ListBoxes and ComboBoxes are very similar. Both allow the user to select an item from a list of choices. By restricting the values the user can select to specific allowed values, these controls help the user make valid selections. This chapter describes some of the ways a program can take full advantage of the capabilities of these two types of controls.

**79. Find Selected Items**

Directory: ShowSels

The ListBox control’s ListIndex property gives the index of the item that is selected in the ListBox. Often a program can check this property to see what item the user has selected.

If the control’s MultiSelect property is 1 (Simple) or 2 (Extended), however, the user can select more than one item at a time. In that case, a single ListIndex property cannot give the indexes of all the selected items.

Example program ShowSels demonstrates a method for determining which items have been selected. The program displays a list of animals in a ListBox with MultiSelect set to 2 (Extended). Hold down the Control key and click on the animals to select them. Whenever you change the selected animals, the program lists the selected items in the TextBox on the right.
How It Works

To see which items are selected when MultiSelect is not 0 (None), a program can use the control's Selected property. This value returns True if the indicated item is selected. For instance, the following code determines whether the first item in the list is selected.

If lstItems.Selected(0) Then ...

The following code shows how program ShowSels displays its list of selected items.

Option Explicit

' Start with nothing selected.
Private Sub Form_Load()
    lstItems.ListIndex = -1
End Sub

' The user clicked an item. Update the display.
Private Sub lstItems_Click()
    Dim i As Integer
    Dim txt As String

    For i = 0 To lstItems.ListCount - 1
        ' See if item i is selected.
        If lstItems.Selected(i) Then _
            txt = txt & lstItems.List(i) & vbCrLf
    Next i

    txtSelected.Text = txt
End Sub
80. **Find an ItemData Value**

**Directory: ItemData**

The ListBox and ComboBox controls both provide an ItemData property that lets you associate numeric data values with the items in the list. For example, `lstItems.ItemData(0)` returns the value associated with the first item in the list `lstItems`.

There is no control property or method that locates an item with a particular ItemData value, but a program can use a simple search to find an ItemData value.

Example program `ItemData` shows how to locate items with particular ItemData values. This program displays a ComboBox and a ListBox containing a series of department names with their telephone extensions. Each entry's ItemData value is set to its extension. For instance, the Finance department's entry reads “Finance 8121.” This entry's ItemData value is 8121. Enter an extension in the TextBox, and click the Find button to make the program locate the corresponding item in both controls.

![Program ItemData](image)

**How It Works**

A program can locate a particular ItemData value by searching all of the ItemData values until it finds the one it needs. The following code shows how program `ItemData` works.

```vbnet
Option Explicit

' Find the items with the given ItemData value.
Private Sub cmdFind_Click()
    SelectItemData lstItems, txtExtension
    SelectItemData cboItems, txtExtension
End Sub

' Search a control's ItemData property for the target value. If found select the item.
```
Public Sub SelectItemData(ctl As Control, target As Integer)
Dim i As Long

' Only do this for ListBoxes and ComboBoxes.
If TypeName(ctl) <> "ListBox" And 
   TypeName(ctl) <> "ComboBox" Then Exit Sub

' Examine all the items.
For i = 0 To ctl.ListCount - 1
   If ctl.ItemData(i) = target Then
      ctl.ListIndex = i
      Exit Sub
   End If
Next

' The target was not found. Select nothing.
ctl.ListIndex = -1
End Sub

81. Change Dropdown Height
Directory: CboHgt

Normally when the user clicks on a ComboBox, the control presents a dropdown list with a certain standard height. This height is not necessarily appropriate for all programs. For instance, a ComboBox might contain one more item than will fit in the default height. In that case, it will be a little awkward for the user to have to scroll to find the last option. If the dropdown area were a little taller, the user could see all the choices at once.

Fortunately, a program can use the MoveWindow API function to resize a ComboBox’s dropdown area. Example program CboHgt does just that. The ComboBox on the left has the standard dropdown height. The one on the right has been modified so that its dropdown area reaches almost to the bottom of the form.
How It Works

The following code shows how program CboHgt resizes its right ComboBox. The MoveWindow API function measures distances in pixels, so the SizeCombo subroutine performs its calculations in pixels.

