

Visual Basic For Beginners

Module I

- Introduction to Programming
- Introduction to Object Oriented Programming
- What is Visual Basic ? Why VB for OOP ?
- Introduction to VB Integrated Development Environment (IDE)
- Understanding Forms and Standard (Basic) Controls
- Using Standard Controls

Introduction to Programming

- A computer program is a set of instructions which computer understands and performs actions according to them.
- 1 or more programs form an application.
- 1 or more applications form a Solution.
- Programs execute under an Operation System (OS)
 - DOS (Disk Operating System)
 - UNIX
 - Windows

Type of Programming Languages

- Traditional Programming
 - Linear Programming
 - Program flow controlled by the programmer
 - Program execution from Start to End, Line by Line (Linear Programming)
 - Procedural Programming
 - Program flow controlled by the programmer
 - Program is divided into number of smaller functional chunks called Procedures
 - All procedures were called from the main procedure

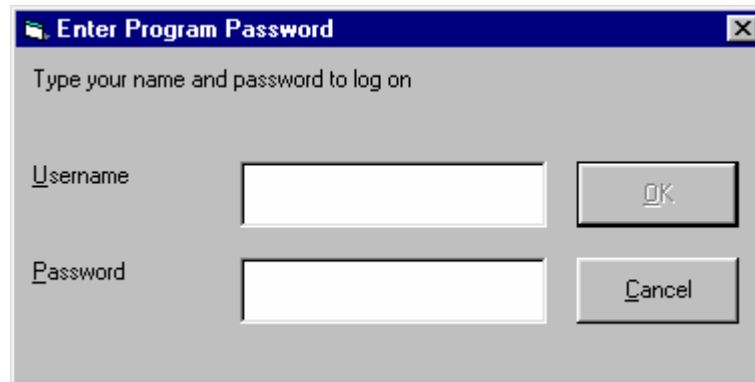
Event Driven Programming

- An event is something to which objects respond.
- Events can be triggered by :
 - A User's action
 - User clicks on a command button.
 - User hits a key on the keyboard.
 - A program instruction.
 - Operating System
 - OS getting shut down.
 - User Logged Off
 - A program is closed from Task Manager.
 - By some other application exchanging information in a multitasking environment.
- Programming applications to respond to these events is Event Driven Programming.
- Program flow is controlled by the User instead of Programmer.

GUI Applications

- A Windows application usually has GUI (Graphical User Interface) which allows users to input data.
- GUI is normally consisted of number of graphical objects like command button, text boxes, labels etc.
- Event Driven Applications.
- Requires an OS which supports graphics.
 - Microsoft Windows
 - Windows 3.1 / 3.11 (16 bit OS)
 - Windows 95 / 98 (32 bit OS)
 - Windows NT (32 bit OS)

An Example of GUI



- Visual Basic is a programming language for developing GUI applications.
- Other choice of languages :
 - Microsoft Visual Studio Suite
 - Visual C++ (for low level / complex programming)
 - Visual Foxpro (for Database centric applications)
 - Visual InterDev (for Internet/Web based apps)

Why Visual Basic?

- Visual Basic allows us to develop fast, powerful applications with less complexity than C++.
- It is RAD tool (Rapid Application Development).
- Easy to learn and easy to develop powerful applications in less time.
- Cost effective programming tool.
- A popular choice.

Introduction to Object Oriented Programming

- What are Objects ?
 - Real World Objects
 - Car, Table, Chair, Pen, Printer etc.
 - Car has attributes like color, no. of doors, no. of gears, make, registration number etc.
 - Car has got methods like steering, acceleration, de-acceleration / braking, changing gears etc.
 - Programming Objects
 - A Object is a programming term to describe a special kind of data item.
 - In addition to containing the data, an Object also knows how to manipulate on its data.
 - Each different processes that the Object can perform on its own data is knows as **METHODS**
 - The technical term used for object data is **ATTRIBUTE / PROPERTY.**
 - The combination of data and associated methods is called **ENCAPSULATION.**

Properties, Methods and Events

- Property
 - A changeable characteristic of an object that represents part of the state of the object.
 - This includes characteristics like color, size and caption.
 - Other objects (or code) can set these properties to various values.
 - Generally, properties are named using nouns (Text, BackColor, DataSource) or adjectives (Visible, Enabled)
 - Property/Attribute corresponds to the data item in an object.

Properties, Methods and Events

- Methods
 - Methods are actions that the object can perform.
 - They may or may not change the state of the object.
 - Generally, they are named using verbs, such as Open, Move and Hide.
 - Methods normally manipulate object's data.

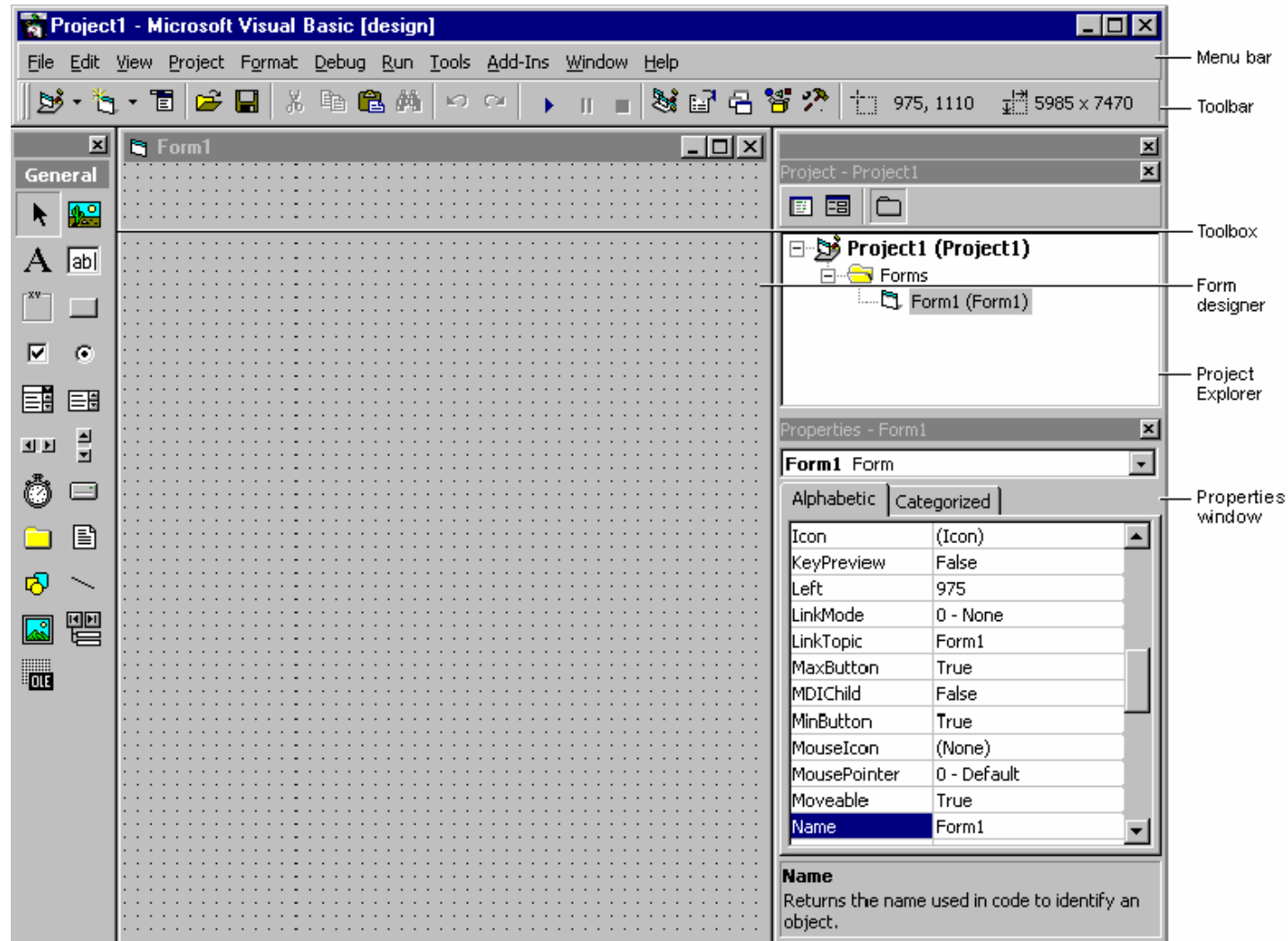
Properties, Methods and Events

- Events
 - An event is a characteristic that the object may react to.
 - An event notifies that something has happened with the object, allowing it to react in some way.
 - For example, the Command Button has a Click Event which notifies that it has been clicked by the user.
 - Events occur in the life of real world objects also like for a human being, events are birth, marriage, death.

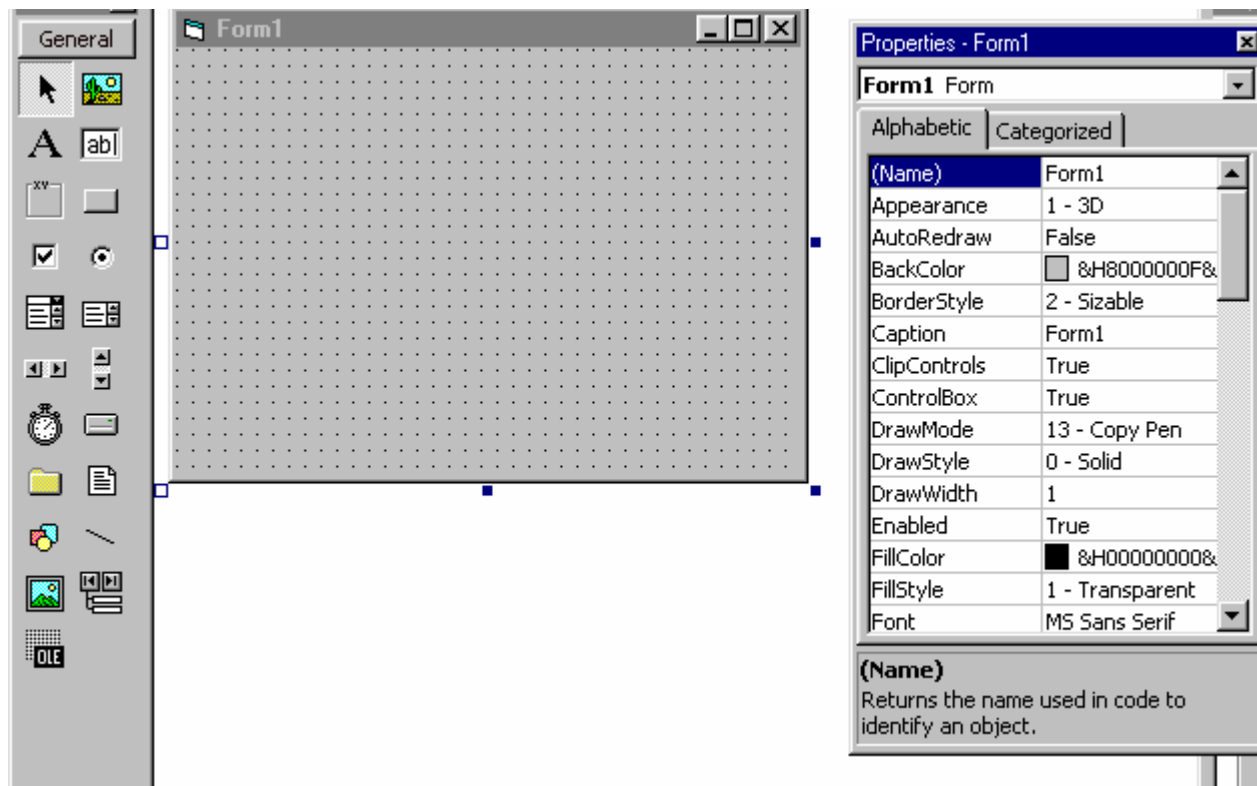
Why VB for OOP?

- Visual Basic is an Object Oriented Language.
- Most of the VB programming involves programming objects like Forms, Text Boxes, Command Buttons etc.
- It is a RAD tool, easy to learn and easy to implement.
- VB provides a very effective IDE (Integrated Development Environment) for Rapid Development.

The Visual Basic IDE

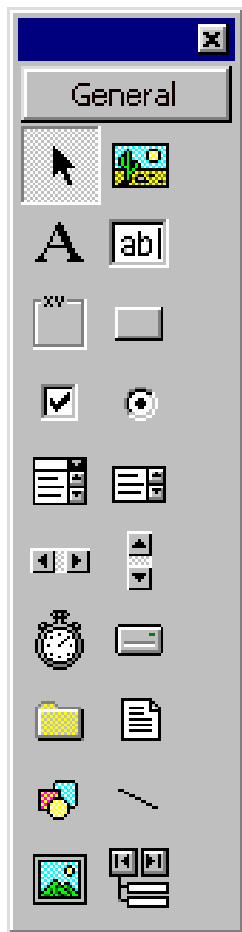


Visual Basic Objects



- Forms and all other controls are Objects.
- They have Properties, Methods and they respond to Events.

ToolBox - A Collection of Controls



Standard Toolbox Controls



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.



CommandButton

Creates a button the user can choose to carry out a command.

Standard Toolbox Controls



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.



