Overview of NCDaudio

Definition Of NCDaudio

NCDaudio is a system developed by NCD for playing, recording, and manipulating audio data over a network. Like the X Window System, it uses the client/server model to separate application code from the specific drivers needed to control audio input and output devices attached to the X terminal or PC display station.

What Does It Provide?

Graphical user interfaces often result in dramatic increases in productivity because they allow information to be presented more effectively than plain text-based screens. Sound can further enhance an application by providing yet another channel for conveying information.

The NCDaudio service provides several new capabilities for applications:

- ❑ Sound data can be sent from the application to the desktop unit for playing on the internal or external speakers (if any).
- ❑ If a microphone or other audio input device is attached, sound data can be recorded and transmitted back to the application.
- ❑ Sound data can be stored in the audio server for later playback. This allows commonly-used sounds to be replayed without having to send them over the network every time.
- ❑ Multiple sources of sound data can be mixed and manipulated in a variety of ways (*e.g.*, made louder or softer, sampled more frequently or less).
- ❑ A variety of sound formats are supported, with conversion performed automatically within the audio server.

Applications use a library of simple function calls to read audio data from files and generate requests to the NCDaudio server.

What Is Audio Data?

Physically, sounds are waves of compressed or expanded air. We are able to hear them because the bones of the inner ear vibrate when struck by the waves. These vibrations are interpreted by the brain as sound. Similarly, we speak by reversing the process, causing air to pass over our vocal cords and vibrate back and forth. Audio devices such as microphones and speakers work essentially the same way.

Conveniently, these vibrations occur along a single direction. They can be measured by noting the position of the ***diaphragm*** that is pushing or being pushed by the air (see Figure 1). Microphones work by translating the position of such a diaphragm into electronic signals. When these signals are sent to a speaker, they cause another diaphragm to reproduce the original motion of the air, allowing us to hear the sound as it was produced.

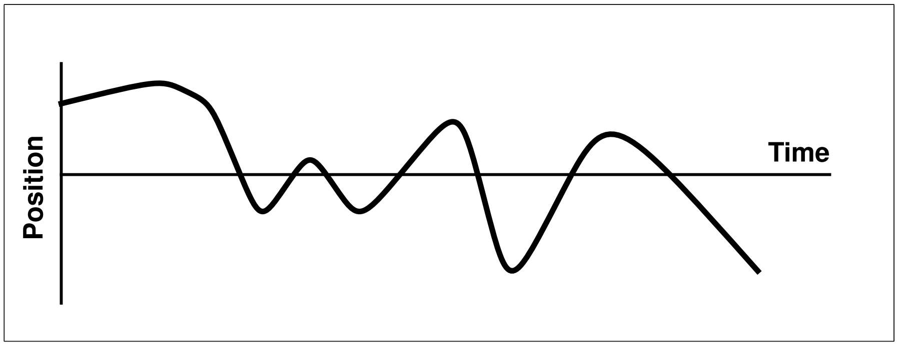

Figure 1: Continuous Graph of Audio Diaphragm Positions

Since the position of the microphone or speaker diaphragm varies smoothly with time, recording it exactly would require an infinite number of measurements. Digital systems such as computers and compact discs instead approximate the motion by taking ***samples*** of the positions, as shown in Figure 2. The number of positions recorded in a second is called the ***sample rate***; higher rates result in a more accurate representation of the original sound, but take more space to store.

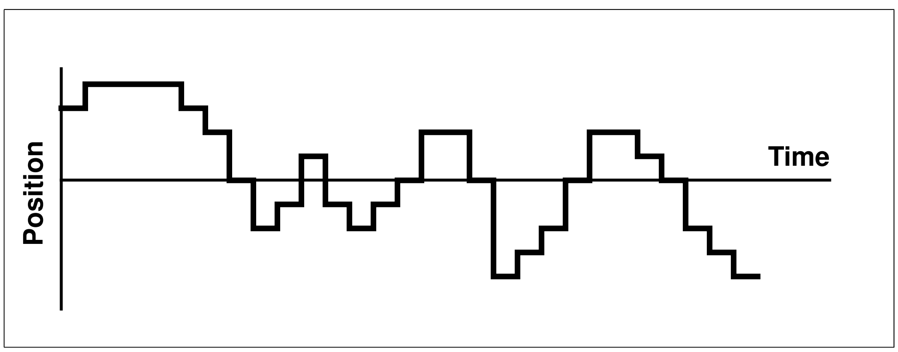

Figure 2: Sampled Graph of Audio Diaphragm Positions

These sequences of position samples are used in the NCDaudio service to represent sounds. Playing a sound involves sending a sequence to a speaker; recording takes a sequence from a microphone or other input device and stores it into the server's memory.

