Course Notes for:

Learn Visual Basic 6.0

© Lou Tylee, 1998

E-Mail: KIDware@jetcity.com
http://www.jetcity.com/~kidware

15600 NE 8th, Suite B1-314
Bellevue, WA 98008
(206) 721-2556
FAX (425) 746-4655

Notice

Learn Visual Basic 6.0

Contents

Learn Visual Basic 6.0

1. Introduction to the Visual Basic Language and Environment

Preview

- In this first class, we will do a quick overview of how to build an application in Visual Basic. You l learn a new vocabulary, a new approach to programming, and ways to move around in the Visual Basic environment. You will leave having written your first Visual Basic program.

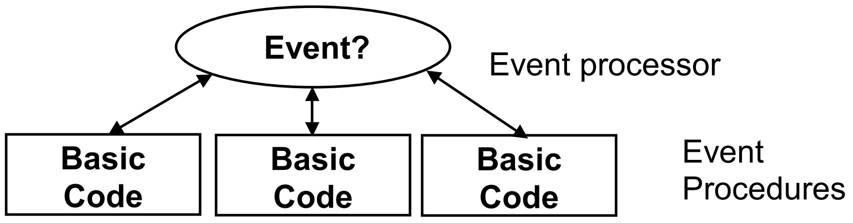
Course Objectives

⇒ Understand the benefits of using Microsoft Visual Basic 6.0 for Windows as an application tool
⇒ Understand the Visual Basic event-driven programming concepts, terminology, and available tools
⇒ Learn the fundamentals of designing, implementing, and distributing a Visual Basic application
⇒ Learn to use the Visual Basic toolbox
⇒ Learn to modify object properties
⇒ Learn object methods
⇒ Use the menu design window
⇒ Understand proper debugging and error-handling procedures
⇒ Gain a basic understanding of database access and management using databound controls
⇒ Obtain an introduction to ActiveX controls and the Windows Application Programming Interface (API)

What is Visual Basic?

- **Visual Basic** is a tool that allows you to develop Windows (Graphic User Interface - **GUI**) applications. The applications have a familiar appearance to the user.

- Visual Basic is **event-driven**; meaning code remains idle until called upon to respond to some event (button pressing, menu selection,). An event processor governs Visual Basic. Nothing happens until an event is detected. Once an event is detected, the code corresponding to that event (event procedure) is executed. Program control is then returned to the event processor.



- Some Features of Visual Basic

  ⇒ Full set of objects - you 'draw' the application
  ⇒ Lots of icons and pictures for your use
  ⇒ Response to mouse and keyboard actions
  ⇒ Clipboard and printer access
  ⇒ Full array of mathematical, string handling, and graphics functions
  ⇒ Can handle fixed and dynamic variable and control arrays
  ⇒ Sequential and random access file support
  ⇒ Useful debugger and error-handling facilities
  ⇒ Powerful database access tools
  ⇒ ActiveX support
  ⇒ Package & Deployment Wizard makes distributing your applications simple

Visual Basic 6.0 versus Other Versions of Visual Basic

- The original Visual Basic for DOS and Visual Basic For Windows were introduced in 1991.

- Visual Basic 3.0 (a vast improvement over previous versions) was released in 1993.

- Visual Basic 4.0 released in late 1995 (added 32-bit application support).
-
- Visual Basic 5.0 released in late 1996. New environment, supported creation of ActiveX controls, deleted 16-bit application support.

- And, now Visual Basic 6.0 - some identified new features of Visual Basic 6.0:

  ⇒ Faster compiler
  ⇒ New ActiveX data control object
  ⇒ Allows database integration with wide variety of applications
  ⇒ New data report designer
  ⇒ New Package & Deployment Wizard
  ⇒ Additional internet capabilities

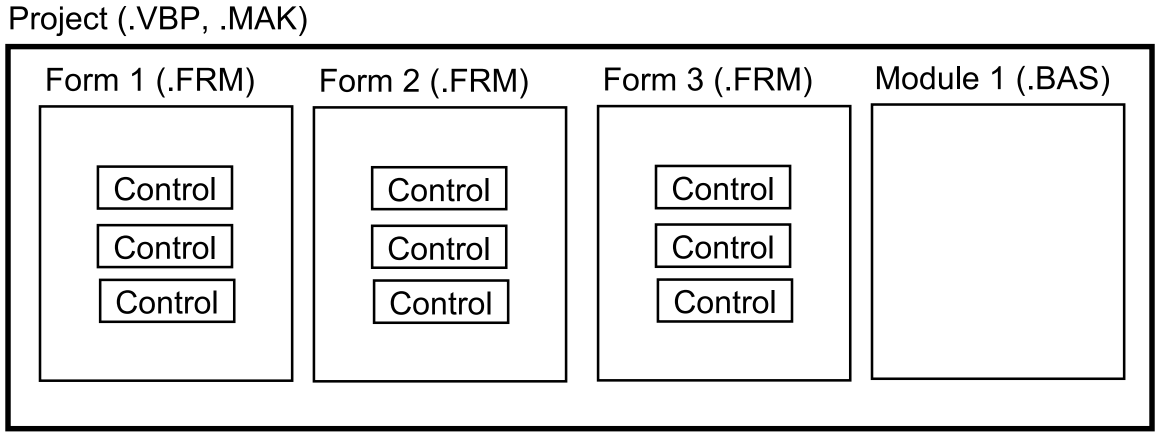
16 Bits versus 32 Bits

- Applications built using the Visual Basic 3.0 and the 16 bit version of Visual Basic 4.0 will run under Windows 3.1, Windows for Workgroups, Windows NT, or Windows 95

- Applications built using the 32-bit version of Visual Basic 4.0, Visual Basic 5.0 and Visual Basic 6.0 will only run with Windows 95 or Windows NT (Version 3.5.1 or higher).

- In this class, we will use Visual Basic 6.0 under Windows 95, recognizing such applications will not operate in 16 bit environments.

Structure of a Visual Basic Application



Application (Project) is made up of:

⇒ **Forms** - Windows that you create for user interface
⇒ **Controls** - Graphical features drawn on forms to allow user interaction (text boxes, labels, scroll bars, command buttons, etc.) (Forms and Controls are **objects**.)
⇒ **Properties** - Every characteristic of a form or control is specified by a property. Example properties include names, captions, size, color, position, and contents. Visual Basic applies default properties. You can change properties at design time or run time.
⇒ **Methods** - Built-in procedure that can be invoked to impart some action to a particular object.
⇒ **Event Procedures** - Code related to some object. This is the code that is executed when a certain event occurs.
⇒ **General Procedures** - Code not related to objects. This code must be invoked by the application.
⇒ **Modules** - Collection of general procedures, variable declarations, and constant definitions used by application.

Steps in Developing Application

- There are three primary steps involved in building a Visual Basic application:

  1. **Draw** the user **interface**
  2. **Assign properties** to controls
  3. **Attach code** to controls

  We I look at each step.

Drawing the User Interface and Setting Properties

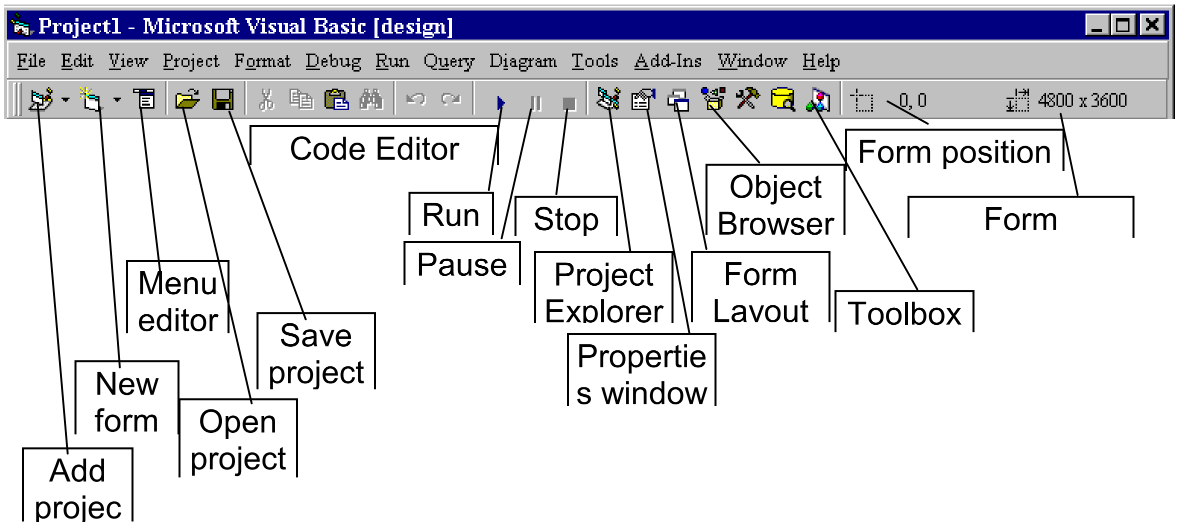
- Visual Basic operates in three modes.

  ⇒ **Design** mode - used to build application
  ⇒ **Run** mode - used to run the application
  ⇒ **Break** mode - application halted and debugger is available

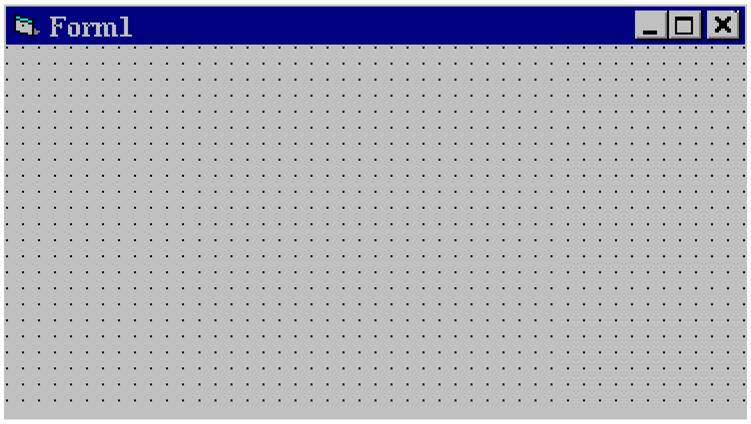
  We focus here on the **design** mode.

- Six windows appear when you start Visual Basic.

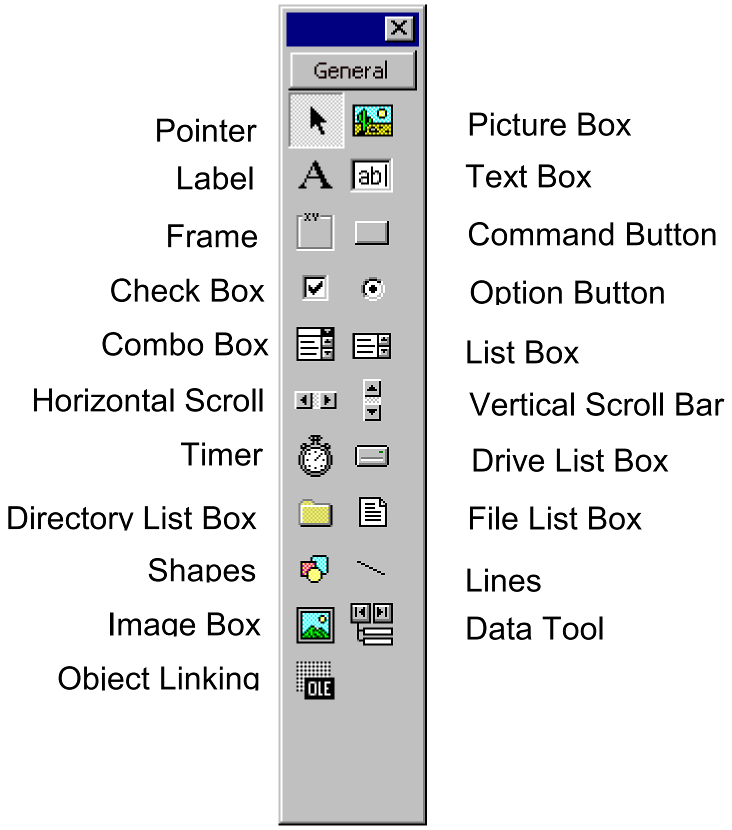
  ⇒ The **Main Window** consists of the title bar, menu bar, and toolbar. The title bar indicates the project name, the current Visual Basic operating mode, and the current form. The menu bar has drop-down menus from which you control the operation of the Visual Basic environment. The toolbar has buttons that provide shortcuts to some of the menu options. The main window also shows the location of the current form relative to the upper left corner of the screen (measured in twips) and the width and length of the current form.



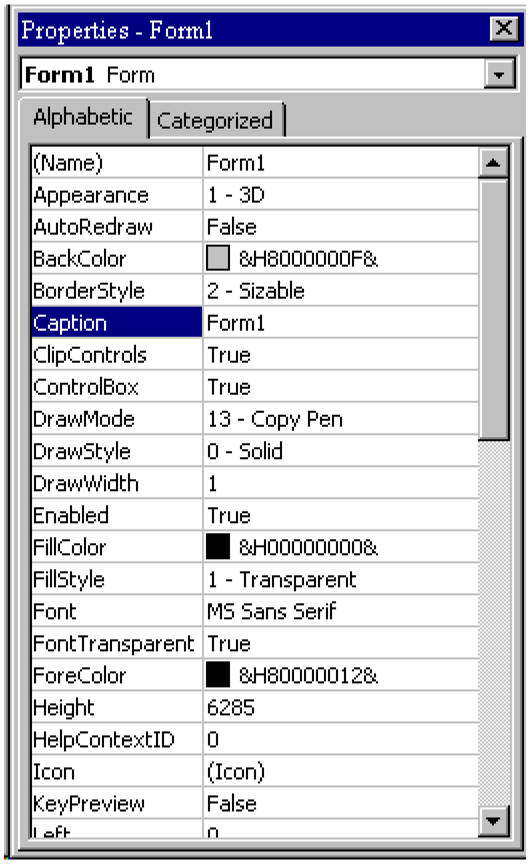
⇒ The **Form Window** is central to developing Visual Basic applications. It is where you draw your application.



⇒ The **Toolbox** is the selection menu for controls used in your application.

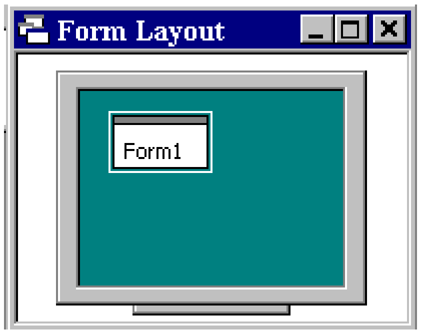


⇒ The **Properties Window** is used to establish initial property values for objects. The drop-down box at the top of the window lists all objects in the current form. Two views are available: Alphabetic and Categorized. Under this box are the available properties for the currently selected object.

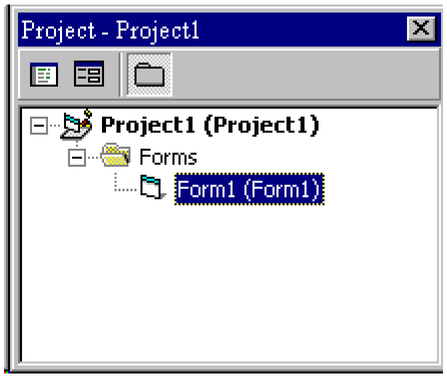


(Name)	Form1
Appearance	1 - 3D
AutoRedraw	False
BackColor	⬜ &H8000000F&
BorderStyle	2 - Sizable
Caption	Form1
ClipControls	True
ControlBox	True
DrawMode	13 - Copy Pen
DrawStyle	0 - Solid
DrawWidth	1
Enabled	True
FillColor	⬛ &H00000000&
FillStyle	1 - Transparent
Font	MS Sans Serif
FontTransparent	True
ForeColor	⬛ &H80000012&
Height	6285
HelpContextID	0
Icon	(Icon)
KeyPreview	False

⇒ The **Form Layout Window** shows where (upon program execution) your form will be displayed relative to your monitor screen:



⇒ The **Project Window** displays a list of all forms and modules making up your application. You can also obtain a view of the **Form** or **Code** windows (window containing the actual Basic coding) from the Project window.



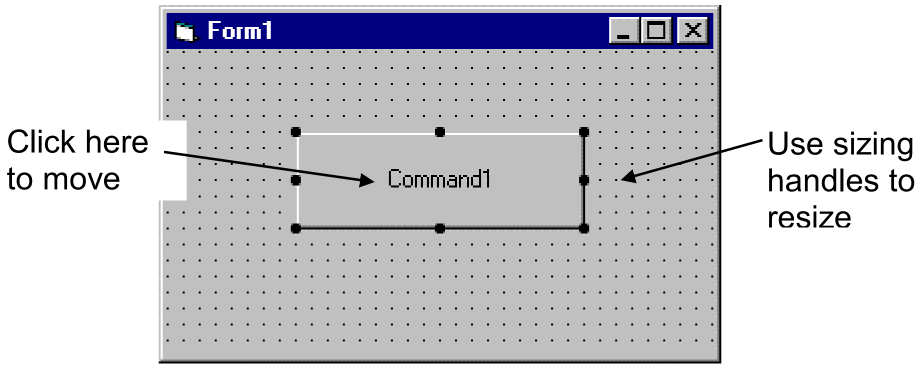
- As mentioned, the user interface is rawn' in the form window. There are two ways to place controls on a form:

  1. Double-click the tool in the toolbox and it is created with a default size on the form. You can then move it or resize it.

  2. Click the tool in the toolbox, then move the mouse pointer to the form window. The cursor changes to a crosshair. Place the crosshair at the upper left corner of where you want the control to be, press the left mouse button and hold it down while dragging the cursor toward the lower right corner. When you release the mouse button, the control is drawn.

- To **move** a control you have drawn, click the object in the form window and drag it to the new location. Release the mouse button.

- To **resize** a control, click the object so that it is select and sizing handles appear. Use these handles to resize the object.

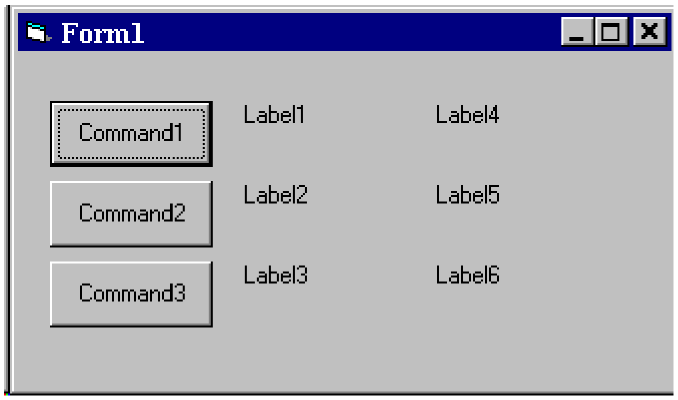


Example 1-1

Stopwatch Application - Drawing Controls

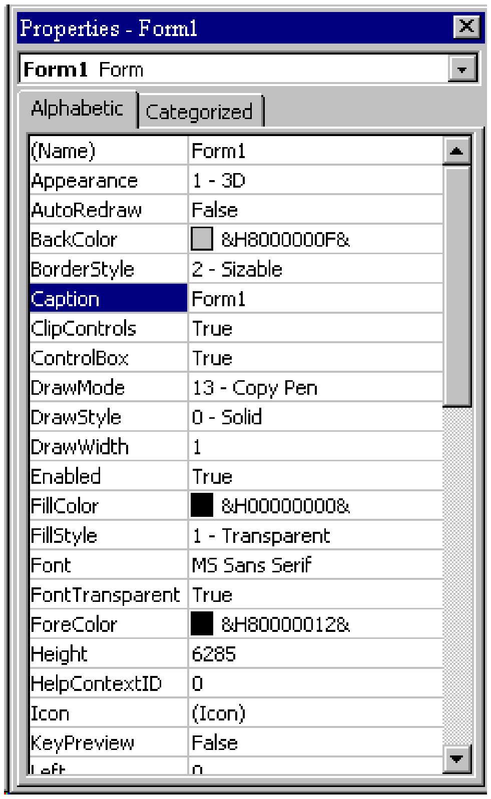
1. Start a new project. The idea of this project is to start a timer, then stop the timer and compute the elapsed time (in seconds).

2. Place three command buttons and six labels on the form. Move and size the controls and form so it looks something like this:



Setting Properties of Objects at Design Time

- Each form and control has **properties** assigned to it by default when you start up a new project. There are two ways to display the properties of an object. The first way is to click on the object (form or control) in the form window. Then, click on the Properties Window or the Properties Window button in the tool bar. The second way is to first click on the Properties Window. Then, select the object from the **Object** box in the Properties Window. Shown is the Properties Window for the stopwatch application:



(Name)	Form1
Appearance	1 - 3D
AutoRedraw	False
BackColor	⬜ &H8000000F&
BorderStyle	2 - Sizable
Caption	Form1
ClipControls	True
ControlBox	True
DrawMode	13 - Copy Pen
DrawStyle	0 - Solid
DrawWidth	1
Enabled	True
FillColor	⬛ &H00000000&
FillStyle	1 - Transparent
Font	MS Sans Serif
FontTransparent	True
ForeColor	⬛ &H80000012&
Height	6285
HelpContextID	0
Icon	(Icon)
KeyPreview	False
Left	0

The drop-down box at the top of the Properties Window is the **Object** box. It displays the name of each object in the application as well as its type. This display shows the **Form** object. The **Properties** list is directly below this box. In this list, you can scroll through the list of properties for the selected object. You may select a property by clicking on it. Properties can be changed by typing a new value or choosing from a list of predefined settings (available as a drop down list). Properties can be viewed in two ways: **Alphabetic** and **Categorized**.

A very important property for each object is its **name**. The name is used by Visual Basic to refer to a particular object in code.

- A convention has been established for naming Visual Basic objects. This convention is to use a three-letter prefix (depending on the object) followed by a name you assign. A few of the prefixes are (we I see more as we progress in the class):

Object	**Prefix**	**Example**
Form	frm	frmWatch
Command Button	cmd, btn	cmdExit, btnStart
Label	lbl	lblStart, lblEnd
Text Box	txt	txtTime, txtName
Menu	mnu	mnuExit, mnuSave
Check box	chk	chkChoice

- Object names can be up to 40 characters long, must start with a letter, must contain only letters, numbers, and the underscore (_) character. Names are used in setting properties at run time and also in establishing procedure names for object events.

Setting Properties at Run Time

- You can also set or modify properties while your application is running. To do this, you must write some code. The code format is:

  ```
  ObjectName.Property = NewValue
  ```

  Such a format is referred to as dot notation. For example, to change the **BackColor** property of a form name **frmStart**, we'd type:

  ```
  frmStart.BackColor = BLUE
  ```

How Names are Used in Object Events

- The names you assign to objects are used by Visual Basic to set up a framework of event-driven procedures for you to add code to. The format for each of these subroutines (all object procedures in Visual Basic are subroutines) is:

  ```
  Sub ObjectName_Event (Optional Arguments)
          .
          .
  End Sub
  ```

- Visual Basic provides the **Sub** line with its arguments (if any) and the **End Sub** statement. You provide any needed code.

Example 1-2

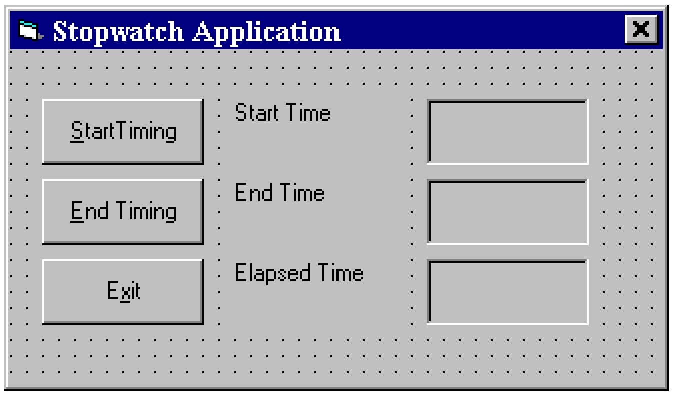
Stopwatch Application - Setting Properties

1. Set properties of the form, three buttons, and six labels:

Form1:		
	BorderStyle	1-Fixed Single
	Caption	Stopwatch Application
	Name	frmStopWatch
Command1:		
	Caption	&Start Timing
	Name	cmdStart
Command2:		
	Caption	&End Timing
	Name	cmdEnd
Command3:		
	Caption	E&xit
	Name	cmdExit
Label1:		
	Caption	Start Time
Label2:		
	Caption	End Time
Label3:		
	Caption	Elapsed Time
Label4:		
	BorderStyle	1-Fixed Single
	Caption	[Blank]
	Name	lblStart
Label5:		
	BorderStyle	1-Fixed Single
	Caption	[Blank]
	Name	lblEnd
Label6:		
	BorderStyle	1-Fixed Single
	Caption	[Blank]
	Name	lblElapsed

In the **Caption** properties of the three command buttons, notice the ampersand (**&**). The ampersand precedes a button's **access key**. That is, in addition to clicking on a button to invoke its event, you can also press its access key (no need for a mouse). The access key is pressed in conjunction with the **Alt** key. Hence, to invoke 'Begin Timing', you can either click the button or press Alt+B. Note in the button captions on the form, the access keys appear with an underscore (_).

2. Your form should now look something like this:



Variables

- We   e now ready to attach code to our application. As objects are added to the form, Visual Basic automatically builds a framework of all event procedures. We simply add code to the event procedures we want our application to respond to. But before we do this, we need to discuss **variables**.

- Variables are used by Visual Basic to hold information needed by your application. Rules used in naming variables:

  ⇒ No more than 40 characters
  ⇒ They may include letters, numbers, and underscore (_)
  ⇒ The first character must be a letter
  ⇒ You cannot use a reserved word (word needed by Visual Basic)

Visual Basic Data Types

Data Type	**Suffix**
Boolean	None
Integer	%
Long (Integer)	&
Single (Floating)	!
Double (Floating)	#
Currency	@
Date	None
Object	None
String	$
Variant	None

Variable Declaration

- There are three ways for a variable to be typed (declared):

  1. Default
  2. Implicit
  3. Explicit

- If variables are not implicitly or explicitly typed, they are assigned the **variant** type by **default**. The variant data type is a special type used by Visual Basic that can contain numeric, string, or date data.

- To **implicitly** type a variable, use the corresponding suffix shown above in the data type table. For example,

  ```
  TextValue$ = "This is a string"
  ```

  creates a string variable, while

  ```
  Amount% = 300
  ```

  creates an integer variable.

- There are many advantages to **explicitly** typing variables. Primarily, we insure all computations are properly done, mistyped variable names are easily spotted, and Visual Basic will take care of insuring consistency in upper and lower case letters used in variable names. Because of these advantages, and because it is good programming practice, we will explicitly type all variables.

