Use Jet for Realtime Device Control

Click & Retrieve Source CODE!

Database apps using realtime data acquisition and device control offer real opportunities to VB programmers.

by Roger Jennings

Everybody's talking client/server and Internet, but many heavy-duty VB applications involve neither. Take real-time data acquisition and device control applications. VB4 makes writing these a breeze.

Applications for supervisory control and data acquisition (SCADA) range from chemical process automation to factory-floor production monitoring to multimedia. Most commercial SCADA applications run under UNIX today, but Windows NT is coming on strong. For example, General Electric's Fanuc Automation North American subsidiary, a leading factory monitoring and control systems maker, announced last December that it is porting the firm's Cimplicity factory automation software to NT. It currently runs on various UNIX flavors as well as DEC VMS. I expect other SCADA systems builders to follow GE Fanuc's lead over this year and next.

The SCADA market looks promising for VB developers, especially those who know manufacturing technology and process control. And you can take the same principles off the factory floor and into places like TV studios, which do scads of realtime data acquisition and control for video editing and broadcasting. VB4's database connectivity and data-bound controls really help you store, retrieve, and analyze all the time-based data associated with video production.

As a consequence Multi-Media Computing Solutions was able to write Video Magician entirely in VB, producing a full-featured, tape-based video editing app. ASC Audio Video's Virtual Recorder, used as a live "sideline editor" at Super Bowl XXX, stored digital video on multiple 9Gb drives to provide almost instant recall of a series of the game's "big plays."

Video Magician and Virtual Recorder handle video data differently, but both store timecodes in a database. Timecodes provide the beginning (in point) and end (out point) of videoclips. You can then automatically replay the videoclips in any sequence. Video Magician uses the Jet database engine to store the data needed to log and edit multiple video projects. High-end interactive video systems, such as Microsoft Media Server, use client/server RDBMSs for real-time video scheduling and distribution.

I recently wrote a simple VB4 database application, VidLog, to control a videotape recorder for logging and automatic playback of several hours of source videotape for an upcoming TV documentary. Several commercial DOS and Windows apps support logging videotapes, but none had everything I needed to automate the digitizing process, which employs Adobe Premiere and Interactive Images' Plum PCI video capture card.

It took me several days to write and test VidLog, but I quickly recovered my investment during the final capture process—mainly by saving rental time on a $250 per day broadcast-quality videotape player. I needed the player to digitize Sony Betacam SP footage shot in rural Mississippi.

You can use VidLog's low-cost real-time image capture and storage coding technique for any multimedia database app that handles live analog video content. And even if you aren't doing video apps, you can use this technique for other forms of realtime control.

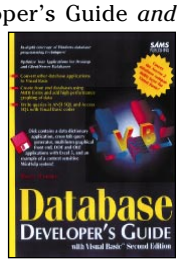
USE VISCA TO GET THE TIMING RIGHT

I based my app on Sony's popular Video Systems Control Architecture (ViSCA) protocol. Windows 95 has a built-in ViSCA driver (MCIVISCA.DRV) for the Media Control Interface (MCI), so you can use the 16-bit mciSendString or 32-bit mciSendStringA functions to issue English-like commands to ViSCA devices, which MCI calls the *vcr* device type. A full discussion of the MCI command string syntax for the *vcr* device type comes with the "Media Control Interface" section of the Win32 SDK on the MSDN Development Library CD-ROM.

Before you can use either function, you must declare two function prototypes (mciSendString and mciGetErrorString) contained in 32-bit WINMM.DLL or 16-bit MMSYSTEM.DLL. Both of these libraries are installed during Windows 95 setup (see Listing 1). Set up the *vcr* device (alias Vdeck in VidLog) in the Form_Load subprocedure of the main VidLog form (see Listing 2 and Figure 1).