Option Explicit

#If Win32 Then
    Private Declare Function MoveWindow Lib "user32" ( _
        ByVal hWnd As Long, ByVal X As Long, ByVal Y As Long, _
        ByVal nWidth As Long, ByVal nHeight As Long, ByVal bRepaint As Long) As Long
#Else
    Private Declare Sub MoveWindow Lib "User" ( _
        ByVal hWnd As Integer, ByVal X As Integer, _
        ByVal Y As Integer, ByVal nWidth As Integer, _
        ByVal nHeight As Integer, ByVal bRepaint As Integer)
#End If

' Resize a ComboBox's dropdown display area.
Public Sub SizeCombo(frm As Form, cbo As ComboBox)
    Dim cbo_left As Integer
    Dim cbo_top As Integer
    Dim cbo_width As Integer
    Dim cbo_height As Integer
    Dim old_scale_mode As Integer

    ' Change the Scale Mode on the form to Pixels.
    old_scale_mode = frm.ScaleMode
    frm.ScaleMode = vbPixels
'Save the ComboBox's Left, Top, and Width
'in pixels.
cbo_left = cbo.Left
cbo_top = cbo.Top
cbo_width = cbo.Width

'Calculate the new height of the combo box.
cbo_height = frm.ScaleHeight - cbo.Top - 5
frm.ScaleMode = old_scale_mode

'Resize the combo box window.
MoveWindow cbo hwnd, cbo_left, cbo_top, _
cbo_width, cbo_height, 1
End Sub

The worst case occurs when a ComboBox contains only a few more items than will fit
on its dropdown list. In that case, it is usually better to enlarge the dropdown list so that
it can display all of the items. If the list contains many more items than will fit at one
time, the user will not mind scrolling through the list as much.

82. Change Dropdown Width

Directory: CboWid

The previous section explains how to use the MoveWindow API function to make a
ComboBox's dropdown area taller than normal. Using the SendMessage API function, a
program can make the dropdown area wider than normal.

Example program CboWid uses this approach to let the user select an acronym. The left
ComboBox has a standard width. When you open its dropdown area, the choices are
truncated. The right ComboBox has a dropdown area large enough to display all of the
text for its list of items.

Program CboWid.
How It Works

The following code shows how program CboWid resizes its right ComboBox. The Size-Combo subroutine examines each of the items in the ComboBox. It uses the form’s TextWidth function to see how wide these items are. It then resizes the dropdown area to make it wide enough to display the widest of the items.

Option Explicit

#If Win32 Then
    Private Declare Function SendMessage Lib "user32" _
        Alias "SendMessageA" (ByVal hWnd As Long, ByVal wMsg As Long, _
        ByVal wParam As Long, lParam As Long) As Long
#End If

Private Const CB_SETDROPPEDWIDTH = &H160

Private Sub Form_Load()
    #If Win32 = False Then
        MsgBox "This example only works in 32-bit Visual Basic."
    End If
    #End If

    ' Resize the ComboBox’s dropdown area.
    SizeCombo Me, cboWide
End Sub

Public Sub SizeCombo(frm As Form, cbo As ComboBox)
    Dim old_font As Object
    Dim old_scale_mode As Integer
    Dim cbo_left As Integer
    Dim cbo_top As Integer
    Dim cbo_width As Integer
    Dim cbo_height As Integer
    Dim i As Integer
    Dim max_width As Long
    Dim new_width As Long

    ' Change the Scale Mode on the form to Pixels
    ' and the font to the ComboBox's font.
    old_scale_mode = frm.ScaleMode
    frm.ScaleMode = vbPixels
    Set old_font = frm.Font
    Set frm.Font = cbo.Font

    ' Save the ComboBox’s Left, Top, and Width
    ' in pixels.
    cbo_left = cbo.Left
cbo_top = cbo.Top
    cbo_width = cbo.Width

' Find the widest ComboBox item.
max_width = 10
For i = 0 To cbo.ListCount - 1
    new_width = frm.TextWidth(cbo.List(i))
    If max_width < new_width Then
        max_width = new_width
    Next i

' Allow room for the scroll bar.
max_width = max_width + 25

' Restore the original ScaleMode.
frm.ScaleMode = old_scale_mode
Set frm.Font = old_font

' Set the width for the dropdown list.
SendMessage cbo.hWnd, _
    CB_SETDROPPEDWIDTH, max_width, 0
End Sub

This technique can be particularly useful on forms that are very crowded. When the ComboBox is closed, only the leftmost part of the selected list item is visible. That part of the item entries can show a brief description or abbreviation of the selected item. For instance, in program CboWid, the items' abbreviations are visible when the ComboBoxes are closed.