HScrollBar (horizontal scroll bar)

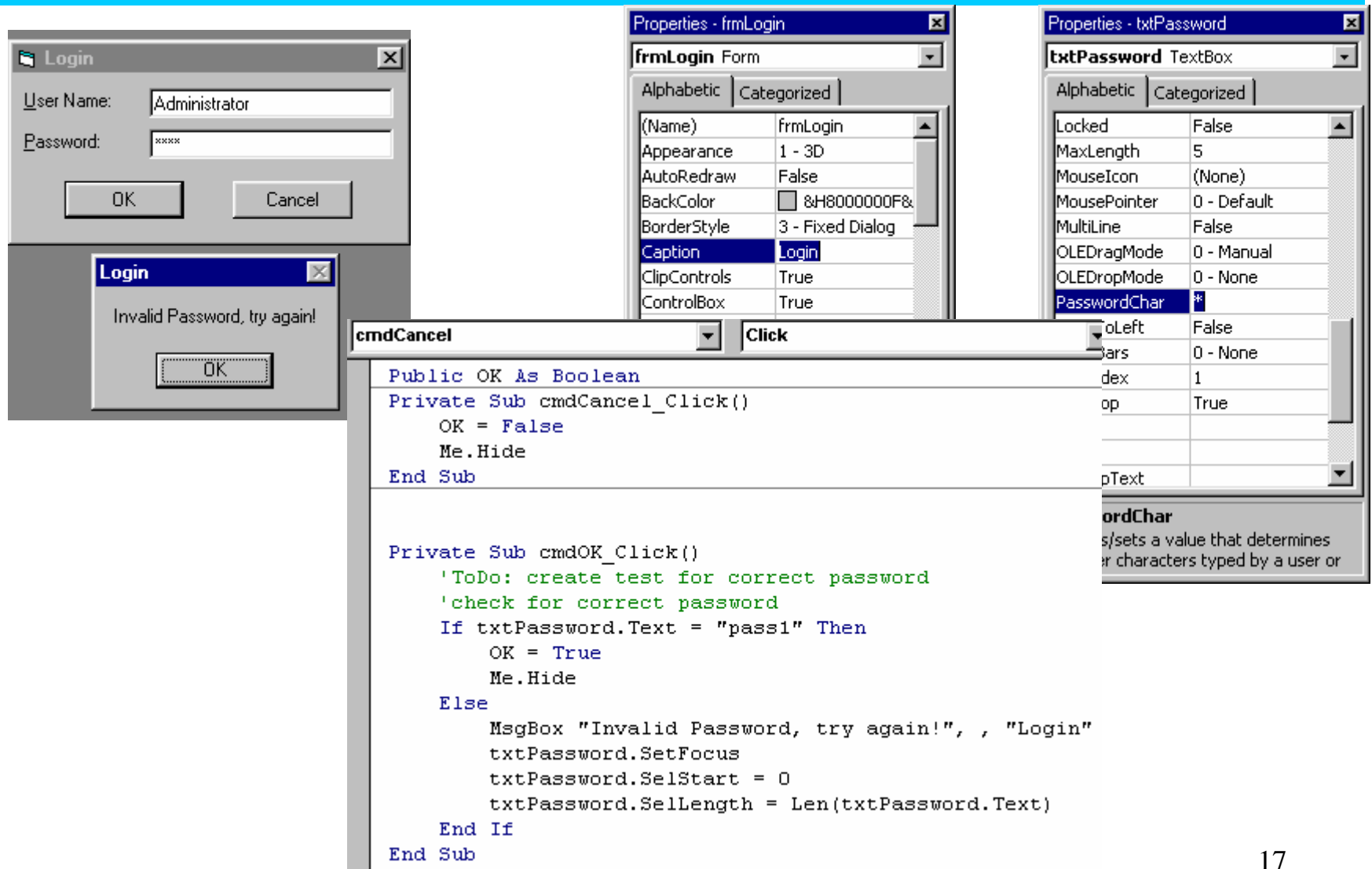
Provides a graphical tool for quickly navigating through a long list of items or a large amount of information, for indicating the current position on a scale, or as an input device or indicator of speed or quantity.



VScrollBar (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.

Using Standard Controls



The image displays a Visual Basic IDE with three windows:

- Login Form:** A dialog box with "User Name:" (Administrator) and "Password:" (masked with asterisks) fields, and "OK" and "Cancel" buttons.
- Properties - frmLogin:** Shows the form's properties, including Name (frmLogin), Caption (Login), and ControlBox (True).
- Properties - txtPassword:** Shows the password control's properties, including PasswordChar (*).

The code editor shows the following VBA code:

```
Public OK As Boolean
Private Sub cmdCancel_Click()
    OK = False
    Me.Hide
End Sub

Private Sub cmdOK_Click()
    'ToDo: create test for correct password
    'check for correct password
    If txtPassword.Text = "pass1" Then
        OK = True
        Me.Hide
    Else
        MsgBox "Invalid Password, try again!", , "Login"
        txtPassword.SetFocus
        txtPassword.SelStart = 0
        txtPassword.SelLength = Len(txtPassword.Text)
    End If
End Sub
```

Module II

- Contents
 - Data Types
 - Constants and Variables
 - Data Type Conversion Functions
 - Programming Constructs

Data Types

- Data type describes the type of information stored within memory.
- VB contains many data types.
- Each data type has a specific name that is used to refer to the information it can hold, as well as the amount of memory used by that data type.
- Data types allow for storage of simple numbers, complex numbers, strings , dates and times, as well as objects.

Data Type Summary

Data type	Storage size	Range
Byte	1 byte	0 to 255
Boolean	2 bytes	True or False
Integer	2 bytes	-32,768 to 32,767
Long (long integer)	4 bytes	-2,147,483,648 to 2,147,483,647
Single (single-precision floating-point)	4 bytes	-3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values
Double (double-precision floating-point)	8 bytes	-1.79769313486232E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values
Currency (scaled integer)	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	14 bytes	+/-79,228,162,514,264,337,593,543,950,335 with no decimal point; +/-7.9228162514264337593543950335 with 28 places to the right of the decimal; smallest non-zero number is +/-0.00000000000000000000000000000001
Date	8 bytes	January 1, 100 to December 31, 9999

Data Type Summary (contd..)

Object	4 bytes	Any Object reference
String (variable-length)	10 bytes + string length	0 to approximately 2 billion
String (fixed-length)	Length of string	1 to approximately 65,400
Variant (with numbers)	16 bytes	Any numeric value up to the range of a Double
Variant (with characters)	22 bytes + string length	Same range as for variable-length String
User-defined (using Type)	Number required by elements	The range of each element is the same as the range of its data type.

Variant

- The **Variant** data type is the data type for all variables that are not explicitly declared as some other type (using statements such as **Dim**, **Private**, **Public**, or **Static**).
- A **Variant** is a special data type that can contain any kind of data except fixed-length String data.
- A **Variant** can also contain the special values Empty, **Error**, **Nothing**, and Null.
- You can use the **Variant** data type in place of any data type to work with data in a more flexible way.

Variant

- Declaration of a Variant

```
Dim MyVar As Variant
```

```
MyVar = 98052
```

- The value Empty denotes a **Variant** variable that hasn't been initialized (assigned an initial value).
- Don't confuse **Empty** with Null. **Null** indicates that the **Variant** variable intentionally contains no valid data.
- Avoid using Variants unless there is a requirement to do so as they occupy more memory than any other data type.

Variables

- A named storage location that can contain data that can be modified during program execution.
- Each **variable** has a name that uniquely identifies it within its scope.
- A data type can be specified or not.
- **Variable** names must begin with an alphabetic character, must be unique within the same scope, can't be longer than 255 characters, and can't contain an embedded period or type-declaration character.

Declaring a Variable

- `<Scope> <VariableName> As <DataType>`
- Scope of the variables
 - Depending on how it's declared, **variable** is scoped in one **of** three ways, as shown in the following table.

Declaration

Procedure **Dim** or **Static** within the procedure

Private **Dim** or **Private** at the top **of** the module

Public **Public** at the top **of** the module

- Static variables retain their values in between calls.

Variable Prefixes

Use the following prefixes to indicate a variable's data type.

Data type	Prefix	Example
Boolean	bln	blnFound
Byte	byt	bytRasterData
Collection object	col	colWidgets
Currency	cur	curRevenue
Date (Time)	dtm	dtmStart
Double	dbl	dblTolerance
Error	err	errOrderNum
Integer	int	intQuantity
Long	lng	lngDistance
Object	obj	objCurrent
Single	sng	sngAverage
String	str	strFName
User-defined type	udt	udtEmployee
Variant	vnt	vntChecksum

Example of a variable declaration

```
Dim strName as String
```

```
Dim blnMarried as Boolean
```

```
Dim curSalary as Currency
```

```
Dim intYears as Integer
```

Constants

- A named item that retains a constant value throughout the execution of a program, as opposed to a variable, whose value can change during execution.
- Constants may be defined by the user with the **Const** statement.

```
Const A = "MyString"
```

- VB itself has many pre-defined constants.

```
Msgbox "This is an example", vbInformation  
(Here vbInformation is a Visual Basic pre-  
defined constant)
```

Data Type Conversion Functions

The function name determines the return type as shown in the following:

Function	Return Type	Range for <i>expression</i> argument
Cbool	Boolean	Any valid string or numeric expression.
Cbyte	Byte	0 to 255.
Ccur	Currency	-922,337,203,685,477.5808 to 922,337,203,685,477.5807.
Cdate	Date	Any valid date expression .
CDbl	Double	-1.79769313486232E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values.
Cdec	Decimal	+/- 79,228,162,514,264,337,593,543,950,335 for zero-scaled numbers, that is, numbers with no decimal places. For numbers with 28 decimal places, the range is +/-7.9228162514264337593543950335. The smallest possible non-zero number is 0.00000000000000000000000000000001.
Cint	Integer	-32,768 to 32,767; fractions are rounded.

Type Conversion Functions (contd..)

Function	Return Type	Range for <i>expression</i> argument
CLng	Long	-2,147,483,648 to 2,147,483,647; fractions are rounded.
CSng	Single	-3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values.
CStr	String	Returns for CStr depend on the <i>expression</i> argument.
Cvar	Variant	Same range as Double for numerics. Same range as String for non-numeric.

Programming Constructs

- The following are the Programming Building Blocks available in Visual Basic
 - Decision Making Constructs
 - If...Then...Else...End If
 - Select...Case...End Select
 - Looping Constructs
 - For...Next
 - Do While...Loop
 - Do Loop While

The If Statement

- Visual Basic procedures can test conditions and then, depending on the results of that test, perform different operations. The If Statement can be used in either of the following forms.
 - If...Then
 - If...Then...Else
 - Select Case

- **If...Then**

Use an If...Then structure to execute one or more statements conditionally. You can use either a single-line syntax or a multiple-line *block* syntax:

If *condition* **Then** *statement*

If *condition* **Then**
statements

End If

If...Then..Else

- **If...Then...Else**

Use an If...Then...Else block to define several blocks of statements, one of which will execute:

If *condition1* **Then**

[*statementblock-1*]

[**ElseIf** *condition2* **Then**

[*statementblock-2*]] ...

[**Else**

[*statementblock-n*]]

End If

Select Case Statement

- Visual Basic provides the Select Case structure as an alternative to If...Then...Else for selectively executing one block of statements from among multiple blocks of statements.
- A Select Case statement provides capability similar to the If...Then...Else statement, but it makes code more readable when there are several choices.
- A Select Case structure works with a single test expression that is evaluated once, at the top of the structure. Visual Basic then compares the result of this expression with the values for each Case in the structure.

Structure of Select Case

```
Select Case testexpression  
  [Case expressionlist1  
    [statementblock-1]]  
  [Case expressionlist2  
    [statementblock-2]]  
  .  
  .  
  .  
  [Case Else  
    [statementblock-n]]  
End Select
```

```
Dim Number  
Number = 8 ' Initialize variable.  
Select Case Number ' Evaluate Number.  
Case 1 To 5 ' Number between 1 and 5, inclusive.  
  Debug.Print "Between 1 and 5"  
  ' The following is the only Case clause that  
  ' evaluates to True.  
Case 6, 7, 8 ' Number between 6 and 8.  
  Debug.Print "Between 6 and 8"  
Case 9 To 10 ' Number is 9 or 10.  
  Debug.Print "Greater than 8"  
Case Else ' Other values.  
  Debug.Print "Not between 1 and 10"  
End Select
```

Looping Constructs

- When a set of program statements are executed again and again, the concept is called Looping.
- A programmer should have answer to one of the following questions for implementing valid looping.
 - How many times the loop needs to be executed ?
(For...Next)
 - What condition can make the loop terminate ?
(Do...Loop Until)
 - What condition can make the loop continue ?
(Do...Loop While)
- Any loop which can never be terminated, is called as an Endless Loop (*a programming flaw*).

Do While..Loop

- Use a Do loop to execute a block of statements an indefinite number of times.
- As with If...Then, the *condition* must be a value or expression that evaluates to False (zero) or to True (nonzero).
- In the following Do...Loop, the *statements* execute as long as the *condition* is True:

Do While *condition*
statements

Loop

- When Visual Basic executes this Do loop, it first tests *condition*. If *condition* is False (zero), it skips past all the statements. If it's True (nonzero), Visual Basic executes the statements and then goes back to the Do While statement and tests the condition again.

Do..Loop While / Do..Loop Until

- Another variation of the Do...Loop statement executes the statements first and then tests *condition* after each execution. This variation guarantees at least one execution of *statements*:

Do

statements

Loop While *condition*

- Two other variations are analogous to the previous two, except that they loop as long as *condition* is False rather than True.

Do Until *condition*

statements

Loop Do

statements

Loop Until *condition*

For..Next

- Do loops work well when you don't know how many times you need to execute the statements in the loop.
- When you know you must execute the statements a specific number of times, however, a For..Next loop is a better choice.
- Unlike a Do loop, a For loop uses a variable called a counter that increases or decreases in value during each repetition of the loop.
- The syntax is:

```
For counter = start To end [Step increment]  
    statements  
Next [counter]
```
- The arguments *counter*, *start*, *end*, and *increment* are all numeric.

Example of For..Next

- Example

```
Dim I as Integer
For I=1 to 100 Step 1
    Text1.Text=CStr(I)
Next
```

- Nested For Statements

```
For I=1 to 5
    For J=1 to 10
        Text1.Text=CStr(I) & “,” & CStr(J)
    Next
Next
```

(How many times the innermost statement will execute ?)

- Use Exit For to terminate a For..Next loop.
- User Exit Do to terminate a Do...While loop.

Module III

Contents

- **Modules**
- **List boxes, Option Boxes, Check Boxes**
- **Status Bars**

Forms, Codes & Files

Visual Basic stores code in three kinds of modules:

- **Form Modules**
- **Standard Modules**
- **Class Modules**

Each standard, class, and form module can contain:

- ◆ Procedures. A Sub, Function, or Property procedure contains pieces of code that can be executed as a unit.
- ◆ Declarations. You can place constant, type, variable, and dynamic-link library (DLL) procedure declarations at the module level of form, class or standard modules.