- To **explicitly** type a variable, you must first determine its **scope**. There are four levels of scope:

  ⇒ Procedure level
  ⇒ Procedure level, static
  ⇒ Form and module level
  ⇒ Global level

- Within a procedure, variables are declared using the **Dim** statement:

  ```
  Dim MyInt as Integer
  Dim MyDouble as Double
  Dim MyString, YourString as String
  ```

  Procedure level variables declared in this manner do not retain their value once a procedure terminates.

- To make a procedure level variable retain its value upon exiting the procedure, replace the Dim keyword with **Static**:

  ```
  Static MyInt as Integer
  Static MyDouble as Double
  ```

- Form (module) level variables retain their value and are available to all procedures within that form (module). Form (module) level variables are declared in the **declarations** part of the **general** object in the form's (module's) code window. The **Dim** keyword is used:

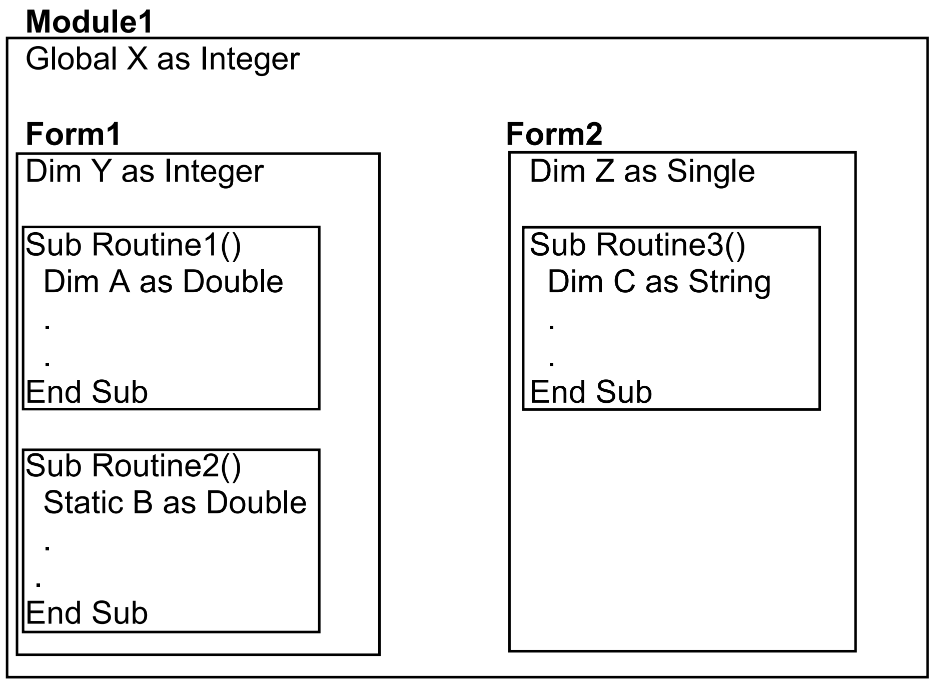
```
Dim MyInt as Integer
Dim MyDate as Date
```

- Global level variables retain their value and are available to all procedures within an application. Module level variables are declared in the **declarations** part of the **general** object of a module's code window. (It is advisable to keep all global variables in one module.) Use the **Global** keyword:

```
Global MyInt as Integer
Global MyDate as Date
```

- What happens if you declare a variable with the same name in two or more places? More local variables **shadow** (are accessed in preference to) less local variables. For example, if a variable MyInt is defined as Global in a module and declared local in a routine MyRoutine, while in MyRoutine, the local value of MyInt is accessed. Outside MyRoutine, the global value of MyInt is accessed.

- Example of Variable Scope:



```
Module1
Global X as Integer

Form1
Dim Y as Integer

Sub Routine1()
  Dim A as Double
  .
  .
End Sub

Sub Routine2()
  Static B as Double
  .
 .
End Sub

Form2
 Dim Z as Single

Sub Routine3()
  Dim C as String
  .
  .
End Sub
```

Procedure Routine1 has access to X, Y, and A (loses value upon termination)
Procedure Routine2 has access to X, Y, and B (retains value)
Procedure Routine3 has access to X, Z, and C (loses value)

Example 1-3

Stopwatch Application - Attaching Code

All that left to do is attach code to the application. We write code for every event a response is needed for. In this application, there are three such events: clicking on each of the command buttons.

1. Double-click anywhere on the form to open the code window. Or, select iew Code' from the project window.

2. Click the down arrow in the Object box and select the object named **(general)**. The Procedure box will show **(declarations)**. Here, you declare three form level variables:

```
Option Explicit
Dim StartTime As Variant
Dim EndTime As Variant
Dim ElapsedTime As Variant
```

The **Option Explicit** statement forces us to declare all variables. The other lines establish **StartTime**, **EndTime**, and **ElapsedTime** as variables global within the form.

3. Select the **cmdStart** object in the Object box. If the procedure that appears is not the Click procedure, choose **Click** from the procedure box. Type the following code which begins the timing procedure. Note the **Sub** and **End Sub** statements are provided for you:

```
Sub cmdStart_Click ()
  stablish and print starting time
StartTime = Now
lblStart.Caption = Format(StartTime, "hh:mm:ss")
lblEnd.Caption = ""
lblElapsed.Caption = ""
End Sub
```

In this procedure, once the **Start Timing** button is clicked, we read the current time and print it in a label box. We also blank out the other label boxes. In the code above (and in all code in these notes), any line beginning with a single quote (‘) is a comment. You decide whether you want to type these lines or not. They are not needed for proper application operation.

4. Now, code the **cmdEnd** button.

```
Sub cmdEnd_Click ()
  ind the ending time, compute the elapsed time
  ut both values in label boxes
EndTime = Now
ElapsedTime = EndTime - StartTime
lblEnd.Caption = Format(EndTime, "hh:mm:ss")
lblElapsed.Caption = Format(ElapsedTime, "hh:mm:ss")
End Sub
```

Here, when the **End Timing** button is clicked, we read the current time (**End Time**), compute the elapsed time, and put both values in their corresponding label boxes.

5. And, finally the **cmdExit** button.

```
Sub cmdExit_Click ()
End
End Sub
```

This routine simply ends the application once the **Exit** button is clicked.

6. Did you notice that as you typed in the code, Visual Basic does automatic syntax checking on what you type (if you made any mistakes, that is)?

7. Run your application by clicking the **Run** button on the toolbar, or by pressing <f5>. Pretty easy, wasn't it?

8. Save your application - see the **Primer** on the next page. Use the **Save Project As** option under the **File** menu. Make sure you save both the form and the project files.

9. If you have the time, some other things you may try with the Stopwatch Application:

   A. Try changing the form color and the fonts used in the label boxes and command buttons.

   B. Notice you can press the nd Timing' button before the tart Timing' button. This shouldn't be so. Change the application so you can do this. And make it such that you can press the tart Timing' until nd Timing' has been pressed. Hint: Look at the command button **Enabled** property.

   C. Can you think of how you can continuously display the nd Time' and lapsed Time'? This is a little tricky because of the event-driven nature of Visual Basic. Look at the **Timer** tool. Ask me for help on this one.

Quick Primer on Saving Visual Basic Applications:

When saving Visual Basic applications, you need to be concerned with saving both the forms (.FRM) and modules (.BAS) and the project file (.VBP). In either case, make sure you are saving in the desired directory. The current directory is always displayed in the Save window. Use standard Windows techniques to change the current directory.

There are four **Save** commands available under the **File** menu in Visual Basic:

Save [Form Name]	Save the currently selected form or module with the current name. The selected file is identified in the Project window.
Save [Form Name] As	Like Save File, however you have the option to change the file name
Save Project	Saves all forms and modules in the current project using their current names and also saves the project file.
Save Project As	Like Save Project, however you have the option to change file names. When you choose this option, if you have not saved your forms or modules, you will also be prompted to save those files. I always use this for new projects.

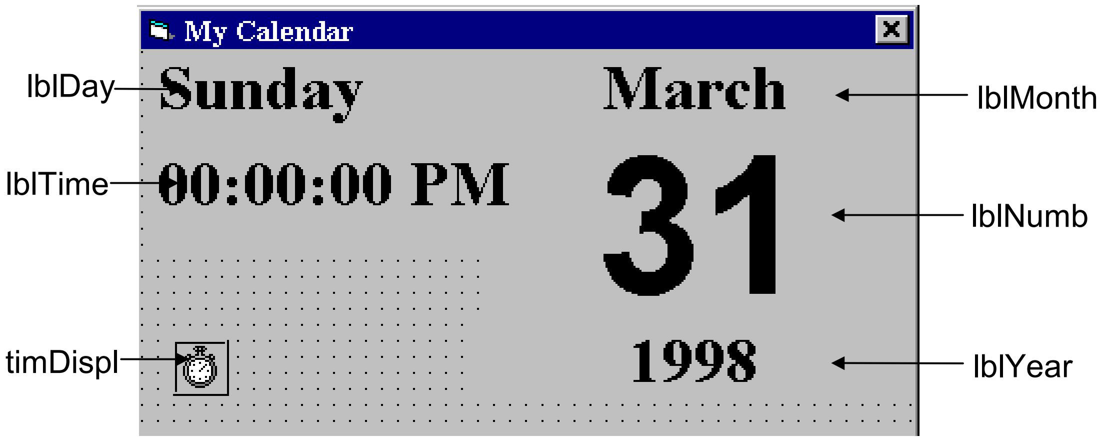
Exercise 1

Calendar/Time Display

Design a window that displays the current month, day, and year. Also, display the current time, updating it every second (look into the **Timer** control). Make the window look something like a calendar page. Play with object properties to make it pretty.

My Solution:

Form:



Properties:

Form **frmCalendar**:
    Caption = My Calendar
    BorderStyle = 1 - Fixed Single

Timer **timDisplay**:
    Interval = 1000

Label **lblDay**:
    Caption = Sunday
    FontName = Times New Roman
    FontBold = True
    FontSize = 24

Label **lblTime**:
    Caption = 00:00:00 PM
    FontName = Times New Roman
    FontBold = True
    FontSize = 24

Label **lblYear**:
    Alignment = 2 - Center
    Caption = 1998
    FontName = Times New Roman
    FontBold = True
    FontSize = 24

Label **lblNumber**:
    Alignment = 2 - Center
    Caption = 31
    FontName = Arial
    FontBold = True
    FontSize = 72

Label **lblMonth**:
    Alignment = 2 - Center
    Caption = March
    FontName = Times New Roman
    FontBold = True
    FontSize = 24

Code:

General Declarations:

```
Option Explicit
```

timDisplay Timer Event:

```
Private Sub timDisplay_Timer()
Dim Today As Variant
Today = Now
lblDay.Caption = Format(Today, "dddd")
lblMonth.Caption = Format(Today, "mmmm")
lblYear.Caption = Format(Today, "yyyy")
lblnumber.Caption = Format(Today, "d")
lblTime.Caption = Format(Today, "h:mm:ss ampm")
End Sub
```

Learn Visual Basic 6.0

2. The Visual Basic Language

Review and Preview

- Last week, we found there were three primary steps involved in developing an application using Visual Basic:

    1. Draw the user interface
    2. Assign properties to controls
    3. Attach code to events

This week, we are primarily concerned with Step 3, attaching code. We will become more familiar with moving around in the Code window and learn some of the elements of the Basic language.

A Brief History of Basic

- Language developed in early 1960's at Dartmouth College:

    B (eginner's)
    A (All-Purpose)
    S (Symbolic)
    I (Instruction)
    C (Code)

- Answer to complicated programming languages (FORTRAN, Algol, Cobol ...). First timeshare language.

- In the mid-1970's, two college students write first Basic for a microcomputer (Altair) - cost $350 on cassette tape. You may have heard of them: Bill Gates and Paul Allen!

- Every Basic since then essentially based on that early version. Examples include: GW-Basic, QBasic, QuickBasic.

- Visual Basic was introduced in 1991.

Visual Basic Statements and Expressions

- The simplest statement is the **assignment** statement. It consists of a variable name, followed by the assignment operator (=), followed by some sort of **expression**.

  Examples:

  ```
  StartTime = Now
  Explorer.Caption = "Captain Spaulding"
  BitCount = ByteCount * 8
  Energy = Mass * LIGHTSPEED ^ 2
  NetWorth = Assets - Liabilities
  ```

  The assignment statement stores information.

- Statements normally take up a single line with no terminator. Statements can be **stacked** by using a colon (**:**) to separate them. Example:

  ```
  StartTime = Now : EndTime = StartTime + 10
  ```

  (Be careful stacking statements, especially with If/End If structures. You may not get the response you desire.)

- If a statement is very long, it may be continued to the next line using the **continuation** character, an underscore (**_**). Example:

  ```
  Months = Log(Final * IntRate / Deposit + 1) _
  / Log(1 + IntRate)
  ```

- Comment statements begin with the keyword **Rem** or a single quote (**'**). For example:

  ```
  Rem This is a remark
  ' This is also a remark
  x = 2 * y ' another way to write a remark or comment
  ```

  You, as a programmer, should decide how much to comment your code. Consider such factors as reuse, your audience, and the legacy of your code.

Visual Basic Operators

- The simplest **operators** carry out **arithmetic** operations. These operators in their order of precedence are:

Operator	Operation
^	Exponentiation
* /	Multiplication and division
\	Integer division (truncates)
Mod	Modulus
+ -	Addition and subutraction

- **Parentheses** around expressions can change precedence.

- To **concatentate** two strings, use the **&** symbol or the **+** symbol:

```
lblTime.Caption = "The current time is" & Format(Now,     h:mm”)
txtSample.Text = "Hook this “ +     o this”
```

- There are six **comparison** operators in Visual Basic:

Operator	Comparison
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
=	Equal to
<>	Not equal to

- The result of a comparison operation is a Boolean value (**True** or **False**).

- We will use three **logical** operators

Operator	**Operation**
Not	Logical not
And	Logical and
Or	Logical or

- The **Not** operator simply negates an operand.

- The **And** operator returns a True if both operands are True. Else, it returns a False.

- The **Or** operator returns a True if either of its operands is True, else it returns a False.

- Logical operators follow arithmetic operators in precedence.

Visual Basic Functions

- Visual Basic offers a rich assortment of built-in **functions**. The on-line help utility will give you information on any or all of these functions and their use. Some examples are:

Function	**Value Returned**
Abs	Absolute value of a number
Asc	ASCII or ANSI code of a character
Chr	Character corresponding to a given ASCII or ANSI code
Cos	Cosine of an angle
Date	Current date as a text string
Format	Date or number converted to a text string
Left	Selected left side of a text string
Len	Number of characters in a text string
Mid	Selected portion of a text string
Now	Current time and date
Right	Selected right end of a text string
Rnd	Random number
Sin	Sine of an angle
Sqr	Square root of a number
Str	Number converted to a text string
Time	Current time as a text string
Timer	Number of seconds elapsed since midnight
Val	Numeric value of a given text string

A Closer Look at the Rnd Function

- In writing games and learning software, we use the **Rnd** function to introduce randomness. This insures different results each time you try a program. The Visual Basic function Rnd returns a single precision, random number between 0 and 1 (actually greater than or equal to 0 and less than 1). To produce random integers (I) between Imin and Imax, use the formula:

  I = Int((Imax - Imin + 1) * Rnd) + Imin

- The random number generator in Visual Basic must be seeded. A **Seed** value initializes the generator. The **Randomize** statement is used to do this:

  Randomize Seed

  If you use the same Seed each time you run your application, the same sequence of random numbers will be generated. To insure you get different numbers every time you use your application (preferred for games), use the **Timer** function to seed the generator:

  Randomize Timer

  Place this statement in the **Form_Load** event procedure.

- **Examples**:

  To roll a six-sided die, the number of spots would be computed using:

  NumberSpots = Int(6 * Rnd) + 1

  To randomly choose a number between 100 and 200, use:

  Number = Int(101 * Rnd) + 100

Example 2-1

Savings Account

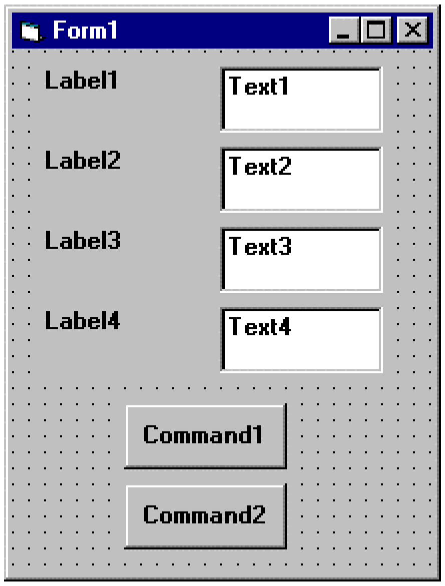
1. Start a new project. The idea of this project is to determine how much you save by making monthly deposits into a savings account. For those interested, the mathematical formula used is:

$$F = D\,[\,(1 + I)^{M} - 1]\,/\,I$$

where

F - Final amount
D - Monthly deposit amount
I - Monthly interest rate
M - Number of months

2. Place 4 label boxes, 4 text boxes, and 2 command buttons on the form. It should look something like this:



3. Set the properties of the form and each object.

Form1:

BorderStyle	1-Fixed Single
Caption	Savings Account
Name	frmSavings

Label1:

Caption	Monthly Deposit

Label2:

Caption	Yearly Interest

Label3:

Caption	Number of Months

Label4:

Caption	Final Balance

Text1:

Text	[Blank]
Name	txtDeposit

Text2:

Text	[Blank]
Name	txtInterest

Text3:

Text	[Blank]
Name	txtMonths

Text4:

Text	[Blank]
Name	txtFinal

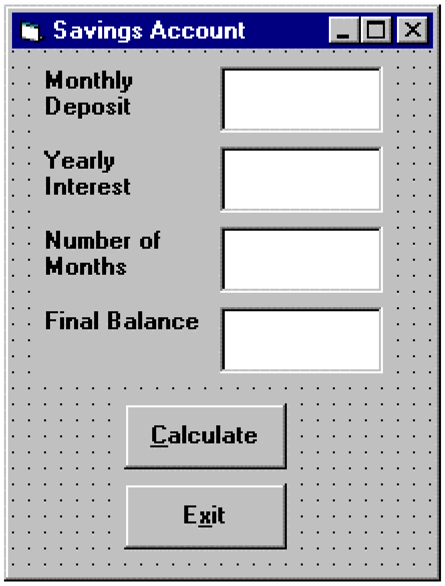
Command1:

Caption	&Calculate
Name	cmdCalculate

Command2:

Caption	E&xit
Name	cmdExit

Now, your form should look like this:



4. Declare four variables in the **general declarations** area of your form. This makes them available to all the form procedures:

```
Option Explicit
Dim Deposit As Single
Dim Interest As Single
Dim Months As Single
Dim Final As Single
```

The **Option Explicit** statement forces us to declare all variables.

5. Attach code to the **cmdCalculate** command button **Click** event.

```
Private Sub cmdCalculate_Click ()
Dim IntRate As Single
  ead values from text boxes
Deposit = Val(txtDeposit.Text)
Interest = Val(txtInterest.Text)
IntRate = Interest / 1200
Months = Val(txtMonths.Text)
  ompute final value and put in text box
Final = Deposit * ((1 + IntRate) ^ Months - 1) /
IntRate
txtFinal.Text = Format(Final, "#####0.00")
End Sub
```

This code reads the three input values (monthly deposit, interest rate, number of months) from the text boxes, computes the final balance using the provided formula, and puts that result in a text box.

6. Attach code to the **cmdExit** command button **Click** event.

```
Private Sub cmdExit_Click ()
End
End Sub
```

7. Play with the program. Make sure it works properly. Save the project.

Visual Basic Symbolic Constants

- Many times in Visual Basic, functions and objects require data arguments that affect their operation and return values you want to read and interpret. These arguments and values are constant numerical data and difficult to interpret based on just the numerical value. To make these constants more understandable, Visual Basic assigns names to the most widely used values - these are called **symbolic constants**. Appendix I lists many of these constants.

- As an example, to set the background color of a form named **frmExample** to blue, we could type:

  ```
  frmExample.BackColor = 0xFF0000
  ```

  or, we could use the symbolic constant for the blue color (**vbBlue**):

  ```
  frmExample.BackColor = vbBlue
  ```

- It is strongly suggested that the symbolic constants be used instead of the numeric values, when possible. You should agree that **vbBlue** means more than the value **0xFF0000** when selecting the background color in the above example. You do not need to do anything to define the symbolic constants - they are built into Visual Basic.

Defining Your Own Constants

- You can also define your own constants for use in Visual Basic. The format for defining a constant named **PI** with a value **3.14159** is:

  ```
  Const PI = 3.14159
  ```

- **User-defined constants** should be written in all upper case letters to distinguish them from variables. The scope of constants is established the same way a variables' scope is. That is, if defined within a procedure, they are local to the procedure. If defined in the general declarations of a form, they are global to the form. To make constants global to an application, use the format:

  ```
  Global Const PI = 3.14159
  ```

  within the general declarations area of a module.

Visual Basic Branching - If Statements

- **Branching** statements are used to cause certain actions within a program if a certain condition is met.

- The simplest is the **If/Then** statement:

```
If Balance - Check < 0 Then Print "You are overdrawn"
```

  Here, if and only if Balance - Check is less than zero, the statement you are overdrawn" is printed.

- You can also have **If/Then/End If** blocks to allow multiple statements:

```
If Balance - Check < 0 Then
  Print "You are overdrawn"
  Print "Authorities have been notified"
End If
```

  In this case, if Balance - Check is less than zero, two lines of information are printed.

- Or, **If/Then/Else/End If** blocks:

```
If Balance - Check < 0 Then
  Print "You are overdrawn"
  Print "Authorities have been notified"
Else
  Balance = Balance - Check
End If
```

  Here, the same two lines are printed if you are overdrawn (Balance - Check < 0), but, if you are not overdrawn (**Else**), your new Balance is computed.

- Or, we can add the **ElseIf** statement:

  ```
  If Balance - Check < 0 Then
    Print "You are overdrawn"
    Print "Authorities have been notified"
  ElseIf Balance - Check = 0 Then
    Print "Whew! You barely made it"
    Balance = 0
  Else
    Balance = Balance - Check
  End If
  ```

  Now, one more condition is added. If your Balance equals the Check amount (**ElseIf** Balance - Check = 0), a different message appears.

- In using branching statements, make sure you consider all viable possibilities in the If/Else/End If structure. Also, be aware that each If and ElseIf in a block is tested sequentially. The first time an If test is met, the code associated with that condition is executed and the If block is exited. If a later condition is also True, it will never be considered.

Key Trapping

- Note in the previous example, there is nothing to prevent the user from typing in meaningless characters (for example, letters) into the text boxes expecting numerical data. Whenever getting input from a user, we want to limit the available keys they can press. The process of intercepting unacceptable keystrokes is **key trapping**.

- Key trapping is done in the **KeyPress** procedure of an object. Such a procedure has the form (for a text box named **txtText**):

  ```
  Sub txtText_KeyPress (KeyAscii as Integer)
       .
       .
       .
  End Sub
  ```

  What happens in this procedure is that every time a key is pressed in the corresponding text box, the ASCII code for the pressed key is passed to this procedure in the argument list (i.e. **KeyAscii**). If KeyAscii is an acceptable value, we would do nothing. However, if KeyAscii is not acceptable, we would set KeyAscii equal to zero and exit the procedure. Doing this has the same result of not pressing a key at all. ASCII values for all keys are available via the on-line help in Visual Basic. And some keys are also defined by symbolic constants. Where possible, we will use symbolic constants; else, we will use the ASCII values.

- As an example, say we have a text box (named **txtExample**) and we only want to be able to enter upper case letters (ASCII codes 65 through 90, or, correspondingly, symbolic constants **vbKeyA** through **vbKeyZ**). The key press procedure would look like (the **Beep** causes an audible tone if an incorrect key is pressed):

```
Sub txtExample_KeyPress(KeyAscii as Integer)
  If KeyAscii >= vbKeyA And KeyAscii <= vbKeyZ Then
    Exit Sub
  Else
    KeyAscii = 0
    Beep
  End If
End Sub
```

- In key trapping, it's advisable to always allow the backspace key (ASCII code 8; symbolic constant **vbKeyBack**) to pass through the key press event. Else, you will not be able to edit the text box properly.

Example 2-2

Savings Account - Key Trapping

1. Note the acceptable ASCII codes are 48 through 57 (numbers), 46 (the decimal point), and 8 (the backspace key). In the code, we use symbolic constants for the numbers and backspace key. Such a constant does not exist for the decimal point, so we will define one with the following line in the **general declaration** area:

```
Const vbKeyDecPt = 46
```

2. Add the following code to the three procedures: **txtDeposit_KeyPress**, **txtInterest_KeyPress**, and **txtMonths_KeyPress**.

```
Private Sub txtDeposit_KeyPress (KeyAscii As Integer)
  nly allow number keys, decimal point, or backspace
If (KeyAscii >= vbKey0 And KeyAscii <= vbKey9) Or
KeyAscii = vbKeyDecPt Or KeyAscii = vbKeyBack Then
  Exit Sub
Else
 KeyAscii = 0
 Beep
End If
End Sub

Private Sub txtInterest_KeyPress (KeyAscii As Integer)
  nly allow number keys, decimal point, or backspace
If (KeyAscii >= vbKey0 And KeyAscii <= vbKey9) Or
KeyAscii = vbKeyDecPt Or KeyAscii = vbKeyBack Then
  Exit Sub
Else
 KeyAscii = 0
 Beep
End If
End Sub
```

```
Private Sub txtMonths_KeyPress (KeyAscii As Integer)
 nly allow number keys, decimal point, or backspace
If (KeyAscii >= vbKey0 And KeyAscii <= vbKey9) Or
KeyAscii = vbKeyDecPt Or KeyAscii = vbKeyBack Then
  Exit Sub
Else
 KeyAscii = 0
 Beep
End If
End Sub
```

(In the If statements above, note the word processor causes a line break where there really shouldn be one. That is, there is no line break between the words **Or KeyAscii** and **= vbKeyDecPt**. One appears due to page margins. In all code in these notes, always look for such things.)

3. Rerun the application and test the key trapping performance.

Select Case - Another Way to Branch

- In addition to If/Then/Else type statements, the **Select Case** format can be used when there are multiple selection possibilities.

- Say we've written this code using the **If** statement:

```
If Age = 5 Then
  Category = "Five Year Old"
ElseIf Age >= 13 and Age <= 19 Then
  Category = "Teenager"
ElseIf (Age >= 20 and Age <= 35) Or Age = 50 Or (Age >= 60 and Age
<= 65) Then
  Category = "Special Adult"
ElseIf Age > 65 Then
  Category = "Senior Citizen"
Else
  Category = "Everyone Else"
End If
```

The corresponding code with **Select Case** would be:

```
Select Case Age
  Case 5
    Category = "Five Year Old"
  Case 13 To 19
    Category = "Teenager"
  Case 20 To 35, 50, 60 To 65
    Category = "Special Adult"
  Case Is > 65
    Category = "Senior Citizen"
  Case Else
    Category = "Everyone Else"
End Select
```

Notice there are several formats for the Case statement. Consult on-line help for discussions of these formats.

