4. Listing, playing and creating soundfiles

(This section last updated August 2008)

Hard disk soundfiles can be created in many ways. The most common means in the ECMC studios include:

1) Recording (digitizing) one or more acoustic sounds with an analog-to-digital converter and writing the samples into soundfile.

2) Digitally copying samples from an audio compact disc, from an audio DAT tape, from a DVD disc, a DV Cam or Mini DV tape or from some other digital medium directly into a soundfile.

3) Using a software synthesizer such as *Csound* to compute samples directly and write these samples into a soundfile.

4) Using an audio software application to perform digital signal processing (mathematical operations that alter a sound) on one or more existing soundfiles and writing the processed (altered) samples to a new soundfile. This include mixing two or more existing soundfiles together to create a new, composite file.

4.1. Soundfile formats

Soundfiles created by any of the means above consist of *samples* — numbers that represent the amplitude of a sound waveform at very short, evenly spaced time intervals. On any computer audio system, samples are created and written to disk according to a particular *data format*, which tells the system hardware and software how to interpret these numbers (how the samples are written to the file — for example, big-endian, in which the most significant byte of each sample is written first, or little-endian, in which the least significant byte is written first). Unfortunately, several types of digital audio formats are in common use today, and most of these formats are incompatible. If we wish to copy a soundfile made on, say, a PC computer running *Windows* to another system that generally employs a different soundfile format, we may need to run the source soundfile through a soundfile conversion or "translator" program that converts the file data to a format usable by audio applications on the receiving system. Perhaps some day there will be a "universal" soundfile format, comparable to the MIDI standard, but that day is not here yet.

Some of the most common soundfile formats and their usual abbreviations (often appended to soundfile anmes) are given in the following table:

Some Common Soundfile Formats

Format	Filename extension	Platforms
WAVE (or RIFF)	*.wav*	Windows, Linux (some Macintosh aplications)
AIFF	*.aiff* or *.aif*	Macintosh, (some Windows and Linux applications)
AIFF-C	*.aifc*	Macintosh
NeXT/Sun	*.snd* or *.au*	Sun, NeXT (rarely used now)
CDR	*.cdr*	audio compact discs
IRCAM	*.sf*	Developed at IRCAM in the early 1980s (no longer widely used)
mp3	*.mp3*	mpeg 1 layer 3 compression
ogg vorbis	*.ogg*	Ogg Vorbis compression
FLAC	.flac	an open source lossless encoding/decoding codec

On *Windows* systems a dot followed by a three letter filename extension (usually *.wav*) generally must be appended to the name of each soundfile to indicate to the system software that the file is a soundfile in the indicated format. Most graphical Macinstosh audio applications also require filename extensions. Some Linux music applications either require or else "strongly prefer" the use of filename extensions. Other Linux music applications, (e.g. the *sweep* soundfile editor) do not require these extensions. To be on the safe side, it is best to append file format extensions to the names of all Linux soundfiles.

☞ Macintosh systems employ **aiff** and, much less frequently, **aiff-c** soundfile formats, originally devised by Apple Computer. However, many Mac music applications also are able to read (but not always also to write) soundfiles in WAVE format.

The *aiff-c* format is an extension to the original *aiff* specifications, and supports some data compression schemes. Besides the standard header information and sample data, *aiff* and *aiff-c* files can include additional "chunks" of information, such as loop points, the pitch of the soundfile, MIDI data, synthesizer or sampler "voice" data, and comments.

☞ On Windows and Linux systems the default audio format is *WAVE*, which was developed by Microsoft. However, many higher end *Windows* audio applications, such as *ProTools*, *Nuendo* and sequencers such as *Cubase* also are able to read AIFF format.

Linux audio development has been concentrated on PCs with Intel or Athlon processors (often, during the 1990s, on dual-boot *Windows/Linux* systems), and WAVE format became the norm. During recent years, however, many audio applications initially developed on SGIs, Macintoshes and other types of Unix boxes, have been ported to Linux. As a result, many Linux audio applications can only read and write WAVE format soundfiles. Applications we use to write audio cds also require WAVE format for input soundfiles. Many other applications, such as the soundfile editors *rezound* and *sweep*, and *Csound*, can both read and write to WAVE, AIFF and certain other formats. A few applications, such as the mixing application *rt*, can read both WAVE and AIFF format, but can only write to one of these formats (generally to WAVE format). Unless you will be exporting soundfiles for further processing or mixing on a Macintosh system, I recommend that you work exclusively with WAVE format on the ECMC Linux systems, since there are excellent applications for practically any audio task that require or permit the use of WAVE format.

Soundfile headers

In addition to its sample data, most soundfiles (and all soundfiles created on ECMC systems) also include a *header*, a short (often 1 kB) summary of information about the file and its format that is written at the very beginning of the file. When a music application, such as a program that plays or mixes soundfiles, opens a soundfile, it immediately reads this header, which provides the information necessary for the program to read and process the sample data correctly. The header specifies the sampling rate, the number of channels, the number of bytes and byte order and certain other necessary format characteristics of the sample data. On Linux and other Unix-based systems, the header also specifies the owner and read/write permissions.

Sampling rates:

The *Delta 1010* audio interfaces and converters installed on the Linux and Windows systems in room 52, as well as the RME interfaces connected to Windows machine *igor* and Macintosh system *wozzeck* in the MIDI studio, all support sampling rates of 44100, 48000, 88200 (rarely used) and 96000. For the present, however, if your music ultimately will wind up on a cd, or unless you are certain that it can be played back at 96 k and/or 24 bits or 32 bit floats, I recommend creating soundfiles at the traditional 44100 sampling rate and 16 bits. Even lower sampling rates such as 22050 or 32000 occasionally can be useful for certain real-time tests and applications (such as running a software synthesizer with complex instruments in real-time).

Bit depth

Another important format element is *sample word size* or *bit depth* — the number of bits contained within each sample. This *sample word size* varies on different types of consumer and professional quality systems between 8 bit, 16 bit, 18, 20, 22 and 24 bit integer resolution. 32 bit floating point resolution (and even 64 bit float resolution) also is available with a few applications (e.g. with *Csound*). However, floating point soundfiles, while offering higher resolution than integer formats, require conversion to integer format (and, in with certain applications, bit depth reduction) before they can be played by some audio applications and systems. Until recently 16 bit integer resolution was the "professional" audio standard, but 24 bit resolution (allowing representation of values between +/- 8,388,608) is becoming the new "professional audio" standard. The additional bits can provide increased signal resolution and fidelity, heard most easily in the smooth decay of tones and in soft passages.

While twenty-four bit integer resolution is becoming the new professional audio standard, it does introduce some complications, since 24 is not a power of 2. Special purpose audio hardware, such as the digital multitrack recorders by such manufacturers as Alesis, Mackie and Tascam, can be designed and constructed with 24 bit registers. General purpose desktop and laptop computers, however, are 32 bit (or 64 bit) systems, loading and processing 32 (or 64) bits simultaneously. There are three ways in which 24 bit samples (comprising 3 rather than 4 bytes) can be written on 32 bit systems:

1) *24 bit packed* (the most common 24 bit format)
Each sample takes up 3/4 of a 32 bit word, and the samples overlap words, with the second sample beginning with the last byte of the first 32 bit word. Four samples are "packed" into three computer words.

	24 bit packed samples of 3 bytes: (4 samples shown here: samples 1, 2, 3 and 4)		
Bytes:	1 1 1 *2*	*2 2* 3 3	3 *4 4 4*
Bytes:	1 1 1 1	*2 2 2 2*	3 3 3 3
	computer words of 32 bits, 4 bytes: (3 words shown here: words 1, 2 and 3)		

This is efficient for storage, wasting no disk space. However, before the samples can be processed or played, they must be "unpacked" into 32 bit words padded with an extra byte of zeros. This overhead means that it takes slightly longer (in processor time) to record, play or mix 24 bit 3-byte packed samples than to record, play or process 16 bit or 32 bit samples.

2) *unpacked into 32 bits, left justified*
Here, the "container" for each sample is a full 32 bit bit word. The three bytes of each sample are written as the first three bytes of the computer word, followed by a "blank" byte of 8 zeros that is ignored by audio applications. This requires 1/4 more disk space for storage of a soundfile, but enables soundfiles to be processed somewhat more quickly.

3) *unpacked into 32 bits, right justified*
This format is similar to #2 above, writing and storing each sample in one 32 bit word, but here the "dummy" (zero) byte is the first rather than the last byte of each word. This is the least common of the three 24 bit storage formats.

24-bit soundfiles written by ECMC Linux audio applications (such as Csound, *pitchshift* and *sfnorm*) employ *packed* the format. Some applications, however, may ask you to specify a particular 24 bit format when saving a file.

Audio channels

With the proliferation of 5.1, 6.1 and 7.1 "surround sound" formats, and of *ambisonic* and other alternative types of multi-channel sound spatialization and diffusion mixing and sound processing techniques, *multiple channel playback* is becoming an increasingly common and important resource in computer music production. However, in order to create multichannel (more than 2 channel) soundfiles, you must have:

1) Hardware that supports the desired number of channels at the desired sampling rate and bit depth
To play a 5.1 surround format soundfile, with all channels at 9624 resolution, you must have
• an audio interface and converters that provide at least 6 channels of 96k 24 bit audio
• 5 matched loudspeakers and a matched subwoofer
(Obviously stereo headphones will be of no use here.)

2) Software that supports the desired number of channels at the desired sampling rate and bit depth
Many audio applications only support stereo input and output. Others support only a limited number of 9624 channels.

One also needs to be cognizant of the storage space requirements of high resolution audio (e.g. 9624 resolution), particularly when creating multichannel works. Consider the following disk space requirements for an 8 second soundfile at various sample rate/bit depth resolutions for various numbers of channels:

44.1k 16 bit mono	.7 MB	96k 24 bit mono	2.3 MB	96 k 32 bit float mono	3.01 MB
44.1k 16 bit stereo	1.4 MB	96k 24 bit stereo	4.6 MB	96 k 32 bit float stereo	6.02 MB
44.1k 16 bit quad (4 channel)	2.8 MB	96 24 bit quad	9.2 MB	96 k 32 bit float quad	12.04 MB
44.1k 16 bit 5.1 (6 channels)	4.2 MB	96k 24 bit 5.1 surround	13.8 MB		
44.1k 16 bit 8 channel	5.6 MB	96 k 24 bit 8 channel	18.4 MB	96 k 32 bit float 8 channel	24.08 MB

WAVE, *WAVE-EX* and *Broadcast WAVE* formats

The *RIFF* ("Resource Interchange File Format") *WAVE* format was originally designed by Microsoft to handle 16 bit integer and 32 bit float mono and stereo soundfiles. It has been used for soundfiles with integer bit depths of more than 16 bits, and for soundfiles with more than two channels, but not always successively or reliably, especially when such soundfiles are used with a variety of software applications or are imported for use on another system. There is nothing in the *WAVE* format specifications that specify how 24 bits are written (whether "packed" or in left- or right-justified 32 bit words), nor how the channels of multichannel soundfiles are to be mapped to loudspeakers. For example, a 5.1 surround mix will include six channels, but is channel *3* intended for the front center speaker, for the left rear speaker or for the right rear speaker?

