The Format 2 'snd ' Resource

The SndPlay function can also play format 2 'snd ' resources, which are designed for use only with the sampled sound synthesizer. The SndPlay function supports this format by automatically opening a channel to the sampled sound synthesizer and using the bufferCmd command to send the data contained in the resource to that synthesizer.

Figure 22-7 illustrates the fields of a format 2 'snd ' resource. The reference count field is for your application's use and is not used by the Sound Manager. The number of sound commands field and the sound command fields are the same as described in a format 1 resource. The last field of this resource contains the sampled sound. The first command should be either a soundCmd command or bufferCmd command with the data offset bit set in the command to specify the location of this sampled sound header.

Listing 22-6 shows a resource specification that illustrates the structure of a format 2 'snd ' resource; it contains the information necessary to create a sound with SndPlay and the sampled sound synthesizer.

Listing 22-6. A format 2 'snd ' resource

```
data 'snd ' (9003, "Pig Squeal", purgeable) {
   $"0002"          /*format type*/
   $"0000"          /*reference count for application's use*/
   $"0001"          /*number of sound commands that follow (1)*/
   $"8051"          /*command 1--bufferCmd*/
   $"0000"          /*param1 = 0*/
   $"0000000E"      /*param2 = offset to sound header (14 bytes)*/
   $"00000000"      /*pointer to data (it follows immediately)*/
   $"00000BB8"      /*number of bytes in sample (3000 bytes)*/
   $"56EE8BA3"      /*sampling rate of this sound (22 kHz)*/
   $"000007D0"      /*starting of the sample's loop point*/
   $"00000898"      /*ending of the sample's loop point*/
   $"00"            /*standard sample encoding*/
   $"3C"            /*baseFrequency at which sample was taken*/
   $"80 80 81 82 84 87 93 84" /*the sampled sound data*/
   $"6F 68 6D 65 72 7B 82 88"
   $"91 8E 8D 8F 86 7E 7C 79"
   $"6F 6D 71 70 70 79 7F 81"
   $"89 8F 8D 8B" /*rest of data omitted in this example*/
};
```

For a complete explanation of the fields following the sampling rate field, see the description of the sampled sound header in "Playing Sampled Sounds." To play the sounds described by these resources, see the instructions given in "Playing 'snd ' Resources." Both sections occur later in this chapter.

Sound Files

Although most sampled sounds that you want your application to produce can be stored as resources of type 'snd ', there are times when it is preferable to store sounds in **sound files,** not in resources. For example, it is usually easier for different applications to share

files than it is to share resources. So if you want your application to play sampled sounds created by other applications (or if you want other applications to be able to play sampled sounds created by your application), it might be better to store the sampled sound data in a file, not in a resource. Similarly, if you are developing versions of your application that are intended to run on other operating systems, you might need a method of storing sounds that is independent of the Macintosh Operating System and its reliance on resources to store data. Generally, it is easier to transfer data stored in files from one operating system to another than it is to transfer data stored in resources.

There are other reasons you might want to store some sampled sounds in files and not in resources. If you have a very large sampled sound, it may be impossible to create a resource large enough to hold all the audio data. Resources are limited in size by the structure of resource files (and in particular because offsets to resource data are stored as 24-bit quantities). Sound files, however, can be much larger because the only size limitations are those imposed by the file system on all files. If the sampled data for some sound occupies more than about a half megabyte of space, you should probably store the sound as a file.

To address these various needs, Apple and several third-party developers have defined two sampled sound file formats, known as the **Audio Interchange File Format (AIFF)** and the Audio Interchange File Format extension for Compression (AIFF-C). The names emphasize that the formats are designed primarily as data interchange formats. However, you should find both AIFF and AIFF-C flexible enough to use as data storage formats as well. Even if you choose to use a different storage format, your application should be able to convert to and from AIFF and AIFF-C if you want to facilitate sharing of sound data among applications.