The GoTo Statement

- Another branching statement, and perhaps the most hated statement in programming, is the **GoTo** statement. However, we will need this to do Run-Time error trapping. The format is **GoTo *Label***, where ***Label*** is a labeled line. Typing the Label followed by a colon forms labeled lines.

- **GoTo** Example:

```
Line10:
    .
    .
GoTo Line10
```

When the code reaches the GoTo statement, program control transfers to the line labeled Line10.

Visual Basic Looping

- Looping is done with the **Do/Loop** format. Loops are used for operations are to be repeated some number of times. The loop repeats until some specified condition at the beginning or end of the loop is met.

- **Do While/Loop** Example:

```
Counter = 1
Do While Counter <= 1000
  Debug.Print Counter
  Counter = Counter + 1
Loop
```

This loop repeats as long as (**While**) the variable Counter is less than or equal to 1000. Note a Do While/Loop structure will not execute even once if the While condition is violated (False) the first time through. Also note the **Debug.Print** statement. What this does is print the value Counter in the Visual Basic Debug window. We'll learn more about this window later in the course.

- **Do Until/Loop** Example:

  ```
  Counter = 1
  Do Until Counter >  1000
    Debug.Print Counter
    Counter = Counter + 1
  Loop
  ```

  This loop repeats **Until** the Counter variable exceeds 1000. Note a Do Until/Loop structure will not be entered if the Until condition is already True on the first encounter.

- **Do/Loop While** Example:

  ```
  Sum = 1
  Do
    Debug.Print Sum
    Sum = Sum + 3
  Loop While Sum <= 50
  ```

  This loop repeats **While** the Variable Sum is less than or equal to 50. Note, since the While check is at the end of the loop, a Do/Loop While structure is always executed at least once.

- **Do/Loop Until** Example:

  ```
  Sum = 1
  Do
    Debug.Print Sum
    Sum = Sum + 3
  Loop Until Sum > 50
  ```

  This loop repeats Until Sum is greater than 50. And, like the previous example, a Do/Loop Until structure always executes at least once.

- Make sure you can always get out of a loop! Infinite loops are never nice. If you get into one, try **Ctrl+Break**. That sometimes works - other times the only way out is rebooting your machine!

- The statement **Exit Do** will get you out of a loop and transfer program control to the statement following the Loop statement.

Visual Basic Counting

- Counting is accomplished using the **For/Next** loop.

  Example

  ```
  For I = 1 to 50 Step 2
    A = I * 2
    Debug.Print A
  Next I
  ```

  In this example, the variable I initializes at 1 and, with each iteration of the For/Next loop, is incremented by 2 (**Step**). This looping continues until I becomes greater than or equal to its final value (50). If Step is not included, the default value is 1. Negative values of Step are allowed.

- You may exit a For/Next loop using an **Exit For** statement. This will transfer program control to the statement following the **Next** statement.

Example 2-3

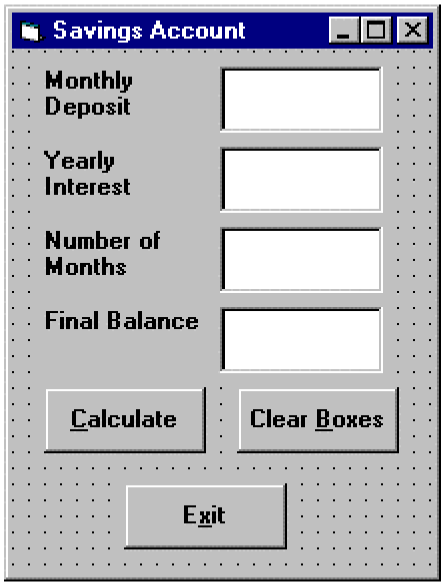
Savings Account - Decisions

1. Here, we modify the Savings Account project to allow entering any three values and computing the fourth. First, add a third command button that will clear all of the text boxes. Assign the following properties:

   Command3:

Caption	Clear &Boxes
Name	cmdClear

   The form should look something like this when you e done:



2. Code the **cmdClear** button **Click** event:

```
Private Sub cmdClear_Click ()
  lank out the text boxes
txtDeposit.Text = ""
txtInterest.Text = ""
txtMonths.Text = ""
txtFinal.Text = ""
End Sub
```

   This code simply blanks out the four text boxes when the **Clear** button is clicked.

3. Code the **KeyPress** event for the **txtFinal** object:

   ```
   Private Sub txtFinal_KeyPress (KeyAscii As Integer)
     nly allow number keys, decimal point, or backspace
   If (KeyAscii >= vbKey0 And KeyAscii <= vbKey9) Or
   KeyAscii = vbKeyDecPt Or KeyAscii = vbKeyBack Then
      Exit Sub
   Else
     KeyAscii = 0
     Beep
   End If
   End Sub
   ```

   We need this code because we can now enter information into the Final Value text box.

4. The modified code for the **Click** event of the **cmdCalculate** button is:

```
Private Sub cmdCalculate_Click()
Dim IntRate As Single
Dim IntNew As Single
Dim Fcn As Single, FcnD As Single
  ead the four text boxes
Deposit = Val(txtDeposit.Text)
Interest = Val(txtInterest.Text)
IntRate = Interest / 1200
Months = Val(txtMonths.Text)
Final = Val(txtFinal.Text)
  etermine which box is blank
  ompute that missing value and put in text box
If txtDeposit.Text = "" Then
  eposit missing
  Deposit = Final / (((1 + IntRate) ^ Months - 1) /
   IntRate)
  txtDeposit.Text = Format(Deposit, "#####0.00")
ElseIf txtInterest.Text = "" Then
  nterest missing - requires iterative solution
  IntNew = (Final / (0.5* Months * Deposit) - 1) / Months
  Do
    IntRate = IntNew
    Fcn = (1 + IntRate) ^ Months - Final * IntRate /
   Deposit - 1
    FcnD = Months * (1 + IntRate) ^ (Months - 1) - Final
   / Deposit
    IntNew = IntRate - Fcn / FcnD
  Loop Until Abs(IntNew - IntRate) < 0.00001 / 12
```

```
  Interest = IntNew * 1200
  txtInterest.Text = Format(Interest, "##0.00")
ElseIf txtMonths.Text = "" Then
 onths missing
  Months = Log(Final * IntRate / Deposit + 1) / Log(1 +
   IntRate)
  txtMonths.Text = Format(Months, "###.0")
ElseIf txtFinal.Text = "" Then
 inal value missing
  Final = Deposit * ((1 + IntRate) ^ Months - 1) /
   IntRate
  txtFinal.Text = Format(Final, "#####0.00")
End If
End Sub
```

In this code. we first read the text information from all four text boxes and based on which one is blank, compute the missing information and display it in the corresponding text box. Solving for missing **Deposit**, **Months**, or **Final** information is a straightforward manipulation of the equation given in Example 2-2.

If the **Interest** value is missing, we have to solve an Mth-order polynomial using something called Newton-Raphson iteration - a good example of using a Do loop. Finding the **Interest** value is straightforward. What we do is guess at what the interest is, compute a better guess (using Newton-Raphson iteration), and repeat the process (loop) until the old guess and the new guess are close to each other. You can see each step in the code.

5. Test and save your application. Go home and relax.

Exercise 2-1

Computing a Mean and Standard Deviation

Develop an application that allows the user to input a sequence of numbers. When done inputting the numbers, the program should compute the mean of that sequence and the standard deviation. If N numbers are input, with the ith number represented by x_i, the formula for the mean ($\overline{x}$) is:

$$\overline{x} = (\sum_{i=1}^{N} x_i)/N$$

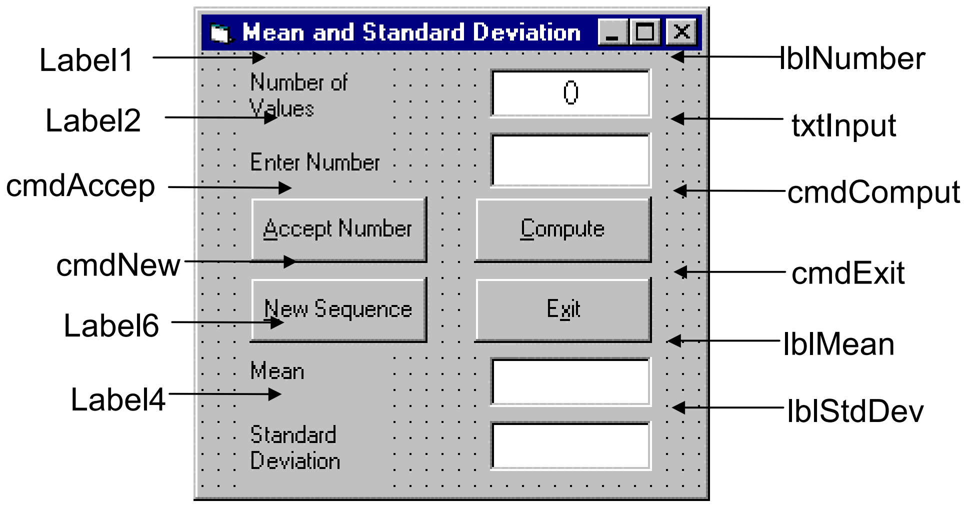
and to compute the standard deviation (s), take the square root of this equation:

$$s^2 = [N\sum_{i=1}^{N} x_i^2 - (\sum_{i=1}^{N} x_i)^2]/[N(N-1)]$$

The Greek sigmas in the above equations simply indicate that you add up all the corresponding elements next to the sigma.

My Solution:

Form:



Properties:

Form **frmStats**:
    Caption = Mean and Standard Deviation

CommandButton **cmdExit**:
    Caption = E&xit

CommandButton **cmdAccept**:
    Caption = &Accept Number

CommandButton **cmdCompute**:
    Caption = &Compute

CommandButton **cmdNew**:
    Caption = &New Sequence

TextBox **txtInput**:
    FontName = MS Sans Serif
    FontSize = 12

Label **lblStdDev**:
    Alignment = 2 - Center
    BackColor = &H00FFFFFF& (White)
    BorderStyle = 1 - Fixed Single
    FontName = MS Sans Serif
    FontSize = 12

Label **Label6**:
    Caption = Standard Deviation

Label **lblMean**:
    Alignment = 2 - Center
    BackColor = &H00FFFFFF& (White)
    BorderStyle = 1 - Fixed Single
    FontName = MS Sans Serif
    FontSize = 12

Label **Label4**:
    Caption = Mean

Label **lblNumber**:
    Alignment = 2 - Center
    BackColor = &H00FFFFFF& (White)
    BorderStyle = 1 - Fixed Single
    FontName = MS Sans Serif
    FontSize = 12

Label **Label2**:
    Caption = Enter Number

Label **Label1**:
    Caption = Number of Values

Code:

General Declarations:

```
Option Explicit
Dim NumValues As Integer
Dim SumX As Single
Dim SumX2 As Single
Const vbKeyMinus = 45
Const vbKeyDecPt = 46
```

cmdAccept Click Event:

```
Private Sub cmdAccept_Click()
Dim Value As Single
txtInput.SetFocus
NumValues = NumValues + 1
lblNumber.Caption = Str(NumValues)
  et number and sum number and number-squared
Value = Val(txtInput.Text)
SumX = SumX + Value
SumX2 = SumX2 + Value ^ 2
txtInput.Text = ""
End Sub
```

cmdCompute Click Event:

```
Private Sub cmdCompute_Click()
Dim Mean As Single
Dim StdDev As Single
txtInput.SetFocus
  ake sure there are at least two values
If NumValues < 2 Then
  Beep
  Exit Sub
End If
  ompute mean
Mean = SumX / NumValues
lblMean.Caption = Str(Mean)
  ompute standard deviation
StdDev = Sqr((NumValues * SumX2 - SumX ^ 2) / (NumValues
* (NumValues - 1)))
lblStdDev.Caption = Str(StdDev)
End Sub
```

cmdExit Click Event:

```
Private Sub cmdExit_Click()
End
End Sub
```

cmdNew Click Event:

```
Private Sub cmdNew_Click()
'Initialize all variables
txtInput.SetFocus
NumValues = 0
lblNumber.Caption = "0"
txtInput.Text = ""
lblMean.Caption = ""
lblStdDev.Caption = ""
SumX = 0
SumX2 = 0
End Sub
```

txtInput KeyPress Event:

```
Private Sub txtInput_KeyPress(KeyAscii As Integer)
'Only allow numbers, minus sign, decimal point,
backspace, return keys
If (KeyAscii >= vbKey0 And KeyAscii <= vbKey9) Or
KeyAscii = vbKeyMinus Or KeyAscii = vbKeyDecPt Or
KeyAscii = vbKeyBack Then
  Exit Sub
ElseIf KeyAscii = vbKeyReturn Then
  Call cmdAccept_Click
Else
  KeyAscii = 0
End If
End Sub
```

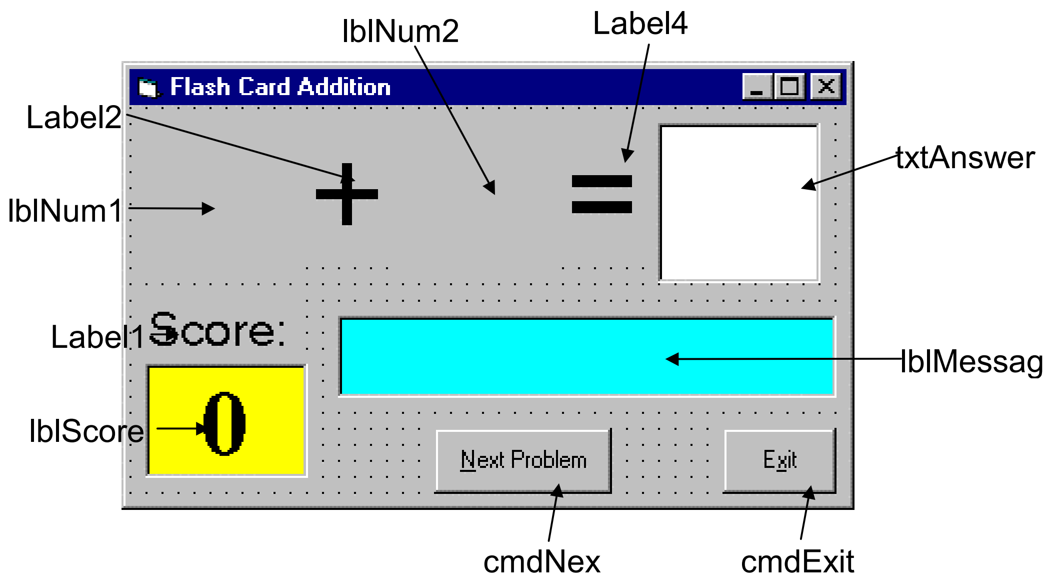
Exercise 2-2

Flash Card Addition Problems

Write an application that generates random addition problems. Provide some kind of feedback and scoring system as the problems are answered.

My Solution:

Form:



Properties:

Form **frmAdd**:
    BorderStyle = 1 - Fixed Single
    Caption = Flash Card Addition

CommandButton **cmdNext**:
    Caption = &Next Problem
    Enabled = False

CommandButton **cmdExit**:
    Caption = E&xit

TextBox **txtAnswer**:
    FontName = Arial
    FontSize = 48
    MaxLength = 2

Label **lblMessage**:
    Alignment = 2 - Center
    BackColor = &H00FFFF00& (Cyan)
    BorderStyle = 1 - Fixed Single
    FontName = MS Sans Serif
    FontBold = True
    FontSize = 24
    FontItalic = True

Label **lblScore**:
    Alignment = 2 - Center
    BackColor = &H0000FFFF& (Yellow)
    BorderStyle = 1 - Fixed Single
    Caption = 0
    FontName = Times New Roman
    FontBold = True
    FontSize = 36

Label **Label1**:
    Alignment = 2 - Center
    Caption = Score:
    FontName = MS Sans Serif
    FontSize = 18

Label **Label4**:
    Alignment = 2 - Center
    Caption = =
    FontName = Arial
    FontSize = 48

Label **lblNum2**:
    Alignment = 2 - Center
    FontName = Arial
    FontSize = 48

Label **Label2**:
    Alignment = 2 - Center
    Caption = +
    FontName = Arial
    FontSize = 48

Label **lblNum1**:
    Alignment = 2 - Center
    FontName = Arial
    FontSize = 48

Code:

General Declarations:

```
Option Explicit
Dim Sum As Integer
Dim NumProb As Integer, NumRight As Integer
```

cmdExit Click Event:

```
Private Sub cmdExit_Click()
End
End Sub
```

cmdNext Click Event:

```
Private Sub cmdNext_Click()
'Generate next addition problem
Dim Number1 As Integer
Dim Number2 As Integer
txtAnswer.Text = ""
lblMessage.Caption = ""
NumProb = NumProb + 1
'Generate random numbers for addends
Number1 = Int(Rnd * 21)
Number2 = Int(Rnd * 21)
lblNum1.Caption = Format(Number1, "#0")
lblNum2.Caption = Format(Number2, "#0")
'Find sum
Sum = Number1 + Number2
cmdNext.Enabled = False
txtAnswer.SetFocus
End Sub
```

Form Activate Event:

```
Private Sub Form_Activate()
Call cmdNext_Click
End Sub
```

Form Load Event:

```
Private Sub Form_Load()
Randomize Timer
NumProb = 0
NumRight = 0
End Sub
```

txtAnswer KeyPress Event:

```
Private Sub txtAnswer_KeyPress(KeyAscii As Integer)
Dim Ans As Integer
'Check for number only input and for return key
If (KeyAscii >= vbKey0 And KeyAscii <= vbKey9) Or
KeyAscii = vbKeyBack Then
  Exit Sub
ElseIf KeyAscii = vbKeyReturn Then
'Check answer
  Ans = Val(txtAnswer.Text)
  If Ans = Sum Then
    NumRight = NumRight + 1
    lblMessage.Caption = "That's correct!"
  Else
    lblMessage.Caption = "Answer is " + Format(Sum, "#0")
  End If
  lblScore.Caption = Format(100 * NumRight / NumProb,
"##0")
  cmdNext.Enabled = True
  cmdNext.SetFocus
Else
  KeyAscii = 0
End If
End Sub
```

This page intentionally not left blank.

Learn Visual Basic 6.0

3. Exploring the Visual Basic Toolbox

Review and Preview

- In this class, we begin a journey where we look at each tool in the Visual Basic toolbox. We will revisit some tools we already know and learn a lot of new tools. First, though, we look at an important Visual Basic functions.

The Message Box

- One of the best functions in Visual Basic is the **message box**. The message box displays a message, optional icon, and selected set of command buttons. The user responds by clicking a button.

- The **statement** form of the message box returns no value (it simply displays the box):

  MsgBox Message, Type, Title

  where

Message	Text message to be displayed
Type	Type of message box (discussed in a bit)
Title	Text in title bar of message box

  You have no control over where the message box appears on the screen.

- The **function** form of the message box returns an integer value (corresponding to the button clicked by the user). Example of use (Response is returned value):

```
Dim Response as Integer
Response = MsgBox(Message, Type, Title)
```

- The **Type** argument is formed by summing four values corresponding to the buttons to display, any icon to show, which button is the default response, and the modality of the message box.
- The first component of the **Type** value specifies the **buttons** to display:

Value	Meaning	Symbolic Constant
0	OK button only	vbOKOnly
1	OK/Cancel buttons	vbOKCancel
2	Abort/Retry/Ignore buttons	vbAbortRetryIgnore
3	Yes/No/Cancel buttons	vbYesNoCancel
4	Yes/No buttons	vbYesNo
5	Retry/Cancel buttons	vbRetryCancel

- The second component of **Type** specifies the **icon** to display in the message box:

Value	Meaning	Symbolic Constant
0	No icon	(None)
16	Critical icon	vbCritical
32	Question mark	vbQuestion
48	Exclamation point	vbExclamation
64	Information icon	vbInformation

- The third component of **Type** specifies which button is **default** (i.e. pressing Enter is the same as clicking the default button):

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

- The fourth and final component of **Type** specifies the **modality**:

Value	Meaning	Symbolic Constant
0	Application modal	vbApplicationModal
4096	System modal	vbSystemModal

If the box is **Application Modal**, the user must respond to the box before continuing work in the current application. If the box is **System Modal**, all applications are suspended until the user responds to the message box.

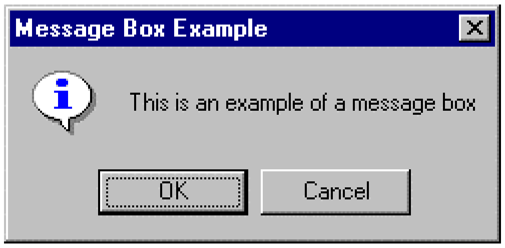
- Note for each option in **Type**, there are numeric values listed and symbolic constants. Recall, it is strongly suggested that the symbolic constants be used instead of the numeric values. You should agree that **vbOKOnly** means more than the number 0 when selecting the button type.

- The value returned by the function form of the message box is related to the button clicked:

Value	Meaning	Symbolic Constant
1	OK button selected	vbOK
2	Cancel button selected	vbCancel
3	Abort button selected	vbAbort
4	Retry button selected	vbRetry
5	Ignore button selected	vbIgnore
6	Yes button selected	vbYes
7	No button selected	vbNo

- Message Box Example:

  MsgBox   his is an example of a message box", vbOKCancel + vbInformation, message Box Example"



This is an example of a message box

- You've seen message boxes if you've ever used a Windows application. Think of all the examples you've seen. For example, message boxes are used to ask you if you wish to save a file before exiting and to warn you if a disk drive is not ready.

Object Methods

- In previous work, we have seen that each object has properties and events associated with it. A third concept associated with objects is the **method**. A method is a procedure or function that imparts some action to an object.

- As we move through the toolbox, when appropriate, we'll discuss object methods. Methods are always enacted at run-time in code. The format for invoking a method is:

  ObjectName.Method {optional arguments}

  Note this is another use of the dot notation.

The Form Object

- The **Form** is where the user interface is drawn. It is central to the development of Visual Basic applications.

- Form Properties:

Appearance	Selects 3-D or flat appearance.
BackColor	Sets the form background color.
BorderStyle	Sets the form border to be fixed or sizeable.
Caption	Sets the form window title.
Enabled	If True, allows the form to respond to mouse and keyboard events; if False, disables form.
Font	Sets font type, style, size.
ForeColor	Sets color of text or graphics.
Picture	Places a bitmap picture in the form.
Visible	If False, hides the form.

- Form Events:

Activate	Form_Activate event is triggered when form becomes the active window.
Click	Form_Click event is triggered when user clicks on form.
DblClick	Form_DblClick event is triggered when user double-clicks on form.
Load	Form_Load event occurs when form is loaded. This is a good place to initialize variables and set any run-time properties.

- Form Methods:

Cls	Clears all graphics and text from form. Does not clear any objects.
Print	Prints text string on the form.

Examples

```
frmExample.Cls ' clears the form
frmExample.Print "This will print on the form"
```

Command Buttons

- We've seen the **command button** before. It is probably the most widely used control. It is used to begin, interrupt, or end a particular process.

- Command Button Properties:

Appearance	Selects 3-D or flat appearance.
Cancel	Allows selection of button with **Esc** key (only one button on a form can have this property True).
Caption	String to be displayed on button.
Default	Allows selection of button with **Enter** key (only one button on a form can have this property True).
Font	Sets font type, style, size.

- Command Button Events:

Click	Event triggered when button is selected either by clicking on it or by pressing the access key.

Label Boxes

- A **label box** is a control you use to display text that a user can't edit directly. We've seen, though, in previous examples, that the text of a label box can be changed at run-time in response to events.

- Label Properties:

Alignment	Aligns caption within border.
Appearance	Selects 3-D or flat appearance.
AutoSize	If True, the label is resized to fit the text specified by the caption property. If False, the label will remain the size defined at design time and the text may be clipped.
BorderStyle	Determines type of border.
Caption	String to be displayed in box.
Font	Sets font type, style, size.
word-wrap	Works in conjunction with AutoSize property. If AutoSize = True, word-wrap = True, then the text

	will wrap and label will expand vertically to fit the Caption. If AutoSize = True, word-wrap = False, then the text will not wrap and the label expands horizontally to fit the Caption. If AutoSize = False, the text will not wrap regardless of word-wrap value.

- Label Events:

Click	Event triggered when user clicks on a label.
DblClick	Event triggered when user double-clicks on a label.

Text Boxes

- A **text box** is used to display information entered at design time, by a user at run-time, or assigned within code. The displayed text may be edited.

- Text Box Properties:

Appearance	Selects 3-D or flat appearance.
BorderStyle	Determines type of border.
Font	Sets font type, style, size.
MaxLength	Limits the length of displayed text (0 value indicates unlimited length).
MultiLine	Specifies whether text box displays single line or multiple lines.
PasswordChar	Hides text with a single character.
ScrollBars	Specifies type of displayed scroll bar(s).
SelLength	Length of selected text (run-time only).
SelStart	Starting position of selected text (run-time only).
SelText	Selected text (run-time only).
Tag	Stores a string expression.
Text	Displayed text.

- Text Box Events:

Change	Triggered every time the **Text** property changes.
LostFocus	Triggered when the user leaves the text box. This is a good place to examine the contents of a text box after editing.
KeyPress	Triggered whenever a key is pressed. Used for key trapping, as seen in last class.

- Text Box Methods:

SetFocus	Places the cursor in a specified text box.

Example

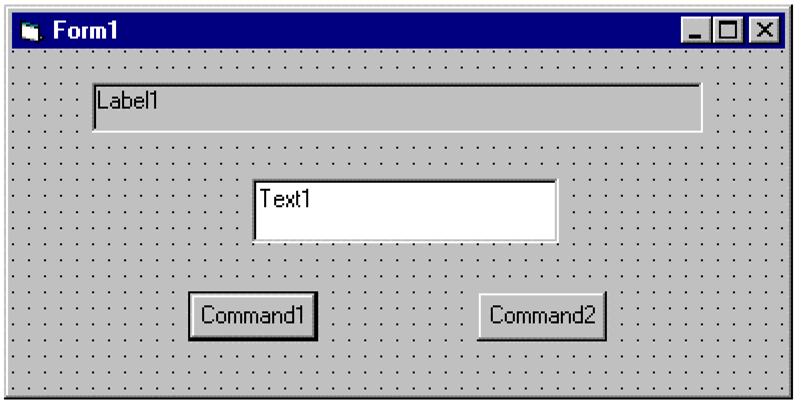
```
txtExample.SetFocus ' moves cursor to box named txtExample
```

Example 3-1

Password Validation

1. Start a new project. The idea of this project is to ask the user to input a password. If correct, a message box appears to validate the user. If incorrect, other options are provided.