In addressing such questions, Microsoft has stated that *WAVE* format should not be used for soundfiles with more than two channels or more than 16 integer bits. Instead, Microsoft has devised a new format, called *WAVE-EX*, that is intended for use with higher bit depths and multiple audio channels. The *WAVE-EX* specifications include header fields that define how non-power-of-two bit depths are mapped onto 32 or 64 bit words, and also 18 speaker positions to which the individual channels of a soundfile can be mapped. These speaker positions are:

1 FRONT LEFT
2 FRONT RIGHT
3 FRONT CENTER
4 LOW FREQUENCY
5 BACK LEFT
6 BACK RIGHT
7 FRONT LEFT OF CENTER
8 FRONT RIGHT OF CENTER
9 BACK CENTER
10 SIDE LEFT
11 SIDE RIGHT
12 TOP CENTER
13 TOP FRONT LEFT
14 TOP FRONT CENTER
15 TOP FRONT RIGHT
16 TOP BACK LEFT
17 TOP BACK CENTER
18 TOP BACK RIGHT

Note that channel locations 1 through 6 specify how the channels within a 6 channel soundfile should be mapped to a 5.1 surround speaker array. Channels 1, 2, 5 and 6 specify a typical American quad array. Distribution of 8 channels to an *ambisonic* cube speaker array with American channel numbering would require the use of channels 1, 2, 5, 6, 13, 15, 16 and 18.

The *WAVE-EX* specifications also include facilities for automated mixdowns of multichannel soundfiles to the number and arrangement of loudspeakers available. Thus, if at 8 channel soundfile designed to be played on 8 speakers in the corners of a cube is played on a system with only 6 loudspeakers, the 8 channels can be automatically mixed down for playback on the 6 speakers. Additionally, floating point files can be played directly, without conversion to integers. When downsampling from higher to lower sampling rates, or from more to fewer bits, dithering can be added automatically so that quantization noise will be reduced.[1]

Somewhat surprisingly, however, the new *WAVE-EX* format has been slow to gain acceptance. As of this writing is supported only by a very few Windows, Mac and Linux audio applications, and therefore is not recommended.

Another offshoot of *WAVE* format that you may run across is *Broadcast WAVE,* a format devised by European broadcasters that includes one or two additional chunks of header information that address ambiguities in WAVE format.[2]

4.2. Soundfile directories

Soundfiles often are much larger than most other types of files, and for optimum system throughput these soundfiles should be stored in large, contiguous data blocks on a hard disk. On ECMC Linux systems, therefore, therefore, all soundfiles are stored on one or more large disks partitions reserved exclusively for this purpose.[3]

[1] The dithering algorithm specified is a Triangular Probability Density Function algorithm

[2] See *www.ebu.ch/departments/technical/pmc/pmc_bwf.html* or *www.ebu.ch/trev_274-chalmers.pdf* in the unlikely event you need more information on *Broadcast Wave* format.

On ECMC audio production systems, there are two basic types of soundfiles:

(1) the soundfiles of individual users, and

(2) a "public domain" collection of soundfiles, permanently available to all users, which we call *sflib* (the "**s***ound***f***ile* **lib***rary*"). In turn, there are two *sound file library* collections:

- our original collection, known as *sflib*, which contains soundfiles at 44.1 k 16-bit resolution
- a newer (and smaller) collection, known as *sflib96*, which contains soundfiles at 96 k 24-bit resolution

Each user has individual control of his/her own soundfiles. But while the *sflib* collection can be accessed (played and used) by all users, only *root* can modify or delete *sflib* soundfiles. Thus on ECMC Linux and Macintosh systems there are three distinct soundfile disk "areas," or bottom level directories, which are known as:

- */snd*, where the soundfiles of all individual users are stored, and
- */sflib*, where all 44.1 k 16-bit public domain samples are stored
- */sflib96*, where all 96 k 24-bit public domain samples are stored

On ECMC *Linux* and *Windows* systems, the *sflib* and *sflib96* soundfiles are in WAVE format, and include *.wav* file name extensions. Within this *Users' Guide*, and in other ECMC documentation, references to particular *sflib* soundfiles may or may not include *.wav* extensions. Just remember that on our Linux and Windows systems all playable *sflib* soundfiles include these extensions.

4.3. Commands to list and play soundfiles

The three most common ways we access previously created soundfiles are by

- *listing* them (to find out what soundfiles currently are available within a particular user's or *sflib* directory);
- viewing *header information* (to find out, or to confirm, the sampling rate, number of channels, duration or other characteristics that we may need to know when using a soundfile); and, of course,
- *playing* soundfiles

The basic commands, and some alias abbreviations, used to accomplish these three functions from a shell window are summarized in the following table, first for soundfiles in the two *sflib* collections, then for soundfiles on your own (and other users') */snd* directories. Detailed explanations of these commands, with examples, are presented in the following pages.

Summary of commands to list and play soundfiles			
	List soundfile(s):	Display **header information:**	**Play** soundfile(s):
sflib (44.1k 16-bit) soundfiles : **sflib96** (96k 24-bit) soundfiles :	**lsflib** (**lsfl**), **findsflib** **lsflib96** (**lsfl96**), **findsflib**	**sflibinfo** or **sfli** **sflibinfo** or **sfli**	**playsflib** or **psfl** **playsflib** or **psfl**
snd (user) soundfiles :	**lsf** **findsnd**	**sfinfo** or **sndinfo** or **si**	**play** or **p** **findsnd -p**

In these and other music software command names, you can think of the character strings *sf* and *snd* as abbreviations for "soundfile," and the character string *sfl* as an abbreviation for "sflib" or "soundfile library."

4.4. The sflib collections

The two ECMC *sflib* collections consist primarily of isolated acoustic instrument and vocal tones and environmental sounds that are available as source material to all users. In order to facilitate use of these thousands of soundfiles, we have organized them by type into directories. There are no soundfiles within */sflib* or */sflib96* directories, but only sub-directories (folders) where the actual soundfiles samples are

[3] On the ECMC Windows and Macintosh systems, by contrast, a user's soundfiles are stored along with other, smaller types of files within the user's folder on a *USERS* disk partition. As disk partitions begin to fill up, this often leads to fragmentation of soundfiles, which is very undesirable.

located.

The 44.1k sflib collection:
The 3600 or so 44.1k */sflib* soundfiles are organized into the following directories:

44.1k /sflib directories : (as of August 2007)
Playable sound samples:

- **/sflib/africa** : sounds from Africa
- **/sflib/america** : sounds of native Americans
- **/sflib/australia** : sounds from Australia
- **/sflib/brassloop** : brass samples designed for looping; many of these samples will not sound well or be usable without looping by programs such as Csound
- **/sflib/chinaperc** : percussive sounds from China
- **/sflib/choir** : a large collection of mono and corresponding stereo (*ST*) chorus multisamples
- **/sflib/choir2loop** : multisamples of child and female choirs
- **/sflib/env** : environmental sounds, such as a waterfall
- **/sflib/gamelan** : metalophones, idiophones and flutes we have digitized from instruments in the Eastman gamelan ensemble
- **/sflib/harp** : harp multisamples
- **/sflib/japan** : sounds from Japan
- **/sflib/kb** : Western keyboard chordophones (several groups of piano and harpsichord multisamples)
- **/sflib/perc** : soundfiles of Western percussive (idiophone and membranophone) acoustic sources
- **/sflib/prosonus** : a small, motley collection of instrument tones and sounds
- **/sflib/string** : non-keyboard chordophone (string instrument) sources,
- **/sflib/tools** : samples of physical tools such as hammers and files
- **/sflib/voice** : vocal sound sources, currently including soprano, alto and bass tones

  (The tenor did now show up for the recording session.)

  including Western orchestral strings and guitar multisamples
- **/sflib/wind** : Western aerophone (wind instrument) tones
- **/sflib/world** : sounds from various non-Western cultures
- **/sflib/worldstring** : samples of non-Western chordophones

- *Example soundfiles*: the **sflib/x** directory

Directories with other types of files (non-playable):

- **/sflib/anal** : This directory is organized into subdirectories that contain binary timbral analysis files of sounds (mostly of sounds from other *sflib* directories). These analysis files are used to resynthesize (with modifications) the source sound.
- **/sflib/csutil** : This directory contains the attacks of certain instrument tones used by certain Csound physical modeling unit generator opcodes.

(The *x*, *anal* and *csutil*directories generally are not included on ECMC Windows and Macintosh systems, since they primarily illustrate or facilitate the use of Linux software.)

Notes on the 44.1k sflib soundfiles:

(1) Almost all of the soundfiles have been normalized to a maximum amplitude of about 32000 (on a 16-bit integer scale of 0 to 32767). Tones played loudly and softly differ very little, if at all, in amplitude, but differ significantly in timbre.

(2) The great majority of these soundfiles are monophonic. The names of stereo soundfiles, many of which are located within the *env* directory, begin with *ST* (for example *STwind.low.wav*, a stereo version of the monophonic */sflib/env* soundfile *wind.low.wav*).

(3) For most of the soundfiles that have a well-defined pitch, this pitch is indicated by an alphanumeric abbreviation, identical to that used with the *Score11* and *nGen* keyword *notes*, which denotes

- the *pitch class* (*a* through *g*), modified, if necessary, by an *s* ("sharp") or an *f* ("flat"), followed by

- a digit representing the *octave number* of the note.

The twelve chromatic tones beginning with the lowest *c* on the piano keyboard comprise octave number1 (*c1, cs1, d1 ... b1*), the next twelve chromatic tones comprise octave *2* (*c2, cs2 , d2*, etc.), and so on, up through octave *8*. *Middle c* and *"tuning" a* (440 herz) are represented by *c4* and *a4*. The highest *c* on the piano keyboard is *c8*.