VidLog's behind-the-scenes data acquisition activity centers on a Timer control that interrogates the Vdeck device to determine tape position. Videotape timecode is based on the Society of Motion Picture and Television Engineers (SMPTE) HH:MM:SS:FF format, where FF represents video or movie frames. SMPTE timecode resembles the time headers in SCADA messages. And every 100 milliseconds (about three video frames), VidLog's timSMPTE_Timer event handler interro-

Roger Jennings, a principal of OakLeaf Systems, consults on Windows client/server database front ends. Roger is the author of Using Access 95, Special Edition *and* Discover Windows 3.1 Multimedia *for Que; and* Access 95 Developer's Guide *and* Database Developer's Guide with Visual Basic 4 *for Sams. He also co-wrote Sams'* Database Developer's Guide with Visual C++ 4. *Roger is now writing* Using Windows NT Server 4.0, Special Edition *and* Using Windows Desktop Video, Special Edition *for Que. Reach him via CompuServe at 70233,2161 or on The Microsoft Network as Roger Jennings.*



gates the Vdeck, which returns the current tape position as an SMPTE HH:MM:SS:FF string to the txtReturn text box.

When SCADA systems aren't interrupt-driven, most use a similar interrogation-response system to obtain data from remote sensors or transducers. If the in and/or out points of the video clip aren't set, I use a routine to write the current tape position continuously to the databound txtIn and/or txtOut text boxes:

```
Private Sub timSMPTE_Timer()
   'Load timecode for in and out point
      'text boxes; timer interval is set
      'at 100 ms (3+ frames)
   Call SendCommand("Status Vdeck Position")
   'Don't overwrite clip's in & out points
   If Not fMarkIn Then
      txtIn.Text = txtReturn.Text
   End If
   If Not fMarkOut Then
      txtOut.Text = txtReturn.Text
   End If
End Sub
```

If you mark the in and out points with the Mark In and Mark Out command buttons while the tape is moving, you can get up to a four-frame offset. Instead, use the tape position buttons at the bottom of the form as the primary means to obtain frame-accurate clips.

Use the Pause, Forward, and Reverse buttons to move to a specific frame with the step command (due to space limitations, this listing is posted on the *Visual Basic Programmer's Journal* online sites described at the end of this column). Or you can type the timecodes in the In and Out text boxes. To visually check in and out points by playing the clip, just click the Test Clip button.

GRAB PICONS WITH SNAPPY

After completing the 32-bit version of VidLog, I decided to add the ability to capture and store Picons in the VidLog database to visually identify each clip. (Picon = picture + icon—small

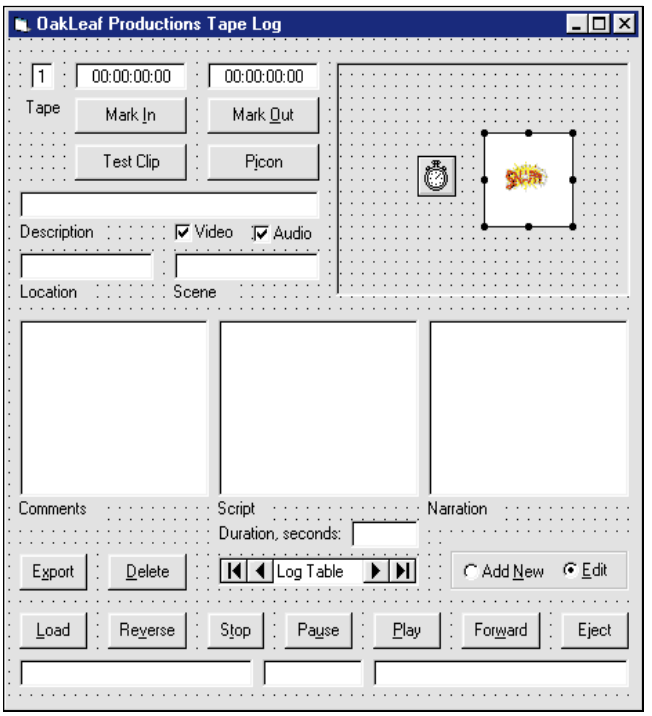


FIGURE 1 ***You Can't Misplace this VCR Remote Control.*** *The main VidLog form includes command buttons at the bottom of the form to control almost every function of the Vdeck, including export, load, reverse, play, and eject.*