2. Place a two command buttons, a label box, and a text box on your form so it looks something like this:

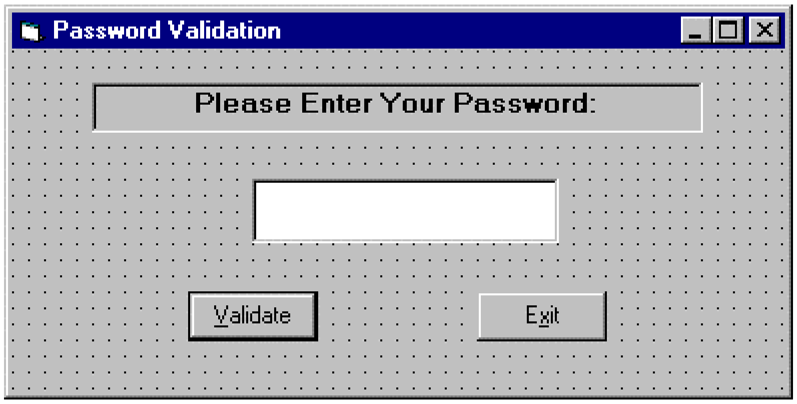


3. Set the properties of the form and each object.

Form1:		
	BorderStyle	1-Fixed Single
	Caption	Password Validation
	Name	frmPassword
Label1:		
	Alignment	2-Center
	BorderStyle	1-Fixed Single
	Caption	Please Enter Your Password:
	FontSize	10
	FontStyle	Bold
Text1:		
	FontSize	14
	FontStyle	Regular
	Name	txtPassword
	PasswordChar	*
	Tag	[Whatever you choose as a password]
	Text	[Blank]
Command1:		
	Caption	&Validate
	Default	True
	Name	cmdValid

Command2:		
	Cancel	True
	Caption	E&xit
	Name	cmdExit

Your form should now look like this:



4. Attach the following code to the **cmdValid_Click** event.

```
Private Sub cmdValid_Click()
'This procedure checks the input password
Dim Response As Integer
If txtPassword.Text = txtPassword.Tag Then
'If correct, display message box
  MsgBox "You've passed security!", vbOKOnly +
   vbExclamation, "Access Granted"
Else
'If incorrect, give option to try again
  Response = MsgBox("Incorrect password", vbRetryCancel +
   vbCritical, "Access Denied")
  If Response = vbRetry Then
    txtPassword.SelStart = 0
    txtPassword.SelLength = Len(txtPassword.Text)
  Else
    End
  End If
End If
txtPassword.SetFocus
End Sub
```

This code checks the input password to see if it matches the stored value. If so, it prints an acceptance message. If incorrect, it displays a message box to that effect and asks the user if they want to try again. If Yes (Retry), another try is granted. If No (Cancel), the program is ended. Notice the use of **SelLength** and **SelStart** to highlight an incorrect entry. This allows the user to type right over the incorrect response.

5. Attach the following code to the **Form_Activate** event.

```
Private Sub Form_Activate()
txtPassword.SetFocus
End Sub
```

6. Attach the following code to the **cmdExit_ Click** event.

```
Private Sub cmdExit_Click()
End
End Sub
```

7. Try running the program. Try both options: input correct password (note it is case sensitive) and input incorrect password. Save your project.

   If you have time, define a constant, TRYMAX = 3, and modify the code to allow the user to have just TRYMAX attempts to get the correct password. After the final try, inform the user you are logging him/her off. You I also need a variable that counts the number of tries (make it a Static variable).

Check Boxes

- **Check boxes** provide a way to make choices from a list of potential candidates. Some, all, or none of the choices in a group may be selected.

- Check Box Properties:

Caption	Identifying text next to box.
Font	Sets font type, style, size.
Value	Indicates if unchecked (0, vbUnchecked), checked (1, vbChecked), or grayed out (2, vbGrayed).

- Check Box Events:

Click	Triggered when a box is clicked. Value property is automatically changed by Visual Basic.

Option Buttons

- **Option buttons** provide the capability to make a mutually exclusive choice among a group of potential candidate choices. Hence, option buttons work as a group, only one of which can have a True (or selected) value.

- Option Button Properties:

Caption	Identifying text next to button.
Font	Sets font type, style, size.
Value	Indicates if selected (True) or not (False). Only one option button in a group can be True. One button in each group of option buttons should always be initialized to True at design time.

- Option Button Events:

Click	Triggered when a button is clicked. **Value** property is automatically changed by Visual Basic.

Arrays

- Up to now, we've only worked with regular variables, each having its own unique name. Visual Basic has powerful facilities for handling multi-dimensional variables, or **arrays**. For now, we'll only use single, fixed-dimension arrays.

- Arrays are declared in a manner identical to that used for regular variables. For example, to declare an integer array named **'Items'**, with dimension **9**, at the procedure level, we use:

  ```
  Dim Items(9) as Integer
  ```

  If we want the array variables to retain their value upon leaving a procedure, we use the keyword **Static**:

  ```
  Static Items(9) as Integer
  ```

  At the **form** or **module** level, in the general declaration area of the Code window, use:

  ```
  Dim Items(9) as Integer
  ```

  And, at the module level, for a **global** declaration, use:

  ```
  Global Items(9) as Integer
  ```

- The index on an array variable begins at 0 and ends at the dimensioned value. For example, the **Items** array in the above examples has **ten** elements, ranging from Items(0) to Items(9). You use array variables just like any other variable - just remember to include its name and its index. For example, to set Item(5) equal to 7, you simply write:

  ```
  Item(5) = 7
  ```

Control Arrays

- With some controls, it is very useful to define **control arrays** - it depends on the application. For example, option buttons are almost always grouped in control arrays.

- Control arrays are a convenient way to handle groups of controls that perform a similar function. All of the events available to the single control are still available to the array of controls, the only difference being an argument indicating the index of the selected array element is passed to the event. Hence, instead of writing individual procedures for each control (i.e. not using control arrays), you only have to write one procedure for each array.

- Another advantage to control arrays is that you can add or delete array elements at run-time. You cannot do that with controls (objects) not in arrays. Refer to the **Load** and **Unload** statements in on-line help for the proper way to add and delete control array elements at run-time.

- Two ways to **create** a control array:

  1. Create an individual control and set desired properties. Copy the control using the editor, then paste it on the form. Visual Basic will pop-up a dialog box that will ask you if you wish to create a control array. Respond yes and the array is created.

  2. Create all the controls you wish to have in the array. Assign the desired control array name to the first control. Then, try to name the second control with the same name. Visual Basic will prompt you, asking if you want to create a control array. Answer yes. Once the array is created, rename all remaining controls with that name.

- Once a control array has been created and named, their name and index refer to elements of the array. For example, to set the **Caption** property of element **6** of a label box array named **lblExample**, we would use:

  ```
  lblExample(6).Caption =    his is an example”
  ```

  We'll use control arrays in the next example.

Frames

- We've seen that both option buttons and check boxes work as a group. **Frames** provide a way of grouping related controls on a form. And, in the case of option buttons, frames affect how such buttons operate.

- To group controls in a frame, you first draw the frame. Then, the associated controls must be drawn in the frame. This allows you to move the frame and controls together. And, once a control is drawn within a frame, it can be copied and pasted to create a control array within that frame. To do this, first click on the object you want to copy. **Copy** the object. Then, click on the frame. **Paste** the object. You will be asked if you want to create a control array. Answer **Yes**.
-
- Drawing the controls outside the frame and dragging them in, copying them into a frame, or drawing the frame around existing controls will not result in a proper grouping. It is perfectly acceptable to draw frames within other frames.

- As mentioned, frames affect how option buttons work. Option buttons within a frame work as a **group**, independently of option buttons in other frames. Option buttons on the form, and not in frames, work as another independent group. That is, the form is itself a frame by default. We'll see this in the next example.

- It is important to note that an independent group of option buttons is defined by physical location within frames, not according to naming convention. That is, a control array of option buttons does not work as an independent group just because it is a control array. It would only work as a group if it were the only group of option buttons within a frame or on the form. So, remember physical location, and physical location only, dictates independent operation of option button groups.

- Frame Properties:

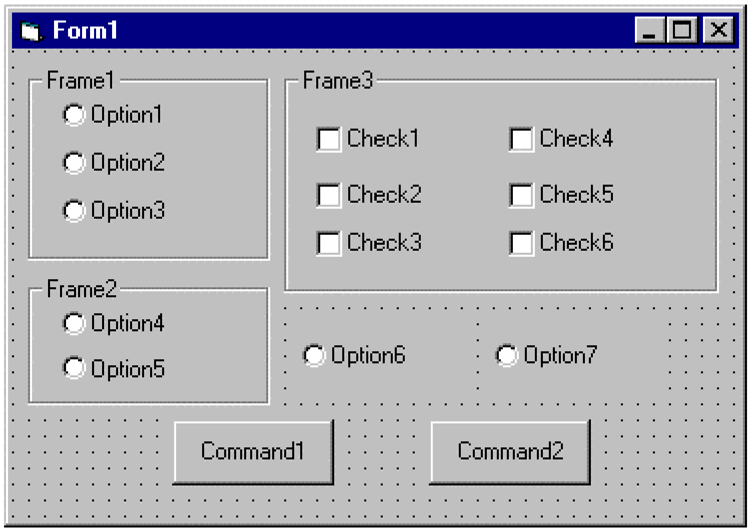
Caption	Title information at top of frame.
Font	Sets font type, style, size.

Example 3-2

Pizza Order

1. Start a new project. We'll build a form where a pizza order can be entered by simply clicking on check boxes and option buttons.

2. Draw three frames. In the first, draw three option buttons, in the second, draw two option buttons, and in the third, draw six check boxes. Draw two option buttons on the form. Add two command buttons. Make things look something like this.



3. Set the properties of the form and each control.

Form1:

BorderStyle	1-Fixed Single
Caption	Pizza Order
Name	frmPizza

Frame1:

Caption	Size

Frame2:

Caption	Crust Type

Frame3

Caption	Toppings

Option1:

Caption	Small
Name	optSize
Value	True

Option2:

Caption	Medium
Name	optSize (yes, create a control array)

Option3:

Caption	Large
Name	optSize

Option4:

Caption	Thin Crust
Name	optCrust
Value	True

Option5:

Caption	Thick Crust
Name	optCrust (yes, create a control array)

Option6:

Caption	Eat In
Name	optWhere
Value	True

Option7:

Caption	Take Out
Name	optWhere (yes, create a control array)

Check1:

Caption	Extra Cheese
Name	chkTop

Check2:

Caption	Mushrooms
Name	chkTop (yes, create a control array)

Check3:

Caption	Black Olives
Name	chkTop

Check4:

Caption	Onions
Name	chkTop

Check5:

Caption	Green Peppers
Name	chkTop

Check6:

Caption	Tomatoes
Name	chkTop

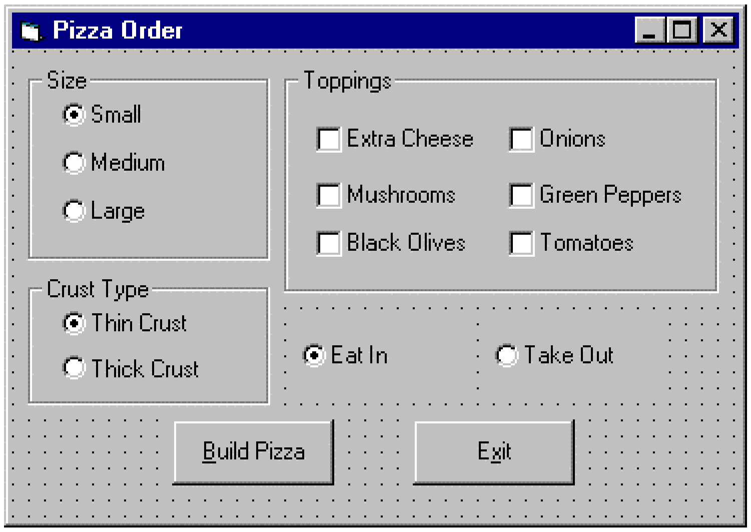
Command1:

Caption	&Build Pizza
Name	cmdBuild

Command2:

Caption	E&xit
Name	cmdExit

The form should look like this now:



4. Declare the following variables in the **general declarations** area:

```
Option Explicit
Dim PizzaSize As String
Dim PizzaCrust As String
Dim PizzaWhere As String
```

This makes the size, crust, and location variables global to the form.

5. Attach this code to the **Form_Load** procedure. This initializes the pizza size, crust, and eating location.

```
Private Sub Form_Load()
'Initialize pizza parameters
PizzaSize = "Small"
PizzaCrust = "Thin Crust"
PizzaWhere = "Eat In"
End Sub
```

Here, the global variables are initialized to their default values, corresponding to the default option buttons.

6. Attach this code to the three option button array **Click** events. Note the use of the Index variable:

```
Private Sub optSize_Click(Index As Integer)
  ead pizza size
PizzaSize = optSize(Index).Caption
End Sub

Private Sub optCrust_Click(Index As Integer)
  ead crust type
PizzaCrust = optCrust(Index).Caption
End Sub

Private Sub optWhere_Click(Index As Integer)
  ead pizza eating location
PizzaWhere = optWhere(Index).Caption
End Sub
```

In each of these routines, when an option button is clicked, the value of the corresponding button   caption is loaded into the respective variable.

7. Attach this code to the **cmdBuild_Click** event.

```
Private Sub cmdBuild_Click()
'This procedure builds a message box that displays your
  pizza type
Dim Message As String
Dim I As Integer
Message = PizzaWhere + vbCr
Message = Message + PizzaSize + " Pizza" + vbCr
Message = Message + PizzaCrust + vbCr
For I = 0 To 5
  If chkTop(I).Value = vbChecked Then Message = Message +
  chkTop(I).Caption + vbCr
Next I
MsgBox Message, vbOKOnly, "Your Pizza"
End Sub
```

This code forms the first part of a message for a message box by concatenating the pizza size, crust type, and eating location (**vbCr** is a symbolic constant representing a arriage return' that puts each piece of ordering information on a separate line). Next, the code cycles through the six topping check boxes and adds any checked information to the message. The code then displays the pizza order in a message box.

8. Attach this code to the **cmdExit_Click** event.

```
Private Sub cmdExit_Click()
End
End Sub
```

9. Get the application working. Notice how the different selection buttons work in their individual groups. Save your project.

10. If you have time, try these modifications:

    A. Add a new program button that resets the order form to the initial default values. You l have to reinitialize the three global variables, reset all check boxes to unchecked, and reset all three option button groups to their default values.

    B. Modify the code so that if no toppings are selected, the message heese Only" appears on the order form. You l need to figure out a way to see if no check boxes were checked.

List Boxes

- A **list box** displays a list of items from which the user can select one or more items. If the number of items exceeds the number that can be displayed, a scroll bar is automatically added.

- List Box Properties:

Appearance	Selects 3-D or flat appearance.
List	Array of items in list box.
ListCount	Number of items in list.
ListIndex	The number of the most recently selected item in list. If no item is selected, ListIndex = -1.
MultiSelect	Controls how items may be selected (0-no multiple selection allowed, 1-multiple selection allowed, 2-group selection allowed).
Selected	Array with elements set equal to True or False, depending on whether corresponding list item is selected.
Sorted	True means items are sorted in 'ASCII' order, else items appear in order added.
Text	Text of most recently selected item.

- List Box Events:

Click	Event triggered when item in list is clicked.
DblClick	Event triggered when item in list is double-clicked. Primary way used to process selection.

- List Box Methods:

AddItem	Allows you to insert item in list.
Clear	Removes all items from list box.
RemoveItem	Removes item from list box, as identified by index of item to remove.

Examples

```
lstExample.AddItem "This is an added item" ' adds text string to list
lstExample.Clear ' clears the list box
lstExample.RemoveItem 4 ' removes lstExample.List(4) from list box
```

- Items in a list box are usually initialized in a Form_Load procedure. It's always a good idea to **Clear** a list box before initializing it.

- You've seen list boxes before. In the standard 'Open File' window, the Directory box is a list box with MultiSelect equal to zero.

Combo Boxes

- The **combo box** is similar to the list box. The differences are a combo box includes a text box on top of a list box and only allows selection of one item. In some cases, the user can type in an alternate response.

- Combo Box Properties:

  Combo box properties are nearly identical to those of the list box, with the deletion of the MultiSelect property and the addition of a Style property.

Appearance	Selects 3-D or flat appearance.
List	Array of items in list box portion.
ListCount	Number of items in list.
ListIndex	The number of the most recently selected item in list. If no item is selected, ListIndex = -1.
Sorted	True means items are sorted in 'Ascii' order, else items appear in order added.
Style	Selects the combo box form.
	Style = 0, Dropdown combo; user can change selection.
	Style = 1, Simple combo; user can change selection.
	Style = 2, Dropdown combo; user cannot change selection.
Text	Text of most recently selected item.

- Combo Box Events:

Click	Event triggered when item in list is clicked.
DblClick	Event triggered when item in list is double-clicked. Primary way used to process selection.

- Combo Box Methods:

AddItem	Allows you to insert item in list.
Clear	Removes all items from list box.
RemoveItem	Removes item from list box, as identified by index of item to remove.

Examples

```
cboExample.AddItem "This is an added item" ' adds text string to list
cboExample.Clear ' clears the combo box
cboExample.RemoveItem 4 ' removes cboExample.List(4) from list
box
```

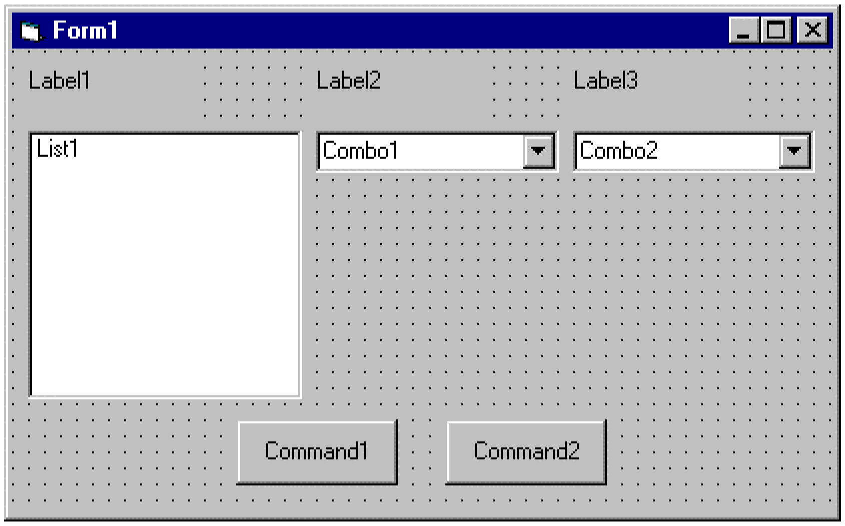
- You've seen combo boxes before. In the standard 'Open File' window, the File Name box is a combo box of Style 2, while the Drive box is a combo box of Style 3.

<u>Example 3-3</u>

Flight Planner

1. Start a new project. In this example, you select a destination city, a seat location, and a meal preference for airline passengers.

2. Place a list box, two combo boxes, three label boxes and two command buttons on the form. The form should appear similar to this:



3. Set the form and object properties:

Form1:

BorderStyle	1-Fixed Single
Caption	Flight Planner
Name	frmFlight

List1:

Name	lstCities
Sorted	True

Combo1:

Name	cboSeat
Style	2-Dropdown List

Combo2:

Name	cboMeal
Style	1-Simple
Text	[Blank]

(After setting properties for this combo box, resize it until it is large enough to hold 4 to 5 entries.)

Label1:

Caption	Destination City

Label2:

Caption	Seat Location

Label3:

Caption	Meal Preference

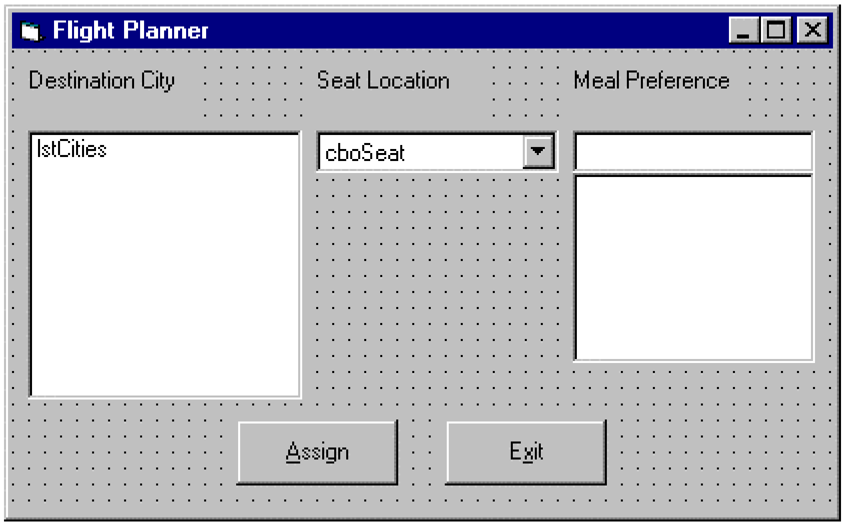
Command1:

Caption	&Assign
Name	cmdAssign

Command2:

Caption	E&xit
Name	cmdExit

Now, the form should look like this:



4. Attach this code to the **Form_Load** procedure:

```
Private Sub Form_Load()
  dd city names to list box
lstCities.Clear
lstCities.AddItem "San Diego"
lstCities.AddItem "Los Angeles"
lstCities.AddItem "Orange County"
lstCities.AddItem "Ontario"
lstCities.AddItem "Bakersfield"
lstCities.AddItem "Oakland"
lstCities.AddItem "Sacramento"
lstCities.AddItem "San Jose"
lstCities.AddItem "San Francisco"
lstCities.AddItem "Eureka"
lstCities.AddItem "Eugene"
lstCities.AddItem "Portland"
lstCities.AddItem "Spokane"
lstCities.AddItem "Seattle"
lstCities.ListIndex = 0

  dd seat types to first combo box
cboSeat.AddItem "Aisle"
cboSeat.AddItem "Middle"
cboSeat.AddItem "Window"
cboSeat.ListIndex = 0

  dd meal types to second combo box
cboMeal.AddItem "Chicken"
cboMeal.AddItem "Mystery Meat"
cboMeal.AddItem "Kosher"
cboMeal.AddItem "Vegetarian"
cboMeal.AddItem "Fruit Plate"
cboMeal.Text = "No Preference"
End Sub
```

This code simply initializes the list box and the list box portions of the two combo boxes.

5. Attach this code to the **cmdAssign_Click** event:

```
Private Sub cmdAssign_Click()
vaild message box that gives your assignment
Dim Message As String
Message = "Destination: " + lstCities.Text + vbCr
Message = Message + "Seat Location: " + cboSeat.Text +
   vbCr
Message = Message + "Meal: " + cboMeal.Text + vbCr
MsgBox Message, vbOKOnly + vbInformation, "Your
   Assignment"
End Sub
```

When the **Assign** button is clicked, this code forms a message box message by concatenating the selected city (from the list box **lstCities**), seat choice (from **cboSeat**), and the meal preference (from **cboMeal**).