The three kinds of modules form a kind of hierarchy that allows you to organise your code. How you organise the code has a significant impact on its performance, as well as on your ability to maintain the code.

Form Modules

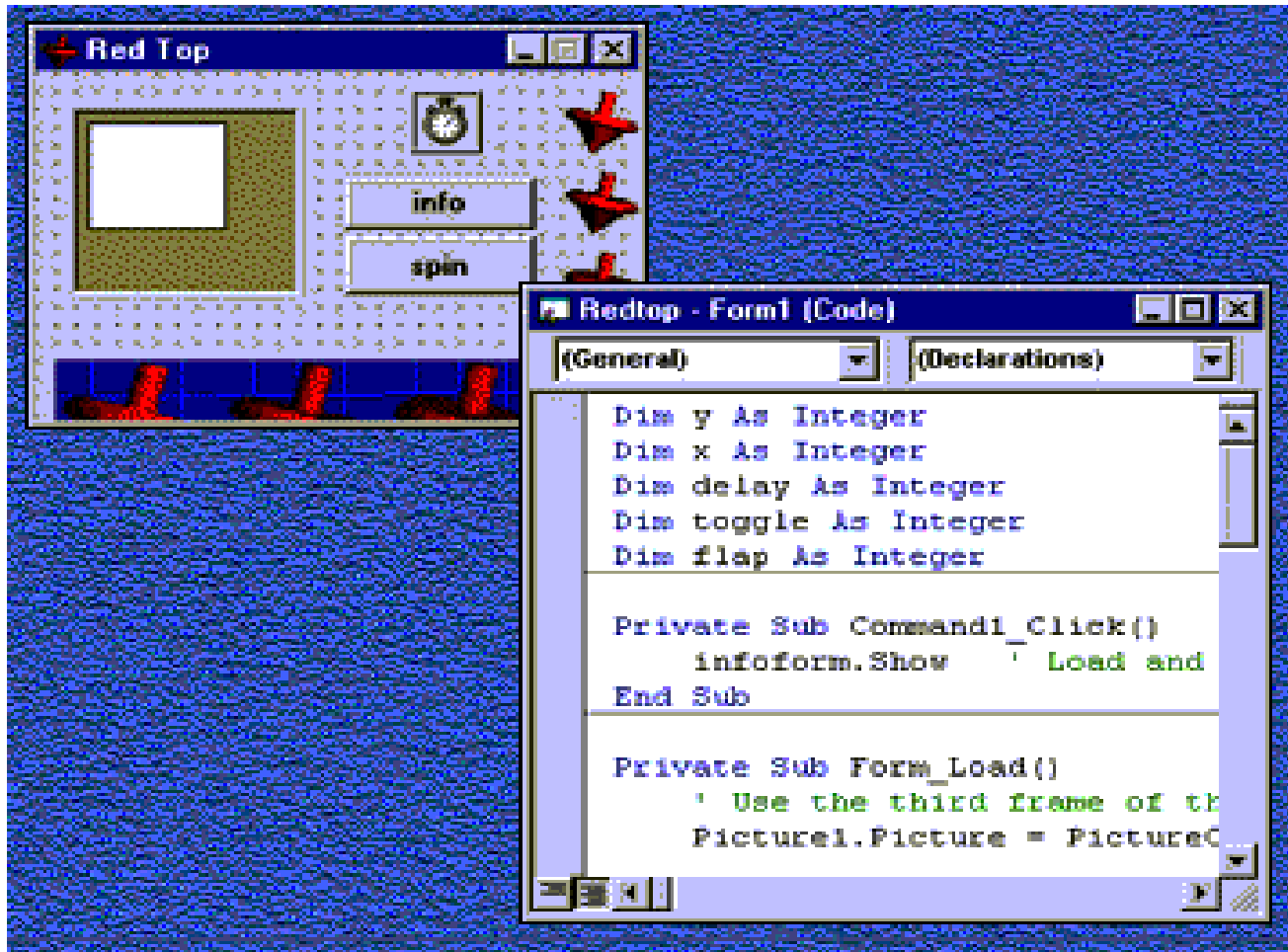
Each form in an application has an associated form module that contains:

- Property settings for the form and its controls.
- Form-level variable declarations.
- Event procedures and form-level general procedures.

Form modules (.frm file extension) can contain procedures that handle events as they occur within a form. Form modules also can contain general procedures and form-level declarations of variables, constants, types, and external procedures. The code that you write in a form module is specific to the application that contains the form.

Illustration

This illustration shows a form module for the form RedTop.



Standard Modules

When an application contains code that is shared by several forms, you should store it in a standard module. Code in a standard module can also be *public* to make it shareable by all modules in the application. Procedures stored in a standard module are called *general procedures*.

Standard modules (.bas files) can contain public (available to the whole application) or module-level declarations of variables, constants, types, external procedures, and global procedures. The code written in a standard module need not be tied to a particular application. You can potentially reuse code from a standard module in many different applications.

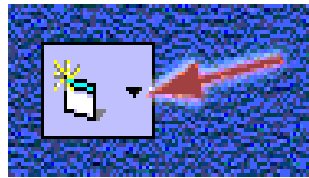
Note The code in standard modules is public by default. This means that it is easily shared with other code modules, such as a form module. In contrast, procedures and functions specified as private are visible only in the module in which they are declared.

Note A standard module is referred to as a code module in earlier versions of Visual Basic.

Creating a Standard Module

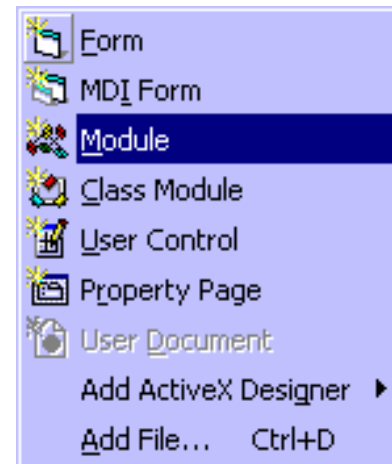
There are two ways to create a standard module:

Click the down-arrow for the Add Form button on the toolbar, shown in the following illustration, and then click Module



– or –

Right-click in the Project Explorer window, click Add, and then click Module from the pop-up menu that appears, as shown in the following illustration.



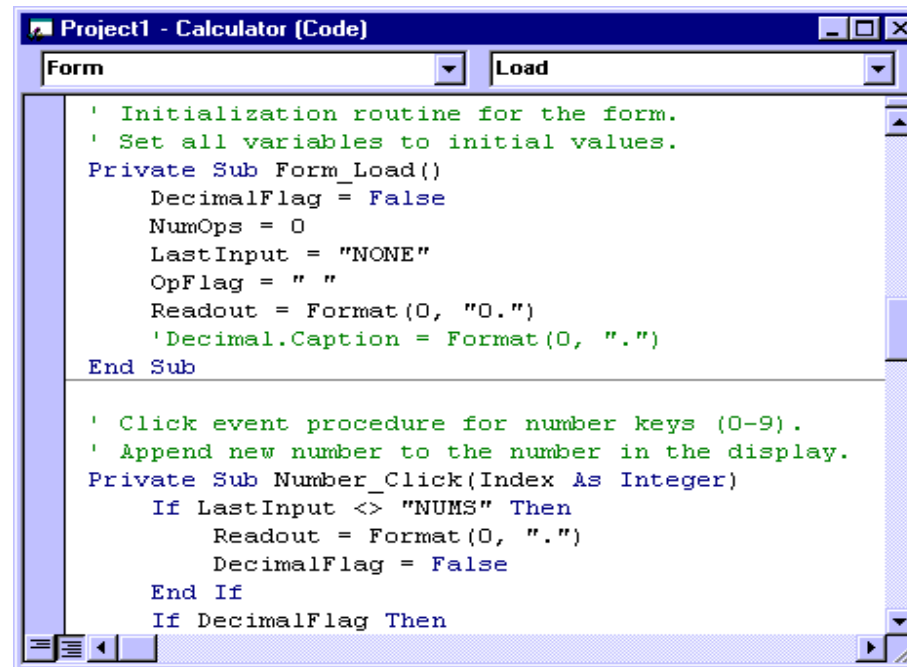
Class Module

The third type of code module is the class module. Class modules contain the definition of classes (property and method definitions) used to create new objects. These new objects can include customised properties and methods.

Examination of class modules is beyond the scope of this course

Using the Code Editor Window

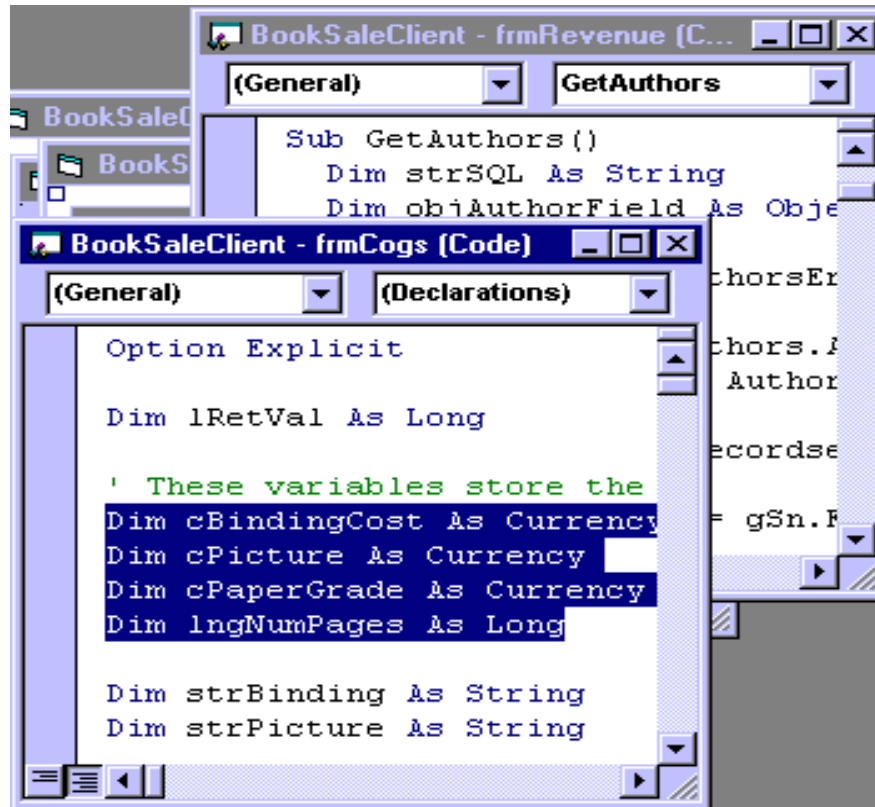
The Visual Basic Code Editor window (also known as the Code window) is where you write Visual Basic code for your application. The Code Editor is like a highly specialised word processor with a number of features that make writing Visual Basic code easier.



```
Project1 - Calculator [Code]
Form Load
' Initialization routine for the form.
' Set all variables to initial values.
Private Sub Form_Load()
    DecimalFlag = False
    NumOps = 0
    LastInput = "NONE"
    OpFlag = " "
    Readout = Format(0, "0.")
    'Decimal.Caption = Format(0, ".")
End Sub

' Click event procedure for number keys (0-9).
' Append new number to the number in the display.
Private Sub Number_Click(Index As Integer)
    If LastInput <> "NUMS" Then
        Readout = Format(0, ".")
        DecimalFlag = False
    End If
    If DecimalFlag Then
```

Using Multiple Code Windows

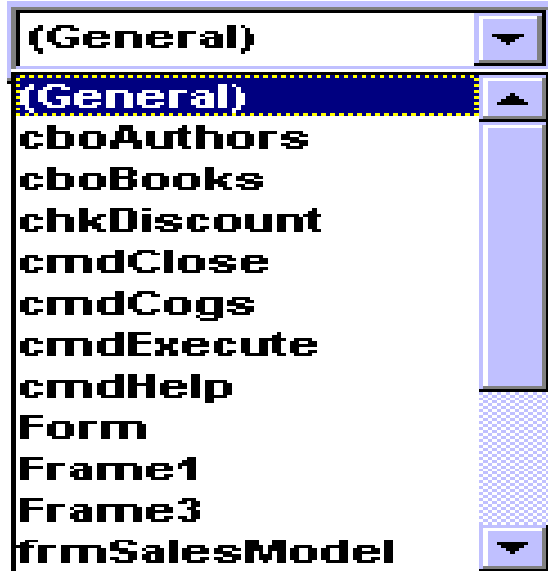


Because you write Visual Basic code in modules, Visual Basic lets you open a separate Code Editor window for each module you select from the Project Explorer

Using the Object and Procedure ListBoxes

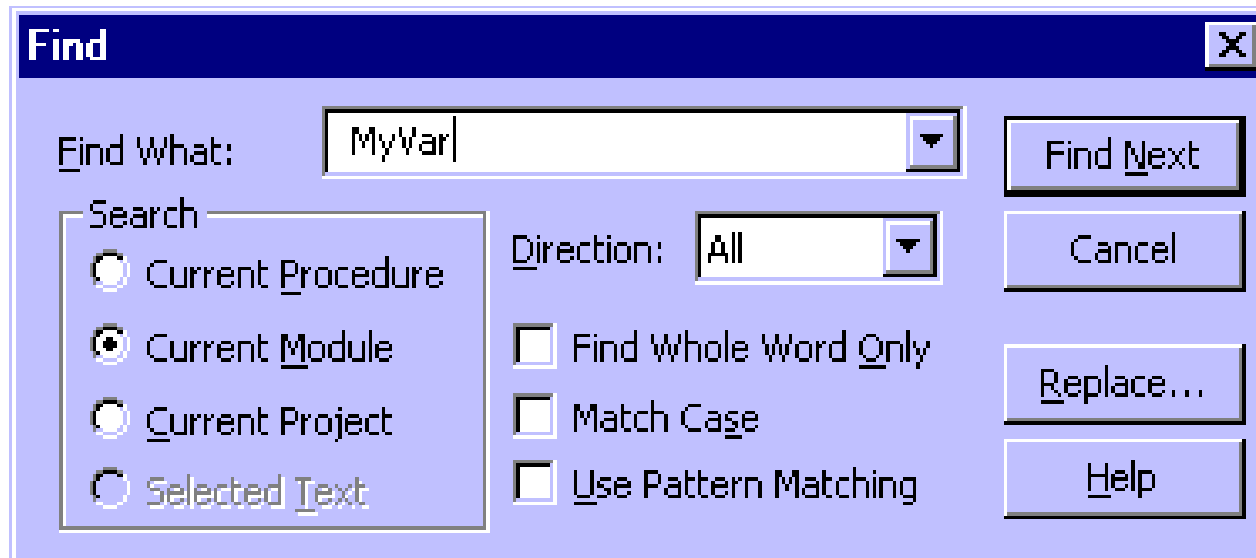
The Object list box in the Code Editor window makes it easier to navigate to a specific object by displaying a list of all objects associated with the module.

