Programming Microsoft Windows with Visual Basic
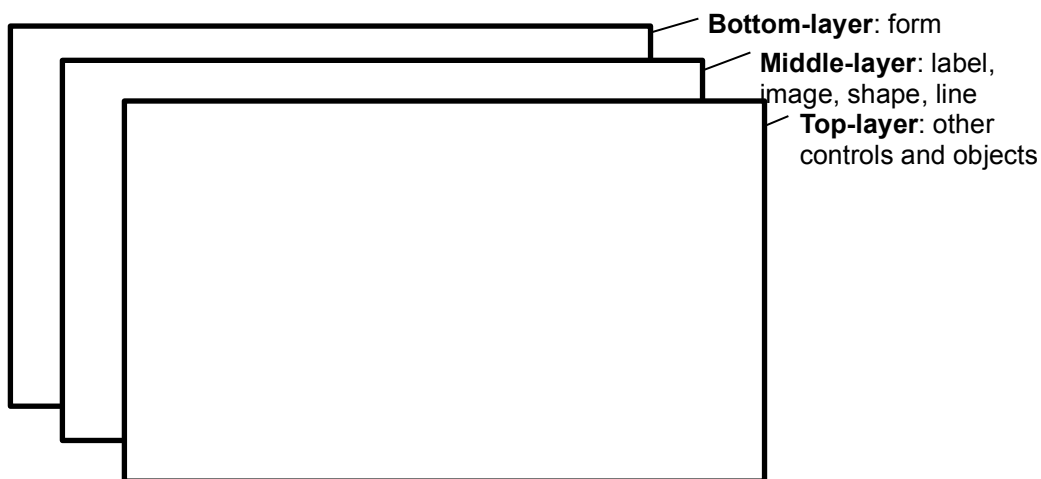
4. More Exploration of the Visual Basic Toolbox

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, text printed on the form would be hidden by a command button placed on top of it. 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, the Z-order can be modified by clicking on the desired object and choosing **Bring to Front** from Visual Basic's **Edit** menu. 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.
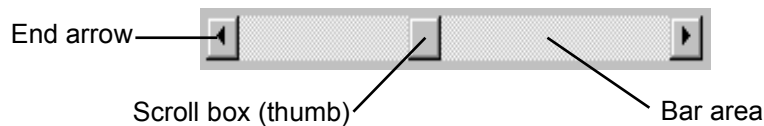
- Shape Tool Properties:

  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:

End arrow
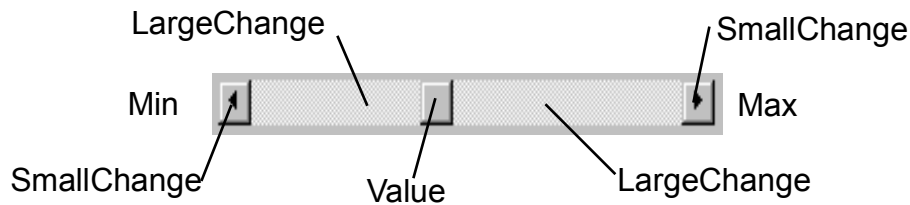
Scroll box (thumb)   Bar area

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.
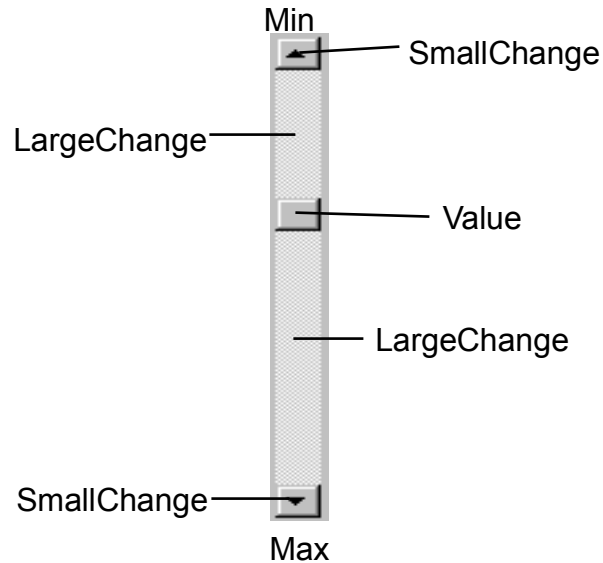
* Scroll Bar Properties:

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:

LargeChange   SmallChange

Min   Max

SmallChange   Value   LargeChange

Properties for vertical scroll bar:

Min

SmallChange

LargeChange

Value

LargeChange

SmallChange

Max

- 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.

