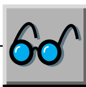


Subclassing?

Class modules aren't just for business rules.

by Karl E. Peterson

Of all the innovations Visual Basic 4.0 introduced, the most revolutionary is probably the new Class module. By now, you've no doubt read many accounts of how to encapsulate "business rules" in class modules. The client/server community is seemingly ecstatic over this newfound ability, and not without reason. Yet, after spending more than a year using Visual Basic 4.0, I still haven't seen anyone (other than the coauthors of *Visual Basic 4 How To*) cover what to me is one of the most exciting possibilities Class modules offer. My favorite technique is adding functionality to common controls and forms with little code once the Class module has been polished. I'll introduce this technique in this column.

No, this is not subclassing in the classic sense, but it's close. Using Class modules, you can provide plug-in event handlers, often with a single line of code. These could be used to extend or modify the standard behavior of your controls and forms. After seeing the transformation ahead, I believe you'll be impressed with the opportunities available.

In my November column, I presented a method you can use in either Visual Basic 3.0 or 4.0 to add "type-amatic" searches to list box controls. I consider that example to be control oriented, because all the code required to implement it was found in control events within the Form module.

This month, those routines will be transformed into a VB4 Class module that can give the same capabilities to any list box, with the addition of three simple lines of code. Even more amazing, two of these instructions simply set up the class for searching list boxes. Only one line of code is required to notify the class whenever the user has entered a keystroke. From there, the class takes care of the dirty work. After building the CListSearch class, you can add it to any existing project, and within minutes greatly enhance your project's user interface.

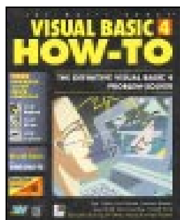
Because I've already dealt with the method used to search a list box in detail in a previous column, this month I will concentrate on working up a generic, control-enhancing Class module that you can plug into any project with ease. To begin, start a new project, and from the Insert menu select Class Module.

In the Properties dialog, set its Name to CListSearch, and leave the other properties set to their defaults. You do not need to set this class as either Public or Creatable because it will be used only within individual projects. While you could wrap it up in an in-process OLE server, I prefer to keep code such as this with the project. In a group programming effort, you may want to consider compiling all the control-enhancing classes you write into a single OLE server if this approach works better in your environment.

Typically, this column covers a number of different techniques. However, I devoted this month's column to fully developing a single technique because I believe it will prove so useful. After reading through this example and considering the potential it offers for enhancing your applications, hopefully you'll agree. You can download the code from this

CONTINUED ON PAGE 132.

Karl E. Peterson is a GIS analyst with a regional transportation planning agency, an independent consultant, a member of the Visual Basic Programmer's Journal *Technical Review Board, and coauthor of* Visual Basic 4 How To, *from Waite Group Press. Online, he's the 32-Bit Bucket section leader in the* VBPJ *Forum and a Microsoft MVP in the MSBASIC Forum. Contact Karl on either CompuServe section at 72302,3707.*



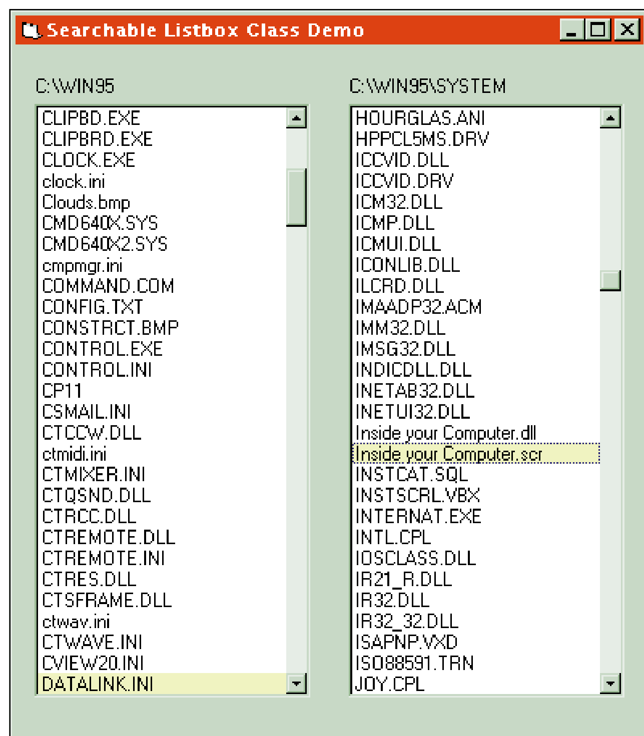


FIGURE 1 ***Two List Boxes Using the Same Searching Class.*** *This project expands on the November technique so that multiple list boxes can use the same code for "type-amatic" searches. A VB4 class module provides the mechanism, requiring only three lines of code in the form for each enhanced list box.*

