QuickTime 4 Reference

For Macintosh and Windows

Apple Computer, Inc.
Technical Publications
July, 1999

QuickTime Audio

This chapter discusses new features of QuickTime audio. For information about audio support and the uses of the Sound Manager, refer to Chapter 2 of *Inside Macintosh: Sound.*

This chapter focuses on issues that are relevant to QuickTime. Note that Sound Manager sources are now multi-platform, so that the same manager is available on all platforms where QuickTime is available, and contains the same API and features.

This chapter also discusses how QuickTime 3 handles compressed audio, in addition to the two recent extensions defined for the `SoundDescription` sample description record. The first extension is the addition of `slope, intercept, minClip`, and `maxClip` parameters for audio. The second is the ability to store data specific to a given audio decompressor in the `SoundDescription` record.

New Features of QuickTime Audio

QuickTime 3 automatically installs the latest version of the Sound Manager.

Multi-platform Support

The Sound Manager sources are now multi-platform, i.e., built for the Mac OS, Windows 95 and NT, and any other platform that supports QuickTime. This means the same Sound Manager software is available for each platform, containing the same API and features.

Dealing with Endian Issues

Multi-platform support raises the issue of “endianness.” Basically, endian conversion is treated as a compression conversion. Any non-native endian format is required to be “decompressed” into the native format.

How QuickTime 3 Handles Compressed Audio

QuickTime 3 defines version 1 of the `SoundDescription` sample description record. Note that for purposes of this discussion, a QuickTime sound sample description chunk describes the format of a collection of audio samples.

The existing description is shown in Listing 20-1 for reference.

Listing 20-1 The original SoundDescription sample description

```
struct SoundDescription {
    long        descSize;       /* total size of SoundDescription
                                   including extra data */
    long        dataFormat;     /* sound format */
    long        resvd1;         /* reserved for apple use. set to zero */
    short       resvd2;         /* reserved for apple use. set to zero */
    short       dataRefIndex;
    short       version;        /* which version is this data */
    short       revlevel;       /* what version of that codec did this */
    long        vendor;         /* whose  codec compressed this data */
    short       numChannels;    /* number of channels of sound */
    short       sampleSize;     /* number of bits per sample */
    short       compressionID;  /* unused. set to zero. */
    short       packetSize;     /* unused. set to zero. */
    UnsignedFixed sampleRate;   /* sample rate sound is captured at */
};
typedef struct SoundDescription SoundDescription;
```

Version 1 of this record includes four extra fields to store information about compression ratios. It also defines how other extensions are added to the `SoundDescription` record.

```
struct SoundDescriptionV1 {
    // original fields
    SoundDescription    desc;
    // fixed compression ratio information
    unsigned long   samplesPerPacket;
    unsigned long   bytesPerPacket;
    unsigned long   bytesPerFrame;
    unsigned long   bytesPerSample;
    // additional atom-based fields --
    // ([long size, long type, some data], repeat)
};
```

The version 1 sound description is a superset of the version 0 sound description. The new fields are taken directly from the `CompressionInfo` structure currently used by the Sound Manager to describe the compression ratio of fixed ratio audio compression algorithms. They are described in detail in *Inside Macintosh: Sound.* If these fields are not used, they are set to 0. File readers only need to check to see if `samplesPerPacket` is 0. The fields have been added to support compression algorithms which can be run at different compression ratios and to support more generic parsing of QuickTime sound tracks

IMPORTANT

It is necessary to know the compression ratio to rechunk or flatten the media. In the past, the only way to know the compression ratio was to directly query the audio decompressor. If this process was running on a computer without the decompressor (such as a server), it would not have enough information to correctly rechunk the audio.s

All other additions to the `SoundDescription` record are made using QT atoms. That means one or more atoms can be appended to the end of the SoundDescription record using the standard [size, type] mechanism used throughout the QuickTime movie resource architecture.

Extensions to the SoundDescription Record

Two extensions are defined to the `SoundDescription` record. The first is the `slope, intercept, minClip,` and `maxClip` parameters for audio as defined in Appendix D. This is represented as an atom of type `siSlopeAndIntercept.` The contents of the atom are:

```
struct SoundSlopeAndInterceptRecord {
    Float64                 slope;
    Float64                 intercept;
    Float64                 minClip;
    Float64                 maxClip;
};
typedef struct SoundSlopeAndInterceptRecord SoundSlopeAndInterceptRecord;
```

The second extension is the ability to store data specific to a given audio decompressor in the `SoundDescription` record. Some audio decompression algorithms, such as Microsoft's ADPCM, require a set of out-of-stream values to configure the decompressor. These are stored in a `siDecompressorSettings`. The contents of the `siDecompressorSettings` atom are dependent on the audio decompressor. If the QuickTime movie was created from a WAVE (`.WAV`) or AVI (`.avi`) file, the `siDecompressorSettings` atom is automatically created and set to the contents of the `WAVEFORMATEX` structure from that file. In this case, the `siDecompressorSettings` atom contains little-endian data.

