Ecasound User's Guide

Kai Vehmanen

11122004

Contents

Chapter 1

Preface

This document describes Ecasound from the user's point of view. In addition to the actual user/client-programs, all essential Ecasound library concepts and features are also discussed. To avoid duplicating documentation, I've used references to other sources whenever suitable. For instance, Ecasound's man pages are a very good (and up-to-date!) source of information. They are also available in HTML-format.

If not otherwise specified, all documentation refers to the latest Ecasound version.

Chapter 2

Document history

- 11.12.2004 - Added section “Filenames with commas not handled correctly”
- 18.12.2003 - Many typo fixes and other corrections from Eric Rzewnicki.
- 18.11.2003 - Typo fixes.
- 20.08.2003 - Capitalize Ecasound in all cases where talking about the software package, not the console mode user-interface. Updated JACK documentation with a description of JACK and Ecasound states.
- 13.08.2003 - Updated documentation concerning JACK transport functions.
- 31.10.2002 - Few section layout bugs fixed.
- 30.10.2002 - Added JACK documentation, minor layout changes.
- 17.10.2002 - Updated Ecasound overview.
- 17.07.2002 - Added documentation for ecasignalview.
- 18.05.2002 - Fixed a few typos.
- 21.10.2001 - Added material from the Ecasound FAQ.
- 21.10.2001 - Added this history section. Document was restructured and all major chapters reviewed.
- 01.02.2001 - Updated the “Current position” section.

Chapter 3

Introduction

3.1 What is Ecasound?

Ecasound is a software package designed for multitrack audio processing. It can be used for simple tasks like audio playback, recording and format conversions, as well as for multitrack effect processing, mixing, recording and signal recycling. Ecasound supports a wide range of audio inputs, outputs and effect algorithms. Effects and audio objects can be combined in various ways, and their parameters can be controlled by operator objects like oscillators and MIDI-CCs. A versatile console mode user-interface is included in the package.

3.2 History

I've programmed Ecasound for my own recording purposes. The first versions ran under IBM Os/2. I used them for finalizing my analog 4-track recordings. My 4-track was a nice tool, but it had its limits. So eventually I started to use Windows-based multitrack software. I still used Ecasound for fx processing and finalizing. When I ported Ecasound to Linux, a lot of the code was rewritten from scratch. During this I also added multitrack capabilities to Ecasound. It took a lot of work, but in the end I was able to get rid of all my Windows recording software. Nowadays I use Ecasound for all my music projects. Because of this, I also continue to improve and develop Ecasound.

Chapter 4

Ecasound concepts

4.1 Audio object

Audio objects are used to transfer audio from and to Ecasound. Usually audio objects are either files (like wav, mp3 or ogg) or devices (soundcard input/output). There are also some special audio object types for transferring data between applications.

4.2 Chain

Chain is the central signal flow abstraction. In many ways chains are similar to audio cables. You have one input and one output to which you can connect audio producers and consumers (like guitar and amplifier for instance).

But there are some differences. First it's possible to attach chain operators (usually effects) to chains. This is somewhat like replacing one cable with two, and putting an effect box between them, but with chains it's just easier. A second important difference is that chains can transport multiple channels of audio. It's possible to attach mono, stereo or 24ch (or bigger) audio feeds to one chain. Also all chain operators can handle these multichannel streams.

In addition to chain operators, chains also have separate "mute" and "bypass" functions.

4.3 Chain operators and controllers

Chain operators are used to process and analyze sample data. They can be divided into gates, converters, signal analyzers and to traditional effects like reverbs, delays and filters.

It's also possible to attach special controller objects to chains. These controllers are used to control chain operator parameters. The typical examples

are various oscillators and MIDI continous controllers (knobs, sliders, etc found on MIDI-devices).

Both types of objects are attached to chains. The term *chain object* refers to all objects that can be attached to chains - ie. operators and controllers.