©1991–1996 Fawcette Technical Publications

VB4

```
Public Property Set Client(NewObj As _
  Object)
  '
  ' Set new ListBox as Client property.
  '
  If TypeOf NewObj Is ListBox Then
     If NewObj.Sorted Then
        Set m_List = NewObj
        m_LastKey = Now
     Else
        Err.Raise Number:=vbObjectError + 2, _
                  Source:="CListSearch.Client", _
                  Description:="Client ListBox" & _
                    must have Sorted = True."
     End If
  Else
     Err.Raise Number:=vbObjectError + 1, _
               Source:="CListSearch.Client", _
               Description:="Client property" & _
                 "must be of type ListBox."
  End If
End Property
```

LISTING 1 ***Setting the Client Property of CListSearch.*** *By accepting a generic Object, rather than insisting on a list box, allowance is made for enhancing the class to handle other object types.*

CONTINUED FROM PAGE 129.
column from the Magazine Library of the *VBPJ* Forum on CompuServe (type GO VBPJ and search for PT0196.ZIP).

BUILDING CLISTSEARCH

To make the CListSearch class truly generic, it's best to decide up front whether to support both 16- and 32-bit applications. Because list boxes are available to and used in both environments, and the API calls are straightforward, this is an easy call to make. Enter this code to conditionally include API declarations and constants in the Declarations section of the class:

```
Option Explicit
'
' API Declarations and constants
'
#If Win16 Then
   Private Declare Function _
      SendMessage Lib "User" (ByVal hWnd _
      As Integer, ByVal wMsg As Integer, _
      ByVal wParam As Integer, lParam As Any) As Long
   Private Const LB_FINDSTRING = &H410
#ElseIf Win32 Then
   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_FINDSTRING = &H18F
#End If
Private Const LB_ERR = (-1)
```

Notice that SendMessage has been aliased so that it has the same name, and therefore you can call it using the same code in either Win16 or Win32 operating systems.

Watch for whether API constants have the same values. For this class, LB_FINDSTRING is different, while LB_ERR is the same in Win16 and Win32. Lack of attention to this sort of detail can cause untold aggravation.

The CListSearch class requires that a number of variables be maintained internally, cordoned off from the outside. The OOP terminology for this is member variable (please don't write to me about my terminology for OOP terminology being off—I just concentrate on practical aspects, and leave the semantics for the zealots). You use such variables to store the values of class properties after validation, as well as to store other data that is only accessible to the code within the class. Add these member variable allocations to the Declarations section of CListSearch:

```
'
' Set aside storage for private member
' variables.
'
Private m_List As ListBox
Private m_LastKey As Double
Private m_TimeLimit As Long
Private m_ExtendSearch As Boolean
Private m_Beep As Boolean
```

Class modules offer an Initialize event in which you can set the default values for class properties and other member variables. This event will be the first code to execute within the class: it's fired upon instantiation. CListSearch must maintain a reference to the list box it is "subclassing," but because this object is unknown at instantiation, you set m_List to Nothing. The m_LastKey variable stores the time of the last keystroke, but again this is unknown at this point so

©1991–1996 Fawcette Technical Publications

you set it to zero.

An arbitrary time limit of two seconds is stored as 2000 milliseconds in the m_TimeLimit variable. This value is used to either extend ongoing, or start new searches as the user enters additional keystrokes. If the last keystroke occurred prior to the time limit, a new search is begun. Otherwise, the current one is extended.

NO, THIS IS NOT SUBCLASSING IN THE CLASSIC SENSE, BUT IT'S CLOSE.

