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 Properti es:

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:

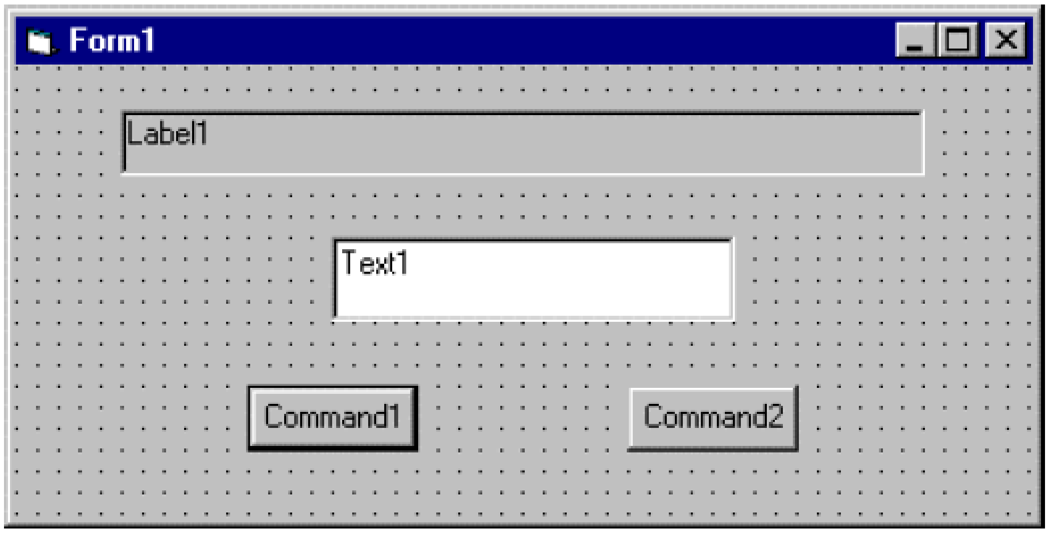
SetFocus	Places the cursor in a specified text box.

Example 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]

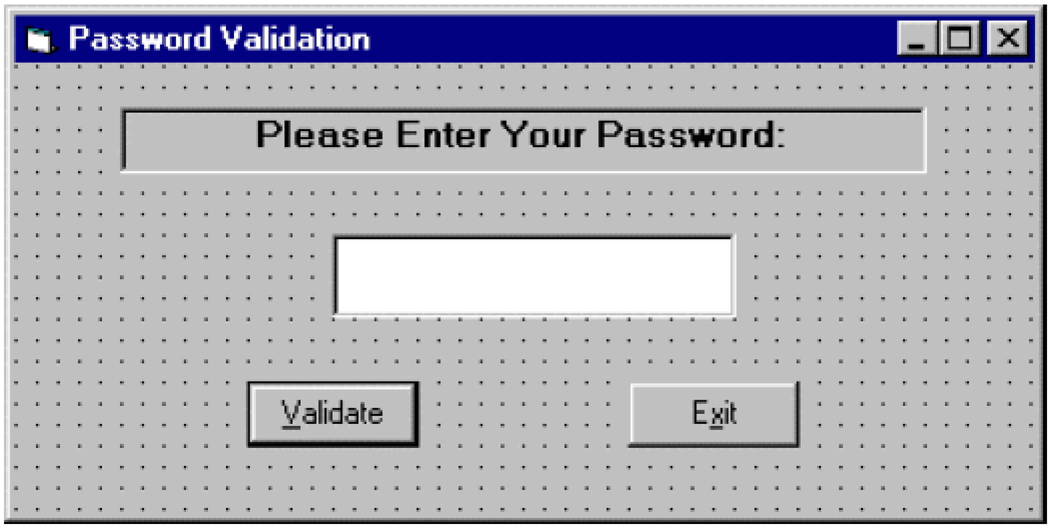
Command1:

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).