



SOUND IT OUT

Hear what you've been missing! With one simple statement, GS BASIC programmers can play any digitized-sound file.

By **JOE ABERNATHY**

YOUR MISSIVES SAY, "LET THERE BE GRAPHICS AND sound," so tune your dial and let's round out your GS' "natural" talents. In addition to the powerful QuickDraw graphics primitives discussed in August ("Tools to Draw On," p. 78), you can add to your BASIC library an efficient and easy-to-use sound-playback capability, handy in education, special-needs, and entertainment programming.

Sound reproduction is within ready grasp of all IIgs programmers. Best of all, you don't even need to understand the technical details to use these sound-software tools. Using Micol Advanced BASIC, TML BASIC, or AC/BASIC, simply type in the accompanying **Program listings** and start listening.

If you're interested in the theory, however, sound presents one of the few occasions on which our normally friendly BASIC compilers begin to balk. Yet learning to use BASIC to program sound can help you master many lesser IIgs programming tasks.

The GS can record and reproduce multichannel sound that's similar in quality to that of expensive music synthesizers. It does so by using a special outboard logic chip, the Ensoniq, which is dedicated solely to sound tasks. For true multichannel sound, you'll also need an expansion stereo card and external speakers.

To help you access the Ensoniq's sound capabilities, Apple has provided a built-in set of software utilities, collectively called the Sound Manager tool set. In turn, your BASIC compiler provides a tool interface to make the Toolbox sound calls available to you as though they were a standard part of the BASIC language—more or less.

ORGANIZATION PAYS

The GS can produce concert-quality sound, but achieving this magnificent effect requires an advanced music degree. So rather than fully master the GS' musical talents, we'll focus herein on a simple goal—digitized-sound playback. This procedure meets the needs of most programmers, as well as those who want to establish a foundation for

further learning. In examining this technique, we also can practice the more general skill of how to coax any program from conception to completion.

As I've emphasized repeatedly in this column, the first step in writing any program is planning. Decide what your program must do, what it should do, and what it would be nice to do. Then strike a balance weighing your goals, abilities, and time. The following

simple outline describes such a plan for sound playback:

- I. Select a file to play.
 - A. Use *Open File* dialog in desktop environment.
 - B. Request user input if in text environment.
- II. Load sound file into memory.
 - A. Reserve memory for sound.
 - B. Load file into reserved memory.
- III. Prepare for playback.
 - A. Create table of sound parameters.
 - B. Fill it with information about this sound.
- IV. Play the sound.
 - A. Start sound tools.
 - B. Begin playback.
 - C. Play until done.
 - D. Release memory.
 - E. Shut down sound tools.

The IIgs Toolbox Sound Manager contains two utilities for sound reproduction: `_FFStartSound` and `_FFSoundDoneStatus`. So our second outline lists the specific sound tool calls you'll need along with related calls and data structures:

- I. Sound Manager tool calls
 - `_FFStartSound` to start sound playback
 - `_FFSoundDoneStatus` to monitor playback progress
- II. Other tool calls
 - `_NewHandle` to get the sound into memory
- III. Related functions and DOS calls
 - `OPEN` file
 - `GET` to read file
 - `LOF` to determine file size
 - `CLOSE` file
- IV. Data structures
 - `RECORD` sound-parameter block
 - `NUMERIC VAR` playback speed
 - `MEMORY HANDLE` for sound

•RECORD file information

The next step is to implement your outline. Begin with the four data structures (part IV) that normally go at the top of a BASIC program. The exact form they'll take depends on your compiler.

What are data structures? IIGS Toolbox calls often require that you feed them a lot of information so that they can react properly to your particular program. They often return quite a lot of information, as well. Data structures represent a way of organizing this information into a logical grouping with which you can work easily.

The Sound Manager is no exception. Ensoniq needs to know how to reproduce your wave form. The sound-parameter table provides that information, including such data as playback speed and volume. By manipulating the sound-parameter block, you can make a soundtrack loop continuously in the background or play a succession of wave forms, for instance.