Each section of code can contain several different procedures. You access the procedures by using the Procedure list box.



Finding Text

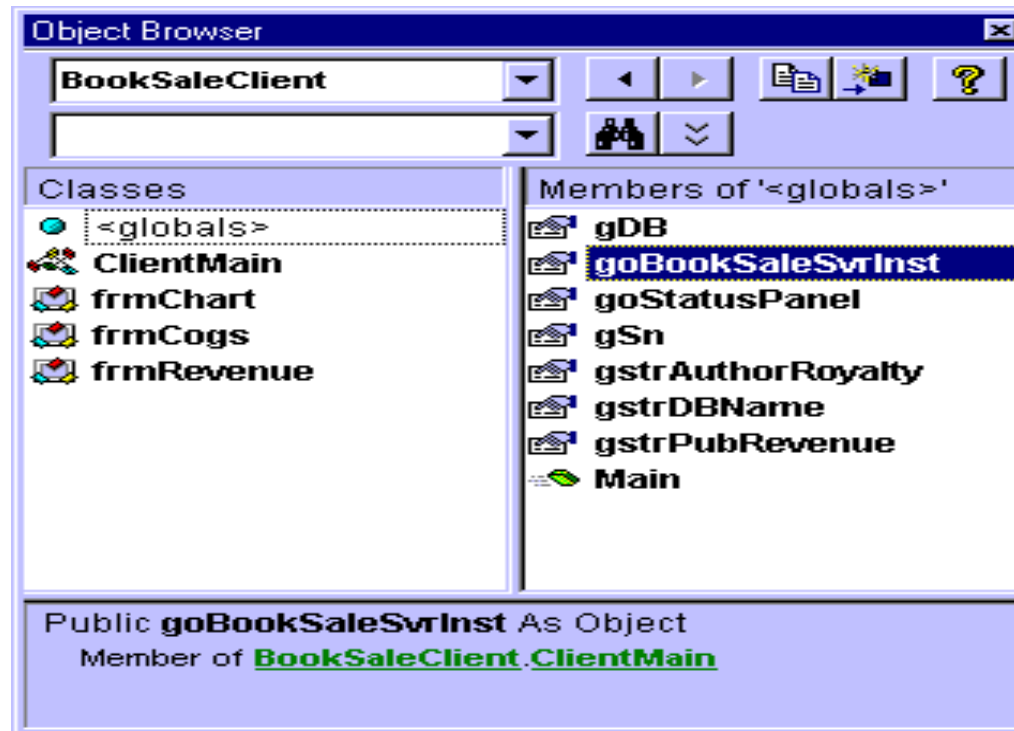
The Find command on the Edit menu is useful when searching for specific text. You can search specific sections of code, such as the current selection, procedure, or module, or the entire project, as shown below



Displaying Procedures Using Object Browser

The Object Browser displays the procedures available in any active or referenced project or library.

This illustration shows the Object Browser



Using Bookmarks

Bookmarks are placeholders in the Code Editor window. You can set them to mark points in the code that you want to access quickly.

To set a bookmark

1. Position the cursor at the desired line of code.
2. On the Edit menu, select Bookmarks, then click Toggle.

To clear a bookmark

1. Position the cursor at the line of code that is marked.
2. On the Edit menu, select Bookmarks, then click Toggle.

To jump to a bookmark

On the Edit menu, select Bookmarks, then click Next Bookmark or Previous Bookmark.

To clear all bookmarks

On the Edit menu, select Bookmarks, then click Clear All Bookmarks

Using Line Continuation Character

Line Continuation Character

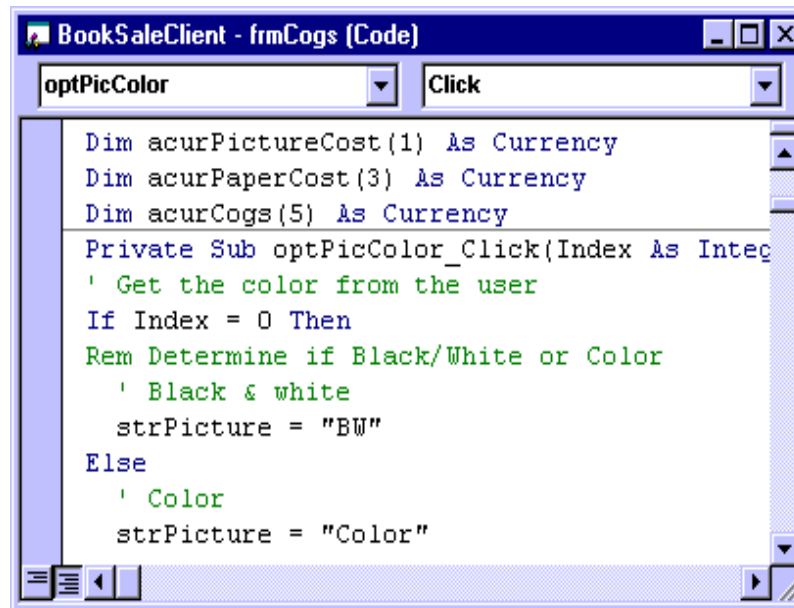
You can use the line-continuation character, the underscore (`_`), to break up a single code statement into multiple lines. This makes the code statement easier to read because it's fully contained within the Code Editor window.

The line-continuation character is placed after a space in the statement. For example:

```
MsgBox prompt:="The password is invalid!", _  
    buttons:=49, _  
    title:=" Sign In", _  
    helpfile:="SignIn.hlp", _  
    context:=3
```

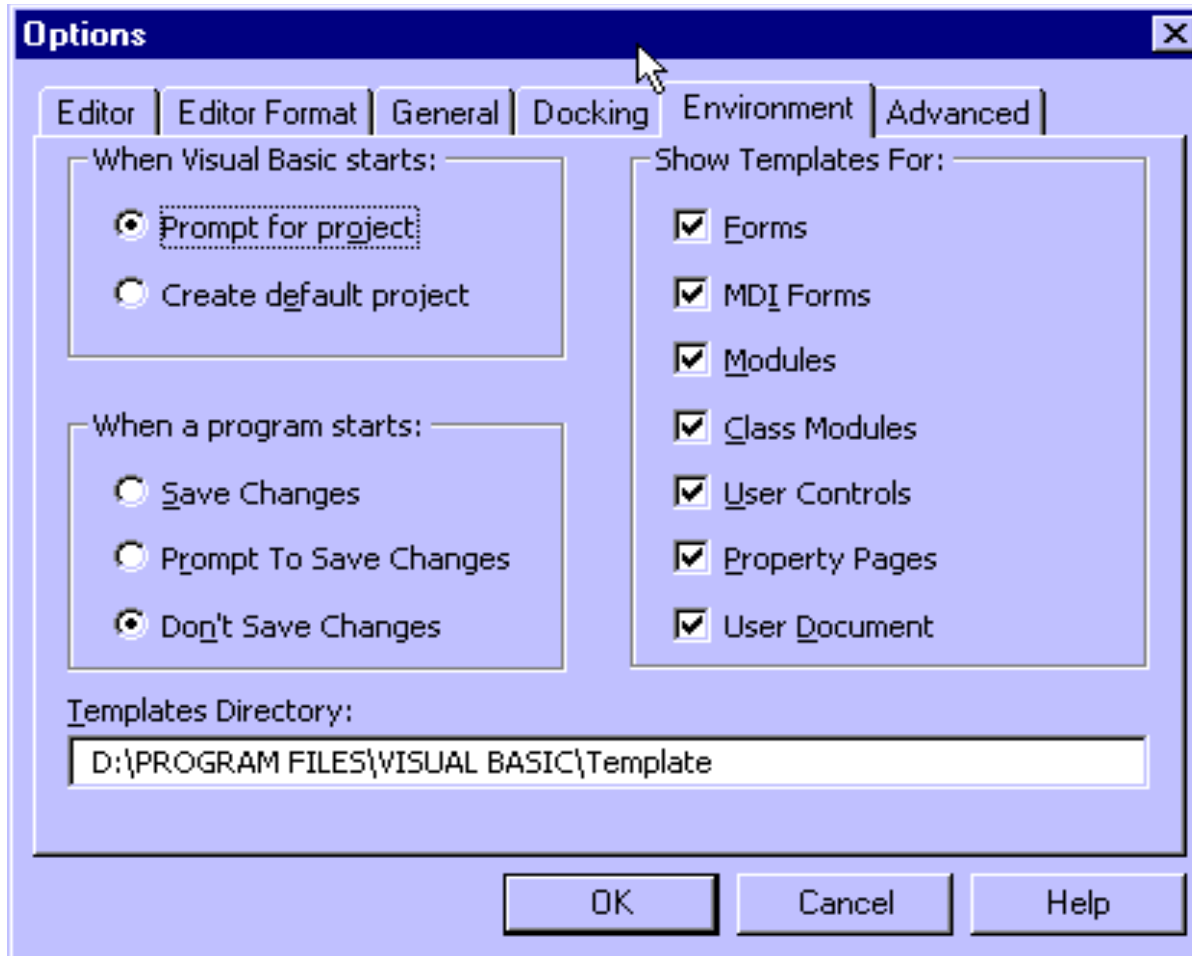
Adding Comments

Adding comments to code makes it easier for someone else to determine what the code does. It also helps you to understand the code at some later date. Visual Basic offers two methods for adding comments to code. Visual Basic ignores anything following a single quote ('), so comments can be placed on their own line or at the end of a line of code. Also, preceding any line of code with Rem (an abbreviation of "Remark") instructs Visual Basic to ignore it.

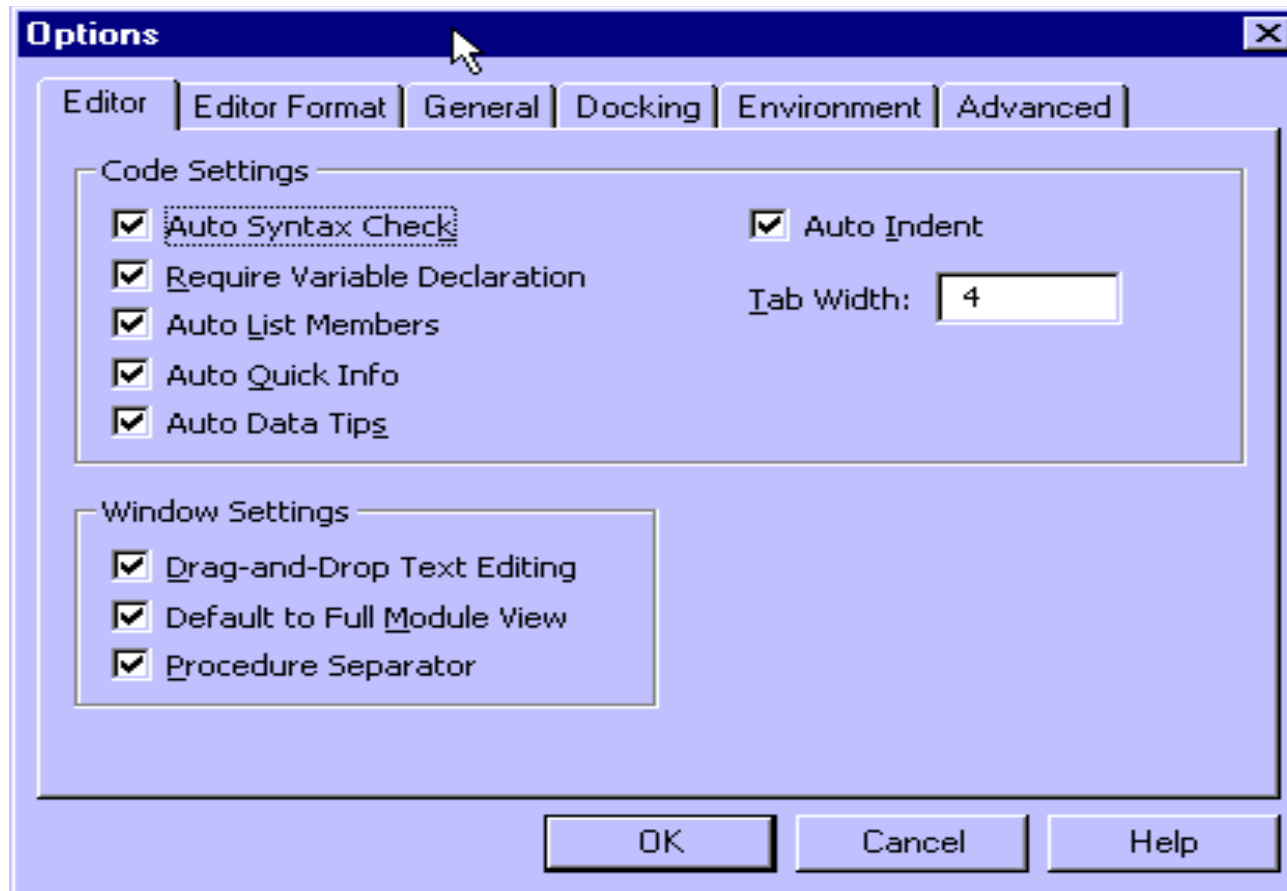


```
BookSaleClient - frmCogs (Code)
optPicColor Click
Dim acurPictureCost(1) As Currency
Dim acurPaperCost(3) As Currency
Dim acurCogs(5) As Currency
Private Sub optPicColor_Click(Index As Integer)
    ' Get the color from the user
    If Index = 0 Then
        Rem Determine if Black/White or Color
        ' Black & white
        strPicture = "BW"
    Else
        ' Color
        strPicture = "Color"
    End If
End Sub
```

