Table of Contents

Introduction 0

PART 1: SAMPLED AUDIO 1

- Basic concepts of digital sound 1.1
  - Resources 1.1.1
  - Sampled audio 1.1.2
  - Sample rate 1.1.3
  - Sample format 1.1.4
  - Frames 1.1.5
  - Pulse-code modulation 1.1.6
  - Over and Under Run 1.1.7
  - Latency 1.1.8
  - Jitter 1.1.9
  - Mixing 1.1.10
  - Conclusion 1.1.11
- User level sound tools 1.2
  - Players 1.2.1
  - Sound tools 1.2.2
  - Recorders, editors, etc 1.2.3

Conclusion 1.2.4
Sound codecs and file formats 1.3
Overview 1.3.1
PCM 1.3.2
WAV 1.3.3
MP3 1.3.4
Ogg Vorbis 1.3.5
WMA 1.3.6
Matroska 1.3.7
Conclusion 1.3.8
Overview of Linux sound architecture 1.4
Resources 1.4.1
Components 1.4.2
Conclusion 1.4.3
ALSA 1.5
Resources 1.5.1
User space tools 1.5.2
Programming ALSA 1.5.3
Mixing audio 1.5.4
Writing an ALSA device driver 1.5.5
Conclusion 1.5.6

Pulse Audio 1.6
Resources 1.6.1
Starting, stopping and pausing PulseAudio 1.6.2
User space tools 1.6.3
Programming with PulseAudio 1.6.4
Simple API 1.6.5
Asynchronous API 1.6.6
Conclusion 1.6.7
Jack 1.7
Introduction 1.7.1
Resources 1.7.2
Starting Jack 1.7.3
User tools 1.7.4
Applications using Jack 1.7.5
Using a different soundcard 1.7.6
How can I use multiple soundcards with JACK? 1.7.7
Mixing audio 1.7.8
Writing Audio Applications With JACK 1.7.9
Libraries 1.7.10
Port information 1.7.11
Copy input to output 1.7.12

Delaying audio 1.7.13
Audacity with Jack 1.7.14
Play a sine wave 1.7.15
Saving input to disk 1.7.16
Interacting with ALSA devices 1.7.17
Conclusion 1.7.18
Session Management 1.8
Resources 1.8.1
Session management issues 1.8.2
jack_connect 1.8.3
LASH 1.8.4
Jack sessions 1.8.5
LADISH 1.8.6
Non-session manager 1.8.7
Jack session API 1.8.8
LADISH API 1.8.9
Non-session management API 1.8.10
Conclusion 1.8.11
Java Sound 1.9
Introduction 1.9.1
Key Java Sound classes 1.9.2

Information about devices 1.9.3
Playing audio from a file 1.9.4
Recording audio to a file 1.9.5
Play microphone to speaker 1.9.6
Where does javaSound get its devices from? 1.9.7
Conclusion 1.9.8
GStreamer 1.10
Resources 1.10.1
libao 1.11
Resources 1.11.1
Examples 1.11.2
FFmpeg 1.12
Resources 1.12.1
FFmpeg command line tools 1.12.2
libavformat or libavdecode 1.12.3
OpenMAX and OpenSL 1.13
Resources 1.13.1
Quotes 1.13.2
OpenMAX AL and OpenSL ES 1.13.3
OpenMAX and Gstreamer 1.13.4
Linux 1.13.5

Raspberry Pi 1.13.6
Android 1.13.7
OpenSL ES examples 1.13.8
OpenMAX AL example 1.13.9
OpenMAX IL 1.13.10
Ogg Vorbis 1.13.11
Audio decoding 1.13.12
Conclusion 1.13.13
PART 2: DIGITAL SIGNAL PROCESSING 2
LADSPA 2.1
Resources 2.1.1
Files 2.1.2
Introduction 2.1.3
User level tools 2.1.4
The type LADSPA_Descriptor 2.1.5
Loading a plugin 2.1.6
The amp program 2.1.7
The analysePlugin client 2.1.8
A mono amplifier client 2.1.9
A stereo amplifer with GUI 2.1.10
Conclusion 2.1.11

PART 3: DIVERSIONS 3

Displaying video with overlays using Gtk and FFMpeg 3.1

Introduction 3.1.1

FFMpeg 3.1.2

Basic Gtk 3.1.3

Versions of Gtk 3.1.4

Displaying the video using Gtk 3.1.5

Overlaying an image on top of an image 3.1.6

Alpha channel 3.1.7

Using Cairo to draw on an image 3.1.8

Drawing text using Pango 3.1.9

Conclusion 3.1.10

Using the Raspberry Pi GPU with OpenGL 3.2

Resources 3.2.1

Accessing the GPU 3.2.2

Drawing a triangle using OpenGL ES 3.2.3

Drawing an image using OpenGL ES 3.2.4

Using the Raspberry Pi GPU with OpenMAX 3.3

Resources 3.3.1

Acknowledgements 3.3.2

OpenMAX 3.3.3

Compilation 3.3.4
State changes 3.3.5
The ilclient library 3.3.6
Creating a component 3.3.7
Decoding a JPEG image 3.3.8
Format of decoded image 3.3.9
Tunnelling 3.3.10
Decoding a JPEG image into RGB format 3.3.11
Rendering a JPEG image 3.3.12
Rendering a JPEG image without tunnelling 3.3.13
Conclusion 3.3.14
PART 4: MIDI 4
MIDI 4.1
Overview of MIDI 4.1.1
User level MIDI tools 4.2
Resources 4.2.1
Sound fonts 4.2.2
Timidity 4.2.3
Rosegarden 4.2.4
GStreamer 4.2.5
FluidSynth 4.2.6

Wild MIDI 4.2.7
Comparison 4.2.8
VLC 4.2.9
Conclusion 4.2.10
JavaSound 4.3
Introduction 4.3.1
Resources 4.3.2
Key JavaSound MIDI classes 4.3.3
Device Information 4.3.4
Dumping a MIDI file 4.3.5
Playing a MIDI file 4.3.6
Playing a file to an external MIDI synthesizer 4.3.7
Changing the soundbank 4.3.8
Changing pitch and speed 4.3.9
Using TiMidity instead of the default Gervill synthesizer 4.3.10
Conclusion 4.3.11
ALSA 4.4
Resources 4.4.1
Introduction 4.4.2
aconnect 4.4.3

seqdemo 4.4.4
aplaymidi 4.4.5
Raw MIDI ports 4.4.6
Conclusion 4.4.7
GStreamer 4.5
Resources 4.5.1
Example 4.5.2
FluidSynth 4.6
Resources 4.6.1
Players 4.6.2
Play MIDI files 4.6.3
Python 4.6.4
Timidity 4.7
Files 4.7.1
Timidity design 4.7.2
Making TiMidity into a library 4.7.3
Building a new interface 4.7.4
Summary 4.7.5
PART 5: KARAOKE 5
Overview 5.1
Overview 5.1.1

User Level Tools 5.2
Karaoke 5.2.1
Video CD systems 5.2.2
CD+G disks 5.2.3
MP3+G files 5.2.4
Buying CD+G or MP3+G files 5.2.5
Converting MP3+G to video files 5.2.6
MPEG 4 files 5.2.7
Karaoke machines 5.2.8
MIDI players 5.2.9
Finding MIDI files 5.2.10
KAR file format 5.2.11
pykaraoke 5.2.12
kmid 5.2.13
Microphone inputs and reverb effects 5.2.14
Conclusion 5.2.15
Playing MP3+G files 5.3
Files 5.3.1
Introduction 5.3.2
File organisation 5.3.3
Song information 5.3.4

Song table 5.3.5
Favourites 5.3.6
All favourites 5.3.7
Swing song table 5.3.8
Playing songs 5.3.9
VLC 5.3.10
Playing songs across the network 5.3.11
Conclusion 5.3.12
Decoding the DKD files on the Sonken Karaoke DVD 5.4
Introduction 5.4.1
Format shifting 5.4.2
Files on the DVD 5.4.3
Decoding DTSMUS20.DKD 5.4.4
The data files 5.4.5
Decoding MIDI files 5.4.6
Playing MIDI files 5.4.7
Playing WMA files 5.4.8
KAR format 5.4.9
Playing songs with pykar 5.4.10
Conclusion 5.4.11
Decoding the DKD files on the Malata Karaoke DVD

Resources 5.5.1 5.5
Files 5.5.2
Song titles 5.5.3
Song data 5.5.4
Linking songs in the song book to song data 5.5.5
Decoding song data 5.5.6
MIDI from data file 5.5.7
Conclusion 5.5.8
JavaSound 5.6
Resources 5.6.1
Introduction 5.6.2
KaraokePlayer 5.6.3
MidiPlayer 5.6.4
DisplayReceiver 5.6.5
MidiGUI 5.6.6
AttributedLyricPanel 5.6.7
PianoPanel 5.6.8
MelodyPanel 5.6.9
SequenceInformation 5.6.10
PinYin 5.6.11
Karaoke player with sampling 5.6.12

Comments on device choices 5.6.13
Performance 5.6.14
Downloads 5.6.15
Conclusion 5.6.16
Subtitles 5.7
Resources 5.7.1
Introduction 5.7.2
Subtitle formats 5.7.3
MPlayer 5.7.4
VLC 5.7.5
GStreamer 5.7.6
Gnome subtitles 5.7.7
SubStation Alpha 5.7.8
Karaoke effects in ASS files 5.7.9
Multi-line Karaoke 5.7.10
libass 5.7.11
Converting KAR files to MKV files with ASS subtitles 5.7.12
HTML5 subtitles 5.7.13
Conclusion 5.7.14
FluidSynth 5.8

Resources 5.8.1
Files 5.8.2
Players 5.8.3
Play MIDI files 5.8.4
Extending FluidSynth with callbacks 5.8.5
Displaying and colouring text with Gtk 5.8.6
Playing a background video with Gtk 5.8.7
Conclusion 5.8.8
Timidity 5.9
Files 5.9.1
Introduction 5.9.2
TiMidity and Jack 5.9.3
TiMidity interface 5.9.4
Adding an interface to TiMidity 5.9.5
Getting the list of lyrics 5.9.6
TiMidity options 5.9.7
Playing lyrics using Pnago + Cairo + Xlib 5.9.8
Playing a background video with Gtk 5.9.9
Background video with TiMidity as library 5.9.10
Background video with TiMidity as front-end 5.9.11
Adding microphone input 5.9.12

Conclusion 5.9.13
Jack 5.10
Using jack rack for effects 5.10.1
Playing MIDI 5.10.2
TiMidity plus Jack Rack 5.10.3
Customising TiMidity build 5.10.4
Playing MP3+G with Jack Rack pitch shifting 5.10.5
Conclusion 5.10.6
PART 6: STREAMING AUDIO 6
HTTP 6.1
HTTP servers 6.1.1
HTTP clients 6.1.2
Streaming vs downloading 6.1.3
Icecast 6.2
Icecast 6.2.1
IceS 6.2.2
DLNA 6.3
Resources 6.3.1
Introduction 6.3.2
DLNA open source projects 6.3.3
Flumotion 6.4

PART 7: MISCELLANEOUS 7
Android 7.1
Resources 7.1.1
Identifying devices 7.1.2
My Android experience 7.1.3
Playing files 7.1.4
Streaming audio 7.1.5
Recording audio 7.1.6
Playing audio from the microphone 7.1.7
MIDI playback 7.1.8
OpenMAX 7.1.9
Conclusion 7.1.10
Raspberry Pi 7.2
Resources 7.2.1
Introduction 7.2.2
No sound 7.2.3
ALSA 7.2.4
Sampled audio players 7.2.5
Sampled audio capture 7.2.6
MIDI players 7.2.7
JavaSound 7.2.8

PulseAudio 7.2.9
Java MIDI 7.2.10
OpenMAX 7.2.11
Conclusion 7.2.12
PART 8: CONCLUSION 8
Conclusion 8.1
Where did we start? 8.1.1
Where did I get to? 8.1.2
The Karaoke files 8.1.3
Communicating choices 8.1.4
The Java controller 8.1.5
Swing song table UI 8.1.6
The audio player side 8.1.7
Jack or PulseAudio? 8.1.8
Playing an MP3+G file 8.1.9
Playing a KAR file 8.1.10
TiMidity plugin 8.1.11
Conclusion 8.1.12

Programming and Using Linux Sound

Linux is a major operating system that can not only do what every other operating system can do, but can also do a lot more. But because of its size and complexity it can be hard to learn how to do any particular task. This is reinforced by its development model: anyone can develop new components, and indeed Linux relies on a huge army of paid and unpaid volunteers to do just that. But that can lead to confusion: if two methods are developed for one task, which one should be chosen? or more subtly, what are the distinguishing features of one solution that make it more appropriate for your problem?

The Linux sound system is a major example of this: there is a large variety of tools and approaches for almost every aspect of sound. This ranges from audio codecs, to audio players, to audio support both within and outside of the Linux kernel.

I've been using Linux since kernel 0.99. I'm not a kernel hacker, more of a user and a programmer at the application layer. But I've always got lost whenever I want to do something complex involving Linux sound. So I've decided out of sheer bravado to

try to describe the range of solutions to Linux sound issues. Of course, I'm way out of my depth, but that makes it a challenge, not a hindrance!

I'm going to rely on the people who really know what they are doing. So rather than re-invent everything from scratch I'm going to borrow the words of experts whenever I can. But the responsibility will still be mine, so please forgive and correct all the errors that I will make...

	Jan Newmarch is Academic Course Manager (ICT) at Box Hill Institute and Adjunct Professor Canberra University . His home page is at jan.newmarch.name . Jan has been writing books and papers for many years on all sorts of aspects of ICT: distributed programming, user interfaces, audio,Java, Go, GPUs, Raspberry Pi, X Window system, Wayland, ...

This version of the book is derived from HTML source at jan.newmarch.name/LinuxSound

If you like this book, please consider contributing using PayPal:	Or Flattr me:	Copyright © Jan Newmarch jan@newmarch.name

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

PART 1: SAMPLED AUDIO

This chapter looks at some basic concepts of audio, both analogue and digital.

Resources

- The Scientist and Engineer's Guide to Digital Signal Processing by Steven W Smith
- Music and Computers - A Theoretical and Historical Approach by Phil Burk, SoftSynth.com Larry Polansky, Department of Music, Dartmouth College Douglas Repetto, Computer Music Center, Columbia University Mary Roberts Dan Rockmore, Department of Mathematics, Dartmouth College

Sampled audio

Audio is an analogue phenomenon. Sounds are produced in all sorts of ways, through voice, instruments and natural events such as trees falling in forests (whether or not there is anyone to hear). Sounds received at a point can be plotted as amplitude against time and can assume almost any functional state, including discontuous.

The analysis of sound is frequently done by looking at its spectrum. Mathematically this is achieved by taking the Fourier transform, but the ear performs almost a similar transform just by the structure of the ear. "Pure" sounds heard by the ear correspond to simple sine waves, and harmonics correspond to sine waves which have a frequency a multiple of the base sine wave.

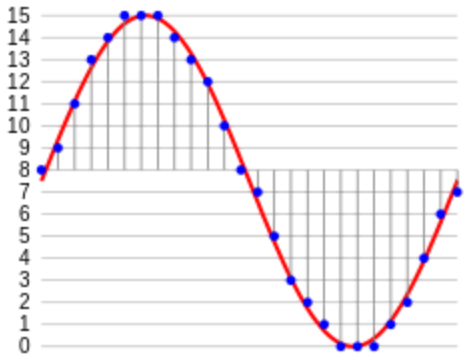
Analogue signals within a system such as an analogue audio amplifier are designed to work with these spectral signals. They try to produce an equal amplification across the audible spectrum.