According to the *IIGS Toolbox Reference Manual*, the sound-parameter block must keep track of the following items, among others:

waveStart: address of wave form to play
waveSize: waveform size in pages
freqOffset: playback rate
docBuffer: DOC-buffer start address
bufferSize: DOC-buffer size
nextWavePtr: pointer to next wave's parameter block
volSetting: DOC-volume setting

BASIC has no way to represent a hodgepodge of information such as that shown above. Standard BASIC arrays consist exclusively of mathematical or string variables. But the sound-parameter block, indeed most data structures, combine integers with memory pointers and more.

All three IIGS BASICs have a special way of feeding data structures to the tool call. Micol lets you do a pseudo POKE, while TML and AC/BASIC support a modified array that imitates a structure data type. The accompanying **Program listings** included with this column show the actual implementations for each language.

Pascal and C are rich with data structures, making the IIGS BASIC compilers pale in comparison. This one fact makes tool programming, including sound, much more difficult in BASIC than it needs to be. The compiler's publishers maintain this is because BASIC programmers don't like powerful data structures resembling those of higher languages; I've yet to meet a programmer who's of that opinion. Perhaps you should write a letter to the language publishers if you share these concerns.

Once you've taken care of data structures, you can just look over your outline to see the most straightforward way of putting together the actual source code. You'll likely want procedures to set up the environment and to let someone select a sound file to play, actually play the file, and do a clean shutdown of the environment.

In a program that implements the desktop, you'd probably get the most use out of a *play sound* option presented in a pull-down menu. Our **Program listings** implement this technique in TML and AC/BASIC.

For text-based programs or background-sound reproduction, a sub-routine such as that shown in the Micol Advanced BASIC code, **Listing 1**, is more appropriate. The subtle differences lie only in how the sound-production routines fit into your application's general structure.

After adding a sound-player utility to your software library, you may wonder how you'll obtain the sounds you'll want to play. For sound effects, which perhaps comprise the bulk of sound playback, you can ▶

Listing 1. Micol Advanced Play.Sound source code.

```
PROGRAM PrintLib

{ ----- }
{ Show CDA Studio }
{ By Joe Abernathy BASIC }
{ (C)1989, inCider Magazine ===== }
{ Version 1.2, July 11, 1989 }
{ All Rights Reserved. }
{ Compiler: Micol Advanced BASIC }
{ ----- }

{ Sound Data structures: }

DIM FFBUFFER% (20)
FFBank% = 0
H_Bank% = 0
H_Address% = 0
FFAD% = 0
No_MEM! = FALSE

{ channel-generator-type word: }
ffSynthMode = 0001 { Free-form Synthesizer Mode }
noteSynthMode = 0002 { Note Synthesizer Mode }
{ END OF SOUND DATA STRUCTURES }

Done! = FALSE { init quit flag }

{ ----- }
{ PROC DoSound
This routine is based on a program
(C)1988 by Ron Lewin of Micol Systems.
Used with permission, and our thanks.
----- }

PROC DoSound [Size, SndFile$, Speed%, Vol%]
IF FILE (SndFile$) THEN BEGIN
TOOLBOX (2, 28: 0000, 0000; No_Bytes%, No_Banks%)
No_Bytes = No_Bytes%
IF No_Bytes < 0 THEN No_Bytes = No_Bytes + 65536 { Twos complement }
Total_Bytes = No_Banks% * 65536 + No_Bytes
IF Total_Bytes < Size THEN Size = Total_Bytes
LSB_BlockAdr = ADDR(LSB_Blocksize%)
MSB_BlockAdr = ADDR(MSB_Blocksize%)
LSB_Size1 = (Size MOD 65536) MOD 256
Pages = (Size MOD 65536) / 256
Banks = INT(Size / 65536)
POKE LSB_BlockAdr, LSB_Size1
POKE LSB_BlockAdr+1, Pages
POKE MSB_BlockAdr, Banks
ID% = PEEK (238) { Get User ID for using the Memory Manager }
AttribAd = ADDR(Attrib%) { Set attributes for the requested block of }
POKE AttribAd,128 { memory. }
TOOLBOX(2,9: 0000, 0000, MSB_Blocksize%, LSB_Blocksize%, ID%, ...
... Attrib%, 0000, 0000; H_Address%, H_Bank%)
IF PEEK (202) = 0 THEN BEGIN
Tmp_Addr = H_Address% { get result returned by tool call above }
IF Tmp_Addr < 0 THEN Tmp_Addr = Tmp_Addr + 65536
L_Address = (H_Bank% * 65536) + Tmp_Addr
R_LSB = ADDR (Address%)
R_MSB = ADDR (Bank%)
POKE R_LSB, PEEK (L_Address)
POKE R_LSB + 1, PEEK (L_Address + 1)
POKE R_MSB, PEEK(L_Address + 2)
Tmp_Addr = Address%
IF Tmp_Addr < 0 THEN Tmp_Addr = Tmp_Addr + 65536 { Twos complement }
Full_Add = (Bank% * 65536) + Tmp_Addr
BLOAD SndFile$, Full_add, Size { Load sound file into memory }
FFAD% = ADDR (FFBUFFER%())
FFBank% = PEEK (202)
FFLoc = ADDR (FFBUFFER%())
FOR Buffer_Size = 0 TO 7 UNTIL 2.0 ^ (8 + Buffer_size) > Size
NEXT Buffer_size
Buffer_size = Buffer_size - 1

{ Consult the accompanying article for a discussion of
this wave parameter table: }

POKE FFLOC, PEEK (L_Address)
POKE FFLOC + 1, PEEK (L_Address + 1)
POKE FFLOC + 2, Bank%
POKE FFLOC + 3, 00
POKE FFLOC + 4, Pages
POKE FFLOC + 5, Banks
```