The m_ExtendSearch variable is used simply as an internal flag for adding characters to the current search string. The m_Beep variable is a flag used to determine if the class should emit a beep when a search fails.

```
' ************************************
'  Initialize
' ************************************
Private Sub Class_Initialize()
   '
   ' Set default values for class
   ' properties.
   '
   Set m_List = Nothing
   m_LastKey = 0
   m_TimeLimit = 2000      '2 seconds
   m_ExtendSearch = False
   m_Beep = True
End Sub
```

The first property to add to CListSearch is one that identifies the list box on which to operate. I've named this property Client and allowed it to accept any type of Object as a parameter. The incoming parameter could have been restricted to list boxes, but leaving the option open allows you to enhance the class to work with other types of controls, such as combo boxes. Similarly, a control-enhancing class that added capabilities to scroll bars would need to handle both horizontal and vertical varieties, and you may set up graphics classes to work with both Forms and Picture boxes.

When the Client property is set, its validation code checks that the proper object type was passed (see Listing 1). If a list box was passed as the new Client, the reference to it is stored in m_List, and m_LastKey is set to the current time. If the passed object was not a list box, an error is raised using all the information necessary for the programmer to identify and handle it within the calling application. If an error is raised in a Class module, the point at which execution halts is determined by a setting in Visual Basic's Tools-Options-Advanced dialog. Select “Break on Unhandled Errors” to allow a calling application to receive the error raised in a Class module.

Next, add a corresponding Get property procedure to allow an application to retrieve a reference to the list box this instance of CListSearch is using. Although this procedure will be called rarely, avoid making properties write only. Omitting the Get half of the property procedure pair would prevent a client application from retrieving this information, even for debugging purposes:

```
Public Property Get Client() As Object
   '
   ' Return ListBox as Client property.
   '
   Set Client = m_List
End Property
```

The next pair of property procedures added to CListSearch serves the purpose of setting and retrieving TimeLimit. These procedures allow the calling application to set the length of time to wait between keystrokes when extending the current search. Remember that in the Initialize event, this value was set to 2000 milliseconds. Before accepting a new value for TimeLimit, the incoming setting is checked to confirm it is positive. If it's negative, m_TimeLimit is set to zero. Otherwise whatever was passed is accepted.

©1991–1996 Fawcette Technical Publications

```
Public Property Let TimeLimit _
   (NewVal As Long)
   '
   ' Set new value for number of
   ' milliseconds to wait between
   ' keystrokes when continuing
   ' a search.
   '
   If NewVal > 0 Then
      m_TimeLimit = NewVal
   Else
      m_TimeLimit = 0
   End If
End Property

Public Property Get TimeLimit() _
   As Long
   '
   ' Return current value for
   ' TimeLimit property.
   '
   TimeLimit = m_TimeLimit
End Property
```

The last pair of property procedures added to CListSearch provide an option to turn off the default beep produced when searches fail. No validation is required because this property is Boolean. I chose the default True to match the behavior found in Win95's Explorer. However, not everyone appreciates a beeping computer so I provided the option to turn it off:

```
Public Property Let AudibleError _
   (NewVal As Boolean)
   '
   ' Store whether or not to beep when
   ' search fails.
   '
   m_Beep = NewVal
End Property

Public Property Get AudibleError() _
   As Boolean
   '
   ' Return current value for
   ' AudibleError property.
   '
   AudibleError = m_Beep
End Property
```

KeyPress is the only Public method offered by CListSearch. The code here is essentially what was used in the KeyPress event of the control-oriented project I covered in a previous column. You'll notice it's virtually identical, with the addition of a few lines of code that determine whether the last user keystroke was within the time limit allowed for an extended search. As you will see shortly, the beauty of wrapping this code up in a class is that you no longer need to enter it in the KeyPress event of every list box you want to enhance (see Listing 2).

AFTER BUILDING THE CLISTSEARCH CLASS, YOU CAN ADD IT TO AN EXISTING PROJECT AND GREATLY ENHANCE YOUR PROJECT'S USER INTERFACE.

PUT IT TO THE TEST

Now that you've developed the CListSearch class module, you need to build a simple project to test how it works and to demonstrate how easy it is to add searching capabilities to any number of list boxes. Start by adding two list boxes and two labels to the default Form1 of your project (see Figure 1). Control positioning is not important because you can handle that in the form's Resize event.