VB4

```
Option Explicit
#If Win32 Then
'Windows 95 MCI functions follow
  Declare Function mciSendString Lib
     "winmm.dll" Alias "mciSendStringA" _
     (ByVal lpstrCommand As String, ByVal _
     lpstrReturnString As String, _
     ByVal uReturnLength As Long, _
     ByVal hWndCallback As Long) As Long
  Declare Function mciGetErrorString Lib "winmm.dll" _
     Alias "mciGetErrorStringA" (ByVal dwError As Long, _
     ByVal lpstrBuffer As String, _
     ByVal uLength As Long) As Long
#Else
  Declare Function mciSendString Li "mmsystem" _
     (ByVal lpstrCommand As String, _
     ByVal lpstrReturnString As String, _
     ByVal uReturnLength As Integer, _
     ByVal hWndCallback As Integer) As Long
  Declare Function mciGetErrorString Lib "mmsystem" _
     (ByVal wError As Long, ByVal lpstrBuffer As String, _
     ByVal uLength As Integer) As Integer
#End If
Public mciCommand  As String
'Command
Public mciReturn   As String * 256
'Return string
Public mciErrString As String * 256
'Error string
Public mciError    As Long
'Error number
#If Win32 Then
  Public mciReturnLen As Long
'Length of return string
  Public mcihWnd      As Long
'Callback window handle
  Public mciErrLen    As Long
'Length of error string
  Public mciErrResult As Long
'Return value of error
#Else
  Public mciReturnLen As Integer
  Public mcihWnd      As Integer
  Public mciErrLen    As Integer
  Public mciErrResult As Integer
#End If
```

LISTING 1 ***Have it Both Ways.*** *The multimedia MCI functions come in 16-bit (MMSYSTEM.DLL) and 32-bit (WINMM.DLL) flavors. Use compiler directives and function aliases to write the declarations section of a module so you can use it in both 16-bit and 32-bit VB4 apps. MMSYSTEM.DLL primarily uses integers while WINMM.DLL requires long integers for arguments and return values. Use this code in any VB4 app that works with MCI command strings, including programs for manipulating waveaudio and digitalvideo devices (WAV and AVI files).*

VB4

```
Private Sub Form_Load()
'Set up initial conditions
   Me.Left = (Screen.Width - Me.Width) / 2
   Me.Top = (Screen.Height - Me.Height) / 2
   fMarkIn = True
   txtIn.Font.Bold = True
   fMarkOut = True
   txtOut.Font.Bold = True
   dtcLog.EOFAction = 0
   Call SendCommand("Open vcr1 alias Vdeck")
   'Open the ViSCA device
   Call SendCommand("Status Vdeck media present")
   'Check for tape in drive
   If txtReturn.Text <> "true" Then
   'No tape in drive
      MsgBox "Please insert a tape _
         in the drive.", 0, "Drive Not Ready"
   End If
   Call SendCommand("Set Vdeck time mode detect")
   'Turn on ViSCA RCTC detection
   If fDropFrame Then
Call SendCommand("Set Vdeck time format smpte 30 drop")
'Set the time code to SMPTE dropframe (29.97 fps)
Else
Call SendCommand("Set Vdeck time format smpte 30")
'Set the time code to SMPTE nondropframe (30.00 fps)
End If
   Call SendCommand("Set Vdeck index timecode")
   'Set screen display to timecode
   Call SendCommand("Index Vdeck on")
   'Turn on timecode display, if available
   Call SendCommand("Set Vdeck preroll 00:00:00:20")
   'Set preroll time to 20 frames
End Sub
Private Sub SendCommand(mciCommand As String)
   'Sends MCI command string, receives return value
   'Retrieves error message if an error is encountered
   mciReturnLen = Len(mciReturn) - 1
   mciErrLen = Len(mciErrString) - 1
   txtCommand.Text = mciCommand
   mcihWnd = 0
   mciError = mciSendString(mciCommand, _
      mciReturn, mciReturnLen, mcihWnd)
   txtReturn.Text = Left$(mciReturn, (mciReturnLen - 1))
   If mciError > 0 Then   'An error occurred
      mciErrResult = mciGetErrorString(mciError, _
         mciErrString, mciErrLen)
      txtError.Text = LTrim$(Str$(mciError)) & "  " & _
         Left$(mciErrString, (mciErrLen - 1))
   Else
txtError.Text = "OK" 'Command executed
   End If
End Sub
```

