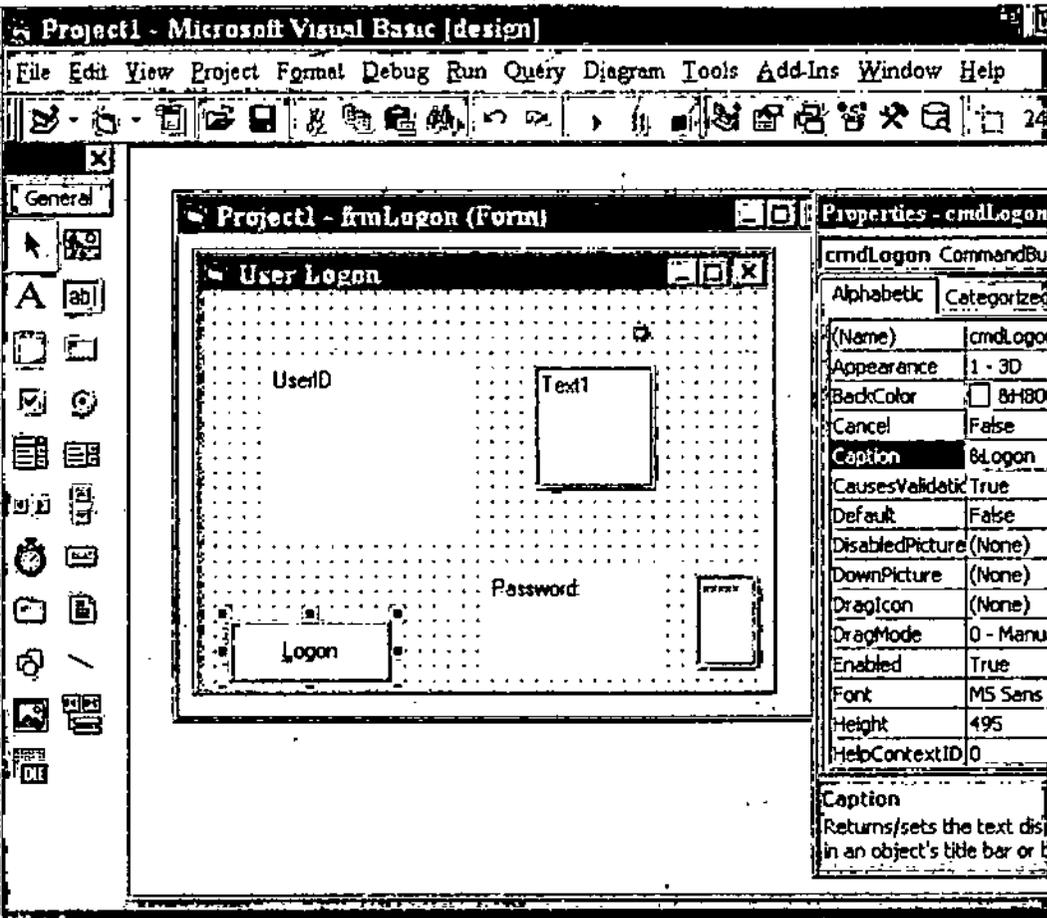


NOTES

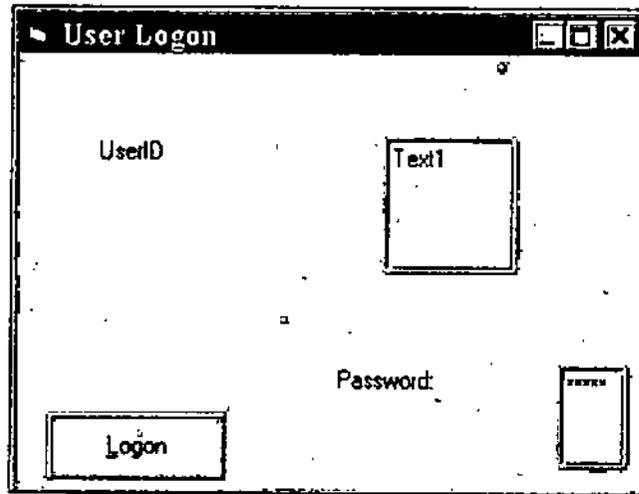
NOTES



NOTES



NOTES



<i>Operator</i>	<i>Description</i>
And	Logical conjunction of two expressions
Eqv	Logical equivalence of two expressions
Imp	Logical implication of two expressions
Not	Logical negation of an expression
OrO	Logical disjunction of two expressions
Xor	Logical exclusion of two expressions

Following are some examples using these expressions to show how they are used.

**And**

The following example would show you the result of the And logical exclusion on two expressions.

For example,

```
Dim X, Y, Z, A, Check
```

This will return the following values depending on the fact that X = 5, Y = 4, Z = 3 and A = Null.

Check = X > Y And Y > Z    Result: True

Check = Y > X And Y > Z    Result: False

Check = X > Y And Y > A    Result: Null

**Eqv**

The following example would show you the result of the Eqv logical exclusion on two expressions.

For example,

```
Dim X, Y, Z, A, Check
```

This will return the following values depending on the fact that X = 5, Y = 4, Z = 3 and A = Null.

Check =  $X > Y \text{ Eqv } Y > Z$  Result: True

Check =  $Y > X \text{ Eqv } Y > Z$  Result: False

Check =  $X > Y \text{ Eqv } Y > A$  Result: Null

### **Imp**

The following example would show you the result of the Imp logical exclusion on two expressions.

For example,

**Dim X, Y, Z, A, Check**

This will return the following values depending on the fact that  $X = 5$ ,  $Y = 4$ ,  $Z = 3$  and  $A = \text{Null}$ .

Check =  $X > Y \text{ Imp } Y > Z$  Result: True

Check =  $X > Y \text{ Imp } Z < Y$  Result: False

Check =  $Y > X \text{ Imp } Z > Y$  Result: True

Check =  $Y > X \text{ Imp } Z > A$  Result: True

Check =  $Z > A \text{ Imp } Y > X$  Result: Null

### **Not**

The following example would show you the result of the Not logical exclusion on two expressions.

For example,

**Dim X, Y, Z, A, Check**

This will return the following values depending on the fact that  $X = 5$ ,  $Y = 4$ ,  $Z = 3$  and  $A = \text{Null}$ .

Check =  $\text{Not}(X > Y)$  Result: False

Check =  $\text{Not}(Y > X)$  Result: True

Check =  $\text{Not}(X > A)$  Result: Null

### **Or**

The following example would show you the result of the Or logical exclusion on two expressions.

For example,

**Dim X, Y, Z, A, Check**

This will return the following values depending on the fact that  $X = 5$ ,  $Y = 4$ ,  $Z = 3$  and  $A = \text{Null}$ .

Check =  $X > Y \text{ Or } Y > Z$  Result: True

Check =  $Y > X \text{ Or } Y > Z$  Result: True

Check =  $X > Y \text{ Or } Y > A$  Result: True

Check =  $Y > A \text{ Or } Y > X$  Result: Null

NOTES

## Xor

The following example would show you the result of the Xor logical exclusion on two expressions.

For example,

```
Dim X, Y, Z, A, Check
```

This will return the following values depending on the fact that X = 5, Y = 4, Z = 3 and A = Null.

```
Check = X > Y Xor Y > Z    Result: False
```

```
Check = Y > X Xor Y > Z    Result: True
```

```
Check = Y > X Xor Z > Y    Result: False
```

```
Check = Y > A Xor X > Y    Result: Null
```

## Logical Operator's Precedence

Like arithmetic operators, logical operators too have a precedence. This is different from arithmetic one. The following is the list of precedence:

<i>Arithmetic</i>	<i>Comparison</i>	<i>Logical</i>
Exponentiation (^)	Equality (=)	Not
Negation (-)	Inequality (<>)	And
Multiplication and division (*, /)	Less than (<)	Or
Integer division (/)	Greater than (>)	Xor
Modulus arithmetic (Mod)	Less than or equal to (<=)	Eqv
Addition and subtraction (+, -)	Greater than or equal to (>=)	Imp
String concatenation (&)	Is	&

## Converting Data Types

Visual Basic provides several conversion functions you can use to convert values into a specific data type. To convert a value to Currency, for example, you use the CCur function:

```
PayPerWeek = CCur(hours * hourlyPay)
```

<i>Conversion</i>	<i>Converts an expression to function</i>
CboolCbool	Boolean
Cbyte	Byte
Ccur	Currency
Cdate	Date
CDbl	Double

NOTES

Cint	Integer
CInt	Long
CSng	Single
CStr	String
Cvar	Variant
CVErr	Error

NOTES

---

### SUMMARY

---

1. Visual Basic, like most programming languages, uses variables for storing values.
2. Constants also store values, but as the name implies, those values remain constant throughout the execution of an application.
3. Data types control the internal storage of data in Visual Basic.
4. An integer is a numeric data type much like a byte data type but it can be signed.
5. Boolean variable gives the values of True or False.
6. Long (long integer) variables are stored as signed 32-bit (4-byte) numbers.
7. Single (single-precision floating-point) variables are stored as IEEE 32-bit (4-byte) floating-point numbers.
8. Double (double-precision floating-point) variables are stored as IEEE 64-bit (8-byte) floating-point numbers.
9. The currency data type is used for monetary values.
10. The string data type is used for non numeric values.
11. The Byte Data type is used for binary data.
12. Date data type is used for storing date.
13. Object variables are stored as 32-bit (4-byte) addresses that refer to objects within an application or within some other application.
14. A Variant variable is capable of storing all system-defined types of data.
15. In Visual Basic, you use variables to temporarily store values during the execution of an application.
16. To declare a variable is to tell the program about it in advance.
17. Variables in Visual Basic require well-formed naming conventions.
18. In a Visual Basic application, global variables should be used only when there is no other convenient way to share data between forms.
19. Values in local variables declared with Static exist the entire time your application is running.
20. Variables declared with Dim exist only as long as the procedure is executing.
21. A module-level variable is available to all the procedures in that module.
22. To make a module-level variable available to other modules, use the Public keyword to declare the variable.
23. DOS did not support graphics.
24. Suffix Visual usually means that the software is for Windows operating system.
25. Modular programming means dividing the program into various modules.
26. In Object Oriented Programming you deal with objects.
27. In event driven programming each application is event-driven, meaning that the flow of program execution is controlled by the events that occur as the program is running.
28. Visual Basic, is the fastest and easiest way to create applications for Microsoft Windows.

NOTES

29. The **Visual** part refers to the method used to create the Graphical User Interface (GUI).
30. The **Basic** part refers to the BASIC (Beginners All-Purpose Symbolic Instruction Code) language, a language used by more programmers than any other language in the history of computing.
31. The Visual Basic Scripting Edition (VBScript) is a widely used scripting language and a subset of the Visual Basic language.
32. In Visual Basic a project is a group of related files, typically all the files required to develop a software component.
33. As you develop an application, you work with a project to manage all the different files that make up the application.
34. As you create, add, or remove editable files from a project, Visual Basic reflects your changes in the Project Explorer window.
35. You can create a new application using Application Wizard in Visual Basic.
36. A command button is a window. Icons, text boxes, option buttons and menu bars are all windows.
37. Visual Basic partially compiles the code as it is entered.
38. Because of the interactive nature of Visual Basic, you'll find yourself running your application frequently as you develop it.
39. Visual Basic uses two main type of dialog boxes, message box and input box.

---

**SELFASSESSMENT QUESTIONS**

---

1. Describe the various types of variables used in Visual Basic.
2. How is DIM statement used?
3. What is a Variant?
4. Describe the naming convention of variables.
5. What is model level variable?
6. What are global variables?
7. What is Visual Basic programming?
8. Describe the following programming methods:  

Modular Programming	Object Oriented Programming
Event Driven Programming.	
9. What do you understand by the concepts of projects in Visual Basic?
10. Describe the various elements of opening screen of Visual Basic.
11. How Windows assist in the working of Visual Basic?
12. What is the use of Project Explorer Window?
13. Describe what do the following do?  

Standard Exe	Active Exe
ActiveX DLL	ActiveX Control
Data Project	IIS Application
Add-In	
14. Describe the various type of files in the project of Visual Basic file.

**Multiple Choice Questions**

1. Variables are used for information to be :  

(a) Stored	(b) Moved	(c) Created
------------	-----------	-------------
2. Values of constants in the program :  

(a) Changes	(b) Remains same	(c) Alters
-------------	------------------	------------
3. An integer can be :  

(a) Unsigned	(b) Both	(c) Signed
--------------	----------	------------

4. Boolean variable gives the value as True and :
  - (a) May be
  - (b) Not True
  - (c) False
5. Single variable stores variable in floating point numbers of :
  - (a) 4 bytes
  - (b) 8 bytes
  - (c) 16 bytes
6. The currency variable is used for :
  - (a) Monetary values
  - (b) Alphabets
  - (c) Numbers
7. Date data type is used for storing :
  - (a) Time
  - (b) Alphabets
  - (c) Date
8. A variable is defined using :
  - (a) Dim
  - (b) Define
  - (c) Light
9. Visual suffixed software are for which operating systems:
  - (a) DOS
  - (b) Windows
  - (c) Both
10. In modular programming you divide program into :
  - (a) Modules
  - (b) Don't divide
11. In Object Oriented Programming you deal with:
  - (a) Objects
  - (b) Modules
  - (c) Events
12. In Event driven programming the program is controlled by the :
  - (a) Modules
  - (b) Objects
  - (c) Events
13. Visual Basic is extension of which language:
  - (a) BASIC
  - (b) COBOL
  - (c) PASCAL
14. In Visual Basic you work on :
  - (a) Projects
  - (b) Degrees
  - (c) Modules
15. In Visual Basic, changes are reflected in:
  - (a) Project Explorer Window
  - (b) Explorer Window
  - (c) Project Window
16. Visual Basic is :
  - (a) Interpreted
  - (b) Compiled
  - (c) Linked
17. A new application can be created using :
  - (a) Application Wizard
  - (b) Visual Wizard
  - (c) Basic Wizard
18. A project file has the extension:
  - (a) vb
  - (b) pp
  - (c) vbp
19. Two type of dialog boxes used in Visual Basic are Message box and :
  - (a) New box
  - (b) Input box
  - (c) In box
20. Buttons are provided for :
  - (a) General controls
  - (b) Pop-menus
  - (c) Nothing

NOTES

### ***True/False Questions***

1. Visual Basic uses variables for storing values.
2. Values of constants change.
3. An integer is a numeric data whose value can be signed.
4. Boolean variable gives the values of True or False.
5. Long (long integer) variables are stored as signed 32-bit (4-byte) numbers.
6. Single (single-precision floating-point) variables are stored as IEEE 32-bit (4-byte) floating-point numbers.
7. The currency data type is used for non-numeric values.
8. The string data type is used for non numeric values.
9. Date data type is used for storing date.
10. A Variant variable is capable of storing all system-defined types of data.
11. To declare a variable is to tell the program about it in advance.

NOTES

12. In a Visual Basic application, global variables should be used only when there is no other convenient way to share data between forms.
13. Variables declared with Dim exist only as long as the procedure is executing.
14. DOS supported graphics.
15. Suffix Visual usually means that the software is for Windows operating system.
16. Modular programming means dividing the program into various objects.
17. Visual Basic, is the fastest and easiest way to create applications for Microsoft Windows.
18. The **Basic** part of Visual Basic refers to the BASIC (Beginners All-Purpose Symbolic Instruction Code) language.
19. The Visual Basic Scripting Edition (VBScript) is a widely used scripting language and a subset of the Visual Basic language.
20. In Visual Basic a project is a group of related files, typically all the files required to develop a software component.
21. As you create, add, or remove editable files from a project, Visual Basic reflects your changes in the Project Explorer window.
22. Visual Basic partially compiles the code as it is entered.

### Short Questions with Answers

1. What are Variables, Constants and Data Types?
 

**Ans.** Visual Basic, like most programming languages, uses **variables** for storing values. Variables have a name (the word you use to refer to the value the variable contains) and a **data type** (which determines the kind of data the variable can store). **Arrays** can be used to store indexed collections of related variables.

**Constants** also store values, but as the name implies, those values remain constant throughout the execution of an application. Using constants can make your code more readable by providing meaningful names instead of numbers. There are a number of built-in constants in Visual Basic, but you can also create your own.

**Data types** control the internal storage of data in Visual Basic. By default, Visual Basic uses the Variant data type. There are a number of other available data types that allow you to optimize your code for speed and size when you don't need the flexibility that Variant provides.
2. What is an Integer?
 

**Ans.** An integer is a numeric data type much like a byte data type but it can be signed, i.e., it may have positive or negative value. It can hold the value of a minimum of -32,768 to a maximum of 32,767.
3. Which are the different ways of using modular programming?
 

**Ans.** There are different ways of using the modular programming. They are:

  1. Sequence structure
  2. Decision/Selection structure
  3. Loop structure
4. What is event driven programming?
 

**Ans.** In event driven programming each application is event-driven, meaning that the flow of program execution is controlled by the events that occur as the program is running. As you will read later most of these events are the direct result of actions initiated by the users; however, some events are invoked by other objects.
5. What does a project in Visual Basic consist of?
 

**Ans.** A project consists of:

  - One project file that keeps track of all the components (.vbp).
  - One file for each form (.frm).
  - One binary data file for each form containing data for properties of controls on the

form (.frm). These files are not editable and are automatically generated for any .frm file that contains binary properties, such as Picture or Icon.

- Optionally, one file for each class module (.cls).
- Optionally, one file for each standard module (.bas).
- Optionally, one or more files containing ActiveX controls (.ocx).
- Optionally, a single resource file (.res).

6. What is an Active X document?

**Ans.** An ActiveX document is a Visual Basic application that utilizes Microsoft Internet Explorer as an application container. ActiveX documents allow you to create portable versions of your application that can be used on laptops, at remote offices, or even from your home. Everything runs from your browser. However, an ActiveX document is not a Web page, it is its own application. In addition, users can navigate between ActiveX documents and Web pages seamlessly through their browser.

7. What are dialog boxes?

**Ans.** Dialog box in Windows environment is used to make a selection, suggest various options, convey messages across, etc. They are there to seek the attention of the user and ask for the response from him. In Visual Basic you have few predefined dialog boxes and then you can create your own too as per your specifications.

Visual Basic uses two main type of dialog boxes, message box and input box. Both of them are readily available in Visual Basic.

8. How would you declare a variable?

**Ans.** To declare a variable is to tell the program about it in advance. You declare a variable with the Dim statement, supplying a name for the variable:

**Dim variablename [As type]**

Variables declared with the Dim statement within a procedure exist only as long as the procedure is executing. When the procedure finishes, the value of the variable disappears.

9. What rules have to followed for writing a variable?

**Ans.** A variable name:

- Must begin with a letter.
- Can't contain an embedded period or embedded type-declaration character.
- Must not exceed 255 characters.
- Must be unique within the same *scope*, which is the range from which the variable can be referenced — a procedure, a form, and so on.

10. What are logical operators?

**Ans.** Logical operators are those which follow some logic, it could be in the form of less than or greater than or even not equal to.

11. How Visual Basic is used in writing DHTML codes?

**Ans.** DHTML, or Dynamic Hypertext Markup Language, brings additional life to ordinary Web pages. Using Visual Basic you can develop your own DHTML applications. DHTML is based on the Document Object Model, which is a hierarchy of Web page elements. A page in DHTML is like a Form object in Visual Basic. You can write a Visual Basic code inside the events of DHTML elements, just like you do in your normal projects.

12. What is a boolean variable?

**Ans.** As mentioned earlier this variable of the type which gives the values of True or False. You can set the return code to True if the function was successful, or to False if the function failed.

13. How does ActiveX support Internet?

**Ans.** ActiveX leverages Internet technology for several reasons:

- It provides a familiar client/server infrastructure to run your applications.

NOTES

- Stand-alone Visual Basic applications can be ported over to ActiveX documents so they can then be downloaded via Internet Explorer.
- You can use this same technology to update ActiveX programs on clients' computers: As the application runs from the browser, the document can be configured to request updates when they are available. Then the control can be automatically installed on the client's computers.
- And, it's all done without disks and without installation programs.

NOTES

**ANSWERS**

**Multiple Choice Questions**

- |       |       |       |       |
|-------|-------|-------|-------|
| 1. a  | 2. b  | 3. b  | 4. c  |
| 5. a  | 6. a  | 7. c  | 8. a  |
| 9. b  | 10. a | 11. a | 12. c |
| 13. a | 14. a | 15. a | 16. b |
| 17. a | 18. c | 19. b | 20. a |

**True False Questions**

- |       |       |       |       |
|-------|-------|-------|-------|
| 1. T  | 2. F  | 3. T  | 4. T  |
| 5. T  | 6. T  | 7. F  | 8. T  |
| 9. T  | 10. T | 11. T | 12. T |
| 13. T | 14. F | 15. T | 16. F |
| 17. T | 18. T | 19. T | 20. T |
| 21. T | 22. T |       |       |

# PROCEDURES AND PROJECTS

---

### LEARNING OBJECTIVES

After going through this chapter, you should appreciate the following:

- Writing Procedures
- VB Program Structure
- Projects
- Forms
- Modules
- Frames
- Project with Multiple Forms
- Displaying Information on Form
- Picture Boxes
- Textboxes

## WRITING PROCEDURES

NOTES

Every professional language provides the facility of breaking the whole program code into smaller set of blocks known as Procedures. These set of blocks contains the code(s) or instruction(s) to perform a particular task. Procedures play vital role when you want to perform such code which is repeating again and again. So you give this repeating code some name, like this later on whenever you call this name you are able to perform or run the commands written inside this repeating code which is nothing but a set of block where you have written all commands.

Using procedures you can break your programs into set of small programs which you can execute again and again by giving a single command. Once the procedure is defined then you can use it in other program too.

Visual Basic supports following procedures:

- Sub Procedure
- Function Procedure
- Property Procedure

### Sub Procedure

Sub Procedures are block of code where you write all commands for execution. You have to give some name to Sub Procedure so whenever you call this name you execute all commands or statements written inside the Sub Procedure. Since using Sub Procedure you break the code into different parts so finding or modifying the code becomes easier.

You can place Sub Procedures in:

- Standard Modules
- Form Modules
- Class Modules

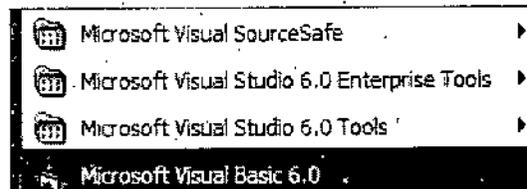
### Syntax

```
[Public| Private][Static]Sub procedurename (arguments)
statements
End Sub
```

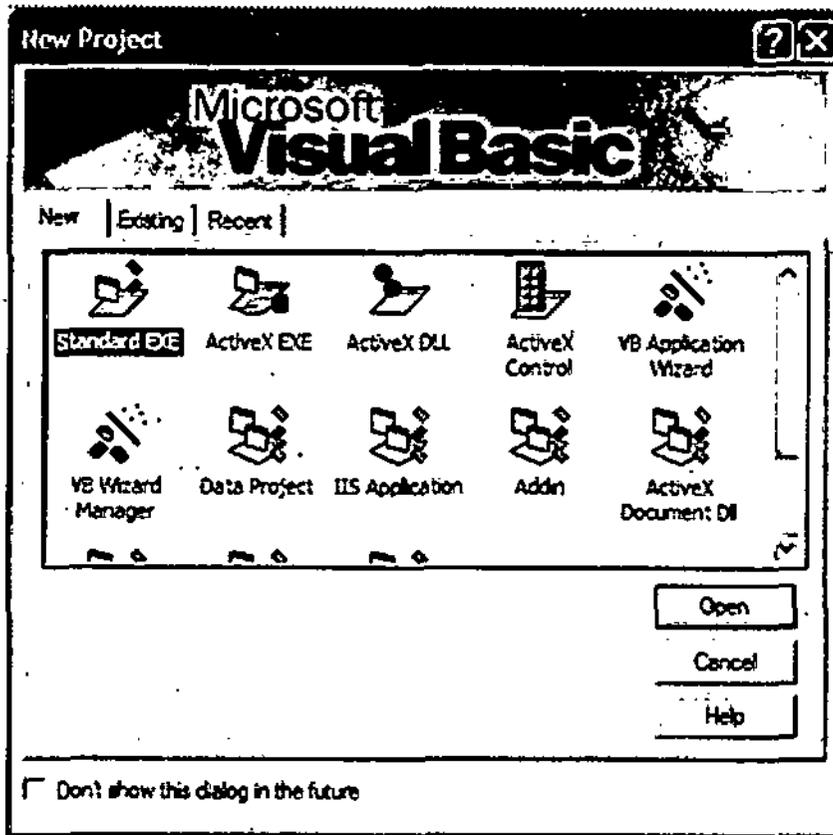
Note: By default Sub Procedures are public in all modules.

### Example: Sub Procedure

1. To start Visual Basic 6, click on start button.
2. Position your mouse over All Programs.
3. Position your mouse over Microsoft Visual Studio 6.0 .
4. Click Microsoft Visual Basic 6.0
5. New Project dialog box appears:
6. Select Standard EXE.

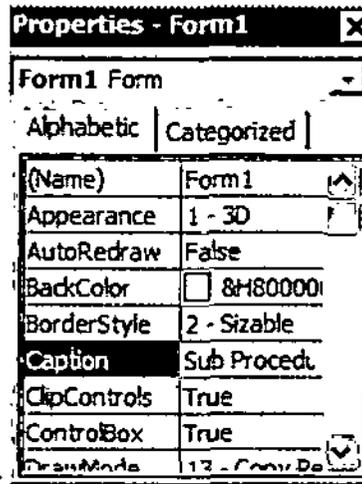


7. Click Open button.
8. Select a Form by single clicking on it.

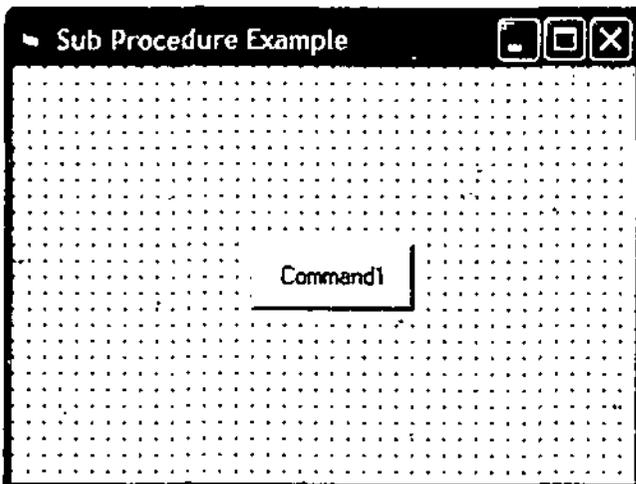


NOTES

9. In Properties Window locate the Caption property and type, Sub Procedure Example.
10. Add a button control to Form.
11. At present your Form looks like this:
12. Select the button control.
13. In Properties window locate the Caption property and type, Sub Procedure Example.

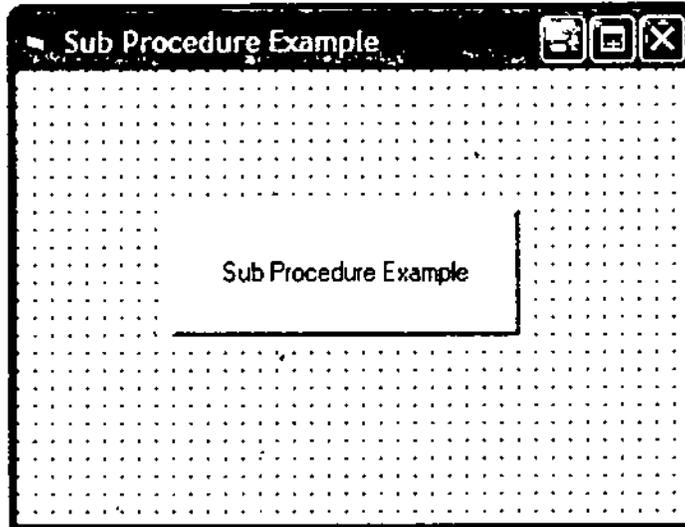


**Caption**  
Returns/sets the text displayed in an object's title bar or below

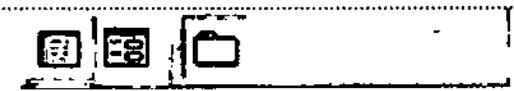
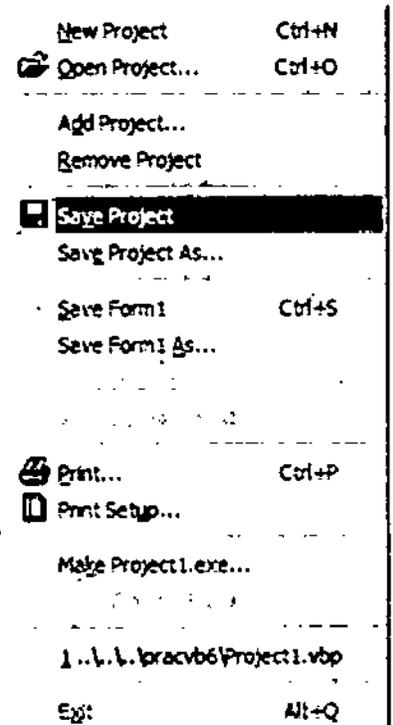


14. Locate the Width property and type 2415.
15. Locate the Height property and type 855.
16. Locate the Left property and type 1080.
17. Locate the Top property and type 960.
18. Now your Form looks like this:

NOTES

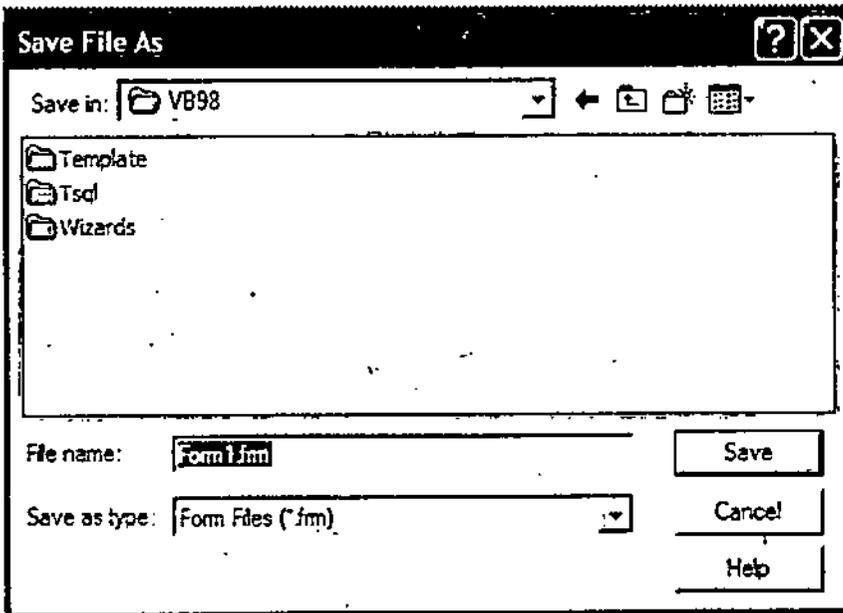


19. Now let us first save the project so on menu bar click on File.
20. Click Save Project.
21. Save File As dialog box appears:
22. In Save in field select C: or any desired drive and folder.
23. Keep all default names and click on Save button.
24. Again Click on Save button to save Project with default name.  
 Note: You can give any other desired name of form or project.
25. Source Code Control dialog box appears:
26. Click No.
27. In Project Window click View Code

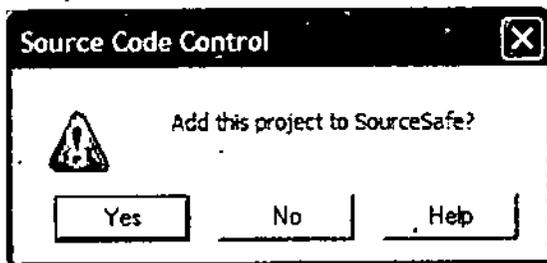


28. Type the following code at top:
 

```
Sub dateshow()
MsgBox ("Todays date is: " & Date)
End Sub
```

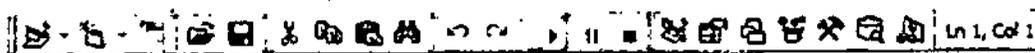


NOTES

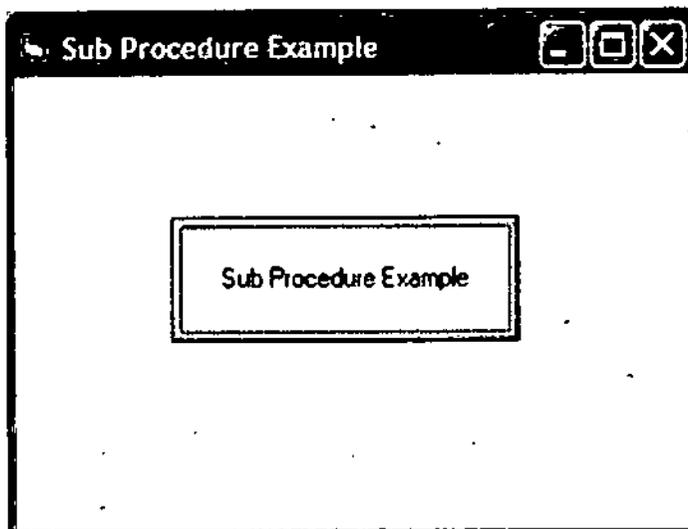


```
Private Sub Command1_Click()  
  dateshow  
End Sub
```

29. To run the program click on Start button.

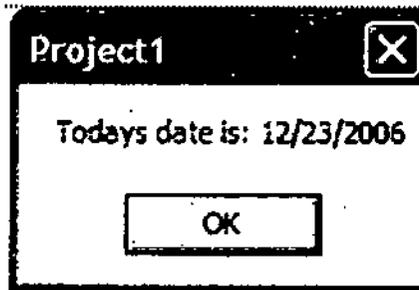


30. You will get following output:



31. Click on button.

NOTES



32. Click OK.

33. Click Close button.

### Event Procedure

Whenever any event occurs then object invokes the event procedure using the name corresponding to event. Generally Event procedures are attached to the forms and controls. An event procedure for a form or control combines the word "Form" or control name and the event name.

For example:

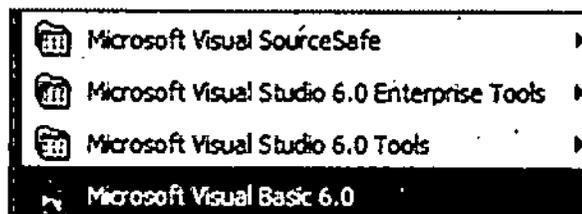
`Button1_Click`

`Form_Click`

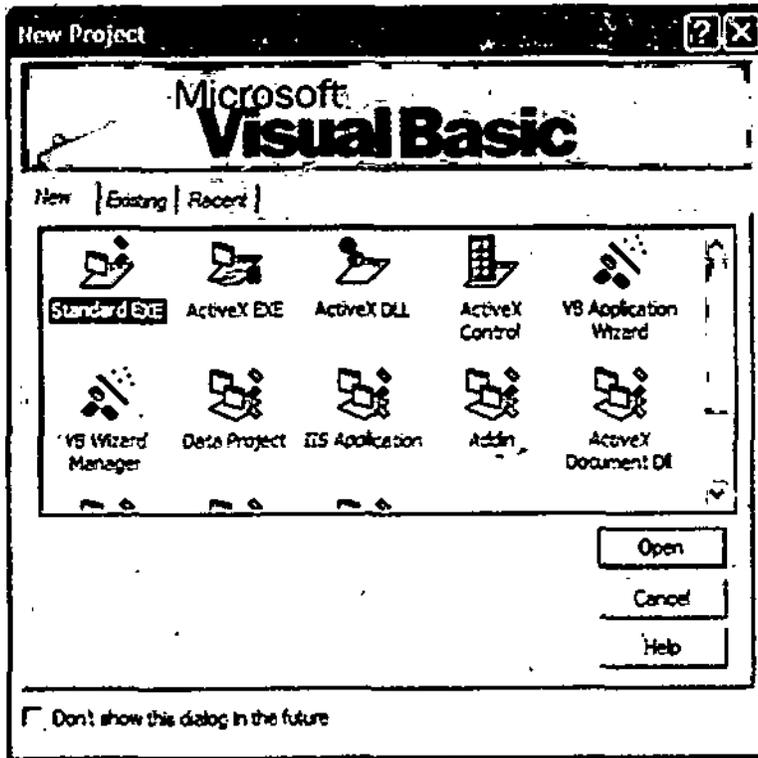
<i>Syntax - Control Event</i>	<i>Syntax - Form Event</i>
Private Sub	Private Sub Form_eventname
controlname_eventname	(arguments)
(arguments)	statements
statements End Sub	End Sub

### Example: Event Procedure

1. To start Visual Basic 6, click on start button.
2. Position your mouse over All Programs.
3. Position your mouse over Microsoft Visual Studio 6.0 .
4. Click Microsoft Visual Basic 6:0

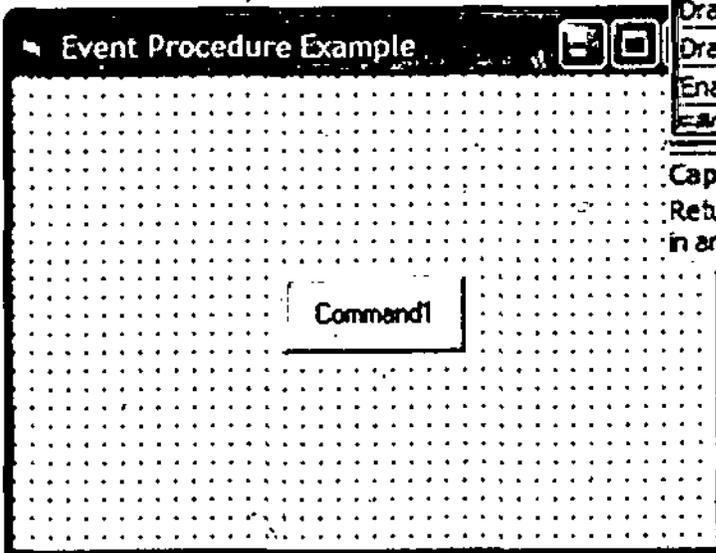
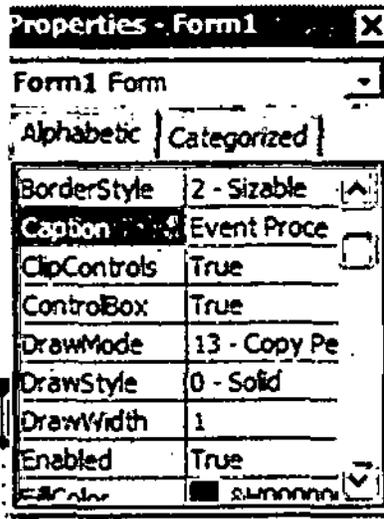


5. New Project dialog box appears:



NOTES

6. Select Standard EXE.
7. Click Open button.
8. Select a Form by single clicking on it.
9. In Properties Window locate the Caption property and type, Event Procedure Example.
10. Add a button control to Form.
11. At present your Form looks like this:

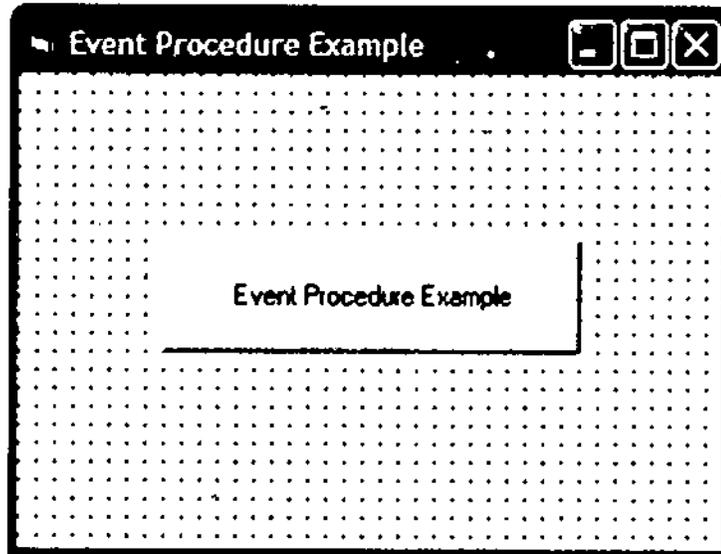


**Caption**  
Returns/sets the text displayed in an object's title bar or below

12. Select the button control.

13. In Properties window locate the Caption property and type, Event Procedure Example
14. Locate the Width property and type 2775.
15. Locate the Height property and type 735.
16. Locate the Left property and type 960.
17. Locate the Top property and type 1080.
18. Now your Form looks like this:

NOTES

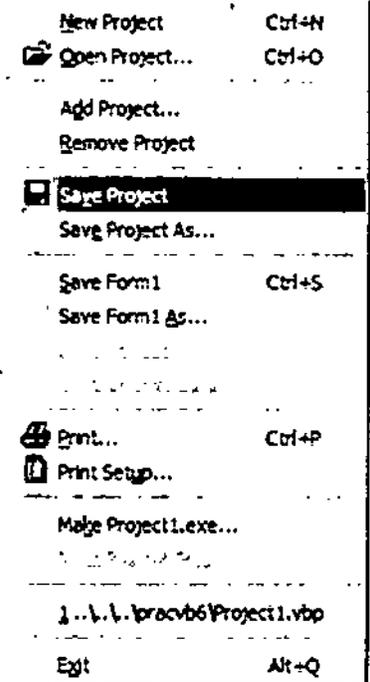


19. Now let we first save the project so on menu bar click on File.
20. Click Save Project.
21. Save File As dialog box appears:
22. In Save in field select C: or any desired drive and folder.
23. Keep all default names and click on Save button.
24. Again Click on Save button to save Project with default name.

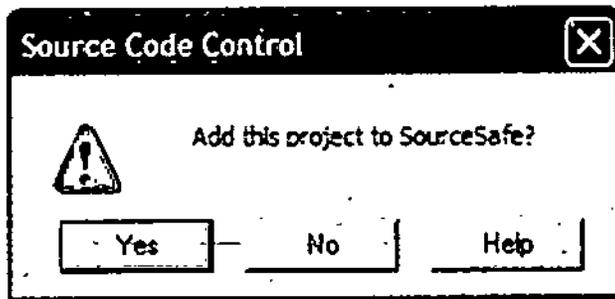
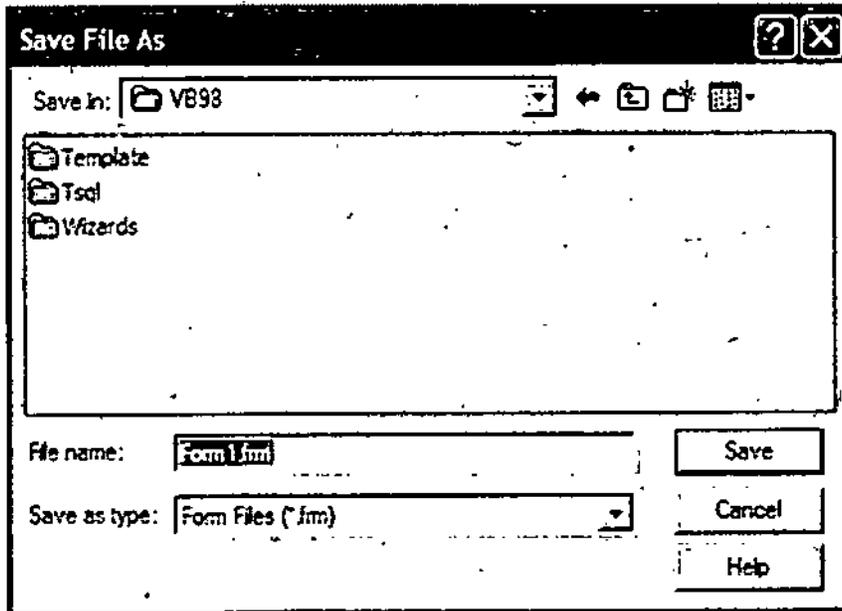
Note: You can give any other desired name of form or project.

25. Source Code Control dialog box appears:
26. Click No.
27. Double-click the command button and type following code:

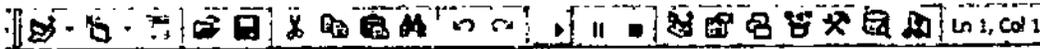
```
Private Sub Command1_Click()
    MsgBox ("Button has been clicked")
End Sub
```



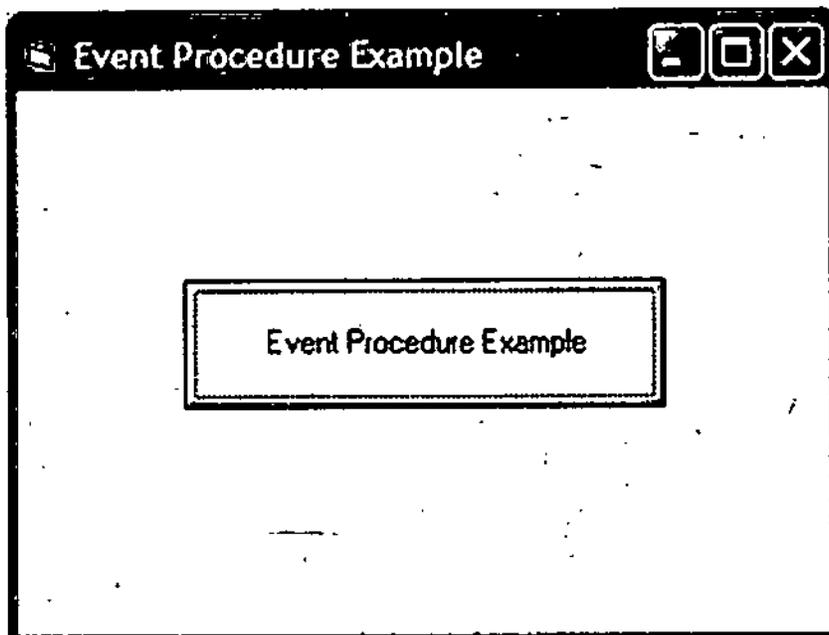
NOTES



28. To run the program click on Start button.



29. You will get the following output:



30. Click on button.

NOTES



31. Click OK.

32. Click close button.

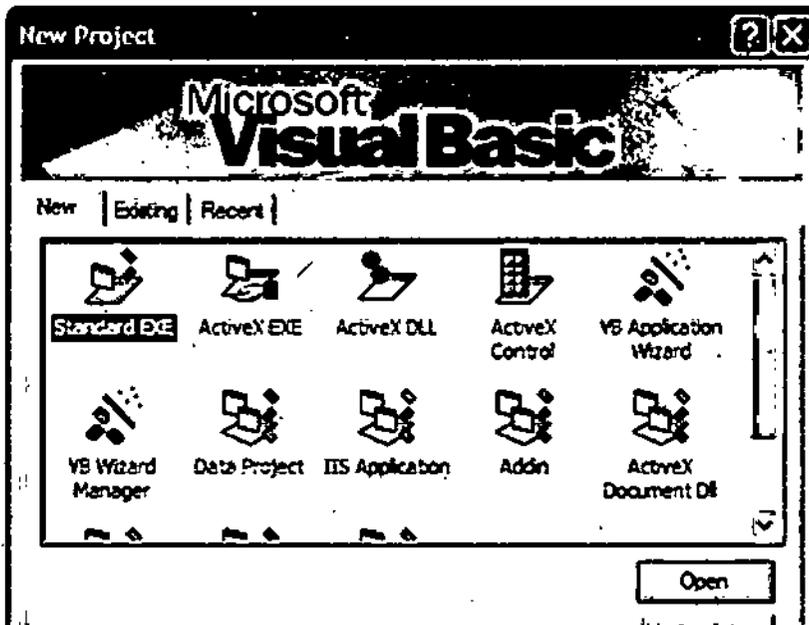
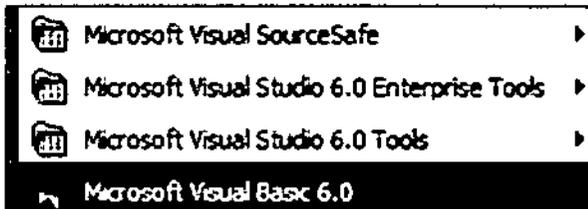
### Function Procedure

Function Procedure returns a value to the calling procedure. Function procedure is also a separate procedure which you can keep in a program as a block of code, once it is defined then function can take the arguments, perform them and can return some specific result according to coding.

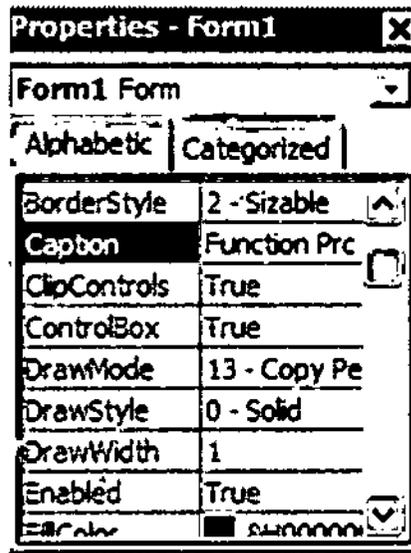
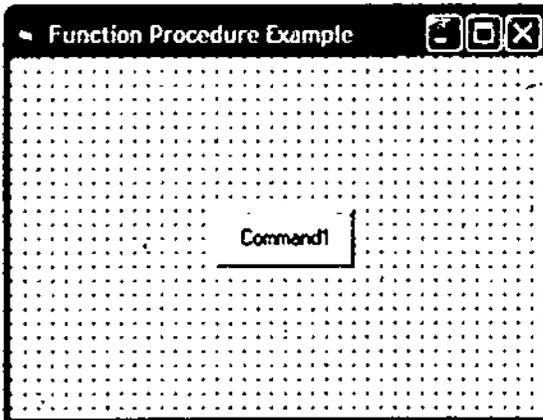
From function you can return a value of specific data type, if data type is not mentioned then the type will be considered as a default variant type.

#### Example: Function Procedure

1. To start Visual Basic 6, click on start button.
2. Position your mouse over All Programs.
3. Position your mouse over Microsoft Visual Studio 6.0.
4. Click Microsoft Visual Basic 6.0
5. New Project dialog box appears:



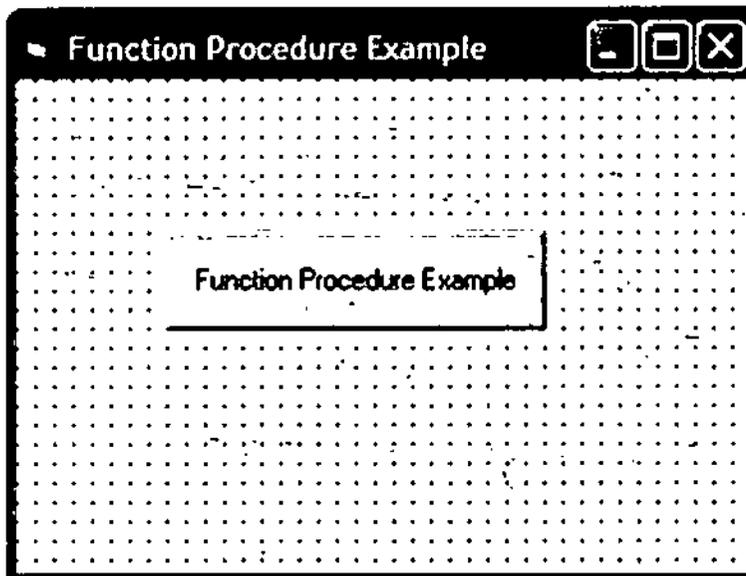
6. Select Standard EXE.
7. Click Open button.
8. Select a Form by single clicking on it.
9. In Properties Window locate the Caption property and type, Function Procedure Example.
10. Add a button control to Form.
11. At present your Form looks like this:



#### Caption

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

12. Select the button control.
13. In Properties window locate the Caption property and type, Function Procedure Example.
14. Locate the Width property and type 2415.
15. Locate the Height property and type 615.
16. Locate the Left property and type 960.
17. Locate the Top property and type 960.
18. Now your Form looks like this:



NOTES



28. Type the following code:

```
Function totalfunction(x As Integer, y As Integer)
```

```
totalfunction = x + y
```

```
End Function
```

```
Private Sub Command1_Click()
```

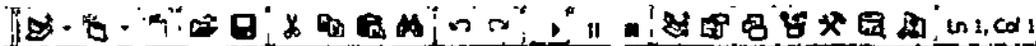
```
Dim rsl As Integer
```

```
rsl = totalfunction(5, 10)
```

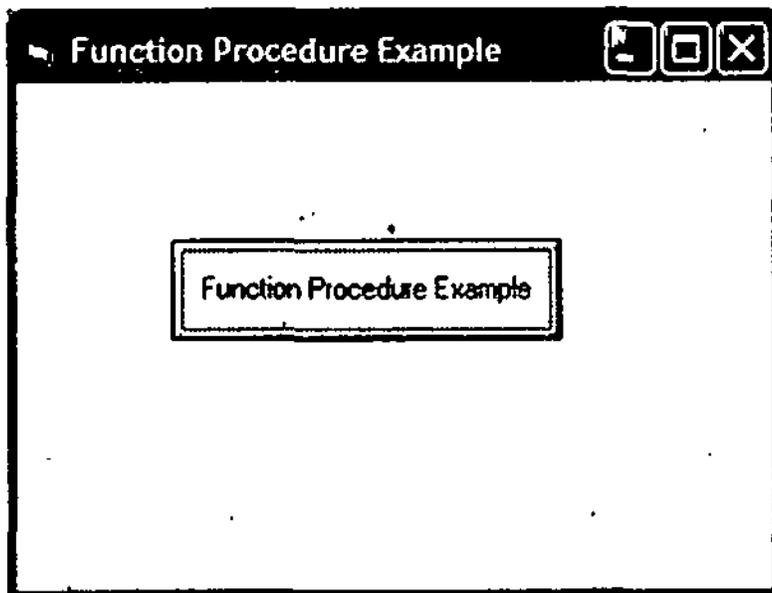
```
MsgBox ("Total of 5 and 10 is:" & rsl)
```

```
End Sub
```

29. To run the program click on Start button.



30. You will get the following output:



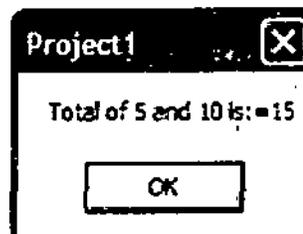
31. Click the button.

32. Click OK.

33. Click Close button.

### Scope

This is related to a lifetime of particular object or variable. Public type are available even the function or procedure



NOTES

executed where as local variable does not. In an application public variables are accessible from anywhere.

So you can say that scope is a range of reference for a variable or object.

### Optional Arguments

Optional means that an argument is not required.

You can use Optional argument in following contexts:

Property Get Statement

Property Let Statement

Sub Statement

Declare Statement

Property Set Statement

Function Statement

NOTES

---

## VB PROGRAM STRUCTURE

---

Visual Basic has a number of controls. Some of them like labels or list boxes, give users feedback, while others, like command buttons and text boxes, elicit responses. Other controls sit quietly, invisible to the user and perform some of the grunt work that makes your application useful. The timer control is one example of an invisible control. Controls are easy to use and when used properly, can add significant functionality to your programs. To add a control to a form you simply double-click the control you want to add, or you can "draw" the controls on a form by clicking the control and then dragging the mouse around the area on the form where you want the control to be. After you add some controls, you can set most of their properties in the Properties window. You simply click the control to make it active and change the appropriate properties in the Properties window.