4.4 Chainsetup

Chainsetup is the central data object. All other objects (inputs, outputs, chains, etc) are connected to some chainsetup. Many chainsetups can exist at the same time (during one session), but only one of them can be in use. In Ecasound documentation, the term *connected* is used to describe a chainsetup that is in use.

Another important chainsetup concept is that of a *selected* chainsetup. If connected chainsetup is the one in use, selected chainsetup is the one that is currently edited.

Loading and saving chainsetups is the primary mechanism for storing and restoring state information. When saving to files, the *.ecs* file format is used. The file syntax uses the same notation as Ecasound's console (and command-line) interface. This makes it easy to edit the chainsetup files outside Ecasound, either manually or using external utils. See `ecasound(1)` man page for details.

4.5 Current position

Information about current position is only stored for audio objects and chainsetups. When you change chainsetup position, all audio objects are affected. On the other hand, positions of different audio objects can be changed independently.

4.6 Ecasound Control Interface - ECI

Ecasound Control Interface is an API for application developers who want to take advantage of libecasound in their own apps. See "Ecasound Control Interface Guide" and "Ecasound Programmer's Guide" for more information.

4.7 Ecasound Interactive Mode - EIAM

Most of Ecasound's functionality is located in one central library (libecasound). One thing that this library provides is a simple interpreter, which can be used for controlling Ecasound. This mode of operation is better known as Ecasound's interactive mode.

The most common frontend for EIAM is the console-mode Ecasound program. You can enter interactive mode by issuing "ecasound -c". For more detailed information about EIAM, see `ecasound-iam(1)` man page.

4.8 Ecasound Option Syntax - EOS

One very notable feature of the console-mode ecasound program is its command-line option syntax. You can do pretty much everything from the command-line.

But it doesn't end with the console-mode ecasound. In fact, interpreting these options is located in the main libecasound library, and is very closely tied to the interactive mode.

As a result, the same syntax (tokens that look like "-prefix:arg1,arg2,...,argN"), is used in various parts of libecasound. To name a few places:

- parsing command-line options
- the interactive-mode (as arguments to the 'cs-option' command [2.1dev4 and newer])
- saved chainsetup-files (.ecs format)
- effect preset definitions (see for example "/usr/local/share/ecasound/effect_presets")
- generic oscillator definitions (see for example "/usr/local/share/ecasound/generic_oscillators"

Chapter 5

Using

5.1 Where to start?

There's no one single way to use Ecasound. You can use it as a simple glue component for doing tasks that aren't handled by other applications you are using, or because Ecasound does these tasks more easily (or better even :)). But Ecasound can also serve as the centre of your studio setup, doing everything from effects processing to multitrack recording and mixing.

This flexibility doesn't come for free. It's difficult to describe Ecasound's features in a few phrases. Because of this, new users are encouraged to start from the `Examples` page - "Ecasound's web site -> Documentation -> Examples". It isn't a perfect introduction, and definitely not the only way to use Ecasound, but it does give an overall view of what can be done with Ecasound, and more importantly, it shows that most tasks are actually quite easy to do.

5.2 Rules for editing chainsetups

Here are a few rules that help writing valid chainsetups. Whether you are editing chainsetup files (.ecs), some graphical frontend, just using command-line options, etc; these rules always apply:

- Every chain has exactly _one_ input and _one_ output.
- All inputs and outputs must be connected to some chain.
- For every input/output, there is one and only one definition (example: "-i:file.wav").
- All routing from and to chains is based on selecting a set of chains and then specifying an input or output (example: "-a:1,2 -i:file.ext").
- All audio copying and mixing is done channel-wise. If you attach a 4-channel input and a two-channel output to a chain, that chain will have

4 channels of audio, but only the first two channels will be written to the output file.

5.3 Chain operators and controllers

The best place to start is to read through the `ecasound(1)` man page, which contains documentation for all native Ecasound chain objects.