LISTING 2 ***Set Up the VCR MCI Device When You Load the Main Form.*** *Call the SendCommand subprocedure with an MCI command string argument to check that a tape is in the VCR, then have the VCR detect the timecode of the tape. Hi8 camcorders and VCRs use non-dropframe timecode, while U.S. and Japanese (NTSC) DV camcorders use SMPTE dropframe timecode (29.97 frames per second). So you must specify the type of timecode in use. If your VCR can display timecode over the video signal, the Index device on command enables this feature (called a "timecode window burn"). The SendCommand subprocedure executes MCI commands and sends return values or error messages, if any, to unbound text boxes at the bottom of the main form.*

bitmaps of representative frames of video). If you have a video overlay card, you can use the MCI overlay device to save the content of the video buffer to a BMP file, then use the LoadPicture function to transfer the data from the file to the Picture property of the Image control.

For a more elegant (and probably lower cost) approach, use Play, Inc.'s $180 (estimated street price) Snappy video capture device and the SNAPPY.VBX Picon capture custom control, available in the SNAP_SDK.EXE file on Play's Web site, http//www.play.com (see Figure 2). Snappy plugs into your PC printer port. Its RCA connectors provide composite analog video input from a VCR or camcorder and optional loop-through output to a television monitor. Installation takes five minutes or less. And it doesn't need an open expansion slot or link to your graphics card's feature connector, unlike high-end video overlay cards. True, SNAPPY.VBX limits you to 16-bit Visual Basic 4.0 and a Jet 2.5 (Access 2.0) MDB file to store the clip data and images, but 16-bit and 32-bit OLE Controls are in the works.

Setting Snappy control's SnapFileType property to 1 - DIB on Clipboard places a copy of the captured device-independent bitmap on the Clipboard. Set SnapSize to 1 - Still, SnapSource to 0 - Tape, SnapState to 0 - Off, Stretch to True, VideoThru to False (unless you connect a TV monitor to Snappy's video output connector), and Visible to False.

My Snappy-adapted VidLog can capture a 320 x 240 pixel image, paste the image into the data-bound 180 x 120 pixel imgPicon control (with the Stretch property set True), then save the data from imgPicon in a LongBinary field of the database:

```
Private Sub cmdSnap_Click()
   'Place deck in pause mode for capture
   Call SendCommand("Pause Vdeck wait")
   'Copy a DIB to the Clipboard with Snappy
   snp320x240.SnapState = 2
   'Paste the data to the Image control
   imgPicon.Picture = Clipboard.GetData()
   'Update the record in place
   dtcLog.Recordset.Edit
   dtcLog.Recordset.Update
End Sub
```

By the way, don't try adding code to the PicSnapped event-handler to paste the DIB to the Image control. PicSnapped triggers when Snappy grabs the video data, long before the capture process is complete.

