Write Your Own Animation Control

by Karl E. Peterson

Replace the VB animation control with one that lets you read and write properties at run time.

Click & Retrieve Source CODE!

Why would a business app require animation? Consider the ultimate business app: Windows. Windows typically informs a user that a long process such as file copying is underway with a multiframe animation of flying paper and a progress bar tossed in for good measure. This approach is far superior to the older style of simply displaying an hourglass, which can leave the user wondering whether the computer is doing anything. The VB implementation of animation retains a couple of serious limitations, however: you must distribute an additional control, COMCT232.OCX, to include animations in your project; and COMCT232 doesn't support playing animations from a resource file.

In this article, I'll show you how to construct a simple class—CANIMATE.CLS—that supports playing AVI files and resources in your apps, but doesn't require you to ship an extra OCX (see Figure 1). CANIMATE.CLS takes advantage of a feature unique to animation controls by allowing the control to use a separate thread for animation processing. Bringing control of this thread into your application permits experimentation in ways that COMCT232 doesn't support. For example, if you break your program under the Integrated Development Environment (IDE), you can set properties of the class from the Immediate Window and watch how the new settings affect the animation. Operating on a separate thread also helps ensure that the animation continues without interruption, no matter how busy your application gets.

A fundamental flaw ("by design" in Microspeak) in the API prevents COMCT232 from using an application's resource file. If animations could be played directly from memory, as sound files can, this wouldn't be an issue. You must pass the instance handle (hInstance) your resources will be read from when Windows creates the window that plays your animation. But changes in Win32 have rendered instance handles less meaningful than they were in Win16. An instance handle is essentially the base address at which a module loads. But under Win32, each process has its own memory space, so modules in different processes can have the same instance handle. Therefore, the animation API can load resources only from the module that creates the animation window. For more complete details, see Knowledge Base articles Q149688 and Q103644 (http://www.microsoft.com/support).

This restriction means you must bring the window creation and control within the application, rather than delegating to a control. You can do this by using a class that

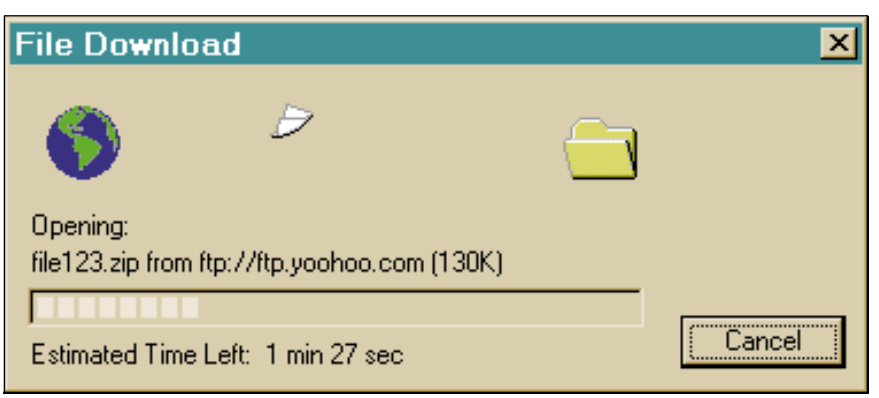

FIGURE 1 ***Implement Animation and Progress Bar Classes.*** *This sample applet demonstrates how to eliminate two OCX files by wrapping their functionality into classes within the project.*

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. Based in Vancouver, Washington, he's also an independent programming consultant specializing in ActiveX controls and a coauthor of* Visual Basic 4 How-To *(Waite Group Press). Online, he's a Microsoft MVP and a section leader in both* VBPJ *online forums. Contact Karl at karl@rtc.wa.gov.*