The main difference between the AIFF and AIFF-C formats is that AIFF-C allows you to store both compressed and noncompressed audio data, whereas AIFF allows you to store noncompressed audio data only. The AIFF-C format is more general than the AIFF format and is easier to modify. The AIFF-C format can be extended to handle new compression types and application-specific data. As a result, you should revise any application that currently supports only AIFF files to also support AIFF-C files. An application that currently reads AIFF files should also be able to read AIFF-C files. An application that currently writes AIFF files should also be able to write AIFF-C files. It is recommended that the default write format be AIFF-C. Table 22-2 summarizes the capabilities of the AIFF and AIFF-C file formats.

Table 22-2. AIFF and AIFF-C capabilities

File type	Read sampled	Read compressed	Write sampled	Write compressed
AIFF	Yes	No	Yes	No
AIFF-C	Yes	Yes	Yes	Yes

The enhanced Sound Manager includes support for reading and writing both AIFF and AIFF-C files. You can play from disk a sampled sound stored in a file of type AIFF or type AIFF-C by opening that file and passing its file reference number to the SndStartFilePlay function. (If the file is of type AIFF-C and if the data is compressed, the data is automatically expanded during playback.) You can create files of type AIFF or AIFF-C by calling the SndRecordToFile and SPBRecordToFile functions. SndRecordToFile creates an AIFF or AIFF-C file, complete with compressed sound data and all the needed chunks. SPBRecordToFile, however, simply records audio data (compressing it if necessary) and saves that data into a specified file. SPBRecordToFile does not create any AIFF or AIFF-C chunks. You can, however, use the SetupAIFFHeader function to create the appropriate headers before you call SPBRecordToFile.

Note: Both SndRecordToFile and SPBRecordToFile automatically compress the recorded audio data if instructed to do so. Neither function does any expansion.

The following six sections describe in detail the structure of AIFF and AIFF-C files. Both of these types of sound files are collections of "chunks" that define characteristics of the sampled sound or other relevant data about the sound. Currently, the AIFF and AIFF-C specifications include the following chunk types.

Chunk types

Form Chunk	Contains all the other chunks of an AIFF or AIFF-C file
Format Version Chunk	Contains an indication of the version of the AIFF-C specification according to which this file is structured (AIFF-C only)
Common Chunk	Contains information about the sampled sound, such as the sampling rate and sample size
Sound Data Chunk	Contains the sample frames that comprise the sampled sound
Marker Chunk	Contains markers that point to positions in the sound data
Comments Chunk	Contains comments about markers in the file
Sound Accelerator Chunk	Contains information intended to allow applications to accelerate the decompression of compressed audio data
Instrument Chunk	Defines basic parameters that an instrument (such as a sampling keyboard) can use to play back the sound data
MIDI Data Chunk	Contains MIDI data
Audio Recording Chunk	Contains information pertaining to audio recording devices
Application Specific Chunk	Contains application-specific information
Name Chunk	Contains the name of the sampled sound
Author Chunk	Contains one or more names of the authors (or creators) of the sampled sound
Copyright Chunk	Contains a copyright notice for the sampled sound
Annotation Chunk	Contains a comment

The following sections document only four of the kinds of chunks that can occur in AIFF and AIFF-C files. A more complete specification of AIFF files is available from APDA.

Chunk Organization and Data Types

An AIFF or AIFF-C file is a file that is organized as a collection of "chunks" of data. For example, there is a Common Chunk that specifies important parameters of the sampled sound, such as its size and sample rate. There is also a Sound Data Chunk that contains the

actual audio samples. A chunk consists of some header information followed by some data. The header information consists of a chunk ID number and a number that indicates the size of the chunk data. In general, therefore, a chunk has the structure illustrated in Figure 22-9.