Continued

buy cassette tapes at any good record store, then record the sounds you need using hardware such as Sonic Blaster or FutureSound. You can also program wave forms directly—if you're feeling brave.

Also, note that you can use the central procedure producing the actual sound in a variety of fashions. If you're writing a game, for instance, the player probably won't select a sound file; he or she just needs to hear the drip of water in a dungeon or the sound of engines on the raceway. You can easily edit these sound utilities for such uses.

Before discussing language-specific features of sound, kudos are in order for several people whose help made this column much more effective. They include Applied Engineering's Phil Montoya, who wrote the software for Sonic Blaster and offered to share his insights; Absoft's Lee Rimar, Jeff Knaggs, and D.K. Keppner, who provided both source code and exceptional product support; Micol Systems' Ron Lewin, who provided insightful source code, along with a sympathetic ear; and TML Systems' newly hired Apple product manager, Vince Cooper, who let me use source code to which TML owns the copyright, and also provided ready product support.

MICOL ADVANCED BASIC

The most interesting aspect of the Micol sound implementation (**Listing 1**) is the demonstration of how it uses PEEK and POKE commands to support tool calls. This technique, which trades familiarity for difficulty of use, will arise whenever you use the Toolbox. The good news is that Micol Systems has been listening to concerns on this subject and will include significant tool improvements in a forthcoming revision to the compiler.

You can add **Listing 1** to the *inCider* ShowFile utility, use it as a permanent entry in your library of software tools, or both. The procedure DoSound is the actual work routine you should add to your library. The procedure DoPlaySound demonstrates a more elegant way of handling user input than demonstrated previously.

Micol BASIC lacks two features important to sound playback: a way of determining a file's length and a way of reading GS/OS directory information concerning the file to be played, especially playback speed, which by default is stored in the auxiliary file-type field.

A compiler revision, which should be available as you read this, will let you open GS/OS DiReCtory files and read them a line at a time, so that you can access the auxiliary file-type information. If you're anxious to work more with sound programming, implementing this capability would be your first project.

AC/BASIC

This version of the sound player (**Listing 2**) suffers from an inability to play long wave forms. This situation arises because AC/BASIC doesn't deal well with the advanced data structures required for this type of sound handling. (In fact, that's the purpose of AC/BASIC—to shield you from such things as memory pointers.)

Theoretically, however, you could study the sound-parameter table in the manual along with this month's examples from the other languages to devise a way around this. I haven't explored it well enough to provide expert advice, but the solution is likely to require an external assembly-language subroutine. Unless your sound-playback feature must support long digitized sounds, I doubt whether the reward justifies the effort involved in this procedure.