Data Formats

The sequence of numbers that are used to encode audio data vary in value from -1.0 to +1.0, representing the two extreme positions that a diaphragm can reach in a given device. The value 0.0 corresponds to the center of the device.

However, since floating point numbers occupy a large amount of space and are sometimes difficult to manipulate, most file formats store the sample values by multiplying them by 128 or 32768 and rounding to the closest integer. The most common formats are:

- ❑ Linear – In this format, each 8- or 16-bit value contains the scaled sample. The larger size allows the value to more accurately represent the original data. Some versions store the signed values directly; others add 128 or 32768 to make them be unsigned. The 16-bit versions also come in most-significant byte first and least-significant byte first varieties.
- ❑ μLAW – In this format, each 8-bit value contains the sign and the *logarithm* of the scaled sample. This enables the larger range of 16-bit values to be represented in 8-bits by trading accuracy at the further positions (which aren't used very often and can tolerate values being a little off) for precision at the positions nearer to zero (which are used most of the time).

The NCDaudio service supports both of these formats and their variants and automatically converts data when needed.

Audio Data Inputs

The NCDaudio service provides several sources of sound data:

- ❑ ***Physical Devices*** – Microphones, CD Players, and other types of audio devices can be plugged into the input jacks of the terminal or PC.
- ❑ ***Virtual Devices*** – The audio server itself provides software that emulates hardware tone generators and radio transmitters and receivers (for data that is broadcast over a local area network).
- ❑ ***Client Data*** – Applications can transmit sound data over the network to the server.
- ❑ ***Buckets*** – Audio data can be stored in the server, either as a result of being recorded from another device or having been sent over the network from the application.

Applications can hook up one or more inputs to one or more outputs. Operations such as mixing sounds, extracting individual tracks of data, and changing the volume are provided. The server routes the data to the appropriate devices automatically.

Audio Data Outputs

The NCDaudio service provides several destinations for sound data:

- ❏ *Physical Devices* – Speakers, tape recorders, and other types of audio device can be plugged into the output jacks of the terminal or PC.
- ❏ *Client Data*– Sound data can be sent over the network to an application. This provides the ability to use the X terminal or PC to record sound.
- ❏ *Buckets* – Sound data can be stored in server memory so that it can later be replayed, sent to the client, or copied to another bucket.

Outputs can receive data from more than one input at once; the server automatically handles mixing the data streams.

How Data Gets From Inputs to Outputs

The instructions that a client provides for arranging various inputs and outputs are called *Flows* (see Figure 3). They are used in much the same way as musicians use audio mixers or patch-panels. They indicate how the components should be connected, how multiple sounds should be mixed, and what changes in volume should be made.

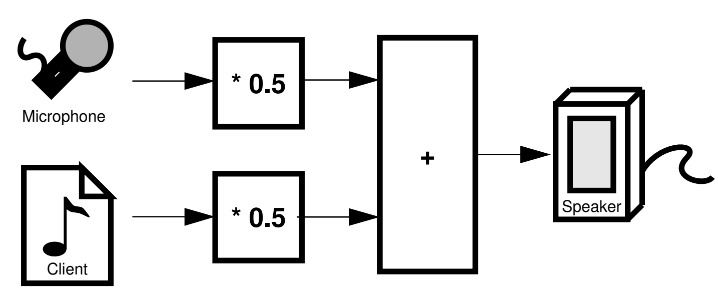

Figure 3: Flows Connect Inputs to Outputs

A flow is made up of a series of ***elements*** which describe sources of input data, operations to perform on data, and places to output the data. Sound data is ***imported*** into a flow from various input elements, and is ***exported*** to one or more output elements. A flow is often represented graphically as a tree whose leaf nodes are import and export elements and whose interior nodes are mathematical or logical operations.