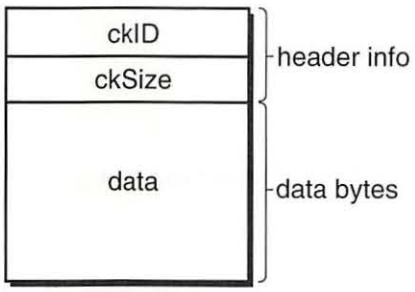

Figure 22-9. The general structure of a chunk

The header information of a chunk has this structure:

```
TYPE ChunkHeader =
     RECORD
        ckID:          ID;                {chunk type ID}
        ckSize:        LongInt            {number of bytes of data}
     END;
```

The ckID field specifies the chunk type. An ID is a 32-bit concatenation of any four printable ASCII characters in the range ' ' (space character, ASCII value $20) through '~' (ASCII value $7E). Spaces cannot precede printing characters, but trailing spaces are allowed. Control characters are not allowed. You can specify values for the four types of chunks described later by using these constants:

```
CONST FormID           = 'FORM';  {chunk ID for Form Chunk}
      FormatVersionID  = 'FVER';  {chunk ID for Format Version Chunk}
      CommonID         = 'COMM';  {chunk ID for Common Chunk}
      SoundDataID      = 'SSND';  {chunk ID for Sound Data Chunk}
```

The ckSize field specifies the size of the data portion of a chunk and does not include the length of the chunk header information.

The Form Chunk

The chunks that define the characteristics of a sampled sound and that contain the actual sound data are grouped together into a container chunk, known as the Form Chunk. The Form Chunk defines the type and size of the file and holds all remaining chunks in the file. The chunk ID for this container chunk is 'FORM'.

A chunk of type 'FORM' has this structure:

```
TYPE ContainerChunk =
     RECORD
        ckID:        ID;                {'FORM'}
        ckSize:      LongInt;           {number of bytes of data}
        formType:    ID                 {type of file}
     END;
```

The fields of this chunk have the following meanings:

Field descriptions

ckID — The ID of this chunk. For a Form Chunk, this ID is 'FORM'.

ckSize — The size of the data portion of this chunk. Note that the data portion of a Form Chunk is divided into two parts, formType and the chunks that follow the formType field. These chunks are called *local chunks* because their chunk IDs are local to the Form Chunk.

formType — The type of audio file. For AIFF files, formType is 'AIFF'. For AIFF-C files, formType is 'AIFC'.

The local chunks can occur in any order in a sound file. As a result, your application should be designed to get a local chunk, identify it, and then process it without making any assumptions about what kind of chunk it is based on its order in the Form Chunk.

The Format Version Chunk

One difference between the AIFF and AIFF-C file formats is that files of type AIFF-C contain a Format Version Chunk and files of type AIFF do not. The Format Version Chunk contains a timestamp field that indicates when the format version of this AIFF-C file was defined. This in turn indicates what format rules this file conforms to and allows you to ensure that your application can handle a particular AIFF-C file. Every AIFF-C file must contain one and only one Format Version Chunk.

In AIFF files, there is no Format Version Chunk.

In AIFF-C files, a Format Version Chunk has this structure:

```
TYPE FormatVersionChunk =
     RECORD
        ckID:        ID;                {'FVER'}
        ckSize:      LongInt;           {4}
        timestamp:   LongInt            {date of format version}
     END;
```

The fields of this chunk have the following meanings:

Field descriptions

ckID	The ID of this chunk. For a Format Version Chunk, this ID is 'FVER'.
ckSize	The size of the data portion of this chunk. This value is always 4 in a Format Version Chunk because the timestamp field is 4 bytes long (the 8 bytes used by ckID and ckSize fields are not included).
timestamp	An indication of when the format version for this kind of file was created. The value indicates the number of seconds since January 1, 1904, following the normal time conventions used by the Macintosh Operating System. (See the Operating System Utilities chapter of Volume II for several routines that allow you to manipulate timestamps.)