On the other hand, a worthy enhancement would be the ability to ►

Continued

```
POKE FFLOC + 6, Speed%
POKE FFLOC + 7, 00
POKE FFLOC + 8, 00
POKE FFLOC + 9, 00
POKE FFLOC + 10, 07
POKE FFLOC + 11, 00
POKE FFLOC + 12, 00
POKE FFLOC + 13, 00
POKE FFLOC + 14, 00
POKE FFLOC + 15, 00
POKE FFLOC + 16, Vol%
POKE FFLOC + 17, 0
ENDIF
```

```
{ FFStopSound:
  PUSH genMask; bit 0-15 indicates generator(s) to stop }

  TOOLBOX (8, 15, 2) { FFStopSound: Used here to init sound tools }
```

```
{ FFStartSound:
  PUSH word genNumFFSynth (channel, generator, and type. Usually $0101)
  PUSH long pBlockPtr (ptr to ff param block created above)
```

```
chan-gen-type word:
15-12, DOC channel num; 11-8, gen number $0-$E;
7-4 must equal 0; 3-0, $1 for free-form synthesizer, $2 = note synthesizer
$0101 = DOC channel 0, generator #1, reserved val 0, ff synthesizer 1
... and ... $0101 (base 16) = 257 (base 10), which we use below.
```

```
pBlockPtr:
Note that Micol syntax requires two PUSHed parameters, FFBank% and
FFAD%, to fulfill the one formal Toolbox pointer requirement. This is
caused by Micol's Applesoft-like implementation of tool calls using
word-length pseudo-PEEKs and POKEs. }
```

```
  TOOLBOX (8, 14: 257, FFBank%, FFAD%) { FFStartSound }
  REPEAT
```

```
{ FFSoundDoneStatus:
  PUSH word space, genNumber;
  PULL genDoneFlag; Boolean var, is TRUE if done playing }
```

```
  TOOLBOX (8, 20: 00, 01; Finished%) { FFSoundDoneStatus }
  UNTIL Finished% < 0 OR Continual
ENDIF
```

```
  TOOLBOX (2,16: H_Bank%, H_Address%) { release memory }
```

```
ENDPROC { DoSound }
```

```
* This procedure gets user input to feed to underlying
* sound file player, DoSound:
```

```
{ ..... }
{ PROC DoPlaySound }
{ Control loop for sound wave player. }
PROC DoPlaySound
  HOME
  GOSUB DrawLines
  x$ = "< Press <RETURN> for Main Menu >"
  x% = (80-LEN(x$))/2
  VTAB (24)
  HTAB (x%)
  PRINT x$;
  HTAB (1)
  vtab (6)
  PRINT "___ Play A Sound File ___"
  VTAB (8)
  INPUT "Enter Pathname -> " ; andname$
  andname$ = UPPER$(andname$)
  IF andname$ = "" THEN Terri = TRUE
  Input "Enter Playback Speed -> " ; playrate%
  IF playrate% < 10 OR playrate% > 1200 THEN Terri = TRUE
  INPUT "Enter File Size in Bytes -> " ; length
  IF length = 0 THEN Terri = TRUE
  myvolume% = 255 { default -- loudest available }
  IF NOT Terri THEN BEGIN
    MUSIC (1,50,1) { Start sound tools }
    QUIET (1) { and silence them. }
    GOSUB DoSound [length, andname$, playrate%, myvolume%]
  ENDIF
ENDPROC { DoPlaySound }
```

Continued

Continued

```
{ ..... }
{ Sub this for MainScreen procedure: }

PROC MainScreen
HOME
GOSUB DrawLines
HTAB (28)
VTAB (6)
PRINT "T -> Type a File"
HTAB (28)
PRINT "P -> Print a File"
HTAB (28)
PRINT "H -> Print File With Header"
HTAB (28)
PRINT "C -> Show Catalog"
HTAB (28)
PRINT "S -> Set GS/OS Prefix"
HTAB (28)
PRINT "I -> Environment Detail"
HTAB (28)
PRINT "D -> Play Digitized Sound"
HTAB (28)
PRINT "V -> View SHR Picture"
HTAB (28)
PRINT "Q -> Quit"
ENDPROC { MainScreen }

{ ..... }
{ Main loop }

PROC Main

' integrate Play Sound option into
' menu event interpreter:

GOSUB MainScreen
HTAB (1)
VTAB (1)
GET a$
a$ = UPPER$(a$)
IF a$ = "Q" THEN BEGIN { Quit }
Done! = TRUE
ELSE IF a$ = "D" THEN BEGIN { Play Sound }
Terror! = FALSE
REPEAT
GOSUB DoPlaySound
UNTIL Terror!
Terror! = FALSE
ELSE IF a$ = "V" THEN BEGIN { SHOW PICTURE }
Terror! = FALSE
REPEAT
GOSUB ShowPic
UNTIL Terror!
Terror! = FALSE
ELSE IF a$ = "S" THEN BEGIN { Set Prefix }

{ ..... }

ENDIF
ENDPROC { Main }

{ ..... }
{ End of v1.3 revisions to ShowSrc CDA }
```