(4) Within the **voice** directory, there are three groups of soprano, alto and bass tones:

- Group **1** tones (e.g. soundfile *sop1.b3*) consist of long tones sung loudly, with vibratos ranging from "normal" to fairly wide;
- Group **2** tones (e.g. *sop2.a3*) consist of long tones sung softly with minimum or no vibrato;
- Group **3** tones consist of short notes (less than one second in duration) sung loudly.

(5) Within the **string** directory, **.p** soundfiles are *pizzicato*, **.h** soundfiles are *harmonics*, and **.m** soundfiles are *martele* (short, loud tones with sharply accented attacks).

(6) Soundfiles whose names include the character string *loop* are designed for looping with signal processing programs such as *Csound*. Some of these soundfiles can be used without looping with acceptable results. But with others there may be no amplitude decay to the tone, or the timbre may not evolve "naturally," and these samples generally will be unusable without some processing.

(7) An *ecmchelp* file, called *sflib*, provides more information on many of the *sflib* soundfiles. Additional *ecmchelp* files (such as *sflibgamelan* and *sflibgamelan*) contain information on the sounds within particular *sflib* directories.

The 96k sflib collection:

Currently the *sflib96* collection is much smaller — a total of about 760 soundfiles as of August, 2006, although we expect this number to grow considerably over the next few years. However, there are more large sets of multiset samples (groups of soundfiles with similar timbres and articulation, designed to be used together). Consequently, the 96 k *sflib* samples are organized into more levels of nested folders (subdirectories) than in the *44.1k /sflib* collection. The following table indicates the current subdirectory structure for the */sflib96* collection:

/sflib96												
brass								perc				voice
trb				btrb		euph		block	cym	gourd	shaker	heather
gliss	harmon	open	straight	open	straight	open	mute					

There are no soundfiles in the bottom level *sflib96* directory — only the subdirectories *brass*, *perc* and *voice*. Likewise, these three subdirectories do not contain soundfiles, but rather further nested subdirectories.

☞ The *perc* folder includes four subdirectories where the actual samples are located.

☞ The *brass* folder, however, contain three additional folders (trombone, bass trumpet and euphonium) which in turn contain yet another level of folders where the samples are located. There are 108 samples in the *brass/trb/open* folder, 73 in the *trb/straight* folder, 63 in the *harmon* folder and 26 in the *gliss* folder. Within the *trb open*, *straight* and *harmon* subdirectories there are multisampled sets of tones played loudly (*.ff*), mezzo forte (*.mf*) and softly (*.pp*), short tones (*.S* under one second in duration, and sforzando (*.sfz*) tones.

☞ The 60 or so soundfiles in the */sflib96/voice/heather* directory are multisample recordings of soprano Heather gardner.

4.4.1. Listing and playing SFLIB soundfiles

☞ *Listing sflib soundfiles:* **lsfsflib** (**lsfl**) and **lsfsflib96** (**lsfl96**)

Listing soundfiles provides us with a list of one or more soundfiles within a particular directory on a sound disk. A "short" listing returns only the names of these soundfiles. A "long" listing returns additional information (which beginning users seldom need) concerning the read/write permissions, and the size and the creation date of these soundfiles.

• To list the contents of the bottom level *sflib* or *sflib96* directories, type:

lsfsflib or its abbreviation **lsfl** (44.1k */sflib* directories)
or
lsfsflib96 or its abbreviation **lsfl96** (96k */sflib* directories)

in a shell window.

• To obtain a "short" listing of the names of all of the soundfiles or additional folders within one of these *sflib* or *sflib96* folders, type:

lsfsflib directory_name (or **lsfl directory_name**) (44.1k */sflib* directories)
or
lsfsflib96 directory_name (or **lsfl96 directory_name**) (96k */sflib96* directories])

Examples:

lsfl wind will return the names of all soundfiles within the *wind* directory of the *sflib* partition.

lsfl96 brass/btrp/open will display the he names of all soundfiles within the 96k *brass/btrp/open* folder

To obtain a "long" listing of either of these folders, include a *-l* flag:

lsfl -l wind or else *lsflib -l wind*
lsfl96 -l brass/btrp/open or else *lsflib96 -l brass/btrp/open*

• To list the names of one or more particular soundfiles within an *sflib* or *sflib96* subdirectory, type:

lsfsflib directory/soundfilename(s) or *lsfsflib96 directory/soundfilename(s)*

Examples: The command: *lsflib wind/clar.d3.wav*
will list this clarinet soundfile. You must include *.wav* filename extensions when listing individual *sflib* soundfiles.

Unix wild card characters, such as ***** (which means "all" or "every"), can be used within arguments to all commands that list soundfiles. Examples:

*lsfl96 brass/trb/open/*sfz** will list all the soundfiles in the */sflib96/brass/trb/open* folder whose names include the character string *sfz*.

lsflib string/.p** will list all of the pizzicato (*.p*) soundfiles within the *string* directory.

Other flag options, identical to those available with the Unix *ls* command, also are available. A *-R* flag will recursively list all soundfiles, folders and soundfiles within these folders beginning with the requested direcory. A *-1* ("one") flag will display the results in a single column rather than in multiple columns. Flag options can be given individually or concatenated (joined together) in any order after a single - sign. Examples:

lsfl -1 voice
will display all soundfiles in the */sflib/voice* folder in a single column.

lsfl96 -1 -R brass or *lsfl96 -R1 brass*
will list all subdirectoires and their contents within the 96k *brass* folder and will display the results in a single column. If the display goes whizzing by too fast, pipe it into a paging program such as *more* or *less*: *lsfl96 -R1 brass | less*

lsf -R /sflib /sflib96
will display a complete list of all 44.1 k and 96 k *sflib* soundfiles.

The output of *lsflib* or *lsflib96* can be piped into another Unix program such as *grep* for filtering:

lsfl -1 string wind | grep .c4
will display all soundfiles within the */sflib/string* and the */sflib/wind* folders whose names include the character string *.c4* ("middle C).

lsfl -1 env/b | grep -v bird*
will display all files within the */sflib/env* directory whose names begin with a "b" except for those whose names also include the character string *bird*.

☞ *Displaying header information for one or more /sflib* or */sflib96* soundfiles: **sflibinfo** or **sfli**

To look at the header information for one or more 44.1k */sflib* or 96k */sflib96* soundfiles, type:
sflibinfo filename(s) or its abbreviation **sfli filename(s)**

You do not need to type the full path to the soundfile[4] but only the soundfile name. *sflibinfo* will search the *sflib* and */sflib96* directories until it finds a match. You can not use metacharacters such as *, but you can omit *.wav* filename extensions if you wish. Typing either

sflibinfo sleighbells or *sflibinfo sleighbells.wav*

will display information format about /sflib/perc/sleighbells.wav

To obtain a succinct, one line header summary, include a *-s* flag before your soundfile argument(s):
The command *sflibinfo -s cym.mlow.edge.ff.wav cym.mlow.edge.mf.wav cym.mlow.edge.pp.wav*
will display concise header information for these three soundfiles in the */sflib96/perc/cym* folder.

☞ *Playing SFLIB soundfiles :* **playsflib** or **psfl**

To play one or more soundfiles located within any 44.1k */sflib* or 96k */sflib96* directory (or a series of soundfiles located within various directories), use the **playsflib** command or its abbreviation **psfl**. In typing in (or copying-and-pasting in) soundfile names you can omit the *.wav* file name extensions if you wish.
Example: *psfl sleighbells* or *playsflib sleighbells.wav*
will cause the soundfile */sflib/perc/sleighbells.wav* to be played.

Example: The command

psfl sax.bari.cs2 vln.g3 trg1 gourd4

will play soundfile *sax.bari.cs2.wav* from the *sflib wind* directory, followed by soundfile *vln.g3.wav* from the *string* directory, then the triangle *trg1.wav* from the *perc* directory and finally, for a grand finale, */sflib96/perc/gourd/gourd4.wav*.

Note that, unlike the command to *list* information, the command to *play* these *sflib* soundfiles need not include the directory path of the soundfile. If the soundfile exists in any */sflib* or */sflib96* directory, it will be hunted down and played.

To **kill** (stop) the *play* program at any time (when you don't wish to hear the rest of a soundfile), type *control* **c** (hold down the *control* key and tap the *c* key).

☞ The **findsflib** utility

Perhaps we would like to obtain a listing of all the drum-like sounds within all of the *sflib* and *sflib96* directories. We might be able to figure out some complicated sequence of commands involving the *lsfl*, *lsfl96* and *grep* commands, but this would be tedious, especially for an impatient person like me. An easier method is to let the *findsflib* utility do this ant work for us:

findsflib drum

The *findsflib* script will search all of the 44.1k and 96k *sflib* directories for files whose names include the character string *drum*, and display these matches. If we include a *-p* flag on the command line, like this:

findsflib -p drum

then *findsflib* will not only list all matches, but also will play each of these soundfiles.
For further information on using this command, type *findsflib* with no arguments. For more complete information, including additional flag options, consult the *man* page for this script, available both online and in the hardcopy *LINUX DOCs* binder. Many other soundfile utility programs similarly provide two levels of usage help; typing the command name with no arguments displays a usage summary, while typing *man commandname* (or else consulting the hardcopy printout of this *man* page in the
LINUX DOCs binders) provides more detailed documentation.

4.5. User soundfiles

Each user has an individual "home soundfile" directory (folder) on the *snd* disk of ECMC Linux system *madking*. Your home soundfile directory has the name
/snd/USERID

[4] In fact, typing the full path name to a soundfile with the *sflibinfo* command likely will produce an error message.

where *USERID* is your login name. My home soundfile directory is */snd/allan*.

Thus, when you log onto *madking* or *wozzeck* and open a shell window, you are "placed" simultaneously in two home directories:

> */home/USERID* (Linux) or */users/USERID* (Macintosh) — the area on the system disk where your ASCII files are stored; and
> */snd/USERID* — the area on the sound disk where your soundfiles (and, possibly, some related sound analysis files) are stored.

Both of these base directories can included any number of nested subdirectories. Users are encouraged to organize their soundfiles by creating subdirectories within their home soundfile directory, storing all soundfiles of a particular type, or all soundfiles used to create a section of a composition, within one of these subdirectories. If I create a soundfile subdirectory called *section1*, for example, this subdirectory has the path name */snd/allan/section1*. A similar tree-like directory structure, with several branches off the main "trunk," is employed to organize our Unix files.