Computers, and an increasingly large number of electronic devices, work on digital signals, comprised of bits of ones and zeroes. Bits are combined into bytes with 256 possible values, or 16 bit words with 65536 possible values, or even larger combinations such as 32 or 64 bit words.

Sample rate

Digitising an analogue signal means taking samples from that signal at regular intervals, and representing those samples on a discrete scale. The frequency of taking samples is the sample rate. For example, audio on a CD is sampled at 44,100hz, that is, 44,100 times each second. On a DVD, samples may be taken upto 192,000 times per second, with a sampling rate of 192kHz. Conversely, the standard telephone sampling rate is 8kHz.

This figure from Wikipedia: Pulse-code modulation illustrates sampling:



The sampling rate affects two major factors. Firstly, the higher the sampling rate, the larger the size of the data. All other things being equal, doubling the sample rate will double the data requirements. On the other hand the Nyquist-Shannon theorem places limits on the accuracy of sampling continous data: an

analogue signal can only be reconstructed from a digital signal (i.e. be distortion-free) if the highest frequency in the signal is less than one-half the sampling rate.

This is often where the arguments about the "quality" of vinyl versus CDs end up, as in Vinyl vs. CD myths refuse to die . With a sampling rate of 44.1kHz, frequencies in the original signal above 22.05kHz may not be reproduced accurately when converted back to analogue for a loudspeaker or headphones. Since the typical hearing range for humans is only upto 20,000hz (and mine is now down to about 10,000hz) then this should not be a significant problem. But some audiophiles claim to have amazingly sensitive ears...

Sample format

The sample format is the other major feature of digitizing audio: the number of bits used to discretize the sample. For example, telephone signals use 8kHz sampling rate and 8-bit resolution (see How Telephones Work) so that a telephone signal can only convey 2^8 i.e. 256 levels.

Most CDs and computer systems use 16-bit formats. (see Audacity - Digital Sampling) giving a very fine gradation of the signal and allowing a range of 96dB.

Frames

A frame holds all of the samples from one time instant. For a stereo device, each frame holds two samples, while for a five-speaker device, each frame would hold five samples.

Pulse-code modulation

Pulse-code modulation (PCM) is the standard form of representing a digitized analogue signal. From Wikipedia

> Pulse-code modulation (PCM) is a method used to digitally represent sampled analog signals. It is the standard form for digital audio in computers and various Blu-ray, DVD and Compact Disc formats, as well as other uses such as digital telephone systems. A PCM stream is a digital representation of an analog signal, in which the magnitude of the analog signal is sampled regularly at uniform intervals, with each sample being quantized to the nearest value within a range of digital steps.
>
> PCM streams have two basic properties that determine their fidelity to the original analog signal: the sampling rate, which is the number of times per second that samples are taken; and the bit depth, which determines the number of possible digital values that each sample can take.

However, even though this is the "standard", there are variations . The principal one concerns the representation as bytes in a word-based system: little-endian or big-endian . The next

variation is signed versus unsigned .

There are a number of other variations which are less important, such as whether the digitisation is linear or logarithmic. See the Multi-media Wikipedia for discussion of these.

Over and Under Run

From Introduction to Sound Programming with ALSA

> When a sound device is active, data is transferred continuously between the hardware and application buffers. In the case of data capture (recording), if the application does not read the data in the buffer rapidly enough, the circular buffer is overwritten with new data. The resulting data loss is known as overrun. During playback, if the application does not pass data into the buffer quickly enough, it becomes starved for data, resulting in an error called underrun.

Latency

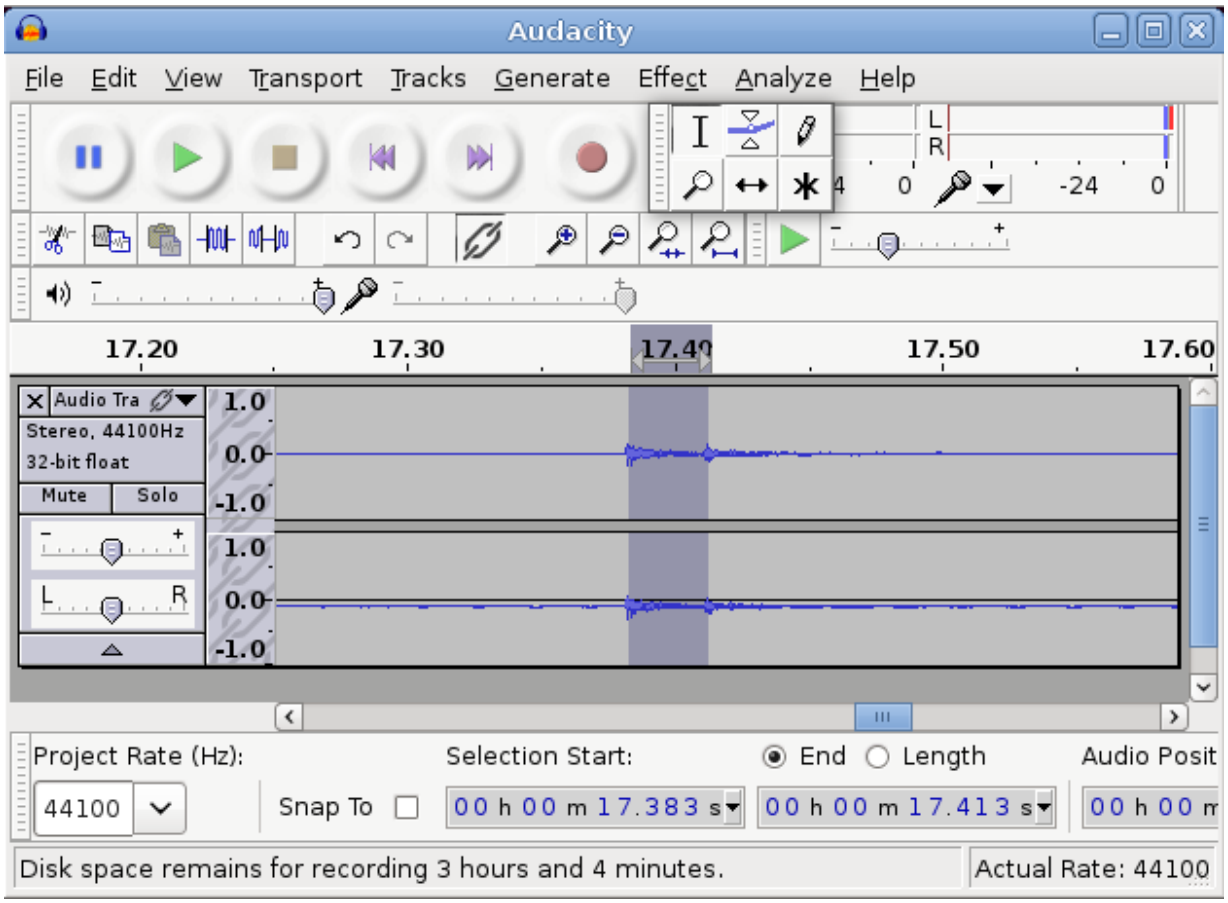
Latency is the amount of time that elapses from when a signal enters a system to when it (or its equivalent such as an amplified version) leaves the system.

From Ian Waugh's Fixing Audio Latency Part 1:

> Latency is a delay. It's most evident and problematic in computer-based music audio systems where it manifests as the delay between triggering a signal and hearing it. For example, pressing a key on your MIDI keyboard and hearing the sound play through your sound card.
>
> It's like a delayed reaction and if the delay is very large it becomes impossible to play anything in time because the sound you hear is always a little bit behind what you're playing which is very distracting.
>
> This delay does not have to be very large before it causes problems. Many people can work with a latency of about 40ms even though the delay is noticeable, although if you are playing pyrotechnic music lines it may be too long.
>
> The ideal latency is 0 but many people would be hard pushed to notice delays of less than 8-10ms and many people can work quite happily with a 20ms latency.

A Google search for "measuring audio latency" will turn up many sites. I use a crude - but simple - test. I installed Audacity on a separate PC, and used it to record simultaneously a sound I made and that sound when picked up and played back by the test PC. I banged a spoon against a bottle to get a sharp percursive sound. When magnified, the recorded sound showed two peaks

and selecting the region between the peaks showed me the latency in the selection start/end. In the figure below, these are 17.383 and 17.413 seconds, with a latency of 30 msecs.



Jitter

Sampling an analogue signal will be done at regular intervals. Ideally, playback should use exactly those same intervals. But, particularly in networked systems, the periods may not be regular. Any irregularity is known as jitter . I don't have a simple way of testing for jitter - I'm still stuck on latency as my major problem!

Mixing

Mixing means taking inputs from one or more sources, possibly doing some processing on these input signals and sending them to one or more outputs. The origin, of course, is in physical mixers which would act on analogue signals. In the digital world the same functions would be performed on digital signals.

A simple document describing analogues mixers is The Soundcraft Guide to Mixing . This covers the functions of

- Routing inputs to outputs
- Setting gain and output levels for different input and output signals
- Applying special effects such as reverb, delay and pitch shifting
- Mixing input signals to a common output
- Splitting an input signal into multiple outputs

Conclusion

This short chapter has introduced some of the basic concepts that will occupy much of the rest of this book. The Scientist and Engineer's Guide to Digital Signal Processing by Steven W Smith has wealth of further detail,

---

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

This chapter looks at the user-level tools that are typical under a Linux system, including players, various sound manipulation tools and editors.

Players

Mplayer

I think mplayer is fantastic and probably use it more than any other player. I usually run it from the command line but there are GUI versions available. It will play almost any media type, video as well as audio. I use it for both.

The man page for mplayer is MPlayer and the reference page is MPlayer - The Movie Player

VLC

VLC is my second favourite.

Its main page is VideoLAN and some documentation is at Welcome to VideoLAN's Documentation

Totem

Very commonly used

Sound tools

sox

SoX is the Swiss Army knife of sound processing programs.

Its home page is SoX - Sound eXchange

FFmpeg/avconv

ffmpeg is generally used as a converter from one format to another. There is a nice series of tutorials at A FFmpeg Tutorial For Beginners by shredder12

It can also be used to record from ALSA devices such as hw:0 or the default device. Recording from hw:0 can be done by

```
ffmpeg -f alsa -i hw:0 test.mp3
```

and from the default ALSA input by

```
ffmpeg -f alsa -i default test.mp3
```

There was a fork some years ago of ffmpeg to give avconv. avconv is the default on Ubuntu systems. There are some differences between the two, not enough to justify the nuisance factor to us users.

GStreamer

GStreamer allows you to build "pipelines" that can be played using `gst-launch`. For example, to play an MP3 file using ALSA you would have the pipeline

```
gst-launch filesrc location="concept.mp3" !
decodebin ! alsasink
```

The pipelines can do more complex tasks such as format conversion, mixing, etc. A tutorial is Multipurpose multimedia processing with GStreamer by Maciej Katafiasz

It can also play MIDI files by e.g.

```
gst-launch filesrc location="rehab.mid" !
decodebin ! alsasink
```

Recorders, editors, etc

Audacity

"Audacity is a free, easy-to-use and multilingual audio editor and recorder for Windows, Mac OS X, GNU/Linux and other operating systems" This is a fantastic tool and well worth using.

Its home page is Audacity

Conclusion

This short chapter has looked at some of the user level tools available under Linux. While we have listed several of the major tools, a casual search will turn up far more.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

There are many different ways of representing sound data. Some of these involve compressing the data, which may or may not lose information. Data may be stored in the file system or transmitted across the network, and this raises additional issues. This chapter considers the major sound codecs and container formats.

Overview

Audio and video data needs to be represented in digital format to be used by a computer. Audio and video data contain an enormous amount of information, and so digital representations of this data can occupy huge amounts of space. Consequently, computer scientists have developed many different ways of representing this information, sometimes in ways that preserve all of the information (lossless) and sometimes in ways that lose information (lossy).

Each way of representing the information digitally is known as a *codec* . The simplest way, described in the next section, is to represent it as "raw" pulse-code modulated data (PCM). Hardware devices such as sound cards can deal with PCM data directly, but PCM data can use a lot of space.

Most codecs will attempt to reduce the memory requirements of PCM data by *encoding* it to another form, called encoded data. It can then be *decoded* back to PCM form when required. Depending on the codec algorithms, the re-generated PCM may have the same information content as the original PCM data (lossless) or may contain less information (lossless).

Encoded audio data may or may not contain information about the properties of the data. This information may be about the original PCM data such as the number of channels (mono, stereo), the sampling rate, the number of bits in the sample, etc. Or it may be information about the encoding process itself, such as the size of framed data. The encoded data along with this additional information may be stored in a file, transmitted across the network, etc. if this is done, the encoded data plus the additional information is amalgamated into a *container* .

It is important at times to know whether you are dealing with just the encoded data, or with a container that holds this data. For example, files on disk will normally be containers, holding additional information along with the encoded data. But audio data manipulation libaries will typically deal with the encoded data itself, after the additional data has been removed.

PCM

From Wikipedia

> Pulse-code modulation (PCM) is a method used to digitally represent sampled analog signals. It is the standard form for digital audio in computers and various Blu-ray, DVD and Compact Disc formats, as well as other uses such as digital telephone systems. A PCM stream is a digital representation of an analog signal, in which the magnitude of the analog signal is sampled regularly at uniform intervals, with each sample being quantized to the nearest value within a range of digital steps.
>
> PCM streams have two basic properties that determine their fidelity to the original analog signal: the sampling rate, which is the number of times per second that samples are taken; and the bit depth, which determines the number of possible digital values that each sample can take.

PCM data can be stored in files as "raw" data. In this case there is no header information to say what the sampling rate and bit depth are. Many tools such as `sox` use the file extension to determine these properties. From man soxformat :

> f32 and f64 indicate files encoded as 32 and 64-bit (IEEE single and double precision) floating point PCM respectively; s8, s16, s24, and s32 indicate 8, 16, 24, and 32-bit signed integer PCM respectively; u8, u16, u24, and u32 indicate 8, 16, 24, and 32-bit unsigned integer PCM respectively

But it should be noted that the file extension is only an aid to understanding some of the PCM codec parameters and how it is stored in the file.

WAV

WAV is a file format wrapper around audio data as a container. The audio data is often PCM. The file format is based on RIFF (Resource Interchange File Format). While it is a Microsoft/IBM format, it does not seem to be encumbered by patents.

A good description of the format is given by Topherlee . The WAV file header contains information about the PCM codec and also about how it is stored (e.g. little- or big-endian).

Because they usually contain uncompressed audio data, WAV files are often huge, around 50Mbytes for a 3 minute song.

MP3

The MP3 and related formats are covered by a patent. Actually, a whole lot of patents. For using an encoder or decoder, users should pay a license fee to an organisation such as the Fraunhofer Society. Most casual users neither do this nor are aware that they should, but it is reported by Wikipedia that the society earned €100,000,000 in revenue for the society in 2005. The Society has at present chosen not to pursue free open source implementations of encoders and decoders for royalties.

The codec used by MP3 is the MPEG-1 Audio Layer III audio compression format. This includes a header component which gives all the additional information about the data and the compression algorithm. There is no need for a separate container format.

Ogg Vorbis

Ogg Vorbis is one of the "good guys". From Vorbis.com "Ogg Vorbis is a completely open, patent-free, professional audio encoding and streaming technology with all the benefits of Open Source"

The names are described as

> Ogg: Ogg is the name of Xiph.org's container format for audio, video, and metadata. This puts the stream data into *frames* which are easier to manage in files other things.
>
> Vorbis: Vorbis is the name of a specific audio compression scheme that's designed to be contained in Ogg. Note that other formats are capable of being embedded in Ogg such as FLAC and Speex.