6. Attach this code to the **cmdExit_Click** event:

```
Private Sub cmdExit_Click()
End
End Sub
```

7. Run the application. Save the project.

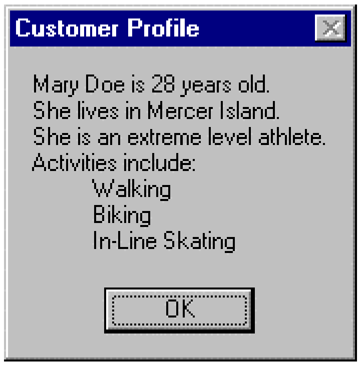
Exercise 3

Customer Database Input Screen

A new sports store wants you to develop an input screen for its customer database. The required input information is:

1. Name
2. Age
3. City of Residence
4. Sex (Male or Female)
5. Activities (Running, Walking, Biking, Swimming, Skiing and/or In-Line Skating)
6. Athletic Level (Extreme, Advanced, Intermediate, or Beginner)

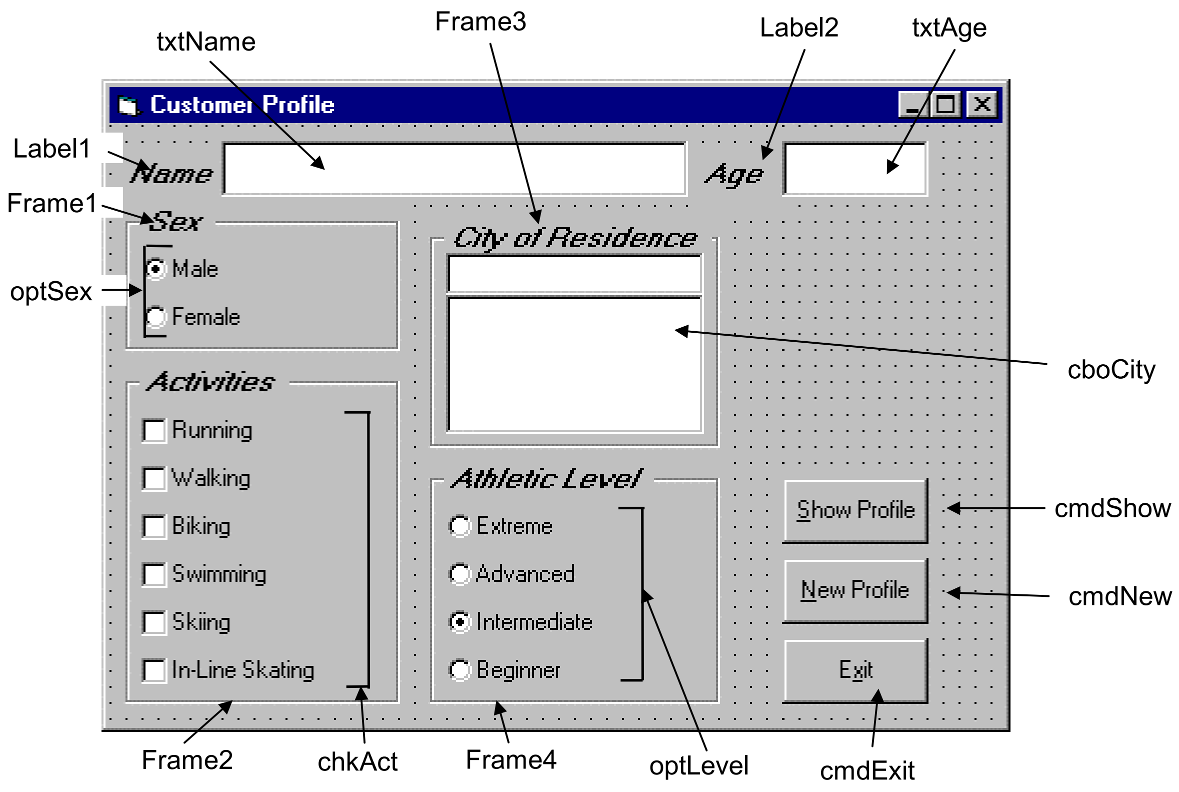
Set up the screen so that only the Name and Age (use text boxes) and, perhaps, City (use a combo box) need to be typed; all other inputs should be set with check boxes and option buttons. When a screen of information is complete, display the summarized profile in a message box. This profile message box should resemble this:



```
Mary Doe is 28 years old.
She lives in Mercer Island.
She is an extreme level athlete.
Activities include:
        Walking
        Biking
        In-Line Skating
```

My Solution:

Form:



Properties:

Form **frmCustomer**:

BorderStyle = 1 - Fixed Single
Caption = Customer Profile

CommandButton **cmdExit**:

Caption = E&xit

Frame **Frame3**:

Caption = City of Residence
FontName = MS Sans Serif
FontBold = True
FontSize = 9.75
FontItalic = True

ComboBox **cboCity**:

Sorted = True
Style = 1 - Simple Combo

```
CommandButton cmdNew:
        Caption = &New Profile

CommandButton cmdShow:
        Caption = &Show Profile

Frame Frame4:
        Caption = Athletic Level
        FontName = MS Sans Serif
        FontBold = True
        FontSize = 9.75
        FontItalic = True

OptionButton optLevel:
        Caption = Beginner
        Index = 3

OptionButton optLevel:
        Caption = Intermediate
        Index = 2
        Value = True

OptionButton optLevel:
        Caption = Advanced
        Index = 1

OptionButton optLevel:
        Caption = Extreme
        Index = 0

Frame Frame1:
        Caption = Sex
        FontName = MS Sans Serif
        FontBold = True
        FontSize = 9.75
        FontItalic = True

OptionButton optSex:
        Caption =   Female
        Index = 1

OptionButton optSex:
        Caption = Male
        Index = 0
        Value = True
```

Frame **Frame2**:
    Caption = Activities
    FontName = MS Sans Serif
    FontBold = True
    FontSize = 9.75
    FontItalic = True

CheckBox **chkAct**:
    Caption = In-Line Skating
    Index = 5

CheckBox **chkAct**:
    Caption = Skiing
    Index = 4

CheckBox **chkAct**:
    Caption = Swimming
    Index = 3

CheckBox **chkAct**:
    Caption = Biking
    Index = 2

CheckBox **chkAct**:
    Caption = Walking
    Index = 1

CheckBox **chkAct**:
    Caption = Running
    Index = 0

TextBox **txtName**:
    FontName = MS Sans Serif
    FontSize = 12

Label **Label1**:
    Caption = Name
    FontName = MS Sans Serif
    FontBold = True
    FontSize = 9.75
    FontItalic = True

TextBox **txtAge**:
    FontName = MS Sans Serif
    FontSize = 12

Label **Label2**:

Caption = Age
FontName = MS Sans Serif
FontBold = True
FontSize = 9.75
FontItalic = True

Code:

General Declarations:

```
Option Explicit
Dim Activity As String
```

cmdExit Click Event:

```
Private Sub cmdExit_Click()
End
End Sub
```

cmdNew Click Event:

```
Private Sub cmdNew_Click()
'Blank out name and reset check boxes
Dim I As Integer
txtName.Text = ""
txtAge.Text = ""
For I = 0 To 5
  chkAct(I).Value = vbUnchecked
Next I
End Sub
```

cmdShow Click Event:

```
Private Sub cmdShow_Click()
Dim NoAct As Integer, I As Integer
Dim Msg As String, Pronoun As String

'Check to make sure name entered
If txtName.Text = "" Then
  MsgBox "The profile requires a name.", vbOKOnly +
vbCritical, "No Name Entered"
  Exit Sub
End If
```

```
'Check to make sure age entered
If txtAge.Text = "" Then
  MsgBox "The profile requires an age.", vbOKOnly +
vbCritical, "No Age Entered"
  Exit Sub
End If

'Put together customer profile message
Msg = txtName.Text + " is" + Str$(txtAge.Text) + " years
old." + vbCr
If optSex(0).Value = True Then Pronoun = "He " Else
Pronoun = "She "
Msg = Msg + Pronoun + "lives in " + cboCity.Text + "." +
vbCr
Msg = Msg + Pronoun + "is a"
If optLevel(3).Value = False Then Msg = Msg + "n " Else
Msg = Msg + " "
Msg = Msg + Activity + " level athlete." + vbCr
NoAct = 0
For I = 0 To 5
  If chkAct(I).Value = vbChecked Then NoAct = NoAct + 1
Next I
If NoAct > 0 Then
  Msg = Msg + "Activities include:" + vbCr
  For I = 0 To 5
    If chkAct(I).Value = vbChecked Then Msg = Msg +
String$(10, 32) + chkAct(I).Caption + vbCr
  Next I
Else
  Msg = Msg + vbCr
End If
MsgBox Msg, vbOKOnly, "Customer Profile"
End Sub
```

Form Load Event:

```
Private Sub Form_Load()
'Load combo box with potential city names
cboCity.AddItem "Seattle"
cboCity.Text = "Seattle"
cboCity.AddItem "Bellevue"
cboCity.AddItem "Kirkland"
cboCity.AddItem "Everett"
cboCity.AddItem "Mercer Island"
cboCity.AddItem "Renton"
cboCity.AddItem "Issaquah"
cboCity.AddItem "Kent"
cboCity.AddItem "Bothell"
cboCity.AddItem "Tukwila"
cboCity.AddItem "West Seattle"
cboCity.AddItem "Edmonds"
cboCity.AddItem "Tacoma"
cboCity.AddItem "Federal Way"
cboCity.AddItem "Burien"
cboCity.AddItem "SeaTac"
cboCity.AddItem "Woodinville"
Activity = "intermediate"
End Sub
```

optLevel Click Event:

```
Private Sub optLevel_Click(Index As Integer)
  etermine activity level
Select Case Index
Case 0
   Activity = "extreme"
Case 1
  Activity = "advanced"
Case 2
  Activity = "intermediate"
Case 3
  Activity = "beginner"
End Select
End Sub
```

txtAge KeyPress Event:

```
Private Sub txtAge_KeyPress(KeyAscii As Integer)
'Only allow numbers for age
If (KeyAscii >= vbKey0 And KeyAscii <= vbKey9) Or
KeyAscii = vbKeyBack Then
  Exit Sub
Else
  KeyAscii = 0
End If
End Sub
```

This page intentionally not left blank.

Learn Visual Basic 6.0

4. More Exploration of the Visual Basic Toolbox

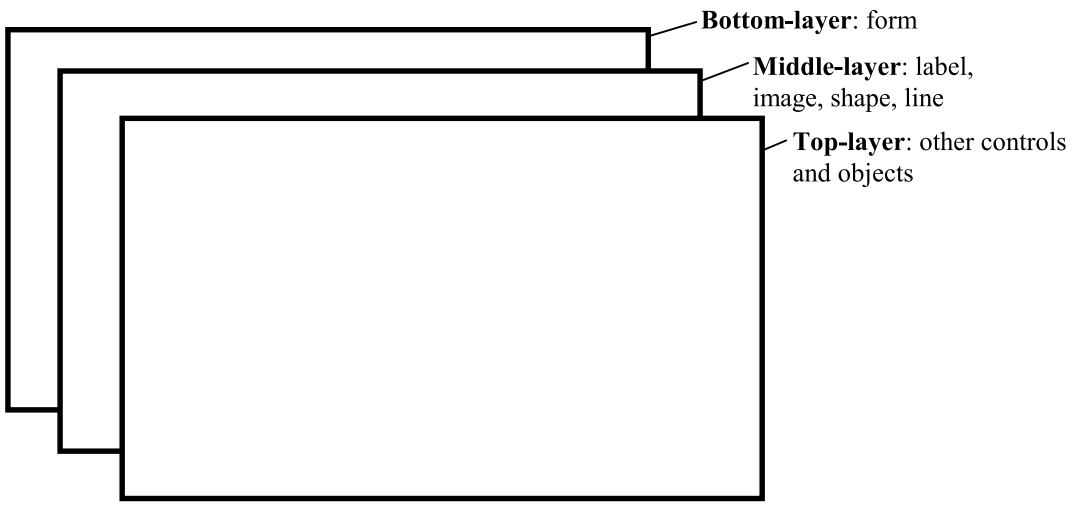
Review and Preview

- In this class, we continue looking at tools in the Visual Basic toolbox. We will look at some drawing tools, scroll bars, and tools that allow direct interaction with drives, directories, and files. In the examples, try to do as much of the building and programming of the applications you can with minimal reference to the notes. This will help you build your programming skills.

Display Layers

- In this class, we will look at our first graphic type controls: line tools, shape tools, picture boxes, and image boxes. And, with this introduction, we need to discuss the idea of display **layers**.

- Items shown on a form are not necessarily all on the same layer of display. A form's display is actually made up of three layers as sketched below. All information displayed directly on the form (by printing or drawing with graphics methods) appears on the **bottom-layer**. Information from label boxes, image boxes, line tools, and shape tools, appears on the **middle-layer**. And, all other objects are displayed on the **top-layer**.


Bottom-layer: form
Middle-layer: label, image, shape, line
Top-layer: other controls and objects

- What this means is you have to be careful where you put things on a form or something could be covered up. For example, a command button placed on top of it would hide text printed on the form. Things drawn with the shape tool are covered by all controls except the image box.

- The next question then is what establishes the relative location of objects in the same layer. That is, say two command buttons are in the same area of a form - which one lies on top of which one? The order in which objects in the same layer overlay each other is called the **Z-order**. This order is first established when you draw the form. Items drawn last lie over items drawn earlier. Once drawn, however, clicking on the desired object and choosing Bring to Front from Visual Basic's Edit menu can modify the Z-order. The **Send to Back** command has the opposite effect. Note these two commands only work within a layer; middle-layer objects will always appear behind top-layer objects and lower layer objects will always appear behind middle-layer objects.

Line Tool

- The **line tool** creates simple straight line segments of various width and color. Together with the shape tool discussed next, you can use this tool to 'dress up' your application.

- Line Tool Properties:

BorderColor	Determines the line color.
BorderStyle	Determines the line 'shape'. Lines can be transparent, solid, dashed, dotted, and combinations.
BorderWidth	Determines line width.

- There are no events or methods associated with the line tool.

- Since the line tool lies in the middle-layer of the form display, any lines drawn will be obscured by all controls except the shape tool or image box.

Shape Tool

- The **shape tool** can create circles, ovals, squares, rectangles, and rounded squares and rectangles. Colors can be used and various fill patterns are available.

- Shape Tool Properties:

BackColor	Determines the background color of the shape (only used when FillStyle not Solid.
BackStyle	Determines whether the background is transparent or opaque.
BorderColor	Determines the color of the shape's outline.
BorderStyle	Determines the style of the shape's outline. The border can be transparent, solid, dashed, dotted, and combinations.
BorderWidth	Determines the width of the shape border line.
FillColor	Defines the interior color of the shape.
FillStyle	Determines the interior pattern of a shape. Some choices are: solid, transparent, cross, etc.
Shape	Determines whether the shape is a square, rectangle, circle, or some other choice.

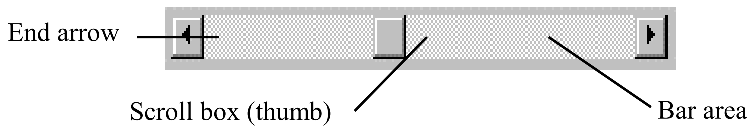
- Like the line tool, events and methods are not used with the shape tool.

- Shapes are covered by all objects except perhaps line tools and image boxes (depends on their Z-order) and printed or drawn information. This is a good feature in that you usually use shapes to contain a group of control objects and you'd want them to lie on top of the shape.

Horizontal and Vertical Scroll Bars

- Horizontal and vertical **scroll bars** are widely used in Windows applications. Scroll bars provide an intuitive way to move through a list of information and make great input devices.

- Both type of scroll bars are comprised of three areas that can be clicked, or dragged, to change the scroll bar value. Those areas are:

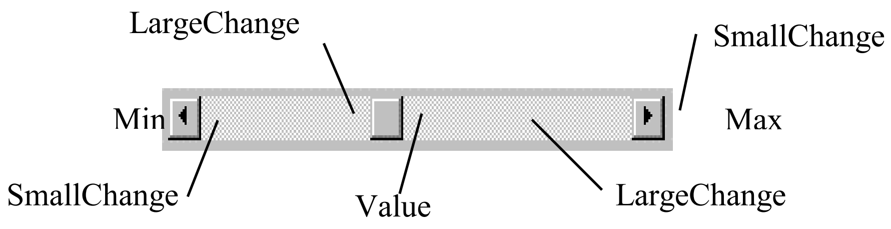


Clicking an **end arrow** increments the **scroll box** a small amount, clicking the **bar area** increments the scroll box a large amount, and dragging the scroll box (thumb) provides continuous motion. Using the properties of scroll bars, we can completely specify how one works. The scroll box position is the only output information from a scroll bar.

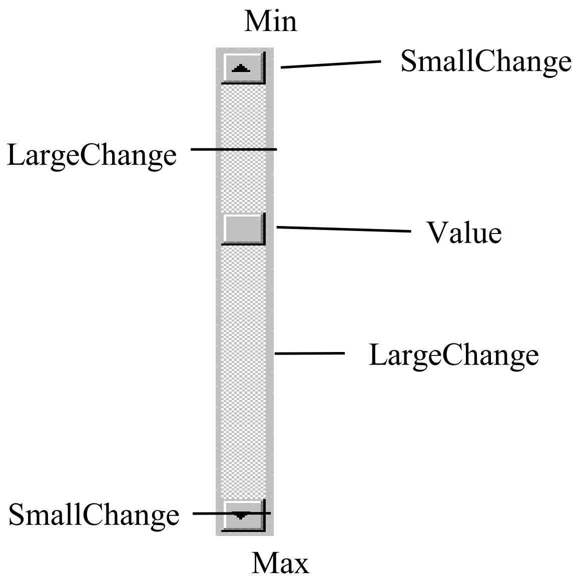
- Scroll Bar Properties:

LargeChange	Increment added to or subtracted from the scroll bar **Value** property when the bar area is clicked.
Max	The value of the horizontal scroll bar at the far right and the value of the vertical scroll bar at the bottom. Can range from -32,768 to 32,767.
Min	The other extreme value - the horizontal scroll bar at the left and the vertical scroll bar at the top. Can range from -32,768 to 32,767.
SmallChange	The increment added to or subtracted from the scroll bar **Value** property when either of the scroll arrows is clicked.
Value	The current position of the scroll box (thumb) within the scroll bar. If you set this in code, Visual Basic moves the scroll box to the proper position.

Properties for horizontal scroll bar:



Properties for vertical scroll bar:



- A couple of important notes about scroll bars:

1. Note that although the extreme values are called **Min** and **Max**, they do not necessarily represent minimum and maximum values. There is nothing to keep the Min value from being greater than the Max value. In fact, with vertical scroll bars, this is the usual case. Visual Basic automatically adjusts the sign on the **SmallChange** and **LargeChange** properties to insure proper movement of the scroll box from one extreme to the other.

2. If you ever change the **Value**, **Min**, or **Max** properties in code, make sure Value is at all times between Min and Max or and the program will stop with an error message.

- Scroll Bar Events:

Change	Event is triggered after the scroll box's position has been modified. Use this event to retrieve the Value property after any changes in the scroll bar.
Scroll	Event triggered continuously whenever the scroll box is being moved.

Example 4-1

Temperature Conversion

Start a new project. In this project, we convert temperatures in degrees Fahrenheit (set using a scroll bar) to degrees Celsius. As mentioned in the **Review and Preview** section, you should try to build this application with minimal reference to the notes. To that end, let's look at the project specifications.

Temperature Conversion Application Specifications

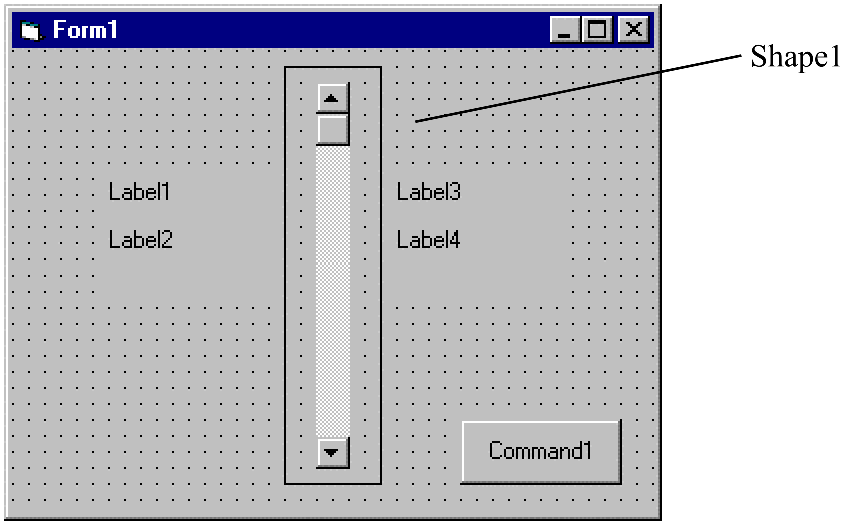
The application should have a scroll bar which adjusts temperature in degrees Fahrenheit from some reasonable minimum to some maximum. As the user changes the scroll bar value, both the Fahrenheit temperature and Celsius temperature (you have to calculate this) in integer format should be displayed. The formula for converting Fahrenheit (F) to Celsius (C) is:

C = (F - 32)*5/9

To convert this number to a rounded integer, use the Visual Basic **CInt**() function. To change numeric information to strings for display in label or text boxes, use the **Str**() or **Format**() function. Try to build as much of the application as possible before looking at my approach. Try incorporating lines and shapes into your application if you can.

One Possible Approach to Temperature Conversion Application:

1. Place a shape, a vertical scroll bar, four labels, and a command button on the form. Put the scroll bar within the shape - since it is in the top-layer of the form, it will lie in the shape. It should resemble this:



2. Set the properties of the form and each object:

Form1:

BorderStyle	1-Fixed Single
Caption	Temperature Conversion
Name	frmTemp

Shape1:

BackColor	White
BackStyle	1-Opaque
FillColor	Red
FillStyle	7-Diagonal Cross
Shape	4-Rounded Rectangle

VScroll1:

LargeChange	10
Max	-60
Min	120
Name	vsbTemp
SmallChange	1
Value	32

Label1:

Alignment	2-Center
Caption	Fahrenheit
FontSize	10
FontStyle	Bold

Label2:

Alignment	2-Center
AutoSize	True
BackColor	White
BorderStyle	1-Fixed Single
Caption	32
FontSize	14
FontStyle	Bold
Name	lblTempF

Label3:

Alignment	2-Center
Caption	Celsius
FontSize	10
FontStyle	Bold

Label4:

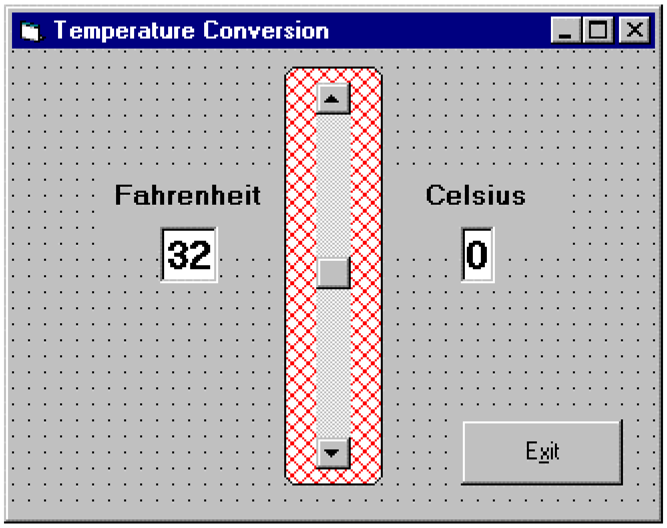
Alignment	2-Center
AutoSize	True
BackColor	White
BorderStyle	1-Fixed Single
Caption	0
FontSize	14
FontStyle	Bold
Name	lblTempC

Command1:

Cancel	True
Caption	E&xit
Name	cmdExit

Note the temperatures are initialized at 32F and 0C, known values.

When done, the form should look like this:



3. Put this code in the general declarations of your code window.

```
Option Explicit
Dim TempF As Integer
Dim TempC As Integer
```

This makes the two temperature variables global.

4. Attach the following code to the scroll bar **Scroll** event.

```
Private Sub vsbTemp_Scroll()
'Read F and convert to C
TempF = vsbTemp.Value
lblTempF.Caption = Str(TempF)
TempC = CInt((TempF - 32) * 5 / 9)
lblTempC.Caption = Str(TempC)
End Sub
```

This code determines the scroll bar Value as it scrolls, takes that value as Fahrenheit temperature, computes Celsius temperature, and displays both values.

5. Attach the following code to the scroll bar **Change** event.

```
Private Sub vsbTemp_Change()
'Read F and convert to C
TempF = vsbTemp.Value
lblTempF.Caption = Str(TempF)
TempC = CInt((TempF - 32) * 5 / 9)
lblTempC.Caption = Str(TempC)
End Sub
```

Note this code is identical to that used in the Scroll event. This is almost always the case when using scroll bars.

6. Attach the following code to the **cmdExit_Click** procedure.

```
Private Sub cmdExit_Click()
End
End Sub
```

7. Give the program a try. Make sure it provides correct information at obvious points. For example, 32 F better always be the same as 0 C! Save the project - we l return to it briefly in Class 5.

   Other things to try:

   A. Can you find a point where Fahrenheit temperature equals Celsius temperature? If you don't know this off the top of your head, it's obvious you've never lived in extremely cold climates. I've actually witnessed one of those bank temperature signs flashing degrees F and degrees C and seeing the same number!

   B. Ever wonder why body temperature is that odd figure of 98.6 degrees F? Can your new application give you some insight to an answer to this question?

   C. It might be interesting to determine how wind affects perceived temperature - the wind chill. Add a second scroll bar to input wind speed and display both the actual and wind adjusted temperatures. You would have to do some research to find the mathematics behind wind chill computations. This is not a trivial extension of the application.

Picture Boxes

- The **picture box** allows you to place graphics information on a form. It is best suited for dynamic environments - for example, when doing animation.

- Picture boxes lie in the top layer of the form display. They behave very much like small forms within a form, possessing most of the same properties as a form.

- Picture Box Properties:

AutoSize	If True, box adjusts its size to fit the displayed graphic.
Font	Sets the font size, style, and size of any printing done in the picture box.
Picture	Establishes the graphics file to display in the picture box.

- Picture Box Events:

Click	Triggered when a picture box is clicked.
DblClick	Triggered when a picture box is double-clicked.

- Picture Box Methods:

Cls	Clears the picture box.
Print	Prints information to the picture box.

Examples

```
picExample.Cls ' clears the box picExample
picExample.Print "a picture box" ' prints text string to picture box
```

- Picture Box LoadPicture Procedure:

  An important function when using picture boxes is the **LoadPicture** procedure. It is used to set the **Picture** property of a picture box at run-time.

  Example

  ```
  picExample.Picture = LoadPicture("c:\pix\sample.bmp")
  ```

  This command loads the graphics file c:\pix\sample.bmp into the Picture property of the picExample picture box. The argument in the LoadPicture function must be a legal, complete path and file name, else your program will stop with an error message.

- Five types of graphics files can be loaded into a picture box:

Bitmap	An image represented by pixels and stored as a collection of bits in which each bit corresponds to one pixel. Usually has a **.bmp** extension. Appears in original size.
Icon	A special type of bitmap file of maximum 32 x 32 size. Has a **.ico** extension. We l create icon files in Class 5. Appears in original size.
Metafile	A file that stores an image as a collection of graphical objects (lines, circles, polygons) rather than pixels. Metafiles preserve an image more accurately than bitmaps when resized. Has a **.wmf** extension. Resizes itself to fit the picture box area.
JPEG	JPEG (Joint Photographic Experts Group) is a compressed bitmap format which supports 8 and 24 bit color. It is popular on the Internet. Has a **.jpg** extension and scales nicely.
GIF	GIF (Graphic Interchange Format) is a compressed bitmap format originally developed by CompuServe. It supports up to 256 colors and is popular on the Internet. Has a **.gif** extension and scales nicely.

Image Boxes

- An **image box** is very similar to a picture box in that it allows you to place graphics information on a form. Image boxes are more suited for static situations - that is, cases where no modifications will be done to the displayed graphics.

- Image boxes appear in the middle-layer of form display, hence picture boxes and other objects could obscure them. Image box graphics can be resized by using the **Stretch** property.

- Image Box Properties:

Picture	Establishes the graphics file to display in the image box.
Stretch	If False, the image box resizes itself to fit the graphic. If True, the graphic resizes to fit the control area.

- Image Box Events:

Click	Triggered when a image box is clicked.
DblClick	Triggered when a image box is double-clicked.

- The image box does not support any methods, however it does use the **LoadPicture** function. It is used in exactly the same manner as the picture box uses it. And image boxes can load the same file types: bitmap (.bmp), icon (.ico), metafiles (.wmf), GIF files (.gif), and JPEG files (.jpg). With Stretch = True, all three graphic types will expand to fit the image box area.

Quick Example: Picture and Image Boxes

1. Start a new project. Draw one picture box and one image box.

2. Set the **Picture** property of the picture and image box to the same file. If you have graphics files installed with Visual Basic, bitmap files can be found in the bitmaps folder, icon files in the icon folder, and metafiles are in the metafile folder.

3. Notice what happens as you resize the two boxes. Notice the layer effect when you move one box on top of the other. Notice the effect of the image box **Stretch** property. Play around with different file types - what differences do you see?

Drive List Box

- The **drive list box** control allows a user to select a valid disk drive at run-time. It displays the available drives in a drop-down combo box. No code is needed to load a drive list box; Visual Basic does this for us. We use the box to get the current drive identification.

- Drive List Box Properties:

Drive	Contains the name of the currently selected drive.

- Drive List Box Events:

Change	Triggered whenever the user or program changes the drive selection.

Directory List Box

- The **directory list box** displays an ordered, hierarchical list of the user's disk directories and subdirectories. The directory structure is displayed in a list box. Like, the drive list box, little coding is needed to use the directory list box - Visual Basic does most of the work for us.

- Directory List Box Properties:

Path	Contains the current directory path.

- Directory List Box Events:

Change	Triggered when the directory selection is changed.