You should not confuse the format version timestamp with the creation date of the file. The format version timestamp indicates the time of creation of the version of the format according to which this file is structured. Because Apple defines the formats of AIFF-C files, only Apple can change this value. The current version is defined by a constant:

```
CONST AIFCVersion1   = $A2805140;          {2726318400 in decimal}
```

The Common Chunk

Every AIFF and AIFF-C file must contain a Common Chunk that defines some fundamental characteristics of the sampled sound contained in the file. Note that the format of the Common Chunk is different for AIFF and AIFF-C files. As a result, you need to determine the type of file format (by inspecting the formType field of the Form Chunk) before reading the Common Chunk.

For AIFF files, the Common Chunk has this structure:

```
TYPE CommonChunk =
     RECORD
        ckID:             ID;          {'COMM'}
        ckSize:           LongInt;     {size of chunk data}
        numChannels:      Integer;     {number of channels}
        numSampleFrames:  LongInt;     {number of sample frames}
        sampleSize:       Integer;     {number of bits per sample}
        sampleRate:       Extended     {number of frames per second}
     END;
```

For AIFF-C files, the Common Chunk has this structure:

```
TYPE ExtCommonChunk =
     RECORD
        ckID:             ID;          {'COMM'}
        ckSize:           LongInt;     {size of chunk data}
```

```
        numChannels:     Integer;      {number of channels}
        numSampleFrames: LongInt;      {number of sample frames}
        sampleSize:      Integer;      {number of bits per sample}
        sampleRate:      Extended;     {number of frames per second}
        compressionType: ID;           {compression type ID}
        compressionName: PACKED ARRAY[0..0] OF Byte
                                       {compression type name}
    END;
```

The fields that exist in both types of Common Chunk have the following meanings:

Field descriptions

ckID — The ID of this chunk. For a Common Chunk, this ID is 'COMM'.

ckSize — The size of the data portion of this chunk. In AIFF files, this field is always 18 in the Common Chunk because the 8 bytes used by the ckID and ckSize fields are not included. In AIFF-C files, this size is 22 plus the number of bytes in the compressionName string.

numChannels — The number of audio channels contained in the sampled sound. A value of 1 indicates monophonic sound; a value of 2 indicates stereo sound; a value of 4 indicates four-channel sound, and so forth. Any number of audio channels may be specified. The actual sound data is stored elsewhere, in the Sound Data Chunk.

numSampleFrames — The number of sample frames in the Sound Data Chunk. Note that this field contains the number of sample frames, not the number of bytes of data and not the number of sample points. For noncompressed sound data, the total number of sample points in the file is numChannels * numSampleFrames. (See the discussion of the Sound Data Chunk in the following section for a definition of a sample frame.)

sampleSize — The number of bits in each sample point of noncompressed sound data. The sampleSize field can contain any integer from 1 to 32. For compressed sound data, this field indicates the number of bits per sample in the original sound data, before compression.

sampleRate — The sample rate at which the sound is to be played back, in sample frames per second.

An AIFF-C Common Chunk includes two fields that describe the type of compression (if any) used on the audio data:

Field descriptions

compressionType — The ID of the compression algorithm, if any, used on the sound data.

compressionName — A human-readable name for the compression algorithm ID specified in the compressionType field. This string is useful when putting up alert boxes (perhaps because a necessary decompression routine is missing).

Remember to pad the end of this array with a byte having the value 0 if the length of this array is not an even number (but do not include the pad byte in the count).

Here are the currently available compression IDs and their associated compression names:

compressionType	compressionName	Description
'NONE'	'not compressed'	Noncompressed samples
'ACE2'	'ACE 2-to-1'	IIGS® 2-to-1 compressed
'ACE8'	'ACE 8-to-3'	IIGS 8-to-3 compressed
'MAC3'	'MACE 3-to-1'	Macintosh 3-to-1 compressed
'MAC6'	'MACE 6-to-1'	Macintosh 6-to-1 compressed

You can define your own compression types, but you should register them with Apple.