The extension .oga is preferred for Ogg audio files, although .ogg was previously used.

At times it is necessary to be closely aware of the distinction between Ogg and Vorbis. For example, OpenMAX IL has a number of standard audio components including one to decode various codecs. The LIM component with role "audio decoder

ogg" can decode Vorbis streams. But even though the component includes the name "ogg", it *cannot* decode Ogg files - they are the containers of Vorbis streams and it can *only* decode the Vorbis stream. To decode an Ogg file requires use of a different component, an "audio decoder with framing".

WMA

From the standpoint of Open Source, WMA files are evil. WMA files are based on two Microsoft proprietary formats. The first is the Advanced Systems Format (ASF) file format which describes the "container" for the music data. The second is the codec, Windows Media Audio 9.

The ASF is the primary problem. Microsoft have a published specification . This specification is strongly antagonistic to anything open source. The license states that if you build an implementation based on that specification then you:

> - cannot distribute the source code
> - can only distribute the object code
> - cannot distribute the object code except as part of a "Solution" i.e. libraries seem to be banned
> - cannot distribute your object code for no charge
> - cannot set your license to allow derivative works

And what's more, you are not allowed to begin any new implementation after January 1, 2012 - and (at the time of writing) it is already July, 2012!

Just to make it a little worse, Microsoft have Patent 6041345 "Active stream format for holding multiple media streams" filed in the US on March 7, 1997. The patent appears to cover the same ground as many other such formats which were in existence at the time, so the standing of this patent (were it to be challenged) is not clear. However, it has been used to block the GPL-licensed project VirtualDub from supporting ASF. The status of patenting a file format is a little suspect anyway, but may become a little clearer now that Oracle has lost its claim to patent the Java API.

The FFmpeg project has nevertheless done a clean-room implementation of ASF, reverse-engineering the file format and not using the ASF specification at all. It has also reverse-engineered the WMA codec. This allows players such as mplayer and VLC to play ASF/WMA files. FFmpeg itself can also convert from ASF/WMA to better formats such as Ogg Vorbis.

There is no Java handler for WMA files, and given the license there is unlikely to be one unless it is a native-code one based on FFmpeg.

Matroska

From the Matroska web site

> Matroska aims to become THE standard of multimedia container formats. It was derived from a project called MCF, but differentiates from it significantly because it is based on EBML (Extensible Binary Meta Language), a binary derivative of XM. It incorporates features you would expect from a modern container format, like:
>
> - Fast seeking in the file
> - Chapter entries
> - Full metadata (tags) support
> - Selectable subtitle/audio/video streams
> - Modularly expandable
> - Error resilience (can recover playback even when the stream is damaged)
> - Streamable over the internet and local networks (HTTP, CIFS, FTP, etc)
> - Menus (like DVDs have)

I hadn't come across it until I started looking at subtitles which can be (optionally) added to videos, where it seems to be one of the major formats.

A GUI tool to create and manage MKV files is mkvmerge, in the Ubuntu repositories. the command mmg is a GUI version. mplayer and vlc will play them happily. There is a list of recognised formats.

Conclusion

There are many codecs for sound, and more are being devised all the time. They vary between being codecs, containers or both, and come with a variety of features, some with encumbrances such as patents.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

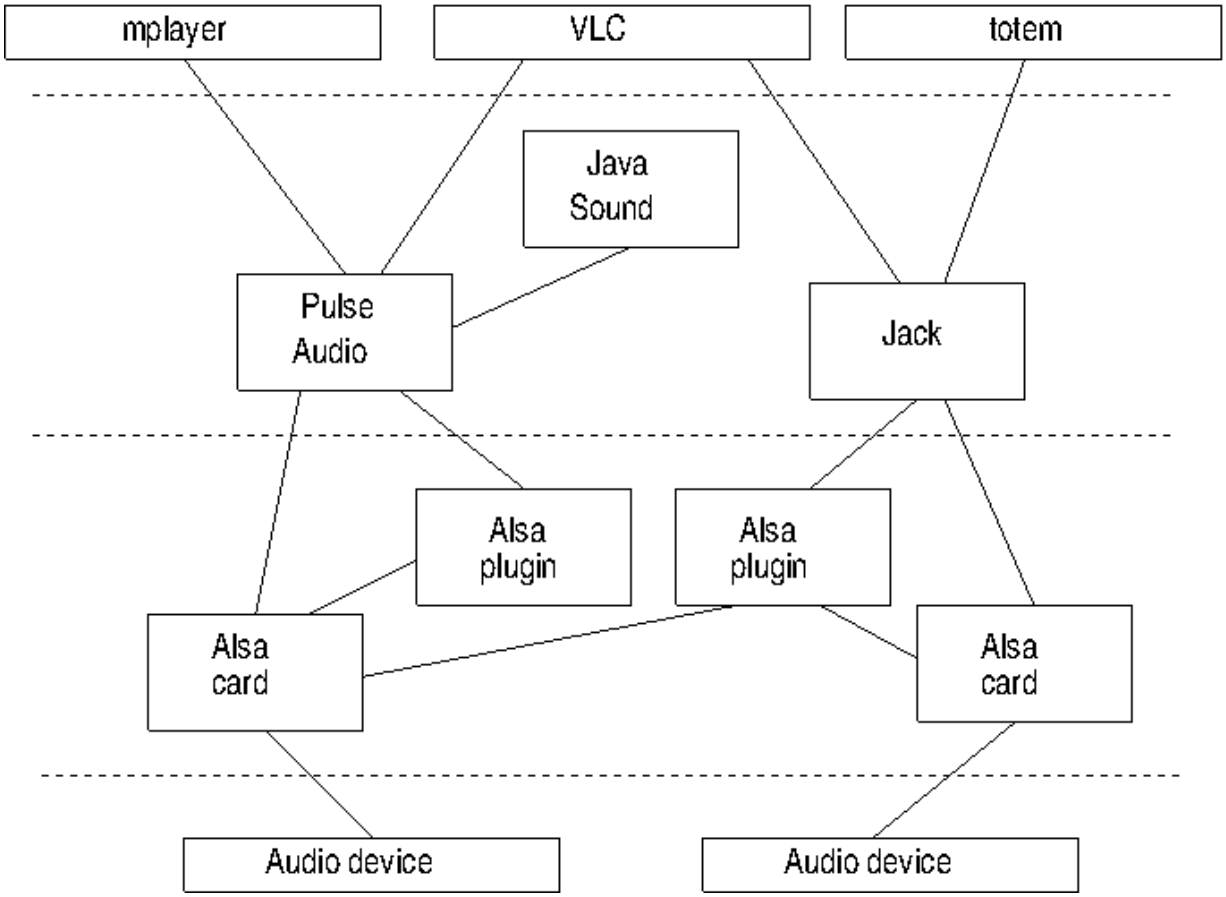
The Linux sound system, like most of Linux, has evolved from a simple system to a much more complex one. This chapter gives a high level overview of the components of the Linux sound system, and which bits are best used for which use cases.

Resources

- Lennart Poettering has A Guide Through The Linux Sound API Jungle
- Also, by TuxRadar: How it works: Linux audio explained
- Insane Coder posted an article in favour of OSSv4 State of sound in Linux not so sorry after all which drew a lot of comments...

Components

The following diagram indicates the different layers:



Device drivers

At the bottom layer is the hardware itself, the audio device. These are the audio cards made by a variety of manufacturers, all with different capabilities, interfaces and prices. Just like any

piece of hardware, in order for it to be visible and useful to the Operating System there must be a device driver. There are of courses thousands of device drivers written for Linux. Writing Linux device drivers is a speciality of itself, and there are dedicated sources for this, such as Linux Device Drivers, Third Edition by Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman.

Device drivers must have standardised APIs "at the top" so that users of the device have a known interface to code to. The OSS device driver API was used for audio devices until it was made closed source, at which point developers switched to the ALSA API. While OSS v4 has become open again, the ALSA interface is supported in the kernel while OSS is not.

Ideally, a device driver API should expose all of the features of hardware while not adding additional baggage. For audio, it is not always so easy to set boundaries for what an audio driver should do. For example, some sound cards will support mixing of analog signals from different sources, while others will not, and some sound cards will have MIDI synthesizers while others will not. If the API is to expose these capabilities for sound cards which support them, then it might have to supply them in software for those sound cards which do not.

There is a somewhat dated document on writing ALSA device drivers by Takashi Iwai

Sound servers

Linux is a multi-tasking, multi-threaded operating system. It is possible that concurrent processes might wish to write sounds to the audio cards concurrently. For example, a mail reader might wish to "ding" the user to report new mail, even if they are in the middle of a noisy computer game. This is distinct from sound card capabilities of being able to mix sounds from different ports, such as an HDMI input port and an analog input port. It requires the ability to mix (or otherwise manage) sounds from different processes. As an example of the subtlety of this, should the volume of each process be individually controllable, or should the destination port (headphones or speaker) be individually controllable?

Such capabilities are beyond the scope of a device driver. Linux resolves this by having "sound servers" which run above the device drivers and manage these more complex tasks. Above these will sit applications which talk to the sound server which in turn will pass the resultant digital signal to the device driver.

Here is where a significant difference occurs between sound servers. For professional audio systems the sound server must be able to process and route audio with the minimal amount of latency or other negative effects. For consumer audio, control over volumes and destinations may be more important than altency - you probably won't care if a new message "ding" takes an extra half-second. Between these may be other cases such as games requiring synchronisation of audio and visual effects, and Karaoke players requiring synchronisation of analog and digital sources.

The two major sound servers under Linux are Jack for professional audio and PulseAudio for consumer systems. They are designed for different use-cases and consequently offer different features.

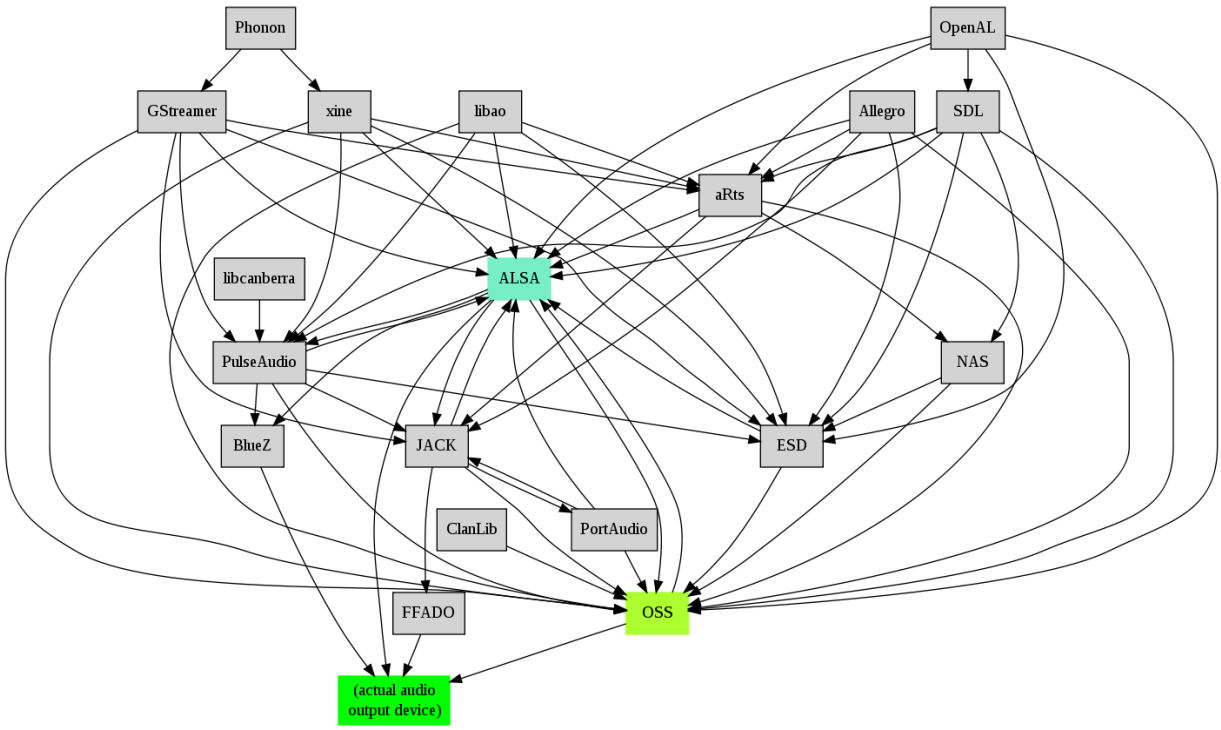
Lennart Poettering in A Guide Through The Linux Sound API Jungle offers a good summary of these different use cases:

- **I want to write a media-player-like application!**: Use GStreamer! (Unless your focus is only KDE in which cases Phonon might be an alternative.)
- **I want to add event sounds to my application!**: Use libcanberra, install your sound files according to the XDG Sound Theming/Naming Specifications! (Unless your focus is only KDE in which case KNotify might be an alternative although it has a different focus.)
- **I want to do professional audio programming, hard-disk recording, music synthesizing, MIDI interfacing!**: Use JACK and/or the full ALSA interface.
- **I want to do basic PCM audio playback/capturing!**: Use the safe ALSA subset.
- **I want to add sound to my game!**: Use the audio API of SDL for full-screen games, libcanberra for simple games with standard UIs such as Gtk+.
- **I want to write a mixer application!**: Use the layer you want to support directly: if you want to support enhanced desktop software mixers, use the PulseAudio volume control APIs. If you want to support hardware mixers, use the ALSA mixer APIs.
- **I want to write audio software for the plumbing layer!**: Use the full ALSA stack.

> - **I want to write audio software for embedded applications!**: For technical appliances usually the safe ALSA subset is a good choice, this however depends highly on your use-case.

Complexities

The simple diagram I drew above hides the real complexities of Linux sound. Mike Melanson (an Adobe engineer) in 2007 produced this diagram



The Linux Audio Mess
Origin: Mike Melanson, http://blogs.adobe.com/penguin.swf/
Updated October 10, 2008

The figure is not up-to-date: for example, OSS is no longer a major part of Linux. Some special-case complexities are, for example, that PulseAudio sits above ALSA, and it also sits *below* ALSA as in this figure from insane coding :

(This diagram is upside-down compared to mine.)

The explanation is as follows:

- PulseAudio is able to do things like mixing application sounds that ALSA cannot do
- PulseAudio installs itself as the default ALSA output device
- An application sends audio to the ALSA default device which sends it to PulseAudio
- PulseAudio mixes it it with any other audio and then sends it back to a particular device in ALSA
- ALSA then plays the mixed sound

Complex, yes, but is accomplishes tasks that wouild be difficult otherwise .

Conclusion

The architecture of Linux sound systems is complex, and new wrinkles are being added on a regular basis. However, this is the same for any audio system. Successive chapters will flesh out the detals of many of these components.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

ALSA is a low-level interface to the sound cards. If you are building your own sound server system, or writing device drivers, then you would be interested in ALSA. It sits at the bottom of most of the current Linux systems so to understand them you may need to understand aspects of ALSA. Otherwise, you are probably not interested so should move on.