Note that soundfiles should never be saved to your */home* directory or to any of its folders, as this will fill up the */home* partition quickly.

4.5.1. Commands to list and play your own (or another user's) soundfiles

☞ *Listing your soundfiles:* **lsf**

- To obtain a short listing of all of the soundfiles within your current working soundfile directory, type:

lsf

- To obtain a long listing of these soundfiles, type: **lsf -l**

[In all of the *listing* examples that follow, you can obtain a "long" rather than "short" listing by adding a **-l** flag to the command name.]

- To list only one, or a few, particular soundfiles, type:

lsf soundfilename(s)

Example: *lsf myvoice2.wav myvoice3.wav*
will list these two soundfiles if they exist in your current working soundfile directory. Note that *.wav, .aif* and *.aiff* extensions can not be omitted.

Unix wild card characters — chiefly ***** — are often employed with *lsf*:

> *lsf rain** will list all soundfiles in your current working soundfile directory that begin with the character string "rain."
>
> *lsf *vln* *vla** will list all soundfiles that include the character string *vln* or the character string *vla* anywhere within the file name.

To list soundfiles sorted by time of creation, from the most recent to the oldest, use the **-t** flag:

> *lsf -t* (list all files in your current working soundfile directory from most recent to oldest)
> lsf -lt myvoice* (all soundfiles whose names begin with the string *myvice* are listed, from newest to oldest)

- To obtain a listing of soundfiles in some subdirectory of your current working soundfile directory, type:

lsf subdirectory

Examples:

> (1) Typing *lsf section2* will provide a list of all soundfiles within your subdirectory *section2*.
> (2) *lsf section2/grunt** will list all soundfiles that begin with the character string *grunt* within subdirectory *section2*.

- To list soundfiles of your own, or of some other user, contained within a directory that does not branch from your current working soundfile directory, include the directory path on your *lsf* command line:

lsf directory_path

Examples: *lsf /snd/fred*
will provide you with a list of all soundfiles within user *fred* 's home soundfile directory.
*lsf /snd/fred/bird**
will list only those soundfiles in user *fred*'s home soundfile directory that begin with the character string

"bird"

To recursively list the contents of all of your soundfile folders in a single column, type:

lsf -R1 or else lsf -1R

☞ *Obtaining header information on user soundfiles :* **sfinfo** (**si**)

• To display the header information for one or more soundfiles within your current working soundfile directory, type:

sfinfo filename(s) or the abbreviation **si filename(s)**

For soundfiles whose names end with the extensions *.wav, .aif* or *.aiff*, the extension can be omitted from the *sfinfo* command line. Unix wild card characters such as ***** often are included within arguments to this command.
Examples: *sfinfo vln**
will display header information for all of your soundfiles whose names begin with the character string *vln*.
*si **
will display the header information for all soundfiles within your current working soundfile directory.
To obtain header information on soundfiles outside of your current working soundfile directory, include the full directory path name of the soundfile(s):

Example: *sndinfo /snd/fred/birdcall2*

For full information consult the *man* page for *sfinfo*.

☞ **Playing your soundfiles: the PLAY command**

To play one or more soundfiles in your current working directory from a shell window, use the **play** command, or its abbreviation **p** :

play filename(s) or the abbreviation **p filename(s)**

For soundfiles whose names end with the extensions *.wav, .aif* or *.aiff*, the extension can be omitted from the *play* command line. Thus, to play a soundfile called *groove2.wav* one could type either

play groove2.wav or else *p groove2*

You can play soundfiles outside of your current working soundfile directory by including the full path name of each soundfile.
Example: *play /snd/fred/birdcall2*

The general purpose ECMC *play* command[5] is actually a powerful shell script, written by Matt Barbour, that can play soundfiles in WAVE, AIFF, MP3, OGG and FLAC formats at any sampling rate supported by the system hardware, 16 or 24 bit ints or 32 bit floats, one, two, three or four channels, whether or not the *jack* audio server is running.[6] The *play* command also can be used to play tracks from an audio compact disc, to send its output to *jack* clients for further processing, and to decode and play soundfiles in ambisonic *B* format. Examples of these varied usages will be provided later in this *Users’ guide*. For a quick usage refresher type *play* with no arguments in a shell window. For more detailed information, see the *play man* page.

☞ The **findsnd** and **playlist** commands

The ECMC **findsnd** utility is similar to the *findsflib* script, but instead searches all of your own soundfile directories for soundfiles whose names include one or more character string arguments.

Example: *findsnd chain rattle*

will display a list of all of the soundfiles in all of your soundfile directories whose names include either the character string *chain* or else the string *rattle*. If we include a *-p* ("play") flag on the command line

Example: *findsnd -p chain rattle*

findsnd will first list the matching soundfiles and then play them.

[5] Before August 2006 the current ECMC *play* command was known as *jplay*.

[6] *jack* is discussed later, in section 4.9 of this *Users’ Guide*.

In addition to the *-p* option *findsnd* has several other fl;ag options. Consult the *man* page for details.

When you first begin work on *madking* you may have little use for this command. But after you have created many soundfiles, you may find it quite useful. The *findsnd* and *findsflib* commands also can be used in conjunction with a utility called **playlist**, which plays a list of soundfiles contained within an ascii file. Type *playlist* with no arguments, or for more complete information see the *playlist man* page.

Quick example:

```
findsnd -s pipe bell > newstuff
playlist newstuff
```

Result: All of your soundfiles whose names include the character strings *pipe* and *bell* are played, from lowest to highest pitched.

4.6. Playing soundfiles with graphical musical applications

Using the command *play* from a shell window is only one of several ways in which we can play soundfiles. The Linux, Macintosh and Windows user environments provide many GUI applications, some bundled with releases of the OS, others distributed by open source or shareware developers or computer music centers, for playing soundfiles, and additional applications, designed primarily for soundfile editing, processing or mixing, that also enable us to audition soundfiles.

Most graphical Linux applications can be opened in one of three ways:

(1) By typing the name of the application in lower case in a shell window. On ECMC Linux systems, this generally will open the application within your current working soundfile directory (which we often abbreviate *$SFDIR*), making it easier for you to locate and save your soundfiles.

Example 1: *audacity &*

Result: The *Audacity* soundfile editor will open. The " *&*" (ampersand) after the argument makes *Audacity* run in the background, so that you still will have the use of the shell window from which you submitted this command.

Example 2: *audacity /sflib/wind/clar.d3.wav &*

Result: *Audacity* will open with soundfile */sflib/wind/clar.d3.wav* loaded and ready to play.

(2) By clicking on a taskbar, desktop or menu launcher icon for the application.

Example: Select *Applications* on the taskbar, then highlight *Sound & Video* and then *Audacity*. This will not place you in your soundfile directory, but rather in your */home* Unix directory. You will need to navigate to your soundfile directory or to one of the *sflib* folders to locate soundfiles, and when go are ready to save an edited or otherwise altered version of the soundfile you will need tofind your way to your own soundfile directory.

Launcher icons for most graphical audio applications can be found by holding down the *Applications* menu tab on the taskbar, selecting either *Planet CCRMA* or else *Sound & Video* and then selecting the desired application.

(3) Another simple but ultimately rather limiting way to play soundfiles with graphical applications is to right click on an icon for the soundfile and highlight *Open with*. You will be presented with a list of graphical applications that can be used to play the soundfile. From this list, the simplest and quickest player application is the *Audio Player* (actually an application called *xmms*). Some of these graphical "player" applications enable you to create a playlist of several soundfiles. Others will play the soundfile once and then close.

Simple GUI Linux applications to play soundfiles, similar to the *Windows Media Player* and *Winamp* applications, are bundled with most Linux distributions. On *madking* these simple *player* applications can be found by clicking on *Applications->Sound & Video* in the taskbar. They also can be opened typing the application name (all lower case) in a shell window.

Simple audio player applications:

amarok	slow to open; can play 96k soundfiles; *jack* cannot be running
xmms (Audio Player)	cannot play 96k soundfiles; *jack* cannot be running
audacious	*jack* must be running; cannot play at 96k
rhythmbox	not recommended; cannot play 96k ; *jack* cannot be running
noatun	bare-boned player; cannot play 96k; *jack* cannot be running; not recommended

I am not enamored of any of these simple GUI soundfile players. They can be rather slow and cumbersome to use if you simply want to play several soundfiles from different directories in quick succession. On the other hand, several soundfile **editing** programs with high resolution waveform displays sometimes can be useful for auditioning soundfiles. I am hesitant to list features of these editing programs here, since these features constantly are changing. Among the recommended Linux soundfile editing programs are:

Soundfile editors:

rezound	reads and writes WAVE and AIFF; currently our recommended soundfile editor; usable with or without *jack* running
sweep	reads & writes WAVE & AIFF; excellent editor, but cannot be used with *jack* running
audacity	reads WAVE & AIFF,writes WAVE; useful general purpose soundfile editor and simple mixer; usable with or without *jack* running,but *Edit->Preferences->Audio I/O Playback* and *Recording* boxes must be toggled between *Alsa* and *jack*; annoyingly always asks if you want to save the soundfile even if it hasn’t been altered
snd	reads & writes WAVE & AIFF; powerful, but not easy to use

Finding icons for graphical music applications

If you will be using an application such as *rezound* or *audacity* frequently, and you prefer to open applications with mouse clicks rather than by typing their names in shell windows, you probably will find it handy to have an icon for the application on your taskbar for quick access. It also can save time to create quickly accessible icons for the *sflib* collections, and for your home soundfile directory on your desktop. In general application icons should be placed on your taskbar while folders ("places") should be placed on your desktop. Procedures for placing icons on your taskbar and on your desktop are covered at the end of section 2.1 of this *Users’ Guide*. Additionally, to find out the path to an application, type (in a shell window):

type application_name or else *which application_name*

(where *application_name* is the name of the application or program)

If this does not work, type: *locate application_name*

(If this display too many file names, type: *locate application_name | grep bin*)

In the resulting list look for the application binary. Most of our open source audio applications are located in the directory */usr/bin*, while software written at the ECMC generally is placed in */usr/local/bin*. The executable for the application *rezound*, for example, is */usr/bin/rezound*.