5.4 Configuration

User preferences are stored in *~/.ecasound/ecasoundrc*. See the `ecasoundrc(5)` manual page for details.

By default, files for effect presets and oscillator presets are in */usr/local/share/ecasound*.

5.5 Common problems

5.5.1 I get occasional audio dropouts during operation? How to get rid of them?

Check "http://www.oreillynet.com/pub/a/linux/2000/11/17/low_latency.html" where you'll find a very good article written by Dave Phillips on Linux low-latency issues. If you are in a hurry (or desperate :)), here's a quick list of things to try:

- Tune your disks (see the article)
- Enable ecasound's double-buffering system by using the em(-z:db) option [note! this is only necessary with Ecasound 2.0.x and older]
- If you're still having problems, run ecasound as root (or with SUID-bit set) and use ecasound's em(-r) option. This will raise ecasound's scheduling priority to realtime (SCHED_FIFO). [with ecasound 2.1 and newer, just run ecasound as root and it will take care of tuning the settings]
- Try increasing ecasound's buffersize with the *-b:sample_frames* option. Something like *-b:4096* should do the trick.
- If all else fails, try the various low-latency kernel patches (again, check the article)

5.5.2 Can I use multiple soundcards?

This is possible, but there are some issues you should be aware of. If you try using multiple cheap soundcards to get more simultaneous inputs for recording, it's likely that the resulting streams will not be in sync. This problem is explained in detail in the Linux Audio-Quality HOWTO - "http://www.linuxdj.com/audio/quality/", section "Notes on Full Duplex Recording, and Other Realtime Issues".

5.5.3 Problems with panning mono files

In situations where you need to convert mono audio objects to multichannel objects, Ecasound can behave in a somewhat unexpected manner.

For instance, the correct way to set panning for three individual mono input files, and mix the resulting stereo output to soundcard, is:

```
ecasound -a:1 -i:monofile1.wav -erc:1,2 -epp:0 \
         -a:2 -i:monofile2.wav -erc:1,2 -epp:50 \
         -a:3 -i:monofile3.wav -erc:1,2 -epp:100 \
         -a:all -f:16,2,44100 -o:/dev/dsp
```

The actual signal chain is something like:

```
monofile1.wav |--'1'---- erc ----| epp |---\
                            \-----|     |---\\
                                          \\
monofile2.wav |--'2'---- erc ----| epp |------- | /dev/dsp
                            \-----|     |------- |ă
                                          //
monofile3.wav |--'3'---- erc ----| epp |---//
                            \-----|     |---/

('---' = mono channel)
```

The critical points to notice are:

- ecasound automatically notices that the three input files are mono files so chains are initialized with one mono input
- chains contain mono signal until -erc operator, which transforms the chain into a stereo chain by copying the data from ch1 to ch2
- now -epp works as expected (sets the stereo balance for one input)
- chains are mixed to the soundcard device channel-wise

If you leave out the -erc operators, chains will still be converted to stereo (as -epp is a stereo operator), but on each chain, only the first channel (left) will contain any audio from the input files.

5.5.4 Filenames with commas not handled correctly

There are some pitfalls in how commas in filenames are handled by ecasound. If you have a filename "foo,bar.ogg", the following will not work:

```
ecasound -i foo,bar.ogg -o alsa
```

The only way around this is to escape all the commas with backslashes:

```
ecasound -i foo\\,bar.ogg -o alsa
```

The backslash has to be a double-backslash as the shell strips one of the backslashes away before passing the string to ecasound.

Chapter 6

User interfaces and Applications

For a complete list of user-interfaces and applications built on top of Ecasound, visit Ecasound's web site at "http://www.eca.cx".

6.1 Ecasound

The standalone program "ecasound" is the primary user interface for Ecasound.

6.1.1 Further Reading

See `ecasound(1)` man page and `Examples` web page - "http://www.eca.cx/ecasound -> Documentation -> Examples"

6.2 Ecasignalview

Ecasignalview is an utility program for monitoring signal amplitude and peak statistics. It's primarily used when adjusting signal levels for recording.