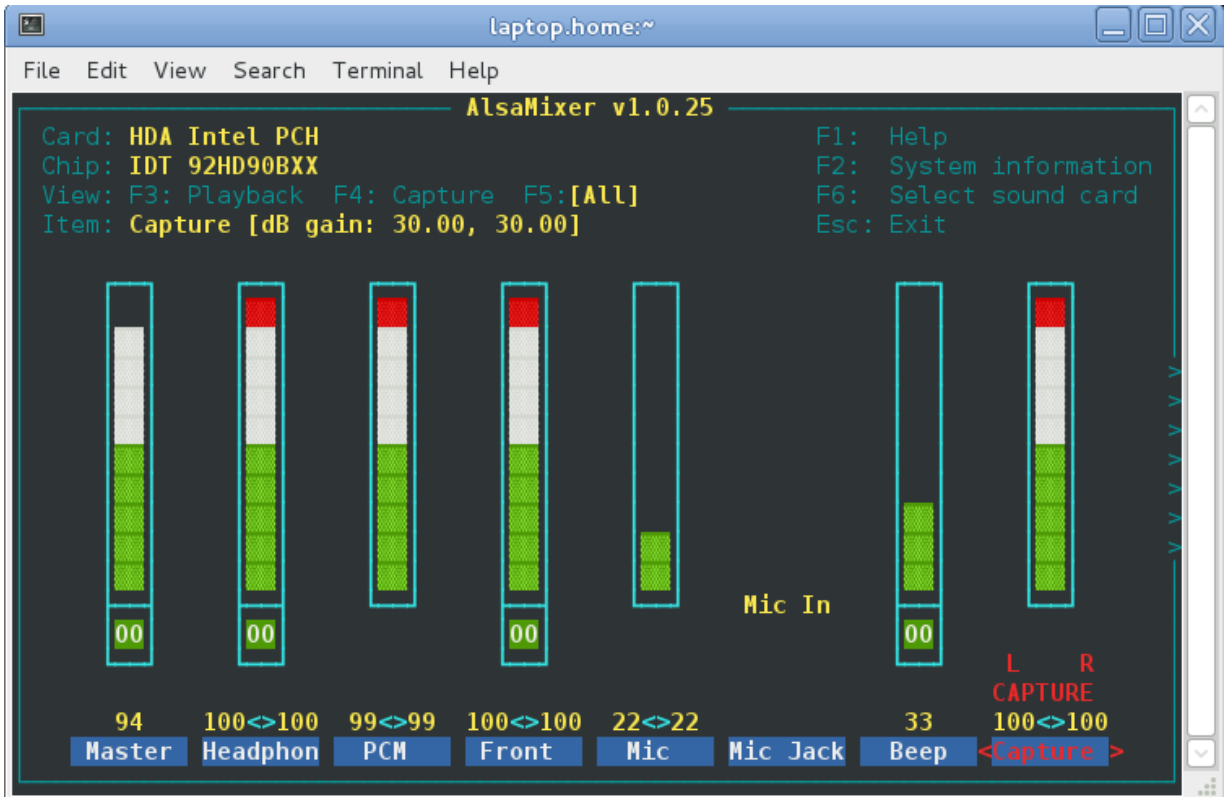
Resources

- Introduction to Sound Programming with ALSA by Jeff Tranter
- A close look at ALSA by Volker Schatz
- ALSA API
- ALSA Programming HOWTO by Matthias Nagorni
- Linux sound HOWTO for ALSA users from Tech Patterns

User space tools

alsamixer

`alsamixer` runs within a terminal window and allows you to select sound cards and controls interfaces on those cards. It looks like



`amixer` is a command line applications with similar functions.

Compared to the general mixer functions described in the Basic chapter , the mixer functions are quite limited:

- Setting the playback and capture volumes on output and input channels
- Muting or unmuting a card

The document Sound configuration on Raspberry Pi with ALSA by Stephen C Phillips is applicable to all other ALSA systems and not just the Raspberry Pi.

alsactl

Simple control programs for ALSA configurations.

speaker-test

This command allows you to test which outputs go where. For e.g. 5 channel sound, running

```
speaker-test -t wav -c 5
```

will produce on my default sound card the text and audio of

```
speaker-test 1.0.25

Playback device is default
Stream parameters are 48000Hz, S16_LE, 5
channels
WAV file(s)
Rate set to 48000Hz (requested 48000Hz)
Buffer size range from 39 to 419430
Period size range from 12 to 139810
Using max buffer size 419428
Periods = 4
was set period_size = 104857
was set buffer_size = 419428
 0 - Front Left
 1 - Front Right
 2 - Rear Left
 3 - Rear Right
 4 - Center
Time per period = 12.948378
```

It will also play the phrases "Front Left" etc to the relevant spekaer.

aplay/arecord

Play a file or record to a file. To play the microphone to the speaker,

```
arecord -r 44100 --buffer-size=128 | aplay --buffer-size=128
```

or to record it to file

```
arecord -f dat -d 20 -D hw:0,0 test.wav
```

will record a 20-second WAV file at DAT quality on your first available soundcard (hw:0,0). DAT quality is defined as stereo digital audio recorded with a 48 kHz samping rate and 16-bit resolution.

Identifying ALSA cards

The simplest ways are to run `aplay` and `arecord` with the '-l' option:

- `arecord -l`

```
**** List of CAPTURE Hardware Devices ****
card 0: PCH [HDA Intel PCH], device 0:
STAC92xx Analog [STAC92xx Analog]
Subdevices: 1/1
Subdevice #0: subdevice #0
card 2: Pro [SB X-Fi Surround 5.1 Pro],
device 0: USB Audio [USB Audio]
Subdevices: 1/1
Subdevice #0: subdevice #0
```

- `aplay -l`

```
**** List of PLAYBACK Hardware Devices ****
card 0: PCH [HDA Intel PCH], device 0:
STAC92xx Analog [STAC92xx Analog]
Subdevices: 1/1
Subdevice #0: subdevice #0
card 1: NVidia [HDA NVidia], device 3: HDMI
0 [HDMI 0]
Subdevices: 1/1
Subdevice #0: subdevice #0
card 1: NVidia [HDA NVidia], device 7: HDMI
1 [HDMI 1]
Subdevices: 1/1
Subdevice #0: subdevice #0
card 1: NVidia [HDA NVidia], device 8: HDMI
2 [HDMI 2]
Subdevices: 1/1
Subdevice #0: subdevice #0
card 2: Pro [SB X-Fi Surround 5.1 Pro],
device 0: USB Audio [USB Audio]
Subdevices: 1/1
Subdevice #0: subdevice #0
card 2: Pro [SB X-Fi Surround 5.1 Pro],
device 1: USB Audio [USB Audio #1]
Subdevices: 1/1
Subdevice #0: subdevice #0
```

Device names

The cards are often given names such as "hw:0" or "hw:2.2" in programs such as `qjackctl` (see the chapter on Jack). The term "hw" refers to hardware devices. The major number refers to the card number, the minor number to the device number. The *name* of the device is in [...] brackets.

Devices may also be known by *aliases* . The command `aplay -L` lists *device aliases* . For example, the "hdmi" alias is defined on my system in the configuration file `/etc/asound.conf`

```
pcm.hdmi0 {
        type hw
        card 1
        device 3 }

pcm.hdmi1 {
        type hw
        card 1
        device 7 }

pcm.hdmi2 {
        type hw
        card 1
        device 8 }
```

so that "hdmi:0" is really "hw:1,3": card 1, device 3.

Other aliases may be defined to cover a range of devices, parameterised by card and device. For example, `/usr/share/alsa/pcm/surround40.conf` defines

```
pcm.!surround40 {
        @args [ CARD DEV ]
        @args.CARD {
                type string
                default {
                        @func getenv
                        vars [

ALSA_SURROUND40_CARD
                                ALSA_PCM_CARD
                                ALSA_CARD
                        ]
                        default {
                                @func refer
                                name
defaults.pcm.surround40.card
                        }
                }
        }
        @args.DEV {
                type integer
                default {
                        @func igetenv
                        vars [

ALSA_SURROUND40_DEVICE
```

```
                            ]
                            default {
                                    @func refer
                                    name
defaults.pcm.surround40.device
                            }
                    }
            }
            ...
}
```

This defines for example "surround40:CARD=PCH,DEV=0" as an alias for hw:0,0 on my system (PCH is card 0).

I don't know an easy programmatic way to go from "card 1, device 3" to "hdmi:0".

The set of aliases can be shown using `aplay` and `arecord`:

- The output from `aplay -L` on my system is

```
default
  Default
sysdefault:CARD=PCH
  HDA Intel PCH, STAC92xx Analog
  Default Audio Device
front:CARD=PCH,DEV=0
  HDA Intel PCH, STAC92xx Analog
  Front speakers
surround40:CARD=PCH,DEV=0
  HDA Intel PCH, STAC92xx Analog
  4.0 Surround output to Front and Rear
speakers
surround41:CARD=PCH,DEV=0
  HDA Intel PCH, STAC92xx Analog
  4.1 Surround output to Front, Rear and
Subwoofer speakers
surround50:CARD=PCH,DEV=0
  HDA Intel PCH, STAC92xx Analog
  5.0 Surround output to Front, Center and
Rear speakers
surround51:CARD=PCH,DEV=0
  HDA Intel PCH, STAC92xx Analog
  5.1 Surround output to Front, Center,
Rear and Subwoofer speakers
surround71:CARD=PCH,DEV=0
  HDA Intel PCH, STAC92xx Analog
```

```
  7.1 Surround output to Front, Center,
Side, Rear and Woofer speakers
hdmi:CARD=NVidia,DEV=0
  HDA NVidia, HDMI 0
  HDMI Audio Output
hdmi:CARD=NVidia,DEV=1
  HDA NVidia, HDMI 1
  HDMI Audio Output
hdmi:CARD=NVidia,DEV=2
  HDA NVidia, HDMI 2
  HDMI Audio Output
sysdefault:CARD=Pro
  SB X-Fi Surround 5.1 Pro, USB Audio
  Default Audio Device
front:CARD=Pro,DEV=0
  SB X-Fi Surround 5.1 Pro, USB Audio
  Front speakers
surround40:CARD=Pro,DEV=0
  SB X-Fi Surround 5.1 Pro, USB Audio
  4.0 Surround output to Front and Rear
speakers
surround41:CARD=Pro,DEV=0
  SB X-Fi Surround 5.1 Pro, USB Audio
  4.1 Surround output to Front, Rear and
Subwoofer speakers
surround50:CARD=Pro,DEV=0
  SB X-Fi Surround 5.1 Pro, USB Audio
```

```
    5.0 Surround output to Front, Center and
Rear speakers
surround51:CARD=Pro,DEV=0
    SB X-Fi Surround 5.1 Pro, USB Audio
    5.1 Surround output to Front, Center,
Rear and Subwoofer speakers
surround71:CARD=Pro,DEV=0
    SB X-Fi Surround 5.1 Pro, USB Audio
    7.1 Surround output to Front, Center,
Side, Rear and Woofer speakers
iec958:CARD=Pro,DEV=0
    SB X-Fi Surround 5.1 Pro, USB Audio
    IEC958 (S/PDIF) Digital Audio Output
```

- The output from `arecord -L` is

```
default
  Default
sysdefault:CARD=PCH
  HDA Intel PCH, STAC92xx Analog
  Default Audio Device
front:CARD=PCH,DEV=0
  HDA Intel PCH, STAC92xx Analog
  Front speakers
surround40:CARD=PCH,DEV=0
  HDA Intel PCH, STAC92xx Analog
  4.0 Surround output to Front and Rear
speakers
surround41:CARD=PCH,DEV=0
  HDA Intel PCH, STAC92xx Analog
  4.1 Surround output to Front, Rear and
Subwoofer speakers
surround50:CARD=PCH,DEV=0
  HDA Intel PCH, STAC92xx Analog
  5.0 Surround output to Front, Center and
Rear speakers
surround51:CARD=PCH,DEV=0
  HDA Intel PCH, STAC92xx Analog
  5.1 Surround output to Front, Center,
Rear and Subwoofer speakers
surround71:CARD=PCH,DEV=0
  HDA Intel PCH, STAC92xx Analog
```

```
    7.1 Surround output to Front, Center,
Side, Rear and Woofer speakers
sysdefault:CARD=Pro
    SB X-Fi Surround 5.1 Pro, USB Audio
    Default Audio Device
front:CARD=Pro,DEV=0
    SB X-Fi Surround 5.1 Pro, USB Audio
    Front speakers
surround40:CARD=Pro,DEV=0
    SB X-Fi Surround 5.1 Pro, USB Audio
    4.0 Surround output to Front and Rear
speakers
surround41:CARD=Pro,DEV=0
    SB X-Fi Surround 5.1 Pro, USB Audio
    4.1 Surround output to Front, Rear and
Subwoofer speakers
surround50:CARD=Pro,DEV=0
    SB X-Fi Surround 5.1 Pro, USB Audio
    5.0 Surround output to Front, Center and
Rear speakers
surround51:CARD=Pro,DEV=0
    SB X-Fi Surround 5.1 Pro, USB Audio
    5.1 Surround output to Front, Center,
Rear and Subwoofer speakers
surround71:CARD=Pro,DEV=0
    SB X-Fi Surround 5.1 Pro, USB Audio
    7.1 Surround output to Front, Center,
```

```
Side, Rear and Woofer speakers
iec958:CARD=Pro,DEV=0
  SB X-Fi Surround 5.1 Pro, USB Audio
  IEC958 (S/PDIF) Digital Audio Output
```

ALSA configuration files

This tutorial by Volker Schatz explains what is going on in ALSA configuration files and looks really good: A close look at ALSA

We just note one thing: the default ALSA device is "hw:0". This is hard-coded into ALSA. But it can be overridden in configuration files. This is done for example by PulseAudio - see next chapter.

