by Mark Novisoff and Craig Leach

Spice Up Your Apps With Multimedia

Now that multimedia is firmly entrenched in the hearts and minds (not to mention eyes and ears) of Windows programmers and users, you might want to spice up your applications with a little sound or two.

If you have VB 2.0 Professional Edition or later, you can use the MCI.VBX custom control to play sounds through a sound board or Microsoft's PC-Speaker driver (more on the PC-Speaker driver later). While the MCI control provides a number of convenient services and sports an attractive cassette deck-style interface, it can be overkill if you simply want to play sound files without user interaction, or trigger the system event sounds defined in the Windows Control Panel.

WINDOWS 3.1 API LETS YOU ADD SOUND WITHOUT USING THE VB 2.0 MCI CUSTOM CONTROL.

The SndPlaySound API function from the MMSYSTEM.DLL file included with Windows 3.1 and the Microsoft Multimedia Extensions for Windows 3.0 allows you to play sounds without using a custom control. By avoiding the MCI control, your application will use less resources and will not require you to distribute MCI.VBX.

In case you were wondering, MCI.VBX also makes calls to MMSYSTEM.DLL, so this technique is simply a case of eliminating the middle man.

To use the SndPlaySound function in your application (for all versions of VB), add the following Declare statement and Constants to your global or code module:

```
Declare Function SndPlaySound Lib "MMSYSTEM.DLL" _
        (ByVal lpszSoundName$, ByVal wFlags%) _
        As Integer
'Flags used by SndPlaySound:
Global Const SND_SYNC = &H0000
Global Const SND_ASYNC = &H0001
Global Const SND_NODEFAULT = &H0002
Global Const SND_LOOP = &H0008
Global Const SND_NOSTOP = &H0010
```

The first parameter, lpszSoundName$, specifies the name of the sound to play. This can be a wave audio file name (*.WAV) or a system event name as defined in the [Sounds] section of the WIN.INI file. While you can configure the system event sounds through the Sound section of the Windows Control Panel, note that the actual system event names are different than the ones that appear in the Control Panel. The standard event names are:

Name To Use With SndPlaySound	Name In Control Panel
SystemAsterisk	Asterisk
SystemDefault	Default Beep
SystemExclamation	Exclamation
SystemExit	Windows Exit
SystemHand	Critical Stop
SystemQuestion	Question
SystemStart	Windows Start

For example, you can specify a sound file such as "c:\windows\chord.wav," or use "SystemDefault" to play whatever sound the user has defined for that event in the Control Panel. In fact, by adding your own event definitions to the [Sounds] section of WIN.INI, you can allow your end users to redefine the sounds used exclusively by your application. One example of this technique is America Online for Windows, which adds to your [Sounds] section with events such as "Welcome," "Goodbye," and "You Have Mail." If you don't like the sounds defined for these events, you can assign new sounds for them in the Control Panel.

The second parameter, wFlags%, defines various options available with SndPlaySound:

Flag	Description
SND_SYNC	Play sound synchronously and return only when sound ends.
SND_ASYNC	Play sound asynchronously and return immediately after sound starts.
SND_NODEFAULT	Do not play the default sound if lpszSoundName$ is not found.
SND_LOOP	Play sound continuously until SndPlaySound is called again with lpszSoundName$ set to NULL. When looping sounds with this flag, you must also specify SND_ASYNC.
SND_NOSTOP	Return immediately if a sound is currently playing.

If the sound was played successfully, the function returns True (-1). If the sound was not found, or if the SND_NOSTOP flag was specified

Mark Novisoff is the founder and president of MicroHelp Inc., and is the creator of several VB add-on products. His company sells products for VB, QuickBasic, PDS, and Windows. Mark is also a contributing editor to Visual Basic Programmer's Journal. You can contact Mark at 4359 Shallowford Industrial Parkway, Marietta, Georgia, 30066.

Craig Leach is senior executive vice president of Capitol Information Service Inc. in Trenton, New Jersey and has written commercial and custom software in Basic since 1983. Contact Craig on CompuServe at 70244,2634.

and a sound was already playing, the function returns False (0).

The following example shows how easy it is to play a standard system event sound, as well as a specific wave sound file:

```
Sub Command1_Click ()
    wFlags% = SND_SYNC Or SND_NODEFAULT
    'Play the sound currently assigned to
    'the "SystemQuestion" event in WIN.INI:
    x% = SndPlaySound("SystemQuestion", wFlags%)
    'Play a specific wave file:
    x% = SndPlaySound("c:\windows\chord.wav", wFlags%)
End Sub
```

Because we specified two sounds back to back, the SND_SYNC flag was used to prevent the first sound from being interrupted by the second sound. When playing only a single sound (such as playing the "SystemQuestion" or "SystemAsterisk" sound immediately before invoking the MsgBox function), use the SND_ASYNC flag to return immediately to the next statement in your program while the sound is still playing.

If you don't have an MCI-compatible sound board at your disposal, you can achieve a crude approximation with your PC's internal speaker by using Microsoft's PC-Speaker driver (SPEAKER.DRV). This driver is available from CompuServe, the Microsoft Download Service (206-936-6735), and other online services. Although Microsoft has stated that

THE SND_SYNC FLAG PREVENTS THE FIRST SOUND FROM BEING INTERRUPTED BY THE SECOND SOUND.

the PC-Speaker driver is not compatible with the MCI control and the Windows 3.1 Media Player "applet," you can use it with the SndPlaySound function.