The Sound Data Chunk

The Sound Data Chunk contains the actual sample frames that make up the sampled sound. The Sound Data Chunk has this structure:

```
TYPE SoundDataChunk =
     RECORD
        ckID:          ID;              {'SSND'}
        ckSize:        LongInt;         {size of chunk data}
        offset:        LongInt;         {offset to sound data}
        blockSize:     LongInt          {size of alignment blocks}
     END;
```

The fields in a Sound Data Chunk have the following meanings:

Field descriptions

ckID — The ID of this chunk. For a Sound Data Chunk, this ID is 'SSND'.

ckSize — The size of the data portion of this chunk. This size does not include the 8 bytes occupied by the values in the ckID and the ckSize fields. If the data following the blockSize field contains an odd number of bytes, a pad byte with a value of 0 is added at the end to preserve an even length for this chunk. If there is a pad byte, it is not included in the ckSize field.

offset — An offset (in bytes) to the beginning of the first sample frame in the chunk data. Most applications do not need to use the offset field and should set it to 0.

blockSize — The size (in bytes) of the blocks to which the sound data is aligned. This field is used in conjunction with the offset field for aligning sound data to blocks. As with the offset field, most applications do not need to use the blockSize field and should set it to 0.

The format of the sound data following the blockSize field depends on whether the data is compressed or noncompressed, which you can determine by inspecting the compressionType field in the Common Chunk. If the compression type is 'NONE', then each sample point in a sample frame is a linear, two's complement value. Sample points are from 1 to 32 bits wide, as determined by the sampleSize parameter in the Common Chunk. Each sample point is stored in an integral number of contiguous bytes. Sample points that are from 1 to 8 bits wide are stored in 1 byte; sample points that are from 9 to 16 bits wide are stored in 2 bytes, and so forth. When the width of a sample point is less than a multiple of 8 bits, the sample point data is left aligned (using a shift-left instruction), and the low-order bits at the right end are set to 0.

For multichannel sounds, a sample frame is an interleaved set of sample points. (For monophonic sounds, a sample frame is just a single sample point.) The sample points within a sample frame are interleaved by channel number. For example, the sound data for a stereo, noncompressed sound is illustrated in Figure 22-10.

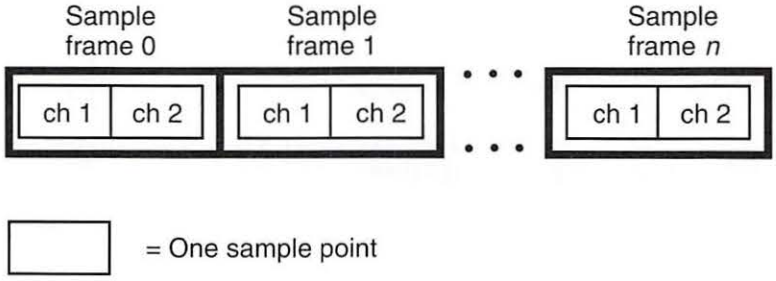

Figure 22-10. Interleaving stereo sample points

Sample frames are stored contiguously in order of increasing time. There are no pad bytes between samples or between sample frames.

Note: The Sound Data Chunk is required unless the numSampleFrames field in the Common Chunk is 0. A maximum of one Sound Data Chunk can appear in an AIFF or AIFF-C file.

Reading and Writing Sound Files

Figure 22-11 illustrates an AIFF-C format file that contains approximately 4.476 seconds of 8-bit monophonic sound data sampled at 22 kHz. The sound data is not compressed. Note that the number of sample frames in this example is odd, forcing a pad byte to be inserted after the sound data. This pad byte is not reflected in the ckSize field of the Sound Data Chunk, which means that special processing is required to correctly determine the actual chunk size.