The import and export elements used in a flow are referred to as the ***components*** of the flow. Data moves along a flow at the highest sample rate used by its components. The server automatically converts data to this sample rate when necessary.

At each step, the components in the flow are said to be in one of three ***states***, depending upon whether or not they are producing or consuming any real data:

- ***Stopped*** – An input element in this state has no real data ready and simply emits values of zero. An output element in this state discards any data that is directed to it.
- ***Paused*** – When an input element temporarily runs out of data or an output element has reached its limit, it becomes Paused. This is similar to being Stopped, except that the component expects that it can resume where it left off when more data or space becomes available.
- ***Started*** – When input and elements are producing or consuming data, respectively, they are said to be started.

When a flow is created, the components are initially placed in the Stopped state, preventing any data from flowing. Once at least one component in the flow has been started, data can begin to move. Components may change state in several different ways:

- The application can explicitly set the state of each element.
- An element in the flow can set the state of itself or other elements in response to changes in its own state.
- If an input temporarily runs out of data, it becomes Paused.
- In an input permanently runs out of data, it becomes Stopped.

By default, all components in a flow will automatically be stopped if any of the elements in the flow become Stopped or Paused (and have not reset themselves).

Performing Actions When Components Change

The ability to force input and output elements to trigger automatic actions in response to other elements allows the server to react immediately to changes that might cause pops and hisses. Each component element in a flow may have associated with it a list of ***actions*** to be performed when the element goes into a particular state:

- ***Change Element State*** – The state of any element (including the one triggering the change) can be explicitly set.
- ***Notify Client*** – An event detailing the current state of the element can be sent to the client that created the flow containing the triggering element.

Actions are often used when an input runs out of data to instruct another input start in its place (chaining the two inputs together) or to restart the input (looping). Applications also use Stopped notices to know when a flow is finished and can be destroyed. The default action for components entering the Stopped or Paused states is to stop or pause, respectively, all of the other components in the flow.

Manipulating Audio Data

One of the key features provided by the NCDaudio service is the ability to manipulate multiple sources of and destinations for sound data without client interaction. Conveniently, simple mathematical operations on the sample values (such as those shown in Figure 3) provide useful physical results:

- ❏ Adding two sets of samples together is equivalent to playing them at the same time.
- ❏ Multiplying a set of samples by a value greater than 1.0 increases the volume of the sound; a value less than 1.0 decreases the volume.

Most audio applications provide dynamic ways of changing the volume of the data being played. To support this, elements, such as the one used to multiply data by a constant, can optionally have ***parameters*** that can be changed at any time. The other information in a flow cannot be changed, except by redefining all of the elements in the flow.

Sound data comes from inputs in the range of -1.0 to +1.0 (even if it is stored differently). Within a flow, values may temporarily exceed this range, but are clipped back down when sent to outputs. Applications can send and retrieve data in a variety of formats; any conversion is handled automatically by the server.

Output devices that are attached to the server are shared among the various applications. When two or more flows direct sound data to the same output, the server automatically averages the data streams together. The result is that both sounds come out at the same time.

When the various components in an flow contain data of differing sample rates, the server internally converts all of the data streams to the highest rate. If this results in more samples for an output than it expected, the server automatically throws away any unneeded values.

Mono And Stereo

In addition to supporting simple monoaural data (such as voice annotations), the NCDaudio service also provides the ability to manipulate multitrack data such as stereo recordings. Applications can extract tracks from sound data and mix multiple tracks together.

Some types of multitrack output devices (especially stereo speakers) can be represented as collections of single-track devices that applications may also wish to access separately. Figure 4 shows an example in which stereo data from two separate clients (A and B) is sent to a stereo speaker which in turn uses two mono speakers. A third client (C) is sending mono data directly to the right speaker.