Setting Environment Options

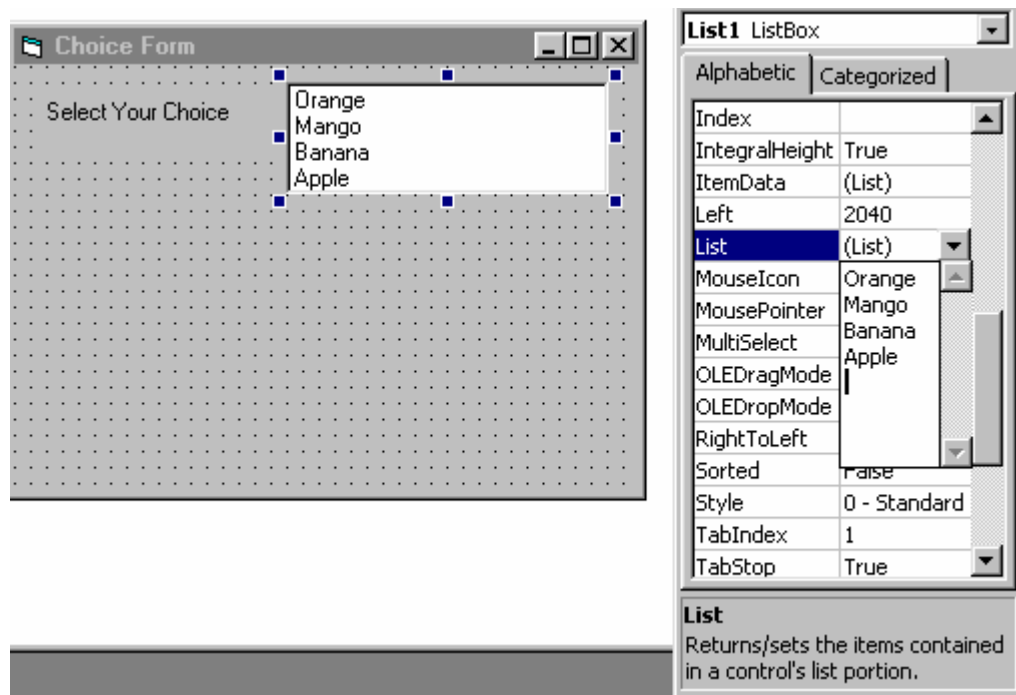


Setting Editor Options



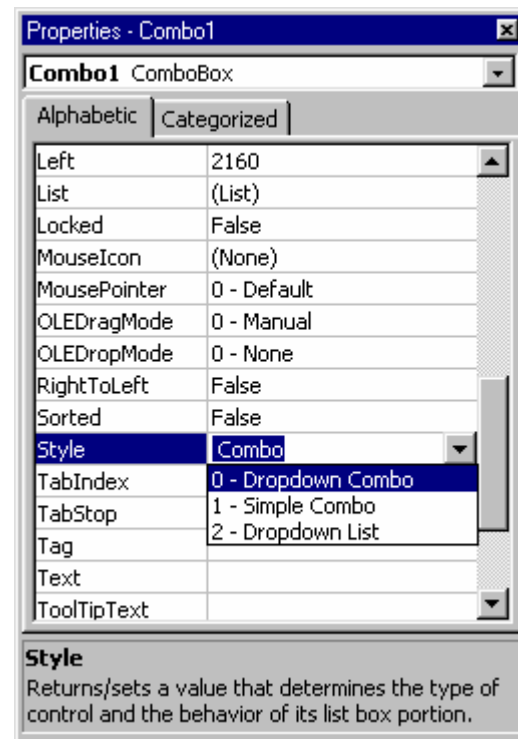
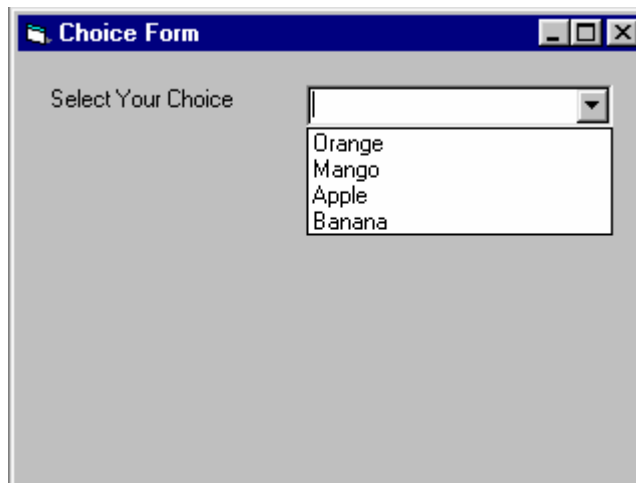
List Box

- List boxes display lists of items, so that the user can see what is available and select one.
- If the list is too big to fit in the list box, vertical or horizontal scroll bars are added.



Combo Box

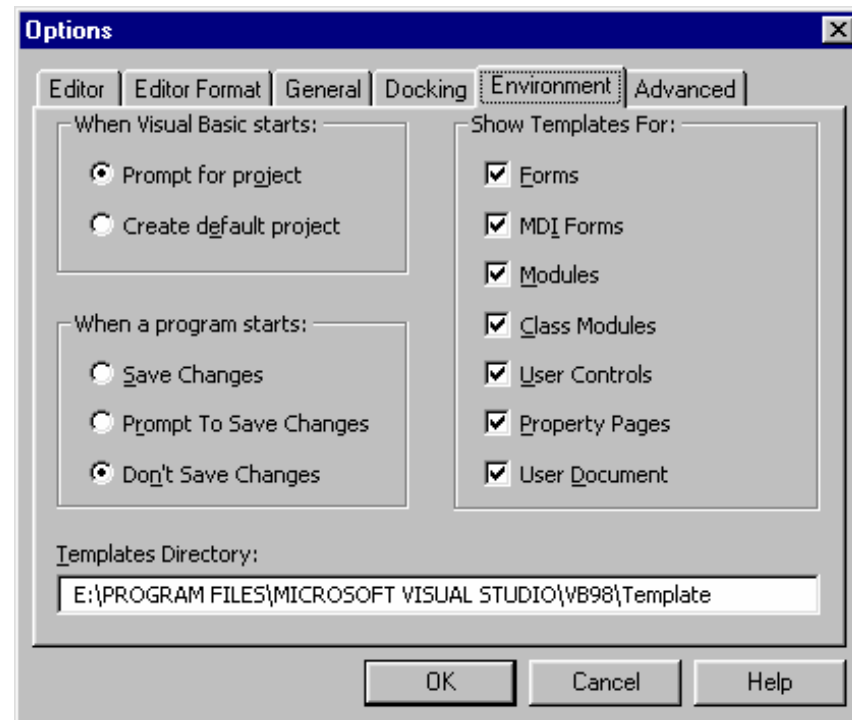
A Combo Box combines a drop down list with a slot in which users can enter their own data



Option Boxes

Option Boxes let you select only one item at a time whereas Check Boxes let you select multiple items at the same time.

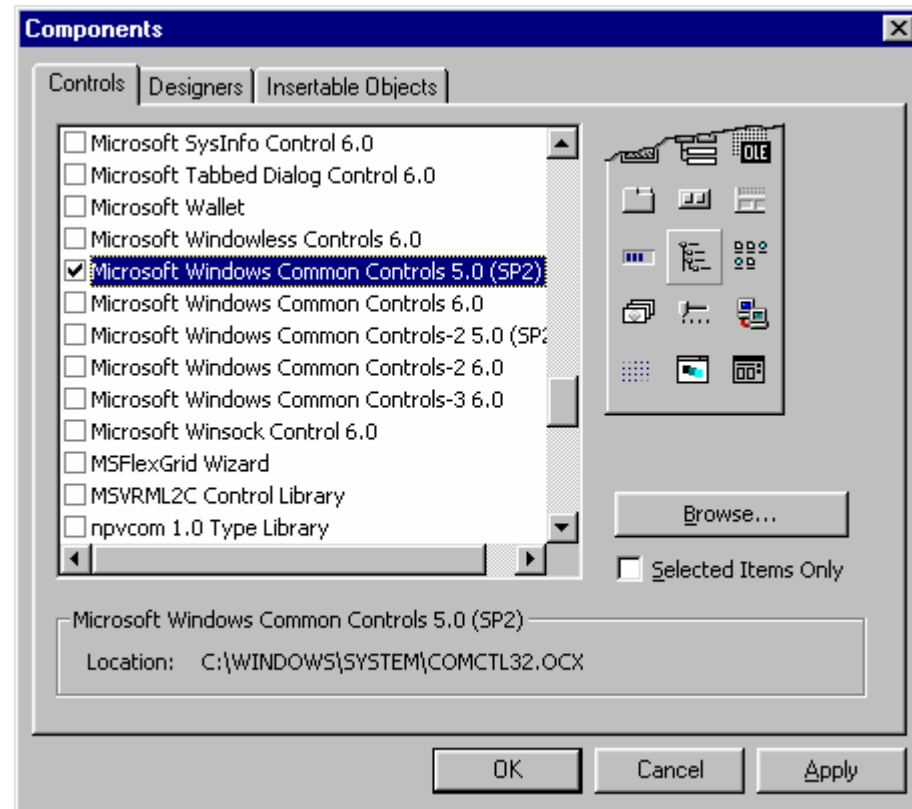
Option Boxes have **Value** property which is set to True when selected and False otherwise



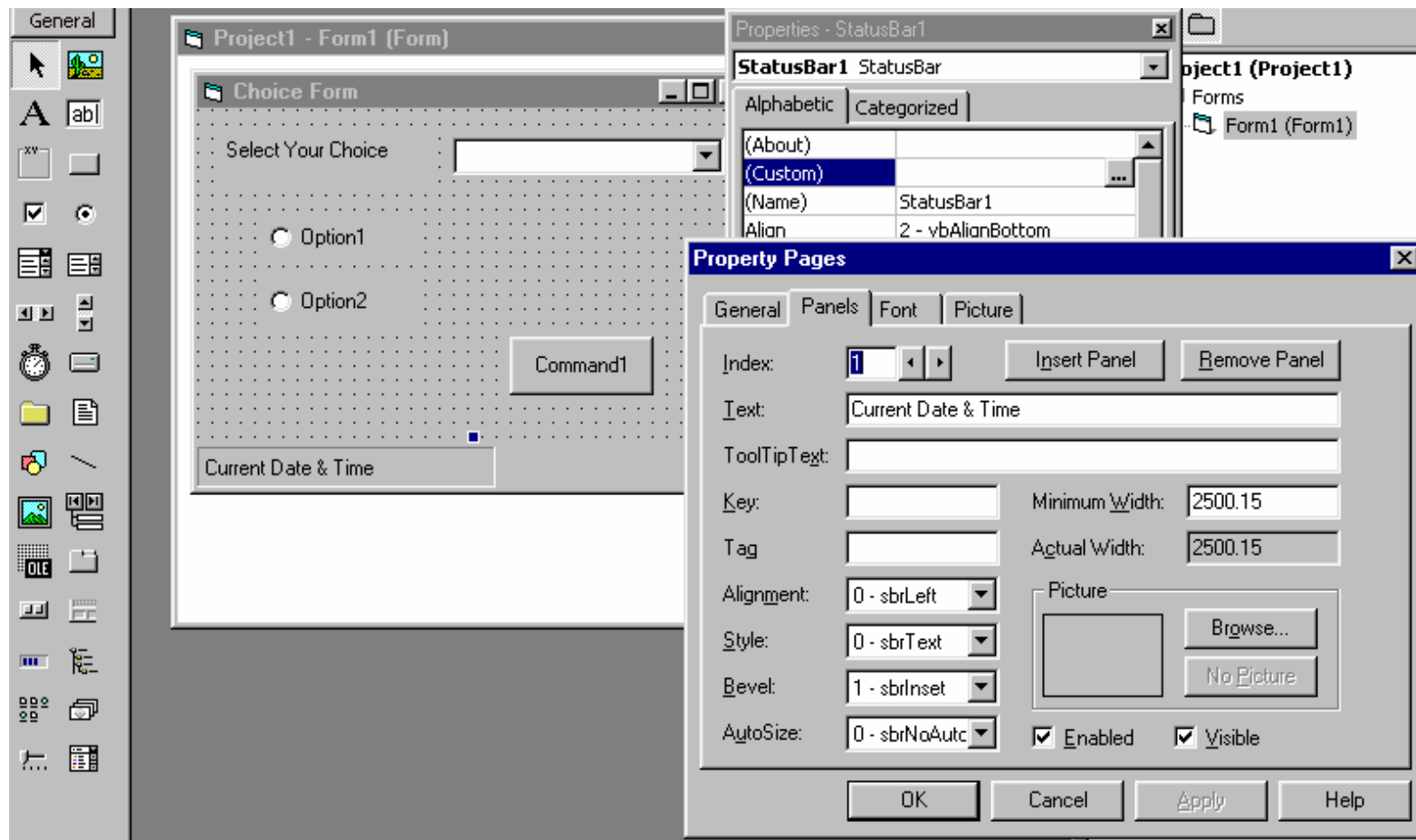
Status Bar

Status Bar display the program status information. You can use it to print current date-time and to print status messages

To place a status bar on a form, Select Project | Components and check Microsoft Windows Common Controls



Inserting Panels in Status Bar



Programming a Status Bar

The image displays the Visual Basic IDE with three windows:

- Project1 - Form1 (Code):** Shows the following code:

```
Private Sub Command1_Click()  
    MsgBox Option1.Value  
End Sub  
  
Private Sub Form_Load()  
    StatusBar1.Panels(2).Text = Now()  
End Sub
```
- Property Pages:** The 'Panels' tab is selected. The 'Index' is set to 2. The 'Text' field is empty. The 'Minimum Width' and 'Actual Width' are both set to 2000.12. The 'Enabled' and 'Visible' checkboxes are checked.
- Choice Form:** A form titled 'Choice Form' with a dropdown menu labeled 'Select Your Choice' and two radio buttons labeled 'Option1' and 'Option2'. The status bar at the bottom shows 'Current Date & Time' and '2/20/99 1:37:41 AM'.

