3.3 Release Notes: Sound Library

There are no new sound library features or bug fixes in Release 3.3 or 3.2. This file contains release notes for both the 3.1 and 3.0 releases of the sound library. Items specific to the 3.1 release are listed first, and the Release 3.0 notes follow.

Notes Specific to Release 3.1

New Features

- Two new ªsound-portingº functions have been added to the sound library and declared in **<sound/utilsound.h>**:

  SNDSwapSoundToHost(void **dest*, void **src*, int *sampleCount*,

int *channelCount*, int *dataFormat***)**

SNDSwapHostToSound(void **dest*, void **src*, int *sampleCount*,
int *channelCount*, int *dataFormat***)**

These functions read the data pointed to by *src*, swap the order of the bytes that comprise each sample (if necessary) and write the swapped data to *dest* (*src* and *dest* can be the same pointer). The other arguments describe the sound data buffer; cumulatively, they tell a function how many samples are in the buffer, and the size of each sample. The functions return error codes that report their success or failure.

The ªSoundº and ªHostº terms in these names have these meanings:

- To play a sound, the sound data must be in ªSoundº format.
- To examine sound data, it must be in ªHostº format.

When you read a soundfile or record a sound, the sound data is guaranteed to be in ªSoundº format; if all you want to do is play the soundfile or write the recorded sound to a fileÐin other words, if you don't want to examine the dataÐthen you don't need to call these functions.

However, if, for example, you read a soundfile in order to look at the value at each sample, you must first call **SNDSwapSoundToHost()** to convert the sound data to ªHostº format. If you modify samples in a converted sound, you must re-convert to ªSoundº format, through **SNDSwapHostToSound()**,

before the sound can be played or written to a file.

Note that you don't have to convert entire soundfiles (or SNDSoundStruct data buffers) at a time.

These functions are built upon the general **NXSwap...** functions that are described in **/NextLibrary/Documentation/NextDev/Concepts/PortabilityGuide**. The primary difference between the sound swapping functions and the general swapping functions is that the sound functions let you convert buffers of data, whereas a single call to a general function can convert only one datum. If you wish, you can call the general functions (iteratively) rather than the sound functions. The correspondences between the functions are:

Sound function	**Analogous general function**
SNDSwapSoundToHost()	NXSwapBigShortToHost()
SNDSwapHostToSound()	NXSwapHostShortToBig()

Known Problems

See the **SoundKit.rtf** release note for information on known problems with sound.

Notes Specific to Release 3.0

These notes were included with the Release 3.0 version of the sound library. Sections that are no longer relevant have been marked with an italicized comment.

New Features

The following are new features that have been added to the Sound Library since Release 2.0.

- A new compression format, *Audio Transform Compression* (ATC), has been added for 3.0. The ATC format gives compression ratios greater than 3 and sometimes as high as 10 without affecting sound quality in most listening situations. It works by stripping out inaudible spectral features in the short-time Fourier transform. It is not ªbit faithful,º in that the decompressed sound is not bit-for-bit identical with the original sound. Any sampling rate may be used, but 44.1 kHz is considered standard, and the algorithm for deciding what is audible in the spectrum assumes this sampling rate.

  ATC compression and decompression each run in *real time* for *44.1 kHz stereo*. This is made possible by the imbedded DSP chip. ATC is now

the default compression format used by the command-line utility **sndcompress**, the sound library function **SNDCompressSound()**, and the **Sound** object's method **convertToFormat:** *SND_FORMAT_COMPRESSED*. Several new functions of the form **SND*ATC*()** have been added to support ATC.

The Audio Transform Compression algorithm was developed by Julius O. Smith at NeXT.

- Support for high-quality sampling-rate conversion by *arbitrary* factors has been added to **SNDConvertSound().** The **SoundEditor** programming example illustrates its usage in the *SaveTo* panel. The sampling-rate conversion algorithm is documented in the paper ªA Flexible Sampling-Rate Conversion Methodº by Julius O. Smith and Phil Gossett, ICASSP-84, San Diego, pp. 19.4.1-4. The algorithm does not yet use the DSP, so it runs out of real time.

