Using Double Buffers Bypassing normal routines

The play-from-disk routines make extensive use of the **SndPlayDoubleBuffer** function. You can use this function in your application if you wish to bypass the normal play-from-disk routines. You might want to do this if you wish to maximize the efficiency of your application while maintaining compatibility with the **Sound Manager**. By using **SndPlayDoubleBuffer** instead of the normal play-from-disk routines, you can specify your own doubleback procedure (that is, the algorithm used to switch back and forth between buffers) and customize several other buffering parameters.

> **Note:** **SndPlayDoubleBuffer** is a very low-level routine and is not intended for general use. You should use **SndPlayDoubleBuffer** only if you require very fine control over double buffering.

You call **SndPlayDoubleBuffer** by passing it a pointer to a sound channel (into which the double-buffered data is to be written) and a pointer to a sound double-buffer header(**SndDoubleBufferHeader**). Here's an example:

```
myErr = SndPlayDoubleBuffer (mySndChan, &myDoubleHeader);
```

The dbhBufferPtr array contains pointers to two records of type **SndDoubleBuffer**. These are the two buffers between which the **Sound Manager** switches until all the sound data has been sent into the sound channel. When the call to **SndPlayDoubleBuffer** is made, the two buffers should both already contain a nonzero number of frames of data.

The following two sections illustrate how to fill out these data structures, create your two buffers, and define a doubleback procedure to refill the buffers when they become empty.

Setting Up Double Buffers

Before you can call **SndPlayDoubleBuffer**, you need to allocate two buffers (of type **SndDoubleBuffer**), fill them both with data, set the flags for the two buffers to dbBufferReady, and then fill out a record of type **SndDoubleBufferHeader** with the appropriate information. The code below illustrates how you might accomplish these tasks.

```
// Listing: Setting up double buffers


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

// Constants used in routine
#define kDoubleBufferSize 4096      // size of each buffer (in bytes)

// variables used by doubleback proc
typedef struct LocalVars {
    long bytesTotal;        // total number of samples
    long bytesCopied;  // number of samples copied to buffer
    Ptr dataPtr;            // pointer to sample of copy
} LocalVars, *LocalVarsPtr;
```

```
// prototype routine like this prior to calling it
// this function uses SndPlayDoubleBuffer to play the sound specified
OSErr DBSndPlay(SndChannelPtr, SoundHeaderPtr);

OSErr DBSndPlay (SndChannelPtr chan, SoundHeaderPtr sndHeader)
{
    LocalVars myVars;
    SndDoubleBufferHeader doubleHeader;
    SndDoubleBufferPtr doubleBuffer;
    SCStatus status;
    short i;
    OSErr err;

    // Prototype for DoubleBackProc
    pascal void MyDoubleBackProc(SndChannelPtr, SndDoubleBufferPtr);
    // Error handling routine
    void DoError(OSErr);

    // set up myVars with initial information
    myVars.bytesTotal = sndHeader->length;
    myVars.bytesCopied = 0;       // no samples copied yet
    myVars.dataPtr = &sndHeader->sampleArea[0];
                            // pointer to first sample

    // set up SndDoubleBufferHeader
    doubleHeader.dbhNumChannels = 1;        // one channel
    doubleHeader.dbhSampleSize = 8;         // 8-bit samples
    doubleHeader.dbhCompressionID = 0;      // no compression
    doubleHeader.dbhPacketSize = 0;       // no compression
    doubleHeader.dbhSampleRate = sndHeader->sampleRate;
    doubleHeader.dbhDoubleBack = &MyDoubleBackProc;

    // initialize both buffers
    for (i = 0; i <= 1; i++) {
        // get memory for double buffer
        doubleBuffer = (SndDoubleBufferPtr) NewPtr (sizeof
                    (SndDoubleBuffer) + kDoubleBufferSize);
        if ( !doubleBuffer ) {
        DoError (MemError());
        return MemError();
    }

        doubleBuffer->dbNumFrames = 0;        // no frames yet
        doubleBuffer->dbFlags = 0;            // buffer is empty
        doubleBuffer->dbUserInfo[0] = (long)&myVars;

        // fill buffer with samples
        MyDoubleBackProc(chan, doubleBuffer);

        // store buffer pointer in header
        doubleHeader.dbhBufferPtr[i] = doubleBuffer;
    }

    // start the sound playing
    err = SndPlayDoubleBuffer(chan, &doubleHeader);
```

```
    if ( err ) {
        DoError(err);
    return err;
    }

    // wait for the sound to complete by watching the channel status
     do {
        err = SndChannelStatus(chan, sizeof(status),&status);
    } while (!status.scChannelBusy);

    // dispose double-buffer memory}
    for ( i = 0;   i <= 1; i++)
        DisposPtr(doubleHeader.dbhBufferPtr[i]);

    return noErr;
}
```