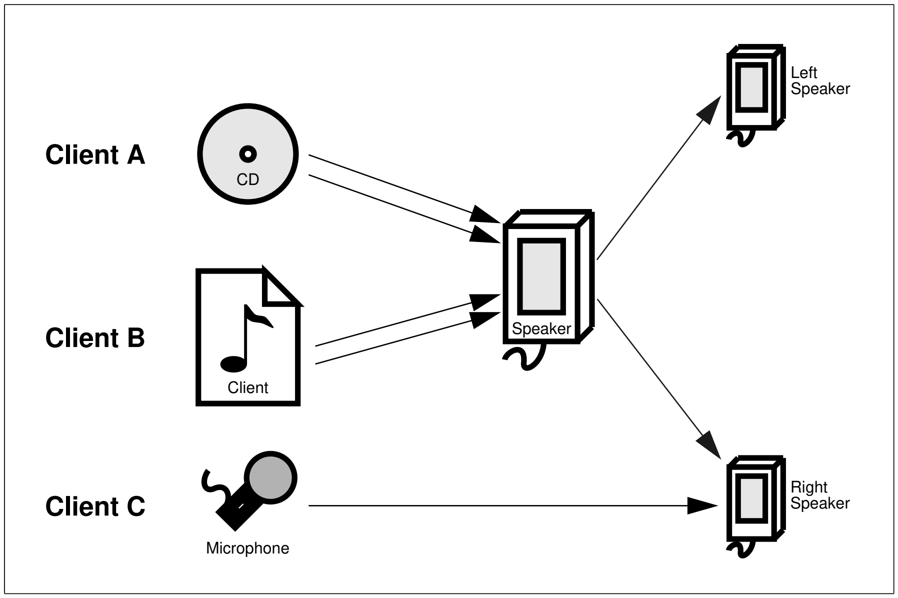

Figure 4: Stereo and Mono Outputs

In this situation, the two sets of stereo data are mixed as if the stereo speaker were a physical device. The individual tracks are then sent to the mono speakers, where the data for the right track is mixed with the data coming from Client C.

Most importantly, however, the volume levels of the left and right track of the data coming from the stereo speaker are automatically synchronized. If the right track is reduced as part of the mixing with the microphone data then the left track is reduced to match. This insures that the two tracks are played at equal levels.

By providing access to the separate mono speakers as well as the joint stereo speaker, the NCDaudio service gives applications fine-grained control over how sound is produced.

For convenience, however, most implementations provide at least one mono and one stereo speaker, even if they have to be simulated in software when appropriate hardware is not available.

Sending Data Over The Network

Although the NCDaudio service provides a number of input devices and built-in sound generators, its primary purpose is to allow applications to play and record audio data over the network.

Clients send data to a special type of import element in a flow. Data from this element is used in the flow just as if it were coming from any other input device. If a sound is to be used more than once, the application can store it in the server by creating a ***bucket*** object and exporting the data to it instead of to an output device.

This process can be reversed to retrieve audio data from the server. Large amounts of data can be passed back to the application directly from the various input devices without requiring temporary storage for all of it in the server. Although the most common use of this mechanism is to record data from microphone, it can be used with all supported input devices (*e.g.*, buckets, tone generators, or radios).