You will use some simple API calls to find the Windows and System directories during the form's Load event so the list boxes can be filled with sample data, but you will need to declare these calls in the form's Declarations section. Again, I've used conditional compilation so this project will run in either the 16- or 32-bit versions of Visual Basic 4.0. To use the CListSearch class, you also declare two New objects of this type in the Declarations section. The search implementation code count so far is one line per control (see Listing 3).

During the Form_Load event, GetWindowsDirectory and GetSystemDirectory are called to locate the respective directories. These paths are then used to fill the two list boxes with the names of all files located in either place. This provides sample data to test the search capabilities offered by CListSearch. Also during

 ©1991–1996 Fawcette Technical Publications

VB4

```
Public Function KeyPress(KeyAscii As Integer)
  Static Search As String
  Dim Index As Long
  Dim DoSearch As Boolean
  Dim Elapsed As Double
  Const SecsPerDay = 86400
  '
  ' Check if more than allowed time has elapsed.
  '
  If m_ExtendSearch Then
     Elapsed = Now - m_LastKey
     If (Elapsed * SecsPerDay) > (m_TimeLimit / _
        1000) Then
        m_ExtendSearch = False
     End If
  End If
  '
  ' Start over if delay was too long.
  '
  If Not m_ExtendSearch Then
     Search = ""
     m_ExtendSearch = True
     Index = m_List.ListIndex
  Else
     Index = -1
  End If
  '
  ' Check for valid keystrokes.
  '
  If KeyAscii = vbKeyBack Then
     '
     ' Allow user to take back last key.
     '
     If Len(Search) Then
        Search = Left(Search, Len(Search) - 1)
        DoSearch = True
     End If
  ElseIf KeyAscii >= vbKeySpace Then
     '
     ' Append latest key.
     '
     Search = Search & Chr(KeyAscii)
     DoSearch = True
  End If
  '
  ' Perform search after valid keystrokes.
  '
  If DoSearch Then
     Index = SendMessage(m_List.hWnd, _
        LB_FINDSTRING, Index, ByVal Search)
     If Index <> LB_ERR Then 'Found a match!
        m_List.ListIndex = Index
     Else 'No match
        Search = Left(Search, Len(Search) - 1)
        If m_Beep Then Beep
     End If
     '
     ' Record when key was pressed, and consume
     ' keystroke (by returning 0) so VB doesn't
     ' automatically move list to entry that
     ' starts with last key.
     '
     m_LastKey = Now
     KeyPress = 0
  Else
     '
     ' Return passed KeyAscii value so original
     ' KeyPress routine can continue processing.
     '
     KeyPress = KeyAscii
  End If
End Function
```

LISTING 2 ***The Core Of the Class.*** *The KeyPress method of CListSearch performs the actual searching whenever it is notified that the user has pressed a key. Code in this method would previously have been inserted in the List_KeyPress event, but has now been abstracted to deal with the "subclassed" list box.*

VB4

```
Option Explicit
'
' Windows API Declarations
'
#If Win32 Then
  Private Declare Function GetWindowsDirectory _
     Lib "kernel32" Alias "GetWindowsDirectoryA" _
     (ByVal lpBuffer As String, ByVal nSize As _
     Long) As Long
  Private Declare Function GetSystemDirectory _
     Lib "kernel32" Alias "GetSystemDirectoryA" _
     (ByVal lpBuffer As String, ByVal nSize As _
     Long) As Long
#ElseIf Win16 Then
  Private Declare Function GetWindowsDirectory _
     Lib "Kernel" (ByVal lpBuffer As String, _
     ByVal nSize As Integer) As Integer
  Private Declare Function GetSystemDirectory _
     Lib "Kernel" (ByVal lpBuffer As String, _
     ByVal nSize As Integer) As Integer
#End If
'
' Create searchable listbox objects
'
Private cLstWin As New CListSearch
Private cLstSys As New CListSearch
```

LISTING 3 ***Test Form Declarations.*** *API functions are declared using conditional compilation to allow use in either 16- or 32-bit versions of VB4. Two instances of CListSearch are created for this test of the class.*

Form_Load, a reference to one of the list boxes is passed to each declared instance of CListSearch, so that the class instance will know which object to perform its searches on. The defaults for other properties of CListSearch are accepted by simply not bothering to change them. The search implementation code count so far is two lines per control (see Listing 4).