The function DBSndPlay takes two parameters, a pointer to a sound channel and a pointer to a sound header. It reads the sound header to determine the characteristics of the sound to be played (for example, how many samples are to be sent into the sound channel). Then DBSndPlay fills in the fields of the double-buffer header, creates two buffers, and starts the sound playing. The doubleback procedure MyDoubleBackProc is defined in the next section.

Writing a Doubleback Procedure

The *dbhDoubleBack* field of a double-buffer header specifies the address of a doubleback procedure, an application-defined procedure that is called when the double buffers are switched and the exhausted buffer needs to be refilled. The doubleback procedure should have this format:

pascal void MyDoubleBackProc (SndChannelPtr chan, SndDoubleBufferPtr exhaustedBuffer);

The primary responsibility of the doubleback procedure is to refill an exhausted buffer of samples and to mark the newly filled buffer as ready for processing. Thecode below illustrates how to define a doubleback procedure. Note that the sound-channel pointer passed to the doubleback procedure is not used in this procedure.

This doubleback procedure extracts the address of its local variables from the *dbUserInfo* field of the double-buffer record passed to it. These variables are used to keep track of how many total bytes need to be copied and how many bytes have been copied so far. Then the procedure copies at most a buffer-full of bytes into the empty buffer and updates several fields in the double-buffer record and in the structure containing the local variables. Finally, if all the bytes to be copied have been copied, the buffer is marked as the last buffer.

> **Note:** Because the doubleback procedure is called at interrupt time, it cannot make any calls that move memory either directly or indirectly. (Despite its name, the **BlockMove** procedure does not cause blocks of memory to move or be purged, so you can safely call it in your doubleback procedure, as illustrated in the Listing below.)

Defining a doubleback procedure

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

// Constants used in routine
#define kDoubleBufferSize 4096     // size of each buffer (in bytes)

// variables used by doubleback proc
typedef struct LocalVars {
    long bytesTotal;        // total number of samples
    long bytesCopied;       // number of samples copied to buffer
    Ptr dataPtr;            // pointer to sample of copy
} LocalVars, *LocalVarsPtr;

// Prototype the routine like this prior to calling it
// Note this routine must take pascal style calling conventions
// since it will be called from the toolbox
pascal void MyDoubleBackProc(SndChannelPtr, SndDoubleBufferPtr);

pascal void MyDoubleBackProc (SndChannelPtr chan,SndDoubleBufferPtr
                                                    doubleBuffer)
{
    LocalVarsPtr myVarsPtr;
    long bytesToCopy;

    // get pointer to my local variables
    myVarsPtr = (LocalVarsPtr)doubleBuffer->dbUserInfo[0];

    // get number of bytes left to copy
    bytesToCopy = myVarsPtr->bytesTotal - myVarsPtr->bytesCopied;

    // If the amount left is greater than double-buffer size,
    // then limit the number of bytes to copy to the size of the buffer.
    if ( bytesToCopy > kDoubleBufferSize )
        bytesToCopy = kDoubleBufferSize;

    // copy samples to double buffer
    BlockMove(myVarsPtr->dataPtr, &doubleBuffer->dbSoundData[0],
            bytesToCopy);

    // store number of samples in buffer and mark buffer as ready
    doubleBuffer->dbNumFrames = bytesToCopy;
    doubleBuffer->dbFlags = doubleBuffer->dbFlags | dbBufferReady;

    // update data pointer and number of bytes copied
    myVarsPtr->dataPtr = myVarsPtr->dataPtr + bytesToCopy;
    myVarsPtr->bytesCopied = myVarsPtr->bytesCopied + bytesToCopy;

    // If all samples have been copied, then this is the last buffer.
    if ( myVarsPtr->bytesCopied == myVarsPtr->bytesTotal )
        doubleBuffer->dbFlags = doubleBuffer->dbFlags | dbLastBuffer;
}
```