6.2.1 Basic use

The basic use scenario is to record audio from a soundcard device, visualize it with vu-meters and write it to a null output.

```
# OSS-drivers (or properly installed ALSA OSS-emulation)
ecasignalview /dev/dsp null

# native ALSA-mode, recording from the 'default' device
ecasignalview alsa,default null
```

To monitor the input signal you can either use the soundcard's analog (or in some cases, digital) monitoring functions by enabling line/mic-in monitoring using *alsamixer* (ALSA), *aumix* (OSS) or some other mixer application. Another option is to use ecasignalview to do the monitoring. In this case the correct command is:

```
# OSS input and output
ecasignalview /dev/dsp /dev/dsp

# corresponding ALSA command
ecasignalview alsa,default alsa,default
```

Ecasignalview command-line options allow you to fine-tune the way monitoring is done:

```
# increased refresh rate 20Hz
ecasignalview -r:50 /dev/dsp null

# larger buffersize (1024 samples)
ecasignalview -b:1024 /dev/dsp null

# recording in mode 32bit/10channels/96000Hz with
# interleaved channels
ecasignalview -f:s32,10,96000,i /dev/dsp null
```

6.2.2 Monitoring Non-realtime Sources

FIXME: to be written...

6.2.3 Use with JACK

FIXME: to be written... See also **??**.

6.2.4 Further Reading

See `ecatools(1)` man page for a detailed listing of available command-line options.

6.3 Ecatools

See `ecatools(1)` man page.

Chapter 7

Advanced features

7.1 Audio loop devices

Just by using normal chain connections it's not possible to route audio from one Ecasound chain to another. One way around this limitation is loop devices. They were introduced in Ecasound 1.7.0.

7.1.1 Example of use

An example use-case where we route audio from chains "1" and "2" to chain "3" which is connected to a soundcard output.

```
--cut--
# note, the second loop parameter is the loop id-number;
# it is used to associate loop inputs with correct loop outputs
ecasound -a:1 -i:some.mp3 -o:loop,1
         -a:2 -i:another.mp3 -o:loop,1
         -a:3 -i:loop,1 -o /dev/dsp -ea:200
--cut--
```

Both inputs are eventually routed to chain "3", where a -ea:200 is applied to the signal. This does have one downside, loop device adds latency (-b:x -> latency of x frames).

7.2 Ecasound wave files - the .ewf format

7.2.1 General

Ecasound wave file (.ewf) is a simple wrapper format for controlling other audio objects. Ewf files are useful for offsetting or time-shifting audio files (for instance play a short audio clip in the middle of a long multitrack mix), for minimizing diskspace usage during multitrack recording (output offsetting) and looping.

7.2.2 File format

Ewf-files are stored in ascii format. The syntax is based on "key=value" pairs. The same syntax is used with Ecasound resource files. See `ecasoundrc(5)` man page for detailed info. Currently recognized ewf keywords are:

- source - audio object name [read,write]
- offset - insert audio object at offset (seconds) [read,write]
- start-position - start offset inside audio object (seconds) [read]
- length - how much of audio object data is used (seconds) [read]
- looping - whether to loop sample data (true or false) [read]

7.2.3 Example of ewf use

Let's look at a simple example .ewf file:

```
-- test.ewf --
source = test.wav
offset = 5.0
start-position = 2.0
length = 3.0
looping = true
--cut--
```

Now what happens when you issue "ecasound -i test.ewf -o /dev/dsp"? Because of the "offset" definition, the first 5 seconds will be silent. After that ecasound will start to read data from "test.wav". But as "start-position" is not zero, ecasound will skip the first 2 seconds. After 8 seconds has passed ("offset" + "length"), ecasound will loop back to "start-position". This looping will continue until the user interrupts the operation.