When the user clicks on the ComboBox, it expands to show the list entries in full. The leftmost part of the entries can give a short abbreviation while the full entries give additional description that takes up space only when the dropdown area is visible.

83. Set Tabs in a ListBox

Directory: ListTabs

When a program uses a proportionally spaced font, some characters are wider than others. That means columns of text will generally not line up neatly unless the program takes special action. For instance, a form can use its CurrentX property to line up columns of text, as shown in the following code.

Private Sub Form_Paint()
    Const COLUMN_1 = 1440
    Const COLUMN_2 = 2880

    Dim TextValue(1 To 10, 1 To 2) As String
Dim i As Integer

' Initialize the strings.
:

' Display the strings.
CurrentY = 1440
For i = 1 To 10
    CurrentX = COLUMN_1
    Print TextValue(i, 1);
    CurrentX = COLUMN_2
    Print TextValue(i, 2)
Next i
End Sub

A program can use the SendMessage API function with the LB_SETTABSTOPS message to line up entries in a ListBox that uses a proportionally spaced font. Example program ListTabs demonstrates this technique.

Program ListTabs.

**How It Works**

The following code shows how program ListTabs works.

Option Explicit

#End If

Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hWnd As Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As Any) As Long

Private Const LB_SETTABSTOPS = &H192

Private Sub Form_Load()
    Dim tabs(1 To 2) As Long
    #If Win32 Then
        Private Declare Function SendMessage Lib "user32" As Long
        Alias "SendMessageA" (ByVal hWnd As Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As Any) As Long
    #End If

Private Const LB_SETTABSTOPS = &H192

Private Sub Form_Load()
    Dim tabs(1 To 2) As Long
    #If Win32 Then
        MsgBox "This program works only in 32-bit Visual Basic."
' Define the tabs (in pixels).
tabs(1) = 30
tabs(2) = 150

' Set the tabs.
SetListTabs lstAligned, 2, tabs

' Enter some values.
lstAligned.AddItem "Price" & vbTab & "Title" & vbTab & "Author"
lstAligned.AddItem "$49.99" & vbTab & _
    "A Short Title" & vbTab & "Anabelle Argus"
lstAligned.AddItem "$39.95" & vbTab & _
    "A Book With a Much Longer Title" & vbTab & "Berndt Barker"
lstAligned.AddItem "$20.00" & vbTab & _
    "Yet Another Longish Book Title" & vbTab & "Cindy Cranston"
lstAligned.AddItem "$74.99" & vbTab & _
    "An Expensive Book" & vbTab & "Danver Delarius"
lstAligned.AddItem "$34.95" & vbTab & _
    "Bugsy, A Tale of Rough Software" & vbTab & "Ellen Edwards"
lstAligned.AddItem "$44.99" & vbTab & _
    "A Tale of Two Bugs" & vbTab & "Franklin Francois"
End Sub

' Set the ListBox's tabs.
Private Sub SetListTabs(ByVal lst As ListBox, ByVal num As Long, _
tabs() As Long)
    SendMessage lst.randn, LB_SETTABSTOPS, 2, tabs(1)
End Sub

84. Move Items between Lists

Directory: MoveList

Many programs allow the user to move items from one list to another. For instance, Visual Basic 5's ActiveX Control Interface Wizard uses two lists to allow the user to pick standard control properties. The user clicks buttons to move items back and forth between a list of selected properties and a list of unselected properties.

Example program MoveList shows how to let a user move items between two lists. Select one or more items in the left list and click the > button to move them into the right list. Click the >> button to move all of the items into the right list. Use the < and << buttons to move items from the right list to the left list.
How It Works

There are two main tasks a program must perform to provide this kind of interface. First, when the user clicks on a list, the program must enable the correct buttons. For instance, when no items are selected in the left list, the > button should be disabled because no items have been selected to move into the right list. Second, the program must move the correct items when the user clicks a button.

The following code shows how program MoveList accomplishes these tasks. When the user clicks on the left or right list and when the form initially loads, the program calls subroutine EnableButtons. This routine examines the lists’ Selected values to see if any items are selected in the lists. It enables or disables the > and < buttons accordingly.

