Use a Label Control as a Hyperlink

Click & Retrieve Source CODE!

by Karl E. Peterson

Q Create a Hyperlink Label

I'd like to use a Label control as a hyperlink on my About box. The design calls for the label to look normal until the user moves the mouse over it, then "light up" to resemble a typical hyperlink—blue and underlined. It's easy enough determining when to turn on this effect, but how do I know when to turn it off?

A You've run into what I've always considered a missing event. It's simple to detect when the mouse first moves over a control by watching for a MouseMove event, but there's no documented way to tell when it leaves. Wouldn't it be nice if there were a MouseLeave event?

Several solutions have been developed over the years. You can monitor the underlying form's MouseMove event to signal that the mouse is no longer over your control. But a user can thwart this strategy by moving the mouse too fast or moving away from your application using Alt-Tab. Another method that works fairly well, if you're using a windowed control, is to call SetCapture on the first MouseMove, and test the passed coordinates on subsequent MouseMove events to determine whether the mouse is still over that window. This technique has its downsides too, not the least of which is that it's useless for windowless controls.

Recently, reader Mike Bolser shared a discovery of his with me that I think you'll like: You can use old-fashioned drag-and-drop techniques to obtain the needed notifications. Place a Label control on your form and assign the familiar finger-pointing cursor as its DragIcon. When the MouseMove event fires, set the Label's ForeColor property to vbBlue and its Font.Underline property to True. The ingenious part, also in MouseMove, is to call the Drag method, telling it to vbBeginDrag (see Listing 1).

The DragIcon cursor then kicks in, and your label looks just like a standard hyperlink (see Figure 1). To turn off the hyperlink look, watch for the State parameter to equal vbLeave in the Label's DragOver event. You then have the equivalent of a MouseLeave event. The only remaining trick is to overcome the loss of standard mouse events during a drag operation. It turns out that when the user clicks on your Label after dragging has begun, the Label's DragDrop event fires. If you have more than one control acting like this on a single form, you might want to check the Source parameter to ensure the dropped control is the same as the one firing the event, just in case the user moved the mouse very fast.

Then you're free to fire off the default browser, pointing it at the appropriate URL:

```
Private Declare Function ShellExecute _
   Lib "shell32.dll" Alias _
   "ShellExecuteA" (ByVal hWnd _
   As Long, ByVal lpOperation _
   As String, ByVal lpFile As String, _
   ByVal lpParameters As String, _
   ByVal lpDirectory As String, _
   ByVal nShowCmd As Long) As Long

Public Sub HyperJump(ByVal URL As String)
   Call ShellExecute(0&, _
      vbNullString, URL, _
      vbNullString, vbNullString, _
      vbNormalFocus)
End Sub
```

You can use the ShellExecute API to fire up nearly any kind of URL. For example, pass the HyperJump function a standard Web site address or pass it "mailto:yourname@yourdomain.com" to ask the

Figure 1 Offer a "Rollover" Effect for Hyperlinks. Using the code in Listing 1, you can provide interesting visual feedback as the user moves the mouse over various controls on your forms. Here, intrinsic Label and Image controls provide hyperlinks to my Web site, but the labels look like a hyperlink only while the mouse hovers over them.

ABOUT THIS COLUMN

Ask the VB Pro provides you with free advice on programming obstacles, techniques, and ideas. Read more answers from our crack VB pros on the Web at www.inquiry.com/thevbpro. You can submit your questions, tips, or ideas on the site, or access a comprehensive database of previously answered questions.

LINK

The resource editor for employing WAV files: msdn.microsoft.com/vbasic/downloads/controls.asp