File List Box

- The **file list box** locates and lists files in the directory specified by its Path property at run-time. You may select the types of files you want to display in the file list box.

- File List Box Properties:

FileName	Contains the currently selected file name.
Path	Contains the current path directory.
Pattern	Contains a string that determines which files will be displayed. It supports the use of * and ? wildcard characters. For example, using *.dat only displays files with the .dat extension.

- File List Box Events:

DblClick	Triggered whenever a file name is double-clicked.
PathChange	Triggered whenever the path changes in a file list box.

- You can also use the **MultiSelect** property of the file list box to allow multiple file selection.

Synchronizing the Drive, Directory, and File List Boxes

- The drive, directory, and file list boxes are almost always used together to obtain a file name. As such, it is important that their operation be synchronized to insure the displayed information is always consistent.

- When the drive selection is changed (drive box **Change** event), you should update the directory path. For example, if the drive box is named drvExample and the directory box is dirExample, use the code:

```
dirExample.Path = drvExample.Drive
```

- When the directory selection is changed (directory box **Change** event), you should update the displayed file names. With a file box named filExample, this code is:

```
filExample.Path = dirExample.Path
```

- Once all of the selections have been made and you want the file name, you need to form a text string that correctly and completely specifies the file identifier. This string concatenates the drive, directory, and file name information. This should be an easy task, except for one problem. The problem involves the backslash (\) character. If you are at the root directory of your drive, the path name ends with a backslash. If you are not at the root directory, there is no backslash at the end of the path name and you have to add one before tacking on the file name.

- Example code for concatenating the available information into a proper file name and then loading it into an image box is:

```
Dim YourFile as String

If Right(filExample.Path,1) = "\" Then
  YourFile = filExample.Path + filExample.FileName
Else
  YourFile = filExample.Path + "\" + filExample.FileName
End If
imgExample.Picture = LoadPicture(YourFile)
```

Note we only use properties of the file list box. The drive and directory box properties are only used to create changes in the file list box via code.

Example 4-2

Image Viewer

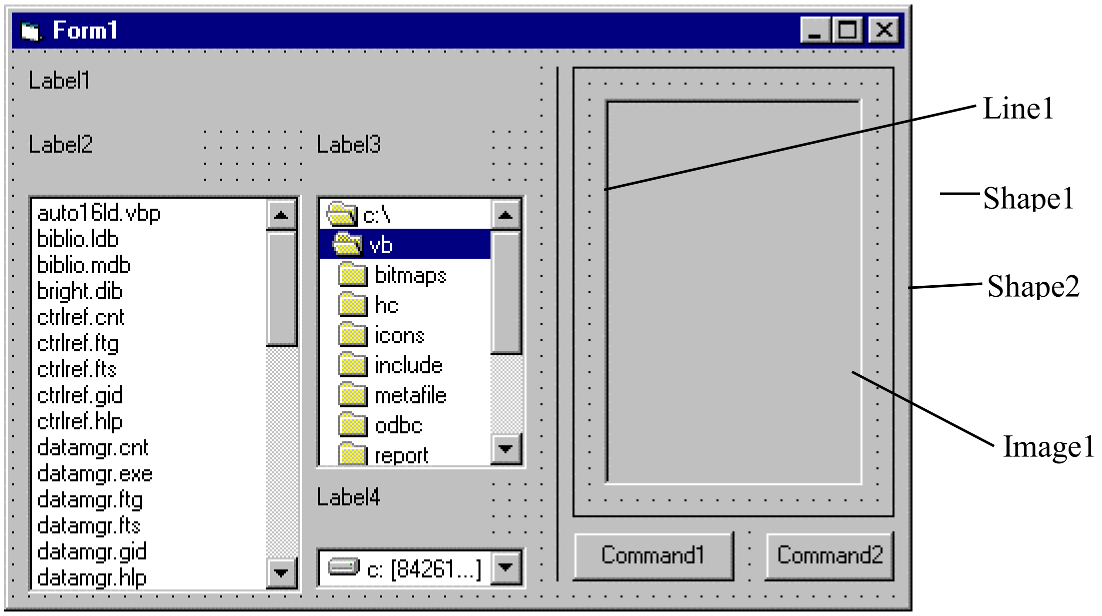
Start a new project. In this application, we search our computer's file structure for graphics files and display the results of our search in an image box.

Image Viewer Application Specifications

Develop an application where the user can search and find graphics files (*.ico, *.bmp, *.wmf) on his/her computer. Once a file is selected, print the corresponding file name on the form and display the graphic file in an image box using the **LoadPicture**() function.

One possible solution to the Image Viewer Application:

1. Place a drive list box, directory list box, file list box, four label boxes, a line (use the line tool) and a command button on the form. We also want to add an image box, but make it look like it's in some kind of frame. Build this display area in these steps: draw a 'large shape', draw another shape within this first shape that is the size of the image display area, and lastly, draw an image box right on top of this last shape. Since the two shapes and image box are in the same display layer, the image box is on top of the second shape which is on top of the first shape, providing the desired effect of a kind of picture frame. The form should look like this:



Note the second shape is directly beneath the image box.

2. Set properties of the form and each object.

Form1:

BorderStyle	1-Fixed Single
Caption	Image Viewer
Name	frmImage

Drive1:

Name	drvImage

Dir1:

Name	dirImage

File1:

Name	filImage
Pattern	*.bmp;*.ico;*.wmf;*gif;*jpg [type this line with <u>no</u> spaces]

Label1:

Caption	[Blank]
BackColor	Yellow
BorderStyle	1-Fixed Single
Name	lblImage

Label2:

Caption	Files:

Label3:

Caption	Directories:

Label4:

Caption	Drives:

Command1:

Caption	&Show Image
Default	True
Name	cmdShow

Command2:

Cancel	True
Caption	E&xit
Name	cmdExit

Line1:

BorderWidth	3

Shape1:

BackColor	Cyan
BackStyle	1-Opaque
FillColor	Blue
FillStyle	4-Upward Diagonal
Shape	4-Rounded Rectangle

Shape2:

BackColor	White
BackStyle	1-Opaque

Image1:

BorderStyle	1-Fixed Single
Name	imgImage
Stretch	True

3. Attach the following code to the **drvImage_Change** procedure.

```
Private Sub drvImage_Change()
'If drive changes, update directory
dirImage.Path = drvImage.Drive
End Sub
```

When a new drive is selected, this code forces the directory list box to display directories on that drive.

4. Attach this code to the **dirImage_Change** procedure.

```
Private Sub dirImage_Change()
'If directory changes, update file path
filImage.Path = dirImage.Path
End Sub
```

Likewise, when a new directory is chosen, we want to see the files on that directory.

5. Attach this code to the **cmdShow_Click** event.

```
Private Sub cmdShow_Click()
'Put image file name together and
'load image into image box
Dim ImageName As String
'Check to see if at root directory
If Right(filImage.Path, 1) = "\" Then
  ImageName = filImage.Path + filImage.filename
Else
  ImageName = filImage.Path + "\" + filImage.filename
End If
lblImage.Caption = ImageName
imgImage.Picture = LoadPicture(ImageName)
End Sub
```

This code forms the file name (**ImageName**) by concatenating the directory path with the file name. It then displays the complete name and loads the picture into the image box.

6. Copy the code from the **cmdShow_Click** procedure and paste it into the **filImage_DblClick** procedure. The code is identical because we want to display the image either by double-clicking on the filename or clicking the command button once a file is selected. Those of you who know how to call routines in Visual Basic should note that this duplication of code is unnecessary - we could simply have the **filImage_DblClick** procedure call the **cmdShow_Click** procedure. We l learn more about this next class.

7. Attach this code to the **cmdExit_Click** procedure.

```
Private Sub cmdExit_Click()
End
End Sub
```

8. Save your project. Run and try the application. Find bitmaps, icons, and metafiles. Notice how the image box Stretch property affects the different graphics file types. Here how the form should look when displaying one example metafile:

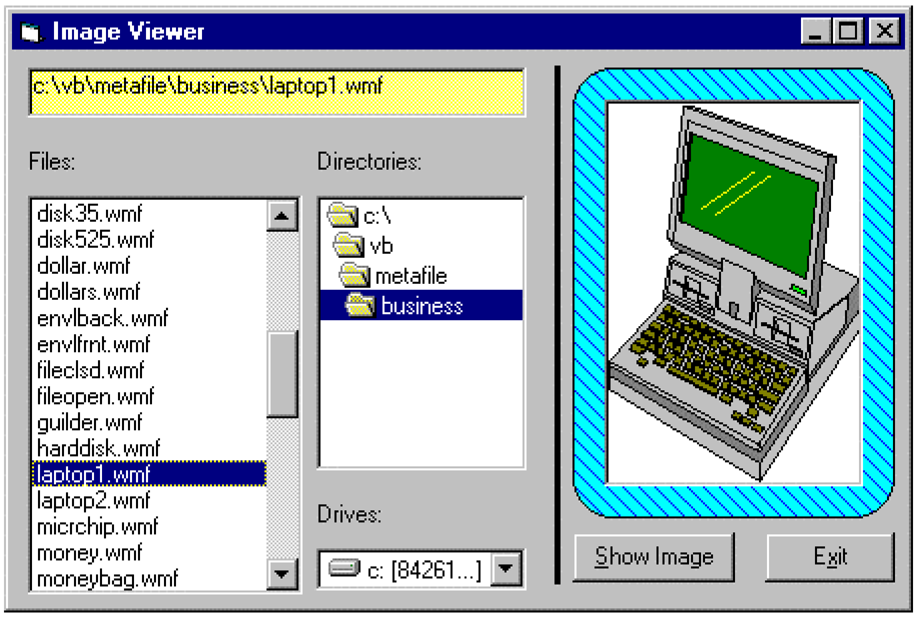


Image Viewer

c:\vb\metafile\business\laptop1.wmf

Files:

disk35.wmf
disk525.wmf
dollar.wmf
dollars.wmf
envlback.wmf
envlfrnt.wmf
fileclsd.wmf
fileopen.wmf
guilder.wmf
harddisk.wmf
laptop1.wmf
laptop2.wmf
micrchip.wmf
money.wmf
moneybag.wmf

Directories:

c:\
vb
metafile
business

Drives:

c: [84261...]

Common Dialog Boxes

- The primary use for the drive, directory, and file name list boxes is to develop custom file access routines. Two common file access routines in Windows-based applications are the **Open File** and **Save File** operations. Fortunately, you don have to build these routines.

- To give the user a standard interface for common operations in Windows-based applications, Visual Basic provides a set of **common dialog boxes**, two of which are the **Open** and **Save As** dialog boxes. Such boxes are familiar to any Windows user and give your application a professional look. And, with Windows 95, some context-sensitive help is available while the box is displayed. Appendix II lists many symbolic constants used with common dialog boxes.

- The Common Dialog control is a **custom control**' which means we have to make sure some other files are present to use it. In normal setup configurations, Visual Basic does this automatically. If the common dialog box does not appear in the Visual Basic toolbox, you need to add it. Selecting Components under the Project menu does this. When the selection box appears, click on **Microsoft Common Dialog Control**, then click **OK**.

- The common dialog tool, although it appears on your form, is invisible at run-time. You cannot control where the common dialog box appears on your screen. The tool is invoked at run-time using one of five **how'** methods. These methods are:

Method	**Common Dialog Box**
ShowOpen	Open dialog box
ShowSave	Save As dialog box
ShowColor	Color dialog box
ShowFont	Font dialog box
ShowPrinter	Printer dialog box

- The format for establishing a common dialog box named **cdlExample** so that an **Open** box appears is:

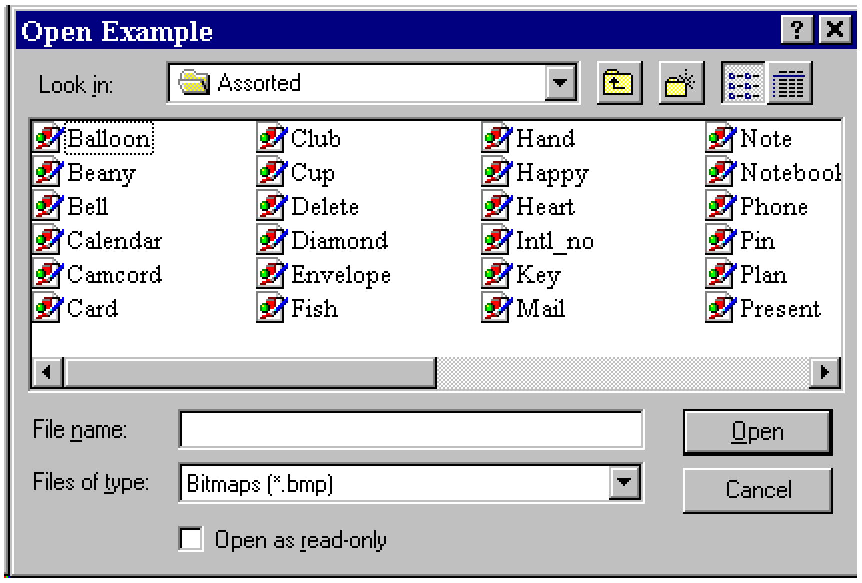
cdlExample.ShowOpen

Control to the program returns to the line immediately following this line, once the dialog box is closed in some manner. Common dialog boxes are system modal.

- Learning proper use of all the common dialog boxes would require an extensive amount of time. In this class, we l limit ourselves to learning the basics of getting file names from the **Open** and **Save As** boxes in their default form.

Open Common Dialog Box

- The **Open** common dialog box provides the user a mechanism for specifying the name of a file to open. We l worry about how to open a file in Class 6. The box is displayed by using the **ShowOpen** method. Here an example of an Open common dialog box:



Open Example

Look in: Assorted

Balloon, Beany, Bell, Calendar, Camcord, Card, Club, Cup, Delete, Diamond, Envelope, Fish, Hand, Happy, Heart, Intl_no, Key, Mail, Note, Noteboo, Phone, Pin, Plan, Present

File name:

Files of type: Bitmaps (*.bmp)

Open as read-only

- Open Dialog Box Properties:

CancelError	If True, generates an error if the Cancel button is clicked. Allows you to use error-handling procedures to recognize that Cancel was clicked.
DialogTitle	The string appearing in the title bar of the dialog box. Default is Open. In the example, the DialogTitle is Open Example.
FileName	Sets the initial file name that appears in the File name box. After the dialog box is closed, this property can be read to determine the name of the selected file.
Filter	Used to restrict the filenames that appear in the file list box. Complete filter specifications for forming a Filter can be found using on-line help. In the example, the Filter was set to allow Bitmap (*.bmp), Icon (*.ico), Metafile (*.wmf), GIF (*.gif), and JPEG (*.jpg) types (only the Bitmap choice is seen).

FilterIndex	Indicates which filter component is default. The example uses a 1 for the FilterIndex (the default value).
Flags	Values that control special features of the Open dialog box (see Appendix II). The example uses no Flags value.

- When the user closes the Open File box, you should check the returned file name to make sure it meets the specifications your application requires before you try to open the file.

Quick Example: The Open Dialog Box

1. Start a new project. Place a common dialog control, a label box, and a command button on the form. Set the following properties:

Form1:

Caption	Common Dialog Examples
Name	frmCommon

CommonDialog1:

DialogTitle	Open Example
Filter	Bitmaps (*.bmp)|*.bmp| Icons (*.ico)|*.ico|Metafiles (*.wmf)|*.wmf GIF Files (*.gif)|*.gif|JPEG Files (*,jpg)|*.jpg (all on one line)
Name	cdlExample

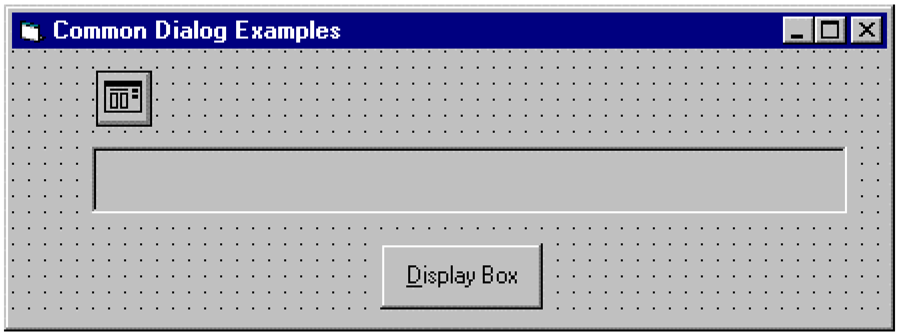
Label1:

BorderStyle	1-Fixed Single
Caption	[Blank]
Name	lblExample

Command1:

Caption	&Display Box
Name	cmdDisplay

When done, the form should look like this (make sure your label box is very long):



2. Attach this code to the **cmdDisplay_Click** procedure.

```
Private Sub cmdDisplay_Click()
cdlExample.ShowOpen
lblExample.Caption = cdlExample.filename
End Sub
```

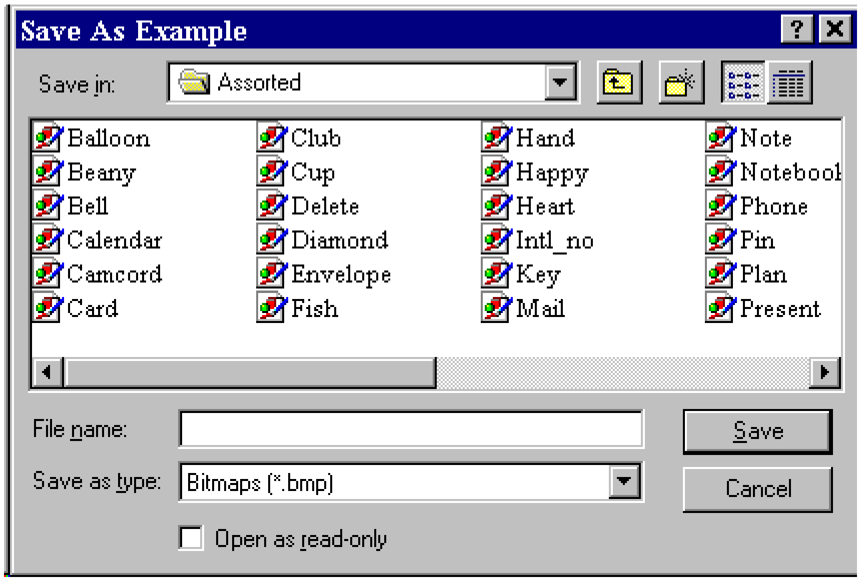
   This code brings up the Open dialog box when the button is clicked and shows the file name selected by the user once it is closed.

3. Save the application. Run it and try selecting file names and typing file names. Notice names can be selected by highlighting and clicking the **OK** button or just by double-clicking the file name. In this example, clicking the **Cancel** button is not trapped, so it has the same effect as clicking **OK**.

4. Notice once you select a file name, the next time you open the dialog box, that selected name appears as default, since the FileName property is not affected in code.

Save As Common Dialog Box

- The **Save As** common dialog box provides the user a mechanism for specifying the name of a file to save. We l worry about how to save a file in Class 6. The box is displayed by using the **ShowSave** method.. Here an example of a Save As common dialog box:



- Save As Dialog Box Properties (mostly the same as those for the Open box):

CancelError	If True, generates an error if the Cancel button is clicked. Allows you to use error-handling procedures to recognize that Cancel was clicked.
DefaultExt	Sets the default extension of a file name if a file is listed without an extension.
DialogTitle	The string appearing in the title bar of the dialog box. Default is Save As. In the example, the DialogTitle is Save As Example.
FileName	Sets the initial file name that appears in the File name box. After the dialog box is closed, this property can be read to determine the name of the selected file.
Filter	Used to restrict the filenames that appear in the file list box.
FilterIndex	Indicates which filter component is default.
Flags	Values that control special features of the dialog box (see Appendix II).

- The Save File box is commonly configured in one of two ways. If a file is being saved for the first time, the **Save As** configuration, with some default name in the FileName property, is used. In the **Save** configuration, we assume a file has been previously opened with some name. Hence, when saving the file again, that same name should appear in the FileName property. You e seen both configuration types before.

- When the user closes the Save File box, you should check the returned file name to make sure it meets the specifications your application requires before you try to save the file. Be especially aware of whether the user changed the file extension to something your application does not allow.

Quick Example: The Save As Dialog Box

1. We l just modify the Open example a bit. Change the **DialogTitle** property of the common dialog control to **save As Example**" and set the **DefaultExt** property equal to **bmp".**

2. In the **cmdDisplay_Click** procedure, change the method to **ShowSave** (opens Save As box).

3. Save the application and run it. Try typing names without extensions and note how **.bmp** is added to them. Notice you can also select file names by double-clicking them or using the **OK** button. Again, the **Cancel** button is not trapped, so it has the same effect as clicking **OK**.

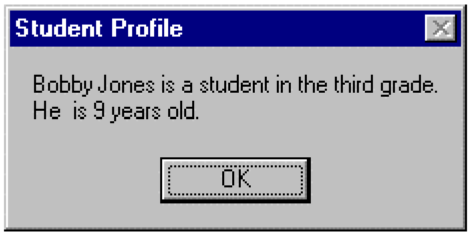
Exercise 4

Student Database Input Screen

You did so well with last week assignment that, now, a school wants you to develop the beginning structure of an input screen for its students. The required input information is:

1. Student Name
2. Student Grade (1 through 6)
3. Student Sex (Male or Female)
4. Student Date of Birth (Month, Day, Year)
5. Student Picture (Assume they can be loaded as bitmap files)

Set up the screen so that only the Name needs to be typed; all other inputs should be set with option buttons, scroll bars, and common dialog boxes. When a screen of information is complete, display the summarized profile in a message box. This profile message box should resemble this:

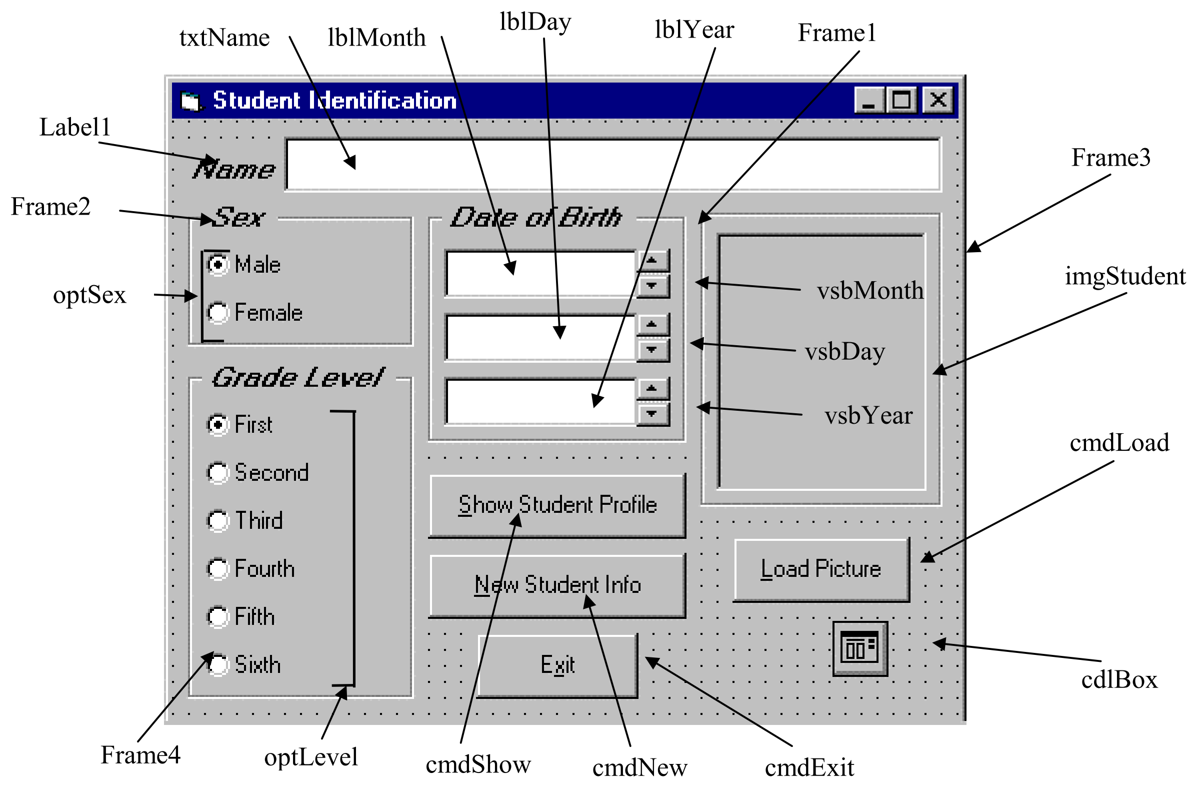


Bobby Jones is a student in the third grade.
He is 9 years old.

Note the student age must be computed from the input birth date - watch out for pitfalls in doing the computation. The student picture does not appear in the profile, only on the input screen.