Finally, if you insist on using your lowly PC speaker to play sounds and want full MCI-compatibility, check out Aristosoft's Wired For Sound—its speaker driver is functionally equivalent to Microsoft's SPEAKER.DRV but will also work with MCI.VBX, Media Player, and others.

KEYWORD OF THE ISSUE: SENDKEYS

If you need to control another Windows application but don't want to wrestle with messy Dynamic Data Exchange commands, the SendKeys statement may be a workable alternative. Simply stated, SendKeys sends one or more keystrokes to the active window (your program or another Windows application) as if they had been entered at the keyboard. The syntax for SendKeys is as follows:

```
SendKeys keytext [,wait]
```

The keytext parameter is a string containing a list of keystrokes to send to the active window. With a few exceptions (which we'll cover below), this string may contain any displayable characters (A-Z, 0-9, etc.). For nondisplayable characters such as Enter or Tab, you must use special codes such as {Enter} or {Tab}. These codes are too numerous to list here and may be found under the SendKeys statement in your VB documentation and online help.

The wait parameter determines when your program regains control after the SendKeys statement is executed. If wait is True, all of the keystrokes in the keytext string must be processed by the active window before control is returned to your program. If wait is False or omitted, your program gains control as soon as the keystrokes are sent—even if they are waiting to be processed. The following example shows how to use SendKeys to send keystrokes from your program back to itself (similar to playing a macro):

```
Sub Command1_Click ()
    'When this button is clicked, set the focus to
    'a text box on the current form and "automatically"
    'type a string in the text box:
    Text1.SetFocus
    Keystrokes$ = "This was typed by SendKeys.{Enter}"
    'The {Enter} code above is processed as the Enter
    'key; the code itself does not appear
    'in the text box.
    SendKeys Keystrokes$
End Sub
```

If you want to send keystrokes to another Windows application, you must first activate the other application's window. If you are running the other application via the Shell statement, specify the Shell windowstyle parameter as 1, 3, 5, or 9 to give the Shelled application the focus. For example:

```
Sub Command2_Click ()
    'Activate the Notepad application and
    'give it the focus:
    x% = Shell("notepad.exe", 1)
    'Send keystrokes to Notepad and wait for processing
    SendKeys "Sending keys to Notepad from _
              VB!{Enter}", True
    'Instruct Notepad to save the file:
    '%F = Alt-F File Menu
    'S = Save File option
    'c:filename.ext = name entered in Save As box
    '{Enter} = Enter
    SendKeys "%FSc:filename.ext{Enter}", True
    'Activate our program again
    AppActivate App.Title
End Sub
```

If the other application is already running and you know its title (it appears in the Windows Task List), you can set the focus to that application with the AppActivate statement, as in this example:

```
Sub Command3_Click ()
 'Display the Help/About box in the Program Manager:
 AppActivate "Program Manager"
 'Send Alt-H and A; wait for processing:
 SendKeys "%HA", True
End Sub
```

As mentioned, there are some exceptions to the displayable characters you may include in the keytext parameter. The following characters have special meaning to the SendKeys statement:

```
+ ^ % ~ ( ) [ ] { }
```

If you wish to include these characters in your string, you must enclose them in braces, as in {%} and {+}. In addition, SendKeys may report an "Illegal function call" if one of the following is not enclosed in

Go to next page.

Go to next page.

braces:

- An unmatched parenthesis ( ) or brace { }.
- A bracket [ ].
- Braces containing an undefined character sequence, such as {abc}.

SendKeys has other limitations to consider. First, keystrokes sent with SendKeys are case sensitive. Some applications may not process uppercase and lowercase versions of the same letter in the same manner. For example, if you send "%f" (lowercase f) to Word for Windows 2.0, the keystrokes are interpreted as Alt-F (which displays the File Menu). If you send "%F" (uppercase F), Word interprets the keystrokes as Alt-Shift-f (lowercase f), which does not display the File Menu.

Second, keystrokes sent with SendKeys must be exact. Naturally, if you fail to supply keystrokes to an application when further input is required, the application will continue to wait for the appropriate input. For example, the Notepad example will work correctly the first time it is executed. But on the second attempt, the file "c:\filename.ext" already exists, and therefore Notepad will ask if you want to overwrite an existing file with that name. Since our SendKeys string does not include a response to the overwrite prompt, Notepad has not actually saved the file when our SendKeys statement has finished executing.

Finally, SendKeys can cause mouse problems with IBM PS/2s running under Windows 3.0. Microsoft has confirmed that when the SendKeys statement is executed in Windows 3.0 on an IBM PS/2 Model 50, Model 50z, Model 60, or Model 80, Windows behaves erratically when you move the mouse. Fortunately, this problem has been corrected in Windows 3.1.

QUIRK: WHEN "TRUE" IS NOT EQUAL TO "NOT FALSE"

Once upon a time, it was necessary for Basic programmers to create True and False variables at the start of each program. (True/False are now reserved words in VBWin/VBDOS, with the values already defined for you.) If you are still performing this ancient ritual in QB/PDS, be sure to set True equal to -1. Otherwise, you may run into this quirk:

```
'Example for QB/PDS only:
DefInt A-Z
True = 1      'Notice this is not -1
False = 0
Value = True
If Value = Not False Then
    Print "Value is True"
Else
    Print "Value is False"
End If
```

If you think this example would display "Value is True," you're wrong. Basic evaluates "Not False" as -1, which is not equal to the True value (1) defined at the top of the program. Therefore, the line "If Value = Not False Then" is actually evaluated as "If 1 = -1 Then." n