Module IV

- Contents
 - Menu, pop up menus
 - File Handling
 - Debugging
 - Error Handling
 - Printing
 - Compilation

Menus, pop up menus

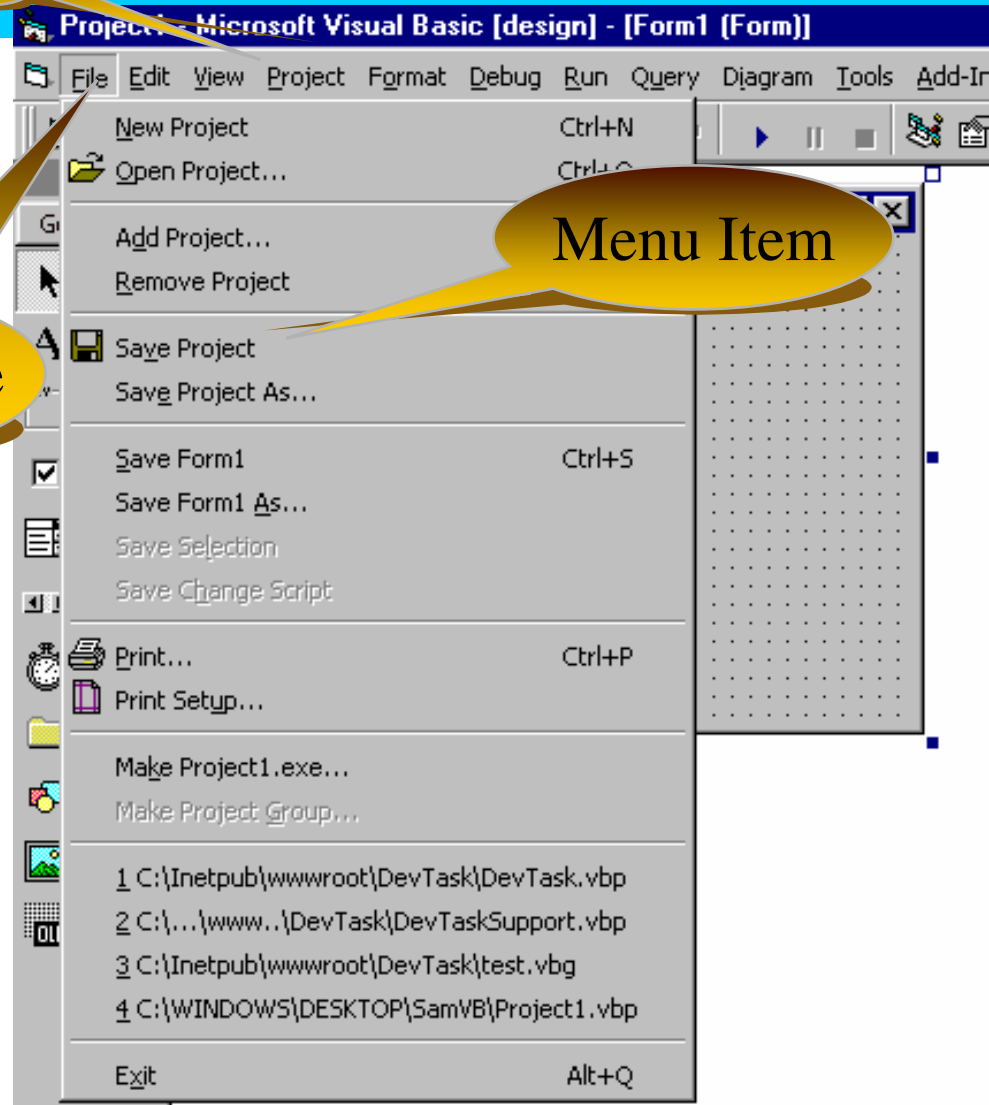
- To increase the functionality of a VB application, menu bars can provide the user with a simple way of controlling the program.
- Menu bar can be found at the top of a program.
- VB provides a Menu Editor to simplify creation of Menus
- Pop up menus are often referred to context sensitive menus. They are very specific to certain controls or areas of the application
- Menus can inform the user of the application's capabilities as well as inabilities.

Menu Terminology

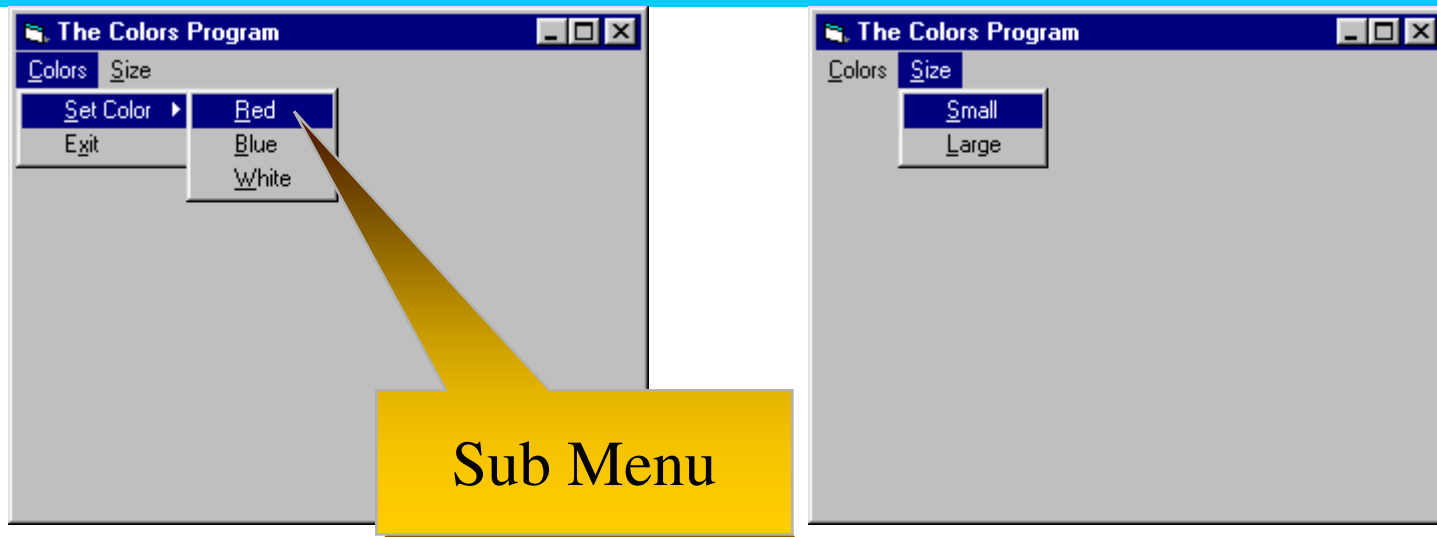
Menu Bar

Menu Title

Menu Item



The Colors Program



- Set Color sub menu will set the form background color to the user selected choice.
- Selecting Small from the Size menu will decrease the form window size and selecting Large from the Size menu will increase the form window

Using Menu Editor

Menu Editor

Caption: OK

Name: Cancel

Index: Shortcut: ▼

HelpContextID: NegotiatePosition: ▼

Checked Enabled Visible WindowList

← → ↑ ↓ Next Insert Delete

&Colors

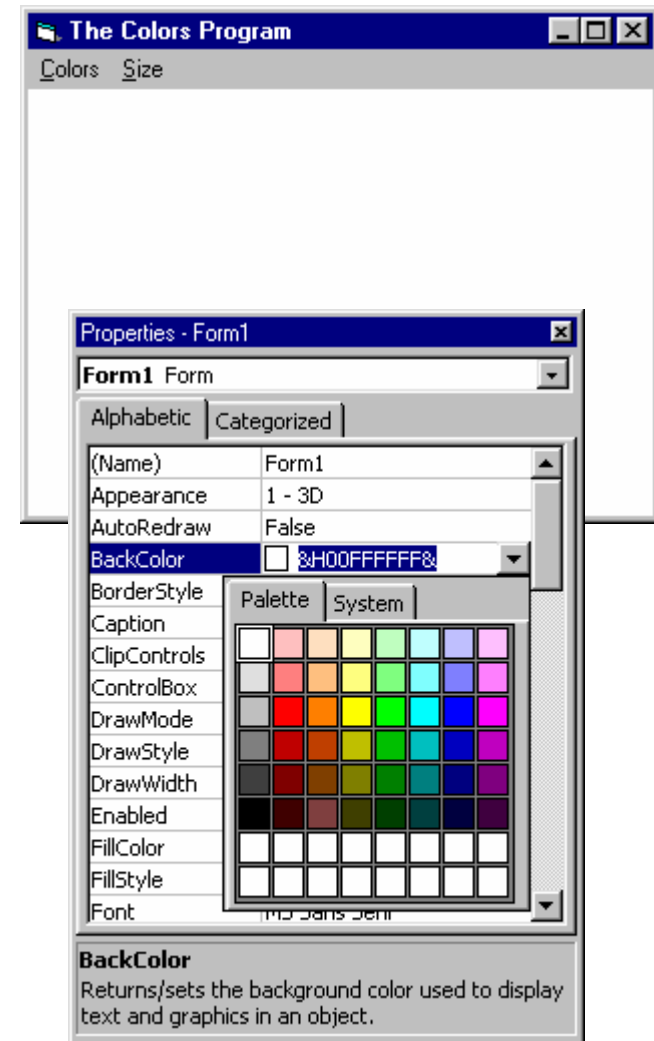
- ...&Set Color
-&Red
-&Blue
-&White
- ...E&xit
- &Size
- ...&Small
- ...&Large

Entering the code for The Colors Program

```
- Microsoft Visual Basic [design] - [Form1 (Code)]
View Project Format Debug Run Query Diagram Tools Add-Ins W
Form Load
Private Sub Form_Load()
    'Because the initial window is white
    'disable the White menu Item
    mnuWhite.Enabled = False

    'Because initially the window is small
    'disable the Small menu item
    mnuSmall.Enabled = False
End Sub
```

- Form when loaded will be initially having White Background
- Set the BackColor property of the Form to White Color



How the Colors Program Works

The image shows a Visual Basic IDE with two code windows and one application window. The application window, titled 'The Colors Program', has a red background and two menu items: 'Colors' and 'Size'. The code window for 'mnuRed' contains the following code:

```
Private Sub mnuRed_Click()  
    'Set the color of the form to Red  
    frmColors.BackColor = QBColor(4)  
  
    'Disable the Red menu Item  
    mnuRed.Enabled = False  
  
    'Enable the Blue and White menu items  
    mnuBlue.Enabled = True  
    mnuWhite.Enabled = True  
  
End Sub
```

The code window for 'mnuWhite' contains the following code:

```
Private Sub mnuBlue_Click()  
    'Set the color of the form to Blue  
    frmColors.BackColor = QBColor(1)  
  
    'Disable the Blue menu Item  
    mnuBlue.Enabled = False  
  
    'Enable the Red and White menu items  
    mnuRed.Enabled = True  
    mnuWhite.Enabled = True  
  
End Sub  
  
Private Sub mnuWhite_Click()  
    'Set the color of the form to White  
    frmColors.BackColor = QBColor(15)  
  
    'Disable the White menu Item  
    mnuWhite.Enabled = False  
  
    'Enable the Blue and Red menu items  
    mnuBlue.Enabled = True  
    mnuRed.Enabled = True  
  
End Sub  
  
Private Sub mnuExit_Click()  
    End  
End Sub
```

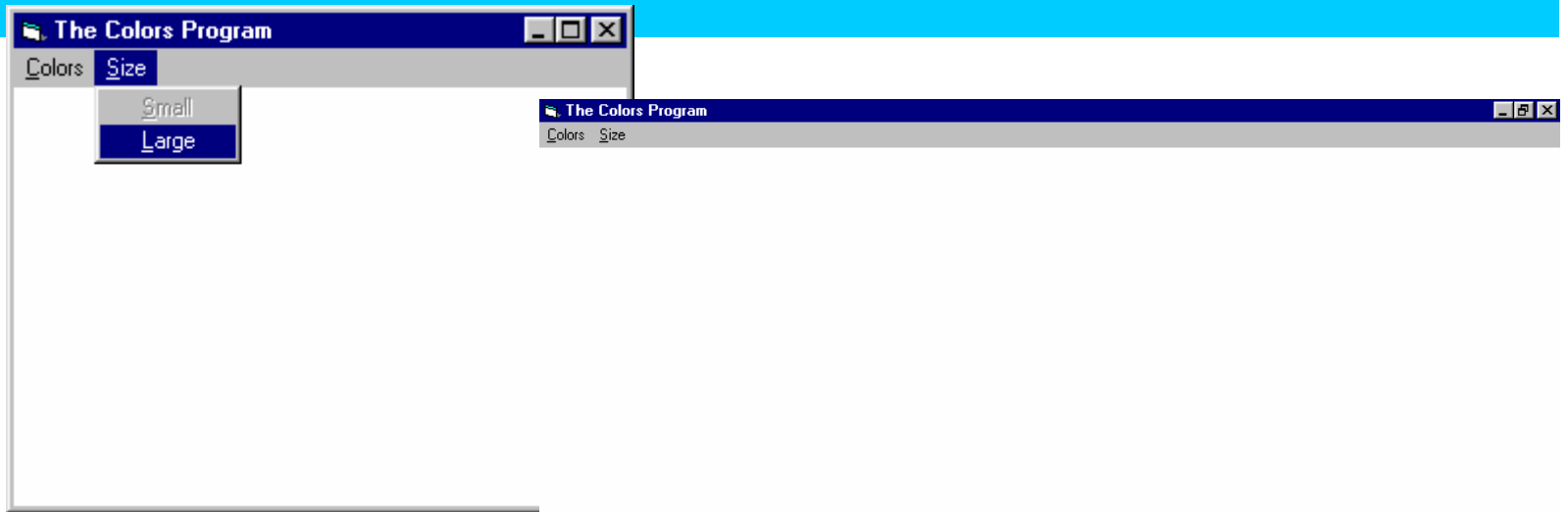
The code window for 'mnuExit' contains the following code:

```
Private Sub mnuExit_Click()  
    End  
End Sub
```

The application window shows the 'Colors' menu item selected, and the form background is red. The 'Size' menu item is also visible.