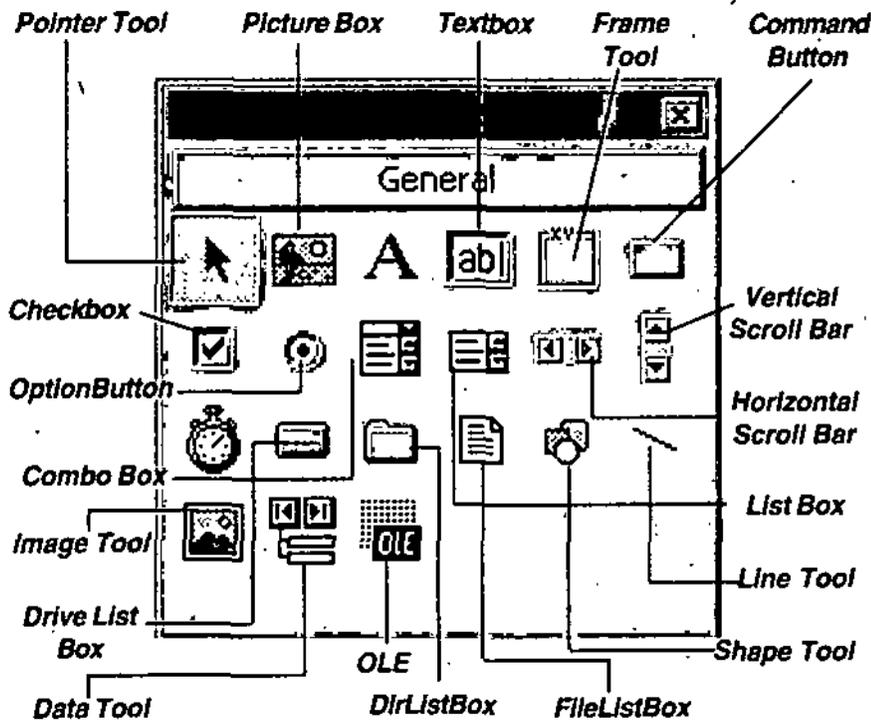
The Toolbox is the containing window that holds the custom controls for your applications. In this chapter, we are going to take a look at some of the basic controls you can use to get your applications up and running.

There are also several advanced controls that come with Visual Basic 6 many of which have been enhanced since the previous version. Some controls work with multimedia, such as the Multimedia control and others Winsock and Internet Transfer controls – utilize the Internet. There are also several new data-aware controls that help you work with database. In addition, you can develop your own ActiveX custom controls and add them to your Toolbox or you can distribute your custom controls to other developers through a variety of methods.

Let us learn about these buttons one by one.

### Using Command Buttons

This button is one of the most common one. You can use a command button to elicit simple responses from the user or to invoke special functions on forms. Every time you click the OK button on a dialog box you click a command button. Let us learn more about its properties, methods and events attached to it.



NOTES

## Properties

Various properties of the command button are given below. More information about these properties can be obtained from the Properties window.

Appearance	BackColor	Cancel
Caption	CausesValidation	Container
Default	DisabledPicture	DownPicture
DragIcon	DragMode	Enabled
Font	FontBold	FontItalic
FontName	FontSize	FontStrikethru
FontUnderline	ForeColor	Height
HelpContextID	Hwnd	Index
Index	Left	MaskColor
MouseIcon	MousePointer	Name
OLEDropMode	Parent	Picture
RightToLeft	Style	TabIndex
TabStop	Tag	ToolTipText
Top	UseMaskColor	Value
Visible	WhatsThisHelpID	Width

The two most important properties of command buttons are Name and Caption. The Name property is used to give the control its own identity. This name is used by your code and Visual Basic to distinguish it from the rest of the controls. The Caption

## NOTES

property determines the text that appears on the command button. Placing an ampersand character (&) in the caption gives a keyboard-across key alternative (called a hotkey) to a mouse-click. You access these controls by holding the Alt key down while you press the underlined letter of the control you wish to access. The user could also tab to the command button and press the spacebar to simulate a mouse-click on the button. Two other useful properties are Cancel and Default. Setting the Default property to True means the user can simulate a click on the button by pressing Enter. Setting Cancel to True means the user can close a form by pressing Esc.

By setting the Style property, you can make the button contain text only or you can add a picture to the button. If you want the button in its normal, unpressed state to have a picture, you can specify the picture's filename in the Picture property. You can also place other graphics on the button by setting them using the DisabledPicture and DownPicture properties.

Two properties can stop the user from accessing a command button Enabled and Visible. If either is set to False, the command button will be unusable. Disabling a command button is a handy technique if you want to force the end user to complete certain actions (such as filling in text boxes) before clicking the next button in the process.

If the user moves around the form with the Tab key, you can determine the order in which they visit controls by specifying the TabIndex property. A control with a TabIndex of 0 (zero) is the first to receive the focus on a form, provided it's not disabled or invisible. If you alter the TabIndex of one control, the other controls' order adjust to accommodate the new order. When you want to prevent a user from tabbing to a control, set its TabStop property to False. This does not prevent a mouse-click on a control - to stop that, use the Enabled or Visible properties described previously.

### Events

The most frequently coded event procedure for a command button corresponds to the Click() event; the other events for a command button are listed here:

Click	DragDrop	DragOver
GotFocus	KeyDown	KeyPress
KeyUp	LostFocus	MouseDown
MouseMove	MouseUp	OLECompleteDrag
OLEDragDrop	OLEDragOver	OLEGiveFeedback
OLESetData	OLEStartDrag	

In your first programs, the Click() event is probably the only event in which you'd be interested. It's the most commonly used event on a command button. You won't use many of the other events until you become more proficient in Visual Basic. However, you can also use the MouseUp() even in place of the Click() event. Many Windows 95/98 applications use this event because it gives the user a chance to back out without firing a Click() event.

### Methods

Listed Below are the methods for the command button. The most commonly used method is Set Focus.



## NOTES

choice) when the user is entering a password. MaxLength and PasswordChar properties are often employed for a text box on a logon form.

The MultiLine property lets the user type more than one line of text into the text box. If MultiLine is used with the ScrollBars property, you can make a simple text editor with no coding—though you would need a couple of code to save the user's typing.

The SelLength, SelStart and SelText properties are useful for dealing with text appropriately. For example, the SelText property returns the text in the text box that the user selected with the mouse or arrow keys. From there it's easy to copy or cut the selected text to the Clipboard.

Note that the ReadOnly property from previous versions has been replaced by the Locked property. Setting the Locked property to True will cause the text box to display data, but will permit no editing. You may have noticed this type of text box on license agreement dialog boxes that appear during program installations. You can select and copy text, but you cannot type or delete text from the box.

To change the order in which the user tabs around the text boxes (and other controls) on a form, change the TabIndex setting. If you don't want the user to tab into a text box, set its TabStop property to False. To prevent a user from clicking in the text box with the mouse, set the Enabled property to False. There may be some situations where you would want to prevent the user from accessing a text box. For example, the user is not allowed to enter a message in an e-mail program until an address has been entered in the address text box. You will discover other examples of why you would want to sue this feature as you become more proficient with Visual Basic.

### Events

The text box control supports a few events that are listed in the table here:

Change	KeyDown	LinkOpen	OLEDragDrop
Click	KeyPress	LostFocus	OLEDragOver
DblClick	KeyUp	MouseDown	OLEGiveFeedback
DragDrop	LinkClose	MouseMove	OLESetData
DragOver	LinkError	MouseMove	OLEStartDrag
GotFocus	LinkNotify	OLECompleteDrag	Validate

The Change() event occurs every time the user inserts, replaces or deletes a character in a text box. You can perform some elementary validation of user entry in the Change() event. You can even use it to restrict entry to certain characters only. However, you may find that the Masked Edit control or one of the Key events is more suitable for this purpose. The Microsoft Masked Edit control lets you specify entry templates or masks. It's a custom control that you need to add to the Toolbox if you want to use it. There's a full reference for all the custom control bundled with Visual Basic in the Microsoft Developer Network, included on your Visual Basic CD.

### Methods

Here is the list of methods supported by the text box control:

Drag	LinkRequest	OLEDrag	ShowWhatsThis
------	-------------	---------	---------------

LinkExecutive	LinkSend	Refresh	ZOrder
LinkPoke	Move	SetFocus	

Most of the methods here are not used very frequently, though the Link methods are necessary if your text box is involved in a DDE (Dynamic Data Exchange) conversation. DDE allows one application to communicate with another. As the user interacts with one application, it sends data automatically to the other application. Unfortunately, it is beyond the scope of this book to cover DDE in detail. If you are interested in pursuing it further, check the online help.

NOTES

The SetFocus method, though is a boon in data-entry applications. When the user clicks a command button (say, an Update button) the focus remains on that button. If the last statement in the Click() event for the command button is a SetFocus method, you can force the focus back to the data-entry text boxes. This saves the user from an extra mouse-click or excessive use of the Tab key just to get back into position. The syntax is:

```
txtMyTextBox.SetFocus
```

## Using Labels

A label control is similar to a text box control in that both display text. The main difference however is that a label displays read-only text as far as the user is concerned though you can alter the caption as a run-time property.

The property of interest in a label control is the Caption property, as opposed to a text box's Text property. Labels are often used for providing information to the user. This can be in the form of a message shown on the form itself or a prompt. The prompt is usually combined with a text box, list box, or other control. It gives the user an idea of the nature of the referenced control. For example, if you had a text box that allowed the user to enter the data for a customer's name, then you might place a label to the left or above the text box with its Caption set to Customer Name.

## Properties

You have already worked some of the properties of a label control. Here is the complete list of properties for this control:

Alignment	DataSource	Height	Parent
Appearance	DragIcon	Index	RightToLeft
AutoSize	DragMode	Left	TabIndex
BackColor	Enabled	LinkItem	Tag
BackStyle	Font	LinkMode	ToolTipText
BorderStyle	FontBold	LinkNotify	Top
Caption	FontItalic	LinkTimeout	UseMnemonic
Container	FontName	LinkTopic	Visible
DataChanged	FontSize	MouseIcon	WhatsThisHelpID
DataField	FontStrikethru	MousePointer	Width
DataFormat	FontUnderline	Name	WorldWrap
DataMember	ForeColor	OLEDropMode	

NOTES

As a remainder, the most important property at the outset is – once again the Name property. For labels the prefix is normally lbl. The Caption property determines the text shown in the label. If you incorporate an ampersand (&) character in the caption, an access key is defined. This raises an interesting question: What happens if you want to show an actual ampersand in the caption? An ampersand does not display; instead, it remains hidden and causes the subsequent character to appear underlined.

You define the size of the label at design time. At run time you might wish to alter the Caption property, only to find it's too big to fit within the label control. You could calculate the length of the caption and adjust the label size accordingly, but this is messy and there's danger of an enlarged label obscuring other controls. To simplify matters, use the AutoSize and WordWrap properties, either by themselves or in conjunction. That way the caption will fit and you can control whether the label expands vertically rather than horizontally.

One more interesting label control property is BorderStyle. This is not related to the form property of the same name – there are only two choices. But by setting BorderStyle to 1 – Fixed Single and the BackColor to white (or whatever), the label looks exactly like a text box, except that it's read-only. Labels were often used in this fashion in prior versions of Visual Basic to show data for browsing purposes only.

**Events**

The label control has many of the same events as any other controls:

Change	LinkClose	MouseMove	OLEGiveFeedback
Click	LinkError	MouseUp	OLESetData
DblClick	LinkNotify	OLECompleteDrag	OLEStartDrag
DragDrop	LinkOpen	OLEDragDrop	
DragOver	MouseDown	OLEDragOver	

Most of the standard events are supported. But note the absence of any Key() events. This is consistent with a label not being able to receive the focus. The Mouse() events are there, because there's nothing to stop you from clicking a label at run time. The ability to click a control does not indicate it must necessarily receive the focus. The Link() events are not shared by many other controls (with the exception of text boxes and picture controls). These events are concerned with DDE (Dynamic Data Exchange) conversations.

**Methods**

The label control also has methods, but you will probably not use them very often in your applications. The following shows the methods supported by the label control:

Drag	LinkRequest	OLEDrag	Zorder
LinkExecute	LinkSend	Refresh	
LinkPoke	Move	ShowWhatThis	

The label methods are not particularly useful, although the LinkRequest method is sometimes used to update a nonautomatic DDE link.

## Using Option Buttons

Option button controls, also called radio buttons are used to allow the user to select one and only one, option from a group of options. Usually option buttons are grouped together within a frame control but they can also be grouped on a plain form, if there is to be only one group of option buttons. Thus, if you had a frame specifying a delivery method, you might have one button for UPS (United Parcel Service) and another four Courier delivery. Products can only be shipped by one of these methods (not both-and not more). In contrast, option buttons representing, say, bold and italic settings for text would not make sense. Text can be both bold and italic or neither (none).

NOTES

## Properties

The option button supports many properties, are shown in the table below:

Alignment	FontSize	Picture
Appearance	FontStrikethru	RightToLeft
BackColor	FontColor	TabIndex
CausesValidation	Height	TabStop
Container	HelpContextID	Tag
DisabledPicture	HWND	ToolTipText
DownPicture	Index	Top
DragIcon	Left	UseMaskColor
DragMode	MaskColor	Value
Enabled	MouseIcon	Visible
Font	MousePointer	WhatsThisHelpID
FontBold	Name	Width
FontItalic	OLEDropMode	
FontName	Parent	

Once again the Name property is the one to set first; option buttons have an opt prefix by convention. The Caption property helps the user determine the purpose of an action button. The other popular property is Value. This is invaluable at both design time and run time. At run time you test the Value property to see if the user has turned on (or off) the option button. The property has two settings, true and False. At design time you can set the Value property to True for one of the buttons if you wish – the default setting is False. This means that the option button (and only that option in a group) is pre-selected when the form opens. If you try to make Value for another button in the group True, then the previous the previous one reverts to a False setting. A new property added in version 6 is the Style property. The default setting of 0 – Standard will draw a normal option button, as shown at the beginning of this section. However, by setting this property to 1 – Graphical, you can make the option button look just like a command button, but it allows only one selection to be made within a group. It works much like the old memory preset buttons on the old stock car radios.

## Events

The option button control has a few events, but only the Click() event is really used:

NOTES

Click	KeyDown	MouseMove	
OLEGiveFeedback			
DbIClick	KeyPress	MouseUp	OLESetData
DragDrop	KeyUp	OLECompleteDrag	OLEStartDrag
DragOver	LostFocus	OLEDragDrop	Validate
GotFocus	MouseDown	OLEDragOver	

The typical way of dealing with option buttons is to test the Value property at run time to see if they're selected. Your code then initiates actions accordingly. It's common to test for the Value property in the Click() event procedure for a command button that's clicked after the user has selected the option button of interest. This allows you to check for a condition before the next procedure is called. You test the Value property in an If ...End If or Select Case...End Select construct. But there may be occasions when you want to initiate an action immediately after the user makes a choice. Then you may want to trap the option button's Click() event.

## Methods

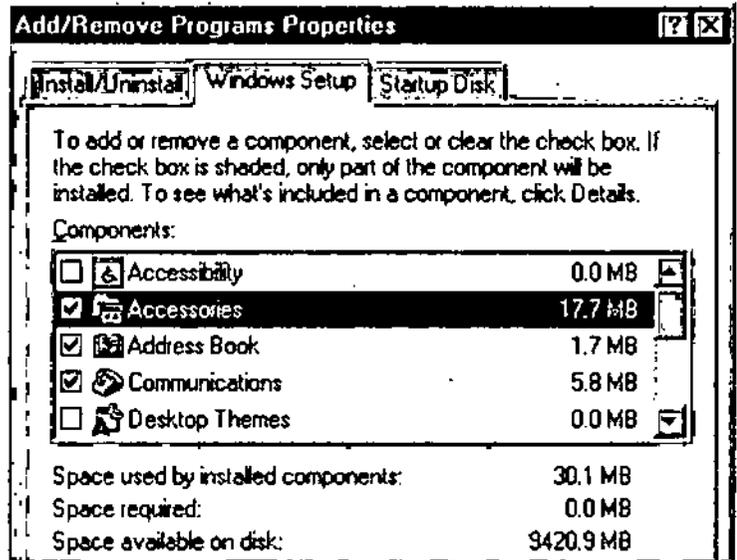
The methods for the option button are of little use in the Visual Basic environment:

Drag	OLEDrag	SetFocus	Zorder
Move	Refresh	ShowWhatsThis	

Open and start the Controls sample application. Click the Option Buttons button to bring up the Options dialog box. The five option buttons are actually in two groups. The option labeled 486, Pentium and Pentium Pro are in their own group directly on the form. The Windows 95/98 and Windows NT options are in a separate group on the Operating System frame. This frame then rests on the form and separates the two groups of option buttons. If you click an option button, you will notice the other option buttons in the same group become deselected. Only one option can be selected in any particular group at one time.

## Using Check Boxes

A check box control is rather similar to an option button, which was described in the last section. Both often partake in group and the Value property is tested to see if a check box is on or off. But there are two fundamental differences between check boxes and option buttons: Check boxes are valid as single controls –



a single option button is probably counter-intuitive. Check boxes (even when in a group) are not mutually exclusive. Finally, check boxes have three possible settings for the Value property.

An option button is either on or it's off. Therefore, Value can be either True or False. Check boxes can be in one of three states on, or grayed. Grayed check box is neither on nor off, though the user can change its setting. If the check box were disabled the user wouldn't be able to turn it on or off. A grayed check box are selected. If you have installed Windows 95/98, then you understand what are grayed check box means.

NOTES

## Properties

The following table lists the properties for the check box control:

Alignment	DownPicture	Height	RightToLeft
Appearance	DragIcon	HelpContextID	Style
BackColor	DragMode	hWnd	TabIndex
Caption	Enabled	Index	TabStop
CausesValidation	Font	Left	Tag
Container	FontBold	MaskColor	ToolTipText
DataChanged	FontItalic	MouseIcon	Top
DataField	FontSize	Name	Value
DataMember	FontStrikethru	OLEDropMode	Visible
DataSource	FontUnderline	Parent	WhatsThisHelpID
DisabledPicture	ForeColor	Picture	Width

Again, as with option buttons, the three most popular properties are Name, Caption and Value. When setting the Name property it's conventional to use a chk prefix.

## Events

The following table shows you that the check box has similar events to the option button control:

Click	KeyPress	MouseUp	OLESetData
DragDrop	KeyUp	OLECompleteDrag	OLEStartDrag
DragOver	LostFocus	OLEDragDrop	Validate
GotFocus	MouseDown	OLEDragOver	
KeyDown	MouseMove	OLEGiveFeedback	

To carry out further processing as soon as the user has clicked the check box, use the Click() event. Normally, though, if you do not put code directly within the click() event, you will have another procedure that will ascertain the value of a check box by retrieving its Value property.

## Methods

Like the events of the check box control, the methods are similar to those for the option button control:

Drag	OLEDrag	SetFocus	ZOrder
Move	Refresh	ShowWhatsThis	

## Using Frame Controls

NOTES

When used by itself, the frame control is not particularly useful. The controls normally placed in a frame are option buttons and check boxes. This has the effect of grouping them together so that when the frame is moved, the other controls move too. For this to work you can't double-click a control (say, an option button) to add it to the form and then drag it into position within the frame. Instead, you must single-click the control in Toolbox and drag a location for it inside the frame. Then all the controls move together. In addition, the option buttons function as a group – that is, if you select one at run time, the others become deselected. If you simply scatter option buttons randomly on a form, then they all function as one large group. To create separate groupings of options buttons, you place them in frames. The button items within each frame act as a self-contained group and have no effect on the option buttons in other frame groups.

Although a frame is often used as a container for check box groups too, each check box is completely independent. Thus the setting for one check box has no effect on the setting for the others in the same group. This is the behavior you would expect of check boxes. Check boxes are not mutually exclusive. This contrasts with option buttons, where the buttons within a single group should be mutually exclusive. The reasons then for placing check boxes in a frame is to enable you to move the group as a whole, when you reposition the frame at design time. The frame also serves as a visual grouping for the check boxes. For example, the check boxes relating to a particular feature can be in one frame and those pertinent to another feature in another frame.

A frame is usually given the prefix fra. You place the frame on the form before you place the controls it's going to contain.

## Properties

The frame control has several properties, listed below:

Appearance	Enabled	Height	Parent
BackColor	Font	HelpContextID	RightToLeft
BorderStyle	FontBold	hWnd	TabIndex
Caption	FontItalic	Index	Tag
Container	FontName	Left	ToolTipNext
ClipControls	FontSize	MouseIcon	Top
Container	FontStrikethru	MousePointer	Visible
DragIcon	FontUnderline	Name	WhatsThisHelpID
DragMode	FreeColor	OLEDropMode	Width

After the Name property, perhaps the single most important property is Caption. You use this to give a meaningful title to the frame on the form. Then it's clear to the end user which feature the option buttons (or check boxes) in the frame refer to. To provide a clue as to how each option button affects the features, you use the Caption

property of the buttons. For example, in an order dispatch system you might have a frame with the caption Delivery. And within that frame you might have two option buttons, with the captions Normal and Express.

## Events

The frame control supports a few events:

Click	MouseMove	OLEGiveFeedBack
Db1Click	MouseUp	OLESetData
DragDrop	OLECompleteDrag	OLEStartDrag
DragOver	OLEDragDrop	
MouseDown	OLEDragOver	

The frame control events are only rarely used. In an application that use drag-and-drop, however, the DragDrop() event is sometimes used to initiate actions when the user drops an object into a frame area.

## Methods

A frame object supports only a few methods. None are very helpful and they're hardly ever seen in Visual Basic projects:

Drag	OLEDrag	ShowWhatsThis
Move	Refresh	Zorder

## Using List Boxes

If you're a regular user of Windows, then you're familiar with list box controls. A list box is an ideal way of representing users with a list of data. Users can browse the data in the list box or select one or more items as the basis for further processing. The user can't edit the data in a list box directly – one way around this is to use a combo box instead; combo boxes are discussed next. When the list of data is too long for the list box, Visual Basic will add a vertical scrollbar. Let's examine most of the important list box control properties, events and methods.

## Properties

Many of the list box properties are shared by a combo box control and some of them are essential for getting the best from the control:

Appearance	FontBold	List	Style
BackColor	FontItalic	ListCount	TabIndex
CausesValidation	FontName	ListIndex	TabStop
Columns	FontSize	MouseIcon	Tag
Container	FontStrikethru	MousePointer	Text
DataChanged	FontUnderline	MultiSelect	ToolTipText
DataField	ForeColor	Name	Top
DataFormat	Height	NewIndex	ToolIndex

NOTES

DataMember	HelpContextID	OLEDragMode	Visible
DataSource	hWnd	Parent	WhatsThisHelpID
DragIcon	Index	RightToLeft	Width
DragMode	IntegralHeight	Selcount	
Enabled	ItemData	Selected	
Font	Left	Sorted	

## NOTES

The Columns property lets you create a multicolumn list box. Unfortunately, the columns are of the snaking or newspaper, type. There's no direct support for the multiple columns of an Access-style list box where different data items are displayed in separate columns. Instead Visual Basic wraps the same type of data items from column to column.

The List property sets or returns the value of an item in the list. You use the index of the item with the List property. The index position of items in a list box start at 0 (zero) and run to 1 less than the total number of items in a list. Thus, if you had 10 items in the list box, the index positions run from 0 to 9.

You use the List property to get the value of any item in the list. For example, to return the value of the third item in the list, use the following `listList1.List(2)`

To get the value of the currently selected item in the list, simply use the Text property of the list box. The ListIndex property sets or returns the index position of the currently selected item – if no item is selected, the ListIndex property is -1.

You can pick up the index position of the last item added to a list with the NewIndex property. The ListCount property returns the number of items in a list box. Confusingly, this is always 1 greater than the value returned by the NewIndex property – this is because the index positions count from 0 (zero) while the ListCount property starts counting from 1 for the first item. ListCount returns 0 if the list box is empty.

The MultiSelect property determines whether the user can select one item or whether they can select more than one. List boxes support both a simple and an extended multiple selection. A simple multiple selection allows the user to select contiguous items – usually accomplished with the mouse and the Shift key. An extended multiple selection lets the user select contiguous and noncontiguous items – usually done by mouse clicks in conjunction with the Ctrl and/or Shift keys.

The Selected property is a Boolean property and is a run-time property only. A Boolean property is one that can take only a True or False setting. The following line preselects the third item in a list box:

```
lstList1.Selected(2) = True
```

note the use of index position (2) to reference the third item in the list.

The final property that we're going to consider is the Sorted property. This is one of those properties that you can set only at design time. You can read it or return it, at run time (that is, see if it's true or False) but you can't set it (that is, change an unsorted list into a sorted one or vice versa). When you set the Sorted property to True at design time, any items you add to a list box (typically, with the AddItem method) are sorted in alphabetical order. The sort can only be in ascending order and it's not case-sensitive.

## Events

The list box control supports a few events, shown here:

Click	KeyDown	MouseUp	OLEStartDrag
Db1Click	KeyPress	OLECompleteDrag	Scroll
DragDrop	KeyUp	OLEDragDrop	Validate
DragOver	LostFocus	OLEDragOver	
GotFocus	MouseDown	OLEGiveFeedback	
ItemCheck	MouseMove	OLESetData	

NOTES

Perhaps the most commonly used event for a list box is Db1Click(). This coincides with the normal operation of list boxes in Windows applications. The first thing to do with a list box is usually to fill it with items for the list. The AddItem method can be used to do this. You can then, if you want, preselect one of the items in the list by setting the Selected property to True. The user either accepts the default selection or chooses another item with a single-click. Then clicking an OK command button carries out a process using the Text property which returns the value of the selected item. However, a popular shortcut is to double-click the item in the list – that way, the user can both select an item and initiate a process based on that item in a single action. Many Windows applications adopt this technique to copy one item from a list box into another list box.

## Methods

The list box has many of its own methods, as well as some common to the other controls discussed so far:

AddItem	Move	SetFocus
Clear	Refresh	ShowWhatsThis
Drag	RemoveItem	Zorder

There are three methods here worthy of note – AddItem, Clear and RemoveItem. AddItem, as already indicated is for adding items to a list box control. The RemoveItem method, as you might expect, removes items from a list box. To remove all the items in one fell swoop, use the Clear method.

An example of this simplest syntax for the AddItem method is:

```
LstList1.AddItem "Hello"
```

This adds the word "Hello" to the list box. Often you employ a number of AddItem methods, one after the other, to populate (fill in) a list box. Many developers place the AddItem methods in a Form\_Load() event procedure so the list box is filling as the form loads. You can specify the position that an item will take in a list by specifying the index position:

```
LstList1.AddItem "hello", 3
```

This places the text "Hello" at the fourth position in the list. If you omit the index position, the item is added to the end of the list-or if the Sorted property is set to True, the item is placed in the correct sorted order.

## Using Combo Boxes

### NOTES

The name combo box comes from "combination box". The idea is that a combo box combines the features of both a text box and a list box. A potential problem with list boxes – in some situations anyway – is that you're stuck with the entries displayed. You can't directly edit an item in the list or select an entry that's not already there. Of course, if you want to restrict the user, than a list box is fine in this respect. A combo box control (at least in two of its styles available in Visual Basic) allows you to select a predefined item from a list or to enter a new item not in the list. A combo box can also incorporate a drop-down section – that means it takes less room on a form than a normal list box. In all, there are three types of combo boxes to choose from at design time: a drop-down combo, a simple combo and a drop-down list. You can specify the type by setting the Style property.

Apart from the Style property, the properties, events and methods of combo boxes are very similar to those of list boxes, described shortly. However, the Text property is different. With a list box, the Text property returns the text of the currently selected item at run time. With a combo box, on the other hand, you can assign a value to the Text property at run time – in effect, you can set the text; even if the item is not already in the list.

### Properties

These are the properties for the combo box control:

Appearance	FontItalic	ListCount	Style
BackColor	FontName	ListIndex	TabIndex
CausesValidation	FontSize	Locked	TabStop
Container	FontStrikethru	MouseIcon	Tag
DataChanged	FontUnderline	MousePointer	Text
DataField	ForeColor	Name	ToolTipText
DataFormat	Height	NewIndex	Top
DataMember	HelpContextID	OLEDragMode	TopIndex
DataSource	hWnd	Parent	Visible
DragIcon	Index	RightToLeft	WhatsThisHelpID
DragMode	IntegralHeight	SelLength	Width
Enabled	ItemData	SelStart	
Font	Left	SelText	
FontBold	List	Sorted	

The List, ListCount, ListIndex, NewIndex and Sorted properties are identical to those for a list box. The property that's different and that consists of one of the fundamentals of designing combo boxes, is the Style property. There are three settings for Style, and the behavior and appearance of the combo box are determined by the setting you choose.

The styles are numbered from 0 to 2 and represent, in order, a drop-down combo, a simple combo and drop-down list.

- The drop-down combo looks like a standard text box with a drop-down arrow to the right. Clicking the arrow opens a list beneath the text box. The user has the choice of selecting an item from the list, which places it in the text box or of entering their own text in the text box. In other words, this is the true combo box.
- The simple combo is a variation of this theme – the only difference being that the list is permanently displayed. This one's an option if your form is not too crowded.
- The final style, the drop-down list is really a type of list box rather than a combo box. It looks identical to a drop-down combo but, as the name suggests, the user is confined to selecting a predefined item from the list. The advantage of this latter style is that it takes up less space than a conventional list box.

## NOTES

## Events

Here are the events of the combo box:

Change	DropDown	LostFocus	OLESetData
Click	GotFocus	OLECompleteDrag	OLEStartDrag
DbClick	KeyDown	OLEDragDrop	Scroll
DragDrop	KeyPress	OLEDragOver	Validate
DragOver	KeyUp	OLEGiveFeedback	

Most of the events for a combo box control are the standard ones. However, the DropDown() event is specific to combo boxes – though not supported by the simple combo box, which is already “dropped-down.”

The Change() event is not supported by the drop-down list because the user can't change the entry in the text box section. To see if that type of combo box has been accessed by the user, try the Click() or the DropDown() event procedures.

The DbClick() event is only relevant to the simple combo box for it's the only one where he user can see the full list by default. Usually the DbClick() event procedure calls the Click() event for a command button. This means the user can simply double-click an item in the list, rather than using a single-click to select followed by a click on a command button to carry out some processing based on the user selection.

## Methods

The methods you can apply to a combo box control are the same as those for a list box:

AddItem	Move	RemoveItem	Zorder
Clear	OLEDrag	SetFocus	
Drag	Refresh	ShowWhatsThis	

Again, the important methods are AddItem, Clear and RemoveItem. And, just as with a list box control, it's common practice to populate a combo box with a series of AddItem methods in the Load vent of a form. Incidentally, especially if you've worked with the database application Microsoft Access, you may be wondering about the flexibility of list and combo boxes. What do you do if the predetermined items

NOTES

in the list continually change? Do you continually have to re-code EDE you generate from your project? Another concern of yours could be regarding the actual tedium of typing long series of AddItem methods.

All of these questions are easily resolved if you exploit the RowSource and ListField properties of a data-bound list or data-bound combo box. Even more flexibility is provided by these data-bound versions of those two controls (DBList and DBCombo). Often you want the user first to select an item from a list box and then to click a command button. The button initiates an action using the selected item. An accepted alternative is for the user to simply double-click the item in the list. This both selects the item and carries out the action.

**Using Image Objects**

The image control (its prefix is often img) is a lightweight equivalent of the picture box control, which is described in a later section. But unlike the picture control, the image control can't act as a container for other objects. In some of its other properties it's not as versatile, but it's a good choice if you simply want to display a picture on a form. Image controls consume far less memory than picture controls. The image control that comes with Visual Basic can now display bitmap (.BMP), icon (.ICO), metafile (.WMF), JPEG (.JPG) and GIF (.GIF) files. This makes it easier to display graphics from the World Wide Web as well as graphics from other popular graphics program.

**Properties**

The image control utilizes several properties, but fewer than the picture box, discussed later.

Appearance	DragIcon	MousePointer	Tag
BorderStyle	DragMode	Name	Top
Container	Enabled	OLEDragMode	Visible
DataField	Height	OLEDropMode	WhatsThis
HelpID			
DataFormat	Index	Parent	Width
DataMember	Left	Picture	
Datasource	MouseIcon	Stretch	

As with most other graphics controls, you add the graphic by setting the Picture property. Perhaps the most interesting property here is Stretch. This is a Boolean property – meaning it takes only the values True or False. When Stretch is set to False (the default), the control resizes to the size of the picture placed inside it. If you later resize the control, then the loaded picture either is cropped or has empty space showing around it or both, depending on the relative directions of horizontal and vertical resizing. But if you set Stretch to true, the picture resizes with the control. Thus you can make the enclosed picture larger or smaller or fatter or thinner, by resizing the control after the picture is loaded. A picture control has no Stretch property. Its nearest equivalent is the AutoSize property. When AutoSize is set to True for a picture box, then the control adapts to the size of the loaded picture. However, unlike an image control with Stretch turned on, if the picture box is resized the enclosed picture remains the same – the picture does not “stretch” with the picture box control.

## Events

The image control doesn't use many events and of those, you may only use a few, if any.

Click	MouseDown	OLEDragDrop	OLEStartDrag
DblClick	MouseMove	OLEDragOver	
DragDrop	MouseUp	OLEGiveFeedBack	
DragOver	OLECompleteDrag	OLESetData	

NOTES

Image controls are sometimes handy as drag-and-drop destinations. This is because you can have a picture inside that gives an indication of the results of dropping onto the control.

## Image Methods.

The following are properties of an image control:

Drag	OLEDrag	ShowWhatsThis
Move	Refresh	ZOrder

You will most likely not use any of these methods in your applications.

## Picture Boxes

As you might expect, picture boxes often display graphics (for example, bitmaps, icons, JPEGs and GIFs). In this role, picture boxes are similar to image controls. However, picture boxes and images have slightly different properties and therefore behave differently. If you just want to show a picture, then an image control is usually a better choice than a picture box. An image control takes up less memory and is a lightweight version of the picture box control. However, if you want to move the graphic around the form, a picture box produces a smoother display. In addition, you can create text and use graphics methods in a picture box at run time. The graphics methods enable you to draw lines, circles and rectangles at run time. But, most importantly for this application, picture boxes can act as containers for other controls. Thus, you can place a command button within a picture box. In this respect, picture boxes function as "forms within forms".

## Properties

The table below lists the properties for the picture box control. Notice that there are many more properties for this control than there are for the image control.

Align	FillStyle	MousePointer
Appearance	Font	Name
AutoRedraw	FontBold	OLEDragMode
AutoSize	FontItalic	OLEDropMode
BackColor	FontName	Parent
BorderStyle	FontSize	Picture
CausesValidation	FontStrikethru	RightToLeft

NOTES

ClipControls	FontTransparent	ScaleHeight
Container	FontUnderline	ScaleLeft
CurrentX	ForeColor	SelectionMode
CurrentY	HasDC	ScaleTop
DataChanged	hDC	ScaleWidth
DataField	Height	TabIndex
DataFormat	HelpContextID	TabStop
DataMember	hWnd	Tag
DataSource	Image	ToolTipText
DragIcon	Index	Top
DragMode	Left	Visible
DrawMode	LinkItem	WhatsThisHelpID
DrawStyle	LinkMode	Width
DrawWidth	LinkTimeout	
Enabled	LinkTopic	
FillColor	MouseIcon	

Quite a lot of properties this time! When you put a picture into a picture box, it appears as its normal size. If it's too big for the picture box, the graphic is clipped. Setting the AutoSize property to True causes the picture box to resize to match the size of the graphic. The graphic displayed in the picture box is determined by the Picture property you can change this property at both design time and run time. There's a similar-sounding property – the Image property. This one's only available at run time and it's used to make a copy from one picture box to another. The syntax for doing this is:

```
Picture2.Picture = Picture1.Image
```

You can place this line of code in any event where it is relevant. For example, may be you want to change the picture in a picture box when the user selects a different record in a database.

The line above places a copy (image) of the picture in the first picture box into the second picture box (using its Picture property). You can even change the picture directly at run time. The syntax is:

```
Picture1.Picture = LoadPicture ("filename")
```

To empty a picture box, use the Visual Basic LoadPicture function with no parameter:

```
Picture1.Picture = LoadPicture ()
```

**Events**

The picture box events are listed in the following table:

Change	KeyPress	MouseDown	OLESetData
Click	KeyUp	MouseMove	OLEStartDrag

Db Click	LinkClose	MouseUp	Paint
DragDrop	LinkError	OLECompleteDrag	Resize
DragOver	LinkNotify	OLEDragDrop	Validate
GotFocus	LinkOpen	OLEDragOver	
KeyDown	LostFocus	OLEGiveFeedback	

NOTES

Two of the popular events for picture boxes are the Click() and DragDrop() events.

## Methods

The picture box controls supports more methods than its counterpart, the image box. The most important ones are listed in boldface in the following table:

Circle	LinkRequest	PSet	TextHeight
Cls	LinkSend	Refresh	TextWidth
Drag	Move	ScaleX	Zorder
Line	OLEDrag	ScaleY	
LinkExecute	PaintPicture	SetFocus	
LinkPoke	Point	ShowWhatsThis	

The Circle, Cls, Line, PaintPicture and PSet methods are all used when drawing graphics or text in the picture at run time – Cls (like the old DOD command for “clear screen”) is actually used to erase entries. The Zorder method is the run-time equivalent of Format – Order – Bring to Front or Format – Order – Send to Back. You can use Zorder to determine which controls overlap other controls. However, you should be aware that there are three layers on a form – Zorder only works within the layer that contains the control. All the nongraphical controls except labels (for example, command buttons) belong to the top layer. Picture boxes and graphical controls (as well as labels) belong to the middle layer. The bottom layer contains the result of the graphics methods – for instance, a circle drawn with the Circle method is on the bottom layer. This contrasts with a circle drawn with the Shape control, which is in the middle layer. What all this means is that you can't position a picture box over a command buttons with ZOrder – the picture box over a command button with ZOrder – the picture box is permanently relegated to the layer behind. The ZOrder method is for rearranging objects within one layer.

## Timers

The timer control is one of the few controls always hidden at run time. This means you don't have to find room for it on a form – it can go anywhere, even on top of existing controls. The timer basically does just one thing: It checks the system clock and acts accordingly.

## Properties

The timer control does not have many properties, as you can see in this table:

Enabled	Left	Tag
Index	Name	Top
Interval	Parent	

NOTES

Apart from the Name Property (a tmr prefix is recommended); there are only two important properties for the time control – the Enabled property and the Interval property. Indeed, you have to set these properties to get the timer to do anything at all (assuming the Enabled property is at its default, true). The Left and Top properties are virtually superfluous – it makes little difference where you put a timer on a form.

The Interval property is measured in milliseconds. This means if you want to count seconds, you have to multiply the number of seconds by 1,000. Once an interval has elapsed (provided the timer is enabled), the timer generates its own Timer event. It does this by checking the system clock at frequent intervals.

**Event**

The timer control has only one event called, appropriately, a Timer() event. As already stated, this event takes place every time an interval elapses. The interval is determined by the Interval property. To stop the Timer() event from occurring, you can set the timer's Enabled property to False at run time.

**Methods**

The timer control does not support any methods at all.

**Using Scroll Bars**

A scroll bar control on a form is not to be confused with a scroll bar on a large text box or list box. The scroll bar controls are completely independent objects that exist without reference to any other control (this is not the case with large text boxes or list boxes). The horizontal scroll bar and the vertical scroll bar are identical except for their orientation. Both controls share the same properties, events and methods. When the term "scroll bar" is used in this section it means both the horizontal scroll bar and the vertical scroll bar.

A scroll bar control is typically employed to increase or decrease a value. For example, you may want to change a color setting, a number or the volume of a digital audio device. The scroll bar acts as a sliding scale with a starting point and an ending point, including all the values in between. If you simply want to increment or decrement a number, then you should also take a look at the spin button custom control.

One problem with a scroll bar is that once it is used it retains the focus and may flicker on screen. To circumvent this you can change the focus to another control.

**Properties**

The scroll bar control has a few properties worth nothing:

CausesValidation	hWnd	MousePointer	Tag
Container	Index	Name	Top
DragIcon	LargeChange	Parent	Value
DragMode	Left	RightToLeft	Visible
Enabled	Max	SmallChange	WhatsThisHelpID
Height	Min	TabIndex	Width
HelpContextID	MouseIcon	TabStop	

The most useful scroll bar properties are the Max, Min, LargeChange and SmallChange properties. The Min and Max properties determine the limits for the Value property of the scroll bar. You can set the Min property to the lowest value allowed, for example 0. Then you would set Max to the maximum allowed value.

For example, the following code could be used to define the lowest and highest volume allowed by your application:

```
Private Sub Form_Load()
    hscVolume.Min = 0 ' The lowest volume
    hscVolume.Max = 255 ' The maximum volume
End Sub
```

The LargeChange property determines how much the Value property changes when the user clicks in the scroll bar. The SmallChange property sets the amount the Value property changes when the user clicks one of the scroll arrows at either end of the scroll bar. You don't have to worry about the direction of the change, only the amount; Visual Basic figures out whether it's an increase or decrease, depending on where you click. There is no property setting to correspond to the user dragging the scroll box (also called thumb or elevator) within the scroll bar.

This is because there's no way of predicting how far the scroll box will be dragged. However, it automatically updates the Value property. You can ascertain the new Value in the Change event procedure for the scroll bar. The value property can also be set at design time to place the scroll box at a certain point within the scroll bar. For example, if you wanted the volume in the previous example to be set half way, you would use the following code:

```
Private Sub Form_Load()
    hscVolume.Min = 0 ' The lowest volume
    hscVolume.Max = 255 ' The maximum volume
    hscVolume.Value = 128 ' Set the volume half way
End Sub
```

The Value, LargeChange and SmallChange property settings must lie within the range dictated by the Min and Max properties. Value is usually set equal to Min or Max so the scroll box is at one end of the scroll bar. LargeChange is ordinarily some integral multiple of SmallChange. Max can be less than Min, which is often counter-intuitive. Max and Min or both, can also be negative.

## Events

This is the list of events supported by the horizontal and vertical scroll bars:

Change	GotFocus	KeyUp	Validate
DragDrop	KeyDown	LostFocus	
DragOver	KeyPress	Scroll	

There are two vital events here – Change() and Scroll(). The Change() event occurs whenever the Value property of the scroll bar is altered at run time. The Value property, in turn, is changed whenever the user clicks a scroll arrow (SmallChange) clicks in the scroll bar to one side of the scroll box (LargeChange) or stops dragging the scroll box along the scroll bar. The latter induces a Value change depending on the Length of

NOTES





## NOTES

the drag – though the Value change can never be greater than the difference between the Min and Max properties.

Although the .Change() event is generated when the user stops dragging the scroll box, it does not occur during the drag. If you want to generate the Change() event as the user drags, then you must call it from the Scroll() event. The Scroll() event is continually triggered as the user drags the scroll box. By calling the Change() event from the Scroll() event, you can continually generate a Change() event. If you don't you have to wait until the user stops dragging to ascertain the results of the action. On the other hand, any kind of click on the scroll bar produces an immediate Change() event.

### Methods

The following scroll bar methods are not terribly important and are rarely beneficial.

Drag	Refresh	ShowWhatThis
Move	SetFocus	ZOrder

### Using Drive Lists

The drive list box control (or just drive), is normally used in conjunction with the directory list and the file list controls. At its most fundamental, these three controls allow the user to select a file in a particular directory on a particular drive. The user changes to another drive via the drive control. They switch directories with the directory control and they select the file from the file control. You can also use the drive and directory controls to let the user choose a destination for a file they wish to save. Although a common dialog controls is better suited for retrieving filenames, we will look at these three controls in the following sections. You will be able to find other uses for them as you develop your skills.

### Properties

The following table lists the many properties for the drive list box control:

Appearance	FontItalic	List	Tag
BackColor	FontName	ListCount	ToolTipText
CausesValidation	FontSize	ListIndex	Top
Container	FontUnderline	MouseIcon	TopIndex
DragIcon	ForeColor	MousePointer	Visible
DragMode	Height	Name	WhatsThisHelpID
Drive	HelpContextID	OLEDropMode	Width
Enabled	hWnd	Parent	
Font	Index	TabIndex	
FontBold	Left	TabStop	

For the Name property a drv prefix is normally adopted. Apart from the Name, the single most important property is Drive. This is a run-time property only, which is used to return the drive the user has selected in the drive control. Your application could retrieve this value and use it to synchronize the directory for file list controls

explained later. The Drive property is invariably accessed in the Change event procedure for the drive control.

## Events

The drive list box has a few events, but few are useful to the beginning programmer.

Change	KeyDown	OLECompleteDrag	OLESetData
DragDrop	KeyPress	OLEDragDrop	OLEStartDrag
DragOver	KeyUp	OLEDragOver	Scroll
GotFocus	LostFocus	OLEGiveFeedback	Validate

NOTES

The Change() event is the most popular event to trap. It is triggered whenever the user makes a selection of a drive in the drive control. The Drive property of the control is used to update the display of directories in the directory list control. Thus, the directories shown are always those on the currently selected drive. A fuller discussion of how to do this appears under the section on the file control, which follows shortly.

## Methods

Below is the list of methods for this control.

Drag	OLEDrag	SetFocus	Zorder
Move	Refresh	ShowWhatsThis	

These methods are rarely used.

## Using Directory List Boxes

As already stated, the directory list box (or simply directory) control, is used in conjunction with the drive control, described earlier and file control. The user can select a directory on the current drive from the directory list. However, it's important to update the directories displayed when the user changes drives in the drive control. It is also important to update the files shown in a file control, too. To do these, the directory's Path property and Change() event are used.

## Properties

The following are the directory list box properties:

Appearance	FontName	List	TabIndex
BackColor	FontSize	ListCount	TabStop
CausesValidation	FontStrikethru	ListIndex	Tag
Container	FontUnderline	MouseIcon	ToolTipText
DragIcon	ForeColor	MousePointer	Top
DragMode	Height	Name	TopIndex
Enabled	HelpContextId	OLEDragMode	Visible
Font	hWnd	OLEDropMode	WhatsThisHelpID
FontBold	Index	Parent	Width

## NOTES

FontItalic                      Left                      Path

A directory control is often given a Name property with a dir prefix. The Path property is a run-time property that sets or returns the path to the directory in the directory list. It's usually accessed in the Change event for the drive control – where it updates the list of directories to match the drive selected by the user. The Path property is also used in the directory control's Change event procedure to update the list of files in a file control when the user changes directories or drives.

### Events

The table below shows the events used by the directory list box.

Change	GotFocus	LostFocus	OLECompleteDrag
Click	KeyDown	MouseDown	OLEDragDrop
DragDrop	KeyUp	OLEDragOver	OLEStartDrag
DragOver	MouseMove	OLEGiveFeedback	Scroll
KeyPress	MouseUp	OLESetData	Validate

Although you may use the Click() event in your code, the Change() event procedure is where you will place code to update the files in a file list control.

### Methods

You probably won't be working with the following directory list box methods too often:

Drag	OLEDrag	SetFocus	ZOrder
Move	Refresh	ShowWhatsThis	

### Using File List Boxes

The File List Box control comes at the end of the drive-directory-file chain. To reiterate, the file control should be updated in the directory Change() event. The directory control itself is updated when the user selects a directory in the directory control – it's also updated when the user selects a new drive in the drive control. For these links to work you must code two Change () event procedures correctly.

### Properties

File list boxes have a lot of properties and many of them quite valuable:

Appearance	FontName	ListCount	ReadOnly
Archive	FontSize	ListIndex	Selected
BackColor	FontStrikethru	MouseIcon	System
CausesValidation	FontUnderline	MousePointer	TabIndex
Container	ForeColor	MultiSelect	TabStop
DragIcon	Height	Name	Tag
DragMode	HelpContextID	Normal	ToolTipText
Enabled	Hidden	OLEDragMode	Top

FileName	hWnd	OLEDropMode	TopIndex
Font	Index	Parent	Visible
FontBold	Left	Path	WhatsThisHelpID
FontItalic	List	Pattern	Width

## NOTES

Let's concentrate on just a few. The Path property is vitally important. It's a run-time property and is often both set and returned. When the Path property is returned, Visual Basic is aware of the path to the currently selected file in the file control. To ascertain the path and the filename together (sometimes called fully qualified path), you have to concentrate the Path property with the FileName property. The fully qualified path can then be used as a basis for opening files.

Saving files is not quite so straightforward – you'll need to provide a text box for the user to enter a new filename rather than writing over the selected file. An alternative approach is to generate the filename for a saved file automatically and use the controls to determine only the drive and directory for the file. In that case, you might want to disable the file control by setting its Enabled property to False, or make it invisible by setting its Visible property to False.

The Path is set in response to the user changing the drive (in a drive control) or the directory) or the directory (in a directory control). You must code the series of possible events correctly for this to work. The following steps give you an idea of how this works:

1. In the Drive1\_Change() event, you would add a line like this: `Dir1.Path = Drive1.Drive`

This line updates the directory list to reflect the selected drive. The fact that the directory Path property changes in code generates a Change() event for the directory control as well. And the same event is generated if the user manually changes directories in the directory control.

2. You now code the Change() event for the directory as follows:

```
File1.Path = Dir1.Path
```

This ensures that the files displayed (governed by the file control's Path property) reflect both the currently selected drive and the directory). Changing a drive automatically selects a new directory.

The Pattern property can be set at design time – it can also be changed at run time. By default the Pattern property as `*.*`, which shows all files in the file control. You can narrow down the list of files by providing a suitable filter, for example `*.txt` to display just your text files.

The Archive, Hidden, Normal, ReadOnly and System properties can all be used to further narrow or expand the list of files. Hidden and System are False by default – ideally, you wouldn't want the end user even to be aware that hidden and system files exist.

By using the above code, you will also synchronize the controls when the application begins.

## Events

This is the list of events supported by the File List Box control.

## NOTES

Click	KeyPress	OLECompleteDrag	PathChange
DbClick	KeyUp	OLEDragDrop	PatternChange
DragDrop	LostFocus	OLEDragOver	Scroll
DragOver	MouseDown	OLEGiveFeedback	Validate
GotFocus	MouseMove	OLESetData	
KeyDown	MouseUp	OLEStartDrag	

In a sense, the events for a file list control are similar to those of an ordinary list box. The standard approach is to have a command button's Click() event procedure carry out some processing based on the Path and FileName properties of the file control. However it's often helpful to give the user a double-click alternative. The way to do this, if you recall from an earlier discussion on list boxes, is to code the file control's DbClick() event procedure to call the command button's Click() event.

Two events that are specific to a file control are PathChange() and PatternChange(). The PathChange() event occurs when the file's Path property alters. Similarly, the PatternChange() event occurs when the Pattern property is changed in code. It's common practice to let the user enter a pattern in a text box and set the Pattern property to the value of the Text property of the text box at run time. You can then use the PatternChange() event procedure to reset the Pattern if the user has entered a pattern that might be dangerous, for example \*.ini.

### Methods

This control only supports a few methods, listed in the table below. None of them are particularly useful for the operation of the control.

Drag	OLEDrag	SetFocus	ZOrder
Move	Refresh	ShowWhatsThis	

### Adding other Controls to the Toolbox

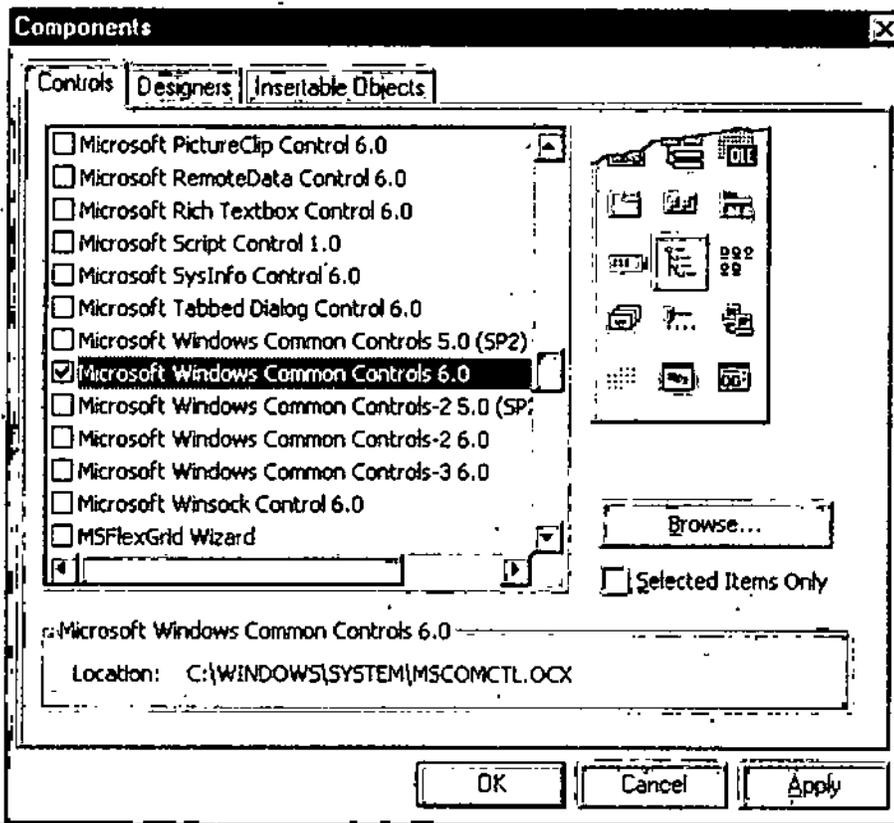
Before we move on to the next set of controls, you will need to learn how to add additional controls to your Toolbox.

You can add other controls by following a few simple steps:

- Add a tab to the Toolbox to keep it neat.
- Select the controls to add.
- Move them to the appropriate tab if necessary.

To teach you how to add controls to the Toolbox, let's add the controls used in the next sections.

1. Right-click the Toolbox to display its pop-up menu.
2. Select Add Tab from the pop-up menu.
3. When prompted for the tab name, type in Common Controls and click the OK button.
4. Now that the Common Controls tab has been added to the Toolbox, click it to make it the active tab.



## NOTES

5. Right-click the Toolbox and select Components from the pop-up menu.
6. From the Controls tab on the Components dialog box, click the check box next to Microsoft Windows Common Controls 6.0 as in figure.
7. Click the OK button to add the controls to the tab.

If the controls don't "land" on the appropriate tab, you can move them by dragging them to the appropriate tab the Toolbox.

You will notice that you may not be able to see "Common Controls" on the tab you just created. You can fix this by either stretching the Toolbox to the right to make it bigger or you can rename the tab. To rename the tab

1. Right-click in the Common controls tab.
2. Select Rename Tab from the pop-up menu.
3. When the input box appears, rename the tab to Common.
4. Click the OK button to close the dialog box. When the dialog box closes the tab will be readable on the Toolbox.

That's all there is to adding controls to the Toolbox. Although it's not required, it is a good idea to create tabs that you can categories your controls by. This will help make it easier for you to locate the control you need. Use the tabs to avoid Toolbox clutter.

## Designing Windows 95/98 – Style Interfaces

Now that you are familiar with the most common Visual Basic controls, we can learn how to design applications that look, feel and behave just like the commercial applications written specifically for Windows 95/98 and Windows NT.

There are five controls that provide most of the functionality found in the most common Windows applications. These include the Tree View, List View, Image List, Status Bar and toolbar.

Figure shows the Windows Explorer applet. It contains all of the controls that I have just described.

## NOTES

At the top of the window, just below the menu, is the Toolbar control. It provides buttons that allow you quick access to the most commonly used functions, such as cut, copy, paste and delete.

The pane on the left side of the window is a Tree View control. Its name describes its function: it displays items in a tree format or tree view.

The pane on the right side is a List View control. Its purpose is to show a list of items. This control is most often paired with the tree View control and lists the contents of a selected folder in the Tree View. At the bottom of the window is the Status Bar control. A status bar is used for many functions, including showing the number of selected objects, displaying the system date and / or time and the amount of free disk space.

Now that you have an idea of what these controls do and how they look, let's learn how to use them in them in our applications.

### Using the Tree View Control

The Tree View control provides a hierarchical view of folders or other items that can be neatly categorized in a tree-style layout. It is often used in conjunction with a List View control which is used to display the contents of the folder selected in the tree view. Let's take at the Tree View's properties.