Beware of cluttering up your taskbar with too many icons or (especially) of turning your desktop into a Christmas tree with icons that frequently will need to be redrawn by the system, and often will be hidden by other windows anyway. And remember that while opening an audio application from a shell window will place you in your home *soundfile* directory (where you will want to save any files created with the application), opening the same application instead by clicking on its icon generally instead will place you in your home Unix directory — not where you want to be when working with audio.

4.7. Some useful soundfile ECMC utility programs

Several soundfile utility programs with a Unix-like command syntax (and names sometimes very similar to corresponding commands used on Unix files) are available to help you organize and manipulate your soundfiles. Many of these scripts (simple programs) and aliases were written at Eastman, while others date all the way back to IRCAM software written in the 1980s. Many of these scripts have mnemonic names in which the string *sf* or *snd* stands for "soundfile." Online *man* pages, also available in hardcopy within the *LINUX DOCs* binder are available for many of these commands, and most of them will display a usage summary if you type the command name with no arguments. All of the scripts and aliases listed below can read and write soundfiles at any sampling rate and bit depth supported by the system hardware (including 44.1k, 96k, 16 and 24 bit integers, 32 bit floats, and AIFF as well as WAVE format).

• **bounce** : mixes a stereo soundfile down to mono

Usage: *bounce inputsoundfile outputsoundfile [output_amp]*

Example: *bounce myvoice.wav myvoice.mono.wav*
Result: The stereo input soundfile *myvoice* is mixed down to mono, with an output amplitude peak of 32000, and the mono samples are written to a new soundfile called *myvoice.mono*. See the *bounce man* page for full details.

• **cpsf** : makes an exact copy of a soundfile (similar to the Unix *cp* command)

Usage: *cpsf sourcesoundfile newsoundfile*

Examples:
cpsf oboe.wav test2.wav
Result: Soundfile *oboe.wav* is copied to a new soundfile called *test2.wav*.

cpsf sflib/perc/bt.wav belltree.wav
Result: Soundfile *bt.wav* from the *sflib perc* directory is copied to a soundfile named *belltree.wav* in your own soundfile directory. (There is generally little reason to do this, unless you will be altering the belltree soundfile.)

cpsf /snd/fred/explosion.wav explosion.wav
Result: User *fred*'s soundfile *explosion.wav* is copied to a soundfile with the same name in your own soundfile directory.

Note: To copy a soundfile from any soundfile directory that you do not own, you must have *read* permission on the soundfile and *execute* permission on the source directory. Even so, don't do this without *fred*'s permission.

See also the *man* page for *cpsf.wav* and *cpsf.aif*, which make a WAVE format copy of an AIFF soundfile or vice versa.

• **mvsf** : change the name of a soundfile, or else move it to another directory (similar to the Unix *mv* command)

Usage: *mvsf oldsoundfilename newsoundfilename* (or else *mvsf soundfilename newdirectoryname*)

Example: *mvsf marimba4.wav marimbasolo.wav*
Result: Soundfile *marimba4.wav* is renamed *marimbasolo.wav*.

• **pitchshift** : transposes a soundfile

Usage: *pitchshift inputfile outputfile transp_factor*
or
pitchshift inputfile outputfile i intervallic_shift

Example: *pitchshift /sflib/env/bird.5.wav bird.5.dnoct.wav .5*
Result: /sflib/env soundfile *bird.5.wav* is transposed down an octave, and the samples are written to a soundfile called *bird.5.dnoct.wav* in the user's current working soundfile directory. See the *pitchshift man* page for full details.

• **rmsf** : deletes (**re**m*oves*) one or more soundfiles

Usage: *rmsf soundfilename(s)*

Examples:
rmsf oboe4.wav SEC1/marimbasolo.wav
Result: Soundfile *oboe4.wav*, and soundfile *marimbasolo.wav* from your soundfile subdirectory *SEC1*, are both deleted. (Bye.)

*rmsf oboe**

Result: All soundfiles within your current working soundfile directory that begin with the character string *oboe* are deleted. (Be very careful when using the wild card character ***** in combination with *rmsf* or any "remove" command.)

- **sfnorm** : normalizes mono or stereo soundfiles to maxamp or to any other desired peak amplitude

Usage: sfnorm [-p peak] infile [infile2 infileN]

Example: *sfnorm newflute explosion*

Result: Maxamp versions of input soundfiles *newflute* and *explosion* (or *newflute.wav* and *explosion.wav*) are created.

- **sfpeak** (**sndpeak**) : finds the peak amplitude within a soundfile

Usage: *sfpeak soundfilename*

Example: *sfpeak tenor2.wav*

Result: *sndpeak* displays the peak amplitude it finds within soundfile *tenor2.wav*

4.8. Making and using soundfile subdirectories

When you begin to accumulate many soundfiles, you may find it helpful to organized these soundfiles into subdirectories, rather than to store all of them in your home soundfile directory. The following commands will help you to do this:

- **mkdirsf** : makes a new soundfile subdirectory

Usage: *mkdirsf directoryname*

Example: *mkdirsf Beginning*

Result: A new subdirectory for your soundfiles is created and is named *Beginning*

- **mvsf** : Like the Unix *mv* command, this soundfile utility program can be used for two distinct purposes:

(1) changing the name of a soundfile; or
(2) moving one or more soundfiles into some other soundfile directory

The syntax used to move soundfiles into another directory is:

mvsf soundfile1 (soundfile2) (soundfile3) directoryname

Example:

mvsf marimbasolo.wav drums3.wav voice3.wav SEC2

Result: Soundfiles *marimbasolo.wav, drums3.wav* and *voice3.wav* are moved from your current working soundfile directory into your soundfile subdirectory *SEC2*.

- **cdsf** : changes your current working soundfile directory

Usage: *cdsf directoryname*

(If no directory name is supplied, *cdsf* moves you back into your home soundfile subdirectory.)

Example: *cdsf Beginning*

Result: You are moved into your soundfile subdirectory *Beginning*. Any soundfile utility commands you now issue, such as *lsf, mvsf, cpsf* or *rmsf*, will affect only soundfiles within this subdirectory.

cdsf sflib/perc

Result: You are placed in the *perc* directory on the *sflib* disk.

cdsf

Result: *cdsf* moves you back into your home soundfile directory.

pwdsf : displays your current working soundfile directory (similar to the Unix command *pwd*)

Example: *pwd*

Result: *pwdsf* tells you what soundfile directory you are in. This is called your *current working soundfile directory*, often abbreviated *$SFDIR* on ECMC systems.

- **rmdirsf** : removes an empty soundfile directory

Usage: *rmdirsf directoryname*

If the specified soundfile subdirectory is not empty — if it still includes one or more soundfiles — *rmdirsf* will warn you, and will refuse to delete the directory. You can delete all of the soundfiles within this directory, and then the soundfile subdirectory itself, in one of two ways:

(1) The slow but careful way:

cdsf directoryname	move into the soundfile subdirectory to be deleted
pwdsf	make *certain* that you are in the correct soundfile directory
lsf	list the soundfiles in this directory;
	if you wish to delete all of these soundfiles, type ...
*rmsf **	as always, be VERY careful when using the Unix ***** ("everything") symbol
cdsf	move back out of this soundfile subdirectory, then ...
rmdirsf directoryname	delete it

(2) The quick way: simply type *rm -rf /snd/USERID/directoryname*

This command will delete all of the soundfiles within the specified soundfile subdirectory, and then will delete the directory itself.

Section 4 of the *Users' Guide* concludes with two tutorials covering two important audio applications on ECMC Linux systems:

- *jack* : an audio server and software patchbay that connects multiple audio applications to the audio hardware, enabling us to run several audio applications simultaneously, and to route signals between these applications with sample accurate synchroniztion
- *pure data* (*pd*) : a graphical program, similar in some respects to *max*, for real-time sound synthesis and signal processing

---

4.9. JACK (The Jack Audio Connection Kit)

jack is a low-latency audio server, written initially for GNU/Linux systems but now available as well for Macintosh systems. *jack* performs many of the same functions that commercially developed digital audio plug-in formats and protocols such as Steinberg's *VST*, Apple's *Audio Units* and Digidesign's *RTAS* perform on Mac and Windows systems, although its implementation differs significantly. *jack* can connect a number of different *client* applications to an audio device, as well as allowing them to share audio between themselves. The *jack* server has been designed to be suitable for professional audio work, especially in two key areas: synchronous execution of all clients (audio is passed with sample accuracy from one client to the next), and low latency operation.

Some *jack* clients, like the *rezound* soundfile editor, can be run either as independent processes (ie. as normal applications) when *jack* is not running, or they can be run as clients ("plugins") within the *jack* server. Other *jack* clients, like the *jack-rack* suite of effects plug-ins, can only be used when *jack* is running. And some audio applications, such as the soundfile editor *sweep*, do not include the code necessary for interfacing with *jack*, and will produce an error message, and no sound, when you try to "play" them while *jack* is running. Conversely, you will not be able to start *jack* when another audio application is open, because the other application will have control of the audio hardware.

As an audio events server, *jack* runs on top of, and connects to, the Linux audio hardware driver (ALSA), and acts as a virtual "patchbay" for other client applications, passing sychronized audio data in real-time from one application to another. This means that the outputs of *Client A* (perhaps a soundfile editor) can be routed to the inputs of *Client B* (such as a delay line plug-in). In turn, the audio output of this delay line can be passed for further processing to *Client C*, then to *Client D*, and so on, all with exact synchronization. Alternatively, each of these clients might be producing sound independently and simultaneously, and we will hear a mix of all of these audio streams summed by *jack*. The only limits are processor speed and RAM capacity. Whenever a new client application is started, *jack* automatically detects all of its available inputs and outputs, which are referred to simply as *ports*. *jack* can be used not only to control the signal path between software applications, but also to route audio to and from available external hardware devices.

This server daemon model is ideal for real-time purposes. It offers a means of interconnecting several audio applications in customized configurations. As with other modular systems (like Unix/Linux itself), *jack* offers enourmous flexibility. In fact, many other modern real-time audio tools like *SuperCollider 3.0* have been or are being rewritten to benefit from a modular server/client design.