- Sounds which formerly required the DSP to be played are now playable when the DSP is not available. In particular, CODEC sounds (8KHz, mulaw, mono) will be converted and played in real time using the 68040 when the DSP is busy. The same applies to mono-to-stereo conversion, mulaw-to-linear conversion and other simple conversions. Of course, the DSP is still used as an accelerator if it is available and if the DSP version has been written. In general, demanding signal processing tasks tend to run about five times faster on the DSP than on the 68040, including the

overhead of shipping data to the DSP and back via DMA. Therefore, some algorithms, such as ATC, simply cannot run in real time when the DSP is busy.

- A new function, **SNDVerifyPlayable()**, is provided for determining if a sound struct is ªplayableº using **SNDStartPlaying()**. The function assumes the DSP will be available if needed.

- The function **SNDConvertSound()** has been generalized to cover many more cases. In particular, *compression* to ATC format, *decompression* of any NeXT compression format, arbitrary *sampling-rate conversion*, *mu-law to or from linear*, *mono to or from stereo*, *float to or from linear, double to or from linear*, meaningful combinations of the above, and so on, are now handled. Unlike the case of playing sounds in real time, multiple-step conversions are supported. This function is normally accessed via the **Sound** object's **convertToFormat:** method. The **SoundEditor** programming example illustrates the more generally useful conversions by means of its new *SaveTo* panel.

Known Problems

- Compressed soundfiles must have either 22 kHz or 44 kHz sampling rates in order to be playable without first decompressing. In particular, an 8 kHz CODEC file (such as from the built-in microphone) must be first resampled to 22 kHz (e.g., by **SNDConvertSound()**) before compressing. Since ATC

format discards empty portions of the sound spectrum, up-sampling does not increase the file size as you might expect.

- Compression to ATC format ªramps onº the first 256 and ªramps onº the last 256 samples of the input soundfile. This means sounds which originally started or ended abruptly sound like they quickly ªfade inº and/or ªfade outº after being compressed. The work-around is to make sure the sound has at least 256 leading and trailing zeros.

- Some ATC-compressed sounds receive a small burst of noise at the end on playback. The noise can be made to disappear by adding 100 or so zeros to the end of the original sound and recompressing. You should never encounter this problem if you are already prepending and appending 256 zeros to avoid ramping distortion.

- The present ATC implementation on the DSP can only handle soundfile headers up to 2048 bytes long. The C version (used automatically when the DSP is unavailable) does not have this restriction. An oversized header will generally produce a DSP hang or a heavily distorted soundfile. To measure the size of a soundfile header, subtract the sound size reported by sndinfo from the soundfile size obtained via "ls -l" (both are in bytes).

- Many previously unplayable formats, such as stereo floating-point, may not keep up with real time on all configurations.

- The built-in sampling-rate conversion utility attenuates the upper 20% of the spectrum to prevent ªaliasingº using a high-speed anti-aliasing filter. This attenuation is most noticeable when up-sampling 8 kHz CODEC files to 22 kHz or higher. What you hear is a loss of ªbrightnessº or ªcrispnessº in the sound. The benefits of being on this point in the trade-off space are conversion speed and aliasing suppression. However, 8 kHz CODEC speech typically sounds better if brightness is preserved and some aliasing is allowed (primarily within noise-like consonants where it is hard to hear). As a result, in the special case where the requested format conversion is from monaural 8 kHz (linear or μ-law) to stereo 22 kHz linear format, a different anti-aliasing filter is used which preserves brightness but admits aliasing; this sounds better for typical CODEC speech.

  To obtain the high quality sampling-rate conversion (at the expense of execution speed), you may use the **resample** program Ða command-line utility available via anonymous ftp from **ccrma-ftp.stanford.edu**. Look for **resample.tar.Z** the **pub** directory there. It uses the same sampling-rate conversion algorithm but provides complete trade-off flexibility. The **resample** program was submitted for inclusion in the NeXT Public Domain CD ROM for Education, and it is contained in the CCRMA Music and DSP tools distribution. See the DSP release notes **DSP.rtf** for more information.

- Real-time compression using **SNDStartRecording()** does not support the bit-faithful or ATC modes, and the mode that is supported (ªdropped bitsº

format) gives a maximum compression factor of about 2 to 1. Use **SNDCompressSound()** to get full ATC or bit-faithful compression. The **sndcompress** command-line utility supports all three as well.

- The function **SNDStartRecordingFile()** does not support recording sounds in the compressed format.