Listing 2. AC/BASIC Play Sound source code.

```
' File: inCider.Shell Version 1.2 Revisions -- July 5, 1989
' By Joe Abernathy (Requires full inCider.Shell)
' (C)1989, inCider IIGS BASICS
' All Rights Reserved.
' Compiler: AC/BASIC for the Apple IIGS.
' Compile with "no default menus" and "no default window" options selected.

' (C)1989, inCider. Portions of this program include material copyrighted
' (C)1988, Absoft Corp. All other copyrights acknowledged. Special thanks
' to Lee Rimar, Jeff Knaggs and D.K. Keppner at Absoft.

' Version 1.2 adds ability to play digitized sound files. Also adds new
' pack, unpack procedures to replace those in the AC/BASIC manual.
```

Continued

Continued

```
' Version 1.2 additions to data structures:
DEFINT a-z
DIM ipack(31),i(43) ' Packed, unpacked instrument arrays

' Revised to add option for sound playback.

menuproc: ' Interpret menu events
menunum = MENU(0) ' Read which menu
itemnum = MENU(1) ' Read which item
IF menunum = 1 THEN ' .. FILE menu
IF itemnum = 1 THEN ' New
GOSUB 10
ELSEIF itemnum = 2 THEN ' Edit
GOSUB 20
ELSEIF itemnum = 3 THEN ' Delete
GOSUB 30
ELSEIF itemnum = 4 THEN ' Print
GOSUB 40
ELSEIF itemnum = 5 THEN ' Type File
GOSUB 50
ELSEIF itemnum = 6 THEN ' Quit
GOSUB 60
END IF
ELSEIF menunum = 2 THEN ' GOODIES menu
IF itemnum = 1 THEN ' Show picture
GOSUB 70
ELSEIF itemnum = 2 THEN ' Play a sound file
GOSUB 80
END IF
END IF
RETURN

' This routine added:

' Play back a digitized wave form.

80:
WINDOW 2
f$ = "null"
WHILE f$ <> ""
f$ = FILES$(1) ' Open file dialog
IF f$ <> "" THEN
WAVE 0 GET,ipack ' Get the instrument array
UNPACK ipack(),i() ' Unpack instrument bytes into integers
i(33)=1 : i(39)=1 ' Set starting DOC address (page 1)
i(35)=2 : i(41)=2 ' Set DOC mode to one-shot
PACK i(),ipack() ' Pack instrument integers into bytes
OPEN "R",1,f$,1 ' Open the disk waveform file
length! = LOF(1) ' Get length of waveform in the file
IF length! > 32767 THEN length! = 32767 ' max allowable size

DIM waveform((((length!/256)+1)*256/2) ' Get space for waveform array
' The long calculation ensures that the array
' size is a multiple of 256 since the minimum
' DOC page size is 256 bytes.

BLOAD #1, waveform, length! ' Load the disk waveform into the array
CLOSE #1
waveform(length!/2)=0 ' Force end to waveform
WAVE 0, waveform, INT((length!/256)+1,1), ipack ' Set the instrument array,
' put waveform into DOC.

mypitch% = 0
WHILE mypitch% = 0
INPUT "Enter playback speed: ", mypitch%
WEND
dur=70 ' Set playback duration
SOUND 0,dur,dur,mypitch%,255 ' Play the digitized sound on voice 0
PRINT
PRINT "Click mouse to continue..."
WHILE MOUSE(0) <> 0 ' Clear event queue of
WEND ' ghost mouse clicks
WHILE MOUSE(0) = 0 ' Await real mouse click
WEND
WAVE 0 ' Turn off voice 0
ERASE waveform ' Free memory
END IF
WEND ' Loop till cancel clicked in dialog

WINDOW CLOSE 2
MENU
RETURN

' Menu option added for sound playback.

SUB DoMenu ' Create menu bar
FOR p = 1 TO 6 ' Eliminate screen flicker
FOR e = 0 TO 12 STEP 4 ' by whitening-out the menu bar
PALETTE p,e+0,1,1,1 ' before building it.
```