4.9.1. Some JACK clients

Once JACK has been started, several client applications may be started to talk with the daemon and to process an audio stream. There is a growing list of client applications that can be used with *jack*, and some day it is likely that most or all Linux applications will be "jackified. Applications shown in bold face have been found particularly useful by ECMC users:

- **ardour2** - a hardisk recording and mixing application in the style of *Nuendo* and *ProTools*
- **audacious** - a graphical WAVE soundfile and music CD player
- **audacity** - partially jackified soundfile editor/mixer
- **csound5** (our current version of *csound*) - powerful synthesis and signal processing language
- **ecasound** - a command-line player, recorder with support for multi-tracking plugins; *http://eca.cx/ecasound/*
- **freqtweak** - an FFT/frequency domain audio processing suite with EQ, pitch scaling, gating, delay line, limiter, compresser and other audio tools; *http://freqtweak.sourceforge.net/*
- *hydrogen* - a pattern based percussion sequencer/sampler
- *jack-mix* - a mixer for *jack* clients
- **jack-rack** - a stereo LADSPA (plugins) effects rack ; *http://jack-rack.sourceforge.net/*
- **mammut** - unique FFt-based sound transformation application
- *timemachine* - a quick one-button recorder for capturing JACK audio events into a soundfile; *http://plugin.org.uk/timemachine/*
- **jamin** - a stereo audio mastering tool that includes EQ, a 3-band compressor, a brickwall limiter and other audio processing tools; *http://jamin.sourceforge.net/en/about.html*
- **meterbridge** - provides several types of meters for monitoring signal levels on arbitrary JACK ports; *http://plugin.org.uk/meterbridge/*
- **qarecord** - simple, no frills recording utility with VU meters
- **rezound** - a powerful soundfile editor with LADSPA effects plug-ins
- **pd** - a powerful real-time music and multimedia environment
- **play** - the ECMC *play* command; (caveat: only works correctly when the soundfile sampling rate matches the *jack* sampling rate)
- **supercollider** - an environment and programming language for real time audio synthesis

For a longer, but still not complete, list of current *jack* client applications, click on the taskbar *Applications* tab and select *Planet CCRMA->Jack*.

4.9.2. Starting and configuring jack with qjackctl

When *jack* is running on *madking* it will have control of the *ALSA* Linux sound server, and thus also of the Delta 1010 audio interface, and you will only be able to play sounds through *jack*. This means that applications that do not support *jack*, such as the soundfile editor *sweep*, cannot be used during a *jack* session. Before attempting to start a *jack* session terminate any other audio applications or processes that may be running, so that *jack* will be able to establish a connection with *ALSA* and the audio interface.

The easiest way to start, stop and configure *jack* is with the graphical application **qjackctl**. Because of its importance and frequency of use, you should have an icon launcher for *qjackctl* on your taskbar. Before starting *jack*, click on the *Setup* box and check the default settings configuration. All *jack* clients that you open during this session will use these settings.

The most important setting is the *Sample Rate*, which defaults to 44100. Set the *Sample Rate* for the value you wish to use for the session. All *jack* clients will use this sampling rate, any soundfiles you create will be at this rate, and (in most cases) any input soundfile also will need to match this sampling rate or else it will not play. Note that changing the session sampling rate also will change the *Latency* time, indicated near the lower right corner of the *Setup* window. Latency is determined by the formula

Frames|Period / *Sample Rate* * *Periods|Buffer*

We want as low a latency time as possible, but without any audio *xruns*, in which the audio buffer is not refilled in time and a click or glitch results in the sound. The *Frames/Period* (buffer size) argument normally is set to 512 or 1024, although a lower setting may be necessary for good real-time performance if processing demands are not too great. The *Periods/Buffer* argument should normally be set to *2*. Note that the default settings of

Frames/Period 1024	*Sample Rate* 44100	*Periods/Buffer* 2

result in a latency of 46.4 milliseconds, which will be audible. If our *jack* session will include realtime audio or MIDI input, and realtime output, this slight but perceptible delay could be very undesirable. We often may wish to try lowering the *Frames/Period* value to 512, 256 or (with fast machines such as *madking* and audio processing that will not be too processor or graphics intensive) 128, and determine if we can still get glitch-free performance. Do not change the *Input Channels* or *Output Channels* default settings of 0 or *jack* will not start.

Other default settings that you normally should not change unless you know what you are doing include:

Server path should be *jackd*, not *jackstart*.
Realtime, H/W Monitor and *H/W Meter* should be checked. The other 5 boxes in the lefthand column, should be unchecked.
Priority should be set to *0* (the highest prioity value).
Port Maximum should be set to 128.
Interface should be *hw:0* (the *madking* Delta 1010)
Audio should be set to *Duplex*.
Input device and *Output device* should be set to *default* (the Delta 1010)

Timeout can be set to most any value, although 500 is recommended. In the unlikely event that one of more of your *jack* clients will require sample rate conversion, *Dither* can be changed from *None* to *Shaped* or to *Triangular*.
Input Latency and *Output Latency* should be set to 0.
Start Delay should be set to *1* or *2*. If **Verbose messages output** you will receive a continuous stream of diagnostic output from *jack*, which normally is unnecessary and can even be distracting. However, there are times that you may wish to turn *verbose* output on, because it provides continuous information on audio buffer activity. This information can be helpful if audio playback is not clean, or if you want to run a JACK configuration on a slower desktop computer perhaps on your home machine or on a laptop during a live performance. The verbose output tells you the following information listed at 1 second intervals:

```
System load:           Buffer usage:    Buffer remaining:
load = 0.0188 max      usecs: 8.000     spare = 21325.000
```

The total buffer is the addition of the buffer usage and the buffer space that remains available. As *Buffer usage* goes up, you will see its value subtracted from the *Buffer remaining*. If the real-time buffer usage exceeds the total of these two, you will see an *xrun*. You may need to adjust your audio buffer settings to accomodate increased activity, though this will increase latency.

If you make any changes in the default *jack* configuration, click on OK to apply the new settings and close the *Setup* box.

To start *jack* with these settings, simply click on the *Start* button in the *qjackctl* main panel. A *jack* window will open for the duration of the *jack* session, and in this window you will see if *jack* has successfully started or not. If not, this may be because some other application has control of ALSA and the audio interface, or else because a *jack* server is already running.[7] Type: *ps aux | grep jack*
and see if you see a *jack* server already loaded. If so, issue the command

jackstop or else *jackkill*

to try to terminate this process, which may have been orphaned by another user who forgot to stop *jack* before logging off. Sometimes such zombie processes can be difficult to terminate. Don't make this

[7] Sometimes, however, the problem is more complex, as when, for whatever reason, a kernel module required by *jack* has not been loaded. You will need help from a staff member to correct this type of problem.

mistake yourself. You can terminate *jack* at any time simply by clicking on the *Stop* button in the main *qjackctl* panel. Be sure to do this before logging off. If you experience problems, consult a staff member.

Throughout your *jack* session you will continue to control *jack* by means of the *qjackctl* panel. After starting *jack*, click on the *qjackctl* Connect to view the current output and input configuration. Make sure that the *Audio* tab is selected. You probably will want to keep the *Connections* window open, because this is where you will connect audio and MIDI *jack* client applications. You can toggle any *qjackctl* windows open or closed by clicking on the launcher box for that window in the main *qjackctl* panel.

In the *Connections* window, you will make "pathcord" connections between *jack* clients by dragging with the mouse from the *Output Port*(s) of one client, in the left column, to the *Input Port*(s) of another client in the right column. Initially, you should see an *alsa_pcm* client (the ALSA Linux sound driver, which connects to *madking*'s Delta 1010 audio and MIDI interface) in both columns. Click on the + box to the left of *alsa_pcm* in both the *Readable Clients* and the *Writable Clients* columns to see the available output and input ports (channels) from and to ALSA and the Delta 1010.

The first *capture* port in the left *Readable Clients* column represents the output from the Focusrite *Voicemaster* preamplifier. The first 4 *Output Channels* in the *Writable Clients* column represent the four Genelec loudspeakers.

4.9.3. A simple jack patch for recording microphone signals

Here is a tutorial *jack* patch that illustrates one way to record a microphone signal from the Focusrite preamplifier into a soundfile on the *madking snd* disk.

(1) Set up the *Focusrite* preamp: Adjust the input and master gain pots, and the controls for any other processing circuits (e.g. EQ) you wish to use on the Focusrite. Do a few practice runs until you are happy with the signal level and quality and are ready to record.

(2) Now we will create a *jack* patch to record this mic signal into a soundfile. Open *qjackctl*, then click on *Setup*. Set the sampling rate to the desired sampling rate for your recording, check that all of the other settings look correct and then click on OK (if you have changed the sampling rate or any other settings) or else on Cancel to close the *Settings* window.

(3) Open the *qarecord* application, which records audio into a soundfile. From a shell window type: *qarecord --jack*
or else, from the taskbar, select *Applications-->Planet CCRMA-->Recorders->Qarecord*.

(4) In the *qjackctl* main window click on *Connect* to open the *jack Connections* window. *qarecord* should appear in the righthand *Writable Clients* column.

(5) Click on the **+** next to *alsa_pcm* in the left *Readable Clients* column and the **+** next to *qarecord* in the right *Writable clients* column to display individual channel inputs and outputs. The *alsa_pcm capture_1* port represents the output of the *Focusrite* preamp. Click on this *capture_1* port, then on the *qarecord In_0* port, and then on the *Connect* button. This will route the Focusrite preamp signal to the left channel (or, in this case, mono) input of *qarecord*.

(6) Select a name for the soundfile you will record. In the *qarecord* window, click on *File->New*. In the *Save As* window that opens, navigate out of your home Unix directory to your home soundfile directory, if necessary, fill in a name for the soundfile in the *File name* box and click on *save*.

(7) Now do a few trial runs of the mic signal you want to record, watching the input meter in the *qarecord* window. Adjust the gain at the *Focusrite* preamp until you have a good, hot signal level at *qarecord*, but without ever clipping. This soundfile name now should appear in the *qarecord* window.

(7) When you are ready to record, click on the Record button in the *qarecord* window. You can Pause recording and then resume recording at any time. When you have finished recording, click on the *qarecord* *Stop* button.

(8) To find out the length of your recording, type *si* ("soundfile information") in a shell window. To play your recorded soundfile, type *play soundfilename.wav*, and watch what happens in the *jack Connections* window during playback. *jack* automatically connects the output of the *play* command to *ALSA* and the audio output playback system, then removes the *play* cient when *play* terminates.