VB

Controlling the size of the form



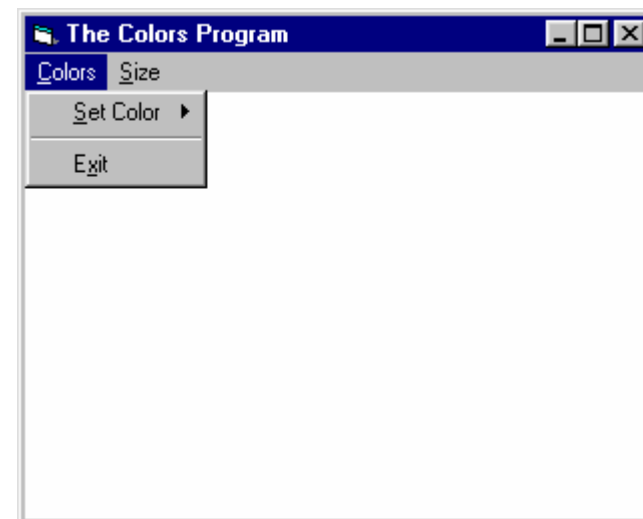
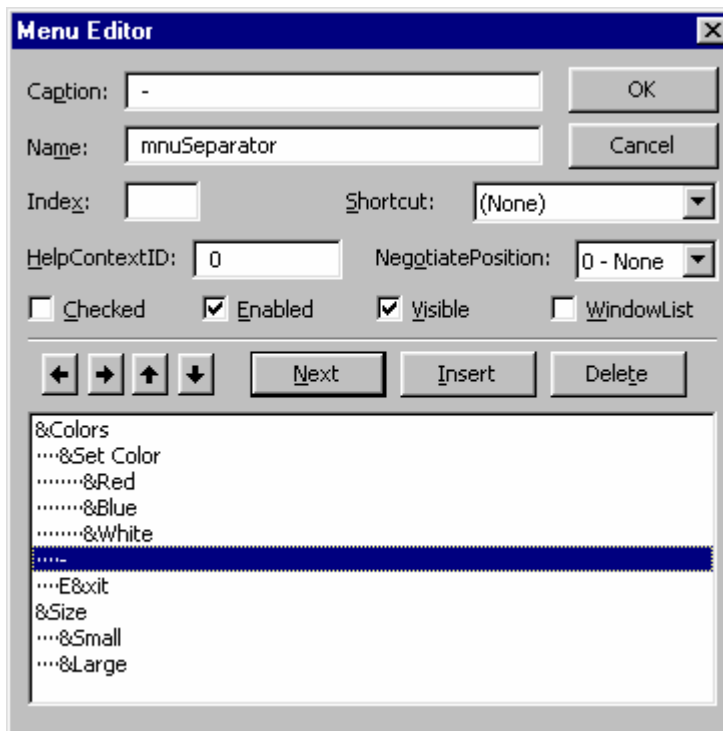
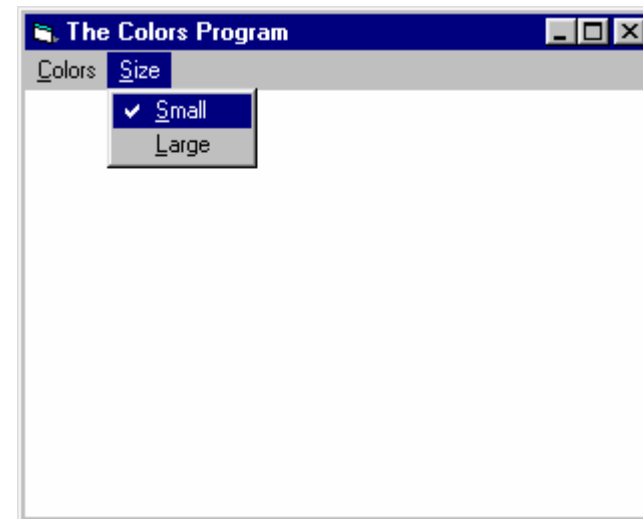
```
Private Sub mnuSmall_Click()  
    frmColors.WindowState = 0  
    mnuSmall.Enabled = False  
    mnuLarge.Enabled = True  
End Sub
```

```
Private Sub mnuLarge_Click()  
    frmColors.WindowState = 2  
    mnuSmall.Enabled = True  
    mnuLarge.Enabled = False  
End Sub
```

Other Menu Item Properties

```
Form Load
Private Sub Form_Load()
    'Because the initial window is white
    'disable the White menu Item
    mnuWhite.Enabled = False

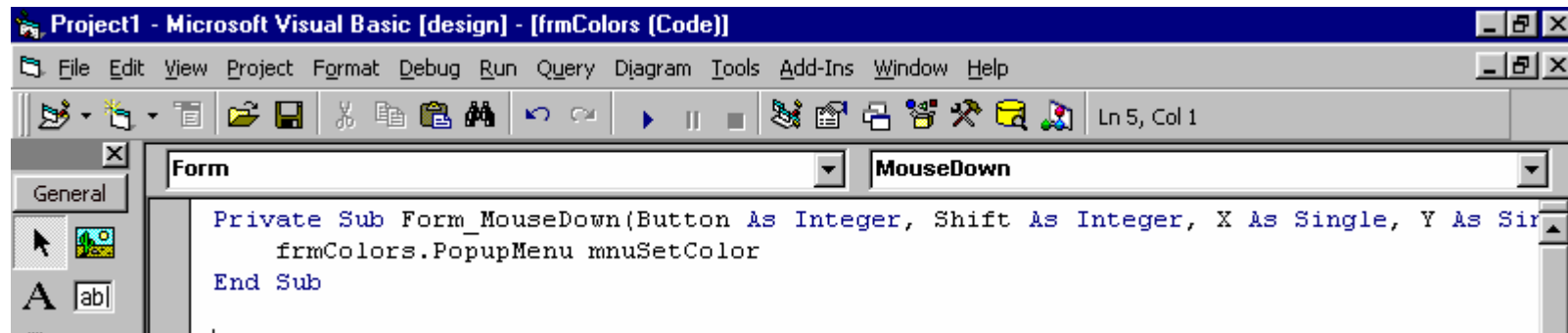
    'Because initially the window is small
    'disable the Small menu item
    mnuSmall.Checked = True
End Sub
```



Creating a Pop Up Menu

- 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 where the pointer was located when the right mouse button was pressed; therefore, pop-up menus are also called *context menus*.
- In Microsoft Windows 95/NT, you activate context menus by clicking the right mouse button.
- 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

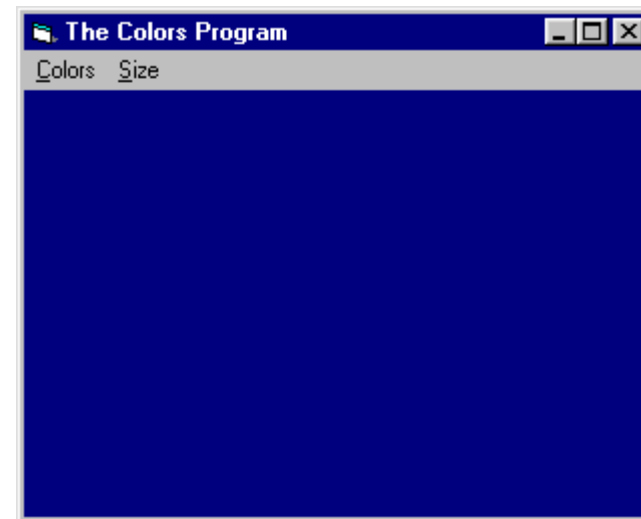
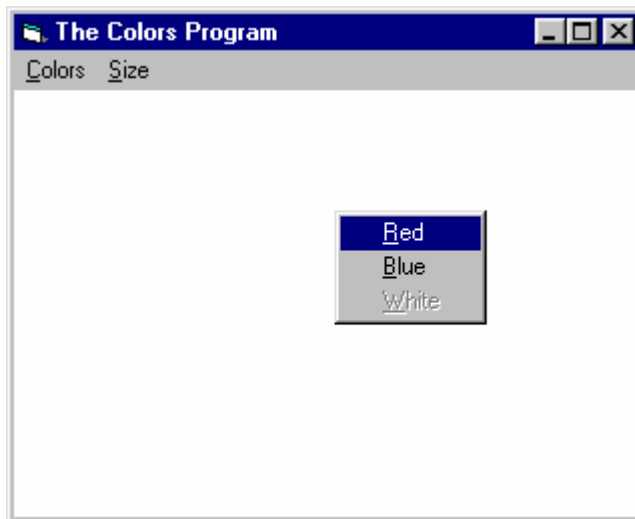
Attaching Pop Menu to an Object



The screenshot shows the Microsoft Visual Basic IDE with the following details:

- Window title: Project1 - Microsoft Visual Basic [design] - [frmColors (Code)]
- Menu bar: File, Edit, View, Project, Format, Debug, Run, Query, Diagram, Tools, Add-Ins, Window, Help
- Toolbar: Standard toolbar with icons for file operations and execution.
- Properties window: Shows 'Form' selected, with 'MouseDown' event attached.
- Code window: Contains the following code:

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    frmColors.PopupMenu mnuSetColor
End Sub
```



File Handling

- Open Statement
- Write# Statement
- Input# Statement
- Close Statement

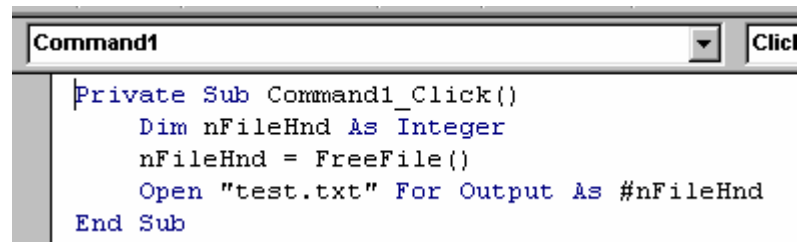
Open Statement

- Enables input/output (I/O) to a file

Syntax

pathname **For mode As** [#]*filename*

- *mode* Required. Keyword specifying the file mode: **Append**, **Binary**, **Input**, **Output**, or **Random**. If unspecified, the file is **Input** for **Random** access.
- *filename* Required. A valid file number in the range 1 to 511, inclusive. Use the **FreeFile** function to obtain the next available file number.



```
Private Sub Command1_Click()  
    Dim nFileHnd As Integer  
    nFileHnd = FreeFile()  
    Open "test.txt" For Output As #nFileHnd  
End Sub
```

Write# Statement

This example uses the Write# statement to write raw data to a sequential file.

```
Open "TESTFILE" For Output As #1 ' Open file for output.
    #1, "Hello World", 234 ' comma-delimited data.
    #1, ' blank line.

Dim MyBool, MyDate, MyNull, MyError
' Assign Boolean, Date, Null, and Error values.
MyBool = False : MyDate = #February 12, 1969# : MyNull = Null
MyError = CVErr(32767)
' Boolean data is written as #TRUE# or #FALSE#. Date literals are
' written in universal date format, for example, #1994-07-13#
' represents July 13, 1994. Null data is written as #NULL#.
' Error data is written as #ERROR errorcode#.
    #1, MyBool ; " is a Boolean value"
    #1, MyDate ; " is a date"
    #1, MyNull ; " is a null value"
    #1, MyError ; " is an error value"
Close #1 ' Close file.
```

Input# Statement

- Reads data from an open sequential file and assigns the data to variables

Input # Statement Example

This example uses the **Input #** statement to read data from a file into two variables. This example assumes that TESTFILE is a file with a few lines of data written to it using the **Write #** statement; that is, each line contains a string in quotations and a number separated by a comma, for example, ("Hello", 234).