On a Macintosh computer, the Form Chunk (and hence all the other chunks in an AIFF or AIFF-C file) is stored in the data fork of the file. The file type of an AIFF format file is 'AIFF' and the file type of an AIFF-C format file is 'AIFC'. Macintosh applications should not store any information in the resource fork of an AIFF or AIFF-C file because that information might not be preserved by other applications that edit sound files.

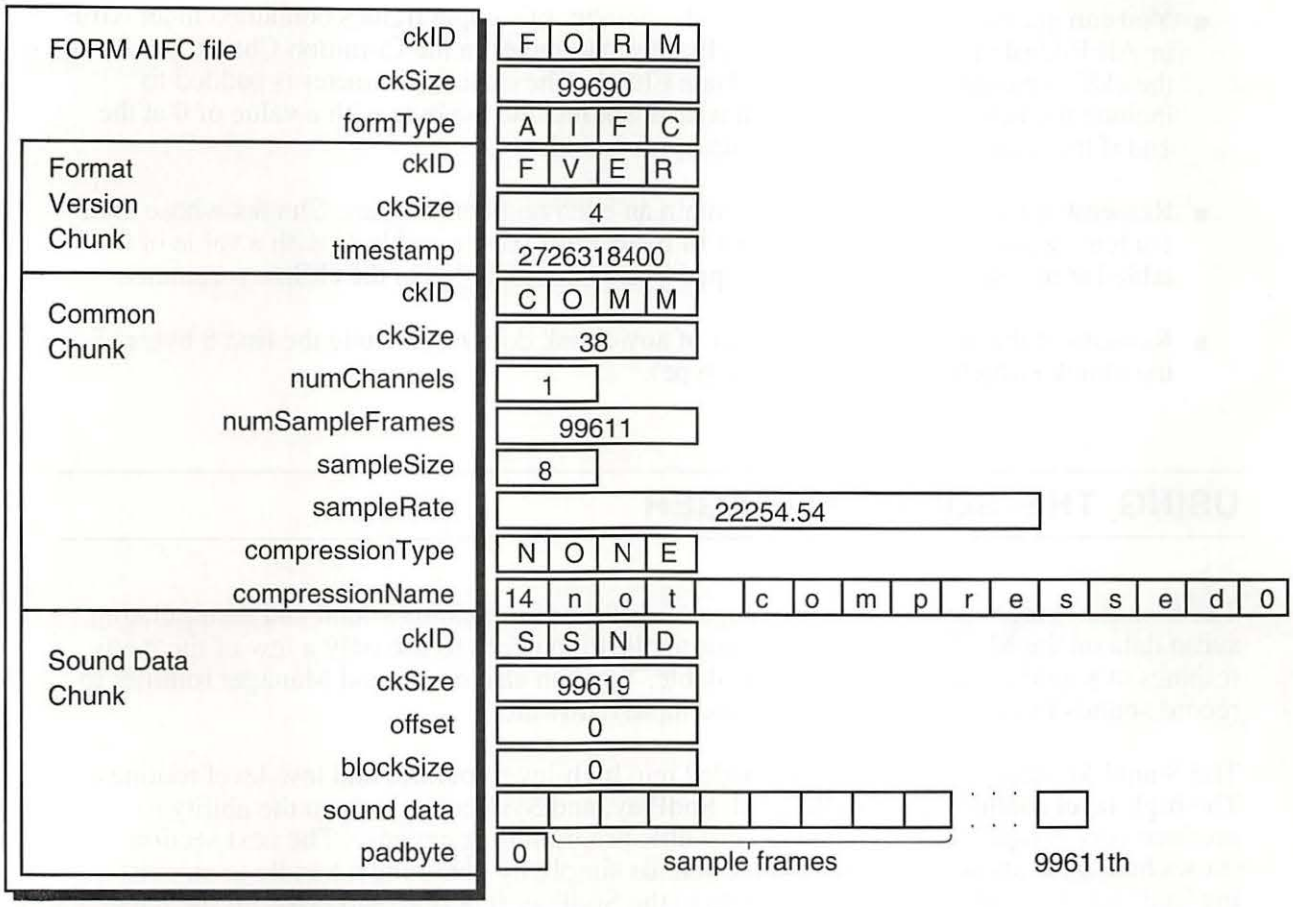

Figure 22-11. A sample AIFF-C file

Every Form Chunk must contain a Common Chunk and every AIFF-C file must contain a Format Version Chunk. In addition, if the sampled sound has a length greater than 0, there must be a Sound Data Chunk in the Form Chunk. All other chunk types are optional. Your application should be able to read all the required chunks if it uses AIFF or AIFF-C files, but it can choose to ignore any of the optional chunks.

When reading or writing AIFF or AIFF-C files, you should keep the following points in mind:

- Remember that the local chunks in an AIFF or AIFF-C file can occur in any order. An application that reads these types of files should be designed to get a chunk, identify it, and then process it without making any assumptions about what kind of chunk it is based on its order in the Form Chunk.
- If your application allows modification of a chunk, then it must also update other chunks that may be based on the modified chunk. However, if there are chunks in the file that your application does not recognize, you must discard those unrecognized chunks. Of course, if your application is simply copying the AIFF or AIFF-C file without any modification, you should copy the unrecognized chunks, too.

- You can get the clearest indication of the number of sample frames contained in an AIFF or AIFF-C file from the numSampleFrames parameter in the Common Chunk, not from the ckSize parameter in the Sound Data Chunk. The ckSize parameter is padded to include the fields that follow it, but it does not include the byte with a value of 0 at the end if the total number of sound data bytes is odd.
- Remember that each chunk must contain an even number of bytes. Chunks whose total contents would yield an odd number of bytes must have a pad byte with a value of 0 added at the end of the chunk. This pad byte is not included in the ckSize parameter.
- Remember that the ckSize parameter of any chunk does not include the first 8 bytes of the chunk (which specify the chunk type).