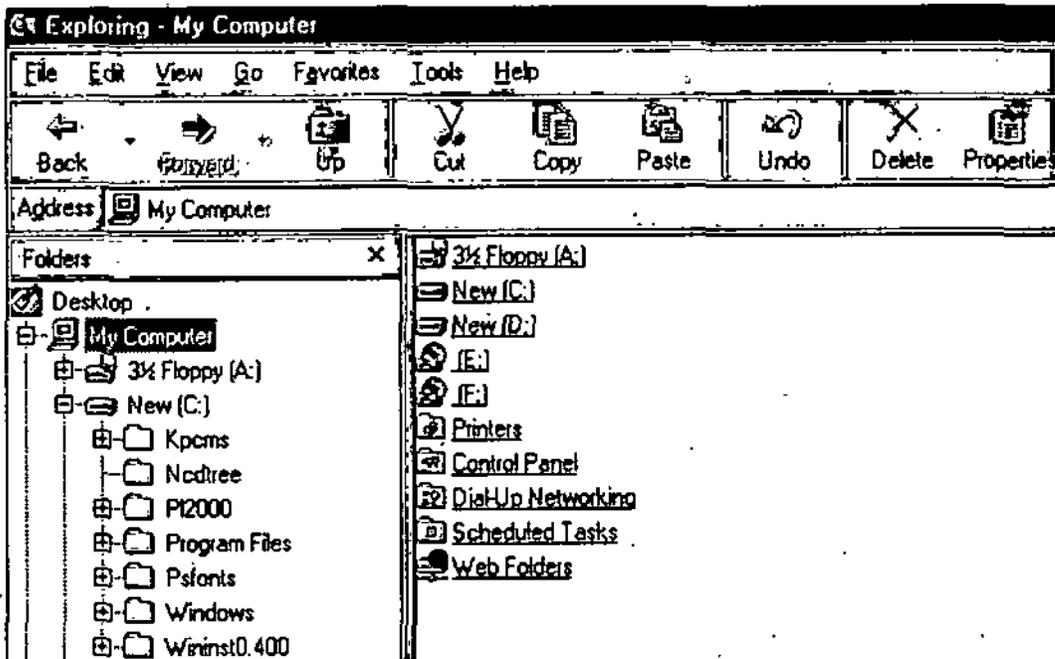
### Properties

The Tree View control has a lot of properties – and many of them quite valuable:

Appearance	Height	MouseIcon	SingleSel
BorderStyle	HelpContextID	MousePointer	Sorted
CausesValidation	HideSelection	Name	Style
CheckBoxes	HotTracking	Nodes	TabIndex
Container	HWND	Object	TabStop
DragIcon	ImageList	OLEDragMode	Tag
DragMode	Indentation	OLEDropMode	ToolTipText
DropHighlight	Index	Parent	Top
Enabled	LabelEdit	PathSeparator	Visible
Font	Left	Scroll	WhatsThisHelpID
FullRowSelect	LineStyle	SelectedItem	Width

Aside from the standard properties, there are some new ones that you must get familiar with to exploit the power of this control.

The Name property is the first property you should set when working with this control.



## NOTES

The standard naming convention prefix is `tw`. For example, if you had a tree view that contained the directory structure of your hard drive, you could name the control `twDirectories`.

If you double-click the Custom field in the Properties window, you can bring up a property page like the one in figure that exposes the most important properties.

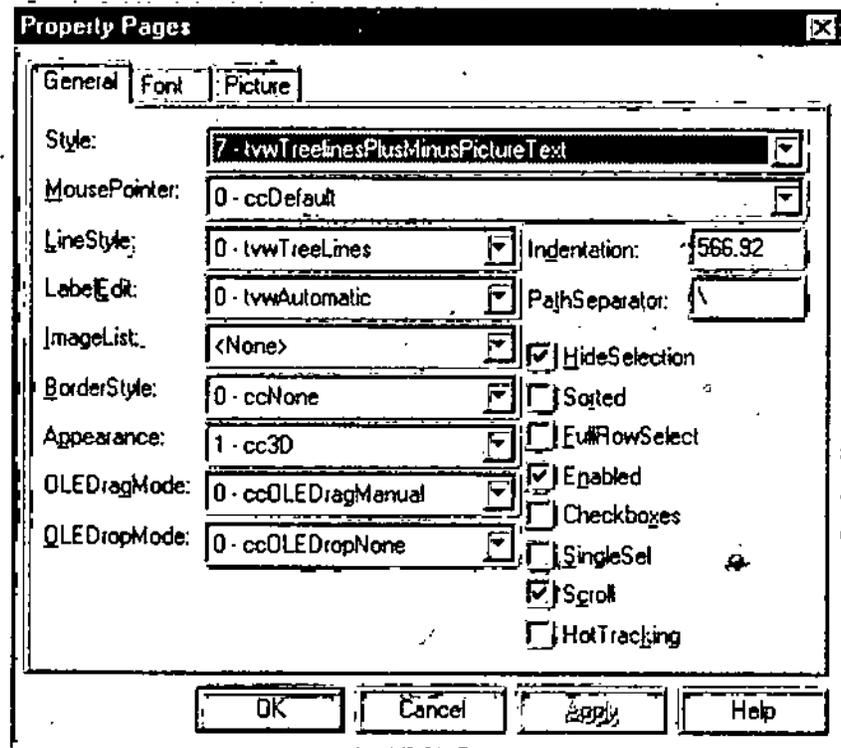
Of the three tabs displayed by the Tree View's property pages, the General tab is the one you will be most concerned with.

## The General Tab

The Style property allows you to determine how you want the control to look and behave on the form. The possible values you can set this property to are as follows:

Setting	Description
0 - <code>twTextOnly</code>	This setting shows only the text of the node.
1 - <code>twPictureText</code>	This setting shows the nodes icon and its text.
2 - <code>twPlusMinustext</code>	Use this setting if you want the collapse/expand symbol (the plus and minus signs) and the text of the node.
3 - <code>twPlusPictureText</code>	This setting displays the collapse/expand symbol, a small icon to the left of the next and the text itself.
4 - <code>twTreelinesText</code>	If you want to have lines connect nodes that are related in the tree's hierarchy, you can show the lines and the text for the node.
5 - <code>twTreelinesPictureText</code>	This setting displays a small icon to the left of the text and connects related nodes.
6 - <code>twTreelinesPictureText</code>	This setting displays a small icon to the left of the text and connects related nodes.

## NOTES



7 - twwTreenLinesPlusMinusPictureText      If you want everything shown choose this setting.

The LineStyle property is used to set the style of lines displayed between nodes. The possible values are

- 0 - TreeLines
- 1 - RootLines

The LabelEdit property is a Boolean property that allows you to enable or disable the automatic label-editing feature of the control. Windows Explorer demonstrates this feature when you single-click a folder or file name. It will turn into a miniature text box that will allow you to change the name. Set this property to True to enable label editing. Set it to False to turn it off.

If you want to have pictures in your Tree View control, you should set the ImageList property to the name of an existing Image List control, explained later in this chapter.

The BorderStyle and Appearance properties are self-explanatory.

OLEDragMode configures the control for either manual or automatic dragging. You set this property to one of the following values:

- 0 - OLEDragManual
- 1 - OLEDragAutomatic

OLEDropMode configures the Tree View control to enable or disable OLE drop operations. The value of the property can be one of the following:

- 0 - OLEDropNone      When set to this value, the target component does not accept OLE drops and displays the No Drop cursor.

The Indentation property determines the horizontal distance between nodes in the view. The lower of the number, the closer the nodes are. If you prefer a tighter-looking interface, I have found that a value of 283 looks nice at run time.

The PathSeparator property allows you to set or retrieve the delimiter character used for the path returned by a node's FullPath property as shown below:

```
Private Sub TreeView1_NodeClick(ByVal Node As Node)
    Dim rc as String
    rc = node.FullPath
    MsgBox rc
End Sub
```

For example, you could set this property to a backslash (\) if you were showing a list of folders on your hard drive. Or, you could use a period (.) to designate an Internet-style path, as if you were mapping IP subnets on your LAN.

Set the Scroll property to True if you want the Tree View to show scroll bars; if there are more nodes than can be listed in the Tree View at one time, this may be necessary. Set it to False to disable scroll bars.

Finally, you can set the HotTracking property to True if you want the full name to be displayed in a tool tip - style box when the name does not fit horizontally within the view, as shown in figure.

## Events

This is the list of events supported by the Tree View control.

AfterLabelEdit	Expand	MouseMove	OLEGiveFeedback
BeforeLabelEdit	GotFocus	MouseUp	OLESetData
Click	KeyDown	NodeCheck	OLEStartDrag
Collapse	KeyPress	NodeClick	Validate
DbtClick	KeyUp	OLECompleteDrag	
DragDrop	LostFocus	OLEDragDrop	
DragOver	MouseDown	OLEDragOver	

The AfterLabelEdit() event is triggered after you perform a label editing function on a node. This event is useful if you want to check the name of the node to make sure that it is valid. Its compliment, the BeforeLabelEdit() event, is triggered right before the node goes into the edit mode.

The Collapse() event is triggered when a user collapses a branch of the tree. This can happen by double-clicking a root of a branch or by clicking the minus sign next to the branch's root. You could place code in this event to remove items from the Tree View if you want to conserve memory.

The Expand() event is obviously the opposite of the Collapse() event. It is triggered when a user expands a branch of the tree by either clicking the plus sign next to the

NOTES

root of the branch or by double-clicking the root of the branch. You could use this event to dynamically load data into the control when the node is expanded.

The NodeClick() event is one of the most important events that the Tree View provides. You can use this event to retrieve information about the node that was clicked or any other function that your application requires. For example, the code

NOTES

```
Private Sub tvwNodes_NodeClick (ByVal Node As ComctlLib.Node)
    MsgBox Node.FullPath
End Sub
```

will display a dialog box that shows the complete path to the node you clicked. If you used the backslash character in the PathSeparator property, with a root node named C: and the node clicked named Windows, the dialog box would display C:\Windows.

OLECompleteDrag() is triggered when OLE data is dropped on the control or the OLE drag-and-drop operation is canceled.

The OLEDragDrop() event is triggered when the OLEDropMode property is set to 1 - Manual and OLE data is dropped on the control. You would place code in the event to determine what the control should do when data is dropped. You could place code in this event to make the control move data, but not copy it. Or you could have it copy the data, but not move it. The functionality depends on the needs of your application.

The OLEDragOver() event is triggered when OLE data is dragged over the control. After this event is triggered, Visual Basic will trigger the OLEGiveFeedback() which will allow you to check the data and provide feedback to the user.

The OLEGiveFeedback event is triggered after every OLEDragOver() event. It allows the Tree View control to provide feedback to the user, such as changing the mouse cursor to indicate what will happen if the user drops the object or provide visual feedback on the selection (in the source component) to indicate what can happen.

The OLESetData() event is triggered when a target control executes the GetData method on the source's DataObject object.

Finally the OLEStartDrag() event is triggered when a user starts dragging data from the control. If the OLEDragMode property is set to 1-Automatic, the control will automatically start. You can use this event to populate data in the control's DataObject object. This will allow the destination control to read the data from the DataObject.

## Methods

These are the methods exposed by the Tree View control:

Drag	Move	SetFocus	StartLabelEdit
GetVisibleCount	OLEDrag	ShowWhatThis	Zorder
HitTest	Refresh		

The GetVisible method is used to retrieve the number of nodes that can be viewed in the Tree View at open time. This is not to be confused with the total number that the control can hold, but just how many can be viewed without scrolling vertically in the control. This method is useful if you need to ensure that a given number of nodes are visible at a time.

The HitTest method is used to determine if a node is available as a drop target. You can use this event during on OLE drag operation to highlight a node or change the mouse pointer when the OLE data is over a target that will allow it to be dropped.

The OLEDrag method is called to initiate an OLE drag operation. After this operation is initiated, the control's OLEStartDRag event is triggered, allowing you to supply data to a target component.

The StartLabelEdit method is used when you want to force a node into label edit mode. You can use this method in special circumstances, for example when you have set the LabelEdit property to False, but need to change the name of this one node. A good example of this can be seen again in Windows Explorer. You can change the names of folders and files within the tree, but you cannot change the names of built-in components, such as drives, Control Panel or Network Neighborhood.

## Using the List View Control

As mentioned previously, the List View control is often used in conjunction with the Tree View control. There are times when it will be used separately, but for this chapter, we will use them together.

## Properties

Below are the properties for the List View control.

AllowColumnReorder	Height	Parent
Appearance	HelpContextID	Picture
Arrange	HideColumnHeaders	PictureAlignment
BackColor	HideSelection	SelectedItem
BorderStyle	HotTracking	SmallIcons
CausesValidation	HoverSelection	Sorted
Checkboxes	HWnd	SortKey
ColumnHeaderIcons	Icons	SortOrder
ColumnHeaders	Index	TabIndex
Container	LabelEdit	TabStop
DragIcon	LabelWrap	Tag
DragMode	Left	TextBackground
DropHighlight	ListItems	ToolTipText
Enabled	MouseIcon	Top
FlatScrollBar	MousePointer	Visible
Font	MultiSelect	WhatsThisHelpID
ForeColor	Name	Width
FontName	Object	View
FullRowSelect	OLEDragMode	
GridLines	OLEDropMode	

NOTES

NOTES

If you double-click the Custom field in the Property window for the List View control, Visual Basic will present you with the property pages for the control as shown in figure. These present the most useful, control-specific, properties that you could set to customize the look and behavior of the control.

Again, the first property you should set is the control's Name property. You can use a prefix like lvw. For example, if your control listed the files on your hard drive, you could name the control lvwFiles.

There are numerous tabs on the List View's property pages. You will be mostly concerned with the first four tabs.

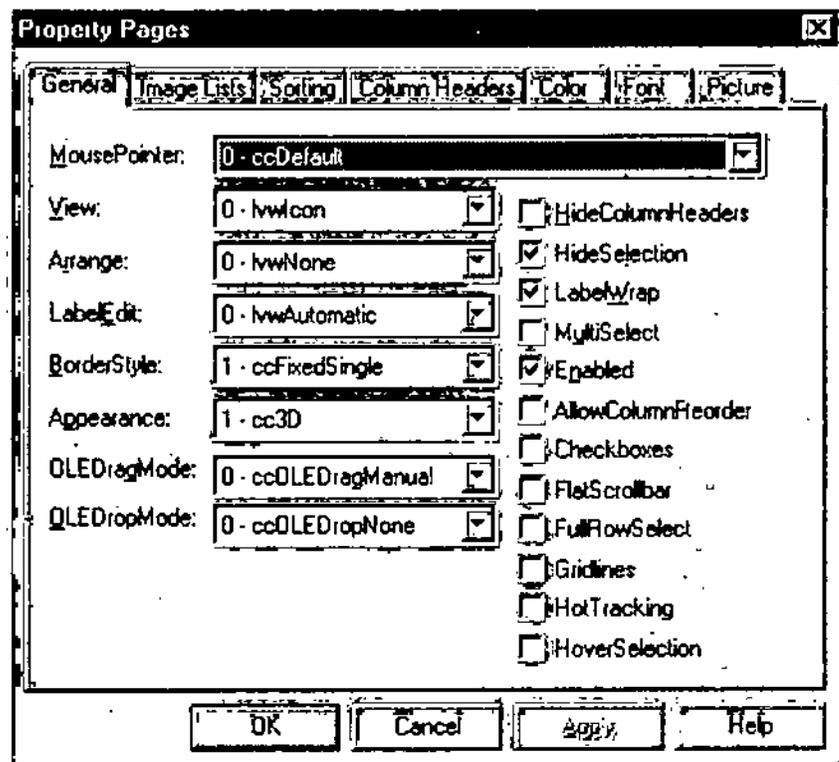
**The General Tab**

The first tab on the property page, the General tab, contains many properties that control the layout of the List View control.

You can set the View property to make the List View display items in one of four different views:

- 0 - lvwIcon                      Use this value to make the List View display large icons with text for each item in the list.
- 1 - lvwList                      This value will list the items much like a List Box control.
- 2 - lvwReport                    This setting is much like lvwList, but it will also show the subitems that belong to each in the list.
- 3 - lvwSmallIcon                This setting is like lvwIcon but it uses smaller icons.

The Average property allows you to set or retrieve the value that determines how the icons in the control's Icon or SmallIcon views are arranged. The possible values of this property are:



- 0 - `lvwNone` This setting will not control how the items are arranged.
- 1 - `lvwAutoLeft` This setting will make the control automatically arrange the items on the left of the control.
- 2 - `lvwAutoTop` This setting will make the control automatically arrange the items on the top of the control.

## NOTES

The `LabelEdit` property is a Boolean property that allows you to enable or disable the automatic label-editing feature of the control. Windows Explorer demonstrates this feature when you click a folder or filename. It will turn into a miniature text box that will allow you to change the name. Set this property to `True` to enable label editing. Set it to `False` to turn it off.

`OLEDragMode` configures the control for either manual or automatic dragging. You set this property to one of the following values:

0 - `OLEDragManual`

1 - `OLEDragAutomatic`

`OLEDropMode` configures the List View control to enable or disable OLE drop operations. The value of the property can be one of the following:

0 - `OLEDropNone` When set to this value the target component does not accept OLE drops the displays the "No Drop" cursor.

1 - `OLEDropManual` The control will trigger the OLE drop events, allowing the programmer to handle the OLE drop operation in code.

The `ColumnHeaders` property is set to `False` when you check the box next to Hide Column Headers.

The `LabelWrap` property allows you to set or retrieve the value that determines if labels are wrapped when the List View is in Icon view. If this box is checked, then the property is set to `True`.

If you want to allow the user to select more than one item at a time, you can set the `MultiSelect` property to `True`. If you want to restrict the user to one selection at a time, set this property to `False`.

The `FullRowSelect` property is interesting, because it allows the List View control to behave much like a cell in a spreadsheet that can contain graphics. You can set this property to `True` to make the control highlight an entire row within a column, just as a spreadsheet will highlight the entire cell, rather than just the text of the item.

If you want to make the control look and behave like a spreadsheet even further, you can set the `GridLines` property to `True`. Then you can set the List View to display in Report view. For example, if you had a List View named `lvwSheet`, you would use the following code to make the spreadsheet:

```
With lvwSheet
    . View = lvwReport
    . GridLines = True
    . FullRowSelect = True
End With
```

You can set the `HotTracking` property to `True` if you want the full name to be displayed in a tool tip - style box when the name does not fit horizontally within the view.

The `HoverSelection` property allows you to set or retrieve a value that determines if an object is selected when the mouse pointer hovers over it. Set this property to `True` to enable hover selection, `False` to disable it.

### The Image Lists Tab

#### NOTES

The image Lists tab contains the properties that enable the List View to use graphics. Normally, if you want to have pictures in your List View control, you should set the `ImageList` property to the name of the existing Image List control, explained later, in the section "Using the Image List Control".

When you set the `Small` field to the name of an Image List, it will set the List View's `SmallIcons` property to the Image List control. When your List View is set to display small icons, the icons will be retrieved from the Image List specified in this property.

The `ColumnHeader` field will set the `ColumnHeaders` property to the Image List that contains the icons to be displayed in the Column Headers of the List View.

### The Sorting Tab

The Sorting tab displays fields that expose the properties related to sorting data in a List View control.

The `Sorted` check box sets the `Sorted` property. If it is checked, then the List View will sort the data within it. If it is unchecked, then the `Sorted` property will be set to `False`.

If you want to specify how to sort the data within the List View, you can set the `SortKey` property. If you set it to 0, then the data is sorted by the item's `Text` property. If you set this property to a number higher than 0, then the List View will sort based on the text within the `SubItems` properties.

The `SortOrder` property determines whether the data is sorted in an ascending or descending fashion. Set this property to 0 to sort in ascending order. Set it to 1 to sort in descending order.

### The Column Headers Tab

The `Index` field is incremented every time you add a `ColumnHeader` object to the List View Control.

When you fill in the `Text` field of this tab, Visual Basic will set the `Text` property of the `ColumnHeader` object with an index specified in the `Index` field above.

You can set the `Alignment` field to one of three values:

- 0 - `lvwColumnLeft`
- 1 - `lvwColumnRight`
- 2 - `lvwColumnCenter`

These determine the position of the text within the `ColumnHeader` object.

The value entered in the `Width` field will set the Column Header's `Width` property. This will set the width of the Column Header specified in the `Index` field.

The `Key` property determines the Column Header's unique by within the collection of the Column Headers. This value can be either numeric or text. What you enter does not matter as long as it is unique.

The Tag property sets the Column Header's Tag property. This property can contain any miscellaneous data that you want associated with the Column Header.

You set the IconIndex field to a number that indicates the index of the desired icon within the associated Image List control. For example, if you have three icons in an Image List and you want the third icon to be displayed on this Column Header, you would set this field to 3.

NOTES

## Events

This is the list of events supported by the List View Control.

AfterLabelEdit	DragOver	KeyUp	OLEDragDrop
BeforeLabelEdit	GotFocus	LostFocus	OLEDragOver
Click	ItemCheck	MouseDown	OLEGiveFeedback
ColumnClick	ItemClick	MouseMove	OLESetData
Db1Click	KeyDown	MouseUp	OLEStartDrag
DragDrop	KeyPress	OLECompleteDrag	Validate

As you can see, many of the events for this control are the similar to those of the Tree View control.

Again, the AfterLabelEdit() event is triggered after you perform a label editing function on an object in the List View, called a ListItem. This event is useful if you want to check the name of the ListItem to make sure that it is valid. Its compliment, the BeforeLabelEdit() event, is triggered right before the ListItem goes into edit mode.

The ColumnClick() event is triggered when a user clicks one of the column headers. A column header is the button that sits at the top of a column and describes the contents of that column. In report view in Windows Explorer, you can see that the column headers are labeled Name, Size, Type and Modified. By inserting code in this event, you can make your List View control re-sort the data or eve re-order the columns.

The ItemClick() event is one of the most important events in this control. You can use this event to retrieve information about the ListItem that was clicked, or any other function that your application requires. It is used in much the same way as the Tree View's NodeClick() event. OLECompleteDrag() is triggered when OLE data is dropped on the control or the OLE drag-and-drop operation is canceled.

The OLEDragDrop() event is triggered when the OLEDropMode property is set to 1 - Manual and OLE data is dropped on the control. You can place code in the event to determine what the control should do when data is dropped. You could place code in this event to make the control move data, but not copy it. Or you could have it copy the data, but not copy it. Or you could have it copy the data, but notmove it. The functionality depends on the needs of your application.

The OLEDragOver() event it triggered when OLE data is dragged over the control. After this event is triggered, Visual Basic will trigger the OLEGiveFeedback() event which will allow you to check the data and provide feedback to the user.

The OLEGiveFeedback() event is triggered after every OLEDragOver() event. It allows the Tree View control to provide feedback to the user, such as changing the

NOTES

- Windows Common Controls 6.0 from the list. Click OK to add the controls to your Toolbox.
7. Add an Image List to frmMain by double-click the Image List control in the Toolbox. When it is added to the form, move it to the lower-right corner of the form. Set its Name property to imlCategories.
  8. In the Property window, double-click the Custom field to open the property page for the control.
  9. Select the Images tab. Click the Insert Picture button to add an image to the list.
  10. When the Select picture dialog box appears, select Closed.bmp from the Common\Graphics\Bitmaps directory. Click the Open button to add the file to the control, then click OK.
  11. Add another Image List to frmMain. Set its Name property to imlItems. Move it next to imlCategories.
  12. Using the same methods as described in steps 8 – 10 add the image Leaf.bmp to the control and click the Open button.
  13. Click the OK button to close the property page.
  14. Add a Tree View control to frmMain. Set its Name property to twvCategories.
  15. In the Properties window, double-click the Custom property field. This will bring up the property page for the control.
  16. On the General Tab of the property page, set the Style property to 7 – twvTreeLinesPlusMinusPictureText, set the LabelEdit property to 1 – Manual the Indentation property to 283 and the ImageList property to imlCategories.
  17. Click the OK button to close the property pages.
  18. Move twvCategories to the upper-left side of the form.
  19. Add a List View control to frmMain and set its Name property to lvwItems.
  20. In the Properties window, double-click the Custom property field. This will bring up the property page for the control.
  21. Set the View property to 3 – lvwReport; the Arrange property to 2 – lvwAutoTop the LabelEdit property to 1 – Manual and the OLEDropMode property to 1 – OLEDropManual.
  22. Click the Image Lists tab to make it the active property page.
  23. Let the Normal field to imlItems. This will link it to the Items Image List control.
  24. Click the Column Headers tab to make it the active property page.
  25. Click the Insert Column button. This will add the first Column Header object to the collection.
  26. Set the text field to Control Name.
  27. Click the OK button to close the property pages.
  28. Move imlItems to the upper-right corner of the form.

29. Save your work by selecting File – Save Project from the Visual Basic menu.
30. Double-click frmMain to open its code window.
31. Add the following code to the Form\_Resize() event:

```
Private Sub form_Resize()

    Dim mid1 As Integer
    Dim mid2 As Integer

    mid1 = (ScaleWidth / 2) - 50
    mid2 = (ScaleWidth / 2) + 50

    If WindowState <> vbMinimized Then
        tvwCategories.Move 0, 0 mid1, ScaleHeight
        lvwItems.Move mid2, 0, ScaleWidth - mid2,
ScaleHeight
    End If
End Sub
```

The code for this example deserves some scrutiny. The first two statements dimension two variables, mid1 and mid2. The next two lines of code set these variables to values that indicate positions just to the left and right of the centerline of the form. By adding and subtracting 50 from these values, we can create a neat border between the two views.

The If...Then statement tells the program to execute the next lines of code only if the form is not minimized. If the form were minimized, then you would get an error when trying to move and size the controls.

In the Form\_Load() event, add the following code:

```
Private Sub Form_Load()
    Dim cat As node

    ' Add the nodes to the tree view
    With tvwCategories.Nodes
        Set cat = .Add(, , "root", "Objects", 1)
        Set cat = .Add(, , "root", tvwChild, , "Intrinsic",
1)
        Set cat = .Add(, , "root", tvwChild, , "Explorer",
1)
        Set cat = .Add(, , "root", tvwChild, , "Internet",
1)
    End With
End Sub
```

The first line creates a Node-type variable. This will allow us to work with the Nodes

## NOTES

collection inside of the List View control. The next line (With...) tells the Visual Basic computer to work specifically with the Nodes collection of tvwCategories.

The first line of code under the With statement adds a node at the root level of the tree. We set its text value to "Objects" because that is what this tree contains. The following three lines of code add child nodes (tvwChild) to the root object. Each node has its own description, one for intrinsic controls, another for the Explorer-style controls, and finally some Internet controls. Remember that these nodes are actually categories. That means that they will "contain" other objects.

Add the following code to the (General) (Declarations) section of frmMain:

```
Option Explicit
```

```
Private Sub ListExplorer()
```

```
    Dim itm As ListItem
```

```
    With lvwItems.ListItems
```

```
        .Clear
```

```
        Set itm = .Add(, , "Tree View", 1)
```

```
        Set itm = .Add(, , "List View", 1)
```

```
        Set itm = .Add(, , "Image List", 1)
```

```
        Set itm = .Add(, , "Toolbar", 1)
```

```
        Set itm = .Add(, , "Status Bar", 1)
```

```
    End With
```

```
End Sub
```

```
Private Sub ListInternet()
```

```
    Dim itm As ListItem
```

```
    With lvwItems.ListItems
```

```
        .Clear
```

```
        Set itm = .Add(, , "Web Browser", 1)
```

```
        Set itm = .Add(, , "Shell Folder View", 1)
```

```
        Set itm = .Add(, , "Inet", 1)
```

```
        Set itm = .Add(, , "Winsock", 1)
```

```
    End with
```

```
End Sub
```

```
Private Sub ListIntrinsics()
```

```
    Dim itm As ListItem
```

```
    With lvwItems.ListItems
```

```
        .Clear
```

```
        Set itm = .Add(, , "Picture", 1)
```

NOTES

```

Set itm = .Add(, , "Label", 1)
Set itm = .Add(, , "Text Box", 1)
Set itm = .Add(, , "Frame", 1)
Set itm = .Add(, , "Command Button", 1)
Set itm = .Add(, , "Check Box", 1)
Set itm = .Add(, , "Radio Button", 1)
Set itm = .Add(, , "Combo Box", 1)
Set itm = .Add(, , "List Box", 1)
Set itm = .Add(, , "Horizontal Scroll Bar", 1)
Set itm = .Add(, , "Vertical Scroll Bar", 1)
Set itm = .Add(, , "Timer", 1)
Set itm = .Add(, , "Drive List", 1)
Set itm = .Add(, , "Directory List", 1)
Set itm = .Add(, , "File List", 1)
Set itm = .Add(, , "Shape", 1)
Set itm = .Add(, , "Line", 1)
Set itm = .Add(, , "Image", 1)
Set itm = .Add(, , "Data", 1)
Set itm = .Add(, , "OLE", 1)

```

End With

End Sub

The first statement, Option Explicit, forces variable declaration within the project. The three subs are very similar. Each declares a ListItem variable, named itm. This variable is used to access the ListItems collection within lwItems.

The With... statement tells the compiler to work with the ListItems collection of the List View control. The next command, Clear tells the List View to clear its ListItems collection. This removes other controls if any already exist within the collection. Finally, the next commands add ListItems to the collection.

Finally, add the following code to the NodeClick() event of lwCategories:

```

Private Sub tvwCategories_NodeClick(ByVal Node As
    _ComctlLib.Node)
    Select Case Node
        Case Is = "Intrinsic"
            ListIntrinsics
        Case Is = "Explorer"
            ListExplorer
        Case Is = "Internet"
            ListInternet
    End Select
End Sub

```

Save your project by selecting File – Save Project from the menu. Press the F5 key to run the project.

NOTES

## Using the Status Bar Control

### NOTES

The Status Bar control is the next important piece of the Windows Common Controls collection. It is used to report various bits of information to the user. It resembles the System Tray, found in the right side of the Windows Taskbar. It can also be found in Windows Explorer. It can reflect the system data and time, show icons or display statistics related to other controls, for example the number of files listed in a List View control.

### Properties

This is the list of status bar's properties;

Align	hWnd	OLEDropMode	Tag
Container	Index	Panels	ToolTipText
DragIcon	Left	Parent	Top
DragMode	MouseIcon	ShowTips	Visible
Enabled	MousePointer	SimpleText	WhatsThisHelpID
Font	Name	Style	Width
Height	Object	TabIndex	

The Name property should be set first. You can use the prefix sts. I prefer to name the status bar stsStatus.

The Panels property returns a reference to the collection of panel objects contained in the Status Bar control.

The Style property determines how the status bar is displayed. The allowed values are:

- 0 - sbrNormal      This setting shows multiple panels on the status bar.
- 1 - sbrSimple      This setting will show only one panel, which extends the width of the status bar.

The SimpleText property allows you to set or retrieve the value of the text in the panel when the Style property is set to 1 - sbrSimple.

### Events

This is the list of events supported by the Status Bar control.

Click	MouseDown	OLEDragDrop	OLEStartDrag
DbtClick	MouseMove	OLEDragOver	PaneClick
DragDrop	MouseUP	OLEGiveFeedback	PanelDbtClick
DragOver	OLECompleteDrag	OLESetData	

Since the status bar is used more to give you feedback, many of these events are not that important. Let's look at the PanelClick() and PanelDbtClick() events.

The PanelClick() event is triggered when the user clicks a panel. So what is a panel, you ask? A panel is a section of the status bar that contains either text or a bitmap, which may be used to reflect the status of an application. The PanelDbtClick() event is triggered when a user double-clicks a panel.

Neither of these actions is likely to occur and so these events won't be used that often. However, everything a control does depends on the design of your applications.

## Methods

This control only supports a few methods, listed below. None of them are particularly useful for the operation of the control.

Drag	OLEDrag	SetFocus	Zorder
Move	Refresh	ShowWhatThis	

NOTES

---

## PROJECTS

---

It is in this portion of the object model that deals with programmatic control of project components and source code.

### The VBProject Class

The VBA IDE is shared component, capable of hosting multiple projects at the same time. You would expect the object model to represent this. In fact, it does so by means of the VBProjects collection of the VBE object. Each VBProject object in the collection represents a loaded VBA project.

The VBProject class implements properties that map to those in the Project options dialog. For example, you can set and retrieve the Name, Description HelpFile and HelpContextID properties. Changing these through code changes the values in the options dialog and vice versa.

The class also implements several read-only properties that can give you additional information about the project. Specifically, the Mode property tells you whether the project is in Run, Break or Design mode. These states are represented by the integer values 0, 1 and 2 and by the constants vbext\_vm\_Run, vbext\_vm\_Break and vbext\_vm\_Design, respectively. Furthermore, the Protection property returns the value 1 (vbext\_pp\_locked) if the project is password protected and 0 (vbext\_pp\_none) if it is not. Finally, the Saved property tells you whether the project has changed since the last time it was saved. A True value indicates that no changes have been made, while False indicates that changes have been made but not yet saved.

### The Reference Class

Part of VBA project is the set of type library references for any Automation components it uses. Simple projects will have but a few references, such as those for VBA itself, Automation and the host application. Complex projects – those that use additional Automation components or ActiveX controls – will have numerous references. You can manage references interactively using the References dialog shown in figure. You can also manipulate them programmatically using the References collection of the VBProject class.

As you might expect, the References collection contains one element for each reference in a particular project. The Reference class itself defines properties that describe the reference, such as Name, Major and Minor version numbers, Description, FullPath, Guide (for type library references), Type, Builtin and IsBroken. The IsBroken property is of particular interest because when a reference is broken (because a type

library or an application has been moved or deleted), the VBA project containing it won't compile. When you determine that a reference is broken, you can delete and re-create it using methods of the References collection.

Listing shows a procedure that prints information on the active project's references to the Immediate window. Note the references to VBA and the host application (Microsoft Excel in this case).

NOTES

### Printing Reference Information to the Immediate Window

```

Sub dhPrintReferences ()
    Dim ref As Reference
    'Iterate the references of the active project
    For Each ref In Application.VBE. _
ActiveVBProject.References
        ' Use each reference and print:
        ' Name and version
        ' Description
        ' Built-in or custom?
        ' Project or typelib?
        ' Broken or intact?
        ' Full path
        ' GUID
    With ref
        Debug.Print .Name & " " & .Major & "." & .Minor
        If Not .IsBroken Then
            Debug.Print " " & .Description
        End If
        Debug.Print " "; IIF(.BuiltIn, "Built-in/", _ "Custom/
        ");
        Debug.Print Iif(.Type = vbext_rk_Project, _ "Project/
        ", "TypeLib/");
        Debug.Print Iif(.IsBroken, "Broken!", "Intact")
        Debug.Print " "; .FullPath
        Debug.Print " "; Iif(.Type = _
            Vbext_rk_TypeLib, .GUID, "")
    End With
    Next
End Sub

```

### Removing References

If a reference is broken, you can rebuild it using methods of the References collection. You can't use Reference class properties because they are all read only and are set when the reference is added to the project. Therefore, you must first delete the invalid reference using the References collection's Remove method. Remove accepts a pointer to a Reference object as an argument. Listing shows the dhRemoveAllBadrefs procedure, which removes all broken references from the active project.

## Procedure for Removing All Broken References

```

Sub dhRemoveAllBadRefs ()
Dim ref AS Reference
' Use the active P.ActiveVBProject
with Application.References
' Iterate through the references
For Each ref In References
If ref.IsBroken Then
References.Remove ref
End If
Next ref
End with
End Sub

```

NOTES

## Adding References

Once you've removed the offending reference, you can then add it back to the project. You can add a reference using one of two methods of the References collection: `AddFromFile` or `AddFromGuid`. (Of course, this works the same way for new references, as well.) Use `AddFromFile` to create a reference to DLL, an XE or another VBA project. For example, to add a reference to an Excel add-in, you might use code like this:

```

Application.VBE.ActiveVBProject.References.AddFromFile
"C:\Excel\Addins\MinMax.xla"

```

If the file does not exist and a path is specified, a run-time error occurs. If a path is specified, VBA searches for the file in the Windows and Windows\System directories, as well as in the current directory. `AddFromGuid` adds a reference to a type or other component based on its Globally Unique Identifier (GUID), which is stored in the Registry. You pass the GUID as a string, along with major and minor version numbers. VBA attempts to find the component in the Registry and if successful, creates a reference to it in the project. For example, to add a reference to Microsoft Access 2000's type library, you would use a statement like this:

```

Application.VBE.ActiveVBProject.References.AddFromGuid
"{4AFFC9A0-5F99-101B-AF4E-00A003F0F07}"

```

If VBA can't find the reference, it raises a run-time error. If the exact version doesn't exist but a more recent version does, VBA adds a reference to the newer version.

## Component Properties

What makes the `VBComponent` class truly useful is its collection of `Property` objects. Each `Property` object corresponds to a property of the particular component. These are the same properties that appear in the IDE's Properties window. You can use the `Properties` collection to examine the name and value of each property. List the following procedure, `dbDumpProps`, that does just that. It accepts a pointer to a `VBComponent` object as an argument and uses a `For Each` loop to examine each of the

properties. You can call the procedure from the Immediate window, as the following code illustrates:

```
Call dhDumpPProps (Application on VBE.  
ActiveVBProject.VBComponent)
```

NOTES

### Printing VBComponent Property Values

```
Sub dbDumpPProps (vbc As VBComponent)  
    On Error GoTo HandleError  
    Dim prp As Property  
    Dim var As Variant  
    Dim fReadingValue As Boolean  
    Const dhcPadding = 25  
    ' Iterate the properties of the given  
    ' component and print the names and values  
    For Each prp In vbc.Properties  
        ' Use each property  
        With prp  
            ' Print the property name, padded  
            ' with spaces  
            If Len(.Name) >= dhcPadding Then  
                Debug.Print .Name & " ";  
            Else  
                Debug.Print .Name & _  
                    Space(dhcPadding - Len(.Name));  
            End If  
            ' Set a flag indicating we're about  
            ' to try to read the actual value  
            fReadingValue = True  
            ' If this is an indexed property,  
            ' print the number of indices  
            If .NumIndices > 0 Then  
                Debug.Print "<indexed (" & _ .NumIndices &  
                    ")>"  
                If the value is an object, just print  
                "<object>"  
                If IsObject(.Value) Then  
                    Debug.Print "<object (" & _ TypeName(prp.Object)  
                        ")>"  
                If the value is an array, print  
                each element  
                If IsArray(.Value) Then  
                    For Each var In .Value
```

```

        Debug.Print var,
    Next
    Debug.Print
    \ If the value is not an object
    \ or an Array, just print it
Else
    Debug.Print prp.Value
End If
    \ Rest flag
    fReadingValue = False
End With
NextProp:
    Next
ExitHere:
    Exit Sub
HandleError:
    \ If we were trying to read the value,
    \ print the error we got and move on
    If fReadingValue Then
        Debug.Print "<error " & Err.Number & _ ": "
        & Err.Description & ">"
        Resume NextProp
    \ Otherwise, bail out
    Else
        MsgBox Err.Description, vbExclamation, _
        "Error " & Err.Number
        Resume ExitHere
    End If
End Sub

```

NOTES

While the `dhDumpProps` procedure might seem needlessly complex, it actually is not. All the code is necessary due to the intricacy of a VBA Property object. To fully understand this, let's look at what the procedure does with each property. After printing the property name, along with some padding to make the output look nice, `dhDumpProps` sets a Boolean flag variable that indicates it is about to try to read the property's value. The procedure does this so that if an error occurs, the error handler can skip to the next property rather than abort the entire procedure. For some reason, trying to read the value of certain properties results in runtime errors, despite efforts to trap for these cases.

## Indexed Properties

After determining that a property value can be read, a series of `If` and `ElseIf` statements try to determine what type of property the current `Property` object is and how best to deal with it. The first `If` statement checks the property's `NumIndices` property. Some

## NOTES

component properties are indexed, which means that to read their values, you must supply up to four index values. An example of an indexed property is the Colors property of an Excel Workbook object. The Colors property is made up to 56 separate values representing the individual RGB color values used for the workbook's palette. You can write VBA code to set or retrieve any one of these values. To do so, you must use the Property object's IndexedValue property, passing a number from 1 to 56. For example:

```
Application.VBE.ActiveVBProject.
  VBComponents("ThisWorkBook").Properties("Colors").
  IndexedValue(2) = RGB(255, 255, 0)
```

Since `dhDumpProps` is a generic procedure and doesn't know what type of component it is manipulating. When it comes across an indexed property, it simply prints the string "<indexed>", along with the number of indices. If you were writing VBA code to manipulate a specific component type, you would certainly want to use the IndexedValue property with particular index values.

### Object Properties

Next, the procedure uses the VBA `IsObject` function to determine whether the current Property object's Value property is itself an object. You will find that many component properties are objects with their own sets of properties and methods. Again, since `dhDumpProps` is a generic procedure, it simply prints the string "<object>" and the object type after the property name. If you know the type of object being returned, you can manipulate the object's properties and methods. However, here's where things get a bit strange. The VBA documentation states that if a property value returns an object, you must use the Property object's Objects property to access the returned object's properties and methods. For instance, to manipulate the font properties of a VBA user form, you should be able to use code like this:

```
Application.VBE.ActiveVBProject.VBComponents("UserForm1").
  Properties("Font").Object.Size = 10
```

However, in our testing, this did not work. VBA generated a compile-time error, "Method or data member not found" on the Size property. What did work was using the Property object's Value property, although not as you'd expect. You might think you could use it in place of the Object property in the preceding statement. In reality, the Value property returned a collection containing the properties of the Font object. We were then able to use a statement like this one:

```
Application.VBE.ActiveVBProject.VBComponents("UserForm1").
  Properties("Font").Value.Item("Size") = 10
```

Note that the Item method is required when passing the property name (Size). While we can't explain why VBA behaves like this with object properties, it at least appears to be consistent. We did find that the Object property worked when we assigned an object pointer to it. For instance, we were able to set the Picture property of a VBA user form using the following statement:

```
Set Application.VBE.ActiveVBProject.
  VBComponents("UserForm1").Properties("Picture").
```

```
Object = LoadPicture("C:\WINDOWS\WAVES.BMP")
```

LoadPicture loads an image file from disk and returns a pointer to it.

---

## FORMS

---

### NOTES

The most basic object you will be working with in Visual Basic is the Form object, which is the visual foundation of your application. It is basically a window that you can add different elements to in order to create a complete application. Every application you can see on the screen is based on some type of form. In the following sections, we will go through each of the parts that make up this object so you can become familiar with what forms are and how you will use them.

All of these elements should be familiar to you if you have spent any time using the Windows operating systems. Let's take a look at how these features are referenced in Visual Basic.

### The Border

The form's border is what gives the form its elasticity. Depending on the type of form you want to display, you can program the border to be fixed, sizable, or even nonexistent. These features can be set with the `BorderStyle` property.

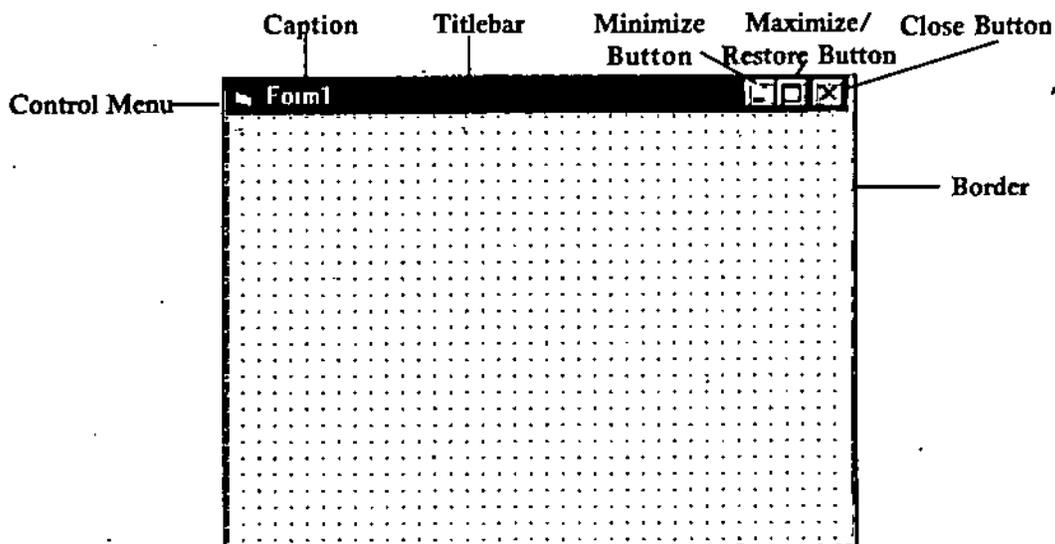
### The Title Bar

The title bar is the coloured bar on the top of most forms. If your desktop colour scheme is set to the Windows default scheme, this bar will be blue. You can use the title bar to drag the window around the screen. In addition, double clicking it will alternately maximize and restore the form.

### The Caption

The form's caption is the text you see in the form's title bar. It can be used to identify the name of the application, the current function of the form, or as a status bar. What you put in the caption depends on what your program is trying to achieve.

If you set a form's `BorderStyle` property to `None`, then the caption, along with the



whole title bar, is hidden. You can set the caption to display the text you want by setting the form's Caption property in the Properties window.

NOTES

### The Control Menu

The Control menu is a simple menu that allows you to restore, move, resize, minimize, maximize, and close a form. To enable this button on your form, set the form's ControlBox property to True in the form's Properties window.

### The Minimize Button

The Minimize button is used to minimize the current form, that is, move it out of the way, to the Windows Taskbar. To enable this button on your form, set the form's MinButton property to True in the form's Properties window.

### The Maximize/Restore Button

The Maximize button has two purposes. If the form is in its normal state, that is, its normal size, you can click the Maximize button to automatically expand the current form to the size of the screen or container of the form. A form's container is also known as a multiple document interface (MDI) form. If the form is maximized, you can click this button again to restore the form to its original size. To enable this button on your form, set the form's MaxButton property to True in the Properties window.

### The Close Button

The Close Button's sole purpose is to close the current window. In Visual Basic, you can control whether the Close button is visible to the user with the ControlBox property. The Close button will not be visible if the Control box is not visible. If you decide not to enable the Close button or the Control box, then you must provide a way for the form to unload. This can be done automatically, or by using a menu or a button to close the form.

### Working with Form Properties

The following shows the properties for a form object:

ActiveControl	ActiveForm	Appearance	AutoRedraw
BackColor	BorderStyle	Caption	ClipControls
ControlBox	Controls	Count	CurrentX
CurrentY	DrawMode	DrawStyle	DrawWidth
Enabled	FillColor	FillStyle	Font
FontBold	FontItalic	FontName	FontSize
FontStrikethru	FontTransparent	FontUnderline	ForeColor
HDC	Height	HelpContextId	HWnd
Icon	Image	KeyPreview	Left
LinkMode	LinkTopic	MaxButton	MouseIcon
MousePointer	Movable	Name	NegotiateMenus

Picture	ScaleHeight	ScaleLeft	ScaleMode
ScaleTop	ScaleWidth	ShowInTaskbar	Tag
Top	Visible	WhatsThisButton	WhatsThisHelp
Width	WindowState		

NOTES

### Working with Multiple Document Interface (MDI) Forms

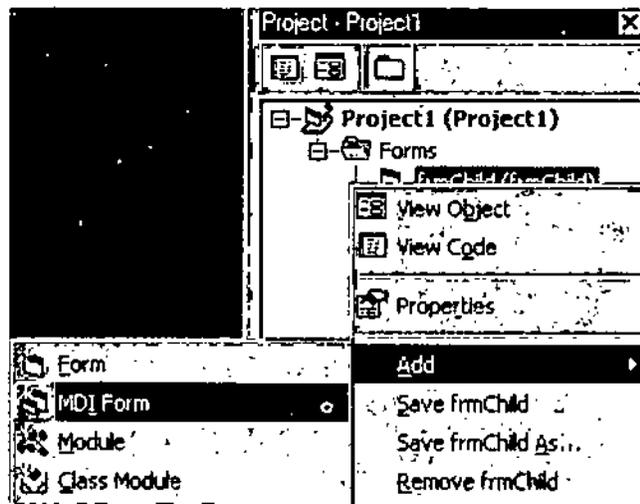
An MDI lets you open windows within a parent container window. If you look at a typical word processor – Word for Windows is a classing example – you can have many documents open simultaneously within the main window. This main window serves as the container- it contains multiple forms. The MDI application was originally developed back when previous versions of Windows were predominant. The multiple document nature allowed users to open more than one file at a time, without having to open several copies of the program itself. This not only saves time, but it also saves memory.

MDI interfaces are more commonly used for document-centric applications, like word processors and paint programs. A program is said to be document-centric when the main objects that you work on are documents. If you plan to allow your users to work on several similar forms at one time form within an application, you should use the MDI model. Visual Basic makes it very simple to create an MDI application. ,

### Creating an MDI

To create an MDI application, you need at least two forms in the application. One is the parent or containing form and the second is the child or contained form. To have more than one type of child form, you can add further forms to project. But you only need one child from for the simplest of MDI projects. Here's how it's done:

1. Start a new project by selecting File – New Project. Select Standard EXE as the project type if you have the Project Wizard enabled.
2. You will already have a form in the project. Set its Name property to frmChild and its Caption property to MDI Child.
3. To create the MDI parent form, right-click the Forms folder in the Project Explorer and select Add – MDI form. In the Form Wizard appears, select MDI Form.
4. Set the Name Property to frmMDI and the Caption property to MDI Parent.
5. Right-click Project1 in the Project Explorer and select Project1 Properties from the pop-up menu. Set the Startup Object list to frmMDI. If you omit this, the application will start with the child from showing.





**Procedure for Removing All Broken References**

```

Sub dhRemoveAllBadRefs()
    Dim ref AS Reference
    ' Use the active project
    with Application.VBE.ActiveVBProject
    ' Iterate through the references
    For Each ref In .References
        ' If reference is broken, remove it
        If ref.IsBroken Then
            .References.Remove ref
        End If
    Next
    End with
End Sub

```

NOTES

**Adding References**

Once you've removed the offending reference, you can then add it back to the project. You can add a reference using one of two methods of the References collection: `AddFromFile` or `AddFromGuid`. (Of course, this works the same way for new references, as well.) Use `AddFromFile` to create a reference to DLL, an EXE or another VBA project. For example, to add a reference to an Excel add-in, you might use code like this:

```

Application.VBE.ActiveVBProject.References.AddFromFile
    _"C:\Excel\Addins\MinMax.xla"

```

If the file does not exist and a path is specified, a run-time error occurs. If no path is specified, VBA searches for the file in the Windows and Windows\System directories, as well as in the current directory. `AddFromGuid` adds a reference to a type library or other component based on its Globally Unique Identifier (GUID), which is stored in the Registry. You pass the GUID as a string, along with major and minor version numbers. VBA attempts to find the component in the Registry and if successful, creates a reference to it in the project. For example, to add a reference to Microsoft Access 2000's type library, you would use a statement like this:

```

Application.VBE.ActiveVBProject.References.AddFromGuid
    _"{4AFFC9A0-5F99-101B-AF4E-00A003F0F07}", 9, 0

```

If VBA can't find the reference, it raises a run-time error. If the exact version specified doesn't exist but a more recent version does, VBA adds a reference to the more recent version.

**Component Properties**

What makes the `VBComponent` class truly useful is its collection of `Property` objects. Each `Property` object corresponds to a property to the particular component. These are the same properties that appear in the IDE's Properties window. You can iterate the collection to examine the name and value of each property. Listing shows a procedure, `dbDumpProps`, that does just that. It accepts a pointer to a `VBComponent` object as an argument and uses a `For Each` loop to examine each of the component's

properties. You can call the procedure from the Immediate window, as the following code illustrates:

```
Call dhDumpPRops (Application.VBE. _
    AcriveVBProject.VBComponents(1))
```

## NOTES

## Printing VBComponent Property Values

```
Sub dbDumpPRops(vbc As VBComponent)
    On Error GoTo HandleError
    Dim prp As Property
    Dim var As Variant
    Dim fReadingValue As Boolean
    Const dhcPadding = 25
    ' Iterate the properties of the given
    ' component and print the names and values
    For Each prp In vbc.Properties
        ' Use each property
        With prp
            ' Print the property name, padded
            ' with spaces
            If Len(.Name) >=dhcPadding Then
                Debug.Print .Name & " ";
            Else
                Debug.Print .Name & _
                    Space(dhcPadding - Len(.Name));
            End If
            ' Set a flag indicating we're about
            ' to try to read the actual value
            fReadingValue = True
            ' If this is an indexed property,
            ' print the number of indices
            If .NumIndices > 0 Then
                Debug.Print "<indexed (" & _ .NumIndices &
                    ")>"
                ' If the value is an object, just print
                ' "<object>"
                ElseIf IsObject(.Value) Then
                    Debug.Print "<object (" & _ TypeName(prp.Object)
                        & ")>"
                ' If the value is an array, print
                ' each element
                ElseIf IsArray(.Value) Then
                    For Each var In .Value
```

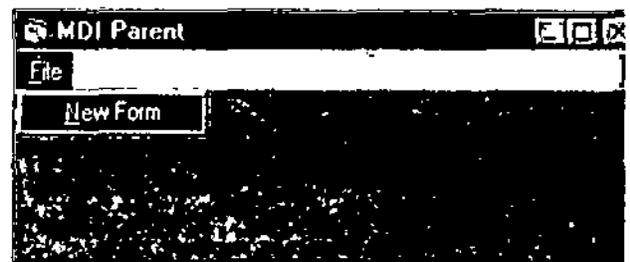
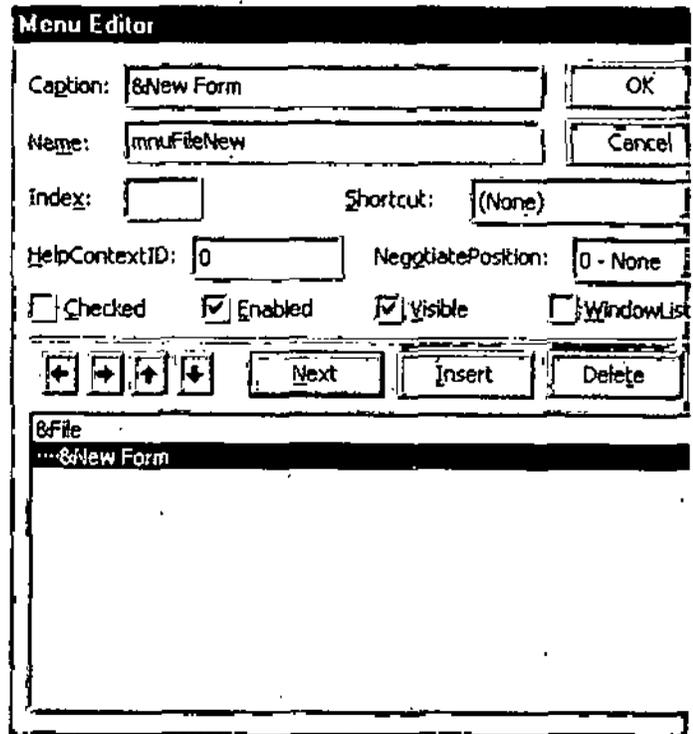
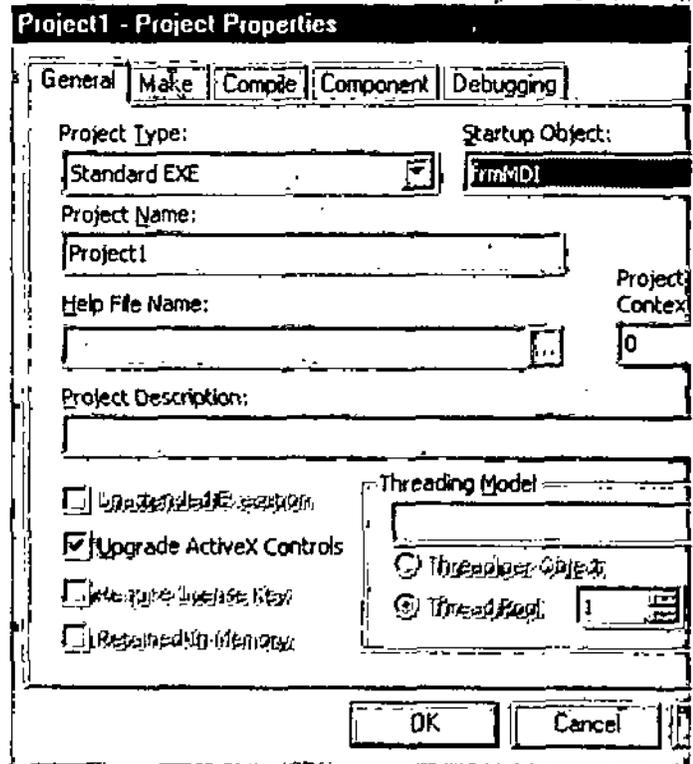


NOTES

6. Select frmChild from the Project Explorer. Set the form's MDI Child property to True. This will cause this form, which is the child, to rest inside of the MDI Parent container.
7. Select frmMDI from the Project Explorer.
8. Start the Menu Designer by selecting Tools - Menu Editor. You will see a window like the one in figure.