Continued

Continued

```

NEXT
NEXT
MENU 1,0,1,"File"      ' Build FILE menu
MENU 1,1,1,"New"       ' and its entries ...
MENU 1,2,1,"Edit"
MENU 1,3,1,"Delete"
MENU 1,4,1,"Print"
MENU 1,5,1,"Type"
MENU 1,6,1,"Quit"
MENU 2,0,1,"Goodies"   ' Goodies menu header
MENU 2,1,1,"View Picture" ' View SHR picture
MENU 2,2,1,"Play Sound" ' Play a sound file
FOR p = 1 TO 6          ' Restore original palette ..
  PALETTE p,e+0,0,0,0
NEXT
NEXT
END SUB

```

' This routine added:

' -----
 ' Unpack a byte-oriented waveform into integers.
 ' Use this UNPACK subprogram rather than the one in the manual.

```

SUB unpack(in%(1), out%(1))
FOR count%=0 TO 21
  out%(2*count%+1)=in%(count%) AND &HFF00
NEXT
SWAP BYTES,out%
FOR count%=0 TO 21
  out%(2*count%)= in%(count%) AND &H00FF
NEXT count%
END SUB

```

' This routine added:

' -----
 ' Pack an integer-oriented waveform into bytes.
 ' Use this PACK subprogram rather than the one in the manual.

```

SUB pack(in%(1), out%(1))
FOR count%=0 TO 21
  out%(count%)=in%(2*count%+1)
NEXT
SWAP BYTES,out%
FOR count%=0 TO 21
  out%(count%)=in%(2*count%) OR out%(count%)
NEXT
END SUB

```

' The End. (inCider.Shell v1.2 revisions)

Listing 3. TML BASIC Play.Sound source code.

```

' File: IDG.TML.SHELL      Version 1.3 Revisions -- 7/10/89
' By Joe Abernathy        (Requires full IDG.TML.SHELL)
' (C)1989, inCider.      =====
' All Rights Reserved.
' Compile with TML BASIC V1.10 for the Apple IIGS

' Change data structures to match this (stdfile array moved to DESKTOOLS,
' sound wave form array added):

DIM WindowTitle!(10)      ' Stores title for a window
DIM anEventRecord!(19)    ' TaskRecord data structure
DIM aLocInfoRec!(15)      ' LocInfo record for QuickDraw
DIM ParamBlkSoundRec!(13) ' sound wave parms (FFparms wave table)

```

' Add the Sound Manager to the list of LIBRARY commands:

```
LIBRARY "Sound"      ' Sound Manager
```

' Add a play sound option to the list of MENUDEF commands:

```

' MENUDEF 14,DoQuit      ' Quit the application
' MENUDEF 15,DoFont      ' Choose Font dialog
' MENUDEF 16,DoSound     ' ADD THIS LINE

```

' Application-specific menu items start with 17 (corresponding to
 ' 267 in SetUpMenus below).

Continued

Continued

' -----
 ' Add goodies menu to menu startup procedure:

```

DEF PROC SetUpMenus
PROC GoodiesMenu      ' ADD THIS LINE
PROC StdFontMenu
PROC StdEditMenu
PROC StdFileMenu(1)
PROC StdAppleMenu
PROC DrawMenus
END PROC

```

' -----
 ' Move the old open file procedure from here into the file IDG.DESKTOOLS.
 ' This makes it available to other procedures, such as DoSound below.

```

DoOpen:
PROC DoOpenFile      ' PROC is now in DESKTOOLS.

```

' If a valid file is selected the var "proceed%" will be set to
 ' a positive value. "auxfiletype%", "filename\$", and "fullpathname\$"
 ' also are set.