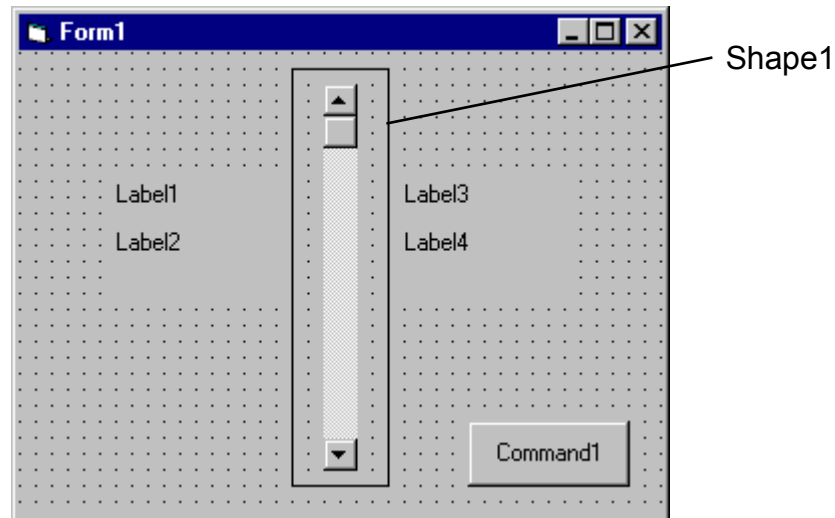### Temperature Conversion Application Specifications

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'll 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'll 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 they could be obscured by picture boxes and other objects. 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. Bitmap files can be found in the c:\vb\bitmaps directory, icon files in the c:\vb\icons directory, and metafiles are in the c:\vb\metafile directory.

   3. Notice what happens as you resize the two boxes. Notice the layer affect 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.
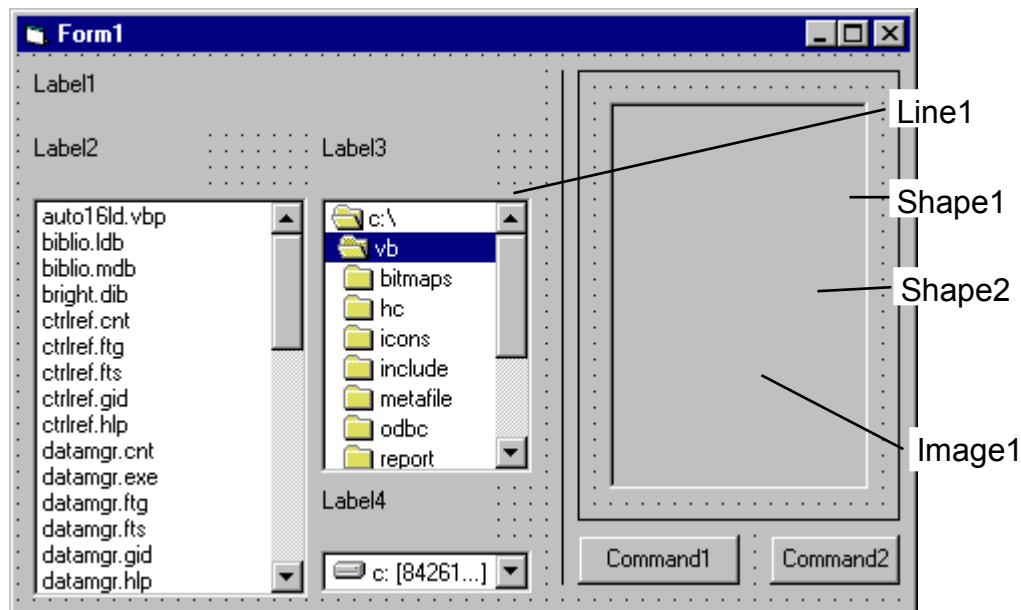
Example 4-2

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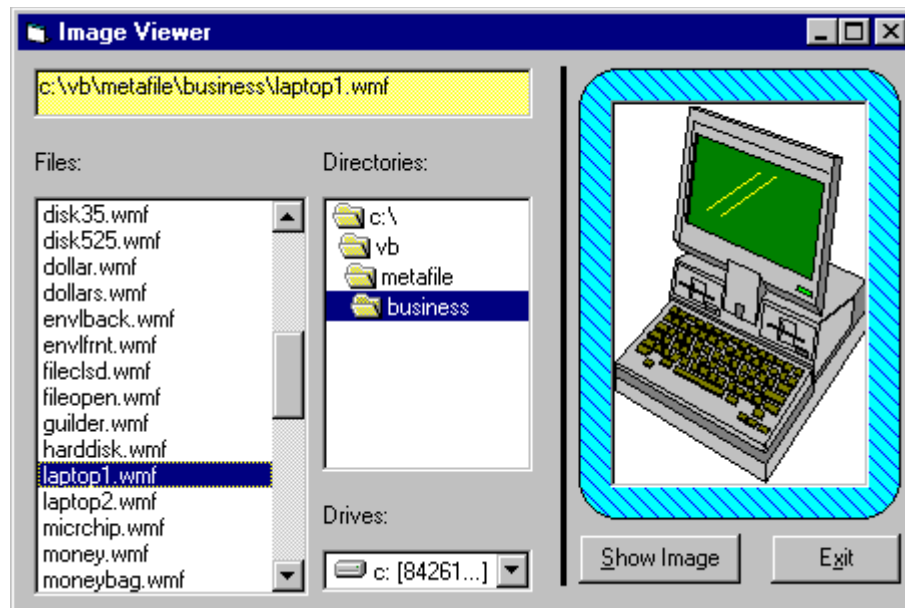
This code forms the file name (**ImageName**) by concatenating the directory path with the file name. It then displays the complete name and loads the picture into the image box.