alsa-info

This will collect information about your system and save it in a file. It is a shell script that gives an enormous amount of information. Here is a heavily elided subset of the information:

```
upload=true&script=true&cardinfo=
!!################################
!!ALSA Information Script v 0.4.60
!!################################

!!Script ran on: Tue Jun 12 04:50:22 UTC 2012



!!Linux Distribution
!!------------------

Fedora release 16 (Verne) Fedora release 16
(Verne) Fedora release 16 (Verne) Fedora
release 16 (Verne)

...

!!ALSA Version
!!------------

Driver version:     1.0.24
Library version:    1.0.25
Utilities version:  1.0.25



!!Loaded ALSA modules
```

```
!!--------------------

snd_hda_intel
snd_hda_intel



!!Sound Servers on this system
!!-----------------------------

Pulseaudio:
      Installed - Yes (/usr/bin/pulseaudio)
      Running - Yes

Jack:
      Installed - Yes (/usr/bin/jackd)
      Running - No



!!Soundcards recognised by ALSA
!!------------------------------

 0 [PCH            ]: HDA-Intel - HDA Intel PCH
                      HDA Intel PCH at
0xe6e60000 irq 47
 1 [NVidia         ]: HDA-Intel - HDA NVidia
                      HDA NVidia at 0xe5080000
irq 17
```

```
!!PCI Soundcards installed in the system
!!----------------------------------------

00:1b.0 Audio device: Intel Corporation 6
Series/C200 Series Chipset Family High
Definition Audio Controller (rev 04)
01:00.1 Audio device: nVidia Corporation HDMI
Audio stub (rev a1)



...

!!HDA-Intel Codec information
!!---------------------------

...

Default PCM:
    rates [0x5e0]: 44100 48000 88200 96000
192000
    bits [0xe]: 16 20 24
    formats [0x1]: PCM

Node 0x0a [Pin Complex] wcaps 0x400583: Stereo
Amp-In
```

```
  Control: name="Mic Jack Mode", index=0,
device=0
    ControlAmp: chs=0, dir=In, idx=0, ofs=0
  Control: name="Mic Capture Volume", index=0,
device=0
    ControlAmp: chs=3, dir=In, idx=0, ofs=0
  Control: name="Mic Jack", index=0, device=0
  Amp-In caps: N/A
  Amp-In vals:  [0x01 0x01]
  Pincap 0x0001173c: IN OUT HP EAPD Detect
    Vref caps: HIZ 50 GRD 80
  EAPD 0x2: EAPD
  Pin Default 0x03a11020: [Jack] Mic at Ext
Left
    Conn = 1/8, Color = Black
    DefAssociation = 0x2, Sequence = 0x0
  Pin-ctls: 0x24: IN VREF_80
  Unsolicited: tag=03, enabled=1
  Power: setting=D0, actual=D0
  Connection: 3
     0x13* 0x14 0x1c



!!ALSA configuration files
!!------------------------
```

```
!!System wide config file (/etc/asound.conf)

#
# Place your global alsa-lib configuration
here...
#

@hooks [
    {
        func load
        files [
            "/etc/alsa/pulse-default.conf"
        ]
        errors false
    }
]

pcm.hdmi0 {
        type hw
        card 1
        device 3 }

pcm.hdmi1 {
        type hw
        card 1
        device 7 }
```

```
pcm.hdmi2 {
        type hw
        card 1
        device 8 }


!!Aplay/Arecord output
!!------------

APLAY

**** List of PLAYBACK Hardware Devices ****
card 0: PCH [HDA Intel PCH], device 0: STAC92xx
Analog [STAC92xx Analog]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 1: NVidia [HDA NVidia], device 3: HDMI 0
[HDMI 0]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 1: NVidia [HDA NVidia], device 7: HDMI 1
[HDMI 1]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 1: NVidia [HDA NVidia], device 8: HDMI 2
[HDMI 2]
  Subdevices: 1/1
```

```
  Subdevice #0: subdevice #0

ARECORD

**** List of CAPTURE Hardware Devices ****
card 0: PCH [HDA Intel PCH], device 0: STAC92xx
Analog [STAC92xx Analog]
  Subdevices: 1/1
  Subdevice #0: subdevice #0

!!Amixer output
!!-------------

!!-------Mixer controls for card 0 [PCH]

Card hw:0 'PCH'/'HDA Intel PCH at 0xe6e60000
irq 47'
  Mixer name    : 'IDT 92HD90BXX'
  Components    :
'HDA:111d76e7,10280494,00100102'
  Controls      : 19
  Simple ctrls  : 10
Simple mixer control 'Master',0
  Capabilities: pvolume pvolume-joined pswitch
pswitch-joined penum
  Playback channels: Mono
  Limits: Playback 0 - 64
```

```
  Mono: Playback 62 [97%] [-1.50dB] [on]
Simple mixer control 'Headphone',0
  Capabilities: pvolume pswitch penum
  Playback channels: Front Left - Front Right
  Limits: Playback 0 - 64
  Mono:
  Front Left: Playback 64 [100%] [0.00dB] [on]
  Front Right: Playback 64 [100%] [0.00dB] [on]
Simple mixer control 'PCM',0
  Capabilities: pvolume penum
  Playback channels: Front Left - Front Right
  Limits: Playback 0 - 255
  Mono:
  Front Left: Playback 254 [100%] [0.20dB]
  Front Right: Playback 254 [100%] [0.20dB]
Simple mixer control 'Front',0
  Capabilities: pvolume pswitch penum
  Playback channels: Front Left - Front Right
  Limits: Playback 0 - 64
  Mono:
  Front Left: Playback 64 [100%] [0.00dB] [on]
  Front Right: Playback 64 [100%] [0.00dB] [on]
Simple mixer control 'Mic',0
  Capabilities: cvolume penum
  Capture channels: Front Left - Front Right
  Limits: Capture 0 - 3
  Front Left: Capture 1 [33%] [10.00dB]
```

```
  Front Right: Capture 1 [33%] [10.00dB]
Simple mixer control 'Mic Jack Mode',0
  Capabilities: enum
  Items: 'Mic In' 'Line In'
  Item0: 'Mic In'
Simple mixer control 'Beep',0
  Capabilities: pvolume pvolume-joined pswitch
pswitch-joined penum
  Playback channels: Mono
  Limits: Playback 0 - 3
  Mono: Playback 1 [33%] [-12.00dB] [on]
Simple mixer control 'Capture',0
  Capabilities: cvolume cswitch penum
  Capture channels: Front Left - Front Right
  Limits: Capture 0 - 46
  Front Left: Capture 46 [100%] [30.00dB] [on]
  Front Right: Capture 46 [100%] [30.00dB] [on]
Simple mixer control 'Dock Mic',0
  Capabilities: cvolume penum
  Capture channels: Front Left - Front Right
  Limits: Capture 0 - 3
  Front Left: Capture 0 [0%] [0.00dB]
  Front Right: Capture 0 [0%] [0.00dB]
Simple mixer control 'Internal Mic',0
  Capabilities: cvolume penum
  Capture channels: Front Left - Front Right
  Limits: Capture 0 - 3
```

```
  Front Left: Capture 0 [0%] [0.00dB]
  Front Right: Capture 0 [0%] [0.00dB]

!!-------Mixer controls for card 1 [NVidia]

Card hw:1 'NVidia'/'HDA NVidia at 0xe5080000
irq 17'
  Mixer name    : 'Nvidia GPU 1c HDMI/DP'
  Components    :
'HDA:10de001c,10281494,00100100'
  Controls      : 18
  Simple ctrls  : 3
Simple mixer control 'IEC958',0
  Capabilities: pswitch pswitch-joined penum
  Playback channels: Mono
  Mono: Playback [on]
Simple mixer control 'IEC958',1
  Capabilities: pswitch pswitch-joined penum
  Playback channels: Mono
  Mono: Playback [off]
Simple mixer control 'IEC958',2
  Capabilities: pswitch pswitch-joined penum
  Playback channels: Mono
  Mono: Playback [off]



!!Alsactl output
```

```
!!-------------

--startcollapse--
state.PCH {
    control.1 {
        iface MIXER
        name 'Front Playback Volume'
        value.0 64
        value.1 64
        comment {
            access 'read write'
            type INTEGER
            count 2
            range '0 - 64'
            dbmin -4800
            dbmax 0
            dbvalue.0 0
            dbvalue.1 0
        }
    }
...
```

Applications using ALSA

Many applications can directly use ALSA by appropriate commandline arguments.

MPlayer

To play a file to an ALSA device using `mplayer`, use code such as

```
mplayer -ao alsa:device=hw=1.0 -srate 48000
bryan.mp3
```

VLC

To play a file to an ALSA device using `vlc`, use code such as

```
vlc --aout alsa ...
```

TiMidity

To play a file to an ALSA device using `timidity`, use code such as

```
timidity -Os ...
```

Programming ALSA

There are several tutorials, including A Tutorial on Using the ALSA Audio API by Paul Davis (who is the lead on Jack).

An overview of the API is at PCM (digital audio) interface . The ALSA API is large and complex. It is not clear to me how it all hangs together or what part to use where. Jeff Tranter Introduction to Sound Programming with ALSA states

> The ALSA API can be broken down into the major interfaces it supports:
>
> - Control interface: a general-purpose facility for managing registers of sound cards and querying the available devices.
> - PCM interface: the interface for managing digital audio capture and playback. [...] it is the one most commonly used for digital audio applications.
> - Raw MIDI interface: supports MIDI (Musical Instrument Digital Interface), a standard for electronic musical instruments. This API provides access to a MIDI bus on a sound card. The raw interface works directly with the MIDI events, and the programmer is responsible for managing the protocol and timing.
> - Timer interface: provides access to timing hardware on sound cards used for synchronizing sound events.
> - Sequencer interface: a higher-level interface for MIDI programming and sound synthesis than the raw MIDI interface. It handles much of the MIDI protocol and timing.
> - Mixer interface: controls the devices on sound cards that route signals and control volume levels. It is built on top of the control interface.

Hardware device information

Finding information about hardware cards and devices is a multi-step operation. The hardware cards first have to be identified. This is done using the Control interface functions. The ones used are

```
snd_card_next
snd_ctl_open
snd_ctl_pcm_next_device
snd_ctl_card_info_get_id
snd_ctl_card_info_get_name
```

Cards are identified by an integer from zero upwards. The *next* card number is found using `snd_card_next`, and the first card is found using a seed value of `-1`. The card is then opened using its ALSA name such as hw:0, hw:1, etc by `snd_ctl_open` which fills in a `handle` value. In turn, this handle is used to fill in card information using `snd_ctl_card_info` and fields are extracted from that using functions such as `snd_ctl_card_info_get_name`. In the program that follows, this gives information such as

```
card 0: PCH [HDA Intel PCH]
```

For further information you need to switch to the PCM functions for the card. The function linking the control and PCM interfaces is `snd_ctl_pcm_info` which fills in a structure of type `snd_pcm_info_t` with PCM-related information. Unfortunately, this function is documented neither in the Control Interface nor the PCM interface sections of the ALSA documentation but is instead in the Files section under control.c The structure `snd_pcm_info_t` is barely documented in the PCM Interface section, and only has a few fields of interest. (see here for the structure). These fields are accessed using the PCM functions `snd_pcm_info_get_id` and `snd_pcm_info_get_name` .

The main value of the `snd_pcm_info_t` structure is that it is the principal parameter into the functions of the PCM Stream . In particular this allows you to get devices and subdevices and information about them.

The program to find and display card and hardware device information is aplay-l.c:

```
/**
 * aplay-l.c
 *
 * Code from aplay.c
 *
 * does the same as aplay -l
 * http://alsa-
utils.sourcearchive.com/documentation/1.0.15/apla
source.html
 */

/*
 * Original notice:
 *
 *  Copyright (c) by Jaroslav Kysela
<perex@perex.cz>
 *  Based on vplay program by Michael Beck
 *
 *
 *   This program is free software; you can
redistribute it and/or modify
 *   it under the terms of the GNU General
Public License as published by
 *   the Free Software Foundation; either
version 2 of the License, or
 *   (at your option) any later version.
```

```
 *
 *   This program is distributed in the hope
that it will be useful,
 *   but WITHOUT ANY WARRANTY; without even the
implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.  See the
 *   GNU General Public License for more
details.
 *
 *   You should have received a copy of the GNU
General Public License
 *   along with this program; if not, write to
the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite
330, Boston, MA  02111-1307 USA
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>
#include <locale.h>

// used by gettext for i18n, not needed here
#define _(STR) STR
```

```
static void device_list(snd_pcm_stream_t
stream)
{
	snd_ctl_t *handle;
	int card, err, dev, idx;
	snd_ctl_card_info_t *info;
	snd_pcm_info_t *pcminfo;
	snd_ctl_card_info_alloca(&info);
	snd_pcm_info_alloca(&pcminfo);

	card = -1;
	if (snd_card_next(&card) < 0 || card < 0)
{
		error(_("no soundcards found..."));
		return;
	}
	printf(_("**** List of %s Hardware
Devices ****\n"),
		 snd_pcm_stream_name(stream));
	while (card >= 0) {
		char name[32];
		sprintf(name, "hw:%d", card);
		if ((err = snd_ctl_open(&handle,
name, 0)) < 0) {
			error("control open (%i):
%s", card, snd_strerror(err));
			goto next_card;
```

```
            }
            if ((err =
snd_ctl_card_info(handle, info)) < 0) {
                  error("control hardware info
(%i): %s", card, snd_strerror(err));
                  snd_ctl_close(handle);
                  goto next_card;
            }
            dev = -1;
            while (1) {
                  unsigned int count;
                  if
(snd_ctl_pcm_next_device(handle, &dev)<0)

error("snd_ctl_pcm_next_device");
                  if (dev < 0)
                        break;

snd_pcm_info_set_device(pcminfo, dev);

snd_pcm_info_set_subdevice(pcminfo, 0);

snd_pcm_info_set_stream(pcminfo, stream);
                  if ((err =
snd_ctl_pcm_info(handle, pcminfo)) < 0) {
                        if (err != -ENOENT)
                              error("control
```

```
digital audio info (%i): %s", card,
snd_strerror(err));
                        continue;
                }
                printf(_("card %i: [%s,%i] %s
[%s], device %i: %s [%s]\n"),
            card, name, dev,
snd_ctl_card_info_get_id(info),
snd_ctl_card_info_get_name(info),
                        dev,

snd_pcm_info_get_id(pcminfo),

snd_pcm_info_get_name(pcminfo));
                count =
snd_pcm_info_get_subdevices_count(pcminfo);
                printf( _("  Subdevices:
%i/%i\n"),

snd_pcm_info_get_subdevices_avail(pcminfo),
count);
                for (idx = 0; idx <
(int)count; idx++) {

snd_pcm_info_set_subdevice(pcminfo, idx);
                        if ((err =
snd_ctl_pcm_info(handle, pcminfo)) < 0) {
```

```
                                error("control
digital audio playback info (%i): %s", card,
snd_strerror(err));
                        } else {
                                printf(_("
Subdevice #%i: %s\n"),
                                        idx,
snd_pcm_info_get_subdevice_name(pcminfo));
                        }
                }
            }
            snd_ctl_close(handle);
      next_card:
            if (snd_card_next(&card) < 0) {
                  error("snd_card_next");
                  break;
            }
      }
}


main (int argc, char *argv[])
{
  device_list(SND_PCM_STREAM_CAPTURE);
  device_list(SND_PCM_STREAM_PLAYBACK);
}
```

The output from running `aplay-l` on my system is

```
**** List of CAPTURE Hardware Devices ****
card 0: [hw:0,0] PCH [HDA Intel PCH], device 0:
STAC92xx Analog [STAC92xx Analog]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
**** List of PLAYBACK Hardware Devices ****
card 0: [hw:0,0] PCH [HDA Intel PCH], device 0:
STAC92xx Analog [STAC92xx Analog]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 1: [hw:1,3] NVidia [HDA NVidia], device 3:
HDMI 0 [HDMI 0]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 1: [hw:1,7] NVidia [HDA NVidia], device 7:
HDMI 1 [HDMI 1]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 1: [hw:1,8] NVidia [HDA NVidia], device 8:
HDMI 2 [HDMI 2]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

PCM device information

PCM alias information may be found from the devices by `aplay -L`. This uses the "hints" mechanism from the device API. Note that the program is responsible for freeing memory allocated by the ALSA library. This means that if a string or table is returned then not only do you have to walk through the string/table but you have to retain a pointer to the start of the string/table so that it can be freed.

The source for this is aplay-L.c:

```
/**
 * aplay-L.c
 *
 * Code from aplay.c
 * does aplay -L
 * http://alsa-
utils.sourcearchive.com/documentation/1.0.15/apla
source.html
 */

/*
 * Original notice:
 *
 *  Copyright (c) by Jaroslav Kysela
<perex@perex.cz>
 *  Based on vplay program by Michael Beck
 *
 *
 *   This program is free software; you can
redistribute it and/or modify
 *   it under the terms of the GNU General
Public License as published by
 *   the Free Software Foundation; either
version 2 of the License, or
 *   (at your option) any later version.
 *
```

```
 *   This program is distributed in the hope
that it will be useful,
 *   but WITHOUT ANY WARRANTY; without even the
implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.  See the
 *   GNU General Public License for more
details.
 *
 *   You should have received a copy of the GNU
General Public License
 *   along with this program; if not, write to
the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite
330, Boston, MA  02111-1307 USA
 *
 */


#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>
#include <locale.h>

#define _(STR) STR

static void pcm_list(snd_pcm_stream_t stream )
```

```
{
	void **hints, **n;
	char *name, *descr, *descr1, *io;
	const char *filter;

	if (snd_device_name_hint(-1, "pcm",
&hints) < 0)
		return;
	n = hints;
	filter = stream == SND_PCM_STREAM_CAPTURE
? "Input" : "Output";
	while (*n != NULL) {
		name = snd_device_name_get_hint(*n,
"NAME");
		descr =
snd_device_name_get_hint(*n, "DESC");
		io = snd_device_name_get_hint(*n,
"IOID");
		if (io != NULL && strcmp(io,
filter) == 0)
			goto __end;
		printf("%s\n", name);
		if ((descr1 = descr) != NULL) {
			printf("    ");
			while (*descr1) {
				if (*descr1 == '\n')
					printf("\n    ");
```