Various commercial apps enhance Snappy's realtime video image acquisition capability. Examples include one for employee identification badges using low-cost color printers,

FIGURE 2 ***Snapping Picons Aids Clip Identification.*** *To keep track of video images, you can click the Picon command button, and the SNAPPY.VBX custom control will grab the current video frame and copy it to the Clipboard as a device-independent bitmap (DIB).*

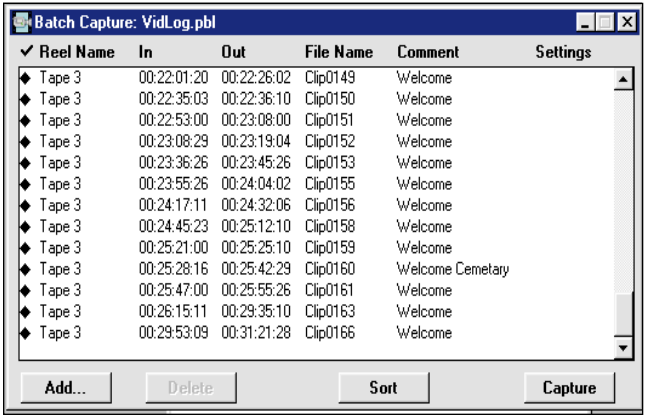


FIGURE 3 ***Command Buttons Engage the Gears.*** *The event-handlers for the Click event of the Test Clip, Eject, Reverse, Stop, Pause, Forward, and Eject command buttons control the Vdeck's tape drive mechanism.*

and others for capturing images for industrial quality control, automating surveillance systems (including sharp blowups of surveillance tapes), and publishing semi-live still pictures on the Internet.

For Internet apps, use Snappy's variable-quality JPEG format (SnapFileType = JPEG) and save the image to a JPG file linked to your home page. MCI commands can start and stop the recording process if your camcorder has a LANC connector as well as

take advantage of the camera-direct output to produce sharper images. Snappy's rendering of video content, even from low-cost camcorders, offers substantially better quality than still pictures shot with today's $1,000 consumer-grade electronic cameras. Images captured from a Sony DCR-VX1000 DV camcorder in Snappy's maximum-resolution (1500 x 1125 pixel) mode with the 4 - Highest value for the SnapQuality property rival pictures shot on 35-mm film.

BATCH THE CAPTURE PROCESS

I built VidLog to create an ASCII batch capture file for Adobe Premiere from the Log table of VidLog.mdb. Premiere uses the ViSCA driver for capturing AVI clip files from Sony DCR-VX1000 camcorder and an Abbate Video (now part of Videonics Inc.) Video ToolKit driver for RS-422 control of the Betacam SP playback deck.

Premiere's batch capture file import format requires a tab-separated text file with Reel Name (tape number), In, Out, and File Name, with optional Comment and Settings fields (see Figure 3). Controls are bound to the Recordset object of the dtcLog Data control, so I used a Recordset clone to create the required text file (due to space limitations, this listing is posted on the *Visual Basic Programmer's Journal* online sites described at the end of this column). I captured video content with the Plum card using predetermined settings for an average data rate of about 3.5 MB per second, including a 16-bit, 44.1-kHz stereo sound track.

This data rate corresponds to a Motion-JPEG compression ratio of roughly 5:1, considered to be Betacam SP quality. You can fill a 4.3Gb drive with 20 minutes of compressed video in AVI format, so the production version of VidLog's text file export subprocedure includes a test for required free drive space before exporting the capture file. Vidlog adds a drive letter prefix to the AVI file name, determined by the estimated accumulated size of the clip files.

I designed VidLog for a highly specialized data acquisition and control application, but the overall design typifies simple SCADA-type device control programs. And you too can write effective realtime control applications with VB4 and the Jet database engine. As Windows NT gains ground as an operating system for real-time SCADA factory-floor monitoring and process control, I think that 32-bit VB4 will replace C/C++ as the SCADA programming language of choice. You'll probably find your first opportunities in non-critical logging applications and prototyping process control programs.

Once these programs demonstrate that 32-bit VB can create industrial-quality code, VB developers will close the loop with realtime process control applications that deliver the availability and reliability generally attributed to C programs ×

To download the sample code for this article, as well as additional text (including a discussion of ViSCA devices you can buy), visit The Development Exchange on the World Wide Web at http://www.windx.com, or the VBPJ *CompuServe Forum, or MSN site. For details, see "How to Reach Us" in Letters to the Editor.*