Recording Sounds Directly From a Device

There are a number of routines that are intended for use by applications that need more control over the recording process (such as the ability to intercept sound input data at interrupt time). You can open a sound input device and read data from it by calling these low-level **Sound Manager** routines. Several of these routines access information through a sound input parameter block, **SPB**.

The Listing below shows how to set up an **SPB** structure and record synchronously using the **SPBRecord** function. This procedure takes one parameter, a handle where the recorded sound data is to be stored. It is assumed that the handle is large enough to hold the sound to be recorded.

Recording directly from a sound input device

```
// Assuming inclusion of MacHeaders
#include <Sound.h>
#include <SoundInput.h>

// Used for fixed math types and operations
#include <FixMath.h>

// Routines constants
#define kAsynch TRUE

// Prototye routine like this prior to calling it
void RecordSnd(Handle);

void RecordSnd (Handle mySndH)
{
    SPB mySPB;        // a sound input parameter block
    OSErr myErr;
    long myInRefNum;                    // device reference number
    long myBuffSize;  // size of buffer to record into
    short myHeadrLen;                   // length of sound header
    short myNumChans;                   // number of channels
    short mySampSize;                   // size of a sample
    Fixed mySampRate;                   // sample rate
    OSType myCompType;                  // compression type

    // User defined routine to get the current settings of sound input device
    void GetDeviceSettings(long, short *, Fixed *, short *, OSType *);

    // open the default input device for reading and writing
    myErr = SPBOpenDevice("\p", siWritePermission, &myInRefNum);

    if ( !myErr ) {
        // get current settings of sound input device
        // using an application-defined routine
        GetDeviceSettings(myInRefNum, &myNumChans, &mySampRate,
            &mySampSize,&myCompType);

        // set up handle to contain the proper 'snd ' resource header
        myErr = SetupSndHeader(mySndH, myNumChans, mySampRate,
```

```
            mySampSize, myCompType, 60, 0, &myHeadrLen);

        // leave room in buffer for the sound header
        myBuffSize = GetHandleSize(mySndH) - myHeadrLen;

        // lock down the sound handle until the recording is over
        HLock(mySndH);

        // set up the sound input parameter block
        mySPB.inRefNum = myInRefNum;     // input device reference number
        mySPB.count = myBuffSize;        // number of bytes to record
        mySPB.milliseconds = 0;          // no milliseconds
        mySPB.bufferLength = myBuffSize; // length of buffer
        mySPB.bufferPtr =  (Ptr)(*mySndH + myHeadrLen);  // put data after
                                                         //'snd' header
        mySPB.completionRoutine = nil;   // no completion routine
        mySPB.interruptRoutine = nil;    // no interrupt routine
        mySPB.userLong = 0;              // no user data
        mySPB.error = noErr;             // clear error field
        mySPB.unused1 = 0;               // clear reserved field

        // record synchronously through the open sound input device
        myErr = SPBRecord(&mySPB, !kAsynch);

        // recording is done, so unlock the sound handle
        HUnlock(mySndH);

        // now fill in the number of bytes actually recorded
        myErr = SetupSndHeader(mySndH, myNumChans, mySampRate,
            mySampSize, myCompType, 60, mySPB.count, &myHeadrLen);

        // close the input device
        myErr = SPBCloseDevice(myInRefNum);
    }
}
```

The RecordSnd procedure defined in the Listing above opens the default sound input device by using the **SPBOpenDevice** function. You can specify one of two values for the permissions parameter of **SPBOpenDevice**:

```
siReadPermission        //open device for reading
siWritePermission       //open device for reading/writing
```

If **SPBOpenDevice** successfully opens the specified device for reading and writing, RecordSnd calls the GetDeviceSettings procedure. That procedure calls the **Sound Manager** function **SPBGetDeviceInfo** (explained in the separate section entitled **Getting and Setting Sound Input Device Information**) to determine the current number of channels, sample rate, sample size, and compression type in use by the device.

This information is then passed to the **SetupSndHeader** function, which loads the initial segment of the handle with a sound header describing the current device settings. After doing this, RecordSnd sets up a sound input parameter block, **SPB** and calls **SPBRecord** to record a sound.

Note that the handle must be locked during the recording because the parameter block contains a pointer to the input buffer. After the recording is done, RecordSnd calls **SetupSndHeader** once again to fill in the actual number of bytes recorded.

Once the procedure defined in the Listing above executes successfully, the handle mySndH points to a resource of type 'snd '. Your application can then play the recorded sound, for example, by executing the following lines of code:

```
RecordSnd(mySndH); //procedure shown in the Listing above
myErr = SndPlay(NULL, mySndH, FALSE);
```

Defining a Sound Input Completion Routine

The completionRoutine field of the **SPB** structure contains the address of a completion routine that executes when the recording terminates normally either by reaching its prescribed time or size limits, or by the application calling **SPBStopRecording**. A completion routine should have the following format:

```
pascal void MyRecordCompletionRoutine ( SPBPtr inParamPtr);
```

The completion routine is passed the address of the sound input parameter block, **SPB**, that was passed to **SPBRecord**. You can gain access to other data structures in your application by passing an address in the *userLong* field of the parameter block. After the completion routine executes, your application should check the error field of the sound input parameter block, , **SPB**, to see if an error code was returned.

Defining an Interrupt Routine

The *interruptRoutine* field of the sound input parameter block , **SPB**, contains the address of a routine that executes when the internal buffers of an asynchronous recording device are filled. The internal buffers contain raw samples taken directly from the input device. The interrupt routine can modify the samples in the buffer in any way it requires. The processed samples are then written to the application buffer. If compression is enabled, then the modified data is compressed after your interrupt routine operates on the samples and before the samples are written to the application buffer. You can determine the size of the sample buffer by calling **SPBGetDeviceInfo** with the 'dbin' selector.

> **Assembly-language note:** Interrupt routines are typically written in assembly language to maximize real-time performance in recording sound. On entry, registers are set up as follows:
>
> A0: Address of the sound parameter block passed to **SPBRecord**
> A1: Address of the start of the sample buffer
> D0: Peak amplitude for sample buffer if metering is on
> D1: Size of the sample buffer in bytes

Your interrupt routine is always called at interrupt time, so it should not call routines that might move or compact memory.