RETURN 0

' -----
 ' PROC DoSound -- Play a disk-based sound wave form.

```

DoSound:
PROC DoOpenFile      ' Standard Open File dialog.
IF proceed% <> 0 THEN
  PlaySpeed@ = 0
  OPEN FileName$, AS #10
  PlaySpeed@ = AUXID@
  Size@ = EOFMARK(10)
  CLOSE
  IF Speed@ <> 0 THEN PlaySpeed@ = Speed@
  IF PlaySpeed@ = 0 THEN PlaySpeed@ = 200
  PlaySpeed% = ((32 * PlaySpeed@) / 1645)
  CALL MaxBlock
  blockSize@ = R.STACK@ (1)
  IF Size@ > blockSize@ THEN Size@ = blockSize@
  DIM DYNAMIC mysound!(Size@)
  OPEN FileName$, AS #10, Size@
  GET #10; mysound!(0)
  CLOSE
  Pages% = (Size@ MOD 65536) / 256
  waveStart@ = VARPTR(mysound!(0))
  SET (ParamBlkSoundRec!(0)) = waveStart@
  SET (ParamBlkSoundRec!(2)) = 0
  SET (ParamBlkSoundRec!(3)) = Pages%
  SET (ParamBlkSoundRec!(5)) = Playspeed%
  SET (ParamBlkSoundRec!(7)) = 0
  SET (ParamBlkSoundRec!(9)) = 0
  SET (ParamBlkSoundRec!(10)) = 0
  SET (ParamBlkSoundRec!(12)) = 0
  SET (ParamBlkSoundRec!(13)) = 0
  SndBlkPtr@ = VARPTR(ParamBlkSoundRec!(0))
  _FFStartSound (257, SndBlkPtr@)

```

' IMPORTANT: See accompanying text to make following
 ' code functional:
 ' DO
 ' CALL FFSoundDoneStatus%(0) ' Lets sound finish playing
 ' Finished% = R.STACK%(1)
 ' UNTIL Finished% <> 0
 ERASE mysound!
 END IF
 RETURN 0
 ' -----
 ' END of v1.3 revisions to IDG.TML.SHELL

Listing 4. TML BASIC desk tools.

```

' File: IDG.DESKTOOLS      Version 1.3 Revisions -- 7/10/89
' (C)1988, TML Systems Inc. (Requires full IDG.DESKTOOLS)
' All Rights Reserved.      =====
' Modified with permission.
' Compiler: TML BASIC V1.10 for the Apple IIGS

```

' Version 1.3 starts up the Sound Manager tools, setting aside memory
 ' for them; adds a goodies menu with a "Play Sound" option; and adds
 ' a library procedure for the standard "Open File" dialog box.

' Add these data structures:

Continued

```
Continued
DIM GoodMenuStr(200)      ' Goodies menu def string.
DIM aReplyRecord(149)    ' Reply record for SFGetFile

' This procedure is used to load and start up the Toolbox tool sets.
' The Mode% parameter indicates which graphics mode to use. The value
' should be either 320 or 640. The PrintTools% parameter indicates
' whether the tool sets necessary for using the Print Manager should
' be loaded or not.

DEF PROC StartUpTools(ScreenMode%,LoadPrintTools%)

' ... Add a line to StartUpTools to start the sound manager,
' fitting in with the existing code like this:

' LIBRARY LOAD "Scrap"
' LIBRARY LOAD "Desk"
' LIBRARY LOAD "Sound"      ' Add this line.

' Change memory manager startup:

' Start the memory manager
AppMemoryID% = EXFN_MMStartUp

' Allocate 7 pages of memory in bank 0 for tool set globals
' (4 pages already allocated by TML.) (1 page = 256K bytes)
ToolZeroPageH@ = EXFN_NewHandle(7*256,AppMemoryID%,-16379,0)
ToolZeroPageP@ = VAR(ToolZeroPageH@,3)
ToolZeroPage% = EXFN_LoWord(ToolZeroPageP@)

' Start the printing tools if requested
' IF LoadPrintTools% THEN
'   _QDAuxStartUp
'   _ListStartUp
'   _FMStartUp(AppMemoryID%,ToolZeroPage%+1024)
'   _PMStartUp(AppMemoryID%,ToolZeroPage%+1280)
'   _SoundStartUp(ToolZeroPage%+1536) ' ADD THIS LINE
' ...

END PROC StartUpTools

' Tool shut down routine.

DEF PROC ShutDownTools
' GRAF OFF
' _SoundShutDown      ' ADD THIS LINE
' IF svLoadPrintTools% THEN

' ...

END PROC ShutDownTools

' Add this procedure before the PROC StdFontMenu:
' -----

DEF PROC GoodiesMenu      ' Create goodies menu
LOCAL MenuStr$
MenuStr$ = ### " >> Goodies W510"
MenuStr$ = MenuStr$ + "==Play Sound W26610"
SET(GoodMenuStr(0)) = "MenuStr$
_InsertMenu(EXFN_NewMenu(VARPTR(GoodMenuStr(1))),0)
END PROC GoodiesMenu

' Add this procedure at the end of the file:
' -----

' Display the standard "get file" dialog box.
' This call will display all files. To display only files of a particular
' type, use a TypeList with the appropriate file types specified.
' After making the call, we get the selected filename from aReplyRecord.

DEF PROC DoOpenFile
_SFGetFile(100,50,"Open which file:",0,0,VARPTR(aReplyRecord(0)))
proceed% = VAR(aReplyRecord(0),2)
IF proceed% THEN      ' IF NOT proceed, CANCEL was clicked.
filetype% = VAR(aReplyRecord(2),2)
auxfiletype% = VAR(aReplyRecord(4),2)
FileName$ = VAR(aReplyRecord(6),7,15)
FullPathName$ = VAR(aReplyRecord(32),7,127)
END IF
END PROC DoOpenFile
' -----
END LIBRARY
```