VB4 32-bit VB5

```
Option Explicit
Private Declare Function InitCommonControlsEx Lib _
  "Comctl32.dll" (iccex As tagInitCommonControlsEx) As _
  Boolean
Private Type tagInitCommonControlsEx
  lngSize As Long
  lngICC As Long
End Type
Private Const ICC_ANIMATE_CLASS = &H80
Private Const ANIMATE_CLASS = "SysAnimate32"
Private Const ACS_CENTER = &H1&
Private Const ACS_TRANSPARENT = &H2&
Private Const ACS_AUTOPLAY = &H4&
Private Const ACS_TIMER = &H8&
Private Const WM_USER = &H400&
Private Const ACM_OPEN = WM_USER + 100
Private Const ACM_PLAY = WM_USER + 101
Private Const ACM_STOP = WM_USER + 102
Private m_hWnd As Long
Private m_hWndParent As Long
Private m_AutoPlay As Boolean
Private m_Center As Boolean
Private m_Transparent As Boolean
Private m_Visible As Boolean
Private m_Playing As Boolean
Private m_AniResID As Long
Private m_AniFile As String
Private Sub Class_Initialize()
  Dim iccex As tagInitCommonControlsEx
  ' Initialize common controls DLL
  With iccex
     .lngSize = LenB(iccex)
     .lngICC = ICC_ANIMATE_CLASS
  End With
  Call InitCommonControlsEx(iccex)
End Sub
Private Sub Class_Terminate()
  Call AniDestroy
End Sub
Public Property Get hWnd() As Long
  hWnd = m_hWnd    ' ReadOnly!
End Property
Public Property Let AutoPlay(ByVal NewVal As Boolean)
  m_AutoPlay = NewVal
  If m_hWnd Then Call AniCreate
End Property
Public Property Get AutoPlay() As Boolean
  AutoPlay = m_AutoPlay
End Property
Public Property Let Center(ByVal NewVal As Boolean)
  m_Center = NewVal
  If m_hWnd Then Call AniCreate
End Property
Public Property Get Center() As Boolean
  Center = m_Center
End Property
Public Property Let FileName(ByVal NewVal As String)
  m_AniFile = NewVal
  m_AniResID = 0
  Call OpenAnimation
End Property
Public Property Get FileName() As String
  FileName = m_AniFile
End Property
Public Property Let ResourceID(ByVal NewVal As Long)
  m_AniResID = NewVal
  m_AniFile = ""
  Call OpenAnimation
End Property
Public Property Get ResourceID() As Long
  ResourceID = m_AniResID
End Property
Public Property Let Parent(ByVal NewVal As Long)
  m_hWndParent = NewVal
  If m_hWnd Then Call AniCreate
End Property
Public Property Get Parent() As Long
  Parent = m_hWndParent
End Property
Public Property Let Transparent(ByVal NewVal As Boolean)
  m_Transparent = NewVal
  If m_hWnd Then Call AniCreate
End Property
Public Property Get Transparent() As Boolean
  Transparent = m_Transparent
End Property
Public Sub AniPlay()
  If m_hWnd = 0 Then Call AniCreate
  m_Playing = CBool(SendMessage(m_hWnd, ACM_PLAY, _
     -1&, ByVal &HFFFF0000))
End Sub
Public Sub AniStop()
  Dim nRet As Long
  If m_hWnd Then
     nRet = SendMessage(m_hWnd, ACM_STOP, 0&, ByVal 0&)
     If nRet Then m_Playing = False
  End If
End Sub
Private Sub AniCreate()
  Dim AniStyle As Long
  Dim WasPlaying As Boolean
  ' Make sure we don't already have one
  If m_hWnd Then
     WasPlaying = m_Playing
     Call AniDestroy
  End If
  ' Combine style bits
  AniStyle = WS_CHILD Or WS_CLIPSIBLINGS
  If m_Visible Then AniStyle = AniStyle Or WS_VISIBLE
  If m_AutoPlay Then AniStyle = AniStyle Or ACS_AUTOPLAY
  If m_Center Then AniStyle = AniStyle Or ACS_CENTER
  If m_Transparent Then AniStyle = AniStyle Or _
     ACS_TRANSPARENT
  ' Create animation window
  m_hWnd = CreateWindowEx(0, ANIMATE_CLASS, _
     vbNullString, _
     AniStyle, m_Left, m_Top, m_Width, m_Height, _
     m_hWndParent, 0&, App.hInstance, ByVal 0&)
  ' Restart animation if was playing before window
  ' recreation
  If WasPlaying Then
     Call OpenAnimation
     Call AniPlay
  End If
End Sub
Private Sub OpenAnimation()
  Dim nRet As Long
  ' Make sure we have a window to
  ' work with, or stop any existing animations.
  If m_hWnd = 0 Then
     Call AniCreate
  Else
     Call AniStop
  End If
  ' Load animation.
  If m_hWnd Then
     If m_AniResID Then
        nRet = SendMessage(m_hWnd, _
        ACM_OPEN, 0&, ByVal m_AniResID)
     ElseIf Len(m_AniFile) Then
        nRet = SendMessage(m_hWnd, _
        ACM_OPEN, 0&, ByVal m_AniFile)
     End If
     If m_AutoPlay Then
        m_Playing = CBool(nRet)
     Else
        m_Playing = False
     End If
  End If
End Sub
Private Sub AniDestroy()
  If m_hWnd Then
     Call AniStop
     DestroyWindow m_hWnd
  End If
End Sub
```

LISTING 1 ***The CAnimate Class Replaces OCX.*** *This class wraps up nearly all existing functionality, and adds significant new functionality to the OCX-based implementation that comes with VB. Numerous elements of this class, including common API declarations and nonessential properties and methods, were omitted from this listing to conserve space.*

contains the animation control wholly within the base module, or by delegating to an in-process server that contains the needed resources. I used a class module because it is more flexible.

Coding CANIMATE.CLS is straightforward, with only a couple exceptions (see Listing 1). The class exposes public properties that parallel those exposed by COMCT232, but it goes a step further in allowing you to read and write at run time all properties other than hWnd. Inside the class, you can set three critical animation control styles—autoplay, centered, and transparent—at the time of window creation only. Thus, your class must be able to destroy and re-create the control to support changing these styles at any time—an approach the designers of VB could have chosen, but didn't.

CODE THE CLASS

On initialization, the class ensures that the Common Controls DLL itself is initialized. Your control uses this DLL to provide its functionality. The class then sets all the private member variables, which are used to track the control state and style, to their defaults. Space doesn't permit printing this portion of the class, but you can acquire this code from the free, Registered Level of The Development Exchange (see the Code Online box for details). The class defers creating the control until it's actually referenced. The Class_Terminate event calls the private AniDestroy method, which stops any running animation and destroys the control window.

Most public properties follow the same strategy when a new value is assigned to them—if a control window exists, the property procedure calls the private method AniCreate. Otherwise, the control window creation is deferred. AniCreate stores information on whether an animation is currently playing, then re-creates the control window if the control window exists. Then AniCreate combines the style bits for the new window, basing them on the public property settings, and calls CreateWindowEx to create a new animation control window. When re-creating a control, you must restore it to its original state, except for the style being changed. The last detail in AniCreate is to return the animation state to what it had been. Do this by restarting an animation that had been playing before the routine was called.

CANIMATE.CLS defers window creation for all properties, except those that assign which animation to play: Filename and ResourceID. This process prevents repeated window re-creations while the animation class is being set up. When the user of the class assigns either the Filename or ResourceID properties, the one that isn't set is assigned a null value, and a call is made to the private OpenAnimation method. Depending on whether the control window already exists, OpenAnimation either creates a new control window with a call to AniCreate, or stops a running animation by calling the public AniStop method. OpenAnimation then opens the requested animation and updates the m_Playing tracking variable.

Setting up an instance of the CAnimate class can be quite simple. If access to the class object is required at various points within a project, declare a form-level, or even global, instance of the class. Then assign the values to the relevant properties in the Form_Load (or other appropriate) event:

```
Set Animate = New CAnimate
With Animate
   .Parent = Me.hWnd
   .AutoPlay = True
   .Move 10, 15
   .ResourceID = 105
End With
```

You can download a complete demo of the CAnimate control, as well as the CprogBar class, from the free, Registered Level of The Development Exchange (see the Code Online box for details).

The class-based approach allows you to use AVIs compiled into your application's resource file. Well, sort of. Here you run into another somewhat frustrating Microsoft design. AniCreate uses App.hInstance in the call to CreateWindowEx, but App.hInstance returns the value of the IDE itself rather than the value of your application when running within VB's IDE. So, you face a dilemma. You can either load the AVI from a file during your debugging sessions, or you can forego the animation until you begin testing compiled EXEs. You can also compile the forms that use animation and animation resources into an ActiveX DLL.

CONTROL THE BACKGROUND COLOR

You'll find it trickier to control the background color of transparent animations. When an animation is opened, the control's parent window is sent a WM_CTLCOLORSTATIC notification message. This message allows the parent to set traits such as the background color or text color for a static control. VB5 forms and most VB5 controls typically respond with a reasonable action to this message: setting the static control's backcolor to the backcolor of the parent. VB4 doesn't behave quite as nicely, however, and neither do some common controls such as TreeView and ListView.

You must subclass the parent window to obtain full control over the backcolor of an animation. When your subclassing routine receives these notification messages, the hDC to be used for the animation is passed in wParam, and the hWnd is passed in lParam. Use the hWnd to validate whether the animation window is requesting colors, and use the hDC to set these colors (this code uses the popular freeware MsgHook custom control):

```
Private Sub MsgHook_Message(_
   ByVal msg As Long, _
   ByVal wp As Long, _
   ByVal lp As Long, result As Long)
   result = MsgHook.InvokeWindowProc(_
      msg, lp, wp)
   If msg = WM_CTLCOLORSTATIC Then
      If lp = Animate.hWnd Then
         Call SetBkColor(wp, _
            NewBackColor)
      End If
   End If
End Sub
```

Of course, if you have VB5, you can implement the subclassing code directly within your app. For more details, see the COMponent Builder column by Jonathan Wood, "Create a Subclassing Control" [*VBPJ* May 1997].

There you have it: an animation control that takes the place of the standard animation control, but requires no separate OCX, and sports increased functionality to boot. Controls such as this illustrate an important point: you don't have to live with something just because it came that way out of the box. With VB and a little ingenuity, you can make it work the way you want it to. ×

Code Online

You can find all the code published in this issue of* VBPJ *on The Development Exchange (DevX) at http://www.windx.com. For details, please see "Get Extra Code in DevX's Premier Club" in Letters to the Editor.

Write Your Own Animation Control
Locator+ Codes

Listings ZIP file, plus expanded code to cover sections that space didn't permit printing. The source also includes a working version of the demo, as well as the text of the KB articles cited in the article (free Registered Level): VBPJ0198

✪ ***Listings for this article, the files described above, plus an ActiveX DLL that implements an enhanced version of the CAnimate class. The enhanced version adds a BackColor property and enforces it by subclassing the parent window. You can also find an assortment of AVIs, some original and some that ship with VB, rolled into this DLL (subscriber Premier Level): CB0198P***