EnableButtons also determines whether the lists are empty and enables or disables the >> and << buttons accordingly. For instance, if the right list is empty, the program cannot move any items into the left list so it disables the << button.

When the user clicks on the < button, the cmdMoveLeft_Click event handler executes. This routine examines the Selected values for the items in the right list to see which are selected. When it finds one that is selected, it adds that item to the left list and removes it from the right list. This routine examines the items in decreasing order so it considers those with the largest indexes first. That prevents confusion caused by the control renumbering items as they are removed from the list.

The subroutine that moves selected items from the left list into the right list and the routines that move all items from one list to another are similar.

```vbnet
' Enable the appropriate buttons.
Private Sub lstLeft_Click()
    EnableButtons
End Sub

' Enable the appropriate buttons.
Private Sub lstRight_Click()
    EnableButtons
End Sub
```
Private Sub Form_Load()
   ' Put some values in the left list.
   lstLeft.AddItem "Ape"
   lstLeft.AddItem "Bear"
   lstLeft.AddItem "Cat"
   lstLeft.AddItem "Dog"
   lstLeft.AddItem "Eagle"
   lstLeft.AddItem "Frog"
   lstLeft.AddItem "Giraffe"
   lstLeft.AddItem "Hen"
   lstLeft.AddItem "Ibex"
   lstLeft.AddItem "Jackel"

   ' Enable the appropriate buttons.
   EnableButtons
End Sub

' Enable the appropriate buttons.
Private Sub EnableButtons()
Dim i As Integer

   ' See if an item is selected in the left list.
   For i = lstLeft.ListCount - 1 To 0 Step -1
      If lstLeft.Selected(i) Then Exit For
   Next i
   cmdMoveRight.Enabled = (i >= 0)

   ' See if an item is selected in the right list.
   For i = lstRight.ListCount - 1 To 0 Step -1
      If lstRight.Selected(i) Then Exit For
   Next i
   cmdMoveLeft.Enabled = (i >= 0)

   ' See if the right list has any items.
   cmdMoveLeftAll.Enabled = (lstRight.ListCount > 0)

   ' See if the left list has any items.
   cmdMoveRightAll.Enabled = (lstLeft.ListCount > 0)
End Sub

' Move the items selected in the right list into the left list.
Private Sub cmdMoveLeft_Click()
Dim i As Integer

   ' Remove the selected items.
   For i = lstRight.ListCount - 1 To 0 Step -1
      ' Move this item.
      If lstRight.Selected(i) Then
         lstLeft.AddItem lstRight.List(i)
         lstRight.RemoveItem i
   Next i
End If
Next i

' Enable the correct buttons.
EnableButtons
End Sub

' Move the items selected in the left list
' into the right list.
Private Sub cmdMoveRight_Click()
Dim i As Integer

' Remove the selected items.
For i = lstLeft.ListCount - 1 To 0 Step -1
' Move this item.
If lstLeft.Selected(i) Then
    lstRight.AddItem lstLeft.List(i)
lstLeft.RemoveItem i
End If
Next i

' Enable the correct buttons.
EnableButtons
End Sub

' Move all items into the left list.
Private Sub cmdMoveLeftAll_Click()
Dim i As Integer

' Move the items.
For i = lstRight.ListCount - 1 To 0 Step -1
' Move this item.
lstLeft.AddItem lstRight.List(i)
lstRight.RemoveItem i
Next i

' Enable the correct buttons.
EnableButtons
End Sub

' Move all items into the right list.
Private Sub cmdMoveRightAll_Click()
Dim i As Integer

' Move the items.
For i = lstLeft.ListCount - 1 To 0 Step -1
' Move this item.
lstRight.AddItem lstLeft.List(i)
lstLeft.RemoveItem i
Next i
This kind of interface is useful when the user must select one or more items from a very long list, particularly if the user is likely to select only a few items. In that case, the user can see all of the selected items in the right list, but the unselected items in the left list do not take up much room.

If the user is selecting from a very short list, you should consider using a series of OptionButtons instead of this two-list interface.

85. **Save and Restore a ComboBox**

Directory: SaveCbo

An application can use simple file operations to load a ComboBox’s data from a file when the program starts and save any changes when it stops. Example program SaveCbo demonstrates this technique. Enter a value in the ComboBox, and click the Add button to make it part of the list of choices. Click the Remove button to remove the selected item from the list. When you close and restart the program, the changes you have made will be reloaded automatically.