My Solution:

Form:



Properties:

Form **frmStudent:**

BorderStyle = 1- Fixed Single
Caption = Student Profile

CommandButton **cmdLoad**:

Caption = &Load Picture

Frame **Frame3**:

Caption = Picture
FontName = MS Sans Serif
FontBold = True
FontSize = 9.75
FontItalic = True

Image **imgStudent**:
    BorderStyle = 1 - Fixed Single
    Stretch = True

CommandButton **cmdExit**:
    Caption = E&xit

CommandButton **cmdNew**:
    Caption = &New Profile

CommandButton **cmdShow**:
    Caption = &Show Profile

Frame **Frame4**:
    Caption = Grade Level
    FontName = MS Sans Serif
    FontBold = True
    FontSize = 9.75
    FontItalic = True

OptionButton **optLevel**:
    Caption = Grade 6
    Index = 5

OptionButton **optLevel**:
    Caption = Grade 5
    Index = 4

OptionButton **optLevel**:
    Caption = Grade 4
    Index = 3

OptionButton **optLevel**:
    Caption = Grade 3
    Index = 2

OptionButton **optLevel**:
    Caption = Grade 2
    Index = 1

OptionButton **optLevel**:
    Caption = Grade 1
    Index = 0

Frame **Frame2**:

    Caption = Sex
    FontName = MS Sans Serif
    FontBold = True
    FontSize = 9.75
    FontItalic = True

OptionButton **optSex**:

    Caption = Female
    Index = 1

OptionButton **optSex**:

    Caption = Male
    Index = 0

Frame **Frame1**:

    Caption = Date of Birth
    FontName = MS Sans Serif
    FontBold = True
    FontSize = 9.75
    FontItalic = True

VScrollBar **vsbYear**:

    Max = 1800
    Min = 2100
    Value = 1960

VScrollBar **vsbDay**:

    Max = 1
    Min = 31
    Value = 1

VScrollBar **vsbMonth**:

    Max = 1
    Min = 12
    Value = 1

Label **lblYear**:

    Alignment = 2 - Center
    BackColor = &H00FFFFFF& (White)
    BorderStyle = 1 - Fixed Single
    FontName = MS Sans Serif
    FontSize = 10.8

Label **lblDay**:

Alignment = 2 - Center
BackColor = &H00FFFFFF& (White)
BorderStyle = 1 - Fixed Single
FontName = MS Sans Serif
FontSize = 10.8

Label **lblMonth**:

Alignment = 2 - Center
BackColor = &H00FFFFFF& (White)
BorderStyle = 1 - Fixed Single
FontName = MS Sans Serif
FontSize = 10.8

TextBox **txtName**:

FontName = MS Sans Serif
FontSize = 10.8

CommonDialog **cdlBox**:

Filter = Bitmaps (*.bmp)|*.bmp

Label **Label1**:

Caption = Name
FontName = MS Sans Serif
FontBold = True
FontSize = 9.75
FontItalic = True

Code:

General Declarations:

```
Option Explicit
Dim Months(12) As String
Dim Days(12) As Integer
Dim Grade As String
```

cmdExit Click Event:

```
Private Sub cmdExit_Click()
End
End Sub
```

cmdLoad Click Event:

```
Private Sub cmdLoad_Click()
cdlbox.ShowOpen
imgStudent.Picture = LoadPicture(cdlbox.filename)
End Sub
```

cmdNew Click Event:

```
Private Sub cmdNew_Click()
'Blank out name and picture
txtName.Text = ""
imgStudent.Picture = LoadPicture("")
End Sub
```

cmdShow Click Event:

```
Private Sub cmdShow_Click()
Dim Is_Leap As Integer
Dim Msg As String, Age As Integer, Pronoun As String
Dim M As Integer, D As Integer, Y As Integer

'Check for leap year and if February is current month
If vsbMonth.Value = 2 And ((vsbYear.Value Mod 4 = 0 And
vsbYear.Value Mod 100 <> 0) Or vsbYear.Value Mod 400 = 0)
Then
  Is_Leap = 1
Else
  Is_Leap = 0
End If
'Check to make sure current day doesn't exceed number of
days in month
If vsbDay.Value > Days(vsbMonth.Value) + Is_Leap Then
  MsgBox "Only" + Str(Days(vsbMonth.Value) + Is_Leap) + "
days in " + Months(vsbMonth.Value), vbOKOnly +
vbCritical, "Invalid Birth Date"
  Exit Sub
End If
'Get current date to compute age
M = Val(Format(Now, "mm"))
D = Val(Format(Now, "dd"))
Y = Val(Format(Now, "yyyy"))
Age = Y - vsbYear
If vsbMonth.Value > M Or (vsbMonth.Value = M And vsbDay >
D) Then Age = Age - 1
'Check for valid age
If Age < 0 Then
```

```
  MsgBox "Birth date is before current date.", vbOKOnly +
vbCritical, "Invalid Birth Date"
  Exit Sub
End If

'Check to make sure name entered
If txtName.Text = "" Then
  MsgBox "The profile requires a name.", vbOKOnly +
vbCritical, "No Name Entered"
  Exit Sub
End If

'Put together student profile message
Msg = txtName.Text + " is a student in the " + Grade + "
grade." + vbCr
If optSex(0).Value = True Then Pronoun = "He " Else
Pronoun = "She "
Msg = Msg + Pronoun + " is" + Str(Age) + " years old." +
vbCr
MsgBox Msg, vbOKOnly, "Student Profile"
End Sub
```

Form Load Event:

```
Private Sub Form_Load()
'Set arrays for dates and initialize labels
Months(1) = "January": Days(1) = 31
Months(2) = "February": Days(2) = 28
Months(3) = "March": Days(3) = 31
Months(4) = "April": Days(4) = 30
Months(5) = "May": Days(5) = 31
Months(6) = "June": Days(6) = 30
Months(7) = "July": Days(7) = 31
Months(8) = "August": Days(8) = 31
Months(9) = "September": Days(9) = 30
Months(10) = "October": Days(10) = 31
Months(11) = "November": Days(11) = 30
Months(12) = "December": Days(12) = 31
lblMonth.Caption = Months(vsbMonth.Value)
lblDay.Caption = Str(vsbDay.Value)
lblYear.Caption = Str(vsbYear.Value)
Grade = "first"
End Sub
```

optLevel Click Event:

```
Private Sub optLevel_Click(Index As Integer)
Select Case Index
Case 0
   Grade = "first"
Case 1
   Grade = "second"
Case 2
   Grade = "third"
Case 3
   Grade = "fourth"
Case 4
   Grade = "fifth"
Case 5
   Grade = "sixth"
End Select
End Sub
```

vsbDay Change Event:

```
Private Sub vsbDay_Change()
lblDay.Caption = Str(vsbDay.Value)
End Sub
```

vsbMonth Change Event:

```
Private Sub vsbMonth_Change()
lblMonth.Caption = Months(vsbMonth.Value)
End Sub
```

vsbYear Change Event:

```
Private Sub vsbYear_Change()
lblYear.Caption = Str(vsbYear.Value)
End Sub
```

Learn Visual Basic 6.0

Appendix II. Common Dialog Box Constants

CommonDialog Control Constants

File Open/Save Dialog Box Flags

Constant	Value	Description
cdlOFNReadOnly	0x1	Checks Read-Only check box for Open and Save As dialog boxes.
cdlOFNOverwritePrompt	0x2	Causes the Save As dialog box to generate a message box if the selected file already exists.
cdlOFNHideReadOnly	0x4	Hides the Read-Only check box.
cdlOFNNoChangeDir	0x8	Sets the current directory to what it was when the dialog box was invoked.
cdlOFNHelpButton	0x10	Causes the dialog box to display the Help button.
cdlOFNNoValidate	0x100	Allows invalid characters in the returned filename.
cdlOFNAllowMultiselect	0x200	Allows the File Name list box to have multiple selections.
cdlOFNExtensionDifferent	0x400	The extension of the returned filename is different from the extension set by the DefaultExt property.
cdlOFNPathMustExist	0x800	User can enter only valid path names.
cdlOFNFileMustExist	0x1000	User can enter only names of existing files.
cdlOFNCreatePrompt	0x2000	Sets the dialog box to ask if the user wants to create a file that doesn't currently exist.

File Open/Save Dialog Box Flags (continued)

Constant	Value	Description
cdlOFNShareAware	0x4000	Sharing violation errors will be ignored.
cdlOFNNoReadOnlyReturn	0x8000	The returned file doesn't have the Read-Only attribute set and won't be in a write-protected directory.
cdlOFNExplorer	0x0008000	Use the Explorer-like Open A File dialog box template. (Windows 95 only.)
cdlOFNNoDereferenceLinks	0x00100000	Do not dereference shortcuts (shell links) default, choosing a shortcut causes it to be dereferenced by the shell. (Windows 95 only.)
cdlOFNLongNames	0x00200000	Use Long filenames. (Windows 95 only.)

Color Dialog Box Flags

Constant	Value	Description
cdlCCRGBInit	0x1	Sets initial color value for the dialog box.
cdlCCFullOpen	0x2	Entire dialog box is displayed, including the Define Custom Colors section.
cdlCCPreventFullOpen	0x4	Disables the Define Custom Colors section of the dialog box.
cdlCCHelpButton	0x8	Dialog box displays a Help button.

Fonts Dialog Box Flags

Constant	Value	Description
cdlCFScreenFonts	0x1	Dialog box lists only screen fonts supported by the system.
cdlCFPrinterFonts	0x2	Dialog box lists only fonts supported by the printer.
cdlCFBoth	0x3	Dialog box lists available screen and printer fonts.
cdlCFHelpButton	0x4	Dialog box displays a Help button.
cdlCFEffects	0x100	Dialog box enables strikeout, underline, and color effects.
cdlCFApply	0x200	Dialog box enables the Apply button.
cdlCFANSIOnly	0x400	Dialog box allows only a selection of fonts that use the Windows character set.
cdlCFNoVectorFonts	0x800	Dialog box should not allow vector-font selections.

Fonts Dialog Box Flags (continued)

Constant	Value	Description
cdlCFNoSimulations	0x1000	Dialog box should not allow graphic device interface (GDI)
cdlCFLimitSize	0x2000	Dialog box should select only font sizes within the range specified by the Min and Max properties.
cdlCFFixedPitchOnly	0x4000	Dialog box should select only fixed-pitch fonts.
cdlCFWYSIWYG	0x8000	Dialog box should allow only the selection of fonts available to both the screen and printer.
cdlCFForceFontExist	0x10000	An error dialog box is displayed if a user selects a font or style that doesn't exist.
cdlCFScalableOnly	0x20000	Dialog box should allow only the selection of scalable fonts.
cdlCFTTOnly	0x40000	Dialog box should allow only the selection of TrueType fonts.
cdlCFNoFaceSel	0x80000	No font name selected.
cdlCFNoStyleSel	0x100000	No font style selected.
cdlCFNoSizeSel	0x200000	No font size selected.

Printer Dialog Box Flags

Constant	Value	Description
cdlPDAllPages	0x0	Returns or sets state of All Pages option button.
cdlPDCollate	0x10	Returns or sets state of Collate check box.
cdlPDDisablePrintToFile	0x80000	Disables the Print To File check box.
cdlPDHidePrintToFile	0x100000	The Print To File check box isn't displayed.
cdlPDNoPageNums	0x8	Returns or sets the state of the Pages option button.
cdlPDNoSelection	0x4	Disables the Selection option button.
cdlPDNoWarning	0x80	Prevents a warning message when there is no default printer.
cdlPDPageNums	0x2	Returns or sets the state of the Pages option button.
cdlPDPrintSetup	0x40	Displays the Print Setup dialog box rather than the Print dialog box.

Printer Dialog Box Flags (continued)

Constant	**Value**	**Description**
cdlPDPrintToFile	0x20	Returns or sets the state of the Print To File check box.
cdlPDReturnDC	0x100	Returns a device context for the printer selection value returned in the hDC property of the dialog box.
cdlPDReturnDefault	0x400	Returns default printer name.
cdlPDReturnIC	0x200	Returns an information context for the printer selection value returned in the hDC property of the dialog box.
cdlPDSelection	0x1	Returns or sets the state of the Selection option button.
cdlPDHelpButton	0x800	Dialog box displays the Help button.
cdlPDUseDevModeCopies	0x40000	Sets support for multiple copies action; depends upon whether or not printer supports multiple copies.

CommonDialog Error Constants

Constant	Value	Description
cdlAlloc	&H7FF0&	Couldn't allocate memory for FileName or Filter property.
cdlCancel	&H7FF3&	Cancel was selected.
cdlDialogFailure	&H8000&	The function failed to load the dialog box.
cdlFindResFailure	&H7FF9&	The function failed to load a specified resource.
cdlHelp	&H7FEF&	Call to Windows Help failed.
cdlInitialization	&H7FFD&	The function failed during initialization.
cdlLoadResFailure	&H7FF8&	The function failed to load a specified string.
cdlLockResFailure	&H7FF7&	The function failed to lock a specified resource.
cdlMemAllocFailure	&H7FF6&	The function was unable to allocate memory for internal data structures.
cdlMemLockFailure	&H7FF5&	The function was unable to lock the memory associated with a handle.
cdlNoFonts	&H5FFE&	No fonts exist.
cdlBufferTooSmall	&H4FFC&	The buffer at which the member lpstrFile points is too small.
cdlInvalidFileName	&H4FFD&	Filename is invalid.
cdlSubclassFailure	&H4FFE&	An attempt to subclass a list box failed due to insufficient memory.
cdlCreateICFailure	&H6FF5&	The PrintDlg function failed when it attempted to create an information context.
cdlDndmMismatch	&H6FF6&	Data in the DevMode and DevNames data structures describe two different printers.
cdlGetDevModeFail	&H6FFA&	The printer device driver failed to initialize a DevMode data structure.
cdlInitFailure	&H6FF9&	The PrintDlg function failed during initialization.
cdlLoadDrvFailure	&H6FFB&	The PrintDlg function failed to load the specified printer's device driver.

CommonDialog Error Constants (continued)

Constant	Value	Description
cdlNoDefaultPrn	&H6FF7&	A default printer doesn't exist.
cdlNoDevices	&H6FF8&	No printer device drivers were found.
cdlParseFailure	&H6FFD&	The CommonDialog function failed to parse the strings in the [devices] section of WIN.INI.
cdlPrinterCodes	&H6FFF&	The PDReturnDefault flag was set, but either the hDevMode or hDevNames field was nonzero.
cdlPrinterNotFound	&H6FF4&	The [devices] section of WIN.INI doesn't contain an entry for the requested printer.
cdlRetDefFailure	&H6FFC&	The PDReturnDefault flag was set, but either the hDevMode or hDevNames field was nonzero.
cdlSetupFailure	&H6FFE&	Failed to load required resources.

Learn Visual Basic 6.0

Appendix I. Visual Basic Symbolic Constants

Contents

Alignment Constants

Align Property

Constant	Value	Description
vbAlignNone	0	Size and location set at design time or in code.
vbAlignTop	1	Picture box at top of form.
vbAlignBottom	2	Picture box at bottom of form.
vbAlignLeft	3	Picture box at left of form.
vbAlignRight	4	Picture box at right of form.

Alignment Property

Constant	Value	Description
vbLeftJustify	0	Left align.
vbRightJustify	1	Right align.
vbCenter	2	Center.

Border Property Constants

BorderStyle Property (Form)

Constant	Value	Description
vbBSNone	0	No border.
vbFixedSingle	1	Fixed single.
vbSizable	2	Sizable (forms only)
vbFixedDouble	3	Fixed double (forms only)

BorderStyle Property (Shape and Line)

Constant	Value	Description
vbTransparent	0	Transparent.
vbBSSolid	1	Solid.
vbBSDash	2	Dash.
vbBSDot	3	Dot.
vbBSDashDot	4	Dash-dot.
vbBSDashDotDot	5	Dash-dot-dot.
vbBSInsideSolid	6	Inside solid.

Clipboard Object Constants

Constant	Value	Description
vbCFLink	0xBF00	DDE conversation information.
vbCFRTF	0xBF01	Rich Text Format (.RTF file)
vbCFText	1	Text (.TXT file)
vbCFBitmap	2	Bitmap (.BMP file)
vbCFMetafile	3	Metafile (.WMF file)
vbCFDIB	8	Device-independent bitmap.
vbCFPalette	9	Color palette.

Color Constants

Colors

Constant	Value	Description
vbBlack	0x0	Black.
vbRed	0xFF	Red.
vbGreen	0xFF00	Green.
vbYellow	0xFFFF	Yellow.
vbBlue	0xFF0000	Blue.
vbMagenta	0xFF00FF	Magenta.
vbCyan	0xFFFF00	Cyan.
vbWhite	0xFFFFFF	White.

System Colors

Constant	Value	Description
vbScrollBars	0x80000000	Scroll bar color.
vbDesktop	0x80000001	Desktop color.
vbActiveTitleBar	0x80000002	Color of the title bar for the active window.
vbInactiveTitleBar	0x80000003	Color of the title bar for the inactive window.
vbMenuBar	0x80000004	Menu background color.
vbWindowBackground	0x80000005	Window background color.
vbWindowFrame	0x80000006	Window frame color.
vbMenuText	0x80000007	Color of text on menus.
vbWindowText	0x80000008	Color of text in windows.
vbTitleBarText	0x80000009	Color of text in caption, size box, and scroll arrow.
vbActiveBorder	0x8000000A	Border color of active window.
vbInactiveBorder	0x8000000B	Border color of inactive window.
vbApplicationWorkspace	0x8000000C	Background color of multiple-document interface (MDI)

System Colors (continued)

Constant	Value	Description
vbHighlight	0x8000000D	Background color of items selected in a control.
vbHighlightText	0x8000000E	Text color of items selected in a control.
vbButtonFace	0x8000000F	Color of shading on the face of command buttons.
vbButtonShadow	0x80000010	Color of shading on the edge of command buttons.
vbGrayText	0x80000011	Grayed (disabled)
vbButtonText	0x80000012	Text color on push buttons.
vbInactiveCaptionText	0x80000013	Color of text in an inactive caption.
vb3DHighlight	0x80000014	Highlight color for 3D display elements.
vb3DDKShadow	0x80000015	Darkest shadow color for 3D display elements.
vb3DLight	0x80000016	Second lightest of the 3D colors after vb3DHighlight.
vbInfoText	0x80000017	Color of text in ToolTips.
vbInfoBackground	0x80000018	Background color of ToolTips.

Control Constants

ComboBox Control

Constant	Value	Description
vbComboDropdown	0	Dropdown Combo.
vbComboSimple	1	Simple Combo.
vbComboDropdownList	2	Dropdown List.

ListBox Control

Constant	Value	Description
vbMultiSelectNone	0	None.
vbMultiSelectSimple	1	Simple.
vbMultiSelectExtended	2	Extended.

ScrollBar Control

Constant	Value	Description
vbSBNone	0	None.
vbHorizontal	1	Horizontal.
vbVertical	2	Vertical.
vbBoth	3	Both.

Shape Control

Constant	Value	Description
vbShapeRectangle	0	Rectangle.
vbShapeSquare	1	Square.
vbShapeOval	2	Oval.
vbShapeCircle	3	Circle.
vbShapeRoundedRectangle	4	Rounded rectangle.
vbShapeRoundedSquare	5	Rounded square.

Data Control Constants

Error Event Constants

Constant	Value	Description
vbDataErrContinue	0	Continue.
vbDataErrDisplay	1	(Default)

EditMode Property Constants

Constant	Value	Description
vbDataEditNone	0	No editing operation in progress.
vbDataEditMode	1	Edit method invoked; current record in copy buffer.
vbDataEditAdd	2	AddNew method invoked; current record hasn't been saved.

Options Property Constants

Constant	Value	Description
vbDataDenyWrite	1	Other users can't change records in recordset.
vbDataDenyRead	2	Other users can't read records in recordset.
vbDataReadOnly	4	No user can change records in recordset.
vbDataAppendOnly	8	New records can be added to the recordset, but existing records can't be read.
vbDataInconsistent	16	Updates can apply to all fields of the recordset.
vbDataConsistent	32	Updates apply only to those fields that will not affect other records in the recordset.
vbDataSQLPassThrough	64	Sends an SQL statement to an ODBC database.

Validate Event Action Constants

Constant	Value	Description
vbDataActionCancel	0	Cancel the operation when the Sub exits.
vbDataActionMoveFirst	1	MoveFirst method.
vbDataActionMovePrevious	2	MovePrevious method.
vbDataActionMoveNext	3	MoveNext method.
vbDataActionMoveLast	4	MoveLast method.
vbDataActionAddNew	5	AddNew method.
vbDataActionUpdate	6	Update operation (not UpdateRecord)
vbDataActionDelete	7	Delete method.
vbDataActionFind	8	Find method.
vbDataActionBookmark	9	The Bookmark property is set.
vbDataActionClose	10	Close method.
vbDataActionUnload	11	The form is being unloaded.

Beginning-of-File Constants

Constant	Value	Description
vbMoveFirst	0	Move to first record.
vbBOF	1	Move to beginning of file.

End-of-File Constants

Constant	Value	Description
vbMoveLast	0	Move to last record.
vbEOF	1	Move to end of file.
vbAddNew	2	Add new record to end of file.

Recordset-Type Constants

Constant	Value	Description
vbRSTypeTable	0	Table-type recordset.
vbRSTypeDynaset	1	Dynaset-type recordset.
vbRSTypeSnapShot	2	Snapshot-type recordset.

Date Constants

***firstdayofweek* Argument Values**

Constant	Value	Description
vbUseSystem	0	Use NLS API setting.
vbSunday	1	Sunday
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

***firstweekofyear* Argument Values**

Constant	Value	Description
vbUseSystem	0	Use application setting if one exists; otherwise use NLS API setting.
vbFirstJan1	1	Start with week in which January 1 occurs (default)
vbFirstFourDays	2	Start with the first week that has at least four days in the new year.
vbFirstFullWeek	3	Start with the first full week of the year.

Return Values

Constant	Value	Description
vbSunday	1	Sunday
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

DBGrid Control Constants

Alignment Constants		
Constant	**Value**	**Description**
dbgLeft	0	Left.
dbgRight	1	Right.
dbgCenter	2	Center.
dbgGeneral	3	General.

BorderStyle Constants		
Constant	**Value**	**Description**
dbgNone	0	None.
dbgFixedSingle	1	FixedSingle.

DataMode Constants		
Constant	**Value**	**Description**
dbgBound	0	Bound.
dbgUnbound	1	Unbound.

DividerStyle Constants

Constant	**Value**	**Description**
dbgNoDividers	0	NoDividers.
dbgBlackLine	1	BlackLine.
dbgDarkGrayLine	2	DarkGrayLine.
dbgRaised	3	Raised.
dbgInset	4	Inset.
dbgUseForeColor	5	UseForeColor.

RowDividerStyle Constants

Constant	**Value**	**Description**
dbgNoDividers	0	NoDividers.
dbgBlackLine	1	BlackLine.
dbgDarkGrayLine	2	DarkGrayLine.
dbgRaised	3	Raised.
dbgInset	4	Inset.
dbgUseForeColor	5	UseForeColor.

Scroll Bar Constants

Constant	**Value**	**Description**
dbgNone	0	None.
dbgHorizontal	1	Horizontal.
dbgVertical	2	Vertical.
dbgBoth	3	Both.
dbgAutomatic	4	Automatic.

DDE Constants

linkerr (LinkError Event)

Constant	Value	Description
vbWrongFormat	1	Another application requested data in wrong format.
vbDDESourceClosed	6	Destination application attempted to continue after source closed.
vbTooManyLinks	7	All source links are in use.
vbDataTransferFailed	8	Failure to update data in destination.

LinkMode Property (Forms and Controls)

Constant	Value	Description
vbLinkNone	0	None.
vbLinkSource	1	Source (forms only)
vbLinkAutomatic	1	Automatic (controls only)
vbLinkManual	2	Manual (controls only)
vbLinkNotify	3	Notify (controls only)

Dir, GetAttr, and SetAttr Constants

Constant	Value	Description
vbNormal	0	Normal (default for Dir and SetAttr)
vbReadOnly	1	Read-only.
vbHidden	2	Hidden.
vbSystem	4	System file.
vbVolume	8	Volume label.
vbDirectory	16	Directory.
vbArchive	32	File has changed since last backup.

Drag-and-Drop Constants

DragOver Event

Constant	Value	Description
vbEnter	0	Source control dragged into target.
vbLeave	1	Source control dragged out of target.
vbOver	2	Source control dragged from one position in target to another.

Drag Method (Controls)

Constant	Value	Description
vbCancel	0	Cancel drag operation.
vbBeginDrag	1	Begin dragging control.
vbEndDrag	2	Drop control.

DragMode Property

Constant	Value	Description
vbManual	0	Manual.
vbAutomatic	1	Automatic.

Drawing Constants

DrawMode Property

Constant	Value	Description
vbBlackness	1	Black.
vbNotMergePen	2	Not Merge pen.
vbMaskNotPen	3	Mask Not pen.
vbNotCopyPen	4	Not Copy pen.
vbMaskPenNot	5	Mask pen Not.
vbInvert	6	Invert.
vbXorPen	7	Xor pen.
vbNotMaskPen	8	Not Mask pen.
vbMaskPen	9	Mask pen.
vbNotXorPen	10	Not Xor pen.
vbNop	11	No operation; output remains unchanged.
vbMergeNotPen	12	Merge Not pen.
vbCopyPen	13	Copy pen.
vbMergePenNot	14	Merge pen Not.
vbMergePen	15	Merge pen.
vbWhiteness	16	White.

DrawStyle Property

Constant	Value	Description
vbSolid	0	Solid.
vbDash	1	Dash.
vbDot	2	Dot.
vbDashDot	3	Dash-dot.
vbDashDotDot	4	Dash-dot-dot.
vbInvisible	5	Invisible.
vbInsideSolid	6	Inside solid.

Form Constants

Show Parameters

Constant	Value	Description
vbModal	1	Modal form.
vbModeless	0	Modeless form.

Arrange Method for MDI Forms

Constant	Value	Description
vbCascade	0	Cascade all nonminimized MDI child forms.
vbTileHorizontal	1	Horizontally tile all nonminimized MDI child forms.
vbTileVertical	2	Vertically tile all nonminimized MDI child forms.
vbArrangeIcons	3	Arrange icons for minimized MDI child forms.

WindowState Property

Constant	Value	Description
vbNormal	0	Normal.
vbMinimized	1	Minimized.
vbMaximized	2	Maximized.

Graphics Constants

FillStyle Property

Constant	Value	Description
vbFSSolid	0	Solid.
vbFSTransparent	1	Transparent.
vbHorizontalLine	2	Horizontal line.
vbVerticalLine	3	Vertical line.
vbUpwardDiagonal	4	Upward diagonal.
vbDownwardDiagonal	5	Downward diagonal.
vbCross	6	Cross.
vbDiagonalCross	7	Diagonal cross.

ScaleMode Property

Constant	Value	Description
vbUser	0	User.
vbTwips	1	Twips.
vbPoints	2	Points.
vbPixels	3	Pixels.
vbCharacters	4	Characters.
vbInches	5	Inches.
vbMillimeters	6	Millimeters.
vbCentimeters	7	Centimeters.

Grid Control Constants

ColAlignment, FixedAlignment Properties

Constant	Value	Description
grdAlignCenter	2	Center data in column.
grdAlignLeft	0	Left-align data in column.
grdAlignRight	1	Right-align data in column.

FillStyle Property

Constant	Value	Description
grdSingle	0	Changing Text property setting affects only active cell.
grdRepeat	1	Changing Text property setting affects all selected cells.

Help Constants

Constant	Value	Description
cdlHelpContext	0x1	Displays Help for a particular topic.
cdlHelpQuit	0x2	Notifies the Help application that the specified Help file is no longer in use.
cdlHelpIndex	0x3	Displays the index of the specified Help file.
cdlHelpContents	0x3	Displays the contents topic in the current Help file.
cdlHelpHelpOnHelp	0x4	Displays Help for using the Help application itself.
cdlHelpSetIndex	0x5	Sets the current index for multi-index Help.
cdlHelpSetContents	0x5	Designates a specific topic as the contents topic.
cdlHelpContextPopup	0x8	Displays a topic identified by a context number.
cdlHelpForceFile	0x9	Creates a Help file that displays text in only one font.
cdlHelpKey	0x101	Displays Help for a particular keyword.
cdlHelpCommandHelp	0x102	Displays Help for a particular command.
cdlHelpPartialKey	0x105	Calls the search engine in Windows Help.

