Programming Microsoft Windows with Visual Basic

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.

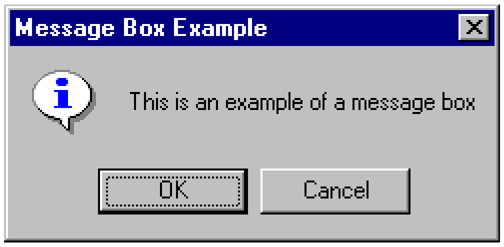
- Note for each option in **Type**, there are numeric values listed and symbolic constants. Recall, it is strongly suggested that the symbolic constants be used instead of the numeric values. You should agree that **vbOKOnly** means more than the number 0 when selecting the button type.

- 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 “This is an example of a message box”, vbOKCancel + vbInformation, “Message Box Example”



- 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 specifed 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.

WordWrap	Works in conjunction with AutoSize property. If AutoSize = True, WordWrap = True, then the text will wrap and label will expand vertically to fit the Caption. If AutoSize = True, WordWrap = 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 WordWrap 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

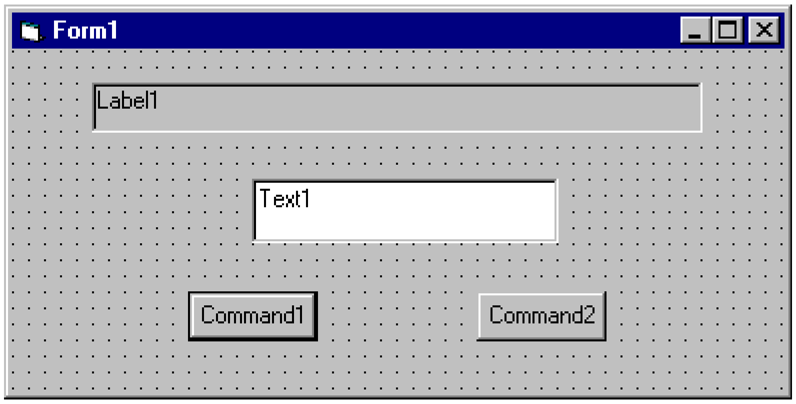
```
txtExample.SetFocus ' moves cursor to box named txtExample
```

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]

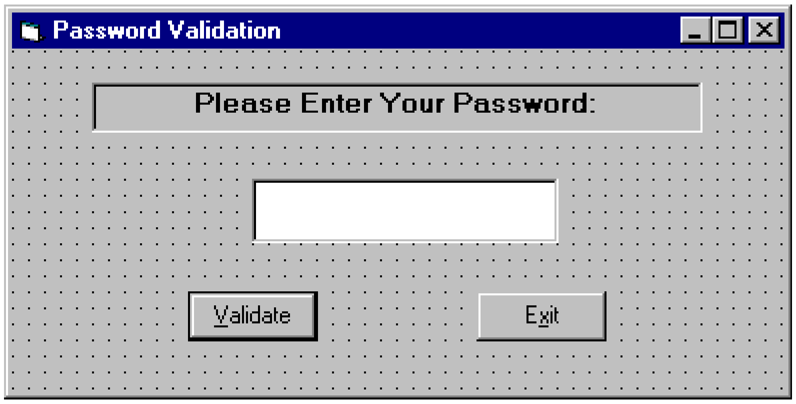
Command1:

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'll 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 declarations 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).