How It Works

Program SaveCbo stores information about its ComboBox in the file animals.dat. This file begins with the number of items in the ComboBox and the index of the currently selected item. It then contains the list values. The following shows a small sample file:

```
6,5
"Aardvark"
"Badger"
"Cow"
"Frog"
"Giraffe"
"Ibex"
```
The program does all its interesting processing in its Form_Load and Form_Unload event handlers. The Form_Load subroutine opens the file animals.dat for input. It reads the number of items in the ComboBox and the index of the selected item. It then reads each of the ComboBox item values and adds them to the ComboBox. Finally, it selects the item that was selected when the file was created.

Subroutine Form_Unload opens the file animals.dat for output. It saves the number of items in the ComboBox and the index of the selected item. It then saves each of the ComboBox item values and closes the file.

```
' Load the data from last time.
Private Sub Form_Load()
    Dim fnum As Integer
    Dim num_items As Integer
    Dim list_index As Integer
    Dim txt As String
    Dim i As Integer

    cboAnimals.Clear
    On Error GoTo NoFile
    fnum = FreeFile
    Open App.Path & "\animals.dat" For Input As fnum
    On Error GoTo CloseFile

    ' Read the number of items and the index of
    ' the selected item.
    Input #fnum, num_items, list_index

    ' Read the list items.
    For i = 0 To num_items - 1
        Input #fnum, txt
        cboAnimals.AddItem txt
    Next i
    cboAnimals.ListIndex = list_index

    CloseFile:
    Close fnum
    NoFile:

    End Sub

' Save the data for next time.
Private Sub Form_Unload(Cancel As Integer)
    Dim fnum As Integer
    Dim i As Integer

    fnum = FreeFile
    Open App.Path & "\animals.dat" For Output As fnum
```
There are several other ways a program can save and load data. For instance, it could save the data in a database or in the system registry. It could also save the data for more than one control in the same file.

Also note that saving changes in the Form_Unload event handler is not completely foolproof. If the user makes changes to the ComboBox and then the program crashes without exiting normally, Form_Unload never executes so the user’s changes are lost. If it is important that the changes always be saved, and if saving the values does not take too long, the program should save them as soon as the changes are made instead of waiting until the form unloads.

86. Match ListBox Prefixes

Directory: FindItem

When a ListBox control has the input focus, you can press a letter to make the list jump to the first item that begins with that letter. If you press the letter again, the list moves to the next item that starts with the same letter. You can continue pressing the letter to move through the items starting with that letter.

This method of selection is convenient when the list contains only a few items that begin with the same letter, but if many items have the same first letter, selecting a particular item this way can be tedious. You may need to press the letter many times before you find the item you want.

Example program FindItem demonstrates a technique for making item selection easier. Enter the first few letters of an item in the program’s TextBox. The program selects the first item in the list that matches all the letters you enter.
How It Works

Program FindItem sends its ListBox control the LB_SELECTSTRING message to make it find and select the first item that begins with the prefix you enter in the TextBox. The following code shows how program FindItem works. The parameter -1 passed to SendMessage tells the ListBox to start searching the list after the item with index -1. That makes it search starting at the first item in the list.

Option Explicit

Private Declare Function SendMessage Lib "user32"_
    Alias "SendMessageA" (ByVal hWnd As Long, ByVal wMsg As Long, _
    ByVal wParam As Long, ByVal lParam As String) As Long

Private Const LB_SELECTSTRING = &H18C

' Select the first item that matches this value.
Private Sub txtValue_Change()
    SendMessage lstWords.hWnd, LB_SELECTSTRING, _
    -1, txtValue.Text
End Sub

Detect ComboBox Dropdown

Occasionally, it is useful for a program to know when a ComboBox control is about to display its dropdown list. For instance, the choices available in the ComboBox’s dropdown list might depend on items selected by the user. When the ComboBox is about to drop down, the program can update the items in its list.
In Visual Basic 5 and 6, a program can subclass to intercept Windows messages. When it sees the WM_COMMAND message, it can examine the message's parameters to see if the program is receiving a ComboBox dropdown event. If so, it can take whatever action is necessary.