```
                        else
                              putchar(*descr1);
                        descr1++;
                  }
                  putchar('\n');
            }
            __end:
                  if (name != NULL)
                        free(name);
            if (descr != NULL)
                  free(descr);
            if (io != NULL)
                  free(io);
            n++;
      }
      snd_device_name_free_hint(hints);
}


main (int argc, char *argv[])
{
  printf("*********** CAPTURE ***********\n");
  pcm_list(SND_PCM_STREAM_CAPTURE);

  printf("\n\n********** PLAYBACK
**********\n");
```

```
    pcm_list(SND_PCM_STREAM_PLAYBACK);
  }
```

The outputfrom running `aplay-L` on my system is

```
*********** CAPTURE ***********
default
    Default
sysdefault:CARD=PCH
    HDA Intel PCH, STAC92xx Analog
    Default Audio Device
front:CARD=PCH,DEV=0
    HDA Intel PCH, STAC92xx Analog
    Front speakers
surround40:CARD=PCH,DEV=0
    HDA Intel PCH, STAC92xx Analog
    4.0 Surround output to Front and Rear
speakers
surround41:CARD=PCH,DEV=0
    HDA Intel PCH, STAC92xx Analog
    4.1 Surround output to Front, Rear and
Subwoofer speakers
surround50:CARD=PCH,DEV=0
    HDA Intel PCH, STAC92xx Analog
    5.0 Surround output to Front, Center and
Rear speakers
surround51:CARD=PCH,DEV=0
    HDA Intel PCH, STAC92xx Analog
    5.1 Surround output to Front, Center, Rear
and Subwoofer speakers
surround71:CARD=PCH,DEV=0
```

```
    HDA Intel PCH, STAC92xx Analog
    7.1 Surround output to Front, Center, Side,
Rear and Woofer speakers
hdmi:CARD=NVidia,DEV=0
    HDA NVidia, HDMI 0
    HDMI Audio Output
hdmi:CARD=NVidia,DEV=1
    HDA NVidia, HDMI 1
    HDMI Audio Output
hdmi:CARD=NVidia,DEV=2
    HDA NVidia, HDMI 2
    HDMI Audio Output


*********** PLAYBACK ***********
null
    Discard all samples (playback) or generate
zero samples (capture)
pulse
    PulseAudio Sound Server
default
    Default
sysdefault:CARD=PCH
    HDA Intel PCH, STAC92xx Analog
    Default Audio Device
front:CARD=PCH,DEV=0
    HDA Intel PCH, STAC92xx Analog
```

```
    Front speakers
surround40:CARD=PCH,DEV=0
    HDA Intel PCH, STAC92xx Analog
    4.0 Surround output to Front and Rear
speakers
surround41:CARD=PCH,DEV=0
    HDA Intel PCH, STAC92xx Analog
    4.1 Surround output to Front, Rear and
Subwoofer speakers
surround50:CARD=PCH,DEV=0
    HDA Intel PCH, STAC92xx Analog
    5.0 Surround output to Front, Center and
Rear speakers
surround51:CARD=PCH,DEV=0
    HDA Intel PCH, STAC92xx Analog
    5.1 Surround output to Front, Center, Rear
and Subwoofer speakers
surround71:CARD=PCH,DEV=0
    HDA Intel PCH, STAC92xx Analog
    7.1 Surround output to Front, Center, Side,
Rear and Woofer speakers
```

Note that this does not include the "plug" devices such as "plughw:0". The list of plug devices does not seem to be accessible.

Configuration space information

In addition to general characteristics, each PCM device is able to support a range of parameters such as the number of channels, sampling rates, etc. The full set and range of parameters form the "configuration space" of each device. For example, a device may support between 2 and 6 channels and a number of different sampling rates. These two parameters form a 2-dimensional space. The full set form an N-dimensional space.

ALSA has functions to query this space and to set values within this space. The space is initialised by `snd_pcm_hw_params_any`. To find the possible values of parameters there are functions `snd_pcm_hw_params_get...`.

The different parameters are

- **Channels**: The number of channels supported - zero for mono, etc
- **Rate**: The sampling rate in hertz, that is, samples per second. Typically CD audio has a sampling rate of 44,100hz per channel, so that each channel has 44,100 samples per second
- **Frames**: Each frame contains one sample for each channel. Stereo audio will contain 2 samples in each frame. The frame rate is the same as the sampling rate. That is, suppose the

sampling rate for stereo audio is 44,100hz. Then each channel will have 44,100 samples per second. But there will also be 44,100 frames per second, so that the overall density of the two channels will be 88,200 samples per second.

- **Period time**: The time in microseconds between hardware interrupts to refresh the buffer
- **Period size**: The number of frames in between each hardware interrupt. These are related in the following way:

```
Period time = period size x time per frame
            = period size x time per sample
            = period size / sampling rate
```

  So for example if the sampling rate is 48000hz stereo and the period size is 8192 frames, then the time between hardware interrupts is 8192 / 48000 seconds = 170.5 millseconds
- **Periods**: Number of periods per buffer
- **Buffer time**: Time for one buffer
- **Buffer size**: Size of the buffer in frames. Again there is a relationship

```
Time of one buffer =  buffer size in frames
x time for one frame
                  = buffer size x number of
channels x time for one sample
                  = buffer size x number of
channels / sample rate
```

The buffer size should be a multiple of the period size, and is typically twice as big.

For further examples, see FramesPeriods

A program to find the range of values of various parameters from the initial state is device-info.c:

```c
/**
 * Jan Newmarch
 */

#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>


void info(char *dev_name, snd_pcm_stream_t
stream) {
  snd_pcm_hw_params_t *hw_params;
  int err;
  snd_pcm_t *handle;
  unsigned int max;
  unsigned int min;
  unsigned int val;
  unsigned  int dir;
  snd_pcm_uframes_t frames;

  if ((err = snd_pcm_open (&handle, dev_name,
stream, 0)) < 0) {
    fprintf (stderr, "cannot open audio device
%s (%s)\n",
         dev_name,
         snd_strerror (err));
```

```
    return;
  }

  if ((err = snd_pcm_hw_params_malloc
(&hw_params)) < 0) {
    fprintf (stderr, "cannot allocate hardware
parameter structure (%s)\n",
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_any (handle,
hw_params)) < 0) {
    fprintf (stderr, "cannot initialize
hardware parameter structure (%s)\n",
         snd_strerror (err));
    exit (1);
  }

  if ((err =
snd_pcm_hw_params_get_channels_max(hw_params,
&max)) < 0) {
    fprintf (stderr, "cannot  (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("max channels %d\n", max);
```

```
  if ((err =
snd_pcm_hw_params_get_channels_min(hw_params,
&min)) < 0) {
    fprintf (stderr, "cannot get channel info
(%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("min channels %d\n", min);

  /*
  if ((err =
snd_pcm_hw_params_get_sbits(hw_params)) < 0) {
      fprintf (stderr, "cannot get bits info
(%s)\n",
           snd_strerror (err));
      exit (1);
  }
  printf("bits %d\n", err);
  */

  if ((err =
snd_pcm_hw_params_get_rate_min(hw_params, &val,
&dir)) < 0) {
    fprintf (stderr, "cannot get min rate
(%s)\n",
```

```
        snd_strerror (err));
    exit (1);
  }
  printf("min rate %d hz\n", val);

  if ((err =
snd_pcm_hw_params_get_rate_max(hw_params, &val,
&dir)) < 0) {
    fprintf (stderr, "cannot get max rate
(%s)\n",
        snd_strerror (err));
    exit (1);
  }
  printf("max rate %d hz\n", val);


  if ((err =
snd_pcm_hw_params_get_period_time_min(hw_params,
 &val, &dir)) < 0) {
    fprintf (stderr, "cannot get min period
time  (%s)\n",
        snd_strerror (err));
    exit (1);
  }
  printf("min period time %d usecs\n", val);

  if ((err =
```

```
snd_pcm_hw_params_get_period_time_max(hw_params,
 &val, &dir)) < 0) {
    fprintf (stderr, "cannot  get max period
time  (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("max period time %d usecs\n", val);

  if ((err =
snd_pcm_hw_params_get_period_size_min(hw_params,
 &frames, &dir)) < 0) {
    fprintf (stderr, "cannot  get min period
size  (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("min period size in frames %d\n",
frames);

  if ((err =
snd_pcm_hw_params_get_period_size_max(hw_params,
 &frames, &dir)) < 0) {
    fprintf (stderr, "cannot  get max period
size (%s)\n",
         snd_strerror (err));
    exit (1);
```

```
  }
  printf("max period size in frames %d\n",
frames);

  if ((err =
snd_pcm_hw_params_get_periods_min(hw_params,
&val, &dir)) < 0) {
    fprintf (stderr, "cannot  get min periods
(%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("min periods per buffer %d\n", val);

  if ((err =
snd_pcm_hw_params_get_periods_max(hw_params,
&val, &dir)) < 0) {
    fprintf (stderr, "cannot  get min periods
(%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("max periods per buffer %d\n", val);

  if ((err =
snd_pcm_hw_params_get_buffer_time_min(hw_params,
 &val, &dir)) < 0) {
```

```
    fprintf (stderr, "cannot get min buffer
time (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("min buffer time %d usecs\n", val);

  if ((err =
snd_pcm_hw_params_get_buffer_time_max(hw_params,
 &val, &dir)) < 0) {
    fprintf (stderr, "cannot get max buffer
time  (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("max buffer time %d usecs\n", val);

  if ((err =
snd_pcm_hw_params_get_buffer_size_min(hw_params,
 &frames)) < 0) {
    fprintf (stderr, "cannot get min buffer
size (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("min buffer size in frames %d\n",
frames);
```

```
  if ((err =
snd_pcm_hw_params_get_buffer_size_max(hw_params,
 &frames)) < 0) {
    fprintf (stderr, "cannot get max buffer
size  (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("max buffer size in frames %d\n",
frames);
}

main (int argc, char *argv[])
{
  int i;
  int err;
  int buf[128];
  FILE *fin;
  size_t nread;
  unsigned int rate = 44100;

  if (argc != 2) {
    fprintf(stderr, "Usage: %s card\n",
argv[0]);
    exit(1);
  }
```

```
  printf("*********** CAPTURE ***********\n");
  info(argv[1], SND_PCM_STREAM_CAPTURE);

  printf("*********** PLAYBACK ***********\n");
  info(argv[1], SND_PCM_STREAM_PLAYBACK);

  exit (0);
}
```

The output from `device-info hw:0` on my system is

```
*********** CAPTURE ***********
max channels 2
min channels 2
min rate 44100 hz
max rate 192000 hz
min period time 83 usecs
max period time 11888617 usecs
min period size in frames 16
max period size in frames 524288
min periods per buffer 2
max periods per buffer 32
min buffer time 166 usecs
max buffer time 23777234 usecs
min buffer size in frames 32
max buffer size in frames 1048576
*********** PLAYBACK ***********
max channels 2
min channels 2
min rate 44100 hz
max rate 192000 hz
min period time 83 usecs
max period time 11888617 usecs
min period size in frames 16
max period size in frames 524288
min periods per buffer 2
max periods per buffer 32
```

```
min buffer time 166 usecs
max buffer time 23777234 usecs
min buffer size in frames 32
max buffer size in frames 1048576
```

This program works with any ALSA device, including the "plug" devices. The output from `device-info plughw:0` shows how the software wrapper can give a wider range of possible values:

```
*********** CAPTURE ***********
max channels 10000
min channels 1
min rate 4000 hz
max rate -1 hz
min period time 83 usecs
max period time 11888617 usecs
min period size in frames 0
max period size in frames -1
min periods per buffer 0
max periods per buffer -1
min buffer time 1 usecs
max buffer time -1 usecs
min buffer size in frames 1
max buffer size in frames -2
*********** PLAYBACK ***********
max channels 10000
min channels 1
min rate 4000 hz
max rate -1 hz
min period time 83 usecs
max period time 11888617 usecs
min period size in frames 0
max period size in frames -1
min periods per buffer 0
max periods per buffer -1
```

```
min buffer time 1 usecs
max buffer time -1 usecs
min buffer size in frames 1
max buffer size in frames -2
```

It can also be run with alias devices, such as `device-info surround40`.

ALSA initialisation

A line-by-line breakdown is at ALSA Tutorial Part 1 - Initialization This explains much of the common code in the programs that follow.

Capture audio to a file

The following program is from Paul Davis A Tutorial on Using the ALSA Audio API alsa_capture.c:

```c
/**
 * alsa_capture.c
 */

/**
 * Paul Davis
 * http://equalarea.com/paul/alsa-
audio.html#howto
 */

/**
 * Jan Newmarch
 */

#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>
#include <signal.h>

#define BUFSIZE 128
#define RATE 44100

FILE *fout = NULL;

/*
 * quit on ctrl-c
```

```
 */
void sigint(int sig) {
  if (fout != NULL) {
    fclose(fout);
  }
  exit(1);
}

main (int argc, char *argv[])
{
  int i;
  int err;
  short buf[BUFSIZE];
  snd_pcm_t *capture_handle;
  snd_pcm_hw_params_t *hw_params;
  snd_pcm_format_t rate = RATE;
  int nread;

  if (argc != 3) {
    fprintf(stderr, "Usage: %s cardname
file\n", argv[0]);
    exit(1);
  }

  if ((fout = fopen(argv[2], "w")) == NULL) {
    fprintf(stderr, "Can't open %s for
writing\n", argv[2]);
```

```
    exit(1);
  }



  signal(SIGINT, sigint);

  if ((err = snd_pcm_open (&capture_handle,
argv[1], SND_PCM_STREAM_CAPTURE, 0)) < 0) {
    fprintf (stderr, "cannot open audio device
%s (%s)\n",
         argv[1],
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_malloc
(&hw_params)) < 0) {
    fprintf (stderr, "cannot allocate hardware
parameter structure (%s)\n",
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_any
(capture_handle, hw_params)) < 0) {
    fprintf (stderr, "cannot initialize
hardware parameter structure (%s)\n",
```

```
        snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_set_access
(capture_handle, hw_params,
SND_PCM_ACCESS_RW_INTERLEAVED)) < 0) {
    fprintf (stderr, "cannot set access type
(%s)\n",
        snd_strerror (err));
    exit (1);
  }


  if ((err = snd_pcm_hw_params_set_format
(capture_handle, hw_params,
SND_PCM_FORMAT_S16_LE)) < 0) {
    fprintf (stderr, "cannot set sample format
(%s)\n",
        snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_set_rate_near
(capture_handle, hw_params, &rate, 0)) < 0) {
    fprintf (stderr, "cannot set sample rate
(%s)\n",
```

```
        snd_strerror (err));
    exit (1);
  }
  fprintf(stderr, "rate set to %d\n", rate);

  if ((err = snd_pcm_hw_params_set_channels
(capture_handle, hw_params, 2)) < 0) {
    fprintf (stderr, "cannot set channel count
(%s)\n",
        snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params (capture_handle,
hw_params)) < 0) {
    fprintf (stderr, "cannot set parameters
(%s)\n",
        snd_strerror (err));
    exit (1);
  }

  snd_pcm_hw_params_free (hw_params);

  /*
  if ((err = snd_pcm_prepare (capture_handle))
< 0) {
    fprintf (stderr, "cannot prepare audio
```