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, elements of the array are referred to by their name and index. For example, to set the **Caption** property of element **6** of a label box array named **lblExample**, we would use:

```
lblExample(6).Caption = “This 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:

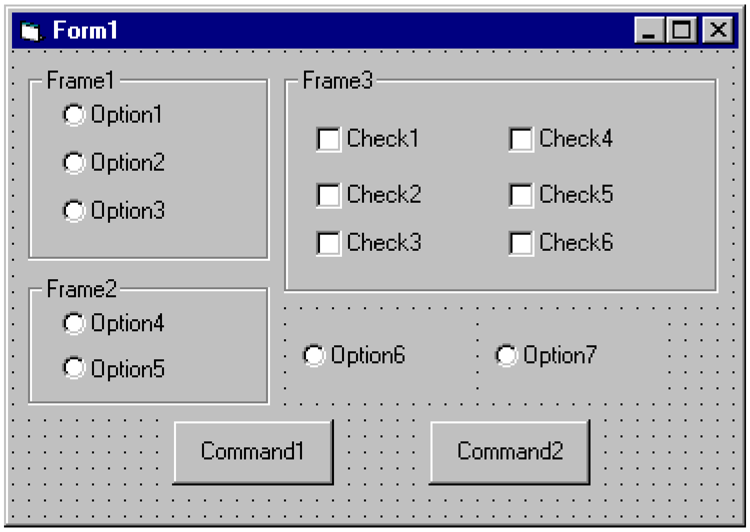
Caption	Title information at top of frame.
Font	Sets font type, style, size.

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

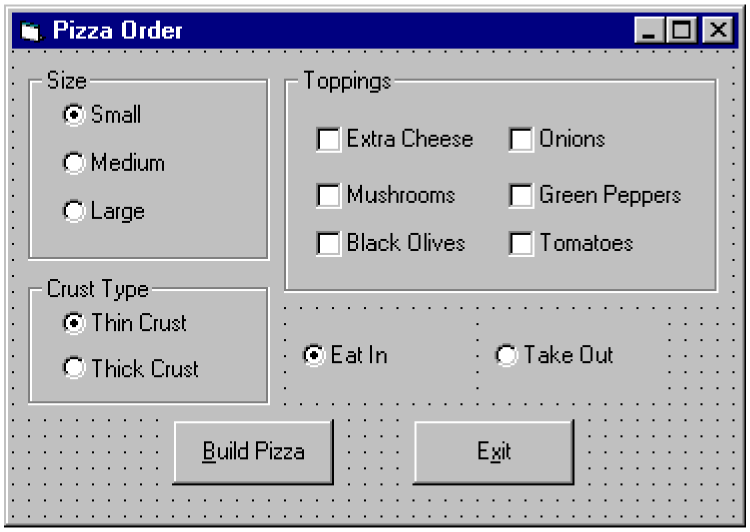
Command1:

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)
‘Read pizza size
PizzaSize = optSize(Index).Caption
End Sub

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

Private Sub optWhere_Click(Index As Integer)
‘Read 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’s 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 'carriage 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'll 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 "Cheese Only" appears on the order form. You'll 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
```

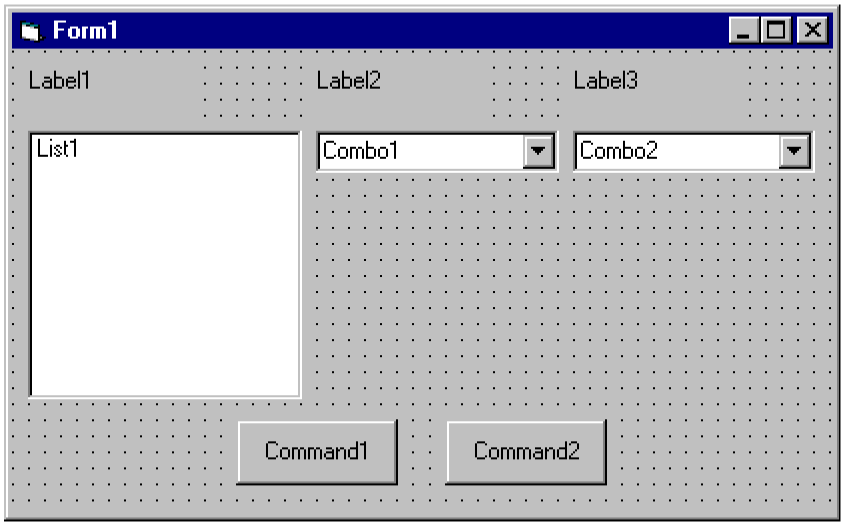
- You've seen combo boxes before. In the standard 'Open File' window, the File Name box is a combo box of Style 2, while the Drive box is a combo box of Style 3.

Example 3-3

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

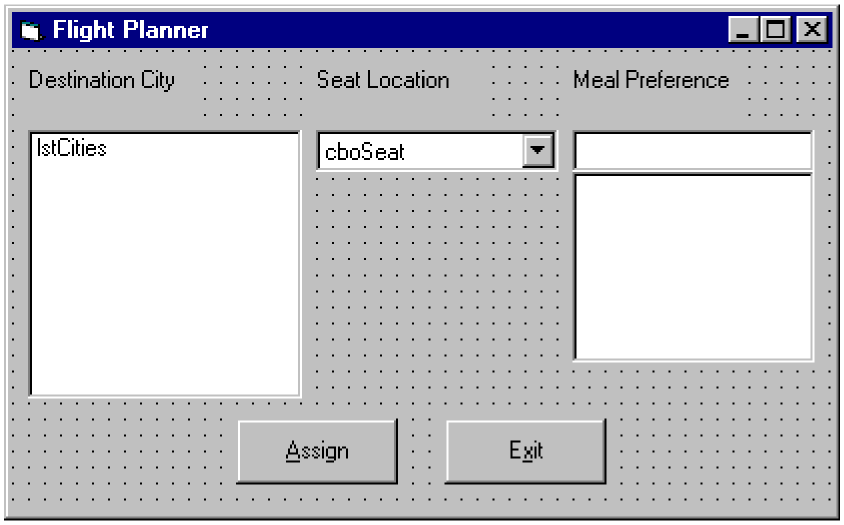
Command1:

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()
'Add 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

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

'Add 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()
'Build 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.