7.3 Effect presets

7.3.1 General

Ecasound has a powerful effect preset system that allows you to create new effects by combining basic effects and controllers.

Presets can be stored into separate files or they can be stored into a global database. Either way, the preset format is the same (also see `ecasoundrc(5)` man page, the same file format and syntax is used):

```
preset_name = effects controllers | ... | effects controllers
```

Effects and controllers are specified using the EOS syntax, the same syntax that is used for parsing command-line options ("-ea:100", "-kl:1,0,100,5", etc). The pipe character is used to separate parallel chains.

Just like in shell scripts, the '\' character can be used to spread definitions across multiple lines.

7.3.2 Example of preset use

Ecasound effect presets are in fact small Ecasound engines that behave just like native effects. Here's an example of a multi-chain effect preset:

```
-- file 'bassbooster.ecp' --
# let's put the low freqs into one chain and high freqs in another
bassbooster = -efl:2000 -ea:200 | -efh:2000 -ea:50
# note, the '|' sign separates parallel chains
--cut--
```

Once defined, you can use the preset in the following way:

```
--cut--
ecasound -a:1 -i:some.mp3 -pf:bassbooster.ecp
         -a:2 -i:another.mp3 -pf:bassbooster.ecp
         -a:1,2 -o:/dev/dsp
--cut--
```

When separate files are used (the "-pf:name" option), Ecasound always loads the first preset it finds. If the file contains more presets (additional "key=value" -pairs), they are ignored.

An alternative way to define presets is to put the definition in the global preset list (usually in "/usr/local/share/ecasound/effect_presets". Once you've added a line defining "bassbooster", you can use it like:

```
--cut--
ecasound -a:1 -i:some.mp3 -pn:bassbooster
         -a:2 -i:another.mp3 -pn:bassbooster
         -a:1,2 -o:/dev/dsp
--cut--
```

7.3.3 Preset parameters

FIXME: to be written...

7.3.4 Parameter descriptors

FIXME: to be written...

7.4 Gate operators

Gates are just like any other chain operators. They are assigned to a chain, and process passing audio data buffers. One special feature of gates is the ability to crop sections of audio files, for instance to achieve automatic volume-based cutting of audio streams:

7.4.1 Example of use

The following sequence cuts the section [60:00 sec -> 61:00 sec] from "guitar.wav" into "gate-test.wav":

```
--cut--
|\$ ls -la guitar.wav
-rw-rw-r--   1 kaiv     kaiv     15790124 Sep 30 23:27 guitar.wav

|\$ ecasound -i guitar.wav -o gate-test.wav -gc:60,1

|\$ ls -la gate-test.wav
-rw-rw-r--   1 kaiv     kaiv       180268 Dec 12 22:13 gate-test.wav
--cut--
```

The threshold gate is used similarly:

```
--cut--
|\$ ecasound -i gate-test.wav -o gate-test-rms.wav -ge:11.2,5,1

|\$ ecasound -i gate-test.wav -o gate-test-peak.wav -ge:5,5,0

|\$ ls -la gate*wav
-rw-rw-r--   1 kaiv     kaiv       163884 Dec 12 22:18 gate-test-peak.wav
-rw-rw-r--   1 kaiv     kaiv       143404 Dec 12 22:17 gate-test-rms.wav
-rw-rw-r--   1 kaiv     kaiv       180268 Dec 12 22:13 gate-test.wav
--cut--
```

In the first case, the gate is opened when the RMS-volume goes over the "11.2%" threshold, and closed when RMS-volume falls below "5%". In the second, case, both entry and close thresholds are "5%" (peak volume).

7.5 LADSPA plugins

Ecasound supports LADSPA-effect plugins (Linux Audio Developer's Simple Plugin API). See `ecasound(1)` man page and the LADSPA web site at "www.ladspa.org" for more information.

