Playing 'snd ' Resources Beginning routine for playing sounds

Perhaps the simplest **Sound Manager** routine is **SndPlay**, which requires nothing more than a handle to an existing 'snd ' resource. An 'snd ' resource contains the information necessary for the **Sound Manager** to create a channel linked to the required synthesizer, together with the sound commands that are to be sent to that synthesizer to play the desired sound. The 'snd ' resource may or may not contain sound data. If it does, as in the case of a sampled sound, that data can be either compressed or uncompressed. As long as the resource is created correctly, **SndPlay** will play it. The following code example illustrates how to play back an 'snd ' resource.

Playing an 'snd ' resource with SndPlay

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

// constants
#define kAsync TRUE        // play is asynchronous
#define mySndID 9000       // resource ID of 'snd'

// prototype your routine like this prior to calling it
void CallSndPlay(void);

void  CallSndPlay()
{
    Handle mySndHandle;                 // handle to an 'snd ' resource
    SndChannelPtr mySndChan;            // pointer to a sound channel
    OSErr myErr;

    // prototype for user defined error handling routine
    void DoError(OSErr);

    mySndChan = nil;     // Initialize channel pointer for error checking

    // Read in 'snd' resource from resource file
    mySndHandle = GetResource ('snd ', mySndID);

    // check for a nil handle
    if ( mySndHandle != nil ) {
        myErr = SndPlay (mySndChan, mySndHandle, kAsync);
        if ( myErr )
        DoError(myErr);
    }
}
```

When your application uses **SndPlay** and passes NULL as the pointer to a sound channel, it does not need to concern itself with any memory allocation or deallocation. The **Sound Manager** automatically opens a sound channel (in the application's heap) and then closes it when the sound is completed.

For more complete control of the sound channel, your application can open a sound channel by using **SndNewChannel**. The application then sends

commands to that channel with **SndDoCommand** or **SndDoImmediate**. When the sound is completed, your application closes the channel with **SndDisposeChannel**. The code example below shows how your application can use low-level **Sound Manager** routines in the simple case of calling the sampled sound synthesizer to play back a buffer of sampled sound. The buffer can be either compressed or uncompressed. This example assumes that the sampled sound is uncompressed and that the sound header whose address is passed as an argument has already been filled out correctly.

Using low-level Sound Manager routines

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

// prototype your buffer routine like this prior to calling it
void DoBufferCmd (SoundHeaderPtr);

void  DoBufferCmd(SoundHeaderPtr mySHPtr)
{
    SndChannelPtr mySndChan;    // pointer to a sound channel
    SndCommand mySndCmd;  // a sound command
    OSErr myErr;

    // prototype your DoError routine like this
    void DoError (OSErr);

    mySndChan = nil; // Sound Mgr allocates channel
    mySndCmd.cmd = bufferCmd; // the command is bufferCmd
    mySndCmd.param1 = 0;   // not used here
    mySndCmd.param2 = (long)mySHPtr;          // pointer to SoundHeader

    // allocate a sound channel
    myErr = SndNewChannel(&mySndChan, sampledSynth, initMono, NULL);
    if ( myErr )
        DoError(myErr);

    // queue bufferCmd in the sound channel
    myErr = SndDoCommand(mySndChan, &mySndCmd, FALSE);
    if ( myErr )
        DoError(myErr);

    // dispose of the channel when finished
    myErr = SndDisposeChannel(mySndChan, FALSE);
    if ( myErr )
        DoError(myErr);
}
```

The **Sound Manager** functions used in the Listing above (**SndNewChannel**, **SndDoCommand**, and **SndDisposeChannel**) are explained in detail in the following sections. Note that the procedure does not do proper error handling; it is intended primarily to illustrate the proper order in which you should call the low-level **Sound Manager** routines.