6. Copy the code from the **cmdShow_Click** procedure and paste it into the **fillmage_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 **fillmage_DblClick** procedure call the **cmdShow_Click** procedure. We'll 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's how the form should look when displaying one example metafile:

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't 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. This is done by selecting **Components** under the **Project** menu. 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 '**Show'** 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:
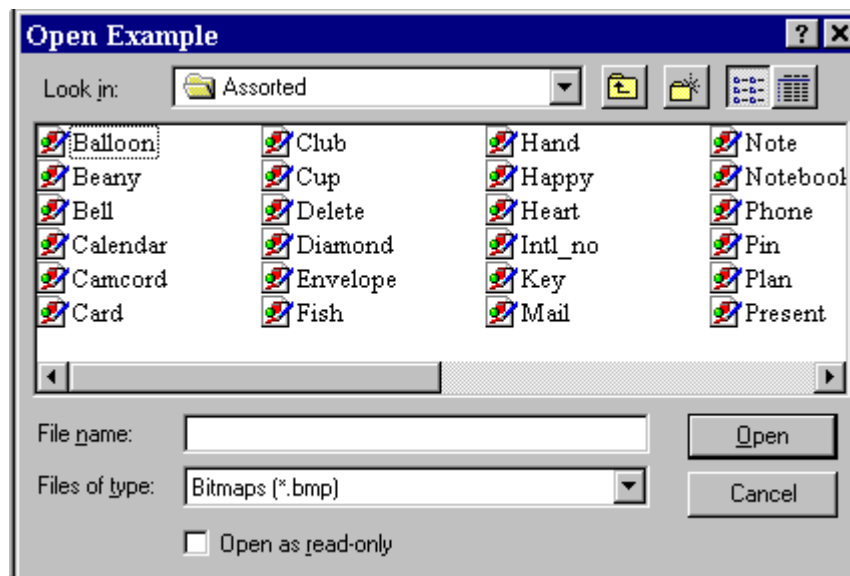
      cdlExample.ShowOpen

  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'll 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'll worry about how to open a file in Class 6. The box is displayed by using the **ShowOpen** method. Here's an example of an Open common dialog box:

- 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
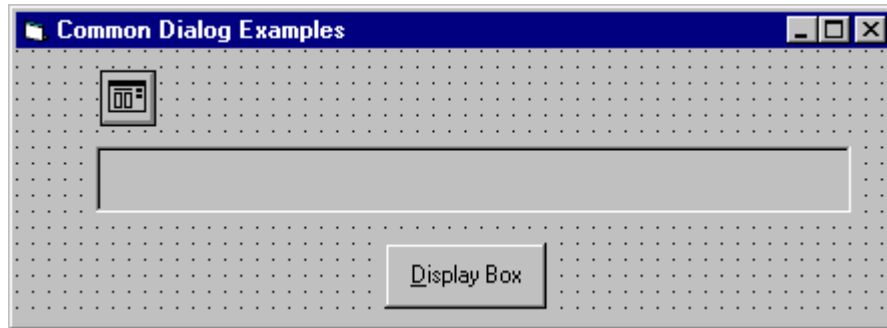   (all on one line)
Name   cdlExample

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):