Now that you have created a MDI child form to reside inside the MDI parent window, let's create a simple menu for the form.

1. Type & File in the Caption field.
2. In the Name field, type mnuFileNew.
3. Click the Next button.
4. Click the right arrow button. This will indent this menu item.
5. Enter & New Form in the Caption field.
6. Type mnuFileNew in the Name field.
7. Click the OK button to close the Menu Editor.
8. The frmMDI form should now have a File menu on it. Select File - New form the MDI menu. This will open up the Code window.



9. In the `mnuFileNew_Click()` event, type the following lines of code:

```
Dim frm As New frmChild
Frm.Show
```

10. Save and run the project. You should now see the MDI.

The code creates (or instantiates) new copies of `frmChild` and shows them. It does this each time you click File – New. Try opening and closing a few child windows. You should have a functioning MDI application.

### Improving the MDI

But there are a few additions to make before it really resembles a commercial Windows MDI application. For example, each child form has the same caption, so it's impossible to tell them apart. Let's fix that. It would also be nice to tile or cascade the children. Further, it's normal to have a menu option (called a window list), which lets you switch easily to children that get hidden behind other children.

1. Open the Menu Editor and add a & Window menu title to the MDI parent, `frmMDI`. Turn on the check box for `WindowList` in the Menu Editor as you do so.
2. Using the same methods as in steps 5 and 6 in the previous section, add a Tile and a Cascade item to this menu title. Name them `mnuWindowTile` and `mnuWindowCascade`, respectively.
3. Click OK to close the Menu Editor.
4. Enter this code for the Click event of the `mnuWindowTile` object:

```
frmMDI.Arrange vbTileHorizontal
```

5. Enter this line for the `mnuWindowCascade` item:

```
frmMDI.Arange vbCascade
```

The `vbCascade` and `vbTileHorizontal` terms are built in Visual Basic constants. There can be obtained from Visual Basic's online help.

6. Change the code for the `mnuFileNew` menu item so that it looks like this:

```
Private Sub mnuFileNew_Click()
    Static Counter As Integer
    Dim frm As new frmChild
    Counter = Counter + 1
    Frm.Caption = "MDI Child" & Counter
    Frm.show
End Sub
```

Now save and run the application and run the notice the difference.

---

## MODULES

---

All coding stuffs whether written to operate or perform certain task in form, etc., are stored in modules. Your Visual Basic application contains minimum one form and this form is nothing but a container which holds all code known as form module.

NOTES

Might be your application can have more than one, three or many forms then you have several form modules in your application. Suppose you want to perform a certain common task through all forms then you can keep that common code in a separate standard module which is accessible to whole application or you can say all forms in an application.

NOTES

There are three types of modules:

- Form
- Standard
- Class

Form modules are stored with the extension .FRM. Form modules can contain procedure(s) which handles general procedure, events, and form level declaration of external procedures, variables, constants and types.

Standard modules are stored with the extension .BAS. Standard modules contain procedure(s) and declaration(s). They are accessible to whole application.

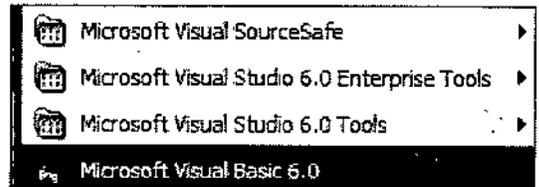
Class modules are stored with .CLS extension. This play vital role when you want to create new object(s). Objects are made up of properties and methods.

**Note:** Form is nothing but a class module with controls on it.

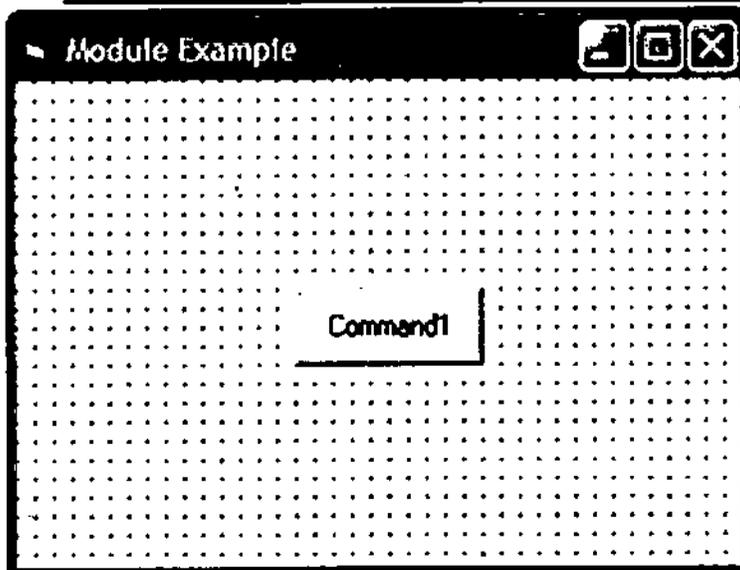
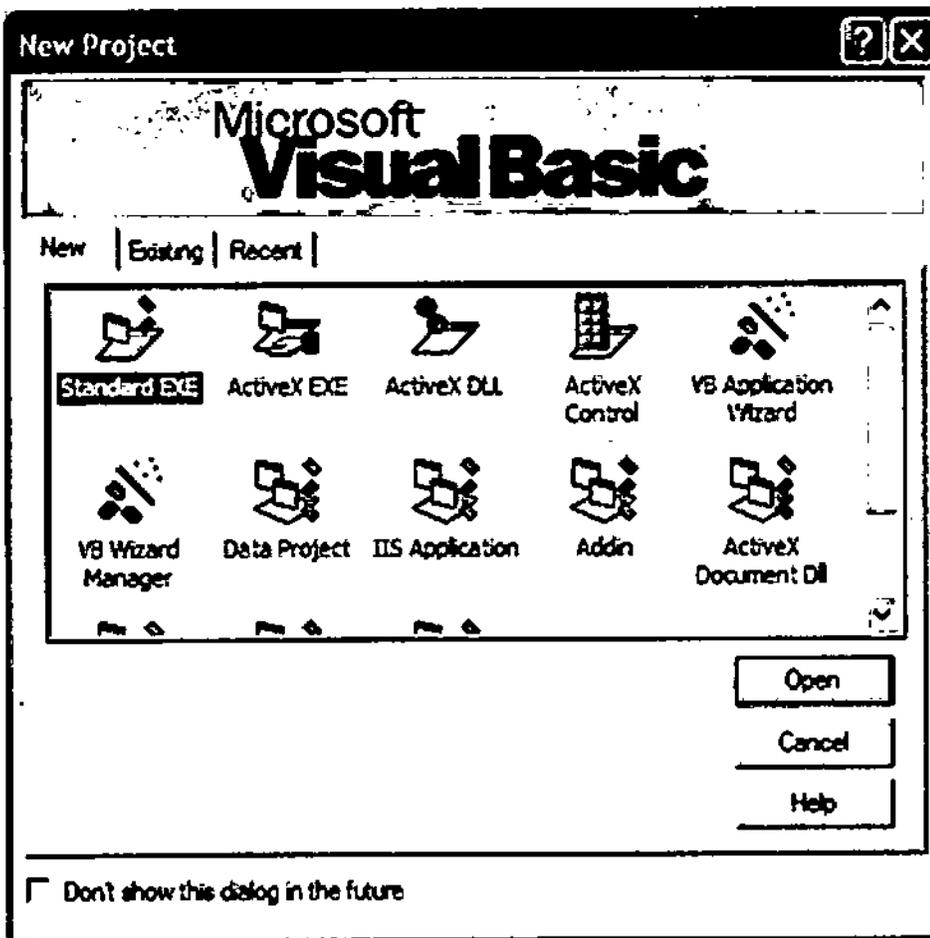
These three modules Form module, Standard module, Class module can hold declarations means you can keep variable, constant, and DLL procedure declarations at the module level of form.

**Example: Module**

1. To start Visual Basic 6, click on start button.
2. Position your mouse over All Programs.
3. Position your mouse over Microsoft Visual Studio 6.0 .
4. Click Microsoft Visual Basic 6.0
5. New Project dialog box appears:
6. Select Standard EXE.
7. Click Open button.
8. Select a Form by single clicking on it.
9. In Properties Window locate the Caption property and type, Module Example.
10. Add a button control to Form.
11. At present your Form looks like this:
12. Select the button control.
13. In Properties window locate the Caption property and type, Module Example.
14. Locate the Width property and type 1815.
15. Locate the Height property and type 615.

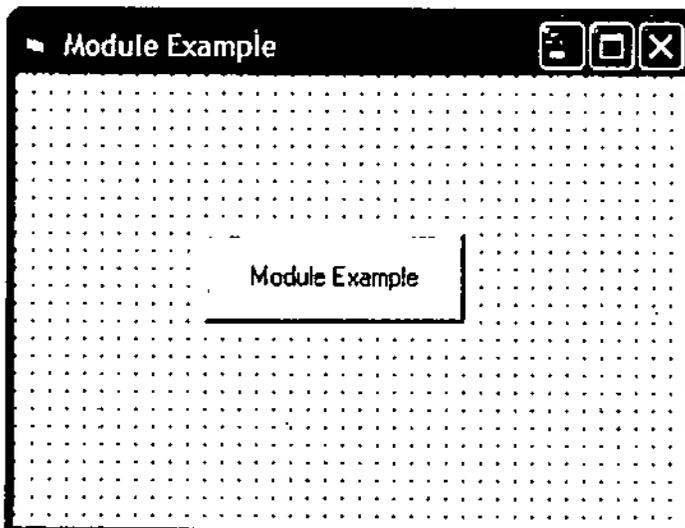


## NOTES

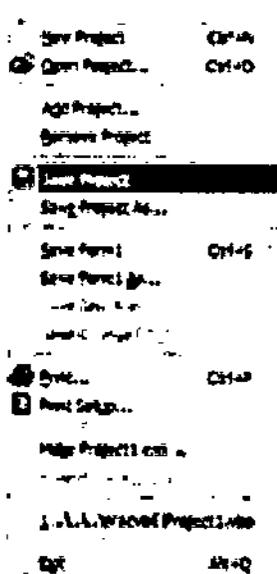


16. Locate the Left property and type 1320.
17. Locate the Top property and type 1080.
18. Now your Form looks like the one shown on the next page.
19. Now let us first save the project so on menu bar click on File.
20. Click Save Project.

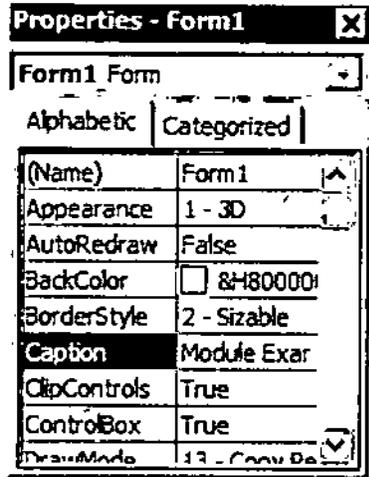
NOTES



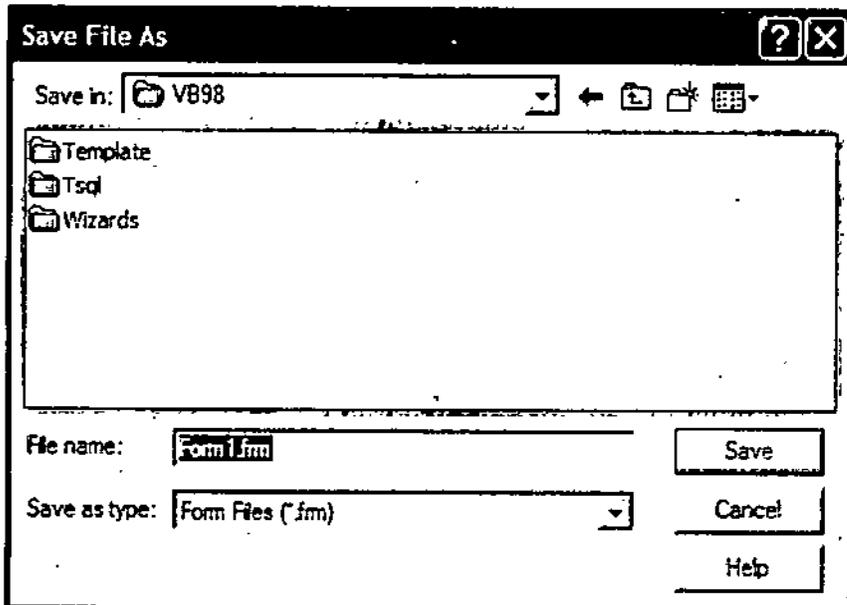
21. Save File As dialog box appears:
22. In Save in field select C: or any desired drive and folder.
23. Keep all default names and click on Save button.
24. Again Click on Save button to save Project with default name.



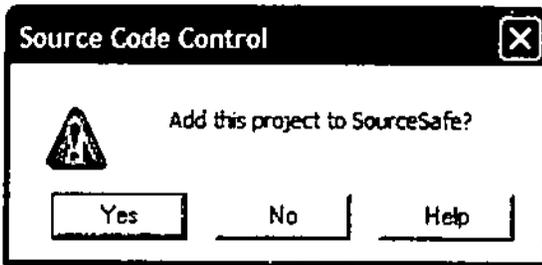
Note: You can give any other desired name of form or project.



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



25. Source Code Control dialog box appears:



NOTES

26. Click No.

27. In Project window right-click over project name.

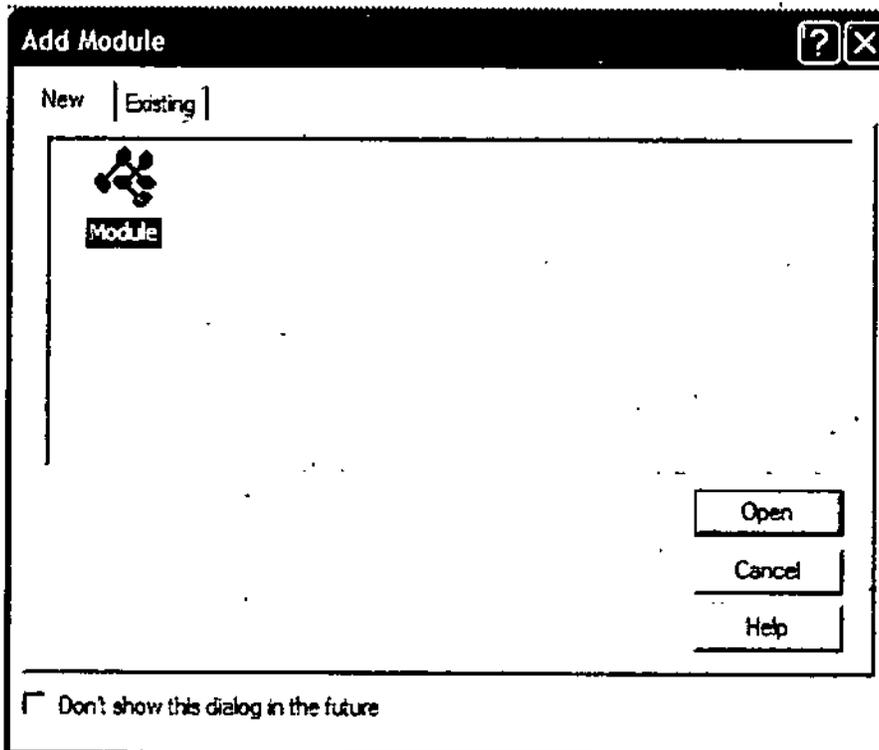
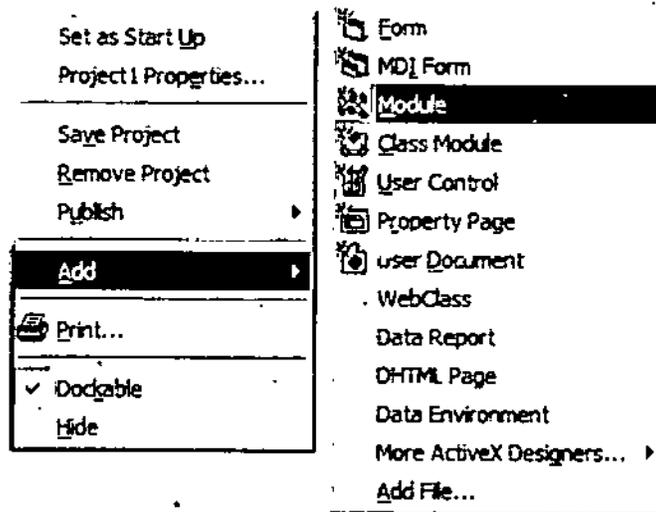
28. Position your mouse over Add item.

29. Click Module.

30. Add Module dialog box appears.

31. Click Open.

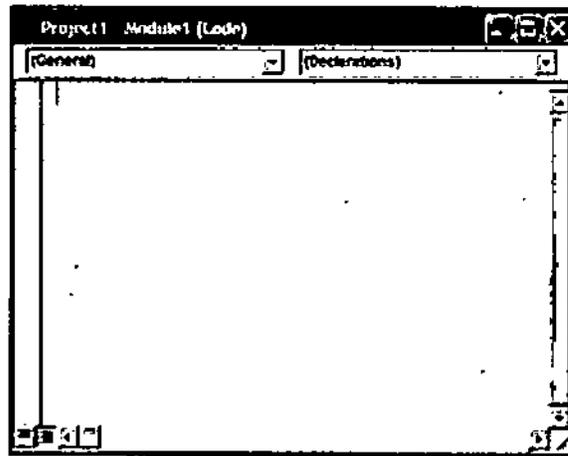
32. Code window of Module appears:



33. Type the following code:

```
Sub displaymessage()  
MsgBox ("I'm displaying this message from Module")
```

NOTES



End Sub

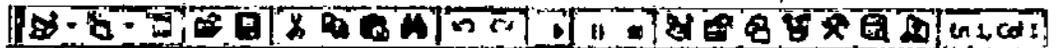
34. Now in Project window click on Form1 to select it.

35. Double-click on command button to open code window.

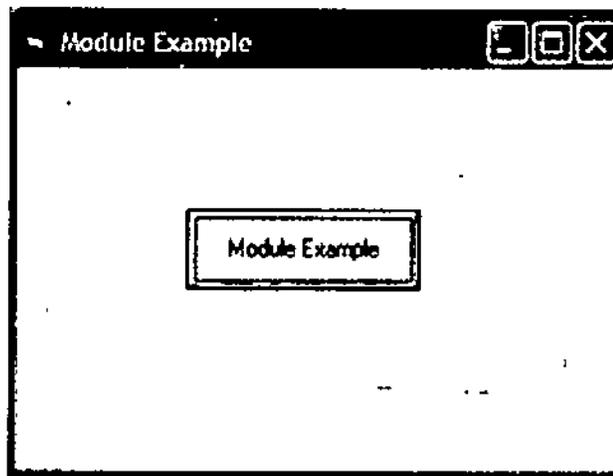
36. Type the following code.

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

37. Now run the program by clicking Start button.

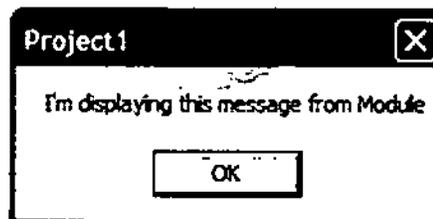


38. You will get the following output:



39. Click on button.

40. You will get the following message:



---

## FRAMES

---

Already discussed earlier.

---

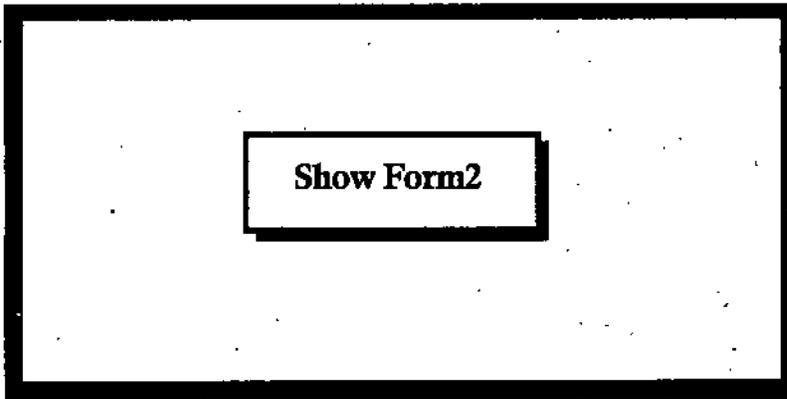
## PROJECT WITH MULTIPLE FORMS

---

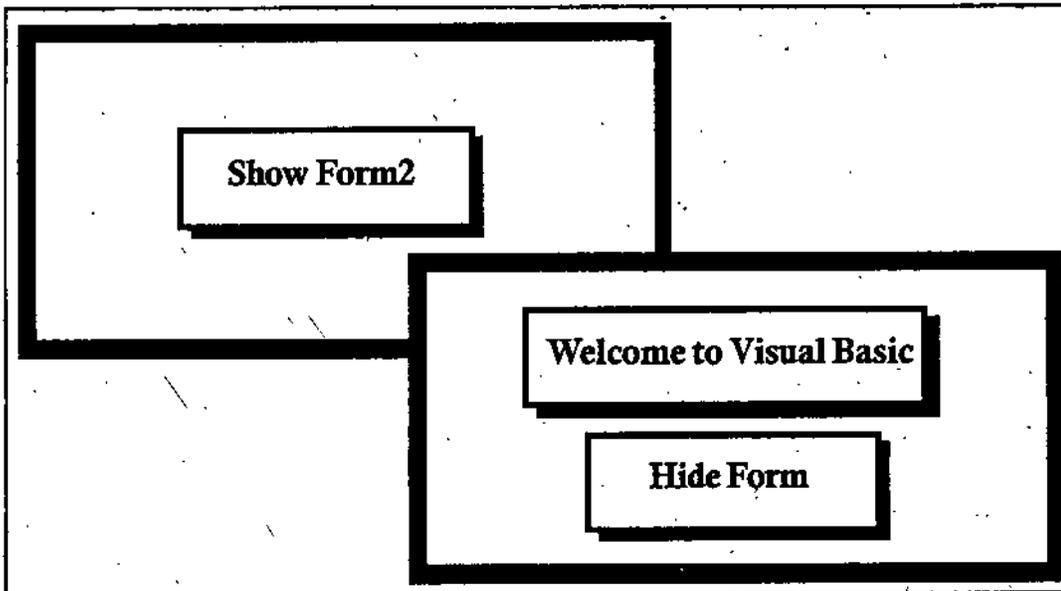
Supposing that you have designed your program and its an introductory form to welcome the user; a data-entry form to get data from the user, a summary form to display the data analysis results, a logon form to connect to the Internet—it's all there.

Suddenly it occurs to you—aren't Visual Basic projects organized into modules and forms? How does the code in one form reach the code in another—that is, how can the code in the analysis module read what the user has entered in the data-entry form? It's time to take a look at working with multiple forms.

For example, let's say that your introductory form looks something like that in the figure shown below:



When the user clicks the Show Form2 button, the program should display Form2 on the screen—and place the text. "Welcome to Visual Basic" in the text box in Form2 as well, as shown next. To be able to do that, we will need to reach one form from another in code.



NOTES

Create a Visual Basic project now. This project has one default form, Form1. To add another form Form2, just select the Add Form item in the Project menu; click on OK in the Add Form dialog box, Text1, to the new form, Form2.

In addition, add a command button to Form1 and give it the caption "Show Form2" and open the code for that button now:

## NOTES

```
Private Sub Command_Click ()
```

```
End Sub
```

When the user clicks the Show Form2 button, we will show Form2, which we do with Form2's Show() method:

```
Private Sub Command_Click()
```

```
Form2.Show
```

```
End Sub
```

Next, to place the text "Welcome to Visual Basic" in the text box, Text1, in Form2, we need to use that text box's fully qualified name: Form2.Text1, indicating that the text box we want is in Form2. We can use that text box's Text property this way to set the text in the box:

```
Private Sub Command_Click()
```

```
Form2.Show
```

```
Form2.Text1.Text = "Hello from Visual Basic"
```

```
End Sub
```

That completes the code for the Show Form2 button. Form2 has a button labeled Hide Form, and we can implement that by hiding Form2 in that button's even handler procedure:

```
Private Sub Command_Click()
```

```
Hide
```

```
End Sub
```

And that's it—we have written a program that handles multiple forms.

---

## DISPLAYING INFORMATION ON FORM

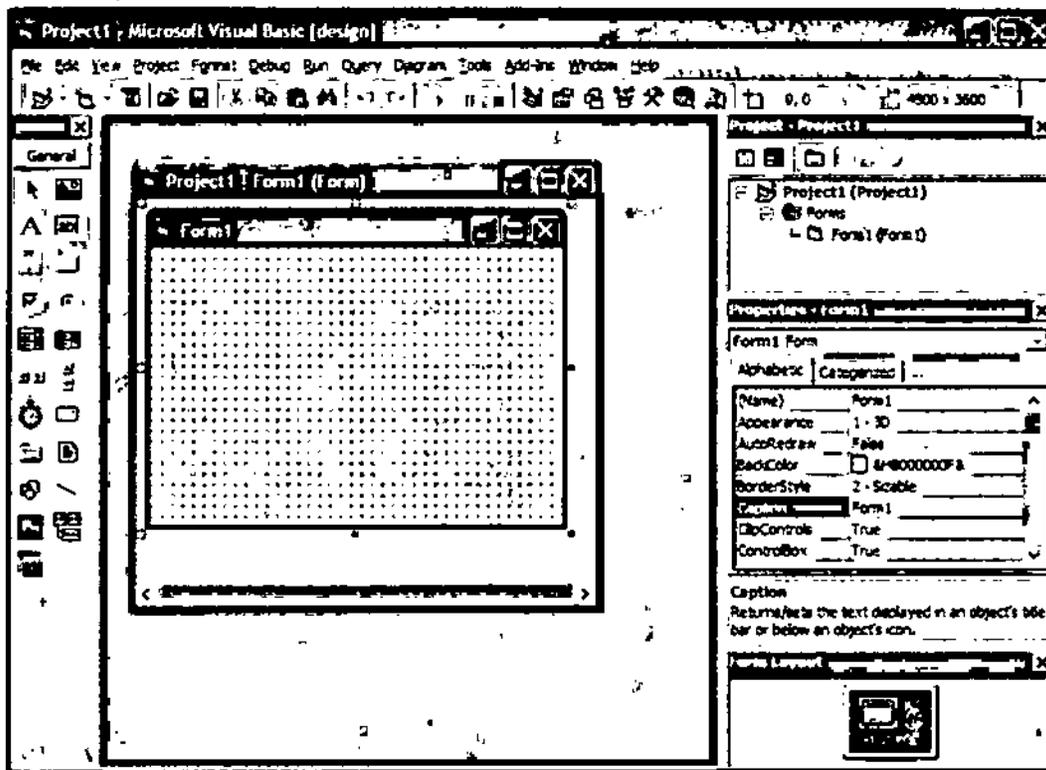
---

Form is the main place where you keep all controls. It is the area which is visible to user. In an application you can keep one or as many forms required according to business logic. It depends on company requirement, If Software demands that you have to maintain many forms then you can add forms in both design or run time.

First time when you start Visual Basic typical form in run time looks like the one shown on the next page.

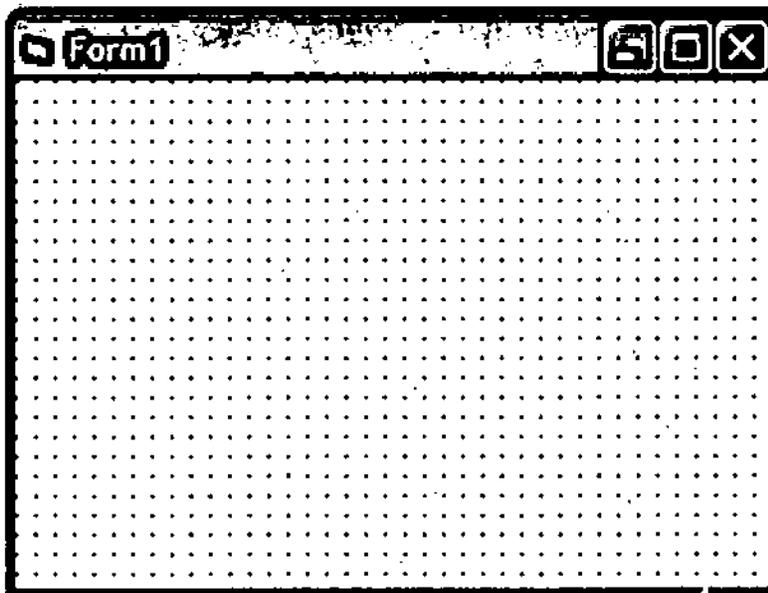
Note: If a form is not visible then you click on form's name or on menu bar click View then click Object.

On title bar of form you can see an icon at beginning. This is a Control menu. After that you can see by default forms name is appearing i.e. Form1. This can be manipulated



NOTES

using Form's Caption property. At last you can see Minimize button, Maximize/Restore button, Close button. In the following figure you can see complete Form object.



### Title Bar

In the above figure you can see the first colorful bar; this complete horizontal bar is known as Title bar. This bar can inherit according to your computer's desktop color scheme. By default its color will be blue. The basic purpose of Title bar is to display the text which you want to show. It can be form's name or any other name which does some functionality like Inventory form or Transaction form. When you double-click the title bar then alternately it will maximize or restore the form depends on its current state. Also you can use title bar to drag the window around the screen.

NOTES

### Border

By default form border is set to Sizable. But you can manipulate the form border in many ways. For this you can use BorderStyle property of form.

Following options are available to you:

- Fixed Single
- Sizable
- Fixed Dialog
- Fixed ToolWindow
- Sizable ToolWindow

**Note:** You can also remove title bar, Maximize button, Minimize button and Control menu box, like this your form will be visible only with border. For this set the form's MaxButton, MinButton and ControlBox properties to False.

### Fixed Single

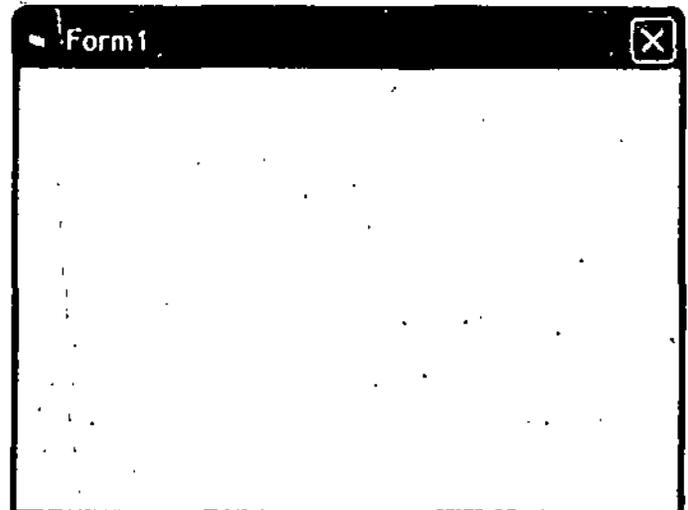
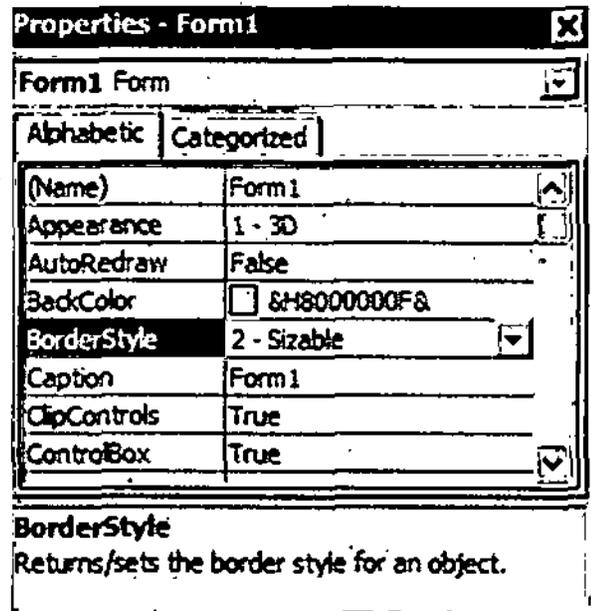
If you set the form border to Fixed Single then by default in run time user is not able to increase or decrease the size of form. You will notice this that Minimize and Maximized button is removed. In this you can only move or close the form. Notice in the figure here that only close button is visible in title bar. For closing the form you can also use Alt + F4 short cut key. But remember this that if your form's BorderStyle property is set to Fixed Single then later on you can add Maximize or Minimize button to the form. For this you have to set the MaxButton or MinButton property to True.

### Sizable

This is a default. Sizable form includes all basic functions like you can minimize, maximize or restore the form. Notice this in the figure here that in title bar all buttons are available.

### Fixed Dialog

Its behaviour is same like Fixed Single: property but main difference here is once the



form's border property is defined as Fixed Dialog then later on you are not able to add Maximize button or Minimize button.

### Fixed ToolWindow

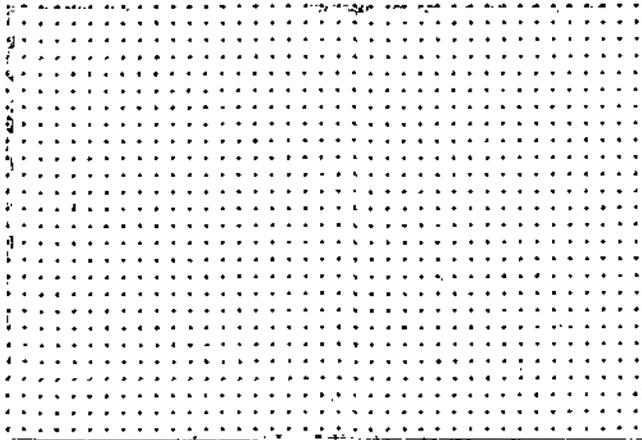
This property removes all basic features like Maximize, Minimize, Increasing or Decreasing of form etc. Also form does not appear in Taskbar.

### Sizable ToolWindow

It behaves same like Fixed ToolWindow property. Only difference is in this in run time user can increase or decrease the size of a form.

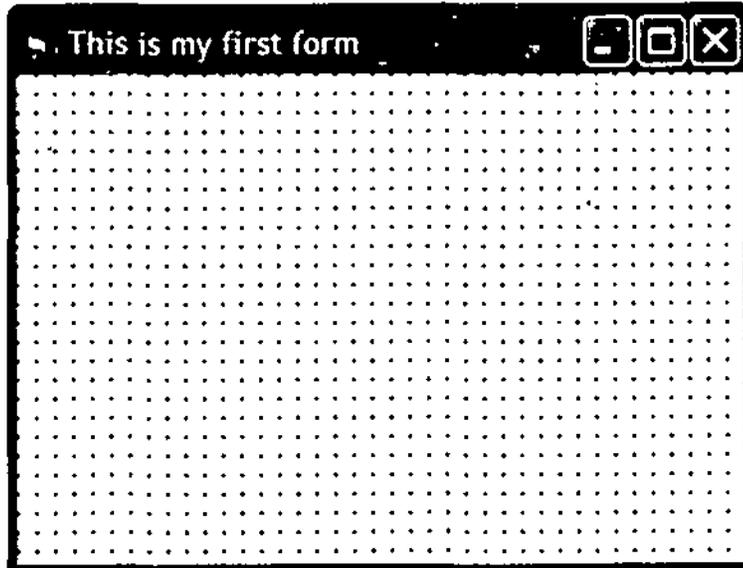
### None

If you will set the BorderStyle property to none then it will remove all Minimize, Maximize, Close button and form's border too. Notice this in the figure here.



### Caption

This property is used to display the information in form's title bar. In Properties windows locate the caption property and type "This is my first form". Notice this in the following figure that text in title bar of form has been changed.



### Minimize Button

It is located at the top-right corner of the form. This button is used to minimize the form to the Windows Taskbar.

### Maximize Button

This is used to maximize or restore the form.

### Close Button

This button is used to close the form.

### Control Menu

This allows you to move, minimize, maximize, close, restore and resize the form. For this set the ControlBox property to True.

NOTES

## Form Properties

In general properties describe the characteristics of an object. You can use properties to manipulate appearance, identity or behavior of an object.

Let me elaborate each property of form one by one.

NOTES

### Appearance Property of Form

It sets or returns the paint style.

#### Syntax

```
object.Appearance
```

#### For example:

```
Private Sub Form_Load()  
Form1.Appearance = 0  
End Sub
```

#### Property Settings of Appearance Property of Form:

0 - Flat

It paints form without visual effects.

#### Example:

#### 1- 3D

This is default setting. It paints with three-dimensional effects.

### AutoRedraw Property of Form

It sets or returns the output from a graphics method to a persistent graphics. Persistent graphics are automatically retained when certain kinds of screen events occur.

#### Syntax

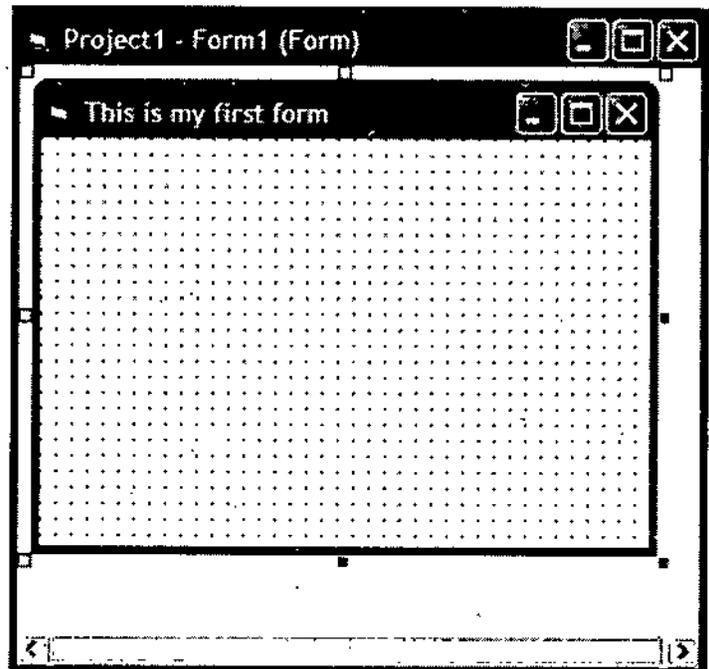
```
object.AutoRedraw  
= boolean
```

#### For example:

```
Private Sub Form_Load()  
Me.AutoRedraw = True  
End Sub
```

#### Property settings of AutoRedraw Property of Form:

True



If set to True then it does automatic repainting of a form.

**False**

This is a default setting. It disables automatic repainting.

## BackColor Property of Form

It sets or returns the background color.

### Syntax

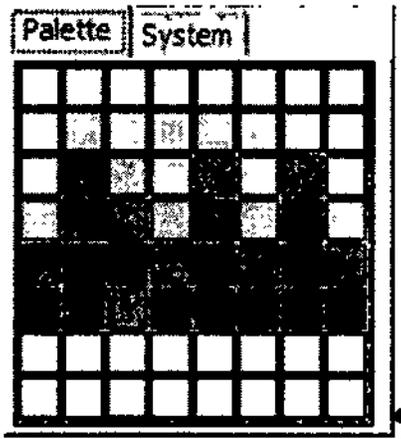
```
object.BackColor = color
```

### For example:

```
Private Sub Form_Load()  
Form1.BackColor = QBColor(9)  
End Sub
```

## Property Settings of BackColor Property of Form:

See the figure here.



## BorderStyle Property of Form

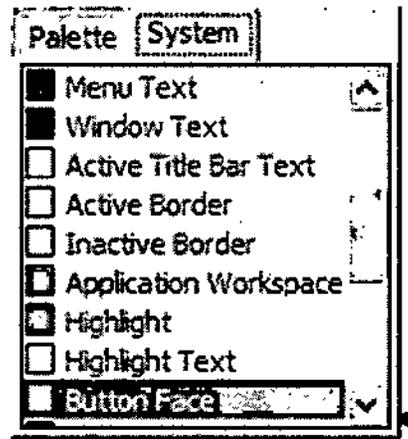
It returns or sets the border style

### Syntax

```
object.BorderStyle = value
```

## Property Settings of BorderStyle Property of Form:

See the figure here.



## Caption Property of Form

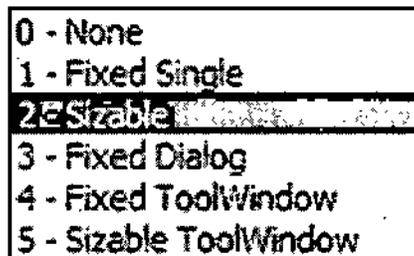
It displays the text in the title bar of Form

### Syntax

```
object.Caption = string
```

### For example:

```
Private Sub Form_Load()  
Me.Caption = "Hello VB"  
End Sub
```



## ClipControls Property of Form

It returns or sets a value that describes whether the graphics methods in Paint events repaint the whole object. If not then it determines whether the graphics method in Paint events repaint the newly exposed areas.

### Syntax

```
object.ClipControls
```

NOTES

### Property Settings of ClipControls Property of Form:



NOTES

### ControlBox Property of Form

It returns or sets a value that indicates whether a Control-menu box should be displayed on a form or not.

**Syntax**

```
object.ControlBox
```

### Property Settings of ControlBox Property of Form:



### DrawMode Property of Form

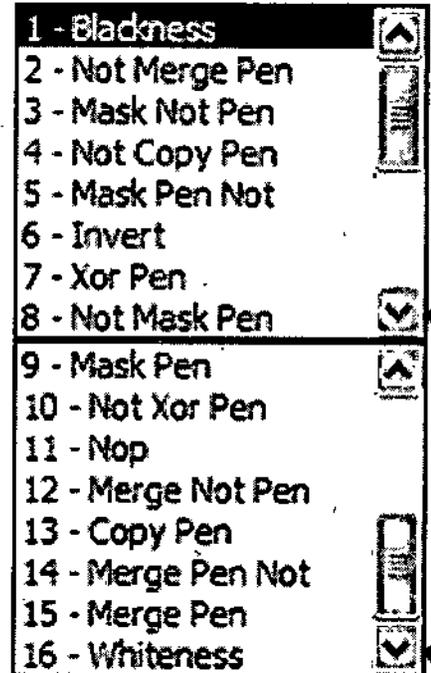
It returns or sets a value that defines the appearance of output from graphics method.

**Syntax**

```
object.DrawMode = number
```

### Property Settings of DrawMode Property of Form:

See the figure here.



### DrawStyle Property of Form

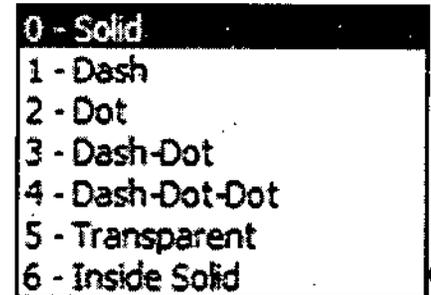
It returns or sets a value that defines the line style for output from graphics methods.

**Syntax**

```
object.DrawStyle = number
```

### Property Settings of DrawStyle Property of Form:

See the figure here.



### DrawWidth Property of Form

It sets or returns the line width for output from graphics methods.

**Syntax**

```
object.DrawWidth = size
```

**For example:**

```
Private Sub Form_Paint()  
    DrawWidth = 50
```

PSet (1150, ScaleHeight / 2)

End Sub

## Enabled Property of Form

It returns or sets a value that decides whether a form can respond to user-generated events or not.

### Syntax

object.Enabled = boolean

### For example:

```
Private Sub Form_Load()  
Me.Enabled = False  
End Sub
```

## Property Settings of Enabled Property of Form:



## FillColor Property of Form

It returns or sets the color has been used to fill in shapes.

### Syntax

object.FillColor = value

## Property Settings of FillColor Property of Form:

### Normal RGB colors

As shown here.

### System default colors

As shown here.

## FillStyle Property of Form

It returns or sets the pattern.

### Syntax

object.FillStyle = number

## Property Settings of FillStyle Property of Form:

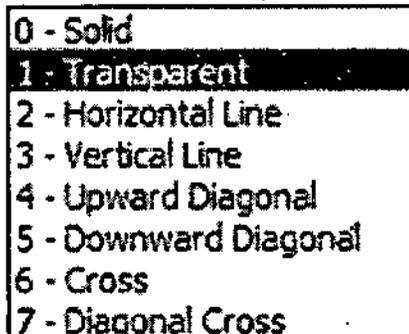
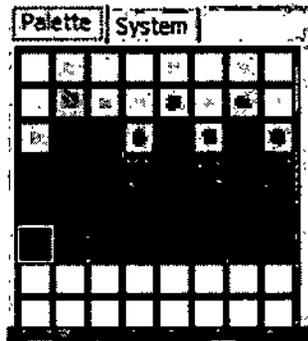
As shown here.

## Font Property of Form

It returns or sets a Font

### Syntax

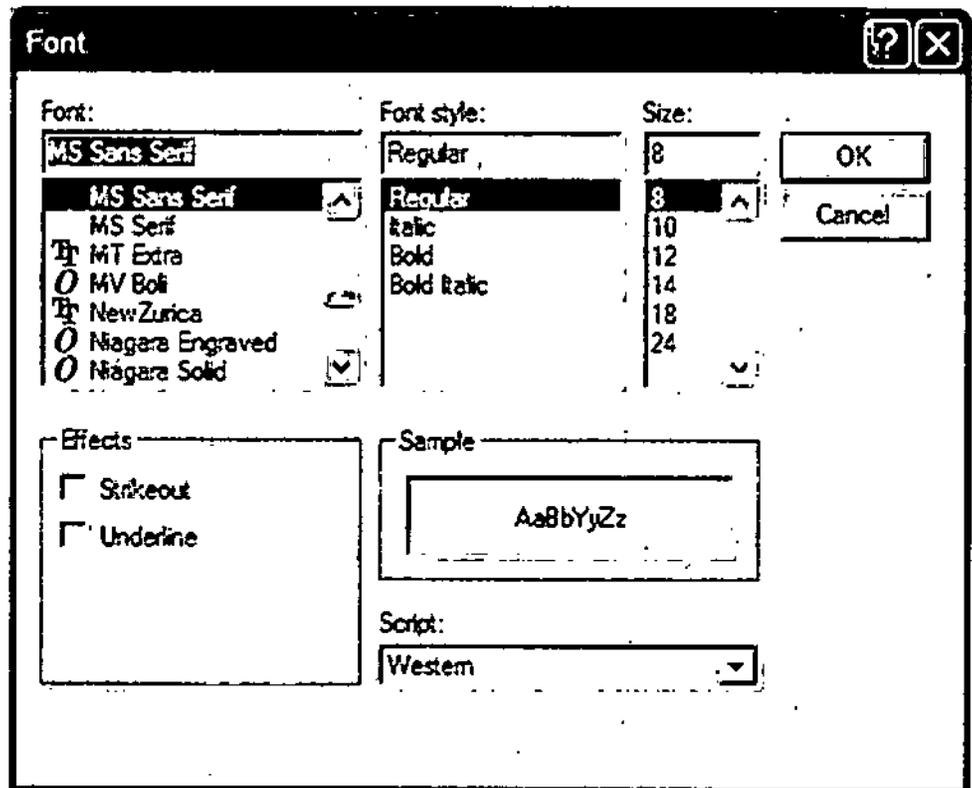
object.Font



NOTES

## Property Settings of Font Property of Form:

NOTES



### FontTransparent Property of Form

It returns or sets a value that decides whether graphics and background text Form are displayed in the spaces around characters or not.

#### Syntax

```
object.FontTransparent = boolean
```

### Property Settings of FontTransparent Property of Form:



### ForeColor Property of Form

It returns or sets the foreground color.

#### Syntax

```
object.ForeColor = color
```

#### Example:

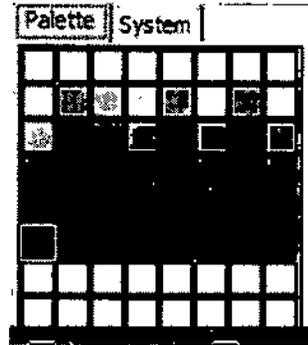
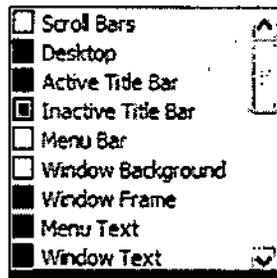
```
Private Sub Form_Paint()  
Form1.ForeColor = QBColor(9)  
Print "hello"  
End Sub
```

**Property Settings of ForeColor Property of Form:****Normal RGB colors**

As shown here.

**System default colors**

As shown here.



NOTES

**HasDC Property of Form**

It returns or sets a value that decides whether a unique display context should allocate or not?

**Syntax**

```
object.HasDC = boolean
```

**Property Settings of HasDC Property of Form:****Height Property of Form**

It describes the height of the form.

**Syntax**

```
object.Height = number
```

**Example:**

```
Private Sub Form_Load()  
Me.Height = 7050  
End Sub
```

**HelpContextID Property of Form**

It returns or sets an associated context number.

**Syntax**

```
object.HelpContextID = number
```

**Property Settings of HelpContextID Property of Form:**

0 It is a default means no context number has been specified.

> 0 An integer which tells a valid context number.

**Icon Property of Form**

This property sets the Icon which you want to display.

**Syntax**

```
object.Icon
```

**Example:**

```
Private Sub Form_Load()
```

```
Form1.Icon = LoadPicture()  
End Sub
```

### KeyPreview Property of Form

NOTES

It returns or sets a value that decides that whether the keyboard events like KeyDown, KeyPress and KeyUp for forms should invoked or not before the keyboard events for controls.

#### Syntax

```
object.KeyPreview = Boolean
```

#### Property Settings of KeyPreview Property of Form:



### Left Property of Form

Defines the distance from left of window screen for form display.

#### Syntax

```
object.Left = value
```

#### Example:

```
Private Sub Form_Load()  
Me.Left = 6000  
End Sub
```

### LinkMode Property of Form

It returns or sets the kind of link used for a DDEconversation

#### Syntax

```
object.LinkMode = number
```

#### Property Settings of LinkMode Property of Form:



### LinkTopic Property of Form

It returns or sets the source application and the topic.

#### Syntax

```
object.LinkTopic = value
```

### MaxButton Property of Form

It returns a value which indicates that whether a form has a Maximize button or not?

#### Syntax

```
object.MaxButton
```

**Property Settings of MaxButton Property of Form:**



**MDIChild Property of Form**

It returns or sets a value which tells that whether a form should be displayed as an MDI Child form inside an MDI form or not?

*Syntax*

```
object.MDIChild
```

**Property Settings of MDIChild Property of Form:**



**MinButton Property of Form**

It returns a value which indicates that whether a form has a Minimize button or not?

*Syntax*

```
object.MinButton
```

Property settings of MinButton property of Form:



**MouseIcon Property of Form**

It returns or sets a custom mouse icon.

*Syntax*

```
object.MouseIcon = LoadPicture(pathname)
```

**MousePointer Property of Form**

It returns or sets a value which indicates the type of mouse pointer should be displayed when the mouse is over an object.

*Syntax*

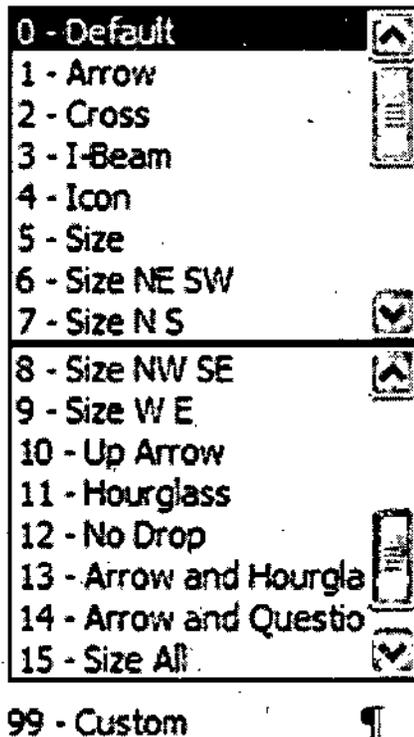
```
object.MousePointer = value
```

*Example:*

```
Private Sub Form_Load()  
Screen.MousePointer = 5  
End Sub
```

**Property Settings of MousePointer Property of Form:**

As shown here.



NOTES

## Moveable Property of Form

It returns or sets a value which identifies if the object can be moved.

### Syntax

```
object.Moveable = boolean
```

NOTES

### Property Settings of Moveable Property of Form:



## NegotiateMenus Property of Form

It sets a value which decides that whether or not a form integrates the menus from an object on the form on the form's menu bar.

Property settings of NegotiateMenus property of Form:



## OLEDropMode Property of Form

It returns or sets that how a target component will handle the drop operations.

### Syntax

```
object.OLEDropMode = mode
```

### Property Settings of OLEDropMode Property of Form:



## Palette Property of Form

It returns or sets an image which contains the palette.

### Syntax

```
object.Palette = path
```

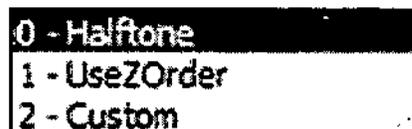
## PaletteMode Property of Form

It returns or sets a value which decides that which palette to use for the controls on an object.

### Syntax

```
object.PaletteMode = integer
```

### Property Settings of PaletteMode Property of Form:



## Picture Property of Form

It returns or sets a graphic to be displayed.

### Syntax

```
object.Picture = picture
```

## RightToLeft Property of Form

It returns a boolean value which specifies the text display direction and also controls the visual appearance on a bidirectional system.

### Syntax

```
object.RightToLeft
```

Property settings of RightToLeft property of Form:



## ScaleHeight Property of Form

It returns or sets the number of units for the horizontal measurement of the inner of a particular object.

### Syntax

```
object.ScaleHeight = value
```

## ScaleLeft Property of Form

It returns or sets the horizontal coordinates for the left of an particular object.

### Syntax

```
object.ScaleLeft = value
```

## ScaleMode Property of Form

It returns or sets a value specifying the unit of measurement for the coordinates of an particular object.

### Syntax

```
object.ScaleMode = value
```

### Property Settings of ScaleMode Property of Form:

As shown here.

## ScaleTop Property of Form

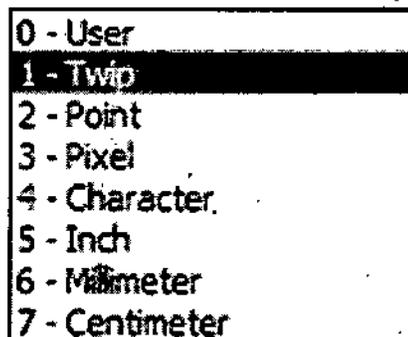
It returns or sets the vertical coordinates for the top edges of a particular object.

### Syntax

```
object.ScaleTop [= value]
```

## ScaleWidth Property of Form

It returns or sets the number of units for the vertical measurement of the inner of a particular object.



NOTES

NOTES

**Syntax**

`object.ScaleWidth [= value]`

**ShowInTaskbar Property of Form**

It returns or sets a value which specifies that whether a Form object should be displayed in the Windows 95 taskbar or not?