```
interface for use (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  */

  while (1) {
    if ((nread = snd_pcm_readi (capture_handle,
buf, BUFSIZE)) < 0) {
      fprintf (stderr, "read from audio
interface failed (%s)\n",
           snd_strerror (err));
      /* recover */
      snd_pcm_prepare(capture_handle);
    } else {
      fwrite(buf, sizeof(short), nread, fout);
    }
  }

  snd_pcm_close (capture_handle);
  exit(0);
}
```

Playback audio from a file

In order to capture or play audio, a device must first be opened as in previous examples. A configuration space is then created and the space is narrowed by setting values on the various parameters. The access type determines if the samples are interleaved or other. The format determines the size of samples and whether little- or big-endian. All of these will return an error if the requested value cannot be set.

Some parameters need care in setting. For example, there is a range of possible values for the sampling rate, but not all of these may be supported. A particular rate may be requested using `snd_pcm_hw_params_set_rate`. But if a requested rate is not possible then an error will be returned. There are several ways of avoiding this:

- Try a number of rates till you get one that is supported
- Test if a rate is supported by `snd_pcm_hw_params_test_rate`
- Request ALSA to give the nearest supported rate by `snd_pcm_hw_params_set_rate_near`. The actual rate chosen is set in the rate parameter
- Instead of a hardware device such as "hw:0" use a plug device such as "plughw:0" which will support many more values by resampling

Finally, once parameters are set for the configuration space, the restricted space is installed onto the device by `snd_pcm_hw_params`

The calls on PCm devices will cause state changes to take place in the device. After opening, the device is in the state `SND_PCM_STATE_OPEN`. After setting the hardware configuration, the device is in the state `SND_PCM_STATE_PREPARE`. Applications can use the `snd_pcm_start` call, write or read data. The state may drop to `SND_PCM_STATE_XRUN` if an overrun or underrun occurs, and then a call to `snd_pcm_prepare` is needed to restore it to `SND_PCM_STATE_PREPARE`.

The call to `readi` reads interlaced data.

The following program is from Paul Davis A Tutorial on Using the ALSA Audio API alsa_playback.c:

```c
/**
 * alsa_playback.c
 */

/**
 * Paul Davis
 * http://equalarea.com/paul/alsa-audio.html#howto
 */

/**
 * Jan Newmarch
 */

#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>

main (int argc, char *argv[])
{
  int i;
  int err;
  int buf[128];
  snd_pcm_t *playback_handle;
  snd_pcm_hw_params_t *hw_params;
  FILE *fin;
```

```
  size_t nread;
  unsigned int rate = 44100;

  if (argc != 3) {
    fprintf(stderr, "Usage: %s card file\n",
argv[0]);
    exit(1);
  }

  if ((err = snd_pcm_open (&playback_handle,
argv[1], SND_PCM_STREAM_PLAYBACK, 0)) < 0) {
    fprintf (stderr, "cannot open audio device
%s (%s)\n",
         argv[1],
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_malloc
(&hw_params)) < 0) {
    fprintf (stderr, "cannot allocate hardware
parameter structure (%s)\n",
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_any
```

```
(playback_handle, hw_params)) < 0) {
    fprintf (stderr, "cannot initialize
hardware parameter structure (%s)\n",
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_set_access
(playback_handle, hw_params,
SND_PCM_ACCESS_RW_INTERLEAVED)) < 0) {
    fprintf (stderr, "cannot set access type
(%s)\n",
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_set_format
(playback_handle, hw_params,
SND_PCM_FORMAT_S16_LE)) < 0) {
    fprintf (stderr, "cannot set sample format
(%s)\n",
         snd_strerror (err));
    exit (1);
  }


  if ((err = snd_pcm_hw_params_set_rate_near
```

```
(playback_handle, hw_params, &rate, 0)) < 0) {
    fprintf (stderr, "cannot set sample rate
(%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("Rate set to %d\n", rate);

  if ((err = snd_pcm_hw_params_set_channels
(playback_handle, hw_params, 2)) < 0) {
    fprintf (stderr, "cannot set channel count
(%s)\n",
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params
(playback_handle, hw_params)) < 0) {
    fprintf (stderr, "cannot set parameters
(%s)\n",
         snd_strerror (err));
    exit (1);
  }

  snd_pcm_hw_params_free (hw_params);

  /*
```

```
  if ((err = snd_pcm_prepare (playback_handle))
< 0) {
    fprintf (stderr, "cannot prepare audio
interface for use (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  */

  if ((fin = fopen(argv[2], "r")) == NULL) {
      fprintf(stderr, "Can't open %s for
reading\n", argv[2]);
      exit(1);
  }

  while ((nread = fread(buf, sizeof(int), 128,
fin)) > 0) {
    //printf("writing\n");
    if ((err = snd_pcm_writei (playback_handle,
buf, nread)) != nread) {
      fprintf (stderr, "write to audio
interface failed (%s)\n",
           snd_strerror (err));
      snd_pcm_prepare(playback_handle);
    }
  }
```

```
  snd_pcm_drain(playback_handle);
  snd_pcm_close (playback_handle);
  exit (0);
}
```

Check that the microphone is enabled using `alsamixer`.
Record by `alsa_capture hw:0 tmp.s16` for example.
Playback by

```
sox -c 2 -r 44100 tmp.s16 tmp.wav
mplayer tmp.wav
```

or by using the next program

```
alsa_playback hw:0 tmp.s16
```

Capturing using callbacks

See A Tutorial on Using the ALSA Audio API

Managing latency

The program /test/latency.c can be run with various parameters to test the latency of your system. *Warning: turn your volume way down low or the feedback might fry your speakers!* For example, on a low setting

```
latency -m 128 -M 128
```

gave a latency of only 0.93 msecs!

The "poor" latency test of

```
latency -m 8192 -M 8192 -t 1 -p
```

gave a latency of 92.9 msecs.

See Low latency howto

Two methods are available to control latency: one is through the ALSA configuration files, the other is programmatically. The following is claimed to work in the configuration file `/etc/asound.conf` :

```
pcm.card0 {
  type hw
  card 0
}

pcm.!default {
  type plug
  slave.pcm "dmixer"
}


pcm.dmixer {
  type dmix
  ipc_key 2048
  slave {
    pcm "hw:0,0"
    period_time 0
    period_size 2048
    buffer_size 65536
    buffer_time 0
    periods 128
    rate 48000
    channels 2
  }
  bindings {
    0 0
```

```
      1 1
    }
  }
```

I haven't tested this yet .

Programmatically you need to set the internal buffer and period sizes using `snd_pcm_hw_params_set_buffer_size_near` and `snd_pcm_hw_params_set_period_size_near` . *I haven't got this to work yet .*

Playback of captured sound

Playback of captured sound involves two handles, possibly for different cards. The direct method of just combining two of these in a loop

```
while (1) {
    int nread;
    if ((nread = snd_pcm_readi (capture_handle,
buf, BUF_SIZE)) != BUF_SIZE) {
      fprintf (stderr, "read from audio
interface failed (%s)\n",
           snd_strerror (nread));
      snd_pcm_prepare(capture_handle);
      continue;
    }

    printf("copying %d\n", nread);

    if ((err = snd_pcm_writei (playback_handle,
buf, nread)) != nread) {
      if (err < 0) {
    fprintf (stderr, "write to audio interface
failed (%s)\n",
         snd_strerror (err));
      } else {
    fprintf (stderr, "write to audio interface
failed after %d frames\n", err);
      }
      snd_pcm_prepare(playback_handle);
    }
}
```

doesn't unfortunately work. On my computer it threw up a variety of errors, including broken pipe, device not ready and device non-existent.

There are many issues that must be addressed in order to playback captured sound directly. The first issue is that each soundcard has its own timing clock. These must be synchronised. This is difficult to maintain for consumer-grade cards as their clocks apparently are low quality and will drift or be erratic. Nevertheless, ALSA will attempt to synchronise clocks by the function `snd_pcm_link` which takes two card handles as parameter.

The next issue is that finer control must be exercised over the buffers and how often ALSA will fill these buffers. This is controlled by two parameters, buffer size and period size (or buffer time and period time). The period size/time controls how often interrupts occur to fill the buffer. Typically, the period size (time) is set to half that of the buffer size (time). Relevant functions are `snd_pcm_hw_params_set_buffer_size_near` and `snd_pcm_hw_params_set_period_size_near`. Corresponding `get` functions can used to discover what values were actually set.

In addition to hardware parameters, ALSA can also set software parameters. The distinction between the two is not clear to me, but anyway, a "start threshold" and "available minimum" have to set as software parameters. I have managed to get working results by setting both of these to the period size, using `snd_pcm_sw_params_set_start_threshold` and `snd_pcm_sw_params_set_avail_min`. Setting software parameters is similar to setting hardware parameters: first a data structure is initialised by `snd_pcm_sw_params_current`, the software space is restricted by setter calls, and then the data is set into the card by `snd_pcm_sw_params`.

ALSA needs to keep the output as full as possible. Otherwise it will generate a "write error". I have no idea why, but it only seems to work if two buffers are written to the playback device before attempts are made to read and copy from the capture device. Sometimes one buffer will do, but no more than two. To avoid extraneous unwanted noise at the beginning of playback, two buffers of silence work well.

The resultant program is playback-capture.c:

```c
/**
 * Jan Newmarch
 */

#define PERIOD_SIZE 1024
#define BUF_SIZE (PERIOD_SIZE * 2)

#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>

void print_pcm_state(snd_pcm_t *handle, char
*name) {
  switch (snd_pcm_state(handle)) {
  case SND_PCM_STATE_OPEN:
    printf("state open %s\n", name);
    break;

  case SND_PCM_STATE_SETUP:
    printf("state setup %s\n", name);
    break;

  case SND_PCM_STATE_PREPARED:
    printf("state prepare %s\n", name);
    break;
```

```
  case SND_PCM_STATE_RUNNING:
    printf("state running %s\n", name);
    break;

  case SND_PCM_STATE_XRUN:
    printf("state xrun %s\n", name);
    break;

  default:
    printf("state other %s\n", name);
    break;

  }
}


int setparams(snd_pcm_t *handle, char *name) {
  snd_pcm_hw_params_t *hw_params;
  int err;


  if ((err = snd_pcm_hw_params_malloc
(&hw_params)) < 0) {
    fprintf (stderr, "cannot allocate hardware
parameter structure (%s)\n",
         snd_strerror (err));
    exit (1);
```

```
  }

  if ((err = snd_pcm_hw_params_any (handle,
hw_params)) < 0) {
    fprintf (stderr, "cannot initialize
hardware parameter structure (%s)\n",
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_set_access
(handle, hw_params,
SND_PCM_ACCESS_RW_INTERLEAVED)) < 0) {
    fprintf (stderr, "cannot set access type
(%s)\n",
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_set_format
(handle, hw_params, SND_PCM_FORMAT_S16_LE)) <
0) {
    fprintf (stderr, "cannot set sample format
(%s)\n",
         snd_strerror (err));
    exit (1);
  }
```

```
  unsigned int rate = 48000;
  if ((err = snd_pcm_hw_params_set_rate_near
(handle, hw_params, &rate, 0)) < 0) {
    fprintf (stderr, "cannot set sample rate
(%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("Rate for %s is %d\n", name, rate);

  if ((err = snd_pcm_hw_params_set_channels
(handle, hw_params, 2)) < 0) {
    fprintf (stderr, "cannot set channel count
(%s)\n",
         snd_strerror (err));
    exit (1);
  }

  snd_pcm_uframes_t buffersize = BUF_SIZE;
  if ((err =
snd_pcm_hw_params_set_buffer_size_near(handle,
hw_params, &buffersize)) < 0) {
    printf("Unable to set buffer size %li:
%s\n", BUF_SIZE, snd_strerror(err));
    exit (1);;
  }
```

```
  snd_pcm_uframes_t periodsize = PERIOD_SIZE;
  fprintf(stderr, "period size now %d\n",
periodsize);
  if ((err =
snd_pcm_hw_params_set_period_size_near(handle,
hw_params, &periodsize, 0)) < 0) {
    printf("Unable to set period size %li:
%s\n", periodsize, snd_strerror(err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params (handle,
hw_params)) < 0) {
    fprintf (stderr, "cannot set parameters
(%s)\n",
         snd_strerror (err));
    exit (1);
  }

  snd_pcm_uframes_t p_psize;
  snd_pcm_hw_params_get_period_size(hw_params,
&p_psize, NULL);
  fprintf(stderr, "period size %d\n", p_psize);

  snd_pcm_hw_params_get_buffer_size(hw_params,
&p_psize);
```

```
  fprintf(stderr, "buffer size %d\n", p_psize);

  snd_pcm_hw_params_free (hw_params);

  if ((err = snd_pcm_prepare (handle)) < 0) {
    fprintf (stderr, "cannot prepare audio
interface for use (%s)\n",
         snd_strerror (err));
    exit (1);
  }

  return 0;
}

int set_sw_params(snd_pcm_t *handle, char
*name) {
  snd_pcm_sw_params_t *swparams;
  int err;

  snd_pcm_sw_params_alloca(&swparams);

  err = snd_pcm_sw_params_current(handle,
swparams);
  if (err < 0) {
    fprintf(stderr, "Broken configuration for
this PCM: no configurations available\n");
    exit(1);
```

```
  }

  err =
snd_pcm_sw_params_set_start_threshold(handle,
swparams, PERIOD_SIZE);
  if (err < 0) {
    printf("Unable to set start threshold:
%s\n", snd_strerror(err));
    return err;
  }
  err = snd_pcm_sw_params_set_avail_min(handle,
swparams, PERIOD_SIZE);
  if (err < 0) {
    printf("Unable to set avail min: %s\n",
snd_strerror(err));
    return err;
  }

  if (snd_pcm_sw_params(handle, swparams) < 0)
{
    fprintf(stderr, "unable to install sw
params:\n");
    exit(1);
  }

  return 0;
}
```

```
/************** some code from latency.c
****************/

main (int argc, char *argv[])
{
  int i;
  int err;
  int buf[BUF_SIZE];
  snd_pcm_t *playback_handle;
  snd_pcm_t *capture_handle;
  snd_pcm_hw_params_t *hw_params;
  FILE *fin;
  size_t nread;
  snd_pcm_format_t format =
SND_PCM_FORMAT_S16_LE;
  if (argc != 3) {
    fprintf(stderr, "Usage: %s in-card out-
card\n", argv[0]);
    exit(1);
  }

  /**** Out card *******/
  if ((err = snd_pcm_open (&playback_handle,
argv[2], SND_PCM_STREAM_PLAYBACK, 0)) < 0) {
    fprintf (stderr, "cannot open audio device
%s (%s)\n",
```

```
        argv[2],
        snd_strerror (err));
    exit (1);
  }

  setparams(playback_handle, "playback");
  set_sw_params(playback_handle, "playback");


  /*********** In card **********/

  if ((err = snd_pcm_open (&capture_handle,
argv[1], SND_PCM_STREAM_CAPTURE, 0)) < 0) {
    fprintf (stderr, "cannot open audio device
%s (%s)\n",
        argv[1],
        snd_strerror (err));
    exit (1);
  }

  setparams(capture_handle, "capture");
  set_sw_params(capture_handle, "capture");

  if ((err = snd_pcm_link(capture_handle,
playback_handle)) < 0) {
    printf("Streams link error: %s\n",
snd_strerror(err));
```

```
    exit(0);
  }

  if ((err = snd_pcm_prepare (playback_handle))
< 0) {
    fprintf (stderr, "cannot prepare playback
audio interface for use (%s)\n",
         snd_strerror (err));
    exit (1);
  }