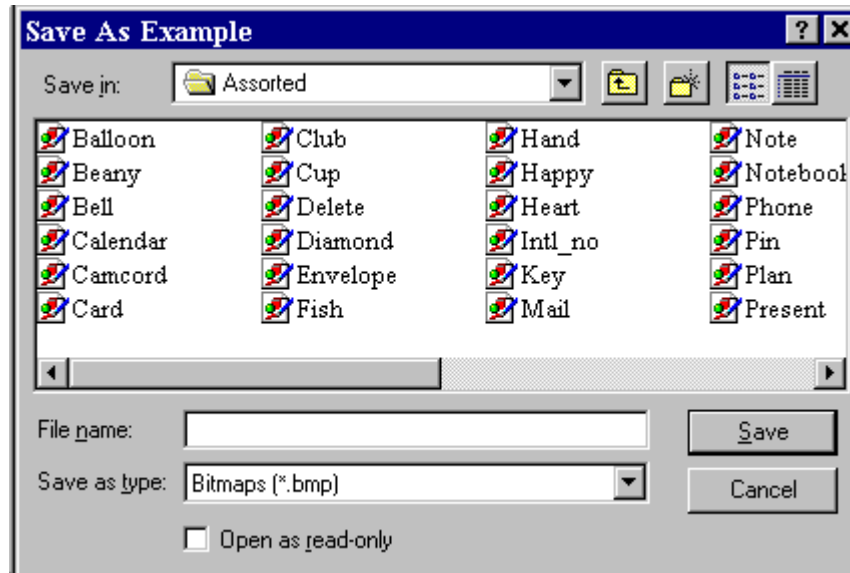2. Attach this code to the **cmdDisplay_Click** procedure.

```
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'll worry about how to save a file in Class 6. The box is displayed by using the **ShowSave** method.. Here's 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've 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'll 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**.
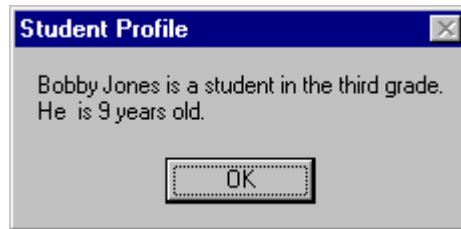
Exercise 4

Student Database Input Screen

You did so well with last week's 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:
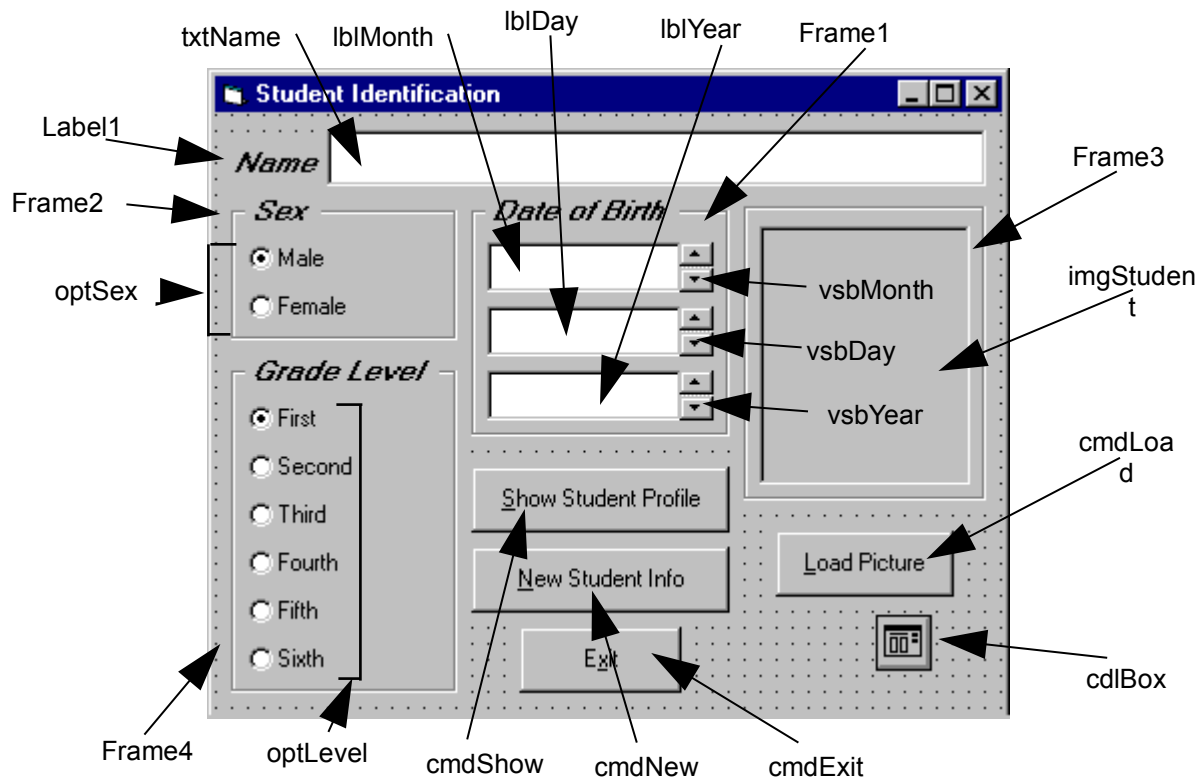

Note the student's age must be computed from the input birth date - watch out for pitfalls in doing the computation. The student's picture does not appear in the profile, only on the input screen.

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
```