Example program CboDown uses this method. Select some items in the ListBox on the left. Then click on the ComboBox's dropdown arrow to make it present its dropdown list. When you click on the ComboBox, the program copies the items you have selected in the ListBox into the ComboBox's dropdown list.

How It Works

The following code shows declarations and routines used by program CboDown and stored in the .bas module APIStuff.bas.

Subroutine CatchDropDown prepares the program to catch dropdown events. It saves the hWnd of the form and ComboBox, and it stores references to the ListBox and ComboBox controls. It then installs the NewWindowProc function as the form's new WindowProc routine. Whenever the form receives a Windows message, the system calls this function.

Function NewWindowProc checks the Window message to see if it is a WM_COMMAND message. If it is, the function then determines whether the message's command is CBN_DROPDOWN and whether the ComboBox that is about to display its dropdown list is the ComboBox to which it has a reference. If the message satisfies these conditions, the program is about to drop the ComboBox down so NewWindowProc copies the selected ListBox choices into the ComboBox's dropdown list.

Option Explicit

' The old WindowProc address.
Private OldWindowProc As Long

' The hWnd for the ComboBox.
Private ComboBoxHwnd As Long
' The ComboBox control.
Private TheComboBox As ComboBox

' The ListBox control from which we are taking
' the ComboBox's values.
Private TheListBox As ListBox

' API declarations.
Private Declare Function CallWindowProc Lib "user32" _
    Alias "CallWindowProcA" (ByVal lpPrevWndFunc As Long, _
    ByVal hWnd As Long, ByVal msg As Long, ByVal wParam As Long, _
    ByVal lParam As Long) As Long
Private Declare Function SetWindowLong Lib "user32" _
    Alias "SetWindowLongA" (ByVal hWnd As Long, ByVal nIndex As Long, _
    ByVal dwNewLong As Long) As Long

Private Const GWL_WNDPROC = (-4)
Private Const WM_COMMAND = &H111
Private Const CBN_DROPDOWN = 7

' Install the new WindowProc.
Public Sub CatchDropDown(ByVal hWnd As Long, ByVal cbo As ComboBox, ByVal lst As ListBox)
    ' Save the two controls and the ComboBox's hWnd.
    ComboHwnd = cbo.hWnd
    Set TheComboBox = cbo
    Set TheListBox = lst

    ' Subclass to catch ComboBox drop events.
    OldWindowProc = SetWindowLong( _
        hWnd, GWL_WNDPROC, _
        AddressOf NewWindowProc)
End Sub

' Look for the ComboBox dropdown.
Public Function NewWindowProc(ByVal hWnd As Long, ByVal msg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    Dim i As Integer
    If msg = WM_COMMAND Then
        ' See if this message was generated by ComboBox
        ' and see if it is a CBN_DROPOWN command.
        If ComboHwnd = (lParam And &HFFFF) And _
            (wParam \ &H10000) = CBN_DROPOWN Then
            ' The ComboBox is being dropped down.
            ' Start with no items in the ComboBox.
            TheComboBox.Clear
Add the items selected in the ListBox
to the ComboBox.
For i = 0 To TheListBox.ListCount - 1
    If TheListBox.Selected(i) Then
        ' This item is selected.
        TheComboBox.AddItem TheListBox.List(i)
    End If
Next i
End If

' Call the original WindowProc.
NewWindowProc = CallWindowProc( _
    OldWindowProc, hWnd, msg, wParam, _
    lParam)
End Function

The code that follows shows how the CboDown program’s main form uses the functions
in module APIStuff.bas. Compared to the previous code, this code is simple. The important
statement is the call to the CatchDropDown subroutine.

Private Sub Form_Load()
    ' Create some list values.
    lstValues.AddItem "Ape"
    lstValues.AddItem "Bear"
    lstValues.AddItem "Cat"
    lstValues.AddItem "Dog"
    lstValues.AddItem "Eagle"
    lstValues.AddItem "Fox"
    lstValues.AddItem "Giraffe"

    ' Subclass to catch the dropdown events.
    CatchDropDown hWnd, cboValues, lstValues
End Sub

As is the case with all subclassing programs, it is extremely important that this program
terminate normally. If you stop it using the Run menu’s End command or if the program
crashes, the Visual Basic development environment will crash as well. You will lose any
changes you have made since the last time you saved. To avoid wasting a lot of time,
save your work often.