read a sound file's auxiliary file type to let the program determine a sound's playback speed. To do this, open the DIRectory file containing the sound file as a random-access file. Input each line as a string, and search for lines beginning with a 0 (zero) byte, which indicates that a filename follows. Bytes 0-15 of this string will contain the name of the file; byte 16 will contain the file type; and bytes 30-31 will contain the auxiliary file type as an unsigned integer. (Don't sweat it if none of this makes sense to you. It's intended for fairly sophisticated programmers.)

TML BASIC

Listings 3 and 4 show how to add a Goodies menu with sound-playback option to your TML desktop program shell. However, I discovered a bug in TML that hampers the code's effectiveness. In the procedure DoSound in Listing 3, you can see that the loop calling _FFSoundDoneStatus to monitor the progress of playback is disabled. This is because _FFSoundDoneStatus is either misspelled or missing from TML's libraries. (Because of this bug, I couldn't test the TML sound code fully, but all major features work as they should.)

As this column was going to press, TML Systems couldn't find the solution to this library problem. You should be able to call TML today, though, and get the spelling of the call the libraries use. Then you can substitute that spelling in Listing 3 and remove the REMark characters in the DO/WHILE loop.

PRODUCT INFORMATION

AC/BASIC

Absoft Corp.
2781 Bond Street
Rochester Hills, MI 48307
(313) 853-0050
\$125

Microl Advanced BASIC

Microl Systems
9 Lynch Road
Willowdale, Ontario M2J 2V6

Canada

(416) 495-6864
\$145

TML BASIC

TML Systems
8837-B Goodbys
Executive Drive
Jacksonville, FL 32217
(904) 636-8592
\$125

There are several other points of interest in the source code. In Listing 4, I added startup and shutdown procedures for the Sound Manager, including the allocation of direct page space using the Memory Manager. I've rewritten the *Open File* dialog and moved it from the IDG.TML.SHELL file to IDG.DESKTOOLS so that you can call it from any procedure. (The Play Sound option demonstrates this.) Also, I added the Goodies menu, showing how you can construct and use custom pull-down menus.

FURTHER EXPLORATION

Stereo-sound playback is one attractive feature you can add easily to your sound tools. You just need to specify the channel and generator number on which a sound should be played.

To conform to Apple standards software must use channel 0 (zero) for the right-hand stereo signal and channel 1 for the left. See *Apple Developer Technical Notes 19* and 37 for more information (Apple Computer, 20525 Mariani Ave., Cupertino, CA 95014, 408-996-1010).□

WRITE TO JOE ABERNATHY C/O INCIDER, 80 ELM ST., PETERBOROUGH, NH 03458. ENCLOSE AN SASE IF YOU'D LIKE A REPLY.