VB4/32, VB5, VB6 | Detect MouseLeave Using Drag-and-Drop

```
Option Explicit
Private m_Active As Boolean
Private Sub Check1_Click()
  m_Active = CBool(Check1.Value)
End Sub
Private Sub Form_Load()
  Label1.Caption = "http://www.mvps.org/vb"
  Label1.Tag = Label1.Caption
  Check1.Value = vbChecked
End Sub

Private Sub Label1_DragDrop(Source As Control, _
  X As Single, Y As Single)
  ' If the mouse is over the label, the control
  ' must be in drag mode. In this case, the
  ' DragDrop event occurs when the mouse is
  ' clicked by the user. Fire up the URL!
If Source Is Label1 Then
    With Label1
      Call HyperJump(.Tag)
      .Font.Underline = False
      .ForeColor = vbBlack
    End With
  End If
End Sub

Private Sub Label1_DragOver(Source As Control, _
  X As Single, Y As Single, State As Integer)
  ' If the control is in dragmode, you can detect
  ' MouseLeave easily by observing the State parameter.
  ' Thanks to Mike Bolser for this observation!
  If State = vbLeave Then
    With Label1
      .Drag vbEndDrag
      .Font.Underline = False
      .ForeColor = vbBlack
    End With
  End If
End Sub

Private Sub Label1_MouseMove(Button As Integer, _
  Shift As Integer, X As Single, Y As Single)
  ' Entering dragmode on the first MouseMove
  ' allows easy detection of MouseLeave.
  If m_Active Then
    With Label1
      .ForeColor = vbBlue
      .Font.Underline = True
      .Drag vbBeginDrag
    End With
  End If
End Sub
```

Listing 1 Use this solution as a simple method to detect when the user moves the mouse off a windowless Label or Image control. The DragOver event's State parameter tells you immediately when to turn off special effects related to the mouse being over your control.

VB5, VB6 | Bundle Sounds Into a Resource DLL

```
Option Explicit

'  *** flag values for uFlags parameter ***
'  play synchronously (default)
Private Const SND_SYNC = &H0
'  play asynchronously
Private Const SND_ASYNC = &H1
'  silence not default, if sound not found
Private Const SND_NODEFAULT = &H2
'  name is a resource name or atom
Private Const SND_RESOURCE = &H40004
'  loop the sound until next sndPlaySound
Private Const SND_LOOP = &H8
'  don't stop any currently playing sound
Private Const SND_NOSTOP = &H10
'  purge non-static events for task
Private Const SND_PURGE = &H40
Private Declare Function PlaySound _
  Lib "winmm.dll" Alias "PlaySoundA" _
  (ByVal lpszName As String, ByVal hModule _
  As Long, ByVal dwFlags As Long) As Long
Private Declare Function PlaySoundData _
  Lib "winmm.dll" Alias "PlaySoundA" _
  (lpData As Any, ByVal hModule As Long, _
  ByVal dwFlags As Long) As Long
Public Enum ThemeSounds
  [SoundFirst] = 101
  Asterisk = 101
  Beep = 102
  CriticalStop = 103
  DefaultSound = 104
  EmptyRecycleBin = 105
  Exclamation = 106
  ExitWindows = 107
  Maximize = 108
  MenuCommand = 109
  MenuPopup = 110
  Minimize = 111
  ProgramError = 112
  Question = 113
  RestoreDown = 114
  RestoreUp = 115
  StartUp = 116
  [SoundLast] = 116
End Enum

Private Const defAsync As Boolean = True
Private Const defRepeat As Boolean = False
Private Const defYield As Boolean = False
Private m_Async As Boolean
Private m_Repeat As Boolean
Private m_Yield As Boolean
' ************************************************
'  Initialization
' ************************************************
Private Sub Class_Initialize()
  m_Async = defAsync
  m_Repeat = defRepeat
  m_Yield = defYield
End Sub

' ************************************************
'  Public Properties
' ************************************************
Public Property Let Async(ByVal NewVal As Boolean)
  m_Async = NewVal
End Property

Public Property Get Async() As Boolean
  Async = m_Async
End Property

Public Property Let Repeat(ByVal NewVal As
Boolean)
  m_Repeat = NewVal
End Property
```

Continued on next page.

Listing 2 Use this class to expose all the sounds stored in a DLL's resource file through a simple interface. Using this strategy, you can update or replace your application's sounds by recompiling and redistributing this single DLL file rather than the entire application.

Continued from previous page.

```
Public Property Get Repeat() As Boolean
  Repeat = m_Repeat
End Property

Public Property Let Yield(ByVal NewVal As Boolean)
  m_Yield = NewVal
End Property

Public Property Get Yield() As Boolean
  Yield = m_Yield
End Property

' ************************************************
'  Public Methods
' ************************************************
Public Function PlayResource(ByVal SndID _
  As ThemeSounds) As Boolean
  Dim Flags As Long
  Flags = SND_RESOURCE Or SND_NODEFAULT
  If m_Async Then Flags = Flags Or SND_ASYNC
  If m_Repeat Then Flags = Flags Or SND_LOOP
  If m_Yield Then Flags = Flags Or SND_NOSTOP

  PlayResource = PlaySound(CStr("#" & SndID), _
    App.hInstance, Flags)
End Function

Public Function StopPlaying() As Boolean
  Const Flags As Long = SND_PURGE
  StopPlaying = PlaySound(vbNullString, _
    App.hInstance, Flags)
End Function
```

default e-mail program to generate a new e-mail addressed to you.