  /**************** stuff something into the
playback buffer ****************/
  if (snd_pcm_format_set_silence(format, buf,
2*BUF_SIZE) < 0) {
    fprintf(stderr, "silence error\n");
    exit(1);
  }

  int n = 0;
  while (n++ < 2) {
    if (snd_pcm_writei (playback_handle, buf,
BUF_SIZE) < 0) {
      fprintf(stderr, "write error\n");
      exit(1);
    }
  }
```

```
  /************* COPY ************/
  while (1) {
    int nread;
    if ((nread = snd_pcm_readi (capture_handle,
buf, BUF_SIZE)) != BUF_SIZE) {
      if (nread < 0) {
    fprintf (stderr, "read from audio interface
failed (%s)\n",
         snd_strerror (nread));
      } else {
    fprintf (stderr, "read from audio interface
failed after %d frames\n", nread);
      }
      snd_pcm_prepare(capture_handle);
      continue;
    }

    if ((err = snd_pcm_writei (playback_handle,
buf, nread)) != nread) {
      if (err < 0) {
    fprintf (stderr, "write to audio interface
failed (%s)\n",
         snd_strerror (err));
      } else {
    fprintf (stderr, "write to audio interface
failed after %d frames\n", err);
```

```
        }
        snd_pcm_prepare(playback_handle);
      }
    }


    snd_pcm_drain(playback_handle);
    snd_pcm_close (playback_handle);
    exit (0);
  }
```

Mixing audio

Mixing using dmix

ALSA contains a plugin `dmix` which is enabled by default. This performs mixing of multiple audio input signals into an output signal in software. A description of this is given in The Dmix Howto . Basically, each application that wishes to write audio to ALSA should use the plugin `plug:dmix` instead of a hardware device such as `hw:0` . For example, the `alsa_playback` program discussed earlier can be called multiple times and have the ALSA inputs mixed together as in

```
alsa_playback plug:dmix tmp1.s16 &
alsa_playback plug:dmix tmp2.s16 &
alsa_playback plug:dmix tmp3.s16
```

Mixing using PulseAudio

PulseAudio isn't covered until the mext chapter, because it is generally considered to be a sound server, acting in the layer *above* ALSA. However, there is also an ALSA plugin module

whereby PulseAudio can appear as a plugin device *below* ALSA! So ALSA can write output to the PulseAudio plugin, which can process it using the full capabilities of PulseAudio, which then feeds it back down into ALSA for rendering on a hardware device.

One of these capabilities is that PulseAudio contains a mixer. So two (or more) applications can send audio to the PulseAudio plugin which will then mix the signals and then send them back to ALSA.

The PulseAudio plugin can appear as the PCM devices `pulse` or as `default`. So the following three outputs will be mixed by PulseAudio and rendered by ALSA.

```
alsa_playback default tmp1.s16 &
alsa_playback pulse tmp2.s16 &
alsa_playback default tmp3.s16
```

Simple mixer API - volume control

ALSA has a separate API for the mixer module. In fact, there are two: the [asynchronous] Mixer Interface and the Simple Mixer Interface . For now, we shall just consider the simple interface.

The ALSA mixer does not have a great deal of functionality apart from mixing. Basically, it can get and set volumes on channels or globally. Setting the volume is illustrated by the following program, based on a function by trenki at Set ALSA master volume from C code :

```c
/*
 *
 * From
http://stackoverflow.com/questions/6787318/set-
alsa-master-volume-from-c-code
 */

#include <alsa/asoundlib.h>
#include <alsa/mixer.h>



void SetAlsaMasterVolume(long volume)
{
    long min, max;
    snd_mixer_t *handle;
    snd_mixer_selem_id_t *sid;
    const char *card = "default";
    const char *selem_name = "Master";

    snd_mixer_open(&handle, 0);
    snd_mixer_attach(handle, card);
    snd_mixer_selem_register(handle, NULL,
NULL);
    snd_mixer_load(handle);
```

```
    snd_mixer_selem_id_alloca(&sid);
    snd_mixer_selem_id_set_index(sid, 0);
    snd_mixer_selem_id_set_name(sid,
selem_name);
    snd_mixer_elem_t* elem =
snd_mixer_find_selem(handle, sid);


snd_mixer_selem_get_playback_volume_range(elem,
&min, &max);

snd_mixer_selem_set_playback_volume_all(elem,
volume * max / 100);

    snd_mixer_close(handle);
}


main(int argc, char **args) {
    long volume = 0;
    char buf[128];

    while (1) {
    puts("Enter an integer 0-100\n");
    fgets(buf, 128, stdin);
    volume = atoi(buf);
    SetAlsaMasterVolume(volume);
```

```
    }
    exit(0);
}
```

Writing an ALSA device driver

If you need to write a device driver for a new soundcard, see Writing an ALSA Driver by Takashi Iwai

Conclusion

ALSA is currently the lowest level of the audio stacks for Linux that is included in the kernel. It supplies device drivers with a standard API to access the different sound devices and cards. There are a variety of user level tools to access and manipulate the devices, built using this API.

This chapter has looked at the user level tools and at building your own tools using the API. There is a pointer to building device drivers.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

PulseAudio is a sound server, sitting above device drivers such as ALSA or OSS. It offers more capablilities than device drivers. PulseAudio is designed for consumer audio, to make it easy to use sound on desktops, laptops and mobile devices. Multiple sources of sound can all play to the PulseAudio server and it will mix them together and play them. Low latency is not a design goal, so it is unsuitable for professional audio.

Resources

- PulseAudio home page
- PulseAudio and Jack by Lennart Poettering
- Pro Audio is Easy, Consumer Audio is Hard - 60 minute talk by Lennart Poettering
- PulseAudio API Documentation

Starting, stopping and pausing PulseAudio

If you have a current Linux system, PulseAudio is probably running. Test by running this from the command line:

```
ps agx | grep pulse
```

If you see a line like "/usr/bin/pulseaudio --start --log-target=syslog" then it is running already.

If it isn't running and you have it installed, then start it by

```
pulseaudio --start
```

Stopping PulseAudio isn't so easy - Carla Schroder shows how . The basic problem is that PulseAudio is set to respawn itself after it is killed. You have to turn that off by editing /etc/pulse/client.conf, changing autospawn = yes to autospawn = no, and setting daemon-binary to /bin/true. Then you can kill the processs, remove it from startup files, etc.

If you want to run another sound system (such as Jack) for a short while, you may just want to pause PulseAudio. You do this by using `pasuspender`. This takes a command (after "--") and will pause access by the PulseAudio server to the audio devices until the subcommand has finished. For example,

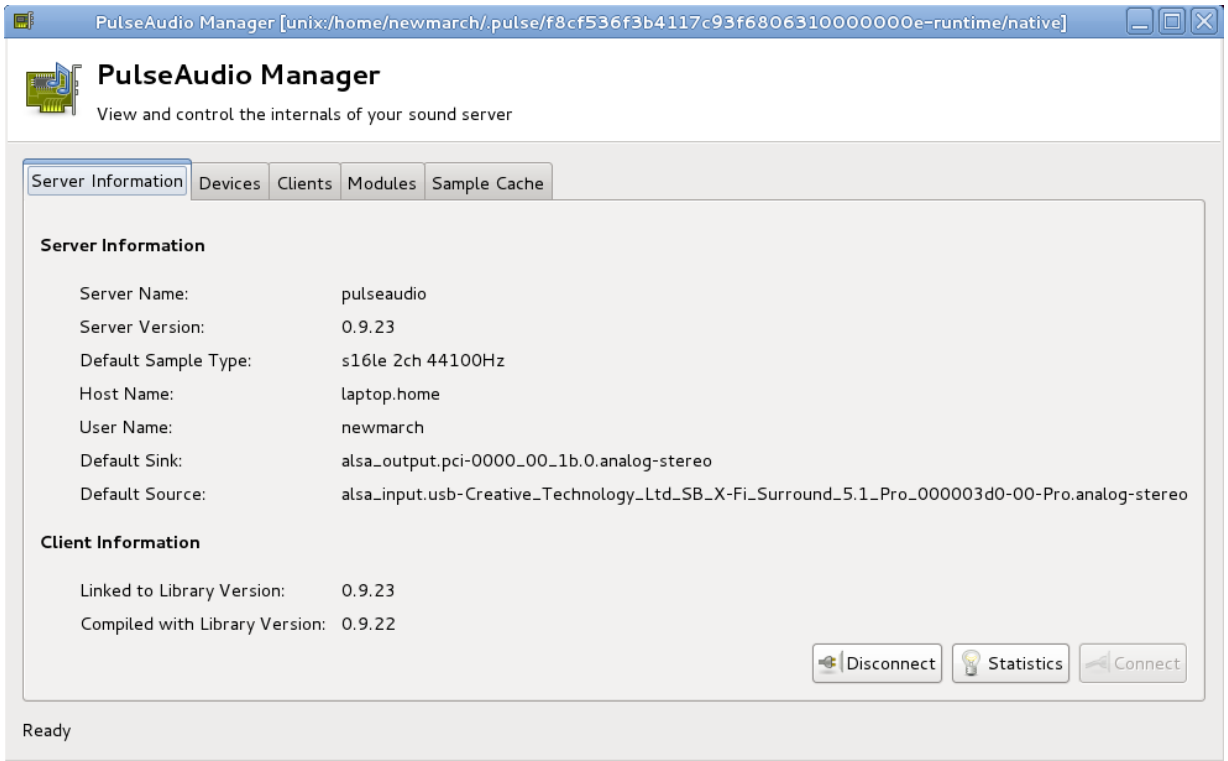
```
pasuspender -- jackd
```

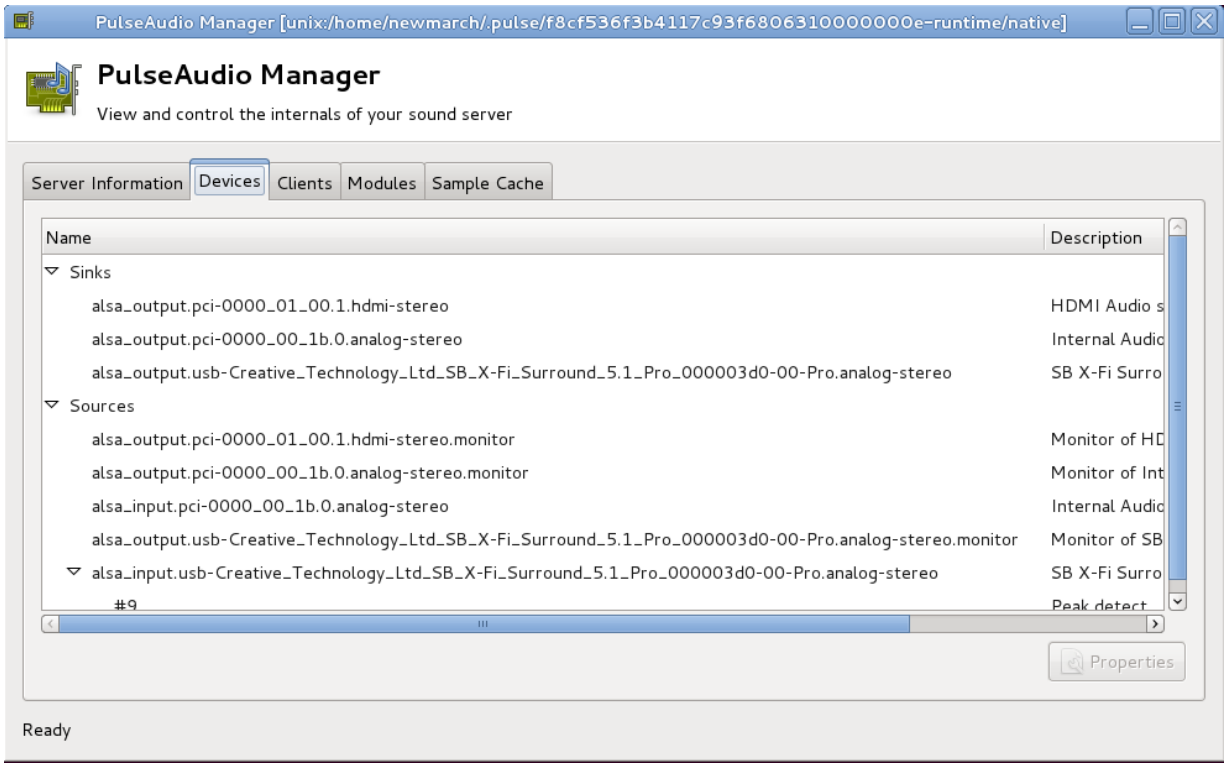
will run the Jack server, with PulseAudio getting out of the way until it has finished.

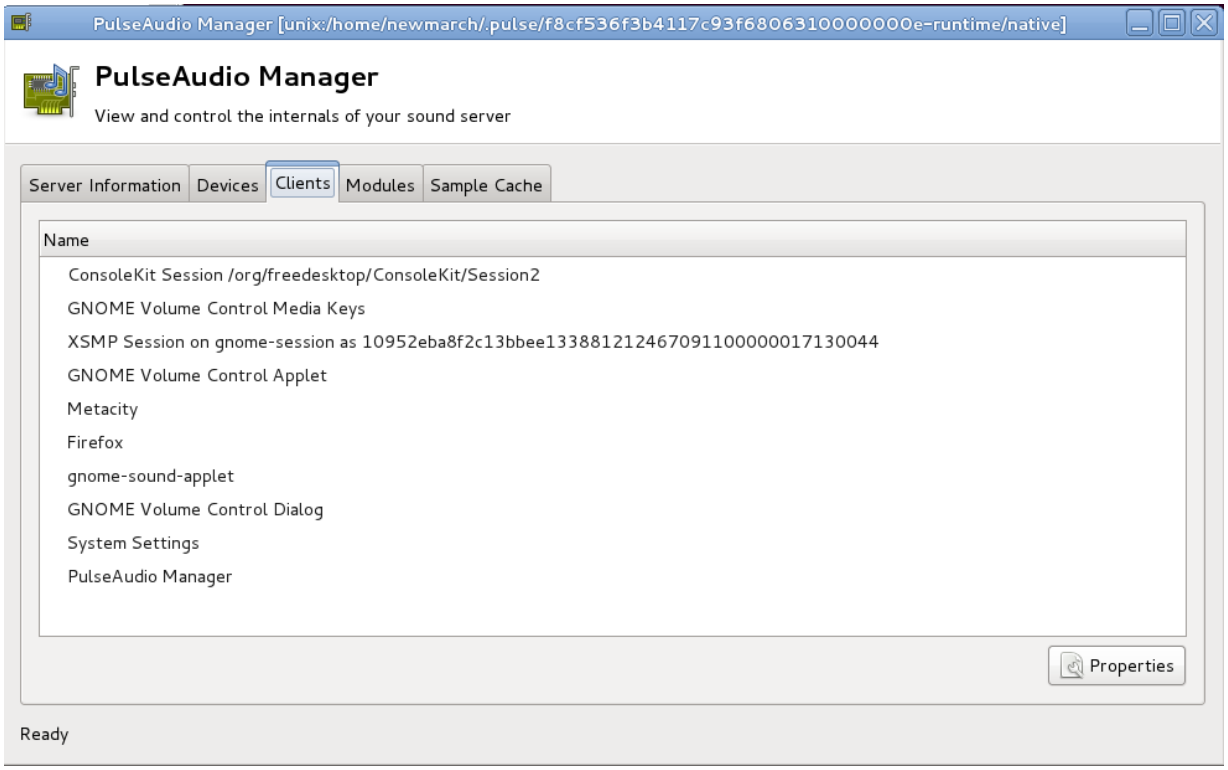
User space tools

paman

This shows you information about the PulseAudio server, its devices and clients. The following three screen dumps show the type of information it gives.