```
Dim MyString, MyNumber
Open "TESTFILE" For Input As #1    ' Open file for input.
Do While Not EOF(1)               ' Loop until end of file.
    Input #1, MyString, MyNumber  ' Read data into two variables.
    Debug.Print MyString, MyNumber ' Print data to the Immediate
window.
Loop
Close #1    ' Close file.
```

Close Statement

- Concludes input/output (I/O) to a file opened using the **Open** statement

```
Dim I, FileName
For I = 1 To 3      ' Loop 3 times.
    FileName = "TEST" & I      ' Create file name.
    Open FileName For Output As #I      ' Open file.
    Print #I, "This is a test."      ' Write string to
    file.
Next I
Close      ' Close all 3 open files.
```

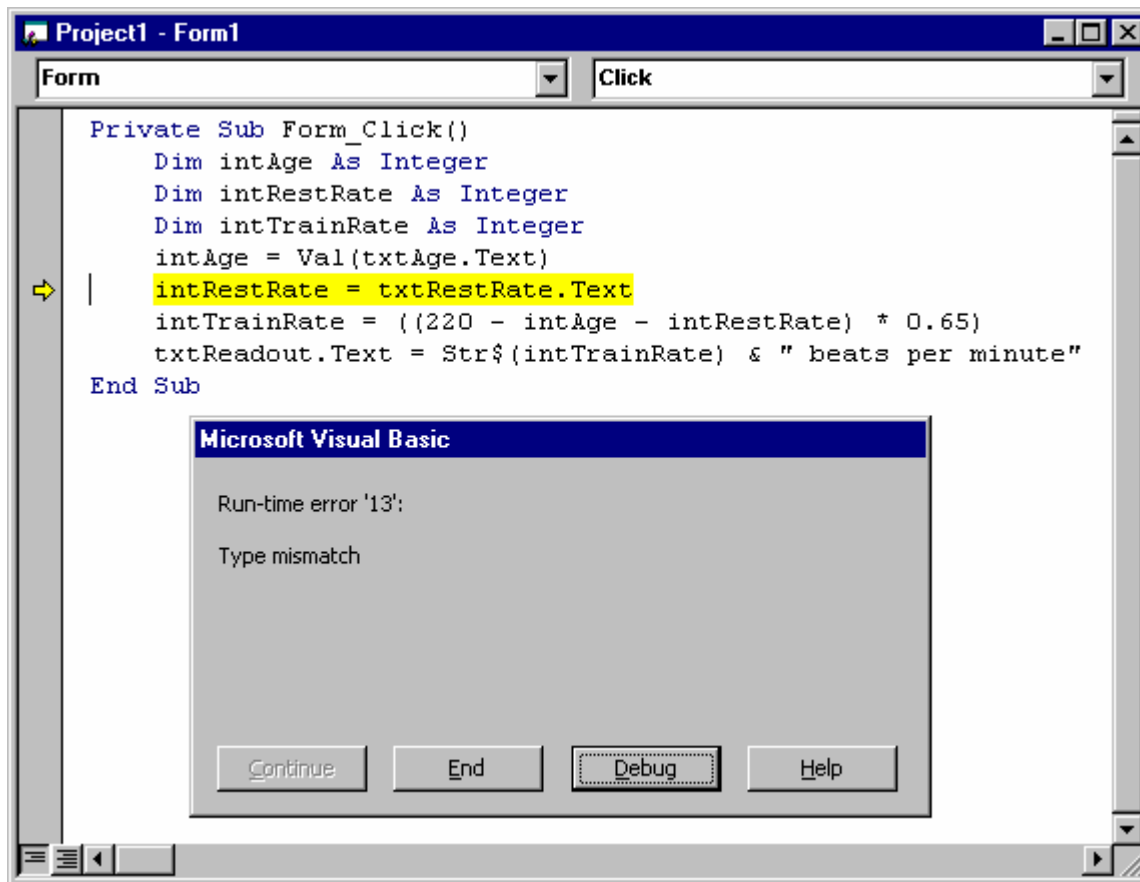
- **Close#** <FileHandle> close one specific file.

Debugging Tools

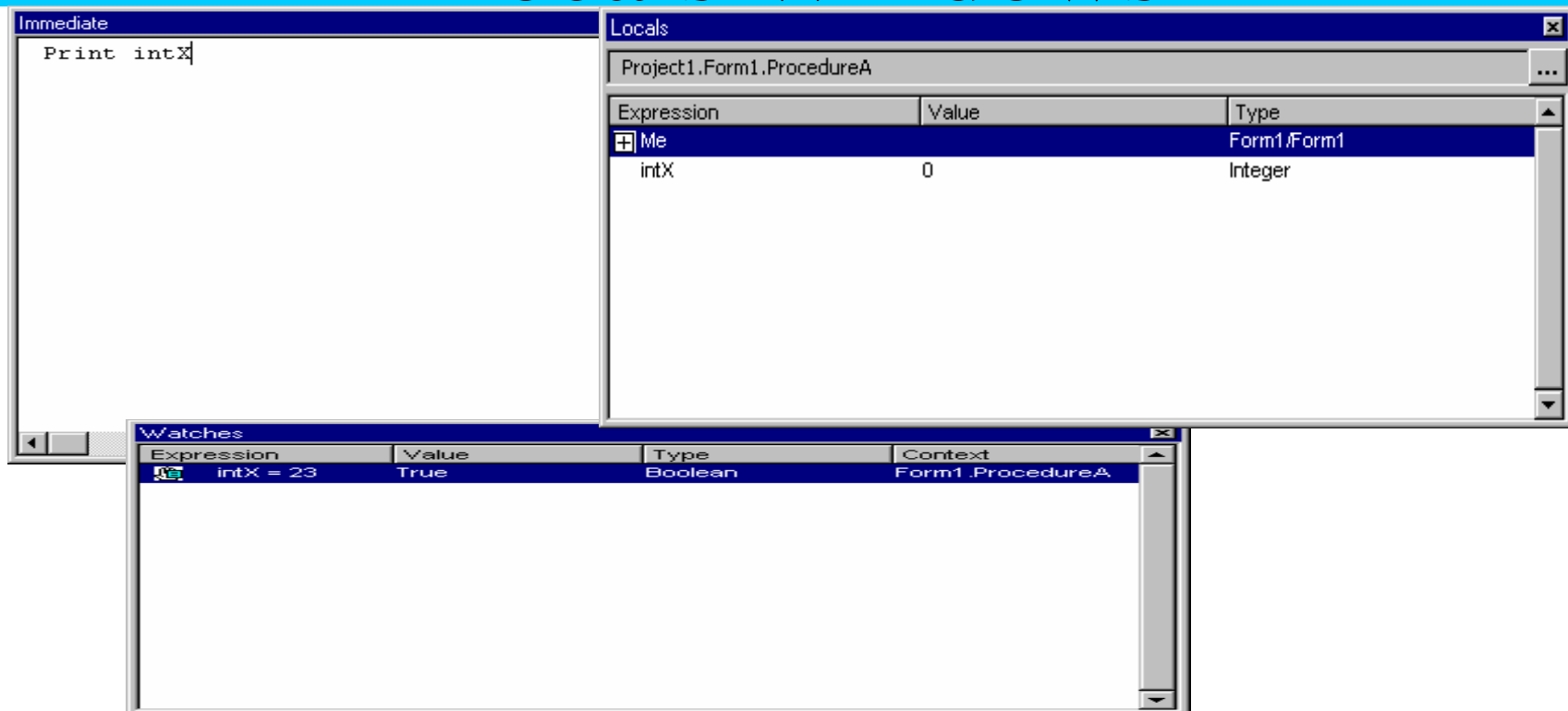
– Using the Break Mode

- You can enter break mode manually if you do any of the following while the application is running:
 - Press CTRL+BREAK.
 - Choose Break from the Run menu.
 - Click the Break button on the toolbar.
- You can also enter break mode automatically when any of the following occurs:
 - A statement generates an untrapped run-time error.
 - A statement generates a run-time error and the Break on All Errors error trapping option has been selected.
 - A break expression defined in the Add Watch dialog box changes or becomes true, depending on how you defined it.
 - Execution reaches a line with a breakpoint.
 - Execution reaches a Stop statement

Fixing Run Time Errors and Continuing



Using the Immediate, Watch and Locals Windows



- The *Immediate window* shows information that results from debugging statements in your code, or that you request by typing commands directly into the window.
- The *Watch window* shows the current *watch expressions*, which are expressions whose values you decide to monitor as the code runs
- The *Locals window* shows the value of any variables within the scope of the current procedure. As the execution switches from procedure to procedure, the contents of the Locals window changes to reflect only the variables applicable to the current procedure.

Error Handling

- On Error Statement
 - Enables error-handling.
- Example

```
Sub Command1_Click()  
    On Error Goto ErrHandler  
    Err.Raise 6 'Raise an overflow error.  
    MsgBox "Program Continues..."  
    ...  
    Exit Sub  
ErrHandler:  
    MsgBox ("Error # " & CStr(Err.Number) & " " & _  
    _Err.Description)  
    Err.Clear ' Clear the error.  
    Resume Next  
End Sub
```

Printing with Printer Object

- Put text and graphics on the Printer object.
- Print the contents of the Printer object with the `NewPage` or `EndDoc` method.

```
For pageno = 1 To 4
Printer.PrintQuality = -1 * pageno
Printer.Print "The quality of this
page is"; pageno
Printer.NewPage
Next
```

Positioning Text and Graphics

```
Printer.CurrentX = 0
Printer.CurrentY = 0
```

Printing (contd.)

- Once you have placed text and graphics on the Printer object, use the EndDoc method to print the contents. The EndDoc method advances the page and sends all pending output to the spooler. A *spooler* intercepts a print job on its way to the printer and sends it to disk or memory, where the print job is held until the printer is ready for it. For example:

```
Printer.Print "This is the first line of text in _  
a pair."  
Printer.Print "This is the second line of text in  
_ a pair."  
Printer.EndDoc
```

Note Visual Basic automatically calls *EndDoc* if your application ends without explicitly calling it.

Creating Multi-Page Documents

- When printing longer documents, you can specify in code where you want a new page to begin by using the `NewPage` method. For example:

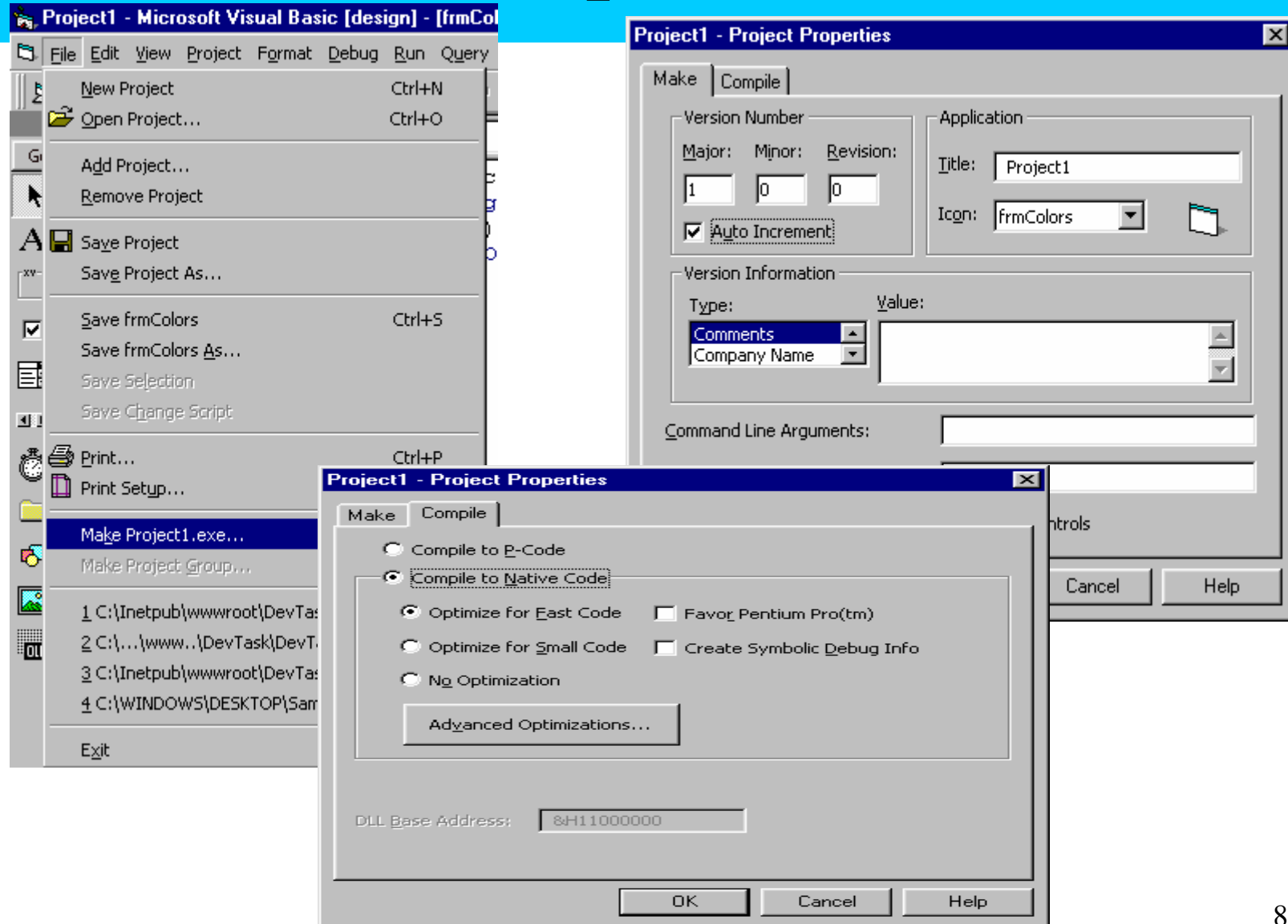
```
Printer.Print "This is page 1."  
Printer.NewPage  
Printer.Print "This is page 2."  
Printer.EndDoc
```

Canceling a Print Job

- You can terminate the current print job by using the KillDoc method. For example, you can query the user with a dialog box to determine whether to print or terminate a document:

```
Sub PrintOrNot()  
    Printer.Print "This is the first line to _  
        illustrate KillDoc method"  
    Printer.Print "This is the second line to _  
        illustrate KillDoc method"  
    Printer.Print "This is the third line to _  
        illustrate KillDoc method"  
    If vbNo = MsgBox("Print this fine document?", _  
        vbYesNo) Then  
        Printer.KillDoc  
    Else  
        Printer.EndDoc  
    End If  
End Sub
```

Compilation



Course Review

- Introduction to Programming
- Introduction to Object Oriented Programming
- Program Constructs
- Data Types
- Forms, Code and Files
- List Boxes and Option Buttons
- ToolBars, Status Bars
- Menu, pop up menus
- File Handling
- Debugging
- Error Handling
- Printing
- Compilation

Next Course: Visual Basic Intermediate to Advanced

- Contents

- Quick Review of Basic Concepts
- Using Advanced OCX controls (TabStrips, Grids, TreeView, Common Dialog Controls)
- Graphics Programming
- Developing Classes
- Data Manipulation (Working with Databases)
- Data Bound Controls
- Creating Dynamic Menus /Advanced Toolbars
- Crystal Reports
- OLE (Object Linking and Embedding) (*Integrating with Word / Excel*)
- Client/Server Programming
- Building ActiveX Controls (Dynamic Link Libraries DLLs)
- Creating Add-Ins
- Customizing Projects