7.5.1 Ecasound is not able to find any LADSPA plugins I have installed!

Just installing the LADSPA SDK - "www.ladspa.org" - should be enough. The plugins themselves are stored in shared library files (.so). They are usually stored in "/usr/local/lib/ladspa". To test whether Ecasound finds the plugins, issue:

echo "ladspa-register" | ecasound -c

You should get a list of all installed LADSPA plugins. If this doesn't work, you need to make sure Ecasound is compiled with LADSPA enabled (ie. ladspa.h header was present when Ecasound was compiled). The precompiled rpm-binaries have this, but if you've compiled Ecasound yourself you should recompile after installing the LADSPA SDK.

Also, check Dave Phillips' great article on Oreillynet - "http://www.oreillynet.com/pub/a/linux/2001/02/02/

7.6 JACK Audio Server

JACK is a low-latency audio server, written primarily for the GNU/Linux operating system. It can connect a number of different applications to an audio device, as well as allowing them to share audio between themselves. Its clients can run in their own processes (ie. as normal applications), or they can run within the JACK server (ie. as a "plugin").

JACK is different from other audio server efforts in that it has been designed from the ground up to be suitable for professional audio work. This means that it focuses on two key areas: synchronous execution of all clients, and low latency operation.

If compiled with JACK support enabled (the "–with-jack" configure option), Ecasound can be used with JACK, both for input and output as well as transport control.

7.6.1 Basic Input and Output

Let's start with how to play a file using Ecasound and JACK:

```
ecasound -i foo.wav -o jack_alsa
```

This will create a separate JACK output port for each channel of foo.wav, and automatically connect these Ecasound ports to the ALSA PCM output ports in the JACK server. The connections are as follows:

```
ecasound:out_1    -->       alsa_pcm:playback_1
ecasound:out_2    -->       alsa_pcm:playback_2
```

To record a file, you'd issue:

```
ecasound -f:32,2,44100 -i jack_alsa -o foo.wav
```

Here we use "-f:bits,channels,srate" to set how many channels to record from the soundcard using JACK. As JACK always uses 32bit samples, and the sampling rate must match the one used by the JACK server, the channel count is the only configurable parameter. You can use another "-f:x,y,z" before "-o foo.wav" if you want to write the file in a different format.

7.6.2 More Advanced Port Creation

Ecasound also offers the following alternative ways to create input and output ports:

```
ecasound -i foo.wav -o jack
ecasound -i foo.wav -o jack_auto,remote_client
ecasound -i foo.wav -o jack_generic,local_portprefix
ecasound -i jack                          -o foo.wav
ecasound -i jack_auto,remote_client       -o foo.wav
ecasound -i jack_generic,local_portprefix -o foo.wav
```

"jack" and "jack_auto" both create a set of input (ecasound:in_X) and/or output (ecasound:out_X) ports. In the former case, Ecasound doesn't make any implicit connections, but in the latter case the created ports are automatically connected to the matching ports of the specified client (client:in_X or client:out_X).

If you want to create ports with a custom port name (ie. not "in_X" or "out_X"), you can use "jack_generic".

7.6.3 Transport Control

Transport controls are functions like "start", "stop", "seek", etc, that are commonly available in audio applications that maintain some kind of current position. JACK's transport control interface allows controlling the transport state of all the apps connected to one JACK server from a single application. Ecasound can support this functionality in four different modes ("notransport", "send", "recv" and "sendrecv").

By default Ecasound will both send and reveive transport events (position and state) to other JACK clients (mode "sendrecv"):

```
ecasound -c -i null -o jack
```

To use transport control in Ecasound, you have to have at least one published input or output JACK port. Here we publish one null output port. After giving the initial "engine-launch" command in Ecasound interactive mode, you are now able to use further EIAM commands to control all other JACK apps connected to the same server. Commands like "stop", "setpos 20", "rw 10", "fw 10", and so should affect other apps.