Now I'll show you how incredibly powerful Visual Basic 4.0 classes can be. To fire the search mechanism in either instance of CListSearch, only one more line of code is required for either instance. When the user presses a key while a "subclassed" list box has focus, that keystroke is passed from the List_KeyPress event to the class' KeyPress method. After performing its search, based on the criteria set in the class, the KeyPress method returns either a zero if a search was performed or the original keystroke for further processing.

Further processing is necessary if a search is found not desirable (for example, in cases where non-alphanumerics, such as an arrow key or the enter key, are pressed). The return value is assigned to the List_KeyPress' KeyAscii parameter so that Visual Basic will ignore the keystroke or act on it based on what happened in the class. If further processing in the List_KeyPress event is desired, for example, to act on the Enter key, code for that may follow the call to the class KeyPress method. The search implementation code count is now three lines per control.

```
Private Sub List1_KeyPress(KeyAscii As Integer)
   '
   ' Allow class module to do all the work
```

©1991–1996 Fawcette Technical Publications

```
   '
   KeyAscii = _
      cLstWin.KeyPress(KeyAscii)
End Sub

Private Sub List2_KeyPress(KeyAscii _
   As Integer)
   '
   ' Allow class module to do all the
   ' work!
   '
   KeyAscii = _
      cLstSys.KeyPress(KeyAscii)
End Sub
```

The method I presented in a previous column used form-level variables to track data now stored in each instance of CListSearch. Obviously, this would get unwieldy if you wanted a number of searching list boxes on a single form. The overall number of lines of code would have nearly doubled as well, due to duplicating the KeyPress code in the events of each enhanced list box. By wrapping up this functionality inside a Visual Basic 4.0 class, and creating a separate instance of it for each enhanced list box, your form requires considerably less code. Plus, the savings are multiplied by the number of enhanced list boxes on the form.

You can consider a number of possibilities for enhancing the CListSearch class. One that comes to mind immediately is to add an ExtendSearch property you could reset when a list box gains focus. Although it would be unlikely that a previous search would be continued, setting this property to False on a List_GotFocus event would ensure that to be the case. Another potential enhancement would be to add support for combo boxes to the CListSearch class.

CLASS MODULES OFFER AN INITIALIZE EVENT IN WHICH YOU CAN SET THE DEFAULT VALUES FOR CLASS PROPERTIES.

VB4

```
Private Sub Form_Load()
  Dim path As String
  Dim file As String
  Dim nRet As Long
  '
  ' Fill List1 with \Windows files.
  '
  path = Space(256)
  nRet = GetWindowsDirectory _
     (path, Len(path))
  file = Dir(Left(path, nRet) _
     & "\*.*")
  Do While Len(file)
     List1.AddItem file
     file = Dir()
  Loop
  Label1.Caption = Left(path, nRet)
  '
  ' Fill List2 with \Windows
  ' \System files.
  '
  path = Space(256)
  nRet = GetSystemDirectory _
     (path, Len(path))
  file = Dir(Left(path, nRet) _
     & "\*.*")
  Do While Len(file)
     List2.AddItem file
     file = Dir()
  Loop
  Label2.Caption = Left(path, nRet)
  '
  ' Setup searchable listbox
  ' objects
  '
  Set cLstWin.Client = List1
  Set cLstSys.Client = List2
  '
  ' Center form
  '
  Me.Move (Screen.Width - Me. _
     Width)\ 2, (Screen. _
     Height - Me.Height) \ 2
End Sub
```

LISTING 4 ***Setting Up the Test Form.*** *While the form is loading, two list boxes are filled with the contents of the Windows and system directories. References to the list boxes are then passed to separate CListSearch instances so the class can take over keystroke handling.*

The technique presented this month should give you all kinds of ideas for enhancing controls and forms in your projects. I'd love to hear about the interesting one you devise. Reach me on either the *VBPJ* Forum or the MSBASIC Forum on CompuServe at 72302,3707. The editors of *Visual Basic Programmer's Journal* and I would also like to know if you want to see more examples of this type, or if this diversion from multiple techniques in one column is something that should be used only on an occasional basis. ☒

©1991–1996 Fawcette Technical Publications