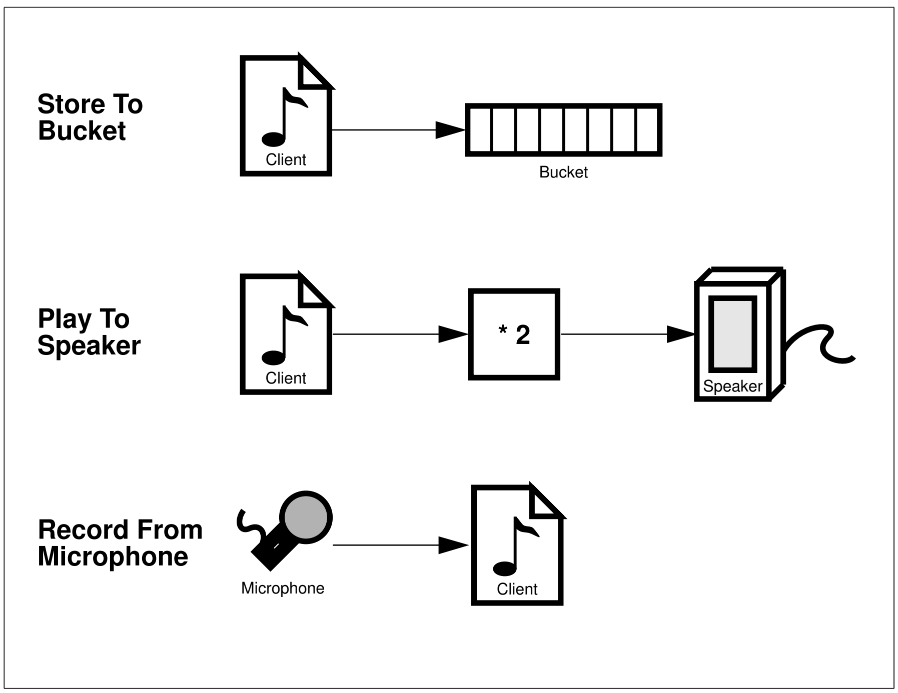

Figure 5: Common Uses For Client Data

Figure 5 shows several examples of very simple flows that transfer data to and from the server. Putting a multiplier immediately before a speaker output allows the volume of the data to be adjusted dynamically.

In a client/server architecture, network transfer delays can sometimes make the arrival of data less predictable than if it were coming from a physical device. This can result in underruns (data not arriving in time) or overruns (more data arriving than there is room for) if the delays are sufficiently large.

The NCDaudio service provides several methods of avoiding these problems. When an underrun occurs in an input element or an overrun occurs in an output element, the component enters the Paused state. Unless a different action is specified by the application, this causes all other components in the flow to be paused.

Storing Data In The Server

Applications that use certain sounds more than once have the ability to store the data in the server in objects called ***buckets***. Such data can then be used repeatedly as input to other flows without have to transfer it across the network again. This technique is commonly used in applications that use audible cues to draw the user's attention.

Some implementations also provide built-in buckets containing prerecorded sounds.

Virtual Input Devices

As a convenience to the application, several types of simulated input devices are provided by the NCDaudio service. A variety of builtin ***wave forms*** (see Figure 6) can used to generate simple waves. Future implementations will also provide ***radio*** devices that reads from or writes to the network using a datagram protocol.

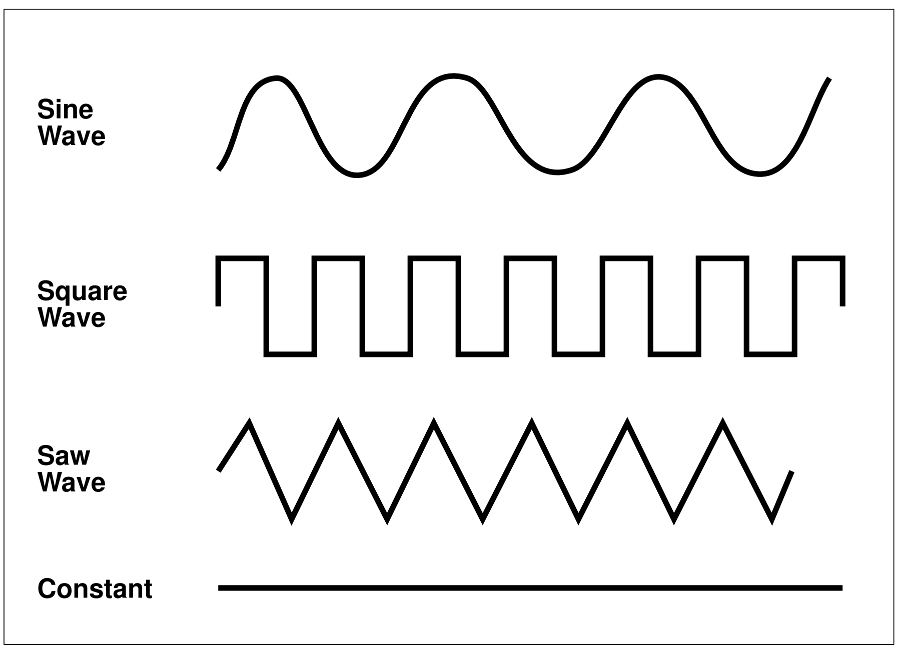

Figure 6: Sample Builtin Wave Forms

Summary

The NCDaudio service provides a mechanism for transferring audio data between applications and the desktop X terminal or PC. Applications specify how various inputs and outputs should be hooked together; the server automatically routes data to the proper destination and does any necessary conversions.

Sounds may be stored in the server and reused multiple times or can be sent directly to attached output devices such as speakers. Applications may dynamically adjust the volume at which the sounds are heard.

Input devices such as microphones can be used to record audio data. Applications can read the data back over the network, store the results in the server for later use, or even redirect it to an output device.