By default, Ecasound doesn't react to outside transport control. To enable this:

```
ecasound -c -i foo.wav -o jack_alsa -G:jack,eca_slave,recv
```

After giving an initial "engine-launch" to Ecasound, you should now be able to use other JACK apps to control Ecasound's playback of "foo.wav".

To combine external control with the ability to control the transport from ecasound's user-interface:

```
ecasound -c -i foo.wav -o jack_alsa -G:jack,eca_slave,sendrecv
```

7.6.4 JACK and Ecasound states

To have a good understanding of the overall system, it's important to understand how Ecasound and JACK states relate to each other.

When an Ecasound chainsetup is connected (EIAM-command "cs-connect"), a connection is established with the JACK server, and all the JACK ports in that chainsetup are registered to it. Once Ecasound's engine is launched with EIAM-command "engine-launch", connections (if any are specified) are made to the ports of other JACK clients. In this state Ecasound is ready to process incoming transport state and position changes.

When Ecasound processing is started (either with "start" or by an incoming transport event), Ecasound's engine runs as a node in the JACK system. When processing is stopped (either with "stop", or by a transport event), Ecasound's engine is not run.

Any connections (initiated by Ecasound) to other clients, are disconnected once "engine-halt" is issued and engine operation is stopped. Connection to the remote JACK server as well as unregistering any ports is performed when chainsetup is disconnected ("cs-disconnect").

Note! Normally you don't need to go through all the steps one by one. Instead issuing "start" will automatically connect the chainsetup and launch the engine. Similarly "cs-disconnect" will stop processing and halt the engine if needed.

7.6.5 Troubleshooting

Ecasound v2.2 and earlier don't have the capability to change the engine buffersize and sampling rate dynamically during processing. As a consequence, running Ecasound will fail if the currrent values for these parameters do not match the ones used by the JACK server. In other words, you have to correctly set the buffersize (with "-b:xxx") and sampling rate (with "-f:bits,channels,srate" and possibly using the *resample* audio object). This is the first thing to check if communication with JACK does not work.

Future versions of Ecasound will hopefully solve this problem. This issue is covered by Ecasound development item "edi-31 - Support for dynamic sampling rate and buffersize changes.".

Chapter 8

Miscellaneous

8.1 Security considerations when running with root privileges

When given the -r option (raise priority), Ecasound tries to raise its scheduling priority (to so called SCHED_FIFO realtime scheduling) and to avoid swapping, locks all its memory. To do this, root-privileges are required. So either Ecasound has to be run as root (logged in as root, or using the 'sudo' program), or it has to be installed with the suid-root bit set. Now is this a safe thing to do?

Although there are no known vulnerabilities, setting Ecasound suid-root is not safe. Whether this is a real problem depends on the particular setup (whether connected to network or not, any untrusted users with shell access, ...).

The basic problem is that Ecasound (or at least 2.0 and earlier) doesn't contain any code for altering privilege levels. If it is run with root-privileges, it does everything as root - including forking external programs such as mp3 and ogg utilities and editors.

But all in all, this shouldn't be that big of an issue. For noncritical uses, just don't set the suid-bit, but run as a normal user. If you have an untrusted setup, and you don't want to login as root, but still need to run in raised-priority mode, the following helps a bit (execute as root):

```
cd /usr/local/bin
chown root.ecausers ecasound
chmod 4750 ecasound
```

In other words, the ecasound binary is set as suid-root (so it is run with root-privileges), but only root and members of the 'ecausers' group can start it. Now just add all trusted Ecasound users to the group and you are set.

The ideal solution would be that ecasound would not need to be run with full root-privileges, only with privileges for changing scheduling and locking memory. Tommi Ilmonen's (author of the Mustajuuri softsynth) "givertcap" program

solves this very elegantly, but unfortunately it requires a custom kernel patch (at least for now). You can check out the program at: 'http://www.tml.hut.fi/ tilmonen/givertcap/'. Ecasound doesn't yet have explicit support for givertcap.