**Syntax**

`object.ShowInTaskbar`

**Property Settings of ShowInTaskbar Property of Form:**



**StartPosition Property of Form**

It returns or sets a value which identifies the position of an object when it appears first.

**Syntax**

`object.StartPosition = position`

**Property Settings of StartUpPosition Property of Form:**



**Tag Property of Form**

It returns or sets an expression that stores any more data needed for your program. It helps in good programming.

**Syntax**

`object.Tag [= expression]`

**Top Property of Form**

It sets the distance of form from top of window screen.

**Syntax**

`object.Top [= value]`

**Example:**

```
Private Sub Form_Load()
Me.Top = 4500
End Sub
```

## Visible Property of Form

This property defines the visible state of Form. If set to false then form becomes hidden.

### Syntax

```
object.Visible [= boolean]
```

### Example:

```
Private Sub Form_Load()  
Me.Visible = False  
End Sub
```

### Property Settings of Visible Property of Form:

True
False

## WhatsThisButton Property of Form

It returns or sets a value that decides that whether the What's This button should display in the title bar or not?

### Syntax

```
object.WhatsThisButton
```

### Property Settings of WhatsThisButton Property of Form:

True
False

## WhatsThisHelp Property of Form

It returns or sets a value which decides that whether context-sensitive Help uses the What's This pop-up provided by main Help or Windows 95 Help.

### Syntax

```
object.WhatsThisHelp [= boolean]
```

### Property Settings of WhatsThisHelp Property of Form:

True
False

## Width Property of Form

This property sets the width of a form.

### Syntax

```
object.Width = number
```

### Example:

```
Private Sub Form_Load()
```

NOTES

```
Me.Width = 3000
```

```
End Sub
```

## WindowState Property of Form

It returns or sets a value which specifies the visual state of a form window during run time.

NOTES

### Syntax

```
object.WindowState [= value]
```

### Property Settings of WindowState Property of Form:

0 - Normal
1 - Minimized
2 - Maximized

---

## PICTURE BOXES

---

Covered earlier in the chapter.

---

## TEXTBOXES

---

Covered earlier in the chapter.

---

## SUMMARY

---

1. The most basic object you will be working with in Visual Basic is the Form object, which is the visual foundation of your application.
2. The form's border is what gives the form its elasticity.
3. The title bar is the coloured bar on the top of most forms.
4. The form's caption is the text you see in the form's title bar.
5. The Control menu is a simple menu that allows you to restore, move, resize, minimize, maximize, and close a form.
6. The Minimize button is used to minimize the current form, that is, move it out of the way, to the Windows Taskbar.
7. The Close Button's sole purpose is to close the current window.
8. An Multiple Document Interface (MDI) lets you open windows within a parent container window.
9. To create an MDI application, you need at least two forms in the application.
10. Each VBProject object in the collection represents a loaded VBA project.
11. Part of VBA project is the set of type library references for any Automation components it uses.
12. Each Project Property object corresponds to a property to the particular component.
13. After determining that a property value can be read, a series of If and ElseIf statements try to determine what type of property the current Property object is and how best to deal with it.
14. One of the primary benefits of object-oriented programming in general and VBA class

- modules in particular is the ability to encapsulate data and behavior in high-level programming constructs.
15. Object-oriented techniques (which include using VBA class modules) can make managing projects easier.
  16. VBA class modules define the properties and methods of an object.
  17. Object instances are the documents you create from a template.
  18. There are seven methods of the CodeModule class that you can use to modify code.
  19. Visual Basic has a number of controls. Some of them like labels or list boxes, give users feedback, while others, like command buttons and text boxes, elicit responses.
  20. You can use a command button to elicit simple responses from the user or to invoke special functions on forms.
  21. Text boxes are commonly used for accepting user input or for entering data.
  22. A label control is similar to a text box control in that both display text.
  23. Option button controls, also called radio buttons are used to allow the user to select one and only one, option from a group of options.
  24. Check boxes are valid as single controls – a single option button is probably counter-intuitive.
  25. Although a frame is often used as a container for check box groups too, each check box is completely independent.
  26. A list box is an ideal way of representing users with a list of data.
  27. A combo box combines the features of both a text box and a list box.
  28. The image control that comes with Visual Basic can now display bitmap (.BMP), icon (.ICO), metafile (.WMF), JPEG (.JPG) and GIF (.GIF) files.
  29. The timer control is one of the few controls always hidden at run time.
  30. A scroll bar control on a form is not to be confused with a scroll bar on a large text box or list box.
  31. The drive list box control (or just drive), is normally used in conjunction with the directory list and the file list controls.
  32. The directory list box (or simply directory) control, is used in conjunction with the drive control.
  33. The File List Box control comes at the end of the drive-directory-file chain.
  34. There are five controls that provide most of the functionality found in the most common Windows applications. These include the Tree View, List View, Image List, Status Bar and toolbar.
  35. The Tree View control provides a hierarchical view of folders or other items that can be neatly categorized in a tree-style layout.
  36. List View control is often used in conjunction with the Tree View control.
  37. The Image List control does not actually appear on a form at run time.
  38. The Status Bar control is used to report various bits of information to the user.

## NOTES

---

**SELF ASSESSMENT QUESTIONS**


---

1. What are Class Modules?
2. How Class Modules work?
3. How do you work with procedures?
4. What are Event Procedures?
5. How would you use Property Procedures?
6. What are Programming Controls?
7. How is Text Box used?

NOTES

8. How is Label button used?
9. What are Option Buttons? How are they used?
10. What are Check Boxes? How are they used?
11. What are Frame Controls?
12. How do you use List Boxes?
13. What are Combo Boxes?
14. How would you use Image Objects button?
15. What are Picture Boxes? How are they used?
16. What are Timers?
17. How would you use Timers?
18. How would you use Drive and Directory List Boxes?
19. How would you use File List Boxes?
20. How would you use Tree View Control?
21. How would you use List View Control?
22. How would you use Image List Control?
23. Describe the use of Status Bar Control.
24. Describe the various components of the Form.
25. What are the Form properties?
26. How would you create a MDI Form?
27. What is VBProject Class?
28. What is Reference Class?
29. How would you remove Referenes?
30. How would you add References?
31. What are Indexed and Object Properties?

**Multiple Choice Questions**

1. Data from the users is accepted using :
 

(a) Text Boxes	(b) Command button	(c) Label control
----------------	--------------------	-------------------
2. Option button is also called :
 

(a) Close button	(b) Radio button	(c) Touch button
------------------	------------------	------------------
3. For single controls you use :
 

(a) Text box	(b) Combo box	(c) Check box
--------------	---------------	---------------
4. For list data you use :
 

(a) Text box	(b) List box	(c) Check box
--------------	--------------	---------------
5. Combo box combines the features of a text box and :
 

(a) List box	(b) Check box	(c) Combo box
--------------	---------------	---------------
6. This control is always hidden at the time of running.
 

(a) Command	(b) Timer	(c) Date
-------------	-----------	----------
7. Directory list and file list can be reflected on the form using:
 

(a) Drive list control	(b) Command list control	(c) Directory list control
------------------------	--------------------------	----------------------------
8. Tree view control provides view of :
 

(a) Folders	(b) Directories	(c) Files
-------------	-----------------	-----------
9. The item on which you can add different elements to create an application is called :
 

(a) Form	(b) Visual Basic	(c) Project
----------	------------------	-------------
10. MDI stands for :
 

(a) Mini Document Interface		
-----------------------------	--	--

- (b) Multiple Document Interface  
 (c) Multiple Document Information
11. A collection of loaded VBA projects is called:  
 (a) VB (b) VB Object (c) VBProject Object
12. Each Property object corresponds to a property to the this particular component:  
 (a) VBComponent (b) VBProjec (c) VB

NOTES

### True/False Questions

1. The form's border is what gives the form its elasticity.
2. The title bar is the coloured bar on the bottom of most forms.
3. The form's caption is the text you see in the form's title bar.
4. The Close Button's sole purpose is to open the current window.
5. To create an MDI application, you need at least two forms in the application.
6. Each VBProject object in the collection represents a loaded VBA project.
7. Part of VBA project is the set of type library references for any Automation components it uses.
8. Each Project Property object corresponds to a property to the particular component.
9. VBA class modules do not define the properties and methods of an object.
10. Object instances are the documents you create from a template.
11. There are seven methods of the CodeModule class that you can use to modify code.
12. You can use a label button to elicit simple responses from the user or to invoke special functions on forms.
13. Text boxes are commonly used for accepting user input or for entering data.
14. A label control is similar to a text box control is that both display text.
15. Option button controls, also called radio buttons are used to allow the user to select one and only one, option from a group of options.
16. List boxes are valid as single controls – a single option button is probably counter-intuitive.
17. A list box is an ideal way of representing users with a list of data.
18. A combo box combines the features of both a text box and a list box.
19. The timer control is one of the few controls always hidden at run time.
20. A scroll bar control on a form is not a scroll bar on a large text box or list box.
21. The directory list box (or simply directory) control, is used in conjunction with the drive control.
22. The File List Box control comes at the end of the drive-directory-file chain.
23. The Tree View control does not provide a hierarchical view of folders or other items that can be neatly categorized in a tree-style layout.
24. List View control is often used in conjunction with the Tree View control.
25. The Status Bar control is used to report various bits of information to the user.

### Short Questions with Answers

1. Which procedures are supported by Visual Basic?
- Ans.** Visual Basic supports following procedures:
- Sub Procedur
  - Function Procedure
  - Property Procedure
2. What is a Sub Procedure?
- Ans.** Sub Procedures are block of code where you write all commands for execution. You have to give some name to Sub Procedure so whenever you call this name you execute all commands or statements written inside the Sub Procedure.

NOTES

3. What is Event Procedure?  
**Ans.** Whenever any event occurs then object invokes the event procedure using the name corresponding to event. Generally Event procedures are attached to the forms and controls. An event procedure for a form or control combines the word "Form" or control name and the event name.
4. What is Function Procedure?  
**Ans.** Function Procedure returns a value to the calling procedure. Function procedure is also a separate procedure which you can keep in a program as a block of code, once it is defined then function can take the arguments, perform them and can return some specific result according to coding.
5. Which are the controls of Visual Basic?  
**Ans.** Visual Basic has a number of controls. Some of them like labels or list boxes, give users feedback, while others, like command buttons and text boxes, elicit responses. Other controls sit quietly, invisible to the user and perform some of the grunt work that makes your application useful. The timer control is one example of an invisible control.
6. What are Text boxes?  
**Ans.** Nearly every Visual Basic project involves at least one text box control. Text boxes are commonly used for accepting user input or for entering data. Their properties are, of course specifically designed for these purposes. If you only want the simplest of user responses, you might consider using an InputBox instead. The InputBox displays a dialog box and prompts the user to enter something and returns this to the application.
7. What is Label Control?  
**Ans.** A label control is similar to a text box control is that both display text. The main difference however is that a label displays read-only text as far as the user is concerned though you can alter the caption as a run-time property.
8. What are Option Button Controls?  
**Ans.** Option button controls, also called radio buttons are used to allow the user to select one and only one, option from a group of options. Usually option buttons are grouped together within a frame control but they can also be grouped on a plain form, if there is to be only one group of option buttons.
9. What is a Check Box Control?  
**Ans.** A check box control is rather similar to an option button, which was described in the last section. Both often partake in group and the Value property is tested to see if a check box is on or off.
10. What is a Frame Control?  
**Ans.** When used by itself, the frame control is not particularly useful. The controls normally placed in a frame are option buttons and check boxes. This has the effect of grouping them together so that when the frame is moved, the other controls move too. For this to work you can't double-click a control (say, an option button) to add it to the form and then drag it into position within the frame. Instead, you must single-click the control in Toolbox and drag a location for it inside the frame. Then all the controls move together.
11. What is a List Box Control?  
**Ans.** If you're a regular user of Windows, then you're familiar with list box controls. A list box is an ideal way of representing users with a list of data. Users can browse the data in the list box or select one or more items as the basis for further processing.
12. What is a Combo Box?  
**Ans.** The name combo box comes from "combination box". The idea is that a combo box combines the features of both a text box and a list box. A potential problem with list boxes – in some situations anyway – is that you're stuck with the entries displayed. You can't directly edit an item in the list or select an entry that's not already there. Of course, if you want to restrict the user, than a list box is fine in this respect. A combo box control (at least in two of its styles available in Visual Basic) allows you to select a predefined item from a

list or to enter a new item not in the list. A combo box can also incorporate a drop-down section – that means it takes less room on a form than a normal list box. In all, there are three types of combo boxes to choose from at design time: a drop-down combo, a simple combo and a drop-down list. You can specify the type by setting the Style property.

13. What is an Image Control?

**Ans.** The image control (its prefix is often `img`) is a lightweight equivalent of the picture box control, which is described in a later section. But unlike the picture control, the image control can't act as a container for other objects. In some of its other properties it's not as versatile, but it's a good choice if you simply want to display a picture on a form. Image controls consume far less memory than picture controls. The image control that comes with Visual Basic can now display bitmap (.BMP), icon (.ICO), metafile (.WMF), JPEG (.JPG) and GIF (.GIF) files. This makes it easier to display graphics from the World Wide Web as well as graphics from other popular graphics programs.

14. What are Picture Boxes?

**Ans.** Picture boxes are similar to image controls. However, picture boxes and images have slightly different properties and therefore behave differently. If you just want to show a picture, then an image control is usually a better choice than a picture box. An image control takes up less memory and is a lightweight version of the picture box control.

15. What is a Timer Control?

**Ans.** The timer control is one of the few controls always hidden at run time. This means you don't have to find room for it on a form – it can go anywhere, even on top of existing controls. The timer basically does just one thing: It checks the system clock and acts accordingly.

16. What is a Scroll Bar?

**Ans.** A scroll bar control on a form is not to be confused with a scroll bar on a large text box or list box. The scroll bar controls are completely independent objects that exist without reference to any other control (this is not the case with large text boxes or list boxes). The horizontal scroll bar and the vertical scroll bar are identical except for their orientation.

17. What is a Drive List Box Control?

**Ans.** The drive list box control (or just `drive`), is normally used in conjunction with the directory list and the file list controls. At its most fundamental, these three controls allow the user to select a file in a particular directory on a particular drive. The user changes to another drive via the drive control.

18. What is Directory List Box Control?

**Ans.** The directory list box (or simply `directory`) control, is used in conjunction with the drive control, described earlier and file control. The user can select a directory on the current drive from the directory list.

19. What are File List Box Controls?

**Ans.** The File List Box control comes at the end of the drive-directory-file chain. To reiterate, the file control should be updated in the `directory Change()` event. The directory control itself is updated when the user selects a directory in the directory control – it's also updated when the user selects a new drive in the drive control.

20. What are Tree View Controls?

**Ans.** The Tree View control provides a hierarchical view of folders or other items that can be neatly categorized in a tree-style layout. It is often used in conjunction with a List View control which is used to display the contents of the folder selected in the tree view.

21. What are Forms?

**Ans.** The most basic object you will be working with in Visual Basic is the Form object, which is the visual foundation of your application. It is basically a window that you can add different elements to in order to create a complete application. Every application you can see on the screen is based on some type of form.

NOTES

22. What are modules?

Ans. All coding stuffs whether written to operate or perform certain task in form, etc., are stored in modules. Your Visual Basic application contains minimum one form and this form is nothing but a container which holds all code known as form module.

NOTES

**ANSWERS**

**Multiple Choice Questions**

- |      |       |       |       |
|------|-------|-------|-------|
| 1. a | 2. b  | 3. c  | 4. b  |
| 5. a | 6. b  | 7. c  | 8. a  |
| 9. a | 10. b | 11. c | 12. a |

**True False Questions**

- |       |       |       |       |
|-------|-------|-------|-------|
| 1. T  | 2. F  | 3. T  | 4. F  |
| 5. T  | 6. T  | 7. T  | 8. T  |
| 9. F  | 10. T | 11. T | 12. F |
| 13. T | 14. T | 15. T | 16. F |
| 17. T | 18. T | 19. T | 20. F |
| 21. T | 22. T | 23. F | 24. T |
| 25. T |       |       |       |

# PRINTERS AND FUNCTIONS

---

### LEARNING OBJECTIVES

After going through this chapter, you should appreciate the following:

- Printer Object controlling Program flow
- Built in Functions
- User Defined Functions and Procedures
- Arrays
- Grids
- Records
- Object Oriented Programming
- Creating Objects
- Building Classes

---

## PRINTER OBJECT CONTROLLING PROGRAM FLOW

---

NOTES

In general everyone treats printing a print out on paper. In Visual Basic you have several methods and ways for printing. Using Visual Basic you can print the entire form or line by line to a form. You can print to Immediate window. You can also print reports created through Crystal report or any third party utility, it is for general example, you can print from other application like MS Word etc.

### Printing Form and Print Method

Print method displays the values or values stored in a variable. Means using print method you can print on form. You can also print on printer using print method.

To print on a form you can write syntax like this:

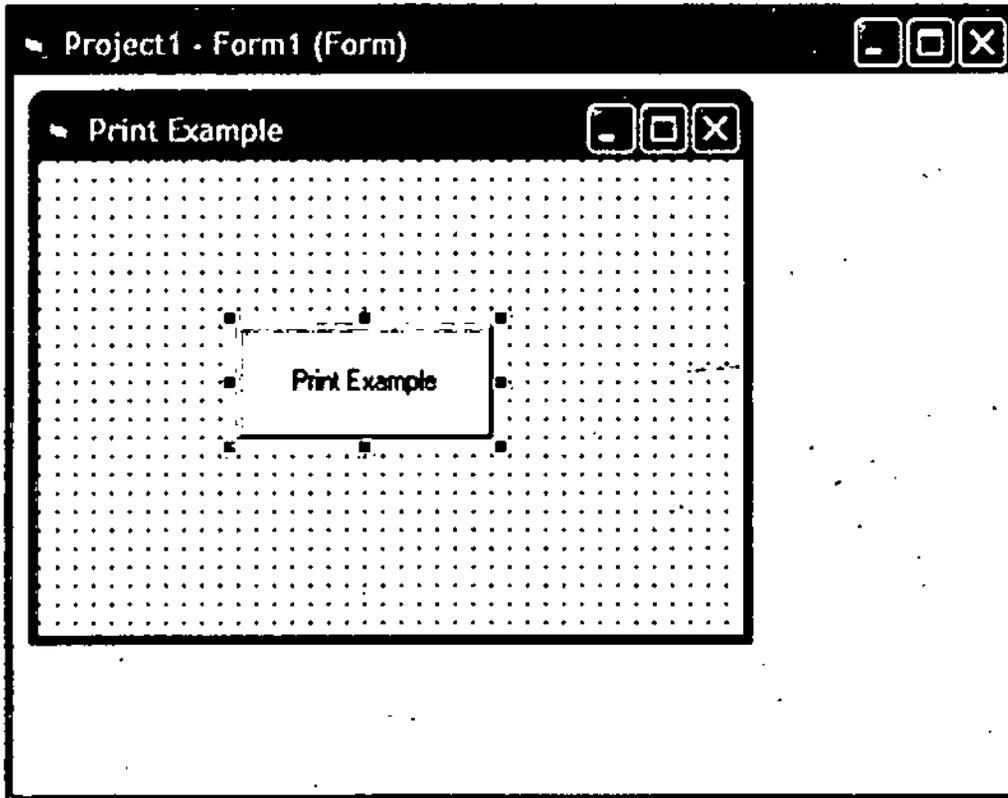
```
Print "hello"
```

### Practice Yourself

1. Start Visual Basic 6.
2. Select Standard Exe from New tab.
3. Click Open.
4. On menu bar click File.
5. Click Save Project.
6. Save File As dialog box appears.
7. Select the desired drive.
8. In File name field type printexample.frm
9. Click Save.
10. Save Project As dialog box appears.
11. In File name field type printexample.vbp
12. Click Save.
13. Source Code Control dialog box appears.
14. Click No.
15. Select the Form by single clicking on it.
16. In Properties window locate the Caption property.
17. Type "Print Example".
18. Double-click the command button to add it on form.
19. Select the command button.
20. Locate the Caption property in Properties window.
21. Type "Print Example".
22. Locate the Height property.
23. Type, 735.

24. Locate the Width property.
25. Type 1695.
26. Locate the Top property.
27. Type 1080.
28. Locate the Left property.
29. Type 1320.

At present form will appear like this:

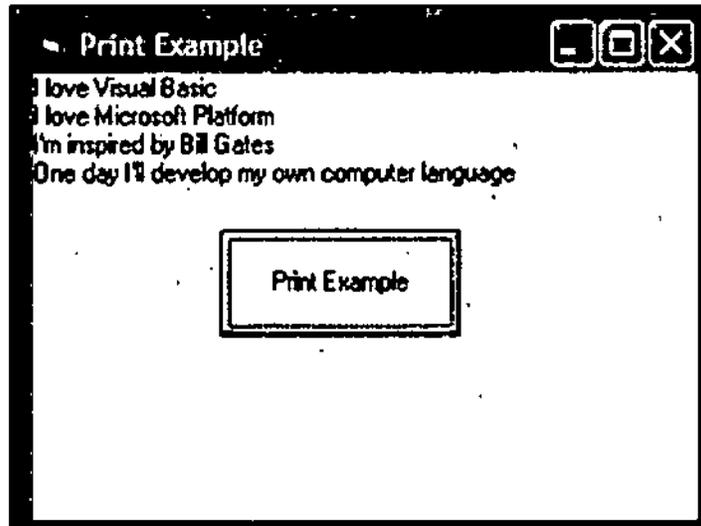
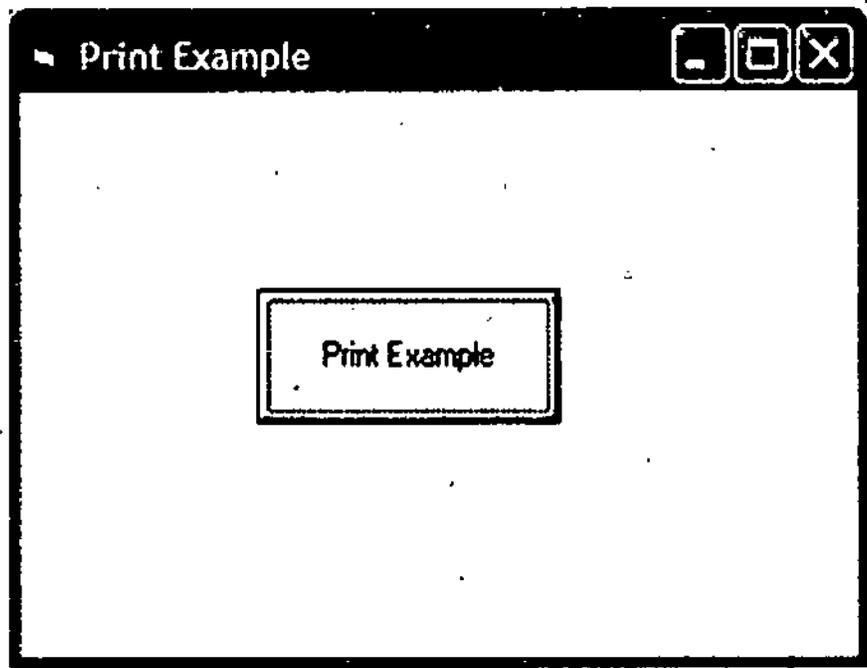


NOTES

30. Double-click the button control to open code window.
31. Type the following code:
 

```
Private Sub Command1_Click()
Print "I love Visual Basic"
Print "I love Microsoft Platform"
Print "I'm inspired by Bill Gates"
Print "One day I'll develop my own computer language"
End Sub
```
32. Press F5 key on your keyboard.
33. At present you will get window shown next.
34. Click on button.
35. You will get the output shown on the next page.

NOTES



### Clearing Form

To clear a form you can use CLS method. CLS method will clear the text on form.

#### Syntax

**CLS**

#### Practice Yourself

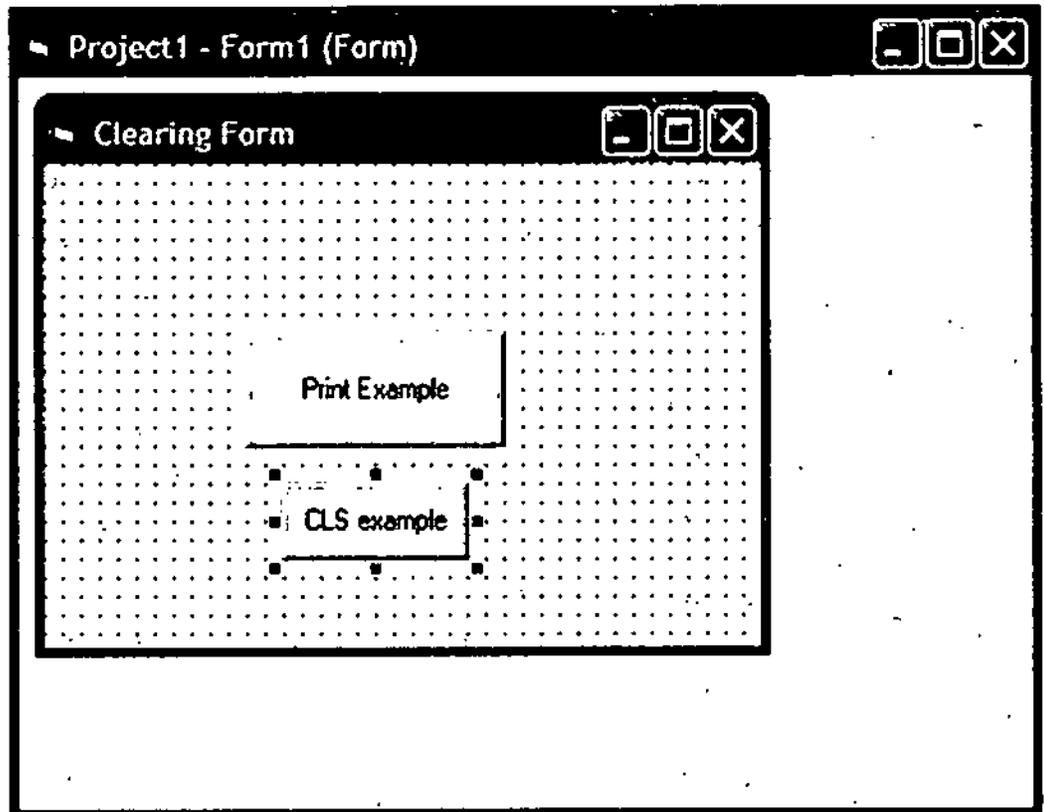
1. Start Visual Basic 6.
2. Select Standard Exe from New tab.
3. Click Open.
4. On menu bar click File.
5. Click Save Project.
6. Save File As dialog box appears.

7. Select the desired drive.
8. In File name field type cls.frm
9. Click Save.
10. Save Project As dialog box appears.
11. In File name field type cls.vbp
12. Click Save.
13. Source Code Control dialog box appears.
14. Click No.
15. Select the Form by single clicking on it.
16. In Properties window locate the Caption property.
17. Type "Clearing Form".
18. Double-click the command button to add it on form.
19. Select the command button.
20. Locate the Caption property in Properties window.
21. Type "Print Example".
22. Locate the Height property.
23. Type, 735.
24. Locate the Width property.
25. Type 1695.
26. Locate the Top property.
27. Type 1080.
28. Locate the Left property.
29. Type 1320.
30. Add one more command button.
31. Select the second command button.
32. Locate the Caption property in Properties window.
33. Type "CLS example".
34. Locate the Height property.
35. Type, 495.
36. Locate the Width property.
37. Type 1215.
38. Locate the Top property.
39. Type 2040.
40. Locate the Left property.
41. Type 1560.

NOTES

At present form will appear like this:

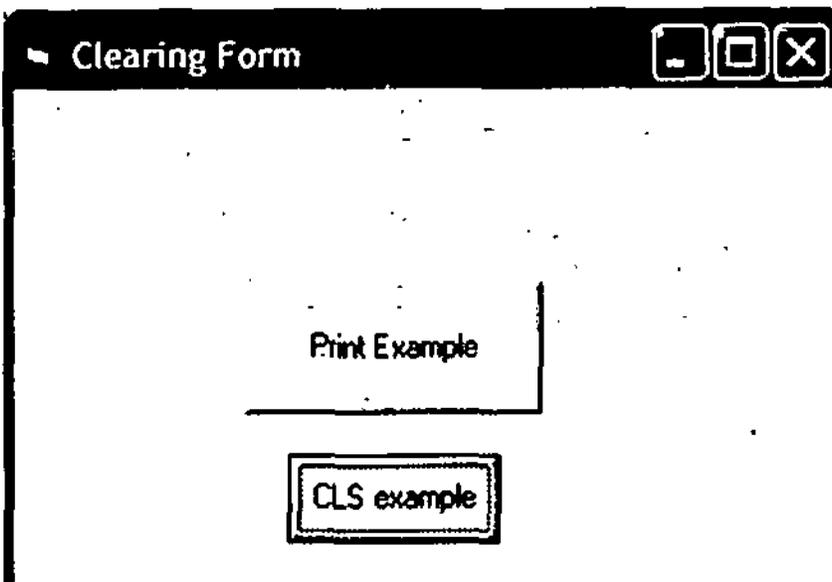
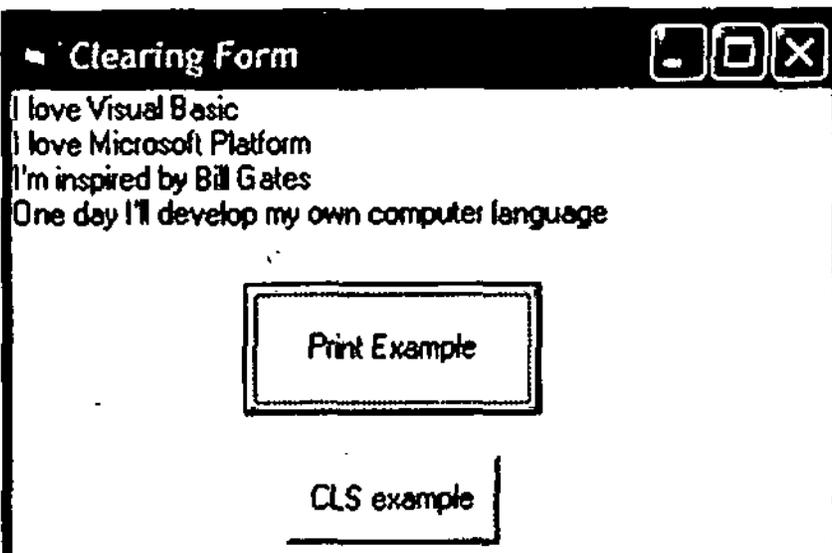
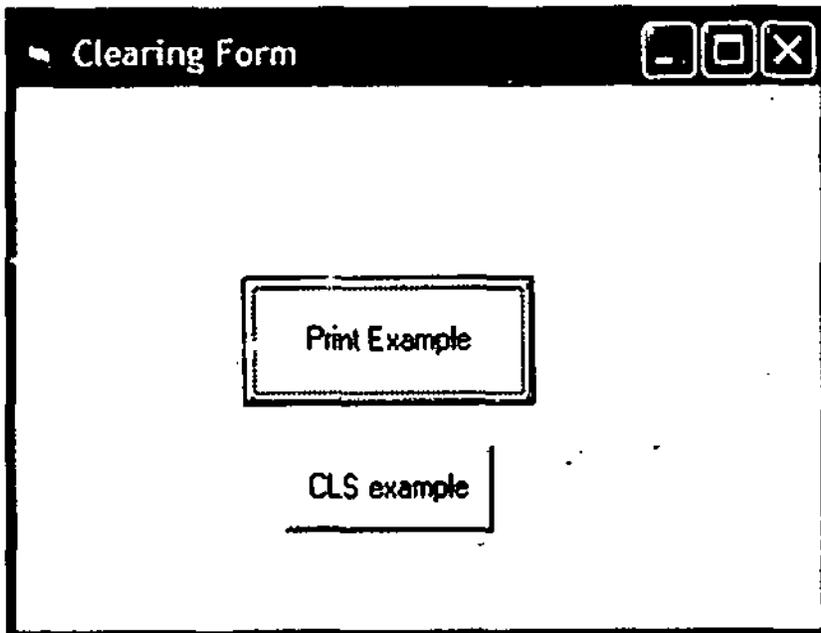
NOTES



42. open the code window.
43. Type the following code:
 

```
Private Sub Command1_Click()
Print "I love Visual Basic"
Print "I love Microsoft Platform"
Print "I'm inspired by Bill Gates"
Print "One day I'll develop my own computer language"
End Sub
Private Sub Command2_Click()
Cls
End Sub
```
44. Press F5 key to run the program.
45. You will get window shown on next page.
46. Click first button.
47. You will get the output shown on the next page.
48. Click second button.
49. You will get the output shown on the next page.

NOTES



## Printing to a Printer

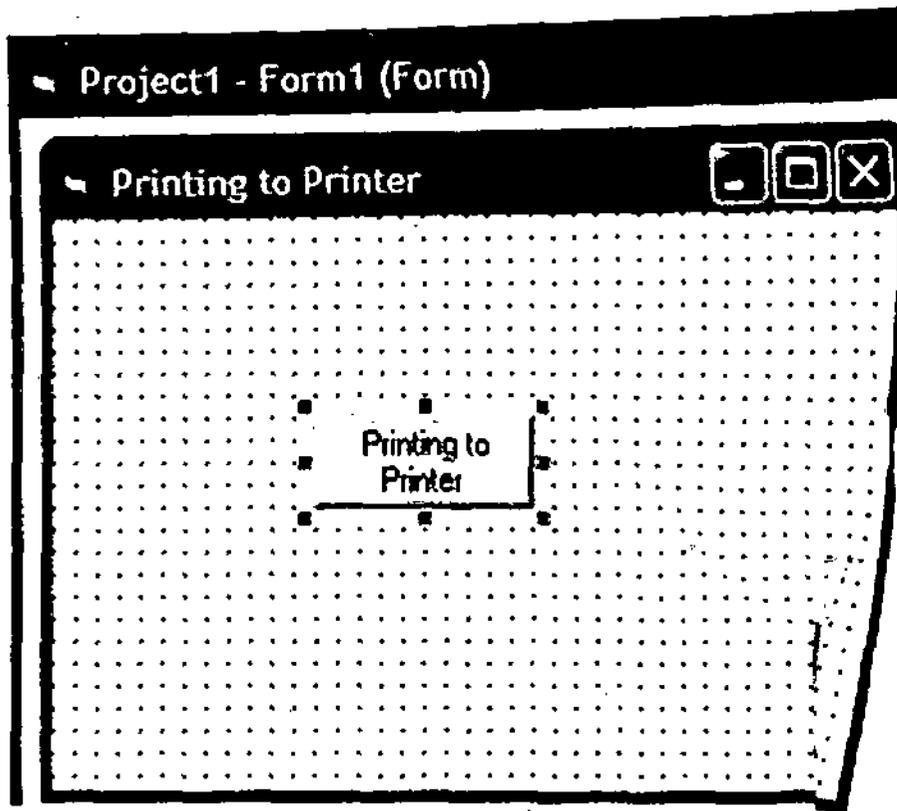
In Visual Basic you have facility to print a copy from printer through your Visual Basic program. To make the printer start printing you have to give `Printer.EndDoc` method.

### NOTES

#### Practice Yourself

1. Start Visual Basic 6.
2. Select Standard Exe from New tab.
3. Click Open.
4. On menu bar click File.
5. Click Save Project.
6. Save File As dialog box appears.
7. Select the desired drive.
8. In File name field type `printingexample.frm`
9. Click Save.
10. Save Project As dialog box appears.
11. In File name field type `printingexample.vbp`
12. Click Save.
13. Source Code Control dialog box appears.
14. Click No.
15. Select the Form by single clicking on it.
16. In Properties window locate the Caption property.
17. Type "Printing to Printer".
18. Double-click the command button to add it on form.
19. Select the command button.
20. Locate the Caption property in Properties window.
21. Type "Printing to Printer".
22. Locate the Height property.
23. Type, 495.
24. Locate the Width property.
25. Type 1215.
26. Locate the Top property.
27. Type 1080.
28. Locate the Left property.
29. Type 1440.

At present your form will appear like the one shown on the next page.



NOTES

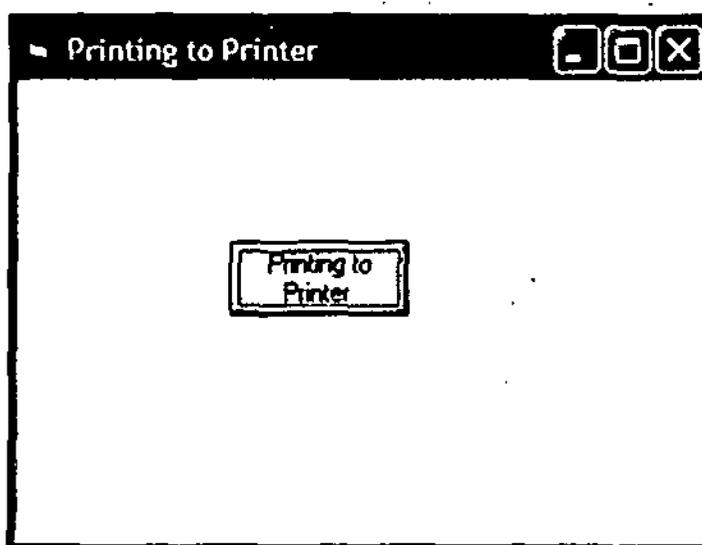
30. Double-click the button control to open code window.

31. Type the following code:

```
Private Sub Command1_Click()  
Printer.Print  
End Sub
```

32. Press F5 key on your keyboard.

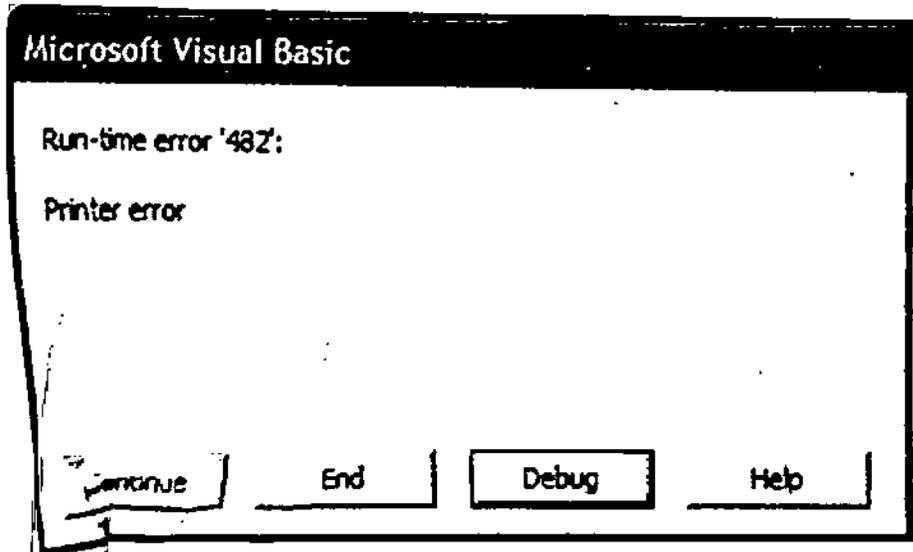
33. You will get following window:



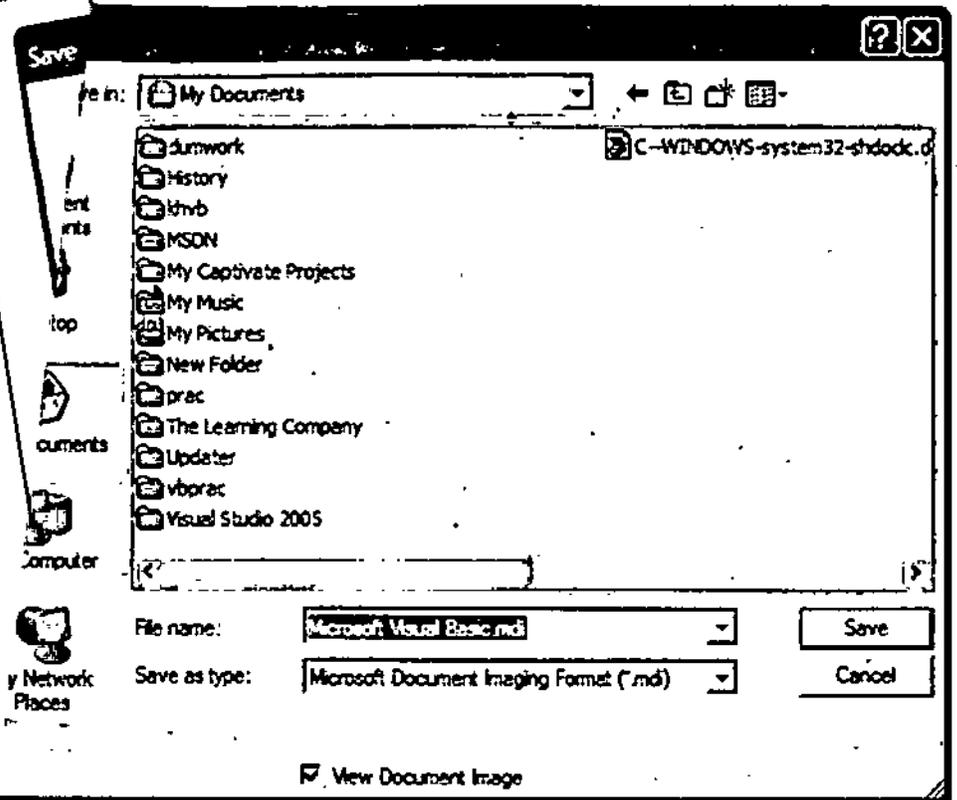
34. Click the button.

Note: If printer is not installed in your system then you can get following error:

NOTES



Note: If printer is not installed in your system then Windows can also force you to save as Microsoft Document Imaging Format.



### PrintForm Method

PrintForm method is used to print an active form. This method sends a pixel-by-pixel image of a form to the printer.

example:

```
Form1.PrintForm
```

1. Start Visual Basic 6.
2. Select Standard Exe from New tab.
3. Click Open.
4. On menu bar click File.
5. Click Save Project.
6. Save File As dialog box appears.
7. Select the desired drive.
8. In File name field type printform.frm
9. Click Save.
10. Save Project As dialog box appears.
11. In File name field type printform.vbp
12. Click Save.
13. Source Code Control dialog box appears.
14. Click No.
15. Select the Form by single clicking on it.
16. In Properties window locate the Caption property.
17. Type "PrintForm method".
18. Double-click the command button to add it on form.
19. Select the command button.
20. Locate the Caption property in Properties window.
21. Type "PrintForm method".
22. Locate the Height property.
23. Type, 495.
24. Locate the Width property.
25. Type 1215.
26. Locate the Top property.
27. Type 1320.
28. Locate the Left property.
29. Type 1800.

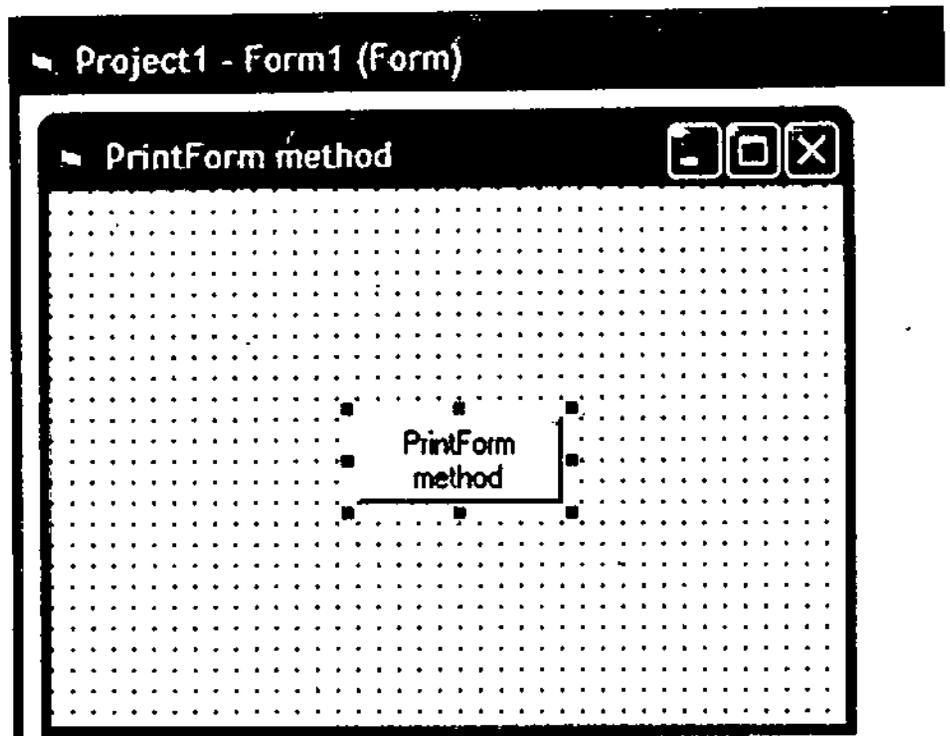
At present your form will appear like the one shown on next page.

30. Double-click the button control to open code window.
31. Type the following code:

```
Private Sub Command1_Click()
```

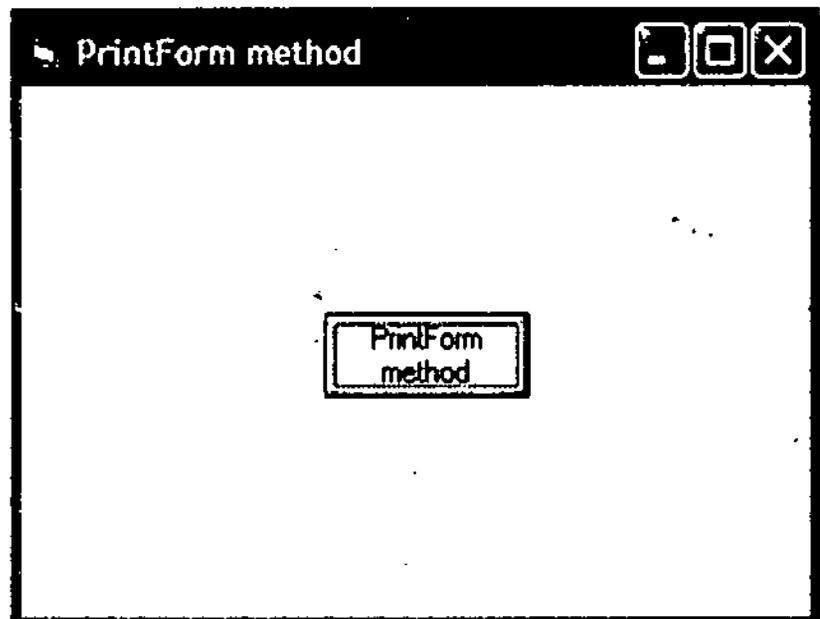
NOTES

NOTES



```
Form1.PrintForm  
End Sub
```

- 32. Press F5 key on your keyboard to run the program.
- 33. You will get the the window shown next.



- 34. For printing click the button.

### Multi-page Document Printing

Using NewPage method you can do multi-page document printing.

For example:

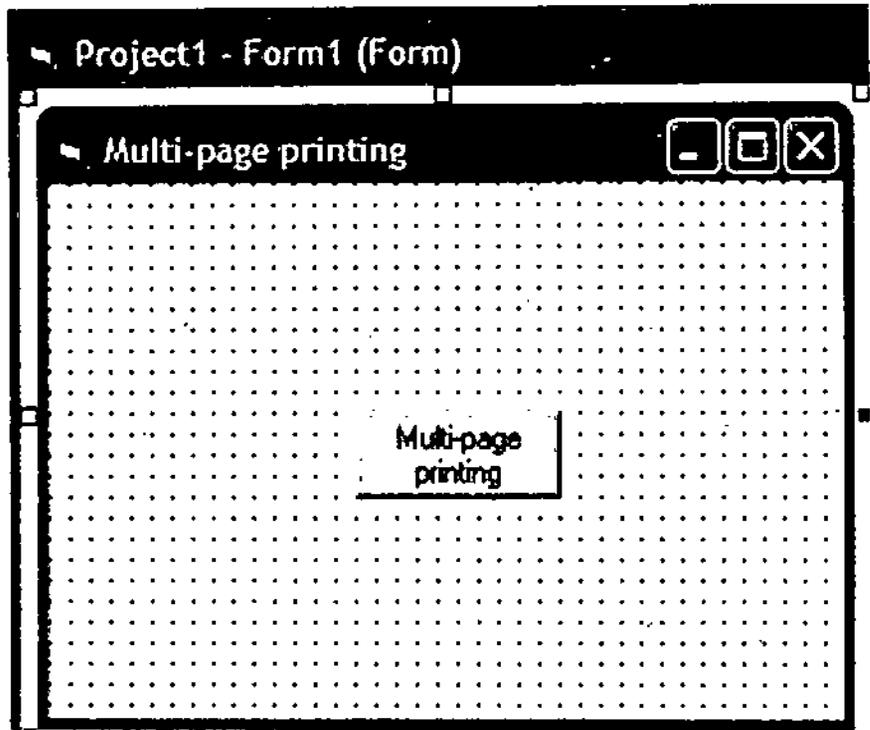
**Printer.NewPage**

Practice Yourself

1. Start Visual Basic 6.
  2. Select Standard Exe from New tab.
  3. Click Open.
  4. On menu bar click File.
  5. Click Save Project.
  6. Save File As dialog box appears.
  7. Select the desired drive.
  8. In File name field type multipageprinting.frm
  9. Click Save.
  10. Save Project As dialog box appears.
  11. In File name field type multipageprinting.vbp
  12. Click Save.
  13. Source Code Control dialog box appears.
  14. Click No.
  15. Select the Form by single clicking on it.
  16. In Properties window locate the Caption property.
  17. Type "Multi-page printing".
  18. Double-click the command button to add it on form.
  19. Select the command button.
  20. Locate the Caption property in Properties window.
  21. Type "Multi-page printing".
  22. Locate the Height property.
  23. Type, 495.
  24. Locate the Width property.
  25. Type 1215.
  26. Locate the Top property.
  27. Type 1320.
  28. Locate the Left property.
  29. Type 1800.
- At present your form will appear like the one shown on the next page.
30. Double-click the button control to open code window.
  31. Type the following code:

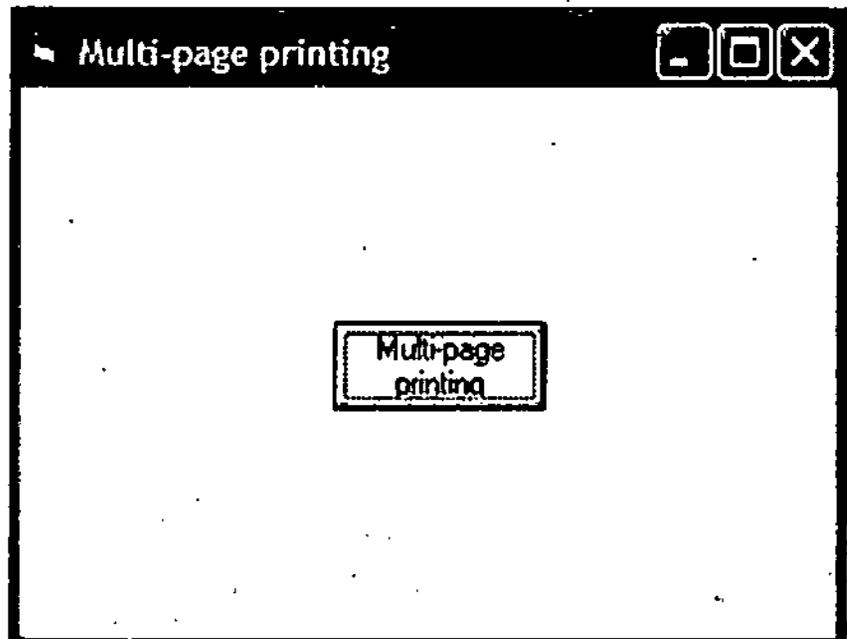
NOTES

NOTES



```
Private Sub Command1_Click()  
Printer.Print "First page printing"  
Printer.NewPage  
Printer.Print "Second page printing"  
Printer.EndDoc  
End Sub
```

32. Press F5 key on your keyboard.
33. You will get following window:
34. Press button for printing.



## Printer Collection

Using Printer collection you can query the list of printer objects which are registered on your computer system. If you query the collection then you are in position to change the default printer or can switch back to normal printer. You can also print raw data.

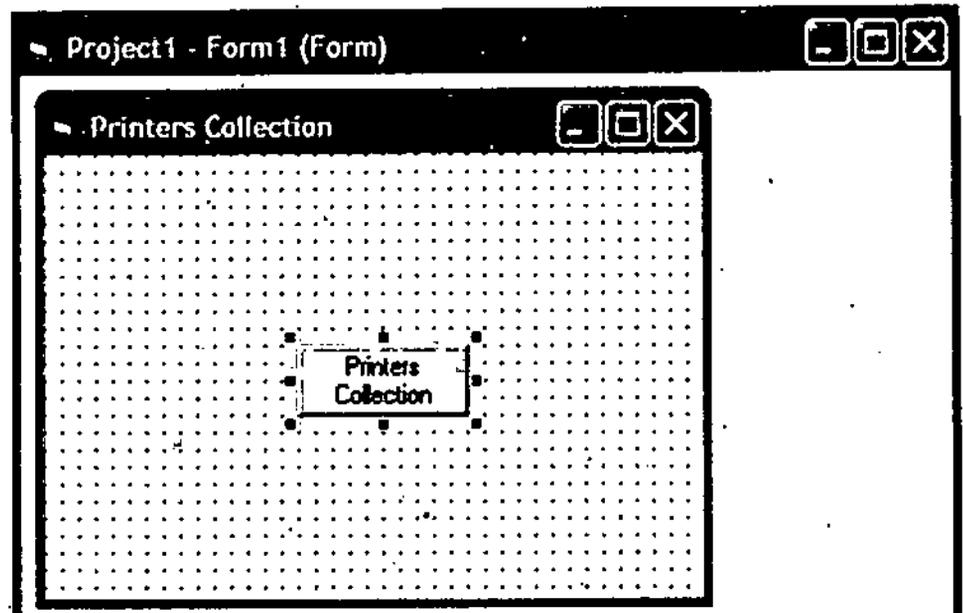
### Practice Yourself

1. Start Visual Basic 6.
2. Select Standard Exe from New tab.
3. Click Open.
4. On menu bar click File.
5. Click Save Project.
6. Save File As dialog box appears.
7. Select the desired drive.
8. In File name field type printerscollection.frm
9. Click Save.
10. Save Project As dialog box appears.
11. In File name field type printerscollection.vbp
12. Click Save.
13. Source Code Control dialog box appears.
14. Click No.
15. Select the Form by single clicking on it.
16. In Properties window locate the Caption property.
17. Type "Printers Collection".
18. Double-click the command button to add it on form.
19. Select the command button.
20. Locate the Caption property in Properties window.
21. Type "Printers Collection".
22. Locate the Height property.
23. Type, 495.
24. Locate the Width property.
25. Type 1215.
26. Locate the Top property.
27. Type 1320.
28. Locate the Left property.
29. Type 1800.

### NOTES

At present your form will appear like this:

NOTES



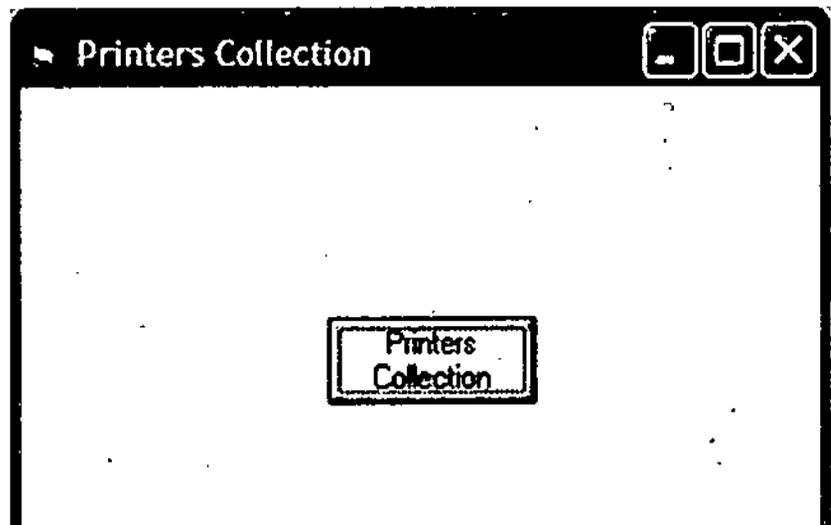
30. Double-click the button control to open code window.