At runtime, the contents of the type `siSlopeAndIntercept` and `siDecompressorSettings` atoms are provided to the decompressor component through the standard `SetInfo` mechanism of the Sound Manager. The `samplesPerPacket, bytesPerPacket, bytesPerFrame,` and `bytesPerSample` fields are also passed to the decompressor component via `SetInfo` in a `CompressionInfo` structure with the `siCompressionFactor` selector.

Constants for Additional Audio Compression Formats

QuickTime 3 defines constants for several additional audio compression formats. These include `kFloat32Format` and `kFloat64Format` for single and double precision (i.e., big endian IEEE) floating-point audio. It also defines `'alaw'` for aLaw audio. All DV audio from NTSC (format 60) DVC streams is `'dvca'`, regardless of the format of the data in the frame. In addition, a standard mapping for audio formats present in the Windows Audio Compression Manager is defined.

All ACM audio formats are defined with a 16-bit integer. These are mapped into a QuickTime four-character code by putting 'ms' in the first two characters and the ACM 16-bit value in the second two characters. So the Microsoft ADPCM algorithm (ACM value of 1) has a QuickTime audio compression code of (('MS' << 16) | 1). This enables standard mapping of ACM audio into the QuickTime movie format.

QuickTime 3 has built-in support to decompress the following additional audio formats:

- single-precision floating point
- double-precision floating point
- Microsoft ADPCM (ACM code 1)
- Intel/DVI IMA (ACM code 7), DV
- aLaw

Both floating-point decompressors support the type `siSlopeAndIntercept` atom to provide scaling and DC-offset support. QuickTime 3 has the ability to compress the following additional audio formats:

- single-precision floating point
- double-precision floating point
- aLaw

QuickTime 3 correctly imports compressed audio from AVI (`.avi`) and WAVE (`.WAV`) files. The AU file importer has also been enhanced to import aLaw, 8- and 16-bit uncompressed, and single- and double-precision floating point audio in addition to the uLaw that it handled previously. A Sound Designer II file importer has been added. An AU file exporter has been added which can generate aLaw, uLaw, single and double precision floating point, and uncompressed AU files. (Note that most other readers can only handle uLaw, however).

DV Audio Decompressor Component

The Sound Manager includes a DV audio decompressor component. This component, which works with Sound Manager version 3.3 or later, can decompress DV audio in any of the formats supported by DV sources. These formats are:

- 2 audio channels, 12-bit encoding, 32K samples per second
- 2 audio channels, 12-bit encoding, 44.1K samples per second
- 2 audio channels, 12-bit encoding, 48K samples per second
- 2 audio channels, 16-bit encoding, 32K samples per second
- 2 audio channels, 16-bit encoding, 44.1K samples per second

- 2 audio channels, 16-bit encoding, 48K samples per second
- 4 audio channels, 12-bit encoding, 32K samples per second

The file type for all DV audio formats is `kDVAudioFormat`.

Note that the DV audio decompressor component provides the ability to decode audio data stored in a DVC stream.

Using QuickTime Audio

Creating a 16-bit, 22K Uncompressed WAVE File Using QuickTime 3

The following is an example of how you can convert an 8-bit, 22K .wav file to a 16-bit, 22K IMAPCM .wav file using QuickTime 3.

While QuickTime can create WAVE files, it does not support creating IMA-compressed WAVE files. It can play back WAVE files that contain IMA-compressed audio.

To create a 16-bit 22k uncompressed WAVE file using QuickTime 3, you perform the following steps:

1. Open an exporter.

```
ComponentInstance ci;
ci = OpenDefaultComponent(MovieExportType, kQTFileTypeWave);
```

2. Create a sound description for the audio format you want. If there are values you don’t care about, you can leave them unspecified. In this example, the number of channels is not indicated. The exporter will base the channel count on the source movie.

```
SoundDescriptionHandle desc;

desc = (SoundDescriptionHandle )NewHandleClear(sizeof(SoundDescripion));
(**desc).descSize = sizeof(SoundDescription);
```

```
(**desc).sampleSize = 16;
(**desc).sampleRate = 22050L << 16;
(**desc).dataFormat = k16BitLittleEndianFormat;
```

3. Specify the export component in which format you want the audio.

```
MovieExportSetSampleDescription(ci, (SampleDescriptionHandle)desc,
SoundMediaType);
```

4. Perform the export operation.

```
ConvertMovieToFile(theMovie, nil, &outputFile, kQTFileTypeWave,
                    OSTypeConst('TVOD'), -1, nil, 0, ci);
```

5. After you have finished, dispose everything that you have created.

```
CloseComponent(ci);
DisposeHandle((Handle)desc);
```