by Carl Franklin
and Ethan Winer

Video Clips In Visual Basic

This is your forum for addressing the intricacies of the Visual Basic language. Send in your questions, clever tips, and techniques. Visual Basic Programmer's Journal will pay $25 for any submission or question that we print. If your submission includes code, please send a disk along with your hard copy.

Mail submissions to Q&A Columnists, c/o Fawcette Technical Publications, 280 Second Street, Suite 200, Los Altos, CA, USA, 94022-3603. INTERNET: 71732.3233@CompuServe.COM. INTERNET: RobertVBPJ@AOL.COM.

DISPLAY AVI VIDEO FILES USING VISUAL BASIC

I cannot seem to get Microsoft Video for Windows AVI files to display in Visual Basic 3.0 Professional Edition, but I know it can be done. Can you define the position and size of the display window? What's involved?

—Keith Schaefer, Pleasant Grove, California

You can use the Multimedia control (MCI.VBX) to display an AVI file in Visual Basic. The MCI control is one of the most misunderstood controls, mostly because it does so much. It can play back digital audio files, MIDI files, movies, AVI files, and CD-audio, with provisions for controlling digital audio tape (DAT) machines, scanners, VCRs, and other devices.

At design time, the MCI control is a button bar that resembles the transport controls of a tape deck. There are buttons for rewind, play, record, stop, pause, and so on. The MCI control can be visible at run time or invisible, in case you want to control it with code.

Listing 1 shows a sample program in which an MCI control and a Visual Basic Picture control have been placed on a form. The program allows the user to display an AVI file in the picture box by manipulating the transport buttons on the MCI control. There are only four lines of code, located in the Form_Load procedure. Since the video is displayed in a VB picture control, you can move the video image anywhere on your form.

```
Sub Form_Load ()

    MMControl1.DeviceType = "AVIVideo"
    MMControl1.FileName = "C:\BIRD.AVI"
    MMControl1.Command = "Open"
    MMControl1.hWndDisplay = Picture1.hWnd

End Sub
```

Listing 1. Display video. This code allows you to display an AVI video clip in a Visual Basic application. Place an MCI.VBX control and a Picture control on a form, and enter this code into the Form_Load procedure. At run time you can click the play button (third from left) to play the video, and the rewind button (far left) to rewind, as well as pause (fourth from left), stop (third from right), and step play (fourth from right).

The MCI control's DeviceType property defines the file format to display (or record, as the case may be). Set this property to "AVIVideo" to use Video for Windows files. The next step is to open your AVI file. Set the FileName property to a valid AVI file name. Make sure you give the full path, unless the AVI file is in the current directory. Then, set the command property to "Open" to load the file. The next line of code tells the MCI control to display the video clip in Picture1, a standard Visual Basic picture control. If you omit this line, the video clip will be displayed in a separate window.

DROPPING FILES ON A FORM

How do I make a Visual Basic program that has the power to drag files to other applications the way File Manager can?

—Robert Lausevic, Sacramento, California

Currently, the File Manager version 3.1 is the only server application that supports the drag-and-drop protocol. However, future versions of Windows will enable an application to be a drag-drop server. A server can initiate a file drag-and-drop sequence, and a client can receive dropped files.

However, Windows has support for drag-and-drop clients. The code in Listing 2 enables a VB form to receive files dropped from the File Manager. When files are dropped on the form from the File Manager, the names are displayed on the form. This simple application requires only that you add the code in Listing 2 to a module in a new project. No custom controls are required. However, there are third-party add-on tools available that supply the same functionality.

In the Main() subroutine, after displaying the form, the first thing to do is tell Windows that Form1 is to receive WM_DROPFILES messages by calling DragAcceptFiles. If you do not make this call, the user will see a circle with a line through it when he attempts to drop a list of files on the form.

Carl Franklin develops programming tools in C, Visual Basic, and DOS Basic for Crescent Software. Carl also writes, plays, produces, and engineers music in his spare time.

Ethan Winer founded Crescent Software and is the author of Crescent's QuickPak Professional and P.D.Q. Basic add-on products. Ethan has written numerous magazine articles on programming and is a contributing editor to Visual Basic Programmer's Journal and PC Magazine.