31. Type the following code:

```
Private Sub Command1_Click()  
Dim p As Printer  
For Each p In Printers  
MsgBox p.DeviceName  
MsgBox p.DriverName  
MsgBox p.Port  
Next  
End Sub
```

32. Press F5 key on your keyboard to run the program.

33. You will get following window:



34. Click the button.

35. If printer is not installed in your system then you can get outputs, shown on the next page, but if printer is installed then output will be differ.

### Printing in Immediate Window

To see the values inside variables you can use Immediate window in Visual Basic. It is possible to display the values in Immediate window so that easily you can track the things. For this you have Debug.method available.

For example:

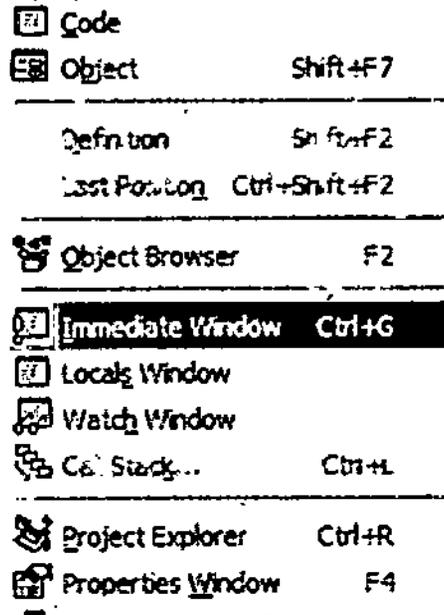
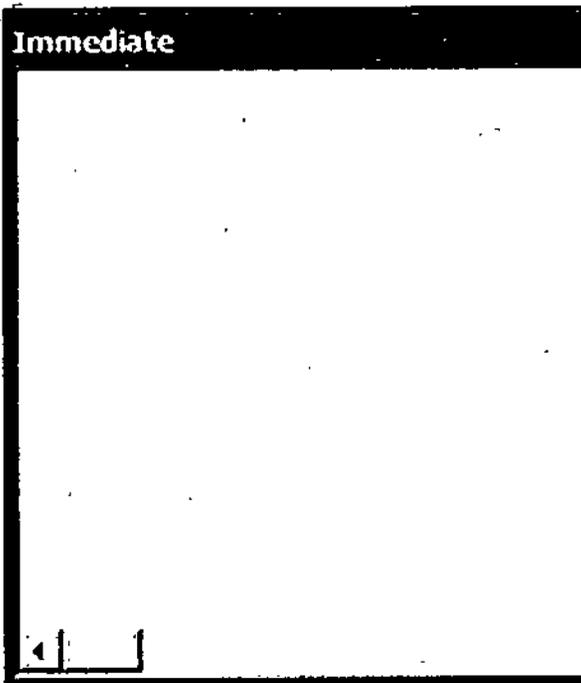
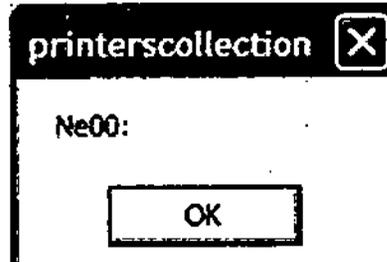
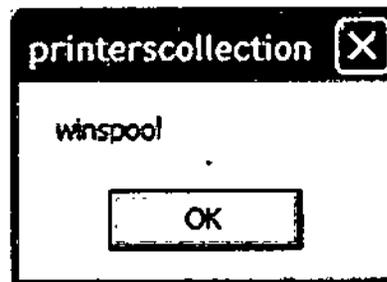
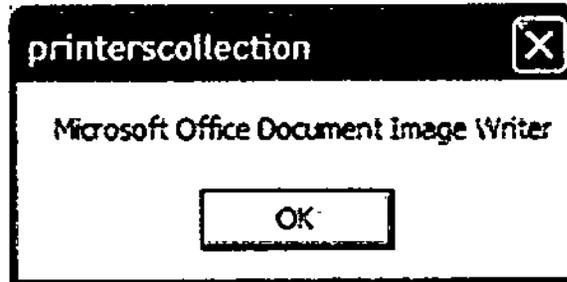
```
Debug.Print x
```

If immediate window is not visible perform the following steps:

1. On menu bar of IDE click View.
  2. Click Immediate Window
- Or

You can also use short cut key Ctrl + G

Immediate Window looks like the one shown below



#### Practice Yourself

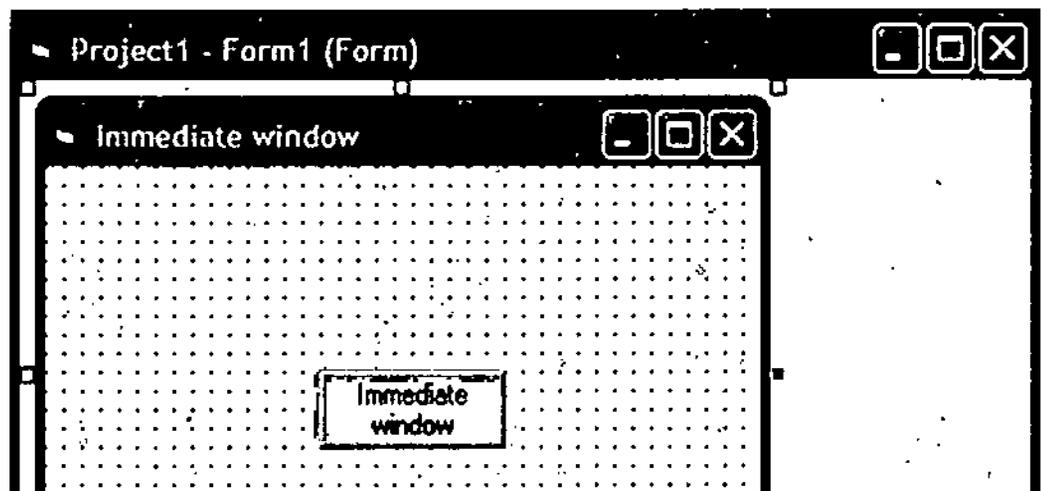
1. Start Visual Basic 6.
2. Select Standard Exe from New tab.
3. Click Open.
4. On menu bar click File.
5. Click Save Project.

NOTES

NOTES

6. Save File As dialog box appears.
7. Select the desired drive.
8. In File name field type immediatewindow.frm
9. Click Save.
10. Save Project As dialog box appears.
11. In File name field type immediatewindow.vbp
12. Click Save.
13. Source Code Control dialog box appears.
14. Click No.
15. Select the Form by single clicking on it.
16. In Properties window locate the Caption property.
17. Type "Immediate window".
18. Double-click the command button to add it on form.
19. Select the command button.
20. Locate the Caption property in Properties window.
21. Type "Immediate window".
22. Locate the Height property.
23. Type, 495.
24. Locate the Width property.
25. Type 1215.
26. Locate the Top property.
27. Type 1320.
28. Locate the Left property.
29. Type 1800.

At present your form will appear like this:



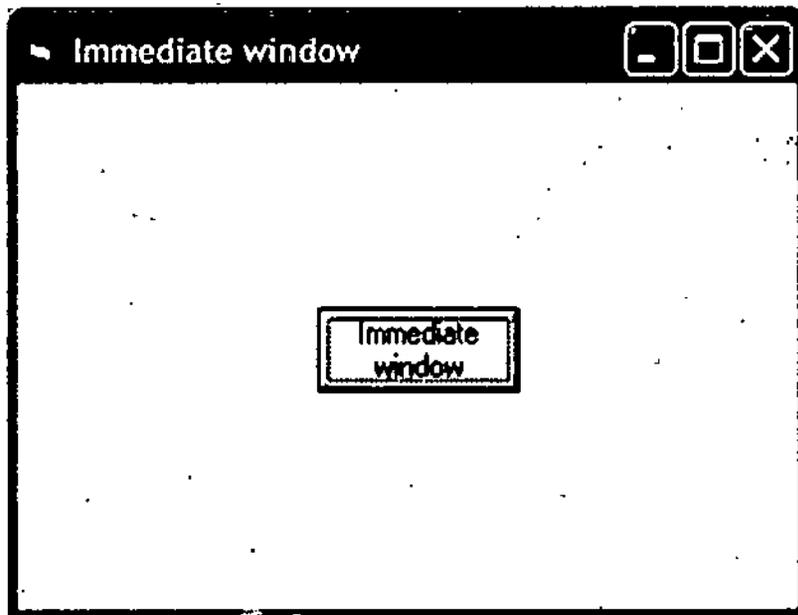
30. Double-click the button control to open code window.

31. Type the following code:

```
Private Sub Command1_Click()  
    x = 15  
    Debug.Print x  
    x = 25  
    Debug.Print x  
End Sub
```

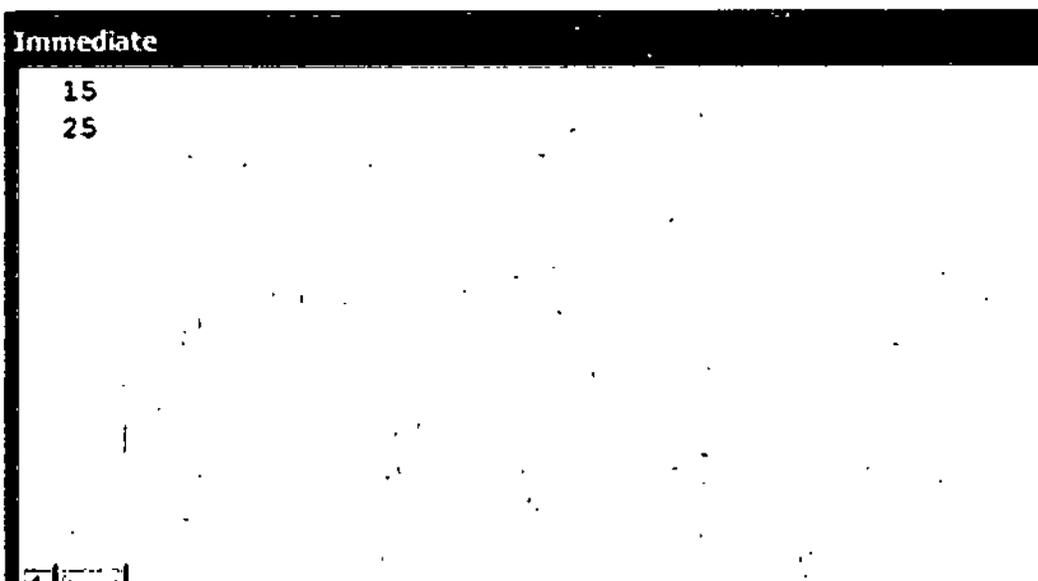
32. Press F5 key on your keyboard to run the program.

33. You will get the following window:



34. Click on the button.

35. Notice the Immediate window



NOTES

## Data Report Designer

In Visual Basic you can also create the reports and Microsoft Data Report designer is used to generate the reports. It is used in conjunction with the data source for example Data Environment designer. You can also export the report into text files or Web pages (HTML).

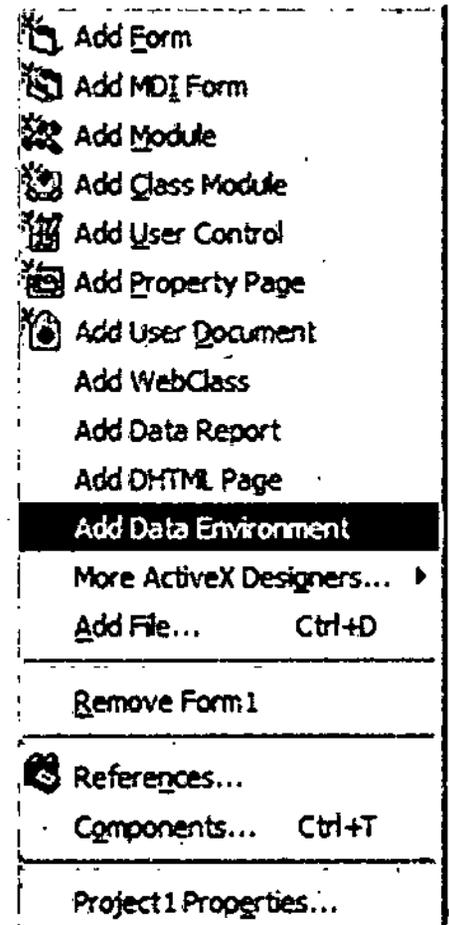
### NOTES

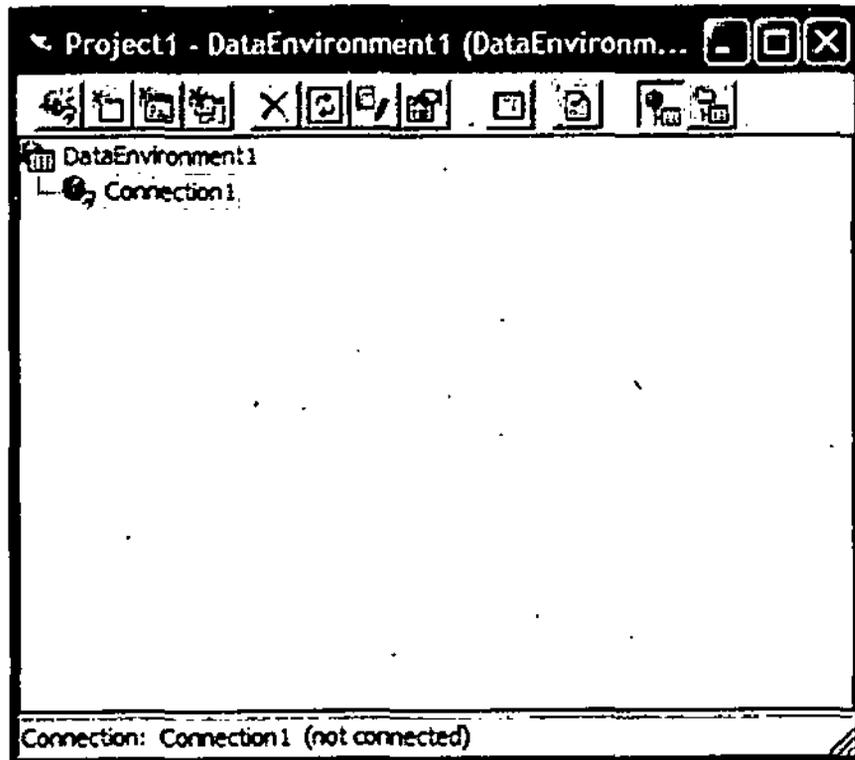
#### Data Report Designer Features

- **Print Preview:** You can take the print previews.
- **Toolbox Controls:** It is having its own set of controls.
- **Drag-and-Drop Functionality for Fields:** You can drag the fields from the Microsoft Data Environment designer to the Data Report designer.
- **File Export:** You can export the data in text and HTML files.
- **Asynchronous Operation:** PrintReport and ExportReport methods support the asynchronous operations.
- **Export Templates:** Using it you can export the reports in many formats.

#### Practice Yourself

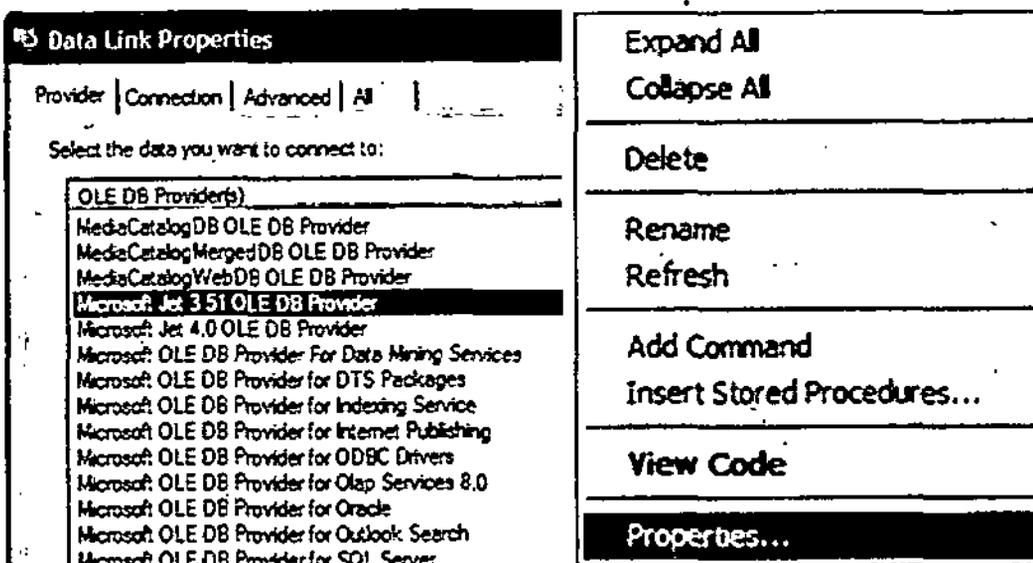
1. Start Visual Basic 6.
2. Select Standard Exe from New tab.
3. Click Ok.
4. On menu bar click File.
5. Click Save Project.
6. Save File As dialog box appears.
7. Select the desired drive.
8. In File name field keep default name form1.frm.
9. Click Save.
10. Save Project As dialog box appears.
11. In File name field keep default name project1.vbp.
12. Click Save.
13. Source Code Control dialog box appears.
14. Click No.
15. Click on Project
16. Click Add Data Environment
17. You will get DataEnvironment1 window, as shown on the next page.
18. Now right-click the Connection1
19. Click Properties



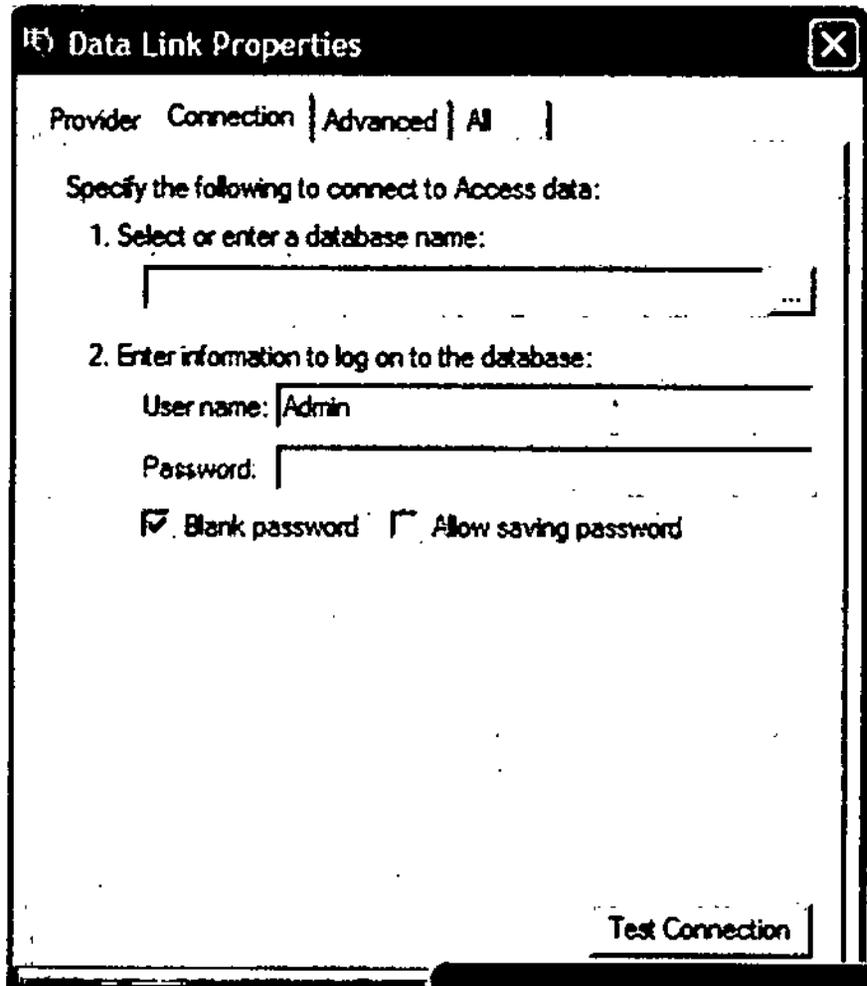


NOTES

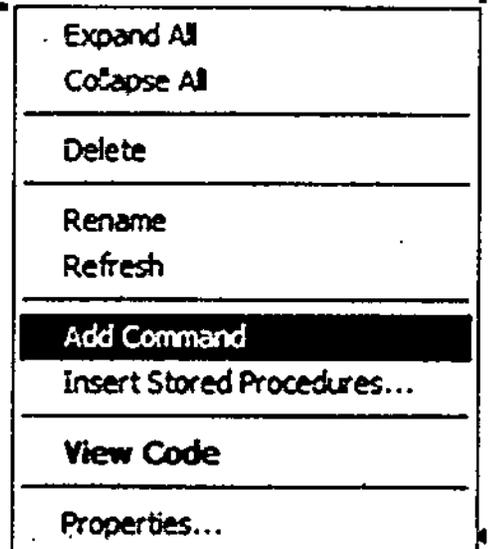
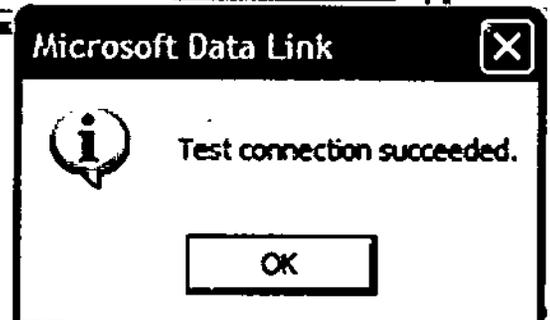
20. You will get Microsoft Jet 3.51 OLE DB Provider
21. Click Next
22. You will Data Link Properties window.
23. Now click ellipses button for selecting Access database.
24. I'm selecting Biblio.MDB database which is available in VB98 folder under Microsoft Visual Studio directory, place where you have installed Visual Studio 6, as can be seen on the next page.
25. Now click Test Connection to check that connection is properly established or not.

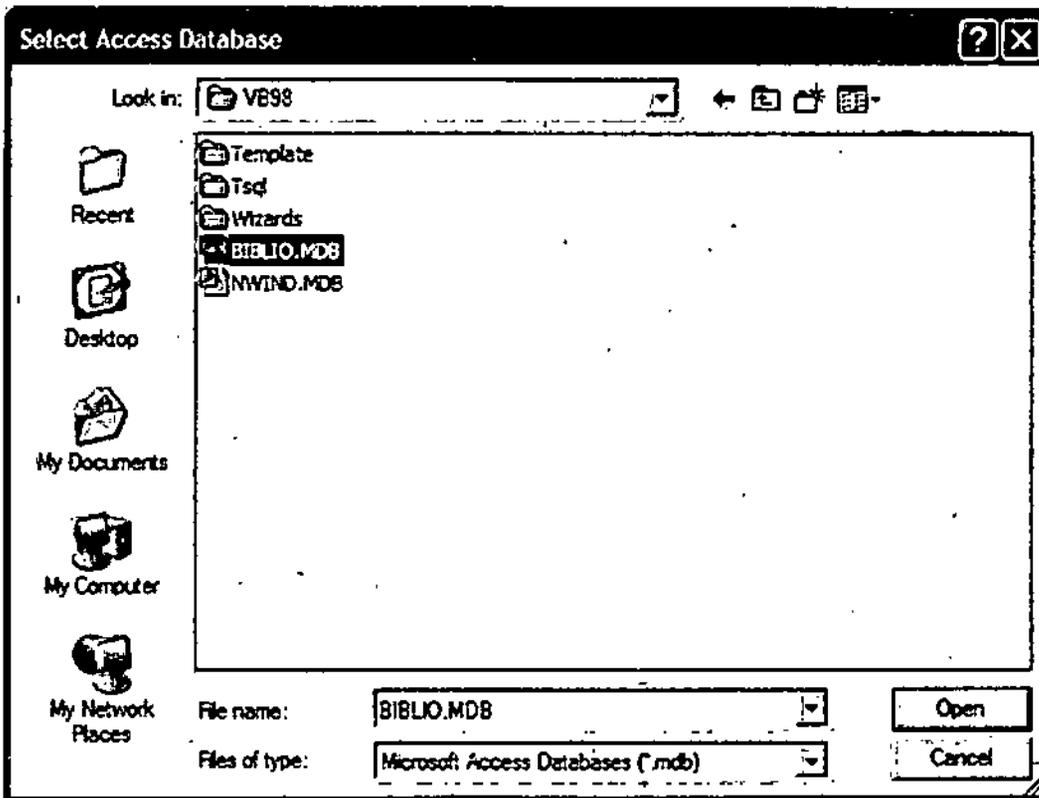


NOTES



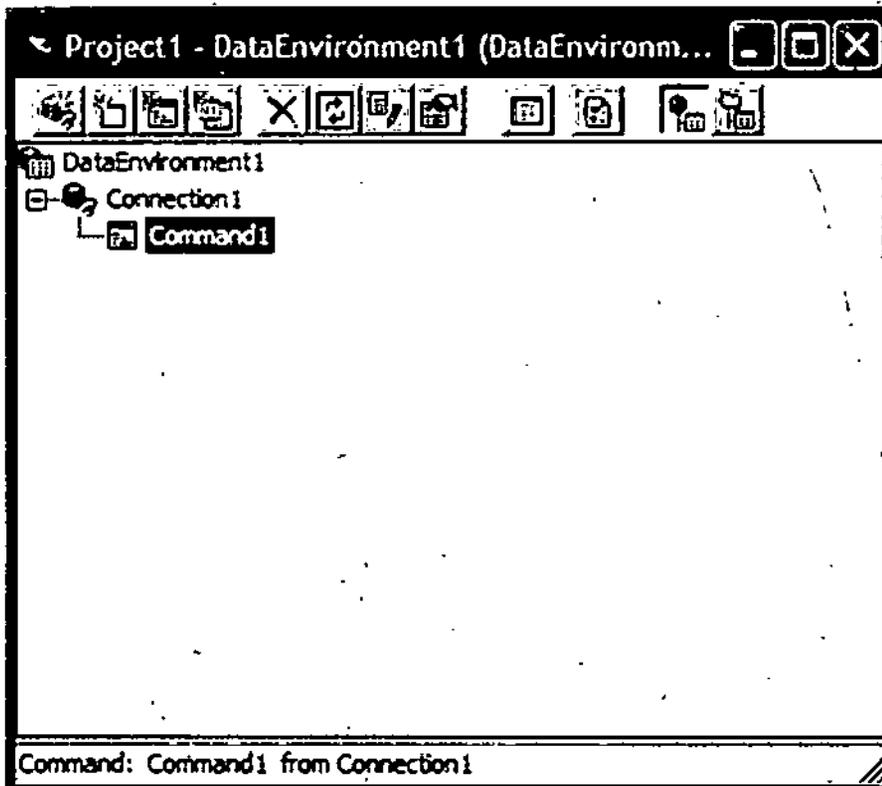
26. Click OK
27. Click OK
28. Now right-click Connection1
29. Click Add Command
30. Now right-click Command1 and click Properties to get the figure shown on the next page.
31. You will get Command1 Properties dialog box, shown on the next page.
32. In Database Object select Table
33. In Object Name select Authors, as shown on next to next page.
34. Click Apply
35. Click OK
36. On menu bar click Project
37. Click Add Data Report





NOTES

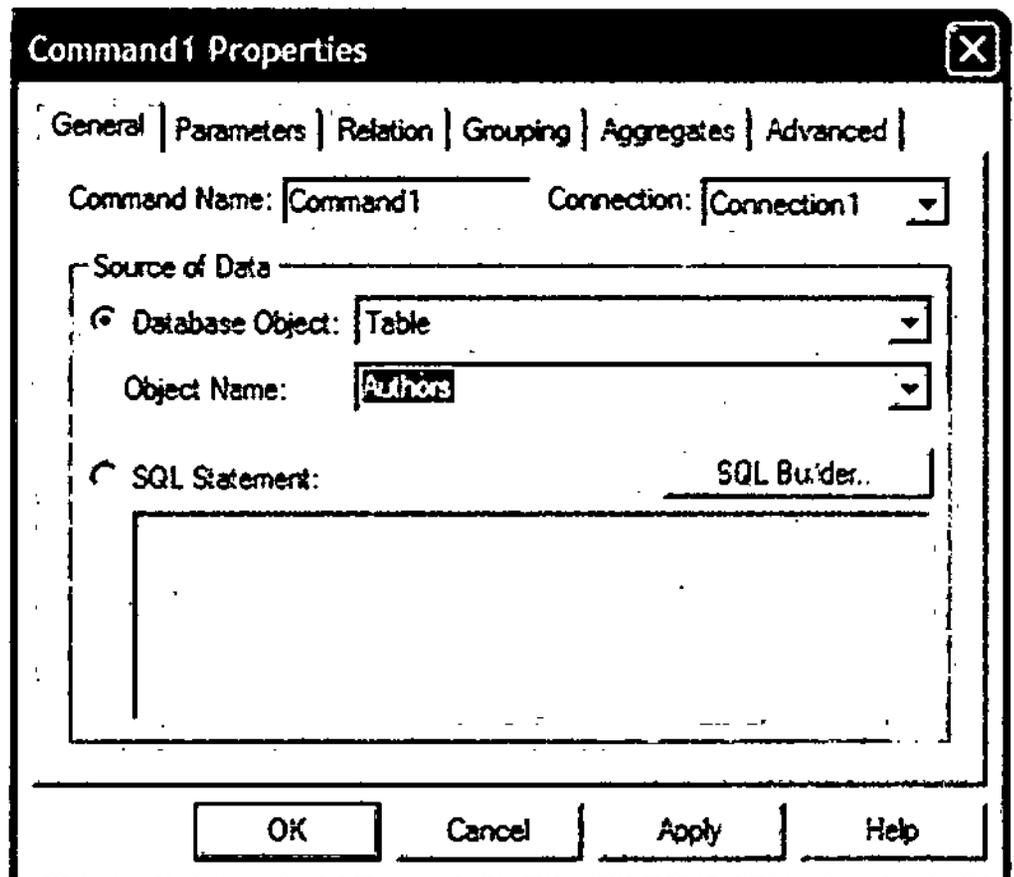
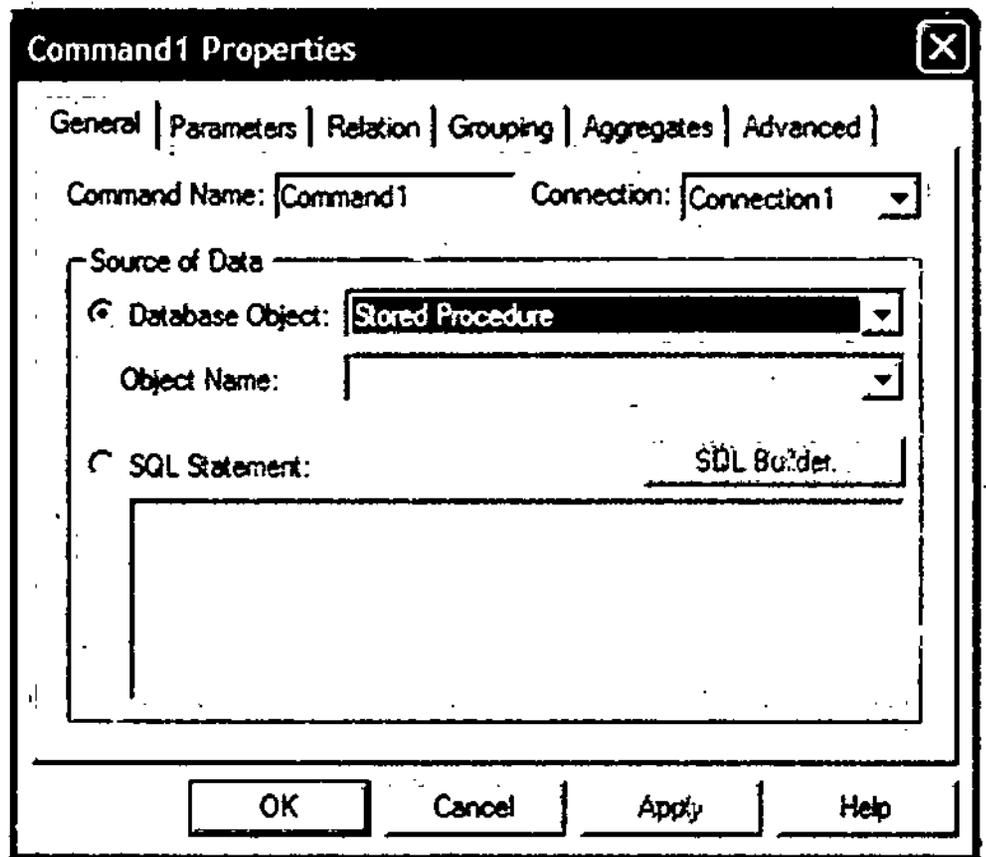
38. You will get DataReport1 window, as shown later.



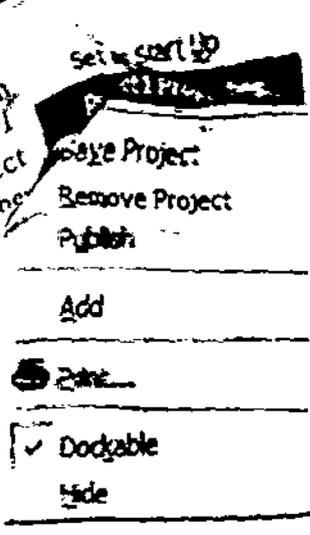
39. Single click the title bar of DataReport1 to confirm its selection.

40. In properties window in DataSource enter DataEnvironment1 and in DataMember field type Command1.

NOTES



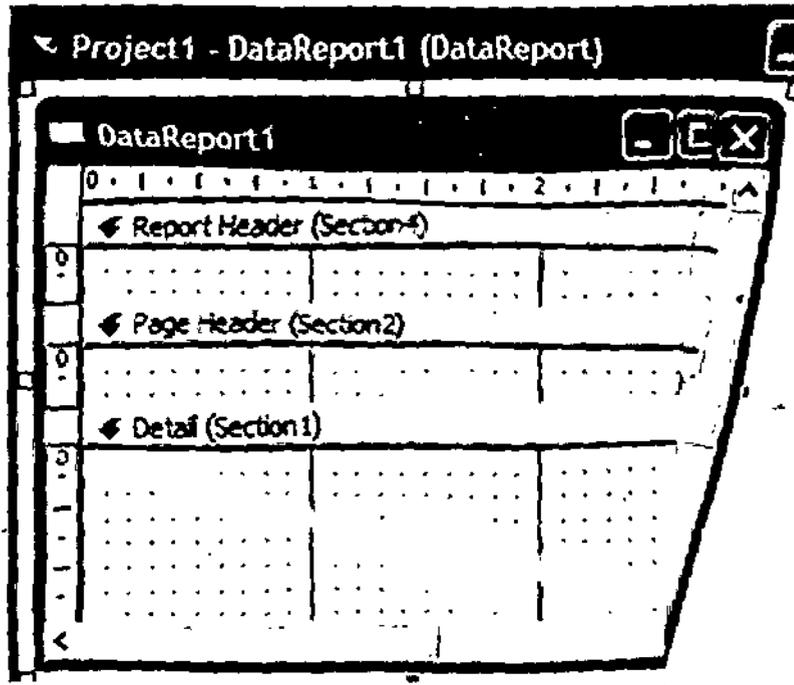
41. Now from DataEnvironment window click and drag Command1 and the Detail from this that icon has changed, as shown on the next to next page.



- Add Form
- Add MDI Form
- Add Module
- Add Class Module
- Add User Control
- Add Property Page
- Add User Document
- Add WebClass
- Add Data Report
- Add Data Page
- Add Data Environment

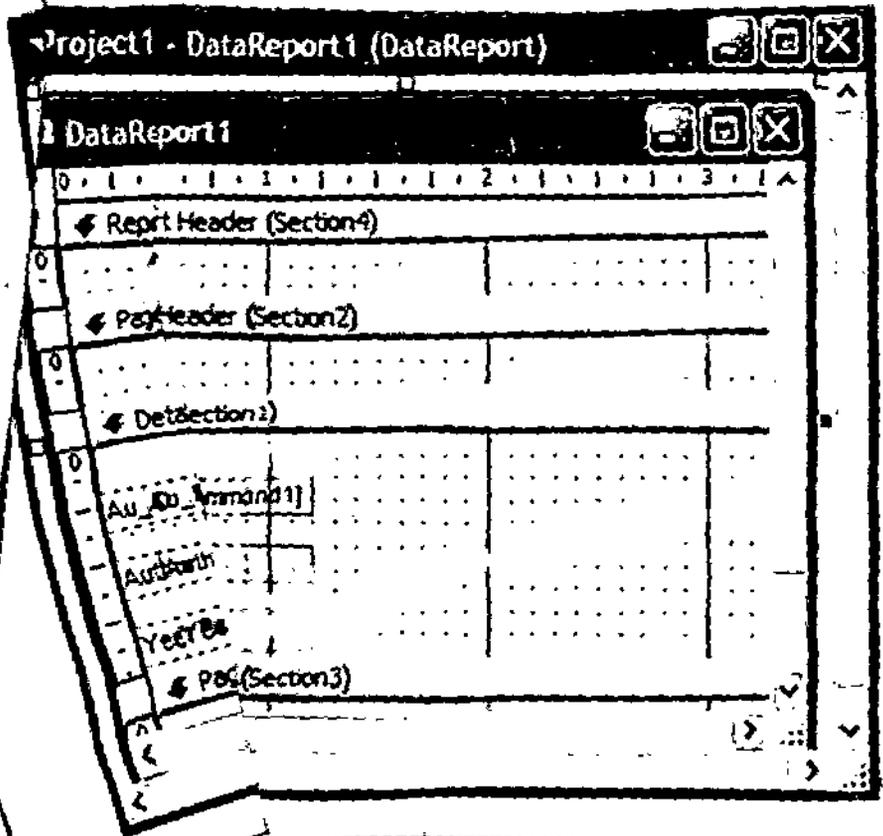
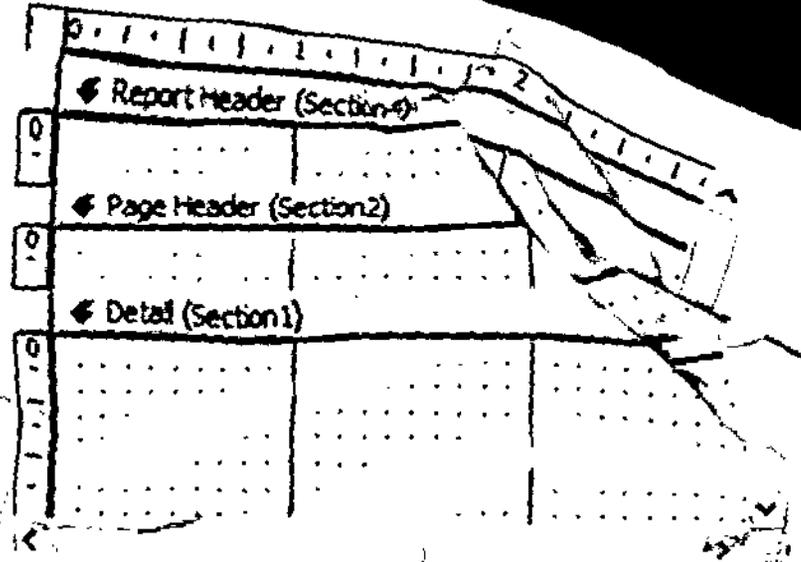
NOTES

- 43. Now in Project Window right-click the Project1Properties..., as shown here.
- 44. In Startup Object select DataReport1



- 45. Click OK
- 46. Now press F5 key to run the program.
- 47. You can see the whole report, even you can zoom in and take print out. Also from bottom located buttons you can also view records.

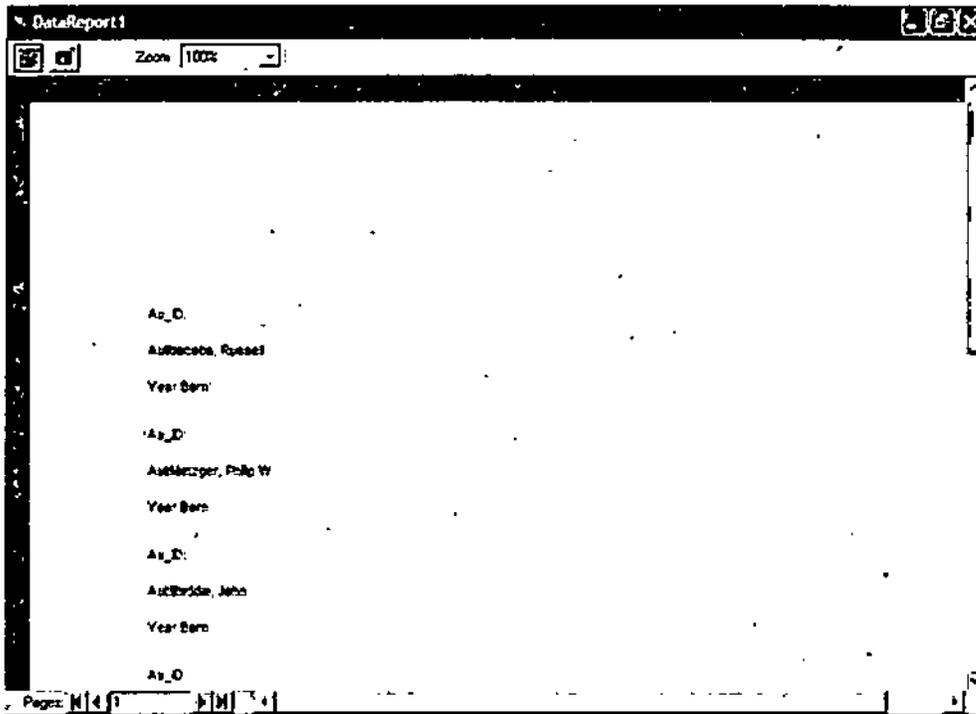
NOTES



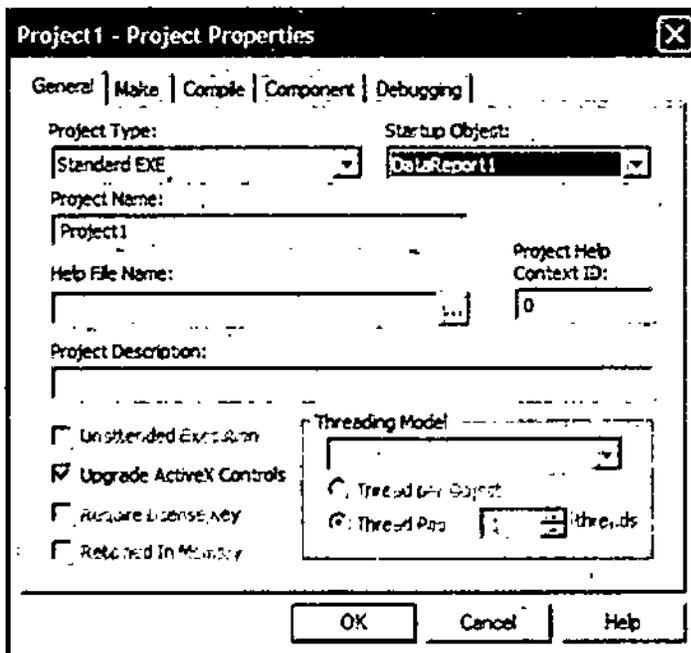
### FUNCTIONS—BUILT IN/USER DEFINED

There are two features in Visual Basic: subroutines and functions. Subroutines can be passed in parentheses but do not return a value; functions do return values (which can be discarded). A function is a block of code that has arguments to, and using functions helps break your code up into parts.

For reference's sake, to declare a function:



NOTES



[Private | Public | Friend] [Static] Function name  
 [(arglist)] [As type]

[statements]

[name - expression]

[Exit Function]





```
[statements]
```

```
End Function
```

## NOTES

The **Public** keyword makes a procedure accessible to all other procedures in all modules and forms.

The **Private** keyword makes a procedure accessible only to other procedures in the module or form in which it is declared.

The **Friend** keyword is used only in class modules and specifies that the procedure is visible throughout the project, but not visible to a controller of an instance of an object.

The **Static** keyword specifies that the procedure's local variables should be preserved between the calls.

The name identifier is the name of the procedure.

The arglist identifier is a list of variables representing arguments that are passed to the procedure when it is called. You separate multiple variables with commas.

The statements identifier is the group of statements to be executed within the procedure.

The arglist identifier as this following syntax:

```
[Optional] [ByVal| ByRef] [ParamArray] varname[( )] [As  
type] [- defaultvalue]
```

In arglist, **Optional** means that an argument is not required; **ByVal** means that the argument is passed by value; **ByRef** means that the argument is passed by reference (**ByRef** is the default in Visual Basic); **ParamArray** is used as the last argument in arglist to indicate that the final argument is an array of **Variant** elements; **varname** is the name of the variable passed as an argument; **type** is the data type of the argument; and **defaultvalue** is a constant, or constant expression, which is used as the argument's default value if you have used the **Optional** keyword. The type identifier is the data type returned by the function. The **Exit** function keywords cause an immediate exit from a Function procedure.

You call a Function procedure using the function name, followed by the argument list in parentheses. You return a value from a function by assigning the value you want to return to the function's name like this; **name = expression**. Finally, **End Function** ends the procedure definition.

Here is an example showing how to use a function.

```
Private Sub Command1_Click()  
    Dim intResult As Integer  
    intResult = Add1(5)  
    MsgBox ("Result = " & Str$(intResult))  
End Sub
```

```
Function Add1(intAddToMe As Integer) As Integer
```

End Function

**Text Functions**

Text functions are shown in the following table:

NOTES

<i>Function</i>	<i>Syntax</i>	<i>Description</i>	<i>Example</i>
LTrim	LTrim(string)	Returns a copy of a string without leading spaces	MyString = " <-Trim-> " TrimString=L Trim(MyString)  Output: " <-Trim-> "
RTrim	RTrim(string)	Returns a copy of a string with any trailing spaces removed	MyString = " <-Trim-> " TrimString=RTrim(MyString) Output: " <-Trim-> "
Trim	Trim( string)	Returns a copy of a string without leading and trailing spaces.	MyString = " <-Trim-> " TrimString=Trim(MyString) Output: " <-Trim-> "
StrComp	StrComp, (String1,  String2  [, compare])	Return the results of a string comparison	Dim MyStr1, MyStr2, MyComp MyStr1 = "ABCD": MyStr2 = "abcd" MyComp=StrComp(MyStr1, MyStr2, 1) Returns 0. MyComp = StrComp(MyStr1, MyStr2, 0) Returns -1. MyComp = StrComp(MyStr2, MyStr1), Returns 1.
StrReverse	StrReverse (expression)	Returns a string in which the character order of a specified string is reversed.	Dim MyStr MyStr = StrReverse("VBScript") Output: tpircSBV.
InStr	InStr ([ start, ] string1, String2 [, compare])	Returns the position of the first occurrence of one string within another.	Dim SearchString, SearchChar, MyPos SearchString ="XXpXXpXXPXXP" SearchChar = "P" MyPos = Instr(4, SearchString, SearchChar, 1) Output: 6.
Mid	Mid (string, start [, length])	Returns a Variant (String) containing a specified number of characters from a string.	Dim MyString, strString MyString = "Mid Function Demo" strString = Mid(MyString, 1,3) Output: Mid
LCase	LCase (string)	Returns a String that has been converted to lowercase	Dim UpperCase, LowerCase Uppercase = "Hello World 1234" Lowercase = Lcase(Uppercase) Output: hello world 1234
UCase	UCase (string)	Returns a String that has been converted to uppercase.	Dim LowerCase, UpperCase LowerCase = "Hello World 1234" UpperCase = UCase(LowerCase) Output: HELLO WORLD 1234

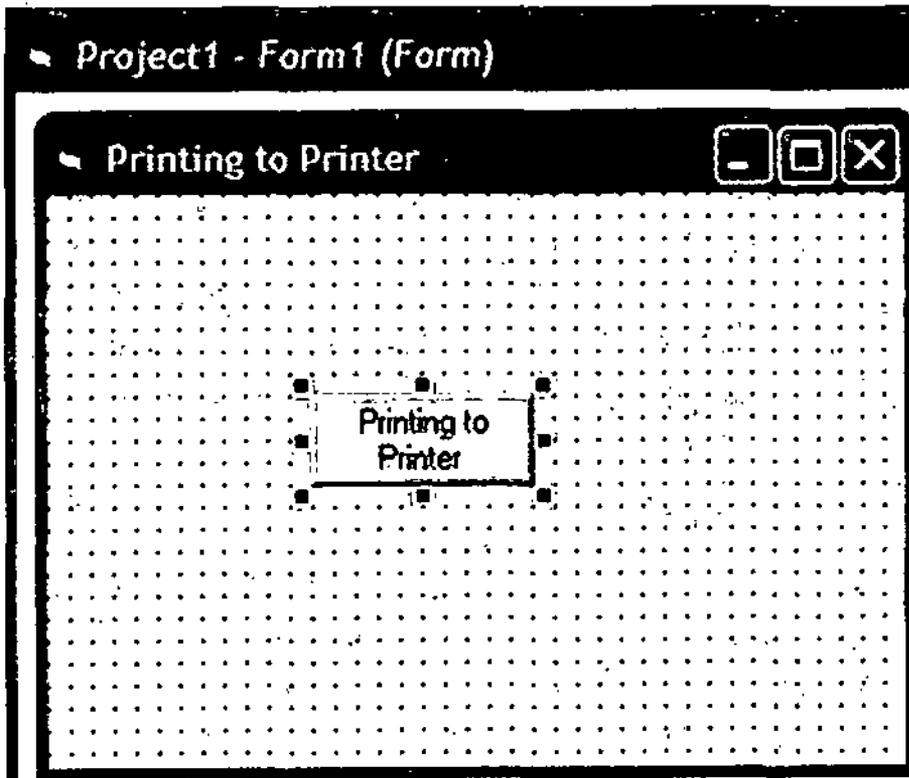
NOTES

Asc	Asc (string)	Returns the numeric code of a character	Dim MyNumber MyNumber = Asc("A") Output: 65
Chr	Chr (charcode)	Returns the corresponding character of the supplied code.	Dim MyChar MyChar = Chr(65) Output: A
Format	Format (expression [, format [, firstdayofweek [, firstweekofyear] ] ] )	Returns a formatted number, date or time MyStr = Format(MyTime, "h:m:s")	Dim MyTime, MyDate, MyStr MyTime = #17:04:23# MyDate = #January 27, 1993# Output: "17:4:23"
Left	Left (string, length)	Returns a specified number of characters from the left side of a string.	Dim AnyString, MyStr AnyString = "Hello World" MyStr = Left(AnyString, 1) Output: H
Right	Right (string, length)	Returns a specified number of characters from the right side of a string.	Dim AnyString, MyStr AnyString = "Hello World" MyStr = Right(AnyString, 1) Output: H
Len	Len (string)	Returns the length of a string	Dim MyString, MyLen MyString = "Hello World" My Len = Len (MyString) Output: 11
Space	Space (number)	Returns a string consisting of a specified number of spaces	Dim MyString MyString = Space(10) Output: Returns a string with 10 spaces.

**Date/Time Functions**

Date/Time functions are shown in the following table.

Function	Syntax	Description	Example
Date	Date ( )	Returns the current system date.	Dim MyDate MyDate = Date
Now	Now ( )	Returns the current date and time according to your computer's system date and time.	Dim Today Today = Now
Day	Day(date)	Returns a whole number between 1 and 31, inclusive, representing the day of the month.	Dim MyDate, MyDay MyDate = #February 12, 1999# MyDay = Day(MyDate) Output: -12.
Month	Month(date)	Returns a whole number between 1 and 12, inclusive, representing the month of the year.	Dim MyDate, MyMonth MyDate = #February 12, 1999# MyMonth = Month(MyDate) Output: 2.
Minute	Minute ( time )	Returns a whole number between 0 and 59, inclusive, representing the minute of the hour.	Dim MyTime, MyMinute MyTime = #4:35:17 PM# MyMinute = Minute(MyTime) Output: 35.



## NOTES

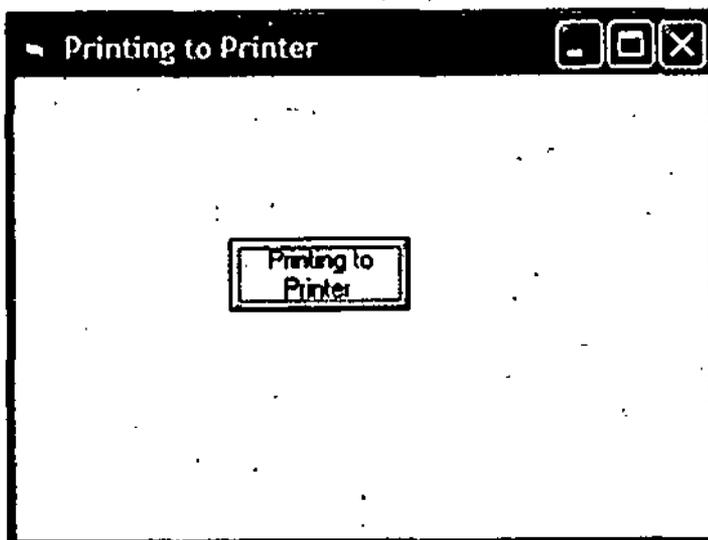
30. Double-click the button control to open code window.