(9) If you are not happy with the recording, make the necessary adjustments and record again to overwrite the soundfile. If you are happy with the recording you will need to trim it, removing dead time before and after the recorded signal. This can be done in any soundfile editor. Note, however, that if we had used *rezound* rather than *qarecord* to record the mic signal into a soundfile, the soundfile already would be loaded in *rezound* and we could trim the soundfile immediately and save this edited version to disk within *rezound*.

4.9.4. An example jack session using several clients

The following sample *jack* session will be used to illustrate some of the ways to interconnect Linux audio applications with *jack*.

(1) First, in the *Setup* window of *qjackctrl* we set up *jack* to run at 44.1 khz with one audio input channel and two audio output channels (stereo). Then we start *jack* and open the connections panel by clicking on the *Connect* box.

(2) *Adding signal level meters : meterbridge* : The client *meterbridge* enables us to add signal level meters to input or output ports. We want to put stereo meters at the end of the output signal chain so that we can monitor stereo output levels. In a shell window, type:

```
meterbridge     -t       dmp          1  2          &
           [type of meter]    [portname(s)]
```

The *-t* ("type of meter") argument of *dpm* specifies "digital peak meters," which I recommend. A *-t* argument of *vu* specifies analog *VU* meters (which will not show peak levels), and a *-t* argument of *ppm* specifies "peak program level" meters. A *-t* argument of *sco* will produce an oscilloscope meter. The number of meters created will depend upon the number of *portname* arguments; here we specify two meters (stereo monitoring).

Now in the *qjackctl Connect* window, you will notice that readable (output) and writable (input) ports have been added for the *bridge* (*meterbridge*) client, and that this client has been assigned a number, such as *bridge-4926*. This number serves to differentiate this client from any other additional *meterbridge* clients that we might add later, in case we wish to compare signal levels at two different points in our signal path.

☞ Click on the ⊞ next to *bridge-xxxx* in the *Readable Clients* (left) column to display both of its outputs.

☞ Then drag a patch-cord connection from the *monitor_1 meterbridge* output (left column) to the *alsa_pcm playback_1* input in the right (*Writable Clients*) column, and from the *monitor_2 meterbridge* output to the *playback_2* ALSA input.

(3) Add a soundfile input: *rezound*

Now let's open a soundfile for processing with *jack* clients. In a shell window, type:

```
rezound  /sflib/env/SThorses.wav &
```

rezound will open as a *jack* client, and should prompt you to specify to which *Input ports* its output should be routed.

☞ Route the left channel output of *rezound* to *meterbridge meter_1*, and the right channel output of *rezound* to *meterbridge meter_2*.

If you are not prompted to make these connections, you can create them easily enough in the *jack Connections* window. Now play the soundfile in *rezound*, and you should be able to visually monitor the left and right channel signal levels in the *meterbridge* meters. *jack* is routing the audio output from *rezound* into *meterbridge*, then routing the *meterbridge* outputs to ALSA and the Delta 1010 audio interface.

In the *rezound* waveform display window we could also select one or more alternate input soundfiles for playing and processing. From the *File* tab select *Open*, then navigate to */sflib/chinaperc* and select a mono soundfile such as *bangu.roll.wav*. Now in *rezound* we can toggle our source sound between the stereo *SThorses* and the mono *baguroll* soundfiles.

(4) Add some effects processing: *jack-rack*

We will use one of the effects plug-ins in the *jack-rack* suite to process our input soundfile. In a shell window, type:

```
jack-rack &
```

to open a *jack-rack* client. In the *jack Connections* window, we now need to insert *jack-rack* in between *rezound* and *meterbrdge*. This requires that we first break the existing connection from *rezound* to *meterbridge*.

☞ Highlight (left click on) both *rezound output_1* and on bridge meter_1, then right click and select *Disconnect*. Also disconnect *rezound output_2* and bridge meter_2.

☞ To connect the output of *rezound* to the input of *jack-rack*, highlight *rezound output_1* and *jack-rack in_1*, right click and select *Connect*.

Repeat this for the right channel ports of these two clients.

☞ Then connect *jack-rack out_1* to *bridge meter_1* and *jack-rack out_2* to *bridge meter_2* in the same manner.

Our complete *jack* "patch," from *rezound* through *jack-rack* to *meterbridge* and finally out to *ALSA*, now should look something like this table, which you might wish to read from bottom to top:

Readable Clients / Output Ports	(patch-cord connection)	Writable Clients / Input Ports
alsa_pcm	(output to Delta 1010 and Genelec speakers)	
bridge_xxxx		**alsa_pcm**
monitor_1	--------------->	*playback_1*
monitor_2	--------------->	*playback_2*
jack-rack_xxxx		**bridge_xxxx**
out_1	--------------->	*meter_1*
out_2	--------------->	*meter_2*
rezound		**jack-rack_xxxx**
output_1	--------------->	*in_1*
output_2	--------------->	*in_2*

Now we need to add one or more *jack-rack* effects. In the *jack-rack* window, click on *Add*, then select *Frequency->EQs->Multiband->Multiband EQ*. A 15 band graphic EQ window should open. Make some changes in the fader settings and click on the *Enable* box to route the signal through this equalizer. Play one of the input soundfiles in *rezound* and make some changes in realtime to the fader settings. You also can put *rezound* into loop mode (click on the "Play all looped" button next to the || "Pause" button) to play the soundfile continuously which trying different EQ settings. To bypass the equalization for A/B comparison, click on the *Enable* button in *jack-rack* so that it is no longer highlighted.

In *jack-rack* we could now open some additional effects by click on the *Add* button. We could try out several effects and only *enable* those that we like.

(5) Record our realtime processing into a soundfile: *Qarecord*

To capture the results of our audio processing we could add the recording application *qarecord* to the end of our *jack* "patch." In a shell window, type *qarecord -jack*. A writable client will appear for *qaRecord* in the *jack Connections* window. Drag a patch chord from *jack-rack output_1* to *qaRecord in_0* (left channel) and from *jack-rack output_2* to *qaRecord in_1* (right channel).

In the *qaRecord* window, check the Capture box. Now, when you play the source soundfile, its signal level(s) should appear in the *qaRecord* meters. When you are ready to record the soundfile, first specify a name and path for the output soundfile. Select *File->New*, navigate to the desired output soundfiles directory and fill in a name for the output soundfile. To begin recording, click on the button, then on the Stop button to end recording.

(6) In addition to adding more *jack* clients, we can remove an existing client at any time by making its window active and clicking on the x in the upper left corner of its window, or else be selecting *File-->Quit*. The client will disappear from the *jack Connections* window. You may need to reconfigure the patch-cord connections in this window. For example, to substitute a microphone input for the *rezound* soundfile input in our "patch" above, we would first

☞ quit the *rezound* application,

☞ then, in the *jack Connections* window, click on the + next to the *alsa_pcm* client in the (lefthand) *Readable Clients* column to display the *capture_1* port with the microphone icon, then

☞ select this *capture_1* port and (in the right column) the *jack-rack in_1* port, right click and select

Connect to make this connections. To connect the mic to both input channels of *jack_rack*, we would repeat this procedure for the *jack-rack in_2* port.

When you have completed a *jack* session it is important to terminate all *jack* clients and, most importantly, *jack* itself. Close each of of the *jack* clients you have opened, until the *jack Connections* window shows only *alsa_pcm* in both the *Readable Clients* and the *Writable Clients* colunmns. Then, in the main *qjackctl* panel, click on the Stop button to shut down *jack*. To be sure, type:

```
ps aux | grep jack
```

and make certain that *jack* is indeed dead. In the event of a hanging jackd process, you may need to run killall at a high level:

```
killall -9 jackd
```

4.10. A PD tutorial

PD "*P*ure *D*ata") is a powerful *object oriented* programming environment for processing and synthesizing audio and for generating and manipulating MIDI control data. *PD* provides us with one or more "canvases" (windows) in which we can create, edit and manipulate graphical *objects*, then save the resulting signal path as a "patch" file (or, les often, as a group of files). PD is distributed with several libraries that convert the graphical symbols in our patches into low level code and execute this code in real time. Additionally, there is a fairly large and active PD community that provides additional objects and libraries and helpful online user forums.

PD is an open source cross-platform application that can be installed in a few minutes on Linux, Windows and Macintosh systems. (As of this writing, however, the Mac version tends to lag slightly behind in the implementation of newer resources, and also tends to be somewhat buggier.) PD was written by Miller Puckette, who also wrote the original code for the similar *MAX/MSP* application that now is commercially distributed for Mac and Windows systems by the *Cycling74* company. Users familiar with *MAX/MSP* will quickly observe that the graphical interface of *PD* is built around an aesthetic of minimal "eye-candy". In fact, the graphical interface of PD has been described by some, (especially some MAX users) as "butt-ugly," but this graphical simplicity is by design, resulting in a very small memory "footprint" and resulting fast execution. It is even possible to run several instances of PD simultaneously and still achieve good performance. Instances of PD can easily be started and stopped as needed, even in live performance conditions.

The *DOCS* page of the ECMC web site includes links to *PD* sites, and rooms 52 and 53 in the ECMC studios include a hardcopy *PD Binder* into which we have placed tutorial and reference information on the application. The most important portion of this binder is Miller Puckette's *PD Documentation* article, which provides a brief but serviceable introduction to the application and which also is available as a *Help* resource while one is working within the application. In particular, you should read section 2 of this article, entitled *Theory of Operation*, which begins with an *overview* and includes subsections on *editing patches*, *messages*, *audio signals* and several other basic topics. Also included in this hardcopy binder are some example ECMC patches and some contributions by other PD users. In addition to the hardcopy *PD Binder*, take a look at the *ecmchelp PD* files.

PD is almost always run as a *jack* client. Thus, before launching *PD*, start *jack* with the sampling rate, the audio buffering *Frames/Period* value and the number of input and output channels that you wish to use. (256 is probably a good initial value for the *Frames/Period* buffer size, although you may need to adjust this upward for more complex patches.) Then open *PD* with the command

pd -jack & or else with the comparable alias *jpd &*

You will immediately be greeted by a small control window for PD. From this window the user can load patch files, turn audio processing on and off, with the *Compute Audio* button, and check for any data throughput errors (when the *DIO errors* box flashes red).

If you check the *qjackctl Connect* window after starting *pd*, you will see that by default the *alsa_pcm capture_1* port, representing the Focusrite *Voicemaster* preamp in the studio, has been connected as an input to PD, and the output from PD has been connected to the ALSA *playback* output(s) that send audio out to the Genelec speakers. Obviously it is possible to modify this default *jack* configuration so that PD receives audio input from other *jack* clients or passes its output to other clients for further processing.