Q Bundle Sounds Into an Application

Can I compile WAV files directly into an EXE or OCX? My application's EXE resides on a network server but runs in the client's workspace. It seems that placing the WAV files on the server and repeatedly calling them from the client would cause excessive and unwarranted network traffic. Placing the WAV files on every client machine is not an option in this case.

A The simple answer: Compile the WAV files into your application's resource file. I covered the basic details of this method in "Programming Techniques" [*VBPJ* January 1997]. That was two VB versions ago, and things have changed a little. A more elegant way to accomplish the same task today: Bundle the resource file within an ActiveX DLL and call this DLL from the client application. The DLL approach allows you to update the sounds easily without recompiling the entire application.

VB6 ships with a handy resource editor you can enable from the Add-In Manager. VB5 users need to download this add-in from the Microsoft Visual Studio Web site (see the Link sidebar). Within the resource editor, add the WAV files as "CUSTOM" resources. For each sound resource, assign the ID you want to use with it and reassign its type to "SOUND" by selecting the Properties dialog for that resource. After you add all your WAVs, press the Save button and dismiss the editor.

You can then rename the default class to CSounds. You might want to expose a public Enum for the resource IDs you assigned to your WAV files. This makes calling the proper resource from the client a cinch. The PlaySound API supports a variety of flags that alter the way sounds are played back. In my demo, I chose to support SND_ASYNC, SND_LOOP, and SND_NOSTOP. You'll find complete descriptions of these and the other flags in the PlaySound SDK documentation. You can toggle each of these flags using a simple class property.

You need to include two critical methods—one to start playing a sound, and another to stop any playing sound. The PlayResource method accepts the ID for the sound you'd like to play, then calculates the flags for PlaySound based on the settings of the class's Async, Repeat, and Yield properties. Finally, use a call to PlaySound to provide the return value for the method (see Listing 2). The StopPlaying method is even easier. It simply passes a null pointer (or "NULL" as it's referred to in the SDK docs) as PlaySound's Name parameter and the flag SND_PURGE.

Calling the finished DLL from your client couldn't be easier, once you set a project reference to the new DLL. You can either declare a persistent CSound reference in a form or global module so it's accessible at any time, or create an instance on demand. Set the flags as desired, and request the appropriate sound:

```
Private m_Snds As CSound

Private Sub DoingSomethingNoisy()
   m_Snds.Async = True
   m_Snds.Repeat = False
   m_Snds.PlayResource CriticalStop
End Sub
```

A word of caution: Be sure you call the StopPlaying method before exiting your application, especially if you find yourself using the Repeat property to loop the WAV continuously. Otherwise, your sounds might continue playing long after your app has died and gone away, leaving one very frustrated user behind. VBPJ

About the Author

Karl E. Peterson is a GIS analyst with a regional transportation planning agency and serves as a member of the **Visual Basic Programmer's Journal** Technical Review and Editorial Advisory Boards. Online, he's a Microsoft MVP and a section leader on several **VBPJ** forums. Find more of Karl's VB samples at www.mvps.org/vb.

DOWNLOAD FREE CODE

Download the code for this issue of **VBPJ free** from www.vbpj.com, part of the DevX family of Web sites (www.devx.com).

To get the free code for this entire issue, click on Locator+, the right-most option on the menu bar at the top of the home page, and type **VBPJ0899** into the box. (You first need to register, for free, on DevX.) The free code for this article includes all code listings, plus a demo using drag-and-drop to provide the equivalent of a MouseLeave event for a rollover hyperlink Label, and an ActiveX DLL example that bundles WAV files for client applications.

✪ To get the bonus code for this article, available to DevX Premier Club members, type **VBPJ0899AP** into the Locator+ field. The bonus code includes all the free code described above, plus a fully functional VB6 windowless UserControl that provides hyperlink functionality for both text and images.