31. Type the following code:

```
Private Sub Command1_Click()  
Printer.Print  
End Sub
```

32. Press F5 key on your keyboard.

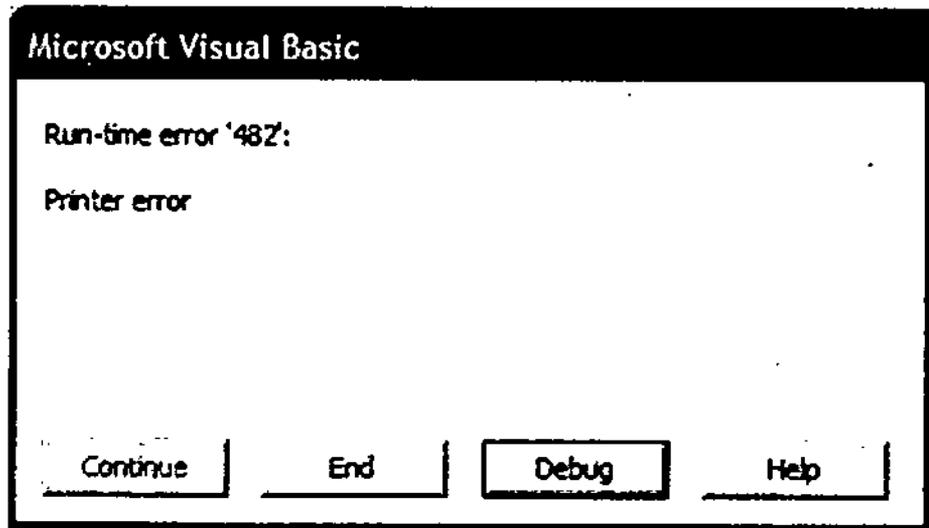
33. You will get following window:



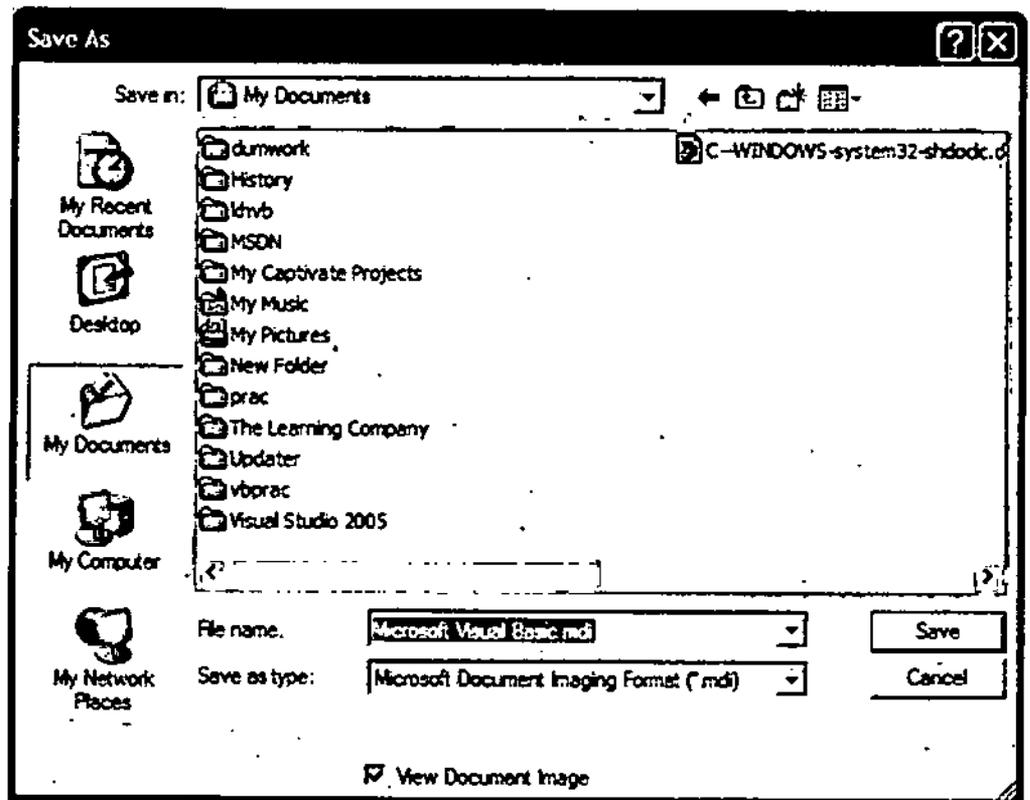
34. Click the button.

Note: If printer is not installed in your system then you can get following error:

NOTES



Note: If printer is not installed in your system then Windows can also force you to save as Microsoft Document Imaging Format.



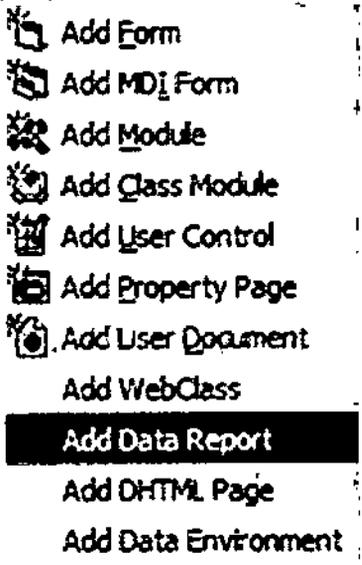
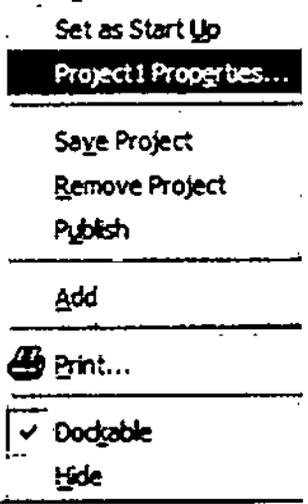
### PrintForm Method

PrintForm method is used to print an active form. This method sends a pixel-by-pixel image of a form to the printer.

For example:

```
Form1.PrintForm
```

41. Now from Dataenvironment1 window select Command1 and then click and drag under Detail section of Data Report1 window, as shown on the next page.

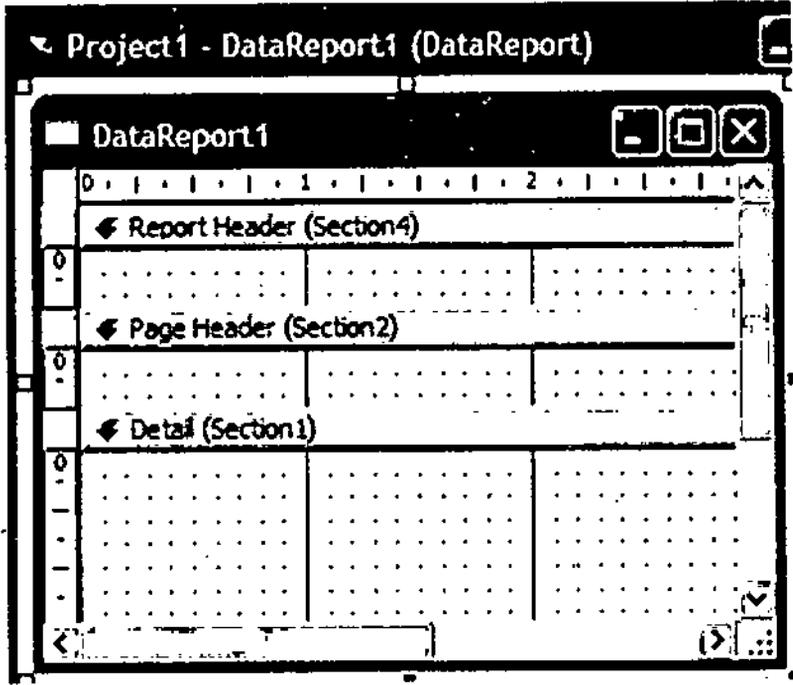


42. You can see this that your mouse icon has been changed, as shown on the next to next page.

43. Now in Project Window right-click the Project1Properties..., as shown here.

44. In Startup Object select DataReport1

NOTES

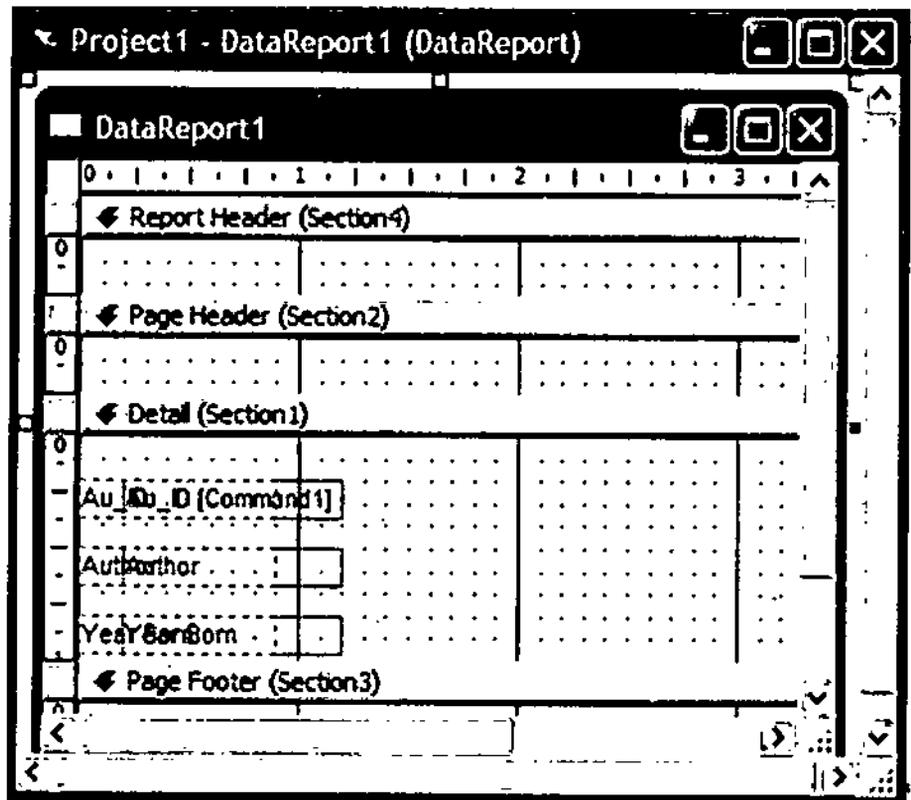
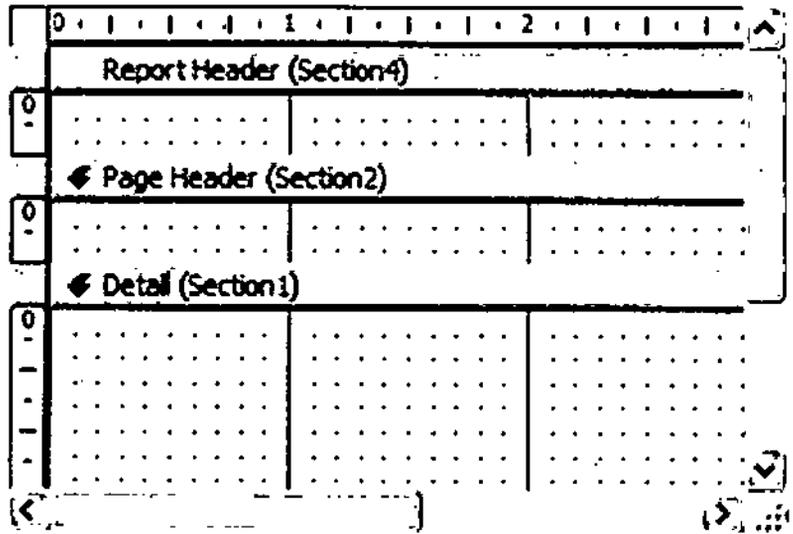


45. Click OK

46. Now press F5 key to run the program.

47. You can see the whole report, even you can zoom in and out. You can also take print out. Also from bottom located buttons you can navigate through records.

NOTES



## FUNCTION PROCEDURES—BUILT IN/USER DEFINED

There are two types of procedures in Visual Basic: subroutines and functions. Subroutines can take arguments passed in parentheses but do not return a value; functions do the same but do return values (which can be discarded). A function is a block of code that you call and pass arguments to, and using functions helps break your code up into manageable parts.

For reference's sake, here is how you declare a function:

Hour	Hour (time)	Returns a whole number between 0 and 23, inclusive, representing the hour of the day.	Dim MyTime, MyHour MyTime = #4:35:17 PM# MyHour = Hour(MyTime) Output: 16
DateAdd	DateAdd (interval, number, date)	DateAdd(interval, number, date)	DateAdd("m",1,"05-Dec-02") Output: 01/05/03
DateDiff	DateDiff (interval, date1, date2 [, first dayofweek [, firstweek of year]])	Returns a Variant (Long) specifying the number of time intervals between two specified dates.	DateDiff("d","13-Jun-99", "13-Oct-99") Output: 122
DatePart	DatePart (interval, date [,firstdayofweek [, firstweekofyear]])	Returns a Variant (Integer) containing the specified part of a given date.	

NOTES

## ARRAYS

An array is an ordered series of data values, called elements that are referenced by number. Because arrays exist in memory, they provide fast data access and ease of manipulation. You can easily specify, locate or manipulate elements in an array. For example, you cannot have one element of the Integer data type and another of the Boolean data type in the same array. Each element of an array has a unique identifying index number. Changes made to one element of an array do not affect the other elements. To refer to any one of the values in an array, you need to use an index.

*An array is a set of sequentially indexed elements having the same type of data.*

Arrays should be defined before they are used in the program. The definition of an array specifies the name of the array and the number elements it can hold.

### Various Features of an Array are:

- An array is a data structure that lets a single variable store multiple values of the same type
- When you declare an array, you must specify the type of value the array will store as well as the number of items the array will hold.
- Each element within an array must be the same type, such as int, float, or char.
- To store a value within an array, you specify the element number within the array at which you want to store the value. For example, the array's first element is element 0, the second is element 1, and so on.
- To access a value stored within an array, your programs specify the array name and the element number, placing the element number within left and right brackets, such as scores{9}.
- When your program declares an array, it can use the assignment operator to initialise array elements.

- Your programs can pass array variables to functions just as they would any parameter.

### Arrays are of Different Types:

**one-dimensional arrays**, comprised of finite homogeneous elements.

**multi-dimensional arrays**, comprised of elements, each of which is itself an array.

A two-dimensional array is the simplest of multidimensional arrays. However, C++ allows arrays of more than two dimensions. The exact limit of dimensions, if any, is determined by the compiler you use.

### Single Dimensional Arrays

The simplest form of an array is a single dimensional array. The array is given a name and its elements are referred to by their subscripts or indices. An array definition specifies a variable type and a name along with one more feature size to specify how many data items the array will contain. The general form of an array declaration is as shown below:

```
type array-name [size] ;
```

where type declares the base type of the array, which is the type of each element in the array. The array-name specifies the name with which the array will be referenced and size defines how many elements the array will hold. The size must be an integer value or integer-constant without any sign.

The data type of array elements is known as the base type of the array. Following statement declares an array marks of base type int and which can hold 50 elements.

```
int marks [50] ;
```

The above statement declared array marks has 50 elements, marks [0] to marks [49].

A vector is a mathematical term, which refers to the collection of numbers which are analogous, i.e., a linear array (one dimensional arrays) and a vector can represent only integers and floating-point numbers.

### String as an Array

A string is defined as a character array that is terminated by a null character '\0'. For this reason, the character arrays are declared one character longer than the largest string they can hold. For instance, to declare an array strg that holds a 10-character string, you would write

```
char strg[11];
```

This makes room for the null character at the end of the string.

Individual characters of a string can be easily accessed as they make the elements of the character array, The index 0 refers to the first character, the index 1 to the second 2 to the third, and so forth. The end of a string is determined by checking for null character.

### Two-Dimensional Arrays

A two-dimensional array is an array in which each element is itself an array. For instance, an array A [M] [N] is an M by N table with M rows and N columns containing

NOTES

M x N elements. The number of elements in a 2-D array can be determined by multiplying number of rows with number of columns. For example, the number of elements in an array A [7] [9] is calculated as  $7 \times 9 = 63$ .

The simplest form of a multidimensional array, the two-dimensional array, is an array having simple-dimension arrays as its elements. The general form of a two-dimensional array declaration is as follows:

```
type array-name [rows] [columns];
```

where type is the base data type of the array having name array-name; rows, the first index, refers to the number of rows in the array and columns, the second index, refers to the number of columns in the array. Following declaration declares an int array sales of size 5, 12.

```
Int sales[5][12];
```

The array sales have 5 elements sales [0], sales [1], sales [2], sales [3] and sales [4] each of which is itself an int array with 12 elements. The elements of sales are referred to as sales [0][0], sales [0][1], . . . . . sales [0][11], sales [1][0], sales [1][1], and so forth.

### Array of Strings

An array of strings is a two-dimensional character array. The size of first index (rows) determines the number of strings and the size of second index (columns) determines maximum length of each string. The following code declares an array of 10 strings, each of which can hold maximum 50 valid characters.

```
char strings[10][51];
```

Notice that the second index has been given value 51, i.e., 1 extra to take care of the null character '\0'.

It is very easy to access an individual string, by just specifying the first index, an individual string can be accessed. For instance, from the array string[5][7] string[1] would give you "Second" ; string[4] would give you "Fifth".

### Array Initialization

The general form of array initialization is as shown below:

```
type array-name [size N] = {value-list};
```

The value-list is a comma-separated list of array elements' values. The element values in the value-list must have the same data type as that of type, the base type of the array. Following code initialises an integer array with 12 elements.

```
int days-of-month [12] = {31, 28, 31, 30, 31, 30, 31, 31,
    30, 31, 30, 31} ;
```

This would place first value 31 in month [1], 28 in month [2], and so on. Character arrays can also be initialized like this as shown below:

```
char string [10] = "Program" ;
```

The above code will initialize the string with "Program". Alternatively, above declaration can be written as :

NOTES

```
char string [10] = {'P', 'r', 'o', 'g', 'r', 'a', 'm',
                    '\0'} ;
```

Because all strings terminate with a null, you must make sure that the array you declare is long enough to include the null.

## NOTES

Two-dimensional arrays are also initialized in the same way as single-dimension ones. For example, the following code initialises a two-dimensional array cube with numbers 1 through 5 and their cubes:

```
int cube[5][2] =: { 1, 1,
                   2, 8,
                   3, 27,
                   4, 64,
                   5 125}
```

**Unsigned Array Initializations**

Following are some examples of unsigned array initializations:

```
char s1 [ ] = "First string" ;
int val [ ] = {2, 3, 4, 5, 6, 7, 8, 9} ;
float amount [ ] = {2341.57, 1900.70, 3986.65, 4466.80,
                   5191.00};
```

```
int cube [ I [2] = {1, 1,
                   2, 8,
                   3, 27,
                   4, 64,
                   5, 125};
```

The advantage of this declaration is that you may lengthen or shorten the value-list without changing the array dimensions.

**Array Characteristics**

Variable can hold only one data but using arrays you can store multiple data by declaring one array variable. In Visual Basic you can refer to these different data by same name or you can use numbers which are indexes. Think this that this different data are elements which are continuously stored in between upper and lower bounds.

**Note:** All the data or elements in an array should be of same data type but if data type is declared as Variant then they can hold different types of data.

In Visual Basic 6 you can define two types of Arrays:

- Fixed-size array
- Dynamic array

**Fixed-size array:** The size of fixed-size arrays is same, means you are not changing it. In this you can create local, public and module-level arrays.

**Dynamic array:** The size of dynamic array can be changed at run-time.

Examples for creating or declaring array:

```
Dim studentsname(10) as string
Dim studentsage(20) as integer
```

## Array Declaration

It is very easy to declare an array. For this you have to give the size and data type of array. If you do not give the data type then declared array will be considered as a variant. To declare an array with in a procedure you can use Dim before array declaration.

For example:

```
Dim x(4) as string
```

To create a public array use public before variable name.

For example:

```
Public x(5) as string
```

## Processing Array Elements

To store value in an array element you can simply assign the value directly inside it.

For example:

```
X(0) = "Raj"
```

You can assign values to all created array elements.

For example:

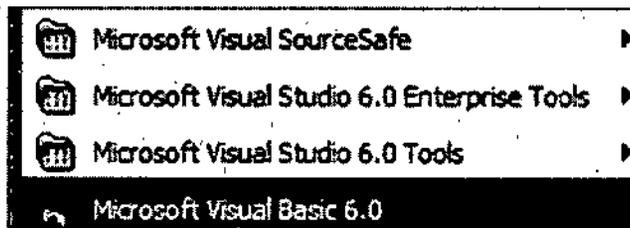
```
X(0) = "Raj"
X(1) = "Mohit"
X(2) = "Priyanka"
X(3) = "Simi"
```

In above example this array will hold 4 elements but remember this that in Visual Basic index starts from 0. To store the value you have to assign them inside the array. But if you want to access the first element then you have to use index 0.

The following example will demonstrate this:

### Example: Processing Array Elements

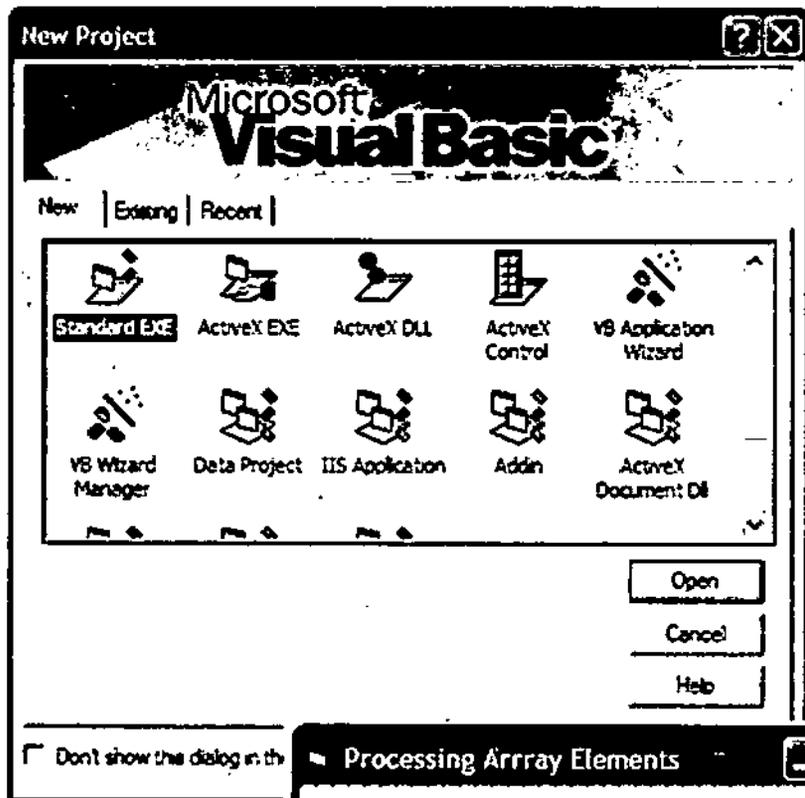
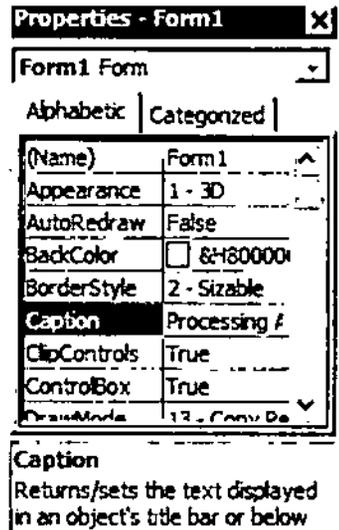
1. To start Visual Basic 6, click on start button.
2. Position your mouse over All Programs.
3. Position your mouse over Microsoft Visual Studio 6.0 .
4. Click Microsoft Visual Basic 6.0
5. New Project dialog box appears:
6. Select Standard EXE.
7. Click Open button.
8. Select a Form by single clicking on it.



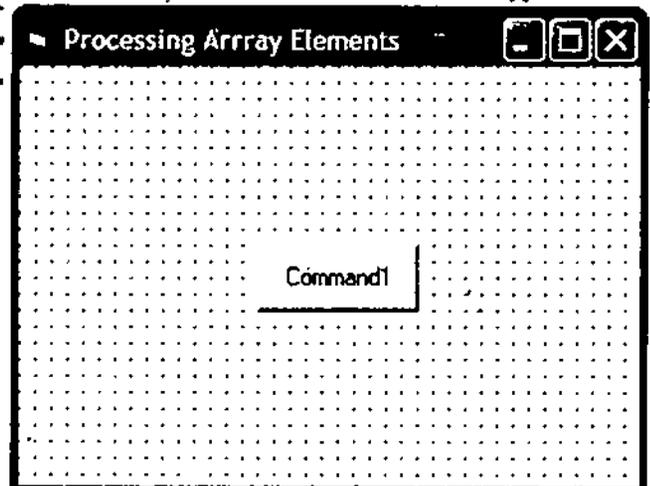
NOTES

NOTES

9. In Properties Window locate the Caption property and Processing Array Elements.
10. Add a button control to Form.
11. At present your Form looks like the one shown next.
12. Select the button control.
13. In Properties window locate the Caption property and type, Processing Array Elements.
14. Locate the Width property and type 2535.
15. Locate the Height property and type 855.
16. Locate the Left property and type 1080.
17. Locate the Top property and type 960.

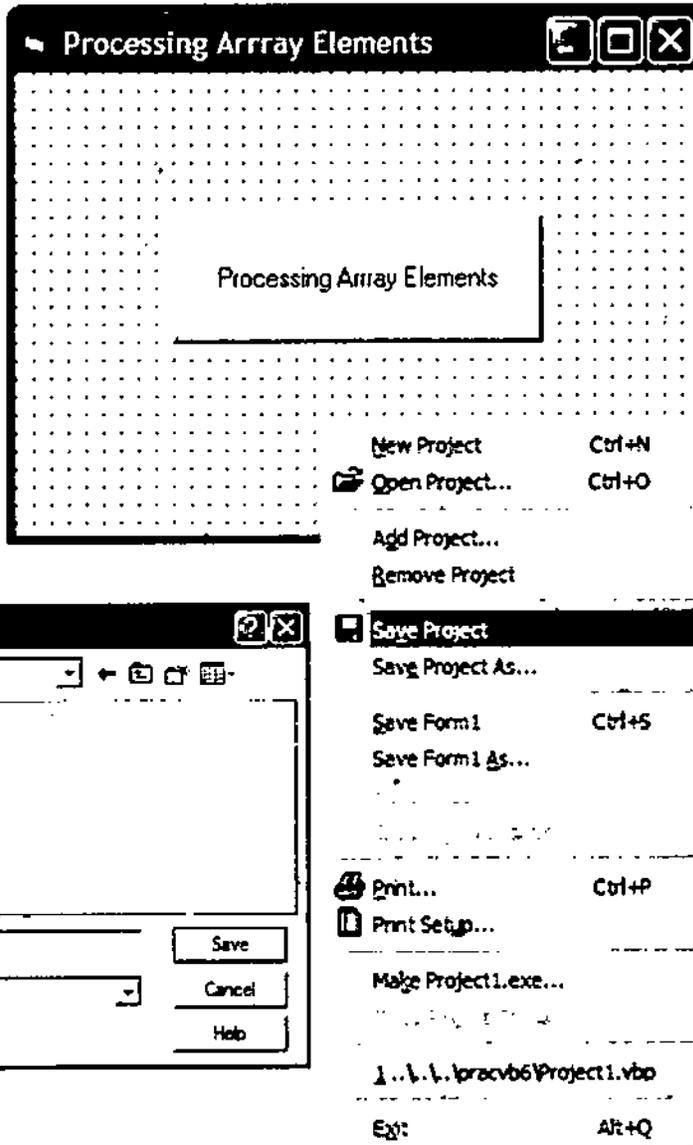


18. Now your Form looks like this:
19. Now let us first save the project so on menu bar click on File.
20. Click Save Project.
21. Save File As dialog box appears:

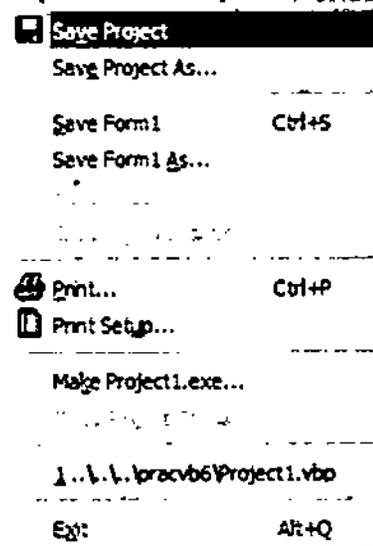
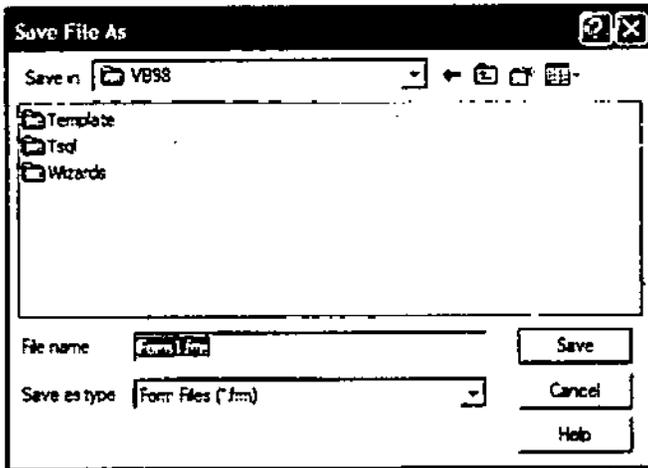


22. In Save in field select C: or any desired drive and folder.
23. Keep all default names and click on Save button.
24. Again Click on Save button to save Project with default name.

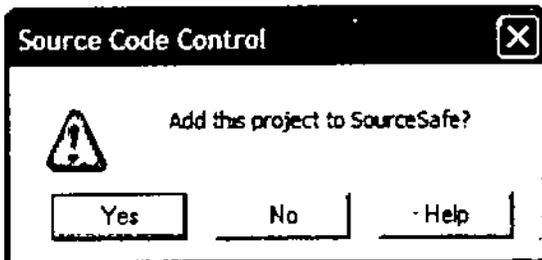
Note: You can give any other desired name of form or project.



NOTES



25. Source Code Control dialog box appears:



26. Click No.
27. Double-click the command button to open code window.
28. Type the following code:

```
Private Sub Command1_Click()

Dim x(4) As String
```

```
x(0) = "Raj"  
x(1) = "Mohit"  
x(2) = "Priyanka"  
x(3) = "Simi"
```

NOTES

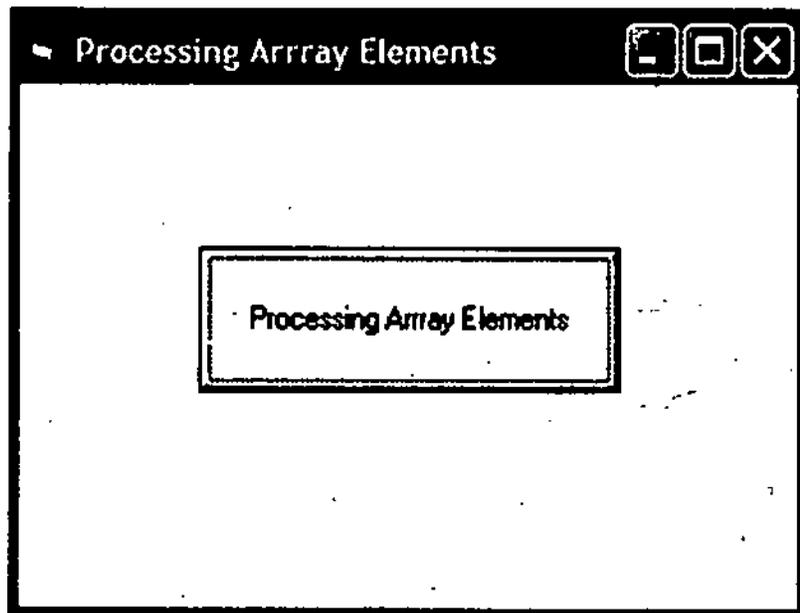
```
For i = 0 To 3  
MsgBox (x(i))  
Next
```

End Sub

29. To run the program click on Start button.



30. You will get the output shown next.



31. Click on the button to check results.

### Dynamic Arrays

In this you can resize an array any time. Specialty is you can decrease the size of array any time if it is not longer used to free up the memory. For this you use Redim statement.

For example:

```
ReDim zarr(y + 1)
```

Redim is appeared only in procedures. It is a executable statement. Redim can change the upper and lower bounds but you can not change number of dimensions in an array.

To preserve the contents of dynamic arrays you can use Preserve keyword.

For example:

```
Redim Preserve x(10)
Array Related Functions
Lbound
```

It displays the first index of declared array.

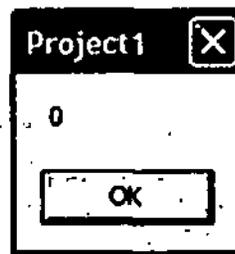
For example:

```
Private Sub Command1_Click()

Dim x(3) As String
x(0) = "raj"
x(1) = "shyam"
x(2) = "lola"
MsgBox LBound(x)

End Sub
```

The above code will display the output shown here.



### Ubound

It displays the last index of declared array.

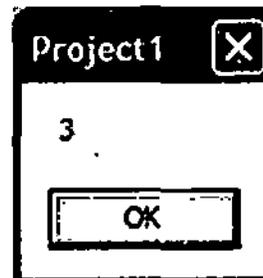
For example:

```
Private Sub Command1_Click()

Dim x(3) As String
x(0) = "raj"
x(1) = "shyam"
x(2) = "lola"
MsgBox UBound(x)

End Sub
```

If you run the above code you will get the output shown here.



### Control Arrays

Control Array is a group of controls which share the same type, event procedures and name. Like this you can save the resources because you are not adding same controls again and again you simply make a copy of one. Control array should contain *minimum one element*. 32767 is the highest index which you can use in control array.

Creating a Control Array at design time

You have three ways for creating a control array at design time:

NOTES

- You can assign the same name to more than one control
- You can first copy an existing control and then you can paste it on the form.
- You can set the control's index property to such value which is not null.

You can also create an instance of control at run time but that control should be a member of control array.

NOTES

---

## GRIDS

---

Grid controls display data in a table-like form, with rows and columns of cells. In fact, you can use grids to do just that; display tables of data. You can also use them to display spreadsheets.

Visual Basic has a number of grid controls; the data grid control, the flex grid control, and the hierarchical flex grid control. Like charts, grids give you a way of displaying data. Whereas charts present data in graphical format, grids appear like spreadsheets (and, in fact, if you want to create a spreadsheet in Visual Basic, you use a grid). A grid presents the user with a two-dimensional array of individual cells. You can make the cells in the grid active just as you had expect in a spreadsheet; for example, you can keep a running sum at the bottom of columns of data.

### Adding a Flex Grid Control to a Program

You can add a flex grid to a Visual Basic project easily; just follow these steps:

1. Select the Project|Components menu item.
2. Click the Controls tab in the Components dialog box.
3. Select the Microsoft FlexGrid Control entry in the Components dialog box.
4. Close the Components dialog box by clicking on OK. This displays the Flex Grid Control tool in the toolbox.
5. Add a flex grid control to your form in the usual way for Visual Basic controls, using the Flex Grid Control tool.
6. Set the flex grid's Rows and Cols properties to the number of rows and columns you want in your flex grid. You can also customize your flex grid by setting such properties as BorderStyle, ForeColor, BackColor, and so on.

### Working with Data in a Flex Grid Control

Flex grids have FixedCols and FixedRows properties, which set the header columns and rows in the flex grid. These columns and rows are meant to label the other columns and rows, and they appear in gray in default (the other cells are white by default). Both FixedCols and FixedRows are set to 1 by default.

We will add a column of numbers here, so we can also place labels in the first column of cells, "Item 1" to "Item 6", and a label at the bottom, "Total", to indicate that the bottom row holds the total of the six above. These labels are not necessary, of course, but we will add them to show that you can use text as well as numbers in a flex grid. These labels will appear in column 1 of the flex grid, and users can place the data they want to add in column 2. The running sum appears at the bottom of column 2, as shown next.

	A	B	C	D	E
1	Item 1	10			
2	Item 2	5			
3	Item 3	18			
4	Item 4	2			
5	Item 5	15			
6	Total	50			

NOTES

To set text in a flex grid cell, you set the Row and Col properties to that location and then place the text in the flex grid's Text property. Here is how we set up the row and column labels in MSFlexGrid1 when the form loads:

```

Sub Form_Load()
    Dim Item(6) As String
    Dim intLoopIndex As Integer

    Items(1) = "Item 1"
    Items(2) = "Item 2"
    Items(3) = "Item 3"
    Items(4) = "Item 4"
    Items(5) = "Item 5"
    Items(6) = "Total"

    For intLoopIndex = 1 To MSFlexGrid1.Rows - 1
        MSFlexGrid1.Col = 0
        MSFlexGrid1.Row = intLoopIndex
        MSFlexGrid1.Text = Str(intLoopIndex)
        MSFlexGrid1.Col = 1
        MSFlexGrid1.Text = Items(intLoopIndex)
    Next intLoopIndex
    MSFlexGrid1.Row = 0
    For intLoopIndex = 1 To MSFlexGrid1.Cols - 1
        MSFlexGrid1.Col = intLoopIndex
        MSFlexGrid1.Text = Chr(Asc("A") - 1 + intLoopIndex)
    Next intLoopIndex

    MSFlexGrid1.Row = 1

```

```
MSFlexGrid1.Col = 1
```

```
End Sub
```

We have set up labels as we want them—but what about reading data when the user types it? We can use the flex grid's KeyPress event for that:

```
Sub MSFlexGrid1_KeyPress(KeyAscii As Integer)
End Sub
```

If the user enters numbers in the cells of column2, we will add those values together in a running sum that appears at the bottom of that column, just as in a real spreadsheet program. To enter a number in a cell, the user can click the flex grid, which sets the grid's Row and Col properties. Then, when the user types, we can add that text to the cell:

```
Sub MSFlexGrid1_KeyPress(KeyAscii As Integer)
    MSFlexGrid1.Text = MSFlexGrid1.Text + Chr$(KeyAscii)
End Sub
```

This represents one way of letting the user enter text into a grid, but notice that we had have to handle all the editing and deleting functions ourselves this way.

Now the user has changed the data in the spreadsheet, we add the numbers in column 2 this way:

```
Sub MSFlexGrid1_KeyPress(KeyAscii As Integer)
    Dim intRowIndex As Integer
    Dim Sum As Integer

    MSFlexGrid1.Text = MSFlexGrid1.Text + Chr$(KeyAscii)

    MSFlexGrid1.Col = 2
    Sum = 0

    For intRowIndex = 1 To MSFlexGrid1.Rows - 2
        MSFlexGrid1.Row = intRowIndex
        Sum = Sum + Val(MSFlexGrid1.Text)
    Next intRowIndex
```

Note that each time you set the Row and Col properties to a new cell, that cell gets the focus. Because we want to place the sum of column 2 at the bottom of that column, that's a problem.

---

## RECORDS

---

Database management deals with the storage, retrieval and editing of data within the database. Important functions of Database management are the security and

NOTES

validations of data items in Database. Visual Basic makes it simple to handle database by supplying a variety of tools and methods for data management. Visual Basic version 6.0 has a full-fledged database engine called **Jet**. This engine contains virtually all the database functions that many applications would need.

*Using Visual Basic version 6.0 you can build complex Client/Server database management systems having remote access feature.*

NOTES

Visual Basic also provides a powerful and easy-to-use front-end development environments for database management. Using this, you can manipulate database in various formats. It also includes an in-built support for **Open Database Connectivity (ODBC)** – an industry standard for data access from multiple database formats.

A multi-user database system, if implemented on PC, generally falls into one of the two categories:

- **File-server system**
- **Client/Server system**

In a file-server system, no intelligent process is active at the server level. All intelligence resides on the individual workstation. When your application needs data, it is the workstations software responsibility to determine which file should be read and how to access the network drive. Because all data in the database file must be sent from the server to the workstation, network traffic is increased.

A multi-user database system within a file-server environment is made up of the following:

- A database that resides on a network file server running an operating system such as **Microsoft Windows 2000 Server** or **Novell NetWare**.
- One or more users accessing the database using a workstation's application program.

In a Client/Server system, the server is responsible for intelligently processing requests for data. The workstation does not request data at the file level. It sends a request to the server to execute a specific query and gets the results. The primary advantage of this technique is that network traffic is reduced because server sends the result to the workstation.

A Client/Server system typically has a back-end database residing on the server controlled and maintained by the server software, such as **Microsoft SQL Server**.

*Tip: One or more users running local application that requests data from the server through an interface such as the Microsoft Open Database Connectivity (ODBC) standard.*

Programming database objects is an enormously complex topic that in itself can take up a dozen volumes. In this chapter, we are going to study about the various types of data access controls and how you can use them to access your database. Let me give you a brief introduction about them first.

## DAO

Working with DAO, you can use the Database and Recordset **Data Access Objects** in your procedures. The Database and Recordset objects each have properties and

## NOTES

methods of their own and you can write procedures that use these properties and methods to manipulate your data. To open a database in DAO, you just open a Database object or create a new one. This object can represent a **Microsoft Jet database** (.mdb) file, an **ISAM database** (for example, Paradox), or an ODBC database connected through the Microsoft Jet database engine. When the Database object is available, you create a Recordset object and use that object's methods, like *MoveFirst* and *MoveNext*, to work with the database.

DAO also supports a client/server connection mode called **ODBCDirect**. ODBCDirect establishes a connection directly to an ODBC data source, without loading the Microsoft Jet database engine into memory and is a good solution when you need ODBC features in your program.

In the ODBCDirect object model, the Connection object contains information about a connection to an ODBC data source, such as the server name, the data source name and so on. It is similar to a Database object; in fact, a Connection object and a Database object represent different references to the same object.

### RDO

With the **Remote Data Objects (RDO)** library of data objects, you establish an *rdoConnection* to an ODBC data source, then create an *rdoResultset*. The *Remote Data Objects* behave like the DAO objects in many ways, because there is a core set of methods that work with both record sets and result sets.

The big difference between DAO and RDO objects is that the RDO objects are largely SQL-driven. For example, although you can move through a database using methods like *MoveNext* and *MoveLast*, just as you would with the DAO objects, programmers often update and modify RDO data sources using SQL statements directly with the *rdoConnection* object's *Execute* method.

### ADO

**ActiveX Data Objects (ADO)** access data from OLE DB providers. The Connection object is used to specify a particular provider and any parameters. To connect to a data source, you use a Connection object. Using that connection, you can create a new record set, and using the Recordset object's methods and properties, you can work with your data.

An *ADO transaction* marks the beginning and end of a series of data operations that are executed across a connection. ADO makes sure that changes to a data source resulting from operations in a transaction either all occur successfully or not at all. If you cancel the transaction or one of its operations fails, then the result will be as if none of the operations in the transaction had occurred.

Later in the chapter, we'll see how to create connections using the ADO connection object and how to open data providers, creating an ADO Recordset object. We'll read data from the data provider and see how to display and modify it. In fact, we'll see how to support data-bound controls directly in code. Although the ADO model is a complex one, and OLE DB is even more complex, we'll see that many of the ADO Resultset methods are the same as the DAO - Resultset methods.

### Connecting with Database using DAO

Let us see how we can create a new database using DAO and use it.

## Using DAOCODE to Create and Edit a Database

To create a database file, select the New Database menu item. Next, add a table to that database with the New Table menu item, then add records to that table. When you're ready to store the database on disk, use the Close Database item.

*If you do not create a table in a database before trying to add data to a table in that database with the Add or Edit buttons, the DAOCODE program generates an error.*

NOTES

The program has buttons that let you add, edit, update and delete records, as well as letting them move through a database. Each time you want to add a record, including when you enter the first record of a new database, click the Add New Record button, type in the data for the record's fields and click the Update Database button to update the database. To edit a record, open the record, click the Edit button, edit the data in the record's fields and click the Update Database button to update the database.

For simplicity, this program only creates tables with two fields, although you can place as many records as you like in each table.

### DAO: Creating a Database

With Microsoft DAO Object Library you can create a DAO database. To add a reference to that library, select the Project I References menu item, select the Microsoft DAO Object Library and click on OK to close the References dialog box. Now we can make use of the data objects in that library to create a new database using CreateDatabase.

CreateDatabase is a method of the DAO Workspace object (there are a collection of Workspace objects in the DAO DBEngine object's Workspaces collection). Here's how you use CreateDatabase:

```
Set database = workspace.CreateDatabase (name, locale [, options])
```

Here are the arguments to CreateDatabase:

- **name** – A string up to 255 characters long that is the name of the database file that you're creating. It can be the full path and file name, such as C:\vbbb\db.mdb. If you don't supply a file name extension, .mdb is added.
- **locale** – A string that specifies a collating order for creating the database, like dbLangGeneral (which includes English), dbLangGreek, and so on.

Here are the possible settings for the options argument:

- **dbEncrypt** – Creates an encrypted database.
- **dbVersion10** – Creates a database that uses the Jet engine version 1 file format.
- **dbVersion11** – Creates a database that uses the Jet database engine version 1.1 file format.
- **dbVersion20** – Creates a database that uses the Jet database engine version 2 file format.
- **dbVersion30** – The default. Creates a database that uses the Jet database engine version 3 file format (compatible with version 3.5);

Let's see an example to make this clearer. When the user selects the New database item in our example DAO program, daocode, we will create a new database. First, we declare that database, db, as a form-wide variable:

```
Dim db As Database
```

Next, we add a Common Dialog control, CommonDialog1, to the program and show it to get the name of the database file the user wants to create:

```
Private Sub NewDatabase_Click()
    CommonDialog1.ShowSave
    If CommonDialog1.FileName < "" Then
        .....
```

Finally, we create the new database, passing the CreateDatabase method the name of the database file and indicating that we want to use the default collating order by passing the constant dbLangGeneral:

```
Private Sub NewDatabase_Click()
    CommonDialog1.ShowSave
    If CommonDialog1.FileName <> "" Then
        Set db =

            DBEngine.Workspaces(0).CreateDatabase_(CommonDialog1.
            FileName, dbLangGeneral).

    End If
End Sub
```

Now that we have created a database, the next step is to add table to that database and we'll take a look at that in the next topic.

### DAO: Creating a Table with a TableDef Object

For creating a table in a DAO database, you define it with a **TableDef** object. After you do so, you can append fields to the table and then you can append the new table definition to a database's **TableDefs** collection. Let's see an example.

After you create a new database with our DAO code example, the daocode project, you can create a new table using the New Table item in the File menu. That item opens the New Table dialog box for you. You can enter the name of the new table to create in the text boxes in the New Table dialog box and we can use that information to create a new TableDef object, td, which we declare as a form - wide variable:

```
Dim td As TableDef
```

We create a new TableDef for the Database object we created in the previous topic, db, using the name for the table the user has placed in Text1 in the New Table dialog box:

```
Sub CreateTable()
    Set td = database.CreateTableDef(TableForm.Text1.Text)
    ....
```

This code creates a new, empty TableDef object named td. Next we add fields to it.

## DAO: Adding Fields to a TableDef Object

How do you add fields to a DAO TableDef object? You can use that object's CreateField method to do that, passing that method the name of the new field and a constant indicating that field's type:

```
TableDef.CreateField ( FileName, FieldType)
```

Here are the constants specifying the possible field types:

- dbBigInt
- dbBinary
- dbBoolean
- dbByte
- dbChar
- dbCurrency
- dbDate
- dbDecimal
- dbDouble
- dbFloat
- dbGUID
- dbInteger
- dbLong
- dbLongBinary (OLE object)
- dbMemo
- dbNumeric
- dbSingle
- dbText
- dbnme
- dbnmeStamp
- dbVarBinary

Let's see an example to make this clearer. In the previous topic, we created a TableDef object named td for the daocode example project and now we can add two fields to that object, which we declare in an array named fields of type Field (which is defined in the DAO library):

```
Dim fields(2) As Field
```

The users have specified what names they want to give to those two new fields in the New Table dialog box's text boxes, so we create the new fields this way:

```
Sub CreateTable()
```

```
Set td = db.CreateTableDef(TableForm.Text1.Text)
```

NOTES

```

Set fields(0) = td.CreateField(TableForm.Text2.Text,
    dbText)
Set fields(1) = td.CreateField(TableForm.Text3.Text,
    dbText)

```

## NOTES

Now that the new fields are created, we can append them to the actual TableDef object td:

```

Sub CreateTable()
    Set td = db.CreateTableDef(TableForm.Text1.Text)

    Set fields(0) = td.CreateField(TableForm.Text2.Text,
        dbText)
    Set fields(1) = td.CreateField(TableForm.Text3.Text,
        dbText)

    td.fields.Append fields(0)
    td.fields.Append fields(1)

End Sub

```

That's it – we've defined two new fields, named them and appended them to a TableDef object.

### DAO: Creating a Record Set

After you've finished defining a database table with a DAO TableDef object, you can append that object to a Database object, which adds that table to that database. After you've installed the new table, you can use the OpenRecordset method to open a record set and start working with data:

```

Set recordset = Database.OpenRecordset (source, type,
    options, lockedits)

```

Here are the arguments for OpenRecordset:

- **source** – A string specifying the source of the records for the new Recordset object. The source can be a table name, a query name, or all SQL statement that returns records. (For table-type Recordset objects in Jet-type databases, the source can only be a table name.)
- **type** – Indicates the type of Recordset to open.
- **options** – Combination of constants that specify characteristics of the new Recordset.
- **lockedits** - Constant that determines the locking for Recordset.

Here are the possible settings for type:

- **dbOpenTable** – Opens a table-type Recordset object.
- **dbOpenDynamic** – Opens a dynamic – type Recordset object, which is like an ODBC dynamic cursor.
- **dbOpenDynaset** – Opens a dynaset – type Recordset object, which is like an ODBC keyset cursor.

- **dbOpenSnapshot** – Opens a snapshot – type Recordset object, which is like an ODBC static cursor.
- **dbOpenForwardOnly** – Opens a forward-only-type Recordset object (where you can only use MoveNext to move through the database).

Here are the possible settings for options:

- **dbAppendOnly** – Allows users to append new records to the Recordset but prevents them from editing or deleting existing records (Microsoft Jet dynaset-type Recordset only).
- **dbSQLPassThrough** – Passes an SQL statement to a Microsoft Jet- connected ODBC data source for processing (Jet snapshot-type Recordset only).
- **dbSeeChanges** – Generates a runtime error if one user is changing data that another user is editing (Jet dynaset-type Recordset only).
- **dbDenyWrite** – Prevents other users from modifying or adding records (Jet Recordset objects only).
- **dbDenyRead** – Prevents other users from reading data in a table (Jet table-type Recordset only).
- **dbForwardOnly** – Creates a forward-only Recordset (Jet snapshot-type Recordset only-), It is provided only for backward compatibility and you should use the dbOpenForwardOnly constant in the type argument instead of using this option.
- **dbReadOnly** – Prevents users from making changes to the Recordset (Microsoft Jet only). The dbReadOnly constant in the lockedits argument replaces this option, which is provided only for backward compatibility.
- **dbRunAsync** – Runs an asynchronous query (ODBCDirect workspaces only).
- **dbExecDirect** – Runs a query by slapping SQLPrepare and directly calling SQLExecDirect (ODBCDirect workspaces only).
- **dbInconsistent** - Allows inconsistent updates (Microsoft Jet dynaset- type and snapshot-type Recordset objects only).
- **dbConsistent** - Allows only consistent updates (Microsoft Jet dynaset- type and snapshot-type Recordset objects only).

Here are the possible settings for the lockedits argument:

- **dbReadOnly** – Prevents users from making changes to the Recordset (default for ODBCDirect workspaces).
- **dbPessimistic** – Uses pessimistic locking to determine how changes are made to the Recordset in a multiuser environment.
- **dbOptimistic** – Uses optimistic locking to determine how changes are made to the Recordset in a multiuser environment.
- **dbOptimisticValue** – Uses optimistic concurrency based on row values (ODBCDirect workspaces only).
- **dbOptimisticBatch** – Enables batch optimistic updating (ODBCDirect workspaces only).

NOTES

## NOTES

Let's see an example to make this clearer. In the previous few topics, we've developed a TableDef object, td, in our DAO code example, the daocode project. To append that object to the Database object we created, db, we use the Append method of the database object's TableDefs collection. After installing the table, we open it for use with the Database object's OpenRecordset method this way, creating a new DAO Recordset, which we name dbrecordset:

```
Sub CreateTable()
    Set td = db.CreateTableDef(TableForm.Text1.Text)

    Set fields(0) = td.CreateField(TableForm.Text2.Text,
        dbText)
    Set fields(1) = td.CreateField(TableForm.Text3.Text,
        dbText)
    td.fields.Append fields(0)
    td.fields.Append fields(1)
    Set dbindex = td.CreateIndex(TableForm.Text2.Text +
        "index")
    Set IxFlds = dbindex.CreateField(TableForm.Text2.Text)
    dbindex.fields.Append IxFlds
    td.Indexes.Append dbindex
    database.TableDefs.Append td
    Set dbrecordset = db.OpenRecordset(TableForm.Text1.Text,
        dbOpenTable)
End Sub
```

In this case, we're opening the new record set as a standard DAO table by passing the constant dbOpenTable. We also declare dbrecordset as a formwide variable:

```
Dim dbrecordset As Recordset
```

At this point in the daocode project, then, we've created a new database with a table in it that has two fields, using the names that the user supplied for the fields and the table itself. And we've opened that table as a record set, so we're ready to work with it. Besides creating a new database as we've done, however, the user may want to open an existing database, and we'll see how to do that in the next topic.

### DAO: Opening a Database

To open an existing DAO database, you use the DAO OpenDatabase method, passing it the name of the database to open and these arguments:

```
Set database = workspace.OpenDatabase (dbname, [optforis
    [, read-only [, connect]])
```

Here are the arguments for OpenDatabase:

- **dbname** – The name of an existing database file or the data source name (DSN) of an ODBC data source.
- **options** – Setting options to True opens the DAO database in exclusive mode; setting it to False (the default) opens the database in shared mode.

- **read-only**- True if you want to open the database with read-only access, or False (the default) if you want to open the database with read/write access.
- **connect**-Optional. A Variant (String subtype) that specifies various connection information, including passwords.

Let's see an example to make this clearer. In our DAO code example, the daocode project, the user can click the Open Database menu item to open a database. In the program, we get the name of the database the user wants to open with a Common Dialog control and open the database like this:

```
Private Sub OpenDatabase_Click()
    CommonDialog1.ShowOpen
    If CommonDialog1.FileName <> "" Then
        Set db =
        DBEngine.Workspaces(0).OpenDatabase
            (CommonDialog1.FileName)
```

Next, if you know the name of the table you want to open in the database, you can open that table by name immediately with the **OpenRecordset** method. However, because we let the user set the name of tables in the databases we create in the daocode project, we don't know the names of the tables in the database we've opened. Instead, we'll open the first user-defined table in this database.

When you open a DAO database, there are a number of system tables already in it, so to open the first user--defined table, we find the index of that table in the TableDefs collection by first skipping the system tables (which have the **dbSystemObject** flag set in their Attributes properties):

```
Private Sub OpenDatabase_Click()
    Dim tableIndex As Integer
    CommonDialog1.ShowOpen
    If CommonDialog1.FileName <> "" Then
        Set db = _ DBEngine.Workspaces(0).OpenDatabase
            (CommonDialog1.FileName)
        TableIndex = 0
        While (db.TableDefs(tableIndex).Attributes And
            dbSystemObject)
            tableIndex = tableIndex + 1
        Wend
```

We'll open the first table after the system tables. We will open a new record set for that table with the **OpenRecordset** method and fill the text boxes Text1 and Text2 in the program's main window with the fields of the first record in that table:

```
Private Sub Openatabase_Click()
    Dim tableindex As Integer
    CommonDialog1.ShowOpen
    If CommonDialog1.FileName <> "" Then
```

NOTES

## NOTES

```

Set db =
DBEngine.Workspaces(0).OpenDatabase(CommonDialog1.
    FileName)
tableindex = 0
While (db.TableDef~(tableindex).Attributes And
    dbSystemObject>
    tableindex =tableindex + 1
Wend
Set dbrecordset = db.OpenRecordset_
    (db.TableDefs (tableindex).Name, dbOpenTable)
Set td = db.TableDefs(tableIndex)
Text1.Text = dbrecordset.fields(0)
Text2.Text = dbrecordset.fields(1)
End if
End Sub

```

So now your database file is open.

### DAO: Adding a Record to a Record Set

To add a new record to a DAO record set, you use the `AddNew` method (this method takes no parameters). After you've updated the fields of the current record, you save that record to the database with the `Update` method. Here's an example using `AddNew`. When the user clicks the Add button in DAO code example, the `daocode` project, it executes the `AddNew` method on the program's record set and clear the two data field text boxes:

```

Private Sub Command1-Click()
    dbrecordset.AddNew
    Text1.Text = " "
    Text2.Text = ""
End Sub

```

Now users can enter data for the new record's fields and click the program's Update button. When they click the Update Database button, the new data is written to the database.

### DAO: Updating a Record in a Record Set

When the user changes the data in a record or adds a new record, we must update the database to record that change and use the record set `Update` method to do that:

```
recordset.Update ([type [, force]])
```

Here are the arguments in this function:

- **type**-Constant indicating the type of update, as specified in Settings (ODBCDirect workspaces only).
- **force**-Boolean value indicating whether or not to force the changes into the database, regardless of whether the data has been changed by another user (ODBCDirect workspaces only).