There are two basic types of PD objects: (1) *audio* objects, which generate or process audio at the sampling rate, and (2) *control* (or "data") objects, which compute at a much slower rate (1000 times per second), and which create or manipulate values that are needed by audio objects, such as the frequency of an oscillator or the cutoff frequency of a low pass filter. The names of audio objects are always appended with a ˜ (tilde character) to indicate that they are part of the audio stream, and compute at the sampling rate. Objects whose names do not end with a ˜ are control objects.

The best way to learn PD is by playing and experimenting with example patches. PD's *Help* resources are very useful in this regard, and should be your starting point, to which you frequently return, in learning how to use *PD*. In the main PD window, click on the *Help* in the upper right corner. If you now click on *Html*, Firefox will open and load Miller Puckette's *Pd Documentation* article, which we have included in the hardcopy ECMC binder in the studio. Clicking on *Browser* opens a browser window with several folders:

- • If you select the *1. manual* folder and then scroll down in the righthand column and double click on *index.htm* you will once again open a copy of the *Pd Documentation* article. The *x1.htm* through *x5.htm* tabs display portions of the article.
- • The *2. control examples* directory contains tutorial documentation and sample patches to illustrate the use of *control* objects.
- • The *3. audio examples* directory contains tutorial documentation and sample patches to illustrate the use of *audio* objects.
- • The *5. reference* includes both tutorial patches that illustrate particular objects and patches that elucidate more advanced techniques or external libraries or objects that are not part of the core PD distribution.
- • The folders *4. data structures*, *6. externs* and *7. stuff* contain more advanced information and examples of interest primarily to experienced PD users.
- • The *ECMC examples* directory includes several sample patches, from simple to complex, submitted by ECMC users. Sample patches *mix4sf.pd*, *randmakenote.pd*, *readsf6sf.pd* and *selrandmakenote.pd* are tutorial-level patches.

You should begin your work with PD by examining the interactive models in the *2. control examples*, *3. audio examples* and the *ECMC examples* folders. Start by looking at three or four of the beginning *control* examples, which generally do not produce sound but which contain vital information on control objects and how to use them. Most of the examples include working patches that you can run, modify and run again. You may want to keep notes, somewhere, on which example patches and object *help* files you have examined, and on any example patches or objects that particualrly interest you, so that you will know where to begin work in your next PD session inthe studio.

PD has two modes: in *run* mode you can run a patch and receive output. In *edit* mode, you can modify and add to a patch, but generally cannot run it. You can toggle back and forth between *edit* and *run* modes by typing *Ctl-e*. In edit mode, clicking in an object or message box selects the entire box or block of comment text for editing, or enables you to drag it elsewhere on the canvas. Tapping on the Delete keyboard key will delete the highlighted selection. Standard editing procedures such as *Copy*, *Paste* and *Duplicate* also are available under the *Edit* tab. Dragging over a portion of a selection, such as a number, will select that "atom" for editing.

If you modify a patch and come up with a result that you like, you can save the edited version of the patch to your own directory by selecting *File-->Save as...*. PD patches (and PD *abstractions* — see below) must be saved with a *.pd* filename extension. Create your own Unix PD subdirectory for saving all of your PD work.

Right clicking on an object box and selecting *Help* will open another patch with information and a working example of that object. Additionally, near the bottom of many of the examples you will find *see also* links to related objects. You should feel free to follow such links. Using the interactive PD examples should be more like web browsing than like reading a book from cover to cover. Follow threads that interest you.

After you have explored a few of the *control* examples you may become increasingly desperate to hear some sound. After making of note of which *control examples* you have studied, move on to a few of the introductory *audio examples* and *ECMC examples*. Then, the next time you work with PD, go back to the *control* examples and pick up where you left off.

After reading the Puckette *Pure Documentation* article and working for a few hours with some of the introductory example PD patches, you should be sure that you understand the following concepts:

- an *object*, contained in an *object box*, performs mathematical or logical operations that generate or process an audio or a control signal
- *message box* : used to send a "message," such as a variable value, to an object
- *number box* : used to send or to display a number; dragging with the mouse up or down will increment or decrement the current number

(*Object*, *message* and *number* boxes, while all roughly rectangular, are distinguished by subtle differences in shape. An *object* box has four straight sides. A *message* box has a curved right side. A *number* box has a diagonal right top corner.)

- various other types of *GUI* boxes, such as graphical horizontal and vertical *sliders*, also are used to send numerical values to objects
- a *bang* is a trigger, or "start button;" clicking on a *bang* sends a signal to an object to cause it to perform its function
- *inlet* : an input to an object, message or number box, shown graphically as a thickened segment along the top of the box
- *outlet* : an output from an object, message or number box, shown graphically as a thickened segment along the bottom of the box

("Patchcord" connections are made by dragging with the mouse from the outlet of one box to an inlet of another box.)

- a *subpatch* is a subroutine, saved along with the main patch, that is displayed in another "canvas" (window) in order to keep the primary pitch from becoming too complicated or cluttered in appearance
- an *abstraction*, like a *subpatch*, contains a subroutine that is displayed in a separate window; unlike a subpatch, however, an abstraction is saved as a separate PD file, with a *.pd* filename extension, and can be inserted into any number of PD patches.

4.10.1. Creating PD patches from scratch

It is easiest to begin work with PD by modifying existing patches. At some point, however, you will need to be able to create a patch from scratch. Before doing this, mentally or on paper sketch out the algorithm (the step-by-step logic and sequence of operations) for at least a portion of this patch. Do not try to do too much all at once. Construct you patch in stages, testing it after making important additions or changes. Our patch will generate an oscillator signal and route this signal to ALSA for stereo playback over the Genelec speakers, and will illustrate some of the mechanics of creating and editing patches.

Start with a blank "canvas" by selecting *File-->New* from the main PD panel, A blank window will appear, ready to accept new objects. By default, when creating a new patch we are automatically placed in *Edit* mode so that we can immediately begin adding things to our patch, but will not be able to hear anything. From the *Put* menu in our blank canvas, choose *Object*. (The keyboard shortcut *Ctl-1* — holding down the *control* key and typing the number "one" — also can be used to "put," or create, an object.) An empty box will appear under the mouse cursor, and by clicking in the patch window we can place the object wherever we want it on the canvas. Inside this empty object box type:

osc˜ 440

and then click outside the box. This will create an oscillator object that will generate a cosine wave (a sine wave with a 90 degree phase shift). The *osc˜* object box has two inlets at the top and one outlet at the bottom, through which we can connect this oscillator with other objects. The left inlet is designed to receive a frequency argument, while any value sent to the right inlet will set the initial phase of the cosine wave. The "440" is called a "creation argument" or "initialization argument." It is a default frequency for the oscillator to produce, but will be overridden by any value received in the left "frequency" inlet. The outlet port in the lower left corner of the *osc˜* box will pass the results of this oscillator's calculations to other objects.

While working with the *osc˜* object, we might like to have *Help* information available for this object. Right click on this box and select *Help*. A *Help* window for *osc˜* will open. If we play around with this

Help patch it may begin to make sound. It is possible to have several PD patches open simultaneously and for two or more of these patches to be creating audio signals at once.

The simplest place to send our oscillator is directly to the Genelec speakers To do this we need to place the object *dac˜* at the end of our patch. One can specify the number of *dac˜* channels by listing them as arguments inside the object, such as *dac˜ 1 2 3 4*. (The number of channels is limited only by the number of output channels initiated when *jack* was started. The maximum number of channels currently is 10.) A *dac˜* object with no arguments will appears as a stereo object, with two inlets on top.

Use the *Put-->Object* (or *Ctl-1*) command to place an object box near the bottom of the canvas, below the *osc˜* box. Then type "dac" into this box. Two inlets should appear across the top of this box. To connect the output of our oscillator to the inputs of the *dac˜*, mouse over the oscillator's outlet until the mouse turns into a circle. Click, hold, and drag toward the left inlet of *dac˜*. The cursor turns into a patch cable. When the cursor is directly over this inlet and turns again into a circle, release the mouse to make the connection. Repeat this mouse procedure to connect *osc˜* output to the right channel of the *dac˜*. The monophonic output of *osc˜* now is connected to the inputs to both stereo loudspeakers. Signals from the outlet of an object can be connected to any number of inlets for other objects.

Now, without leaving *edit mode*, if you click in the compute audio box in the main PD window, PD should begin to compute samples and you should hear the oscillator tone. While still in edit mode, with the tone playing, click in the *osc˜* box, then drag over the *440* argument, change this to some other value, then click outside the]*osc˜* box. The pitch of the oscillator should change. Simple editing changes in a bare-boned patch such as this one can be made while PD is computing. However, you generally will want to use *run* mode while computing sound.

Obviously our patch is primitive and not yet musically useful. We need to introduce a better, quicker means of controlling the frequency of the oscillator, and a method to control its amplitude and amplitude envelope. To control the pitch we will use a vertical slider. Toggle into *edit* mode. From the *Put* menu select *Vslider* (or else type the keyboard shortcut *Alt-v*) and place the vertical slider above the osc˜ object. Right click inside the *vslider* object and select *Properties*. In the *VSLIDER-PROPERTIES* window that opens we can set the range of frequency values that the slider will output. In the *output-range* arguments near the top of this box, set the *bottom* argument to a value such as 50 or 100, and the *top* argument to a value such as 800. Just for kicks, lets color the slider green by clicking on one of the green color buttons. Then click on Apply and then on OK to apply these values.

Connect the outlet of the slider to the left (frequency) inlet of the oscillator. Let's also add a comment to explain the function of the slider. From the *Put* menu, select *Comment*, or else type *Ctl-5*. Place the "comment" next to the slider and release the mouse button. Select the "comment," drag over it, tap the *Delete* key to delete the word "comment," type in something like "Frequency control ->" and then click outside the comment. Your patch should now look something like the patch *userguideosc.pd* in the *ECMC examples* folder of PD's *Help*.

Now toggle into run mode, click on the compute audio box in the main PD panel to turn on the oscillator and move the slider up and down to change its frequency.

Our next improvement in this patch probably would be to add a method for time varying control of the oscillator signal. This would require

- breaking the patch cord connections between osc˜ and dac˜
- inserting one or more objects such as line˜ to control the amplitude; then
- connecting the output of osc˜ to the input of line˜ and the output of line˜ to the inputs of dec˜