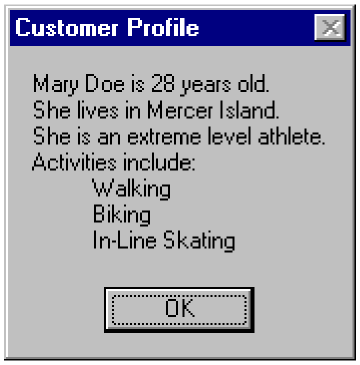
<u>Exercise 3</u>

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)

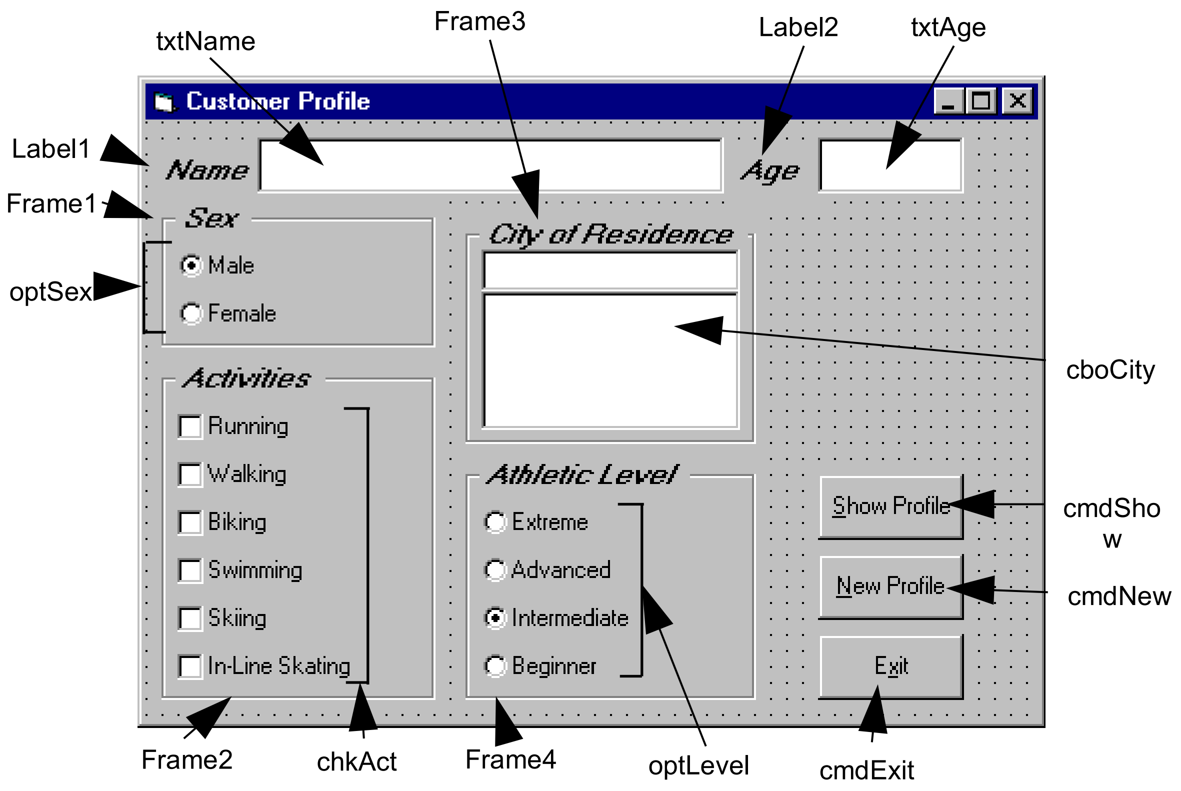
Set up the screen so that only the Name and Age (use text boxes) and, perhaps, City (use a combo box) need to be typed; all other inputs should be set with check boxes and option buttons. When a screen of information is complete, display the summarized profile in a message box. This profile message box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