Let's see an example. When the user clicks the Update button in our DAO code example, the daocode project, we will update the database with the new data for the current record. We get the new data for the current record from the text boxes Text1 and Text2, where the user has entered that data and load the data into the record set's fields using the fields collection:

```
Private Sub Command3_Click()
    dbrecordset.fields(0) = Text1.Text
    dbrecordset.fields(1) = Text2.Text
    .....
End Sub
```

After loading the data into the current record's fields, we save that record to the database using the Update method:

```
Private Sub Command3_Click()
    dbrecordset.fields(0) = Text1.Text
    dbrecordset.fields(1) = Text2.Text
    dbrecordset.Update
End Sub
```

### DAO: Deleting a Record in a Record Set

To delete a record in a DAO record set, you use the Delete method and then you update the record set. For example, when the user clicks the Delete button in our DAO code example, the daocode project, we clear the two text boxes, Text1 and Text2, that display the data for the current record and delete that record:

```
Private Sub Command8_Click()
    Text1.Text = " "
    Text2.Text = " "
    dbrecordset.Delete
End Sub
```

### DAO: Sorting a Record Set

To sort a record set, you can install the index you want to sort within the record set's Index property. For example, we can sort the record set in our DAO code example, the daocode project, with the index we've created this way:

```
Sub Sort_Click( )
    Set dbindex = td.indexes(0)
    Dbrecordset.Index = dbindex.Name
    ....
```

After the record set is sorted, we display the first record in the two main text boxes, Text1 and Text2:

```
Sub Sort_Click( )
    Set dbindex = td.Indexes(0)
    dbrecordset.Index = dbindex.Name
    Text1.Text = dbrecordset.fields(0)
```

NOTES





```
Text2.Text = dbrecordset.fields(1)
```

```
End Sub
```

## DAO: Searching a Record Set

### NOTES

You can search a record set with an index; we just set its Index property to the index we want to search and then set its Seek property to the string we want to search for. Let's see an example. When the user selects the Search menu item in our DAO code example, the daocode project, we install the index based on the first field in the record set and show the dialog box named Search.....:

```
Private Sub Search_Click()
    Set dbindex = td.Indexes(0)
    dbrecordset.Index = dbindex.Name
    SearchForm.Show
End Sub
```

After the user dismisses the Search. .dialog box, we retrieve the text to search for from that dialog box's text box and place that text in the record set's Seek property, along with the command "=", which indicates we want to find exact matches to the search text:

```
Sub SearchTable()
    dbrecordset.Seek "=", SearchForm.Text1.Text
```

Besides =, you can also search using <, <=, >= and >. When the search is complete, we display the found record in the daocode project's main text boxes, Text1 and Text2:

```
Sub SearchTable()
    dbrecordset.Seek "-" SearchForm.Text1.Text
    text1.Text = dbrecordset.fields(0)
    text2.Text = dbrecordset.fields(1)
End sub
```

## DAO: Executing SQL

You can execute an SQL statement when you create a DAO record set using the OpenRecordset method by placing that SQL statement in the source argument:

```
Set recordset = Database.OpenRecordset (source, type,
options, lockedjts)
```

Here are the arguments for OpenRecordset:

- **source**-A string specifying the source of the records for the new Recordset. The source can be a table name, a query name, or an SQL statement that returns records. (For table-type Recordset objects in Jet-type databases, the source can only be a table name.)
- **type**-Indicates the type of Recordset to open.
- **options**-Combination of constants that specify characteristics of the new Recordset.

- **lockedits**-Constant that determines the locking for Recordset.

Here are the possible settings for type:

- **dbOpenTable**-Opens a table-type Recordset object.
- **dbOpenDynamic**-Opens a dynamic-type Recordset object, which is like an ODBC dynamic cursor.
- **dbOpenDynaset**-Opens a dynaset-type Recordset object, which is like an ODBC keyset cursor.
- **dbOpenSnapshot**-Opens a snapshot-type Recordset object, which is like an ODBC static cursor.
- **dbOpenForwardOnly**-Opens a forward-only-type Recordset object.

Here are the possible settings for options:

- **dbAppendOnly**-Allows users to append new records to the Recordset but prevents them from editing or deleting existing records (Microsoft Jet dynaset-type Recordset only).
- **dbSQLPassThrough**-Passes an SQL statement to a Microsoft Jet- connected ODBC data source for processing (Microsoft Jet snapshot-type Recordset only).
- **dbSeeChanges**-Generates a runtime error if one user is changing data that another user is editing (Microsoft Jet dynaset-type Recordset only).
- **dbDenyWrite**-Prevents other users from modifying or adding records (Microsoft Jet Recordset objects only).
- **dbDenyRead**-Prevents other users from reading data in a table (Microsoft Jet table-type Recordset only).
- **dbForwardOnly**-Creates a forward-only Recordset (Microsoft Jet snapshot-type Recordset only). It is provided only for backward compatibility and you should use the **dbOpenForwardOnly** constant in the type argument instead of using this option.
- **dbReadOnly**-Prevents users from making changes to the Recordset (Microsoft Jet only). The **dbReadOnly** constant in the **lockedits** argument replaces this option, which is provided only for backward compatibility.
- **dbRunAsync**-Runs an asynchronous query (ODBCDirect workspaces only).
- **dbExecDirect**-Runs a query by skipping SQLPrepare and directly calling SQLExecDirect (ODBCDirect workspaces only).
- **dbInconsistent**-Allows inconsistent updates (Microsoft Jet dynaset- type and snapshot-type Recordset objects only).
- **dbConsistent**-Allows only consistent updates (Microsoft Jet dynaset- type and snapshot-type Recordset objects only).

Here are the possible settings for the **lockedits** argument:

- **dbReadOnly**-Prevents users from making changes to the Recordset (default for ODBCDirectworkspaces).

NOTES

## NOTES

- **dbPessimistic**-Uses pessimistic locking to determine how changes are made to the Recordset in a multiuser environment.
- **dbOptimistic**-Uses optimistic locking to determine how changes are made to the Recordset in a multiuser environment.
- **dbOptimisticValue**-Uses optimistic concurrency based on row values (ODBCDirect workspaces only).
- **dbOptimisticBatch**-Enables batch optimistic updating (ODBCDirect workspaces only).

### Connecting with Database using RDO

Now we will build a fully functional RDO project – the rdocode project. This program is designed to open the ODBC data source and let the user move around in it record by record. Using the buttons in the rdocode project, you can move through the database and we'll see how to write the code for the rdocode project in the following few topics.

#### RDO: Opening a Connection

To open an RDO connection to a database, you can use the RDOOpenConnection method. OpenConnection is a method of the rdoEnvironment object and you'll find a collection of those objects in the rdoEngine object's rdoEnvironments collection.

To add the RDO objects to a program, select the Project|References menu item in Visual Basic, select the Microsoft Remote Data Object entry in the References dialog box and click on OK. Now we're free to use rdoEnvironment methods like OpenConnection:

```
workspace.OpenConnection(datasource, [prompt, {read-only,
                                     [connect, _ roptfons]]])
```

Here are the arguments to OpenConnection:

- **datasource** - The name of the data source.
- **prompt** - ODBC prompting characteristic: rdDriverPrompt asks the user for a driver/database, rdDriverNoPrompt uses specified driver/ database, rdDriverComplete specifies the connection string itself and rdDriverCompleteRequired is the same as rdDriverComplete, with the additional requirement that the driver should disable the controls for information not needed for the connection.
- **read-only** – True if you want to open the data source as read-only.
- **connect** – The connect string.
- **Options** – set to rdAsyncEnable if you want to execute commands asynchronously (that is; without waiting for the command to be completed).

Let's see an example.

In our RDO code example, the rdocodeproject, we create an rdoEnvironment object named re this way when the form loads:

```
Dim re As Object
Private Sub Form_Load()
```

```
Set re = rdoEngine.rdoEnvironments(0)
```

```
End Sub
```

Now we open a connection named db to the ODBC source (we set up this ODBC source in the previous chapter) this way:

```
Dim re As Object
Dim db As rdoConnection
Private Sub Form_Load()
    Set re = rdoEngine.rdoEnvironments(0)
    Set re = re.Openconnection("db")
End Sub
```

That's it—now we have a connection to our ODBC data source in the rdoConnection object named db.

## RDO: Creating a Result Set

After opening an ODBC data source and creating an rdoConnection Object, we can create an RDO result set to start working with the records in that data source. To create a result set, we can use the rdoConnection method OpenResultset:

```
Set reserset = rdoConnection.OpenResultset (name, [type,
    [locktype, _ [options]])
```

Here are the arguments for OpenResultset:

- **name** – Source for the result set; can be an rdoTable object, an rdoQuery object, or an SQL statement.
- **type**—Specifies the result set type (see the following list).
- **locktype**—Can be one of these values: rdConcurReadonly (read-only), rdConcurLock (pessimistic concurrency), rdConcurRowVer (optimistic row-based concurrency), rdConcurValues (optimistic value-based concurrency), or rdConcurBatch (optimistic concurrency using batch updates).
- **options**—Set to rdAsyncEnable if you want to execute commands asynchronously (that is, without waiting for the command to be completed).

Here are the possible values for the type argument:

- **dbOpenKeyset**—Opens a dynaset-type rdoResultset object, which is like an ODBC keys at cursor.
- **dbOpenDynamic**—Opens a dynamic-type rdoResultset object, which lets the application see changes made by other users.
- **dbOpenStatic**—Opens a static-type rdoResultset object.
- **dbOpenForwardOnly**—Opens a forward-only-type rdoResultset object, where you can only use MoveNext to move.

Let's see an example. Here, we'll create an SQL-based result set in our RDO code example, the rdocode project, when the form loads, using the rdoConnection object we've created—db. In this case, we'll set up an SQL statement, SQLSel, to place all the fields from the data source's table named students in the result set:

NOTES

## NOTES

```

Dim re As Object
Dim db As rdoConnection
Dim SQLSel As String
Private Sub Form_Load()
    SQLSel = "Select  from students"
    Set re = rdoEngine.rdoEnvironments(0)
    Set db = re.OpenConnection("db")

```

Now we use OpenResultset to create an rdoResultset object, resultset:

```

Dim re As Object
Dim db As rdoConnection
Dim resultset As rdoResultset
Dim SQLSel As String
Private Sub Form_Load()
    SQLSel = "Select * from students"
    Set re = rdoEngine.rdoEnvironments(0)
    Set db = re.OpenConnection("db")
    Set resultset = db.OpenResultset(SQLSel, rdOpenKeyset)

```

Now that we've opened a result set, we can use rdoResultset methods like MoveFirst to move to the first record and display the data in that record's Name and Grade fields with the rdocode project's text boxes, Text1 and Text2:

```

Dim re As Object
Dim db As rdoConnection
Dim resultset As rdoResultset
Dim SQLSel As String
Private Sub Form_Load()
    SQLSel = "Select * from students"
    Set re = rdoEngine.rdoEnvironments(0)
    Set db = re.OpenConnection("db")
    Set resultset = db.OpenResultset(SQLSel, rdOpenKeyset)
    resultset.MoveFirst
    Text1.Text = resultset("Name")
    Text2.Text = resultset("Grade")
End Sub

```

And that's it—we've opened an RDO result set and displayed some of the data in that result set.

### RDO: Moving to the First Record in a Result Set

To move to the first record in an RDO result set, you can use the rdoResultset method MoveFirst. Let's see an example. In this case, we'll move to the first record in the result set named resultset that we've opened in our RDO code example, the rdocode project, when the user clicks the appropriate button:

```

private sub cmdFirst_Click()
    On Error GoTo ErrLabel
    resultset.MoveFirst
    ...
Exit Sub
ErrLabel
    MsgBox Err.Description
End Sub

```

NOTES

After moving to the new record, we display the data in that record's fields in the program's text boxes, Text1 and Text2:

```

Private Sub cmdFirst_Click()
    On Error GoTo ErrLabel
    resultset.MoveFirst
    text1.Text = resultset("Name")
    text2.Text = result ("Grade")
Exit Sub
ErrLabel:
    MsgBox Err.Description
End Sub

```

### RDO: Moving to the Last Record in a Result Set

To move to the last record in an RDO result set, you can use the `rdoResultset` method `MoveLast`. Let's see an example. In this case, we'll move to the last record in the result set named `resultset` that we've opened in our RDO code example, the `rdocode` project, when the user clicks the appropriate button:

```

Private Sub cmdLast_Click1()
    On Error GoTo ErrLabel
    resultset.MoveLast
    ...
Exit Sub
ErrLabel :
    MsgBox Err.Description
End Sub

```

After moving to the new record, we display the data in that record's fields in the program's text boxes, Text1 and Text2:

```

Private Sub cmdLast_Click()
    On Error GoTo ErrLabel
    resultset.MoveLast
    Text1.Text = resultset("Name")
    Text2.Text = result ("Grade")
Exit Sub
ErrLabel:

```

```
MsgBox Err.Description
```

```
End Sub
```

### RDO: Moving to the Next Record in a Result Set

NOTES

To move to the next record in an RDO result set, you can use the `rdoResultset` Result Set method `MoveNext`. Let's see an example. In this case, we'll move to the next, record in the result set named `resultset` that we've opened in our RDO code example, the `rdocode` project, when the user clicks the appropriate button. We check to make sure we're not trying to move past the end of the record set with the `EOF` property and if so, we make sure to move to the last record instead:

```
Private Sub cmdNext_Click()
    On Error GoTo ErrLabel
    If Not resultset.EOF Then resultset.MoveNext
    If resultset.EOF And resultset.RowCount > 0 Then
        resultset.MoveLast
    End If

Exit Sub
ErrLabel :
    MsgBox Err.Description
End Sub
```

After moving to the new record, we display the data in that record's fields in the program's text boxes, `Text1` and `Text2`:

```
Private Sub cmdNext_Click()
    On Error GoTo ErrLabel
    If Not resultset.EOF Then resultset.MoveNext
    If resultset.EOF And resultset.RowCount > 0 Then
        resultset.MoveLast
    End If
    Text1.Text = resultset("Name")
    Text2.Text = resultset("Grade")
Exit Sub
ErrLabel :
    MsgBox Err.Description
End Sub
```

### RDO: Moving to the Previous Record in a Result Set

To move to the previous record in an RDO result set, you can use the `Result Set` `rdoResultset` method `MovePrevious`. Let's see an example. In this case, we'll move to the previous record in the result set named `resultset` that we've opened in our RDO code example, the `rdocode` project, when the user clicks the appropriate button. We check to make sure we're not trying to move past the beginning of the record set with the `BOF` property and if so, we make sure to move to the first record instead:

```
Private Sub cmdPrevious_Click()
```

```

On Error GoTo ErrLabel
If Not resultset.BOF Then resultset.MovePrevious
If resultset.BOF And resultset.RowCount > 0 Then
    resultset.MoveFirst
End If
Text1.Text = resultset("Name")
Text2.text = result ("Grade")
Exit Sub

ErrLabel :
    MsgBox Err.Description
End Sub

```

NOTES

After moving to the new record, we display the data in that record's fields in the program's text boxes, Text1 and Text2:

```

Private Sub cmdPrevious_Click()
    On Error GoTo ErrLabel
    If Not resultset.BOF Then resultset.MovePrevious
    If resultset.BOF And resultset.RowCount > 0 Then
        resultset.MoveFirst
    End If
    Text1.Text = resultset("Name")
    Text2.text = result ("Grade")
    Exit Sub

ErrLabel:
    MsgBox Err.Description
End Sub

```

## RDO: Executing SQL

You can execute SQL statements with RDO objects when you open a result set, as we saw in "RDO: Creating A Result Set" in this chapter. You can also execute an SQL statement with the rdoConnection object's. Execute statements like this:

```

SQLSel = "Select * from students"
rdoConnection.Execute SQLSel

```

## Connecting with Database using ADO

To illustrate ADO data handling in code, we'll build an ADO project – the adocode project. You can also move through the database using the arrow buttons. To edit a record, just type the new value(s) into the text box(es) and click the Update button. To add a record, use the Add button, type the new value(s) into the text box(es) and click the Update button. That's all there is to it – your changes will be reflected in the original database.

## ADO: Opening a Connection

The first step in editing an ADO database is to open that database, which is called a data source in ADO terminology, by setting up a Connection object. To use that and

other ADO objects in code, you use the Project I References item, select the Microsoft ActiveX Data Objects Library item and click on OK, adding the ADO Object Library to your program.

Now we're free to create a new ADO Connection object with the Connection object's Open method:

NOTES

```
connection.Open ConnectionString [,Use/:ID [ Password [,
    OpenOptions]]]
```

Here are the arguments for this method:

- **ConnectionString**-String containing connection information.
- **UserID**-String containing a username to use when establishing the connection.
- **Password**-String containing a password to use when establishing the connection.
- **OpenOptions**-If set to adConnectAsync, the connection will be opened asynchronously.

Let's see an example. When we start our ADO code example, the adocode example, we'll establish a connection, db, to the database we built in the previous chapter, db.mdb:

```
Private Sub Form_Load()
    Dim db As connection
    Set db = NewConnection
    db.Open "PROVIDER=Microsoft.Jet.OLEDB.3.51:Data_Source
        = C:\vb\adocode\db.mdb;"
    ...
End Sub
```

And that's it – now we have a connection to the data source. To actually work with the data in that data source, we'll create an ADO record set in the next topic.

### ADO: Creating a Record Set from a Connection

Now that you've created an ADO connection, you can open a record set from Connection that connection using the Recordset object's Open method:

```
recordset.Open Source. [ActiveConnection, [Type. [Lock
    Type, [Options]]]
```

Here are the arguments for this method:

- **Source**-A valid Command object variable name, an SQL statement, a table name, a stored procedure call or the file name of a Recordset.
- **ActiveConnection**-A valid Connection object variable name or a string containing ConnectionString parameters.
- **Type**-Sets the Recordset type (see the following list).
- **LockType** - A value that determines what type of locking (concurrency) the provider should use when opening the Recordset (see the following list).
- **Options**-A Long value that indicates how the provider should evaluate the Source argument if it represents something other than a Command object or

that the Recordset should be restored from a file where it was previously saved (see the following list).

Here are the possible values for the Type argument:

- **dbOpenKeyset**-Opens a dynaset-type Recordset object, which is like an ODBC keyset cursor.
- **dbOpenDynamic**-Opens a dynamic-type Recordset object, which lets the application see changes made by other users.
- **dbOpenStatic**-Opens a static-type Recordset object.
- **dbOpenForwardOnly**-Opens a forward-only-type Recordset object, where you can only use MoveNext to move.

Here are the possible values for the Lock Type argument:

- **adLockReadOnly**- The default; read-only.
- **adLockPessimistic**-Pessimistic locking, record by record.
- **adLockOptimistic**-Optimistic locking, record by record.
- **adLockBatchOptimistic**-Optimistic batch updates.

Here are the possible values for the Options argument:

- **adCmdText**-Provider should evaluate Source as a definition of a command.
- **adCmdTable**-ADO should generate an SQL query to return all rows from the table named in Source.
- **adCmdTableDirect**-Provider should return all rows from the table named in Source.
- **adCmdStoredProc**-Provider should evaluate Source as a stored procedure.
- **adCmdUnknown**-Type of command in the Source argument is not known.
- **adCommandFile**-Record set should be restored from the file named in Source.
- **adExecuteAsync**-Source should be executed asynchronously.
- **adFetchAsync**-After the initial quantity specified in the CacheSize property is fetched, any remaining rows should be fetched asynchronously.

Let's see an example. In our ADO code example, the adocode example, we create a record set, adoRecordset, by first declaring it as a form-wide variable:

```
Dim adoRecordset As Recordset
```

Next, we select all the records in the students table this way when the form loads, using the Open method:

```
Private Sub Form_Load()
```

```
    Dim db As Connection
```

```
    Set db = New Connection
```

```
    db.Open "PROVIDER=Microsoft.Jet.OLEDB.3.51:Data_
    Source-C:\vbbb\adocode\db.mdb:"
```

```
    Set adoRecordset = New Recordset
```

NOTES

```
adoRecordset.Open "selectGrade. Name from students",_db,
    adOpenStatic, adLockOptimistic
```

```
....
```

```
End Sub
```

## NOTES

Now that we've opened our result set, we can bind that result set to various controls, like text boxes, as we'll do in the next topic.

**ADO: Binding Controls to Record Sets**

To bind a control to an ADO Recordset object, you just set that control's DataSource property to that object, and then set whatever other data properties that control needs to have set.

Let's see an example. In our ADO code example, the adocode example, we create a record set, adoRecordset and open the dp.mdb database we created in the last chapter in it. We can bind the fields in that database to the text boxes Text1 and Text2 this way when the adocode main form loads:

```
Private Sub Form_Load()
    Dim db As Connection
    Set db = new Connection
    db.Open "PROVIDER=Microsoft.Jet.OLEDB.3.51;Data_Sourc
        =C:\vbbb\adocode\db.mdb;"
    Set adoRecordset = New Recordset
    adoRecordset.Open "select Grade, Name from students",
        db. adOpenStatic, adlockOptimistic
    Set Text1.DataSource = adoRecordset
    Text1.DataField = "Name"
    SetText2.DataSource = adoRecordset
    Text2.DataField = "Grade"
```

```
....
```

```
End Sub
```

That's all it takes—now we've bound two text boxes to an ADO record set.

**ADO: Adding a Record to a Record Set**

To add a new record to an ADO record set, you use the AddNew method. After you've updated the fields of the current record, you save that record to the database with the Update method. Here's how you use AddNew:

```
recordset.AddNew [Fields [, Values]]
```

Here are the arguments for this method:

- **Fields**—A single name or an array of names or ordinal positions of the fields in the new record.
- **Values**—A single value or an array of values for the fields in the new record. If Fields is an array, Values must also be an array with the same number of members. The order of field names must match the order of field values in each array.

Let's see an example. Here, we'll add a new record to the record set `adoRecordset` in our ADO code example, the `adocode` example, when the user clicks the appropriate button:

```
Private Sub cmdAdd_Click()
    On Error GoTo ErrLabel
    adoRecordset.AddNew
    Text1.Text = " "
    Text2.Text = " "
    Exit Sub
ErrLabel:
    MsgBox Err.Description
End Sub
```

Note that we also clear the two text boxes that display the field Data, `Text1` and `Text2`, so users can enter the data they want in the new record. When done, they press the Update button to update the data source.

### ADO: Refreshing the Record Set

Sometimes you want to refresh the data in a record set – you might be dealing with multiply-connected databases, for instance, where other users are making changes as well – and you can use the ADO Refresh method for that. Let's see an example. Here, we'll refresh the record set `adoRecordset` in our ADO code example, the `adocode` example, when the user clicks the appropriate button:

```
Private Sub cmdRefresh_Click()
    On Error GoTo ErrLabel
    adoRecordset.Requery
    Exit Sub
ErrLabel
    MsgBox Err.Description
End Sub
```

And that's all it takes to refresh the record set.

### ADO: Updating a Record in a Record Set

After changing the data in a record's fields or adding a new record, you update the data source to record the change, using the Update method:

```
recordset.Update Fields, Values
```

Here are the arguments for this method:

- **Fields**-A single name or an array of names or ordinal positions of the fields in the new record.
- **Values**-A single value or an array of values for the fields in the new record. If `Fields` is an array, `Values` must also be an array with the same number of members. The order of field names must match the order of field values in each array.

Let's see an example. When users want to update records in our ADO code example,

NOTES

## NOTES

the adocode example, they click the appropriate button and we'll update the data source this way:

```
Private Sub cmdUpdate_Click()
    On Error Goto ErrLabel
    adoRecordset.Update
Exit Sub
ErrLabel :
    MsgBox Err.Description
End Sub
```

That's all we need – now we're ready to update records in an ADO record set.

### ADO: Moving to the First Record in a Record Set

To move to the first record in an ADO record set, you use the Recordset object's MoveFirst method (this method takes no parameters). Let's see an example. When the user clicks the appropriate button in our ADO code example, the adocode example, we'll move to the first record in our record set, adoRecordset:

```
Private Sub cmdFirst_Click()
    On Error GoToErrLabel
    adoRecordset.MoveFirst
Exit Sub
ErrLabel:
    MsgBox Err.Description
End Sub
```

And that's the code we need to move to the first record.

### ADO: Moving to the Last Record in a Record Set

In A Record Set To move to the last record in an ADO record set, you use the Recordset object's MoveLast method (this method takes no parameters).

Let's see an example. When the user clicks the appropriate button in our ADO code example, the adocode example, we'll move to the last record in our record set, adoRecordset:

```
Private Sub cmdLast_Click()
    On Error GoToErrLabel
    adoRecordset.Movelast
Exit Sub
ErrLabel:
    MsgBox Err.Description
End Sub
```

And that's all the code we need to move to the last record.

### ADO: Moving to the Next Record in a Record Set

To move to the next record in an ADO record set, you use the Recordset object's MoveNext method (this method takes no parameters). Let's see an example.

When the user clicks the appropriate button in our ADO code example, the adocode example, we'll move to the next record in our record set, adoRecordset. Note that we make sure we don't move-past the end of the record set by checking the record set's EOF property:

```
Private Sub cmdNext_Click()
    On Error GoTo ErrLabel
    If Not adoRecordset.EOF Then
        adoRecordset.MoveNext
    End If
    If rdoRecordset.EOF and adoRecordset.Recordcount > 0 Then
        adoRecordset.MoveLast
    End if
    Exit Sub
ErrLabel:
    MsgBoxErr.Description
End Sub
```

NOTES

### ADO: Moving to the Previous Record in a Record Set

To move to the next record in an ADO record set, you use the recordset objects MovePrevious method (this method takes no parameters). Let's see an example. When the user clicks the appropriate button in our ADO code example, the adocode example, we'll move to the previous record in our record set, adoRecordset.

Note that we make sure we don't move past the end of the record set by checking the record set's BOF property:

```
Private Sub cmdPrevious_Click()
    On Error GoTo ErrLabel
    If Not adoRecordset.BOF Then adoRecordset.MovePrevious
    If adoRecordset.BOF And adoRecordset.RecordCount > 0
        Then
        adoRecordset.MoveFirst
    End If
    Exit Sub
ErrLabel:
    MsgBox Err.Description
End Sub
```

### ADO: Deleting a Record in a Record Set

To delete a record in an ADO record set, you use the Delete method:

```
record.set.Delete AffectRecords
```

Here, AffectRecords is a value that determines how many records the Delete method will affect. It can be one of the following constants:

- **adAffectCurrent**- The default; deletes only the current record.

- **adAffectGroup**-Deletes the records that satisfy the current Filter property setting.

Let's see an example. Here, we delete a record in our ADO code example, the adocode example, when the user presses the appropriate button:

NOTES

```
Private Sub cmdDelete_Click()
    On Error GoTo ErrLabel
    adoRecordset.Delete
    ...
```

In addition, we move to the next record this way

```
Private Sub cmdDelete_Click()
    On Error GoTo ErrLabel
    adoRecordset.Delete
    adoRecordset.MoveNext
    If adoRecordset.EOF Then
        adoRecordset.MoveLast
    End If
    Exit Sub
ErrLabel:
    MsgBox Err.Description
End Sub
```

And that's it—now we've deleted a record.

### ADO: Executing SQL in a Record Set

You can execute an SQL statement when you open a record set using the Open method by passing that statement as the Source argument:

```
recordset.Open [Source, [ActiveConnection, Type, LockType,
    Options]]]
```

Here are the arguments for this method:

- **Source** – A valid Command object variable name, an SQL statement, a table name, a stored procedure call, or the file name of a Recordset.
- **ActiveConnection**-A valid Connection object variable name or a sting containing ConnectionString parameters.
- **Type**-Sets the Recordset type (see the following list)
- **LockType**-A value that determines what type of locking ( concurrency) the provider should use when opening the Recordset object (see the following list).
- **Options**-A Long value that indicates how the provider should evaluate the Source argument if it represents something other than a Command object or that the Recordset object should be restored from a file where it was previously saved (see the following list).

Here are the possible values for the Type argument:

- **dbOpenKeyset**-Opens a dynaset-type Recordset object, which is like an ODBC keyset cursor.
- **dbOpenDynamic**-Opens a dynamic-type Recordset object, which lets the application see changes made by other users.
- **dbOpenStatic**-Opens a static-type Recordset object.
- **dbOpenForwardOnly**-Opens a forward-only-type Recordset object, where you can only use MoveNext to move.

Here are the possible-values for the Lock Type argument:

- **adLockReadOnly**- The default; read-only.
- **adLockPessimistic**-Pessimistic locking, record by record.
- **adLockOptimistic**-Optimistic locking, record by record.
- **adLockBatchOptimistic**-Optimistic batch updates.

Here are the possible values for tile Options argument:

- **adCmdText**-Provider should evaluate Source as a definition of a command.
- **adCmdTable**-ADO should generate an SQL query to return all rows from tile table named in Source.
- **adCmdTableDirect**-Provider should return all rows from the table named in Source.
- **adCmdStoredProc**-Provider should evaluate Source as a stored procedure.
- **adCmdUnknown**- Type of command in the Source argument is not known.
- **adCommandFile**-Record set should be restored from tile file named in Source.
- **adExecuteAsync**-Source should be executed asynchronously.
- **adFetchAsync**-After the initial quantity specified in tile CacheSize property is fetched, any remaining rows should be fetched asynchronously.

Here's an example where we open a record set with the SQL statement "select \* from students":

```
adoRecordset.Open "select * from students", db.
    adOpenStatic. adLockOptimistic
```

---

## OBJECT ORIENTED PROGRAMMING

---

Today without OOP it is difficult to divide the typical programming task into different pieces. OOPS can be used for reusability of objects. Everything here is associated with classes and objects. I will describe you one by one. First of all OOP stands for Object-oriented programming. You do OOP based programming using objects. For example you can add many textbox control onto form, each textbox control have same properties and methods but code is written once by VB6 developers. This is a kind of OOP example. So like this you can save money and time both.

Technically OOP consist of Inheritance, polymorphism and encapsulation.

NOTES

## Object

Object can be referred as class. In this class you can find properties and methods. You create objects according to business rule. In object you define Public, Private and Friend etc keywords which create the security model. Using public property or method object is accessible through out the application. If properties and methods are private then they are accessible only with in the object. In general object is made up of properties, methods and events.

### Characteristics of Object

As I have told you above that technically OOP consist of Inheritance, polymorphism and encapsulation but VB6 does not support polymorphism so we can say that VB6 is not a language which supports full OOP characteristics.

### Object Properties

Property provides the description of particular object. Properties of a class can make the default interface. In general properties represent the data about an object.

To define the properties for a class you can add public variables to the class module.

For example:

```
Public x as string
Public y as integer
```

To create private data for a class you can create a Private variable but this is accessible only from code within the class module.

For example:

```
Private x as double
Private y as string
```

### Object Methods

In general behavior of method is they represent the actions which an object takes. You can say this that methods are the behavior of object.

You declare the methods of class as public Sub or Function procedures.

By default these methods are treated as public.

For example:

```
Public Function mysum(x As integer, y As integer) As
    integer
    Statements
    Statements
    Statements...

End Function
```

### Friend Properties and Methods

Friend procedures are not members of a class interface but these are visible throughout the entire project.

## Inheritance

To inherit the properties and methods of its parent class is known as inheritance.

In Visual Basic 6 you have to instantiate the new object and to create a new object you use New keyword.

For example:

```
Dim x as myclass
Set x = New myclass
```

## Encapsulation

Encapsulation hides the data and methods from the programmer.

In Visual Basic 6 you can use Private keyword for this.

## Polymorphism

Meaning of polymorphism is that caller does not care that which class an object belongs to before calling the property or method. You can say that many classes can provide the same property or method.

For example:

You have two classes x and y and both have mysum method. Now you can use mysum method without knowing this that object is x or y.

## Property Procedures

VB6 provides the three types of property procedures:

Procedure	Description
Property Get	It returns the value of a property.
Property Let	It sets the value of a property.
Property Set	It sets the value of an object.

Examples:

```
Public Property Get thevar() As Variant
End Property
Public Property Let thevar(ByVal vNewValue As Variant)
End Property
```

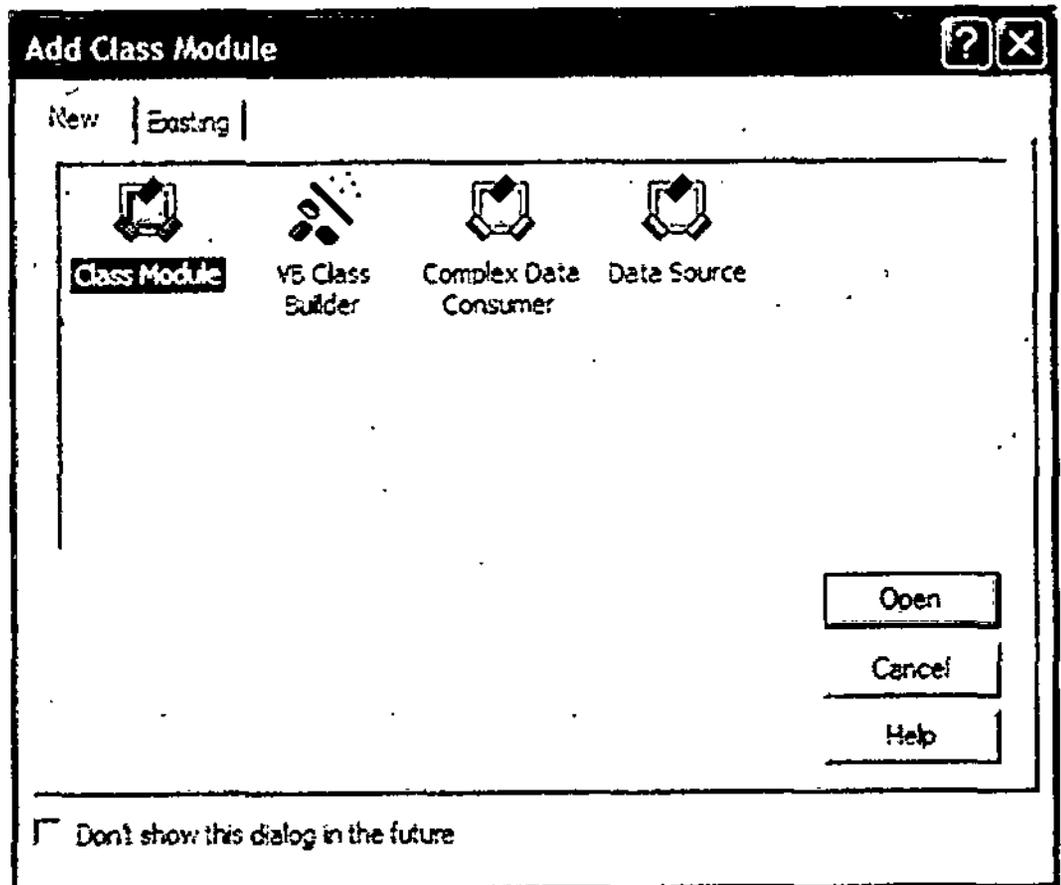
Practice Yourself

1. Start Visual Basic 6.
2. Select Standard Exe from New tab.
3. Click Ok.
4. On menu bar click File.
5. Click Save Project.
6. Save File As dialog box appears.

NOTES

NOTES

7. Select the desired drive.
8. In File name field keep default name form1.frm.
9. Click Save.
10. Save Project As dialog box appears.
11. In File name field keep default name project1.vbp.
12. Click Save.
13. Source Code Control dialog box appears.
14. Click No.
15. In Project window right-click the Project1
16. Position your mouse over Add
17. Click Class Module
18. Now Add Class Module dialog box appears.



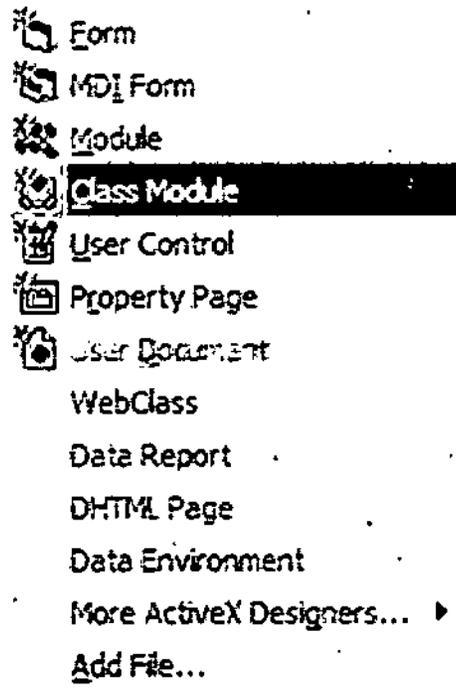
19. Click Open
20. In Project window double-click the Class1 to open its code window
21. Type the following code:  

```
Private x As String  
Public Property Let takename(ByVal vNewValue As Variant)
```

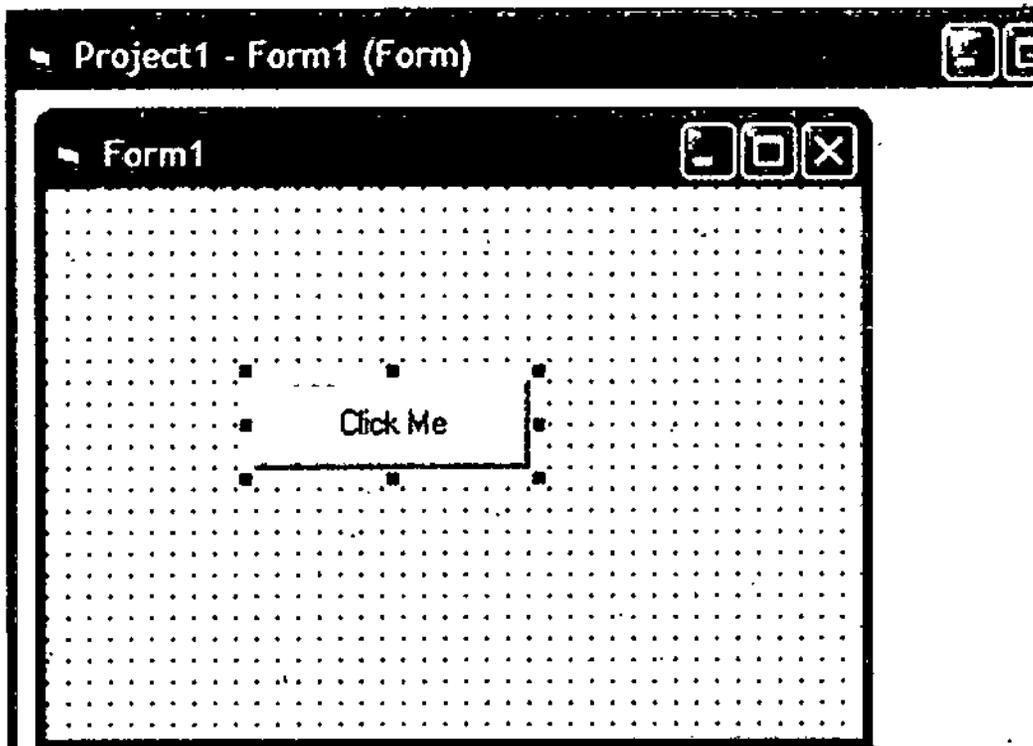
```
x = vNewValue
End Property
Public Sub showname()
MsgBox ("Hello: " & x)
End Sub
```

22. Now in Project window double-click the Form1 to bring it in design view.
23. Add the button control.
24. Change its caption property to "Click Me"
25. At present your form have following appearance:
26. Now double-click the button control to open the code window.
27. Type the following code:

```
Private Sub Command1_Click()
```



NOTES

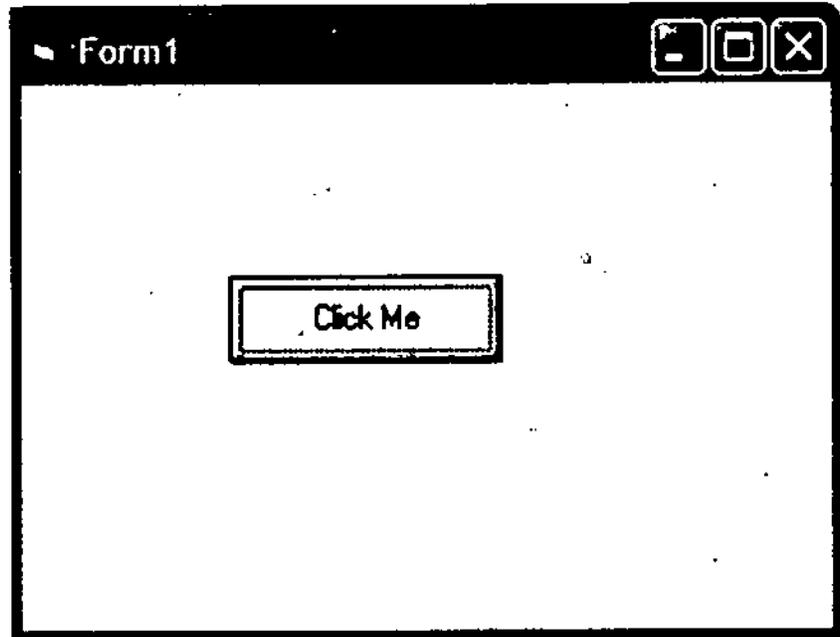


```
Dim y As Class1
Set y = New Class1
y.takenname = "gurmeet"
y.showname
End Sub
```

28. Now press the F5 key to run the program.

29. You will get the window shown next.

NOTES



30. Press the button.

31. You will get the output shown here.

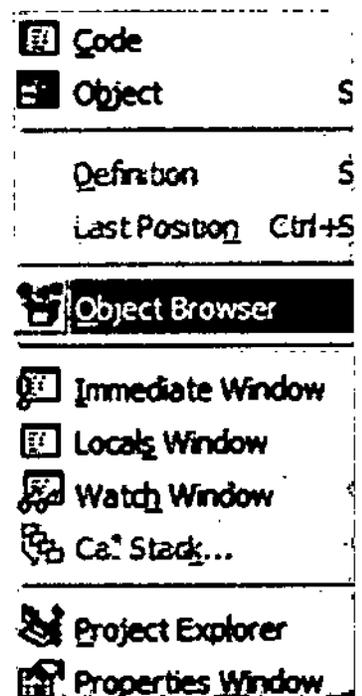
### Object Browser

Role of Object Browser is to displays the available properties, methods, classes, events, and constants from object libraries and procedures.

You can also use the Object Browser to find the objects you created.

To start the Object Browser perform the following steps:

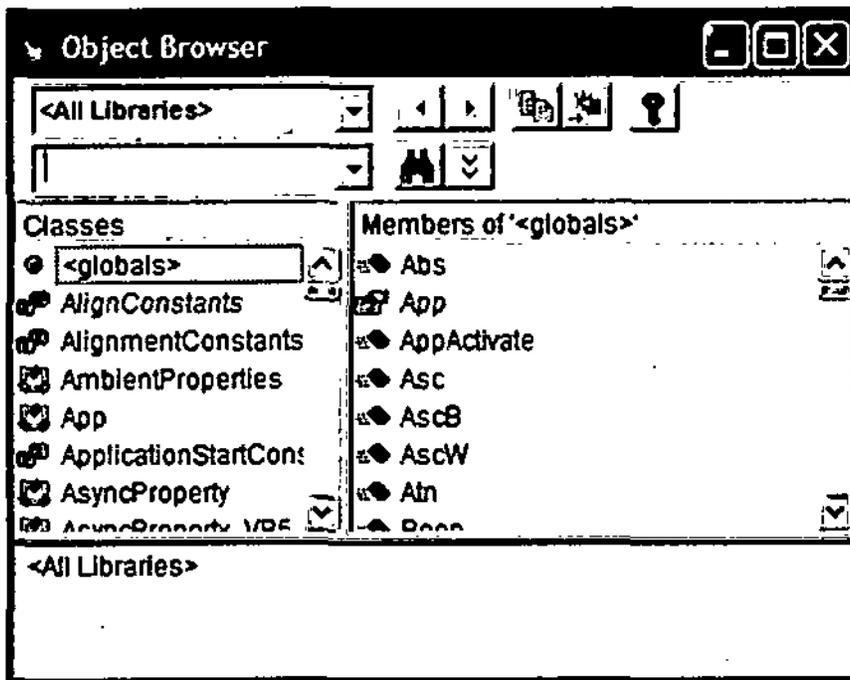
1. On menu bar click View
2. Click Object Browser
3. You will get the Object Browser window.



## CREATING OBJECTS

When you want to create an object from a code component, you first add a reference to that code component using the Visual Basic Project|References menu item (note that the code component must be registered with Windows, which most applications will do automatically when installed). Next, you create that object in code, and there are three ways to create objects from code components in Visual Basic.

- Declaring the variable using the New keyword (in statements like Dim, Public, or Private), which means Visual Basic automatically assigns a new object



## NOTES

reference the first time you use the variable (for example, when you refer to one of its methods or properties). This technique only works with code components that supply type libraries (as the code components created with Visual Basic do). These type libraries specify what's in the code component.

- Assigning a reference to a new object in a Set statement by using the New keyword or CreateObject function.
- Assigning a reference to a new or existing object in a Set statement by using the GetObject function.

Let's see some examples. Here, we will assume we have already added a reference to the code component containing the classes we will work with. If the code component you are using supplies a type library, as the ones built with Visual Basic do, you can use the New keyword when you declare the object to create it. As an example, here we are creating an object named objSorter of the Sorter class.

```
Dim objSorter As New Sorter
objSorter.Sort
```

The object is actually only created when you refer to it for the first time.

Whether or not a code component supplies a type library, you can use the CreateObject function in a Set statement to create a new object and assign an object reference to an object variable. To use CreateObject, you pass it the name of the class you want to create an object of. For example, here is how we created a new object named objExcel from the Microsoft Excel code component library in the previous topic, using CreateObject.

```
Dim objExcel As Object
Set objExcel = CreateObject("Excel.Sheet")
```

Note the way we specify the class name to create the object from—as Excel.Sheet (that is, as CodeComponent.Class). Because the code component you create with CreateObject do not need a type library, you must refer to the class you want to use as CodeComponent.Class instead of just by class, as you can with New.

You can also use the `GetObject` function to assign a reference to a new class, although it's usually used to assign a reference to an existing object. Here is how you use `GetObject`.

```
Set objectvariable = GetObject([pathname] [,class])
```

## NOTES

Here is what the arguments of the function mean:

- `pathname`—The path to an existing file or an empty string. This argument can be omitted.
- `class`—The name of the class you want to create an object of. If `pathname` is omitted, then `class` is required.

Passing an empty string for the first argument makes `GetObject` work like `CreateObject`, creating a new object of the class whose name is in `class`. For example, here is how we create an object of the class `ExampleClass` in the code component `NewClass` using `GetObject` (once again, we refer to the class we are using as `CodeComponent.Class`).

```
Set objNewClass = GetObject("", "NewClass.ExampleClass")
```

---

## BUILDING CLASSES

---

When you create a code component, you add code in class module(s), and when you register the code component with Windows, you make the class(es) available to client applications. Those applications, in turn, can add a reference to your code component and create an object of the class they want to use with the `New`, `CreateObject`, or other Visual Basic instruction. When the client application has an object corresponding to one of your classes, it can use that class's properties and methods.

That's how it works—it's all about reusing your code. You get an object corresponding to a class in a code component like this, where we are creating an object, `objCalendar`, of the class `CalendarClass` from the hypothetical code component named `PlannerCodeComponent`.

```
Dim objCalendar As Object
Set objCalendar = CreateObject("PlannerCodeComponent.
    CalendarClass")
. . .
```

Then you can use that object's properties and methods to give you access to the code in the class, something like this:

```
Dim objCalendar As Object
Set objCalendar = CreateObject("PlannerCodeComponent.
    CalendarClass")
. . .
```

If you have created your code component as an ActiveX EXE, that code component is an out-of-process server and runs separately from the client application; if you have created your code component as an ActiveX DLL, that code component is an in-process server, which means it will run as part of the client application's process.

There is even a way to use code components without creating an object. To make it easy to create code components that can be used with desktop tools like the Microsoft

Office Suite, Visual Basic allows you to label objects in a code component as global, which means they are part of a global object. In practice, this means that you do not have to create an object to use the methods and properties of this code component—you just add a reference to the code component in the client application, and you can use the component's properties and methods as though they were part of the client application.

A client application and an in-process component share the same memory space, so calls to the methods of an in-process code component can use the client's stack to pass arguments. That's not possible for an out-of-process component; there, method arguments must be moved across the memory boundary between the two processes which is called marshaling.

## NOTES

---

**SUMMARY**


---

1. Important functions of Database management are the security and validations of data items in Database.
2. In a file-server system, no intelligent process is active at the server level.
3. In a Client/Server system, the server is responsible for intelligently processing requests for data.
4. Working with DAO, you can use the Database and Recordset Data Access Objects in your procedures.
5. With the Remote Data Objects (RDO) library of data objects, you establish an rdoConnection to an ODBC data source, then create an rdoResultset.
6. ActiveX Data Objects (ADO) access data from OLE DB providers.
7. With Microsoft DAO Object Library you can create a DAO database.
8. For creating a table in a DAO database, you define it with a TableDef object.
9. To open an existing DAO database, you use the DAO OpenDatabase method.
10. To add a new record to a DAO record set, you use the AddNew method.
11. To delete a record in a DAO record set, you use the Delete method and then you update the record set.
12. You can search a record set with an index; we just set its Index property to the index we want to search and then set its Seek property to the string we want to search for.
13. You can execute an SQL statement when you create a DAO record set using the OpenRecordset method by placing that SQL statement in the source argument.
14. To open an RDO connection to a database, you can use the RDOOpenConnection method.
15. To move to the first record in an RDO result set, you can use the rdoResultset method MoveFirst.
16. To move to the last record in an RDO result set, you can use the rdoResultset method MoveLast.
17. To move to the next record in an RDO result set, you can use the rdoResultset Result Set method MoveNext.
18. To move to the previous record in an RDO result set, you can use the Result Set rdoResultset method MovePrevious.
19. The first step in editing an ADO database is to open that database, which is called a data source in ADO terminology, by setting up a Connection object.
20. To add a new record to an ADO record set, you use the AddNew method.
21. To move to the first record in an ADO record set, you use the Recordset object's MoveFirst method.

NOTES

22. You can execute an SQL statement when you open a record set using the Open method by passing that statement as the Source argument.
23. In Visual Basic you have several methods and ways for printing. Using Visual Basic you can print the entire form or line by line to a form. You can print to Immediate window.
24. To clear a form you can use CLS method. CLS method will clear the text on form.
25. To make the printer start printing you have to give Printer.EndDoc method.
26. PrintForm method is used to print an active form. This method sends a pixel-by-pixel image of a form to the printer.
27. Using NewPage method you can do multi-page document printing.
28. Using Printer collection you can query the list of printer objects which are registered on your computer system. If you query the collection then you are in position to change the default printer or can switch back to normal printer.
29. To see the values inside variables you can use Immediate window in Visual Basic. It is possible to display the values in Immediate window so that easily you can track the things.
30. In Visual Basic you can also created the reports and Microsoft Data Report designer is used to generate the reports.
31. A function is a block of code that you call and pass arguments to, and using functions helps break your code up in into manageable parts.
32. An array is an ordered series of data values, called elements that are referenced by number.
33. One-dimensional arrays, comprised of finite homogeneous elements.
34. Multi-dimensional arrays, comprised of elements, each of which is itself an array.
35. A two-dimensional array is an array in which each element is itself an array.
36. Control Array is a group of controls which share the same type, event procedures and name. Like this you can save the resources because you are not adding same controls again and again you simply make a copy of one.
37. OOPS can be used for reusability of objects. Everything here is associated with classes and objects.
38. Property provides the description of particular object. Properties of a class can make the default interface.
39. When you want to create an object from a code component, you first add a reference to that code component using the Visual Basic Project | References menu item

---

### SELFASSESSMENT QUESTIONS

---

1. In DAO, how would you create a database?
2. In DAO, how would you create a table with TableDef Object.
3. In DAO, how would you add fields to a TableDef Object?
4. In DAO, how would you create a Record Set?
5. In DAO, how would you open a database?
6. In DAO, how would you add a record to a RecordSet?
7. In DAO, how would you update a record in a RecordSet?
8. In RDO, how would you open a connection?
9. In RDO, how would you create a Record Set?
10. In RDO, how would you create a Result Set?
11. In RDO, how would you do the following:

Move to the first record	Move to the last record
Move to the next record	Move to the previous record.
12. In ADO, how would you do the following:

Open a connection	Create a Record set
-------------------	---------------------

Bind controls to Record Sets

Update a Record

Printers and Functions

Move to the first record

Move to the last record.

13. What is Dynamic array?
14. How are Dynamic arrays defined in Visual Basic?
15. What is an Array?
16. What is an One-dimensional arrays?
17. What are Multi-dimensional arrays?
18. What are Control Arrays?
19. What is OOPs programming?

### Multiple Choice Questions

1. Important functions of Database management are the \_\_\_\_\_ and validations of data items in Database.  
(a) security (b) management (c) tools
2. DAO means :  
(a) Data Access Object (b) Database Access Objects (c) Delete Access Object
3. RDO means :  
(a) Renew Data Objects (b) Remove Data Objects (c) Remote Data Objects
4. ADO means :  
(a) ActiveX Data Objects (b) ActiveX Delete Objects (c) Active Data Objects
5. OLE means :  
(a) Object Linking Ending  
(b) Object Liking Embedding  
(c) Object Linking Embedding
6. SQL means .  
(a) Standard Query Language (b) Structured Query Language  
(c) Selective Query Language

### True/False Questions

1. In a file-server system, intelligent process is active at the server level.
2. In a Client/Server system, the server is responsible for intelligently processing requests for data.
3. With the Remote Data Objects (RDO) library of data objects, you establish an rdoConnection to an ODBC data source, then create an rdoResultset.
4. ActiveX Data Objects (ADO) access data from OLE DB providers.
5. To open an existing DAO database, you use the DAO OpenDatabase method.
6. To add a new record to a DAO record set, you use the AddNew method.
7. You can execute an SQL statement when you create a DAO record set using the OpenRecordset method by placing that SQL statement in the source argument.
8. To move to the first record in an RDO result set, you can use the rdoResultset method MoveLast.
9. To move to the next record in an RDO result set, you can use the rdoResultset Result Set method MoveNext.
10. To add a delete a record to an ADO record set, you use the AddNew method.
11. You can execute an SQL statement when you open a record set using the Open method by passing that statement as the Source argument.

### Short Questions with Answers

1. How does Visual Basic handle database programming?
- Ans.** Visual Basic makes it simple to handle database by supplying a variety of tools and methods

NOTES

NOTES

for data management. Visual Basic version 6.0 has a full-fledged database engine called Jet. This engine contains virtually all the database functions that many applications would need. Visual Basic also provides a powerful and easy-to-use front-end development environments for database management. Using this, you can manipulate database in various formats. It also includes an in-built support for Open Database Connectivity (ODBC) – an industry standard for data access from multiple database formats.

2. What is DAO?

**Ans.** DAO is Data Access Objects. Working with DAO, you can use the Database and Recordset Data Access Objects in your procedures. The Database and Recordset objects each have properties and methods of their own and you can write procedures that use these properties and methods to manipulate your data.

3. What is RDO?

**Ans.** RDO is Remote Data Objects. With the Remote Data Objects (RDO) library of data objects, you establish an rdoConnection to an ODBC data source, then create an rdoResultset. The Remote Data Objects behave like the DAO objects in many ways, because there is a core set of methods that work with both record sets and result sets.

4. What is ADO?

**Ans.** ADO is ActiveX Data Objects. It access data from OLE DB providers. The Connection object is used to specify a particular provider and any parameters. To connect to a data source, you use a Connection object. Using that connection, you can create a new record set, and using the Recordset object's methods and properties, you can work with your data.

5. What is an array?

**Ans.** An array is an ordered series of data values, called elements that are referenced by number. Because arrays exist in memory, they provide fast data access and ease of manipulation. You can easily specify, locate or manipulate elements in an array. For example, you cannot have one element of the Integer data type and another of the Boolean data type in the same array. Each element of an array has a unique identifying index number. Changes made to one element of an array do not affect the other elements. To refer to any one of the values in an array, you need to use an index.

**ANSWERS**

**Multiple Choice Questions**

- |      |      |      |      |
|------|------|------|------|
| 1. a | 2. b | 3. c | 4. a |
| 5. c | 6. b |      |      |

**True False Questions**

- |      |       |       |      |
|------|-------|-------|------|
| 1. F | 2. T  | 3. T  | 4. T |
| 5. T | 6. T  | 7. T  | 8. F |
| 9. T | 10. F | 11. T |      |