Key Code Constants

Key Codes

Constant	Value	Description
vbKeyLButton	0x1	Left mouse button.
vbKeyRButton	0x2	Right mouse button.
vbKeyCancel	0x3	CANCEL key.
vbKeyMButton	0x4	Middle mouse button.
vbKeyBack	0x8	BACKSPACE key.
vbKeyTab	0x9	TAB key.
vbKeyClear	0xC	CLEAR key.
vbKeyReturn	0xD	ENTER key.
vbKeyShift	0x10	SHIFT key.
vbKeyControl	0x11	CTRL key.
vbKeyMenu	0x12	MENU key.

Key Codes (continued)

Constant	Value	Description
vbKeyPause	0x13	PAUSE key.
vbKeyCapital	0x14	CAPS LOCK key.
vbKeyEscape	0x1B	ESC key.
vbKeySpace	0x20	SPACEBAR key.
vbKeyPageUp	0x21	PAGE UP key.
vbKeyPageDown	0x22	PAGE DOWN key.
vbKeyEnd	0x23	END key.
vbKeyHome	0x24	HOME key.
vbKeyLeft	0x25	LEFT ARROW key.
vbKeyUp	0x26	UP ARROW key.
vbKeyRight	0x27	RIGHT ARROW key.
vbKeyDown	0x28	DOWN ARROW key.
vbKeySelect	0x29	SELECT key.
vbKeyPrint	0x2A	PRINT SCREEN key.
vbKeyExecute	0x2B	EXECUTE key.
vbKeySnapshot	0x2C	SNAPSHOT key.
vbKeyInsert	0x2D	INS key.
vbKeyDelete	0x2E	DEL key.
vbKeyHelp	0x2F	HELP key.
vbKeyNumlock	0x90	NUM LOCK key.

KeyA Through KeyZ Are the Same as Their ASCII Equivalents: 'A' Through 'Z'

Constant	Value	Description
vbKeyA	65	A key.
vbKeyB	66	B key.
vbKeyC	67	C key.
vbKeyD	68	D key.
vbKeyE	69	E key.
vbKeyF	70	F key.
vbKeyG	71	G key.
vbKeyH	72	H key.
vbKeyI	73	I key.
vbKeyJ	74	J key.
vbKeyK	75	K key.
vbKeyL	76	L key.
vbKeyM	77	M key.
vbKeyN	78	N key.
vbKeyO	79	O key.
vbKeyP	80	P key.
vbKeyQ	81	Q key.
vbKeyR	82	R key.
vbKeyS	83	S key.

vbKeyT	84	T key.

KeyA Through KeyZ (continued)

Constant	Value	Description
vbKeyU	85	U key.
vbKeyV	86	V key.
vbKeyW	87	W key.
vbKeyX	88	X key.
vbKeyY	89	Y key.
vbKeyZ	90	Z key.

Key0 Through Key9 Are the Same as Their ASCII Equivalents: '0' Through '9'

Constant	Value	Description
vbKey0	48	0 key.
vbKey1	49	1 key.
vbKey2	50	2 key.
vbKey3	51	3 key.
vbKey4	52	4 key.
vbKey5	53	5 key.
vbKey6	54	6 key.
vbKey7	55	7 key.
vbKey8	56	8 key.
vbKey9	57	9 key.

Keys on the Numeric Keypad

Constant	Value	Description
vbKeyNumpad0	0x60	0 key.
vbKeyNumpad1	0x61	1 key.
vbKeyNumpad2	0x62	2 key.
vbKeyNumpad3	0x63	3 key.
vbKeyNumpad4	0x64	4 key.
vbKeyNumpad5	0x65	5 key.
vbKeyNumpad6	0x66	6 key.
vbKeyNumpad7	0x67	7 key.
vbKeyNumpad8	0x68	8 key.
vbKeyNumpad9	0x69	9 key.
vbKeyMultiply	0x6A	MULTIPLICATION SIGN (*)
vbKeyAdd	0x6B	PLUS SIGN (+)
vbKeySeparator	0x6C	ENTER key.
vbKeySubtract	0x6D	MINUS SIGN (-)
vbKeyDecimal	0x6E	DECIMAL POINT (.)
vbKeyDivide	0x6F	DIVISION SIGN (/)

Function Keys

Constant	Value	Description
vbKeyF1	0x70	F1 key.
vbKeyF2	0x71	F2 key.
vbKeyF3	0x72	F3 key.
vbKeyF4	0x73	F4 key.
vbKeyF5	0x74	F5 key.
vbKeyF6	0x75	F6 key.
vbKeyF7	0x76	F7 key.
vbKeyF8	0x77	F8 key.
vbKeyF9	0x78	F9 key.
vbKeyF10	0x79	F10 key.
vbKeyF11	0x7A	F11 key.
vbKeyF12	0x7B	F12 key.
vbKeyF13	0x7C	F13 key.
vbKeyF14	0x7D	F14 key.
vbKeyF15	0x7E	F15 key.
vbKeyF16	0x7F	F16 key.

Menu Accelerator Constants

Constant	Value	Description
vbMenuAccelCtrlA	1	User-defined shortcut keystrokes.
vbMenuAccelCtrlB	2	User-defined shortcut keystrokes.
vbMenuAccelCtrlC	3	User-defined shortcut keystrokes.
vbMenuAccelCtrlD	4	User-defined shortcut keystrokes.
vbMenuAccelCtrlE	5	User-defined shortcut keystrokes.
vbMenuAccelCtrlF	6	User-defined shortcut keystrokes.
vbMenuAccelCtrlG	7	User-defined shortcut keystrokes.
vbMenuAccelCtrlH	8	User-defined shortcut keystrokes.
vbMenuAccelCtrlI	9	User-defined shortcut keystrokes.
vbMenuAccelCtrlJ	10	User-defined shortcut keystrokes.
vbMenuAccelCtrlK	11	User-defined shortcut keystrokes.

Menu Accelerator Constants (continued)

Constant	Value	Description
vbMenuAccelCtrlL	12	User-defined shortcut keystrokes.
vbMenuAccelCtrlM	13	User-defined shortcut keystrokes.
vbMenuAccelCtrlN	14	User-defined shortcut keystrokes.
vbMenuAccelCtrlO	15	User-defined shortcut keystrokes.
vbMenuAccelCtrlP	16	User-defined shortcut keystrokes.
vbMenuAccelCtrlQ	17	User-defined shortcut keystrokes.
vbMenuAccelCtrlR	18	User-defined shortcut keystrokes.
vbMenuAccelCtrlS	19	User-defined shortcut keystrokes.
vbMenuAccelCtrlT	20	User-defined shortcut keystrokes.
vbMenuAccelCtrlU	21	User-defined shortcut keystrokes.
vbMenuAccelCtrlV	22	User-defined shortcut keystrokes.
vbMenuAccelCtrlW	23	User-defined shortcut keystrokes.
vbMenuAccelCtrlX	24	User-defined shortcut keystrokes.
vbMenuAccelCtrlY	25	User-defined shortcut keystrokes.
vbMenuAccelCtrlZ	26	User-defined shortcut keystrokes.
vbMenuAccelF1	27	User-defined shortcut keystrokes.
vbMenuAccelF2	28	User-defined shortcut keystrokes.
vbMenuAccelF3	29	User-defined shortcut keystrokes.
vbMenuAccelF4	30	User-defined shortcut keystrokes.
vbMenuAccelF5	31	User-defined shortcut keystrokes.
vbMenuAccelF6	32	User-defined shortcut keystrokes.
vbMenuAccelF7	33	User-defined shortcut keystrokes.

Menu Accelerator Constants (continued)

Constant	Value	Description
vbMenuAccelF8	34	User-defined shortcut keystrokes.
vbMenuAccelF9	35	User-defined shortcut keystrokes.
vbMenuAccelF11	36	User-defined shortcut keystrokes.
vbMenuAccelF12	37	User-defined shortcut keystrokes.
vbMenuAccelCtrlF1	38	User-defined shortcut keystrokes.
vbMenuAccelCtrlF2	39	User-defined shortcut keystrokes.
vbMenuAccelCtrlF3	40	User-defined shortcut keystrokes.
vbMenuAccelCtrlF4	41	User-defined shortcut keystrokes.
vbMenuAccelCtrlF5	42	User-defined shortcut keystrokes.
vbMenuAccelCtrlF6	43	User-defined shortcut keystrokes.
vbMenuAccelCtrlF7	44	User-defined shortcut keystrokes.
vbMenuAccelCtrlF8	45	User-defined shortcut keystrokes.
vbMenuAccelCtrlF9	46	User-defined shortcut keystrokes.
vbMenuAccelCtrlF11	47	User-defined shortcut keystrokes.
vbMenuAccelCtrlF12	48	User-defined shortcut keystrokes.
vbMenuAccelShiftF1	49	User-defined shortcut keystrokes.
vbMenuAccelShiftF2	50	User-defined shortcut keystrokes.
vbMenuAccelShiftF3	51	User-defined shortcut keystrokes.
vbMenuAccelShiftF4	52	User-defined shortcut keystrokes.
vbMenuAccelShiftF5	53	User-defined shortcut keystrokes.
vbMenuAccelShiftF6	54	User-defined shortcut keystrokes.
vbMenuAccelShiftF7	55	User-defined shortcut keystrokes.

Menu Accelerator Constants (continued)

Constant	Value	Description
vbMenuAccelShiftF8	56	User-defined shortcut keystrokes.
vbMenuAccelShiftF9	57	User-defined shortcut keystrokes.
vbMenuAccelShiftF11	58	User-defined shortcut keystrokes.
vbMenuAccelShiftF12	59	User-defined shortcut keystrokes.
vbMenuAccelShiftCtrlF1	60	User-defined shortcut keystrokes.
vbMenuAccelShiftCtrlF2	61	User-defined shortcut keystrokes.
vbMenuAccelShiftCtrlF3	62	User-defined shortcut keystrokes.
vbMenuAccelShiftCtrlF4	63	User-defined shortcut keystrokes.
vbMenuAccelShiftCtrlF5	64	ser-defined shortcut keystrokes.
vbMenuAccelShiftCtrlF6	65	User-defined shortcut keystrokes.
vbMenuAccelShiftCtrlF7	66	User-defined shortcut keystrokes.
vbMenuAccelShiftCtrlF8	67	User-defined shortcut keystrokes.
vbMenuAccelShiftCtrlF9	68	ser-defined shortcut keystrokes.
vbMenuAccelShiftCtrlF11	69	User-defined shortcut keystrokes.
vbMenuAccelShiftCtrlF12	70	User-defined shortcut keystrokes.
vbMenuAccelCtrlIns	71	User-defined shortcut keystrokes.
vbMenuAccelShiftIns	72	User-defined shortcut keystrokes.
vbMenuAccelDel	73	User-defined shortcut keystrokes.
vbMenuAccelShiftDel	74	User-defined shortcut keystrokes.
vbMenuAccelAltBksp	75	User-defined shortcut keystrokes.

Menu Control Constants

PopupMenu Method Alignment

Constant	Value	Description
vbPopupMenuLeftAlign	0	Pop-up menu left-aligned.
vbPopupMenuCenterAlign	4	Pop-up menu centered.
vbPopupMenuRightAlign	8	Pop-up menu right-aligned.

PopupMenu Mouse Button Recognition

Constant	Value	Description
vbPopupMenuLeftButton	0	Pop-up menu recognizes left mouse button only.
vbPopupMenuRightButton	2	Pop-up menu recognizes right and left mouse buttons.

Miscellaneous Constants

ZOrder Method

Constant	Value	Description
vbBringToFront	0	Bring to front.
vbSendToBack	1	Send to back.

QueryUnload Method

Constant	Value	Description
vbAppWindows	2	Current Windows session ending.
vbFormMDIForm	4	MDI child form is closing because the MDI form is closing.
vbFormCode	1	Unload method invoked from code.
vbFormControlMenu	0	User has chosen Close command from the Control-menu box on a form.
vbAppTaskManager	3	Windows Task Manager is closing the application.

Shift Parameter Masks

Constant	Value	Description
vbShiftMask	1	SHIFT key bit mask.
vbCtrlMask	2	CTRL key bit mask.
vbAltMask	4	ALT key bit mask.

Button Parameter Masks

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

Application Start Mode

Constant	Value	Description
vbSModeStandalone	0	Stand-alone application.
vbSModeAutomation	1	OLE automation server.

LoadResPicture Method

Constant	Value	Description
vbResBitmap	0	Bitmap resource.
vbResIcon	1	Icon resource.
vbResCursor	2	Cursor resource.

Check Value

Constant	Value	Description
vbUnchecked	0	Unchecked.
vbChecked	1	Checked.
vbGrayed	2	Grayed.

Mouse Pointer Constants

Constant	Value	Description
vbDefault	0	Default.
vbArrow	1	Arrow.
vbCrosshair	2	Cross.
vbIbeam	3	I beam.
vbIconPointer	4	Icon.
vbSizePointer	5	Size.
vbSizeNESW	6	Size NE, SW.
vbSizeNS	7	Size N, S.
vbSizeNWSE	8	Size NW, SE.
vbSizeWE	9	Size W, E.
vbUpArrow	10	Up arrow.
vbHourglass	11	Hourglass.
vbNoDrop	12	No drop.
vbArrowHourglass	13	Arrow and hourglass. (Only available in 32-bit Visual Basic 4.0.)
vbArrowQuestion	14	Arrow and question mark. (Only available in 32-bit Visual Basic 4.0.)
vbSizeAll	15	Size all. (Only available in 32-bit Visual Basic 4.0.)
vbCustom	99	Custom icon specified by the MouseIcon property.

MsgBox Constants

MsgBox Arguments

Constant	Value	Description
vbOKOnly	0	OK button only (default)
vbOKCancel	1	OK and Cancel buttons.
vbAbortRetryIgnore	2	Abort, Retry, and Ignore buttons.
vbYesNoCancel	3	Yes, No, and Cancel buttons.
vbYesNo	4	Yes and No buttons.
vbRetryCancel	5	Retry and Cancel buttons.
vbCritical	16	Critical message.
vbQuestion	32	Warning query.
vbExclamation	48	Warning message.
vbInformation	64	Information message.
vbDefaultButton1	0	First button is default (default)
vbDefaultButton2	256	Second button is default.
vbDefaultButton3	512	Third button is default.
vbApplicationModal	0	Application modal message box (default)
vbSystemModal	4096	System modal message box.

MsgBox Return Values

Constant	Value	Description
vbOK	1	OK button pressed.
vbCancel	2	Cancel button pressed.
vbAbort	3	Abort button pressed.
vbRetry	4	Retry button pressed.
vbIgnore	5	Ignore button pressed.
vbYes	6	Yes button pressed.
vbNo	7	No button pressed.

OLE Container Control Constants

OLEType Property

Constant	Value	Description
vbOLELinked	0	OLE container control contains a linked object.
vbOLEEmbedded	1	OLE container control contains an embedded object.
vbOLENone	3	OLE container control doesn't contain an object.

OLETypeAllowed Property

Constant	Value	Description
vbOLEEither	2	OLE container control can contain either a linked or an embedded object.

UpdateOptions Property

Constant	Value	Description
vbOLEAutomatic	0	Object is updated each time the linked data changes.
vbOLEFrozen	1	Object is updated whenever the user saves the linked document from within the application in which it was created.
vbOLEManual	2	Object is updated only when the Action property is set to 6 (Update)

AutoActivate Property

Constant	Value	Description
vbOLEActivateManual	0	OLE object isn't automatically activated.
vbOLEActivateGetFocus	1	Object is activated when the OLE container control gets the focus.
vbOLEActivateDoubleclick	2	Object is activated when the OLE container control is double-clicked.
vbOLEActivateAuto	3	Object is activated based on the object's default method of activation.

SizeMode Property

Constant	Value	Description
vbOLESizeClip	0	Object's image is clipped by the OLE container control's borders.
vbOLESizeStretch	1	Object's image is sized to fill the OLE container control.
vbOLESizeAutoSize	2	OLE container control is automatically resized to display the entire object.
vbOLESizeZoom	3	Object's image is stretched but in proportion.

DisplayType Property

Constant	Value	Description
vbOLEDisplayContent	0	Object's data is displayed in the OLE container control.
vbOLEDisplayIcon	1	Object's icon is displayed in the OLE container control.

Updated Event Constants

Constant	Value	Description
vbOLEChanged	0	Object's data has changed.
vbOLESaved	1	Object's data has been saved by the application that created the object.
vbOLEClosed	2	Application file containing the linked object's data has been closed.
vbOLERenamed	3	Application file containing the linked object's data has been renamed.

Special Verb Values

Constant	Value	Description
vbOLEPrimary	0	Default action for the object.
vbOLEShow	-1	Activates the object for editing.
vbOLEOpen	-2	Opens the object in a separate application window.
vbOLEHide	-3	For embedded objects, hides the application that created the object.
vbOLEInPlaceUIActivate	-4	All UI's associated with the object are visible and ready for use.
vbOLEInPlaceActivate	-5	Object is ready for the user to click inside it and start working with it.
vbOLEDiscardUndoState	-6	For discarding all record of changes that the object's application can undo.

Verb Flag Bit Masks

Constant	Value	Description
vbOLEFlagEnabled	0x0	Enabled menu item.
vbOLEFlagGrayed	0x1	Grayed menu item.
vbOLEFlagDisabled	0x2	Disabled menu item.
vbOLEFlagChecked	0x8	Checked menu item.
vbOLEFlagSeparator	0x800	Separator bar in menu item list.
vbOLEMiscFlagMemStorage	0x1	Causes control to use memory to store the object while it's loaded.
vbOLEMiscFlagDisableInPlace	0x2	Forces OLE container control to activate objects in a separate window.

VBTranslateColor/OLETranslateColor Constants

Constant	Value	Description
vbInactiveCaptionText	0x80000013	Color of text in an inactive caption.
vb3DHighlight	0x80000014	Highlight color for 3-D display elements.
vb3DFace	0x8000000F	Dark shadow color for 3-D display elements.
vbMsgBox	0x80000017	Background color for message boxes and system dialog boxes.
vbMsgBoxText	0x80000018	Color of text displayed in message boxes and system dialog boxes.
vb3DShadow	0x80000010	Color of automatic window shadows.
vb3DDKShadow	0x80000015	Darkest shadow.
vb3DLight	0x80000016	Second lightest of the 3-D colors (after vb3DHighlight)

Picture Object Constants

Constant	Value	Description
vbPicTypeBitmap	1	Bitmap type of Picture object.
vbPicTypeMetafile	2	Metafile type of Picture object.
vbPicTypeIcon	3	Icon type of Picture object.

Printer Object Constants

Printer Color Mode

Constant	Value	Description
vbPRCMMonochrome	1	Monochrome output.
vbPRCMColor	2	Color output.

Duplex Printing

Constant	Value	Description
vbPRDPSimplex	1	Single-sided printing.
vbPRDPHorizontal	2	Double-sided horizontal printing.
vbPRDPVertical	3	Double-sided vertical printing.

Printer Orientation

Constant	Value	Description
vbPRORPortrait	1	Documents print with the top at the narrow side of the paper.
vbPRORLandscape	2	Documents print with the top at the wide side of the paper.

Print Quality

Constant	Value	Description
vbPRPQDraft	-1	Draft print quality.
vbPRPQLow	-2	Low print quality.
vbPRPQMedium	-3	Medium print quality.
vbPRPQHigh	-4	High print quality.

PaperBin Property

Constant	Value	Description
vbPRBNUpper	1	Use paper from the upper bin.
vbPRBNLower	2	Use paper from the lower bin.
vbPRBNMiddle	3	Use paper from the middle bin.
vbPRBNManual	4	Wait for manual insertion of each sheet of paper.
vbPRBNEnvelope	5	Use envelopes from the envelope feeder.
vbPRBNEnvManual	6	Use envelopes from the envelope feeder, but wait for manual insertion.
vbPRBNAuto	7	(Default)
vbPRBNTractor	8	Use paper fed from the tractor feeder.

PaperBin Property (continued)

Constant	Value	Description
vbPRBNSmallFmt	9	Use paper from the small paper feeder.
vbPRBNLargeFmt	10	Use paper from the large paper bin.
vbPRBNLargeCapacity	11	Use paper from the large capacity feeder.
vbPRBNCassette	14	Use paper from the attached cassette cartridge.

PaperSize Property

Constant	Value	Description
vbPRPSLetter	1	Letter, 8 1/2 x 11 in.
vbPRPSLetterSmall	2	+A611Letter Small, 8 1/2 x 11 in.
vbPRPSTabloid	3	Tabloid, 11 x 17 in.
vbPRPSLedger	4	Ledger, 17 x 11 in.
vbPRPSLegal	5	Legal, 8 1/2 x 14 in.
vbPRPSStatement	6	Statement, 5 1/2 x 8 1/2 in.
vbPRPSExecutive	7	Executive, 7 1/2 x 10 1/2 in.
vbPRPSA3	8	A3, 297 x 420 mm.
vbPRPSA4	9	A4, 210 x 297 mm.
vbPRPSA4Small	10	A4 Small, 210 x 297 mm.
vbPRPSA5	11	A5, 148 x 210 mm.
vbPRPSB4	12	B4, 250 x 354 mm.
vbPRPSB5	13	B5, 182 x 257 mm.
vbPRPSFolio	14	Folio, 8 1/2 x 13 in.
vbPRPSQuarto	15	Quarto, 215 x 275 mm.
vbPRPS10x14	16	10 x 14 in.
vbPRPS11x17	17	11 x 17 in.
vbPRPSNote	18	Note, 8 1/2 x 11 in.
vbPRPSEnv9	19	Envelope #9, 3 7/8 x 8 7/8 in.
vbPRPSEnv10	20	Envelope #10, 4 1/8 x 9 1/2 in.
vbPRPSEnv11	21	Envelope #11, 4 1/2 x 10 3/8 in.
vbPRPSEnv12	22	Envelope #12, 4 1/2 x 11 in.
vbPRPSEnv14	23	Envelope #14, 5 x 11 1/2 in.
vbPRPSCSheet	24	C size sheet.
vbPRPSDSheet	25	D size sheet.
vbPRPSESheet	26	E size sheet.
vbPRPSEnvDL	27	Envelope DL, 110 x 220 mm.
vbPRPSEnvC3	29	Envelope C3, 324 x 458 mm.
vbPRPSEnvC4	30	Envelope C4, 229 x 324 mm.
vbPRPSEnvC5	28	Envelope C5, 162 x 229 mm.
vbPRPSEnvC6	31	Envelope C6, 114 x 162 mm.

vbPRPSEnvC65	32	Envelope C65, 114 x 229 mm.

PaperSize Property (continued)

Constant	Value	Description
vbPRPSEnvB4	33	Envelope B4, 250 x 353 mm.
vbPRPSEnvB5	34	Envelope B5, 176 x 250 mm.
vbPRPSEnvB6	35	Envelope B6, 176 x 125 mm.
vbPRPSEnvItaly	36	Envelope, 110 x 230 mm.
vbPRPSEnvMonarch	37	Envelope Monarch, 3 7/8 x 7 1/2 in.
vbPRPSEnvPersonal	38	Envelope, 3 5/8 x 6 1/2 in.
vbPRPSFanfoldUS	39	U.S. Standard Fanfold, 14 7/8 x 11 in.
vbPRPSFanfoldStdGerman	40	German Standard Fanfold, 8 1/2 x 12 in.
vbPRPSFanfoldLglGerman	41	German Legal Fanfold, 8 1/2 x 13 in.
vbPRPSUser	256	User-defined.

RasterOp Constants

Constant	Value	Description
vbDstInvert	0x00550009	Inverts the destination bitmap.
vbMergeCopy	0x00C000CA	Combines the pattern and the source bitmap.
vbMergePaint	0x00BB0226	Combines the inverted source bitmap with the destination bitmap by using Or.
vbNotSrcCopy	0x00330008	Copies the inverted source bitmap to the destination.
vbNotSrcErase	0x001100A6	Inverts the result of combining the destination and source bitmaps by using Or.
vbPatCopy	0x00F00021L	Copies the pattern to the destination bitmap.
vbPatInvert	0x005A0049L	Combines the destination bitmap with the pattern by using Xor.
vbPatPaint	0x00FB0A09L	Combines the inverted source bitmap with the pattern by using Or. Combines the result of this operation with the destination bitmap by using Or.
vbSrcAnd	0x008800C6	Combines pixels of the destination and source bitmaps by using And.

RasterOp Constants (continued)

Constant	Value	Description
vbSrcCopy	0x00CC0020	Copies the source bitmap to the destination bitmap.
vbSrcErase	0x00440328	Inverts the destination bitmap and combines the result with the source bitmap by using And.
vbSrcInvert	0x00660046	Combines pixels of the destination and source bitmaps by using Xor.
vbSrcPaint	0x00EE0086	Combines pixels of the destination and source bitmaps by using Or.

Shell Constants

Constant	Value	Description
vbHide	0	Window is hidden and focus is passed to the hidden window.
vbNormalFocus	1	Window has focus and is restored to its original size and position.
vbMinimizedFocus	2	Window is displayed as an icon with focus.
vbMaximizedFocus	3	Window is maximized with focus.
vbNormalNoFocus	4	Window is restored to its most recent size and position. The currently active window remains active.
vbMinimizedNoFocus	6	Window is displayed as an icon. The currently active window remains active.

StrConv Constants

Constant	Value	Description
vbUpperCase	1	Uppercases the string.
vbLowerCase	2	Lowercases the string.
vbProperCase	3	Uppercases first letter of every word in string.
vbWide*	4*	Converts narrow (single-byte)(double-byte)
vbNarrow*	8*	Converts wide (double-byte)(single-byte)
vbKatakana**	16**	Converts Hiragana characters in string to Katakana characters.
vbHiragana**	32**	Converts Katakana characters in string to Hiragana characters.
vbUnicode***	64***	Converts the string to Unicode using the default code page of the system.
vbFromUnicode***	128***	Converts the string from Unicode to the default code page of the system.

*Applies to Far East locales
**Applies to Japan only.
***Specifying this bit on 16-bit systems causes a run-time error
.

Variant Type Constants

Constant	Value	Description
vbVEmpty	0	Empty (uninitialized)
vbVNull	1	Null (no valid data)
vbVInteger	2	Integer data type.
vbVLong	3	Long integer data type.
vbVSingle	4	Single-precision floating-point data type.
vbVDouble	5	Double-precision floating-point data type.
vbVCurrency	6	Currency (scaled integer)
vbVDate	7	Date data type.
vbVString	8	String data type.

VarType Constants

Constant	Value	Description
vbEmpty	0	Uninitialized (default)
vbNull	1	Contains no valid data.
vbInteger	2	Integer.
vbLong	3	Long integer.
vbSingle	4	Single-precision floating-point number.
vbDouble	5	Double-precision floating-point number.
vbCurrency	6	Currency.
vbDate	7	Date.
vbString	8	String.
vbObject	9	OLE Automation object.
vbError	10	Error.
vbBoolean	11	Boolean.
vbVariant	12	Variant (used only for arrays of Variants)
vbDataObject	13	Non-OLE Automation object.
vbByte	17	Byte
vbArray	8192	Array.