USING THE SOUND MANAGER

The Sound Manager provides a wide range of methods for creating sound and manipulating audio data on the Macintosh. Usually, your application needs to use only a few of the many routines or sound commands that are available. You can also use Sound Manager routines to record sounds through any available sound input hardware.

The Sound Manager routines can be divided into high-level routines and low-level routines. The high-level routines (like SndRecord, SndPlay, and SysBeep) give you the ability to produce very complex audio output at very little programming expense. The next section shows how your application can produce sounds simply by obtaining a handle to an existing 'snd ' resource and passing that handle to the SndPlay function. Moreover, if the data in the 'snd ' resource is stored in a compressed format, SndPlay automatically expands it for play-back in real time without further intervention from your application.

Although the high-level Sound Manager routines are sufficient for many applications, low-level Sound Manager routines are available to provide your application with much greater control over sound recording and production than is provided by the high-level routines. Using these low-level routines, your application can record directly from sound input devices, allocate and release sound channels, queue sound commands to a channel or bypass a sound queue altogether, perform modifications on sound data and commands sent into a channel, create and mix multiple channels of sound, compress and expand audio data, disable and enable the system alert sound, obtain information about current sound activity, and play sounds continuously from disk.

Some of these operations are carried out by specialized low-level routines, but most of them are accomplished by passing appropriate sound commands to the SndDoCommand, SndDoImmediate, and SndControl functions. For example, your application can alter the pitch of a sampled sound that is currently playing by calling SndDoImmediate with the rateCmd command as one of its parameters.

Some of the Sound Manager routines and commands cannot be called at interrupt time because they attempt to allocate or release memory. In particular, the routines SndNewChannel, SndDisposeChannel, SndAddModifier, SysBeep, SndPlay, SndStartFilePlay, SndRecord, and SndRecordToFile cannot be called at interrupt time. In addition, callback procedures,