Contact Ethan or Carl at Crescent Software, 11 Bailey Ave, Ridgefield, Connecticut 06877; 203-438-5300. Ethan's CompuServe number is 72241,63. Carl's is 70662,2605.

```
DefInt A-Z

Type POINTAPI
    x As Integer
    y As Integer
End Type

Type Msg
    hWnd As Integer
    Message As Integer
    wParam As Integer
    lParam As Long
    time As Long
    pt As POINTAPI
End Type

Dim Message As Msg

Declare Function PeekMessage Lib "User" _
    (lpMsg As Msg, ByVal hWnd As Integer,_
    ByVal wMsgFilterMin As Integer,_
    ByVal wMsgFilterMax As Integer,_
    ByVal wRemoveMsg As Integer) As Integer

Declare Function DragQueryFile Lib "SHELL.DLL"_
    (ByVal hDrop, ByVal Index, ByVal FileName$,_
    ByVal Size) As Integer

Declare Sub DragAcceptFiles Lib "SHELL.DLL"_
    (ByVal hWnd, ByVal fAccept)

Declare Sub DragFinish Lib "SHELL.DLL"_
    (ByVal hDrop)

Const WM_DROPFILES = &H233
Const PM_REMOVE = 1
Const PM_NOYIELD = 2

Sub Main ()

'-- Show the form
Form1.Show
Form1.Refresh

'-- Tell Windows that we want Form1 to receive
'-- drop file messages
Call DragAcceptFiles(Form1.hWnd, True)

'-- This loop occurs in the background
Do While DoEvents()

    '-- Ask for a dropfile message
    Success = PeekMessage(Message,_
        0, WM_DROPFILES, WM_DROPFILES,_
        PM_REMOVE Or PM_NOYIELD)

    '-- Did we get a drop file message?
    If Success Then
        '-- Yes, hDrop returns a handle to the
        '-- "dropped files" structure
        hDrop = Message.wParam

        '-- How many files were dropped?
        NumFiles = DragQueryFile(hDrop, -1,_
            FileName$, 127)

        '-- Clear the form
        Form1.Cls

        '-- Read all the file names.
        For i = 0 To NumFiles - 1

            '-- Initialize a string with blanks
            FileName$ = Space$(127)

            '-- Get the next file name. L returns
            '-- the string length.
            L = DragQueryFile(hDrop, i,_
                FileName$, 127)

            '-- Print the file name on the form.
            Form1.Print Left(FileName$, L)

        Next

        '-- DragFinish deletes the drop file
        '-- structure.
        Call DragFinish(hDrop)

    End If

Loop

End Sub
```

Drag and drop files from File Manager. Start a new project, add a new module, and add this code to the module to enable a form to receive files from the File Manager. At run time, select one or more files from the Windows File Manager and drop them on your form. The file names are displayed on the form. You must set the "Startup Form" to Sub Main in the Project Options menu for this code to work. In addition to working with a form, this code will work with any control that has an hwnd, such as a picture control.

The DoEvents loop that follows will occur whenever the system is not busy. The PeekMessage API call returns a value of -1 (True) if a received message falls between the specified lower and upper range. In this case, you only want to see one message: WM_DROPFILES, so both the lower and upper range message values are set to WM_DROPFILES. If one or more files are dropped on the form, PeekMessage will return a nonzero value.

When a WM_DROPFILES message is received, the wParam integer element of the Message TYPE variable contains a handle to a list of file

```
SUB Reboot ()
    DEF SEG = 0
    POKE &H473, &H12
    POKE &H472, &H34
    DEF SEG = &HFFFF
    CALL ABSOLUTE(0)
END SUB
```

Restart DOS. This VB-DOS subroutine will reboot the PC when called.

```
Declare Function ExitWindows Lib "User"_
    (ByVal dwReturnCode As Long,_
    ByVal wReserved As Integer) As Integer

Const EW_REBOOTSYSTEM = &H43
Const EW_RESTARTWINDOWS = &H42

Sub Command1_Click ()
    '-- Restart Windows
    Dummy = ExitWindows(EW_RESTARTWINDOWS, 0)
End Sub

Sub Command2_Click ()
    '-- Reboot the PC
    Dummy = ExitWindows(EW_REBOOTSYSTEM, 0)
End Sub
```

Restart your engines! This code can be used to declare and call the ExitWindows API function to reboot the PC or restart Windows.

names (hDrop). This handle has no real value, except that it can be passed to the DragQueryFile function to return the number of files dropped on the form and their names. If you pass DragQueryFile a value of -1 as the second parameter (Index), DragQueryFile returns the number of files that were dropped on the form.

To return a particular file name, pass the zero-based file number as the second parameter, and a fixed-length string as the third parameter; and the string will return the file name. The return value of DragQueryFile is the length of the file name. You can simply use the Left$ function to return only the file name from the fixed-length string.

REBOOTING THE PC

REBOOTING IN WINDOWS IS EASY, BUT IN VB-DOS YOU HAVE TO DO A LITTLE POKING AROUND

FROM DOS OR WINDOWS

How can I reboot the PC from VB-DOS or Windows, and is there a way to just restart Windows from a VB app without rebooting?
—Ross Lally, Waterford, Connecticut

Rebooting the PC is one of those things that isn't required often, so there isn't much help available on the subject. However, when you need to reboot, there's no substitute.

Rebooting in Windows is easy; there is an API call that reboots. But in VB-DOS, you have to do a little poking around.

The code in Listing 3 is a VB-DOS routine that reboots the PC, and Listing 4 is a VB-Windows example of both rebooting the PC and restarting Windows using the ExitWindows API function.

Note the use of POKE in the VB-DOS Reboot subroutine (see Listing 3). If you didn't already know, POKE sets the value of a byte at a particular address in memory. Intel 8(x86-based systems use a full address of four bytes. Two bytes define the location of the segment (or 64K block), and two bytes define the offset within that segment. Since POKE accepts only the address or offset portion, you must define the segment. The DEF SEG = XXXX statement is required to define the segment within 64K of the POKE address. Therefore, the address that the first POKE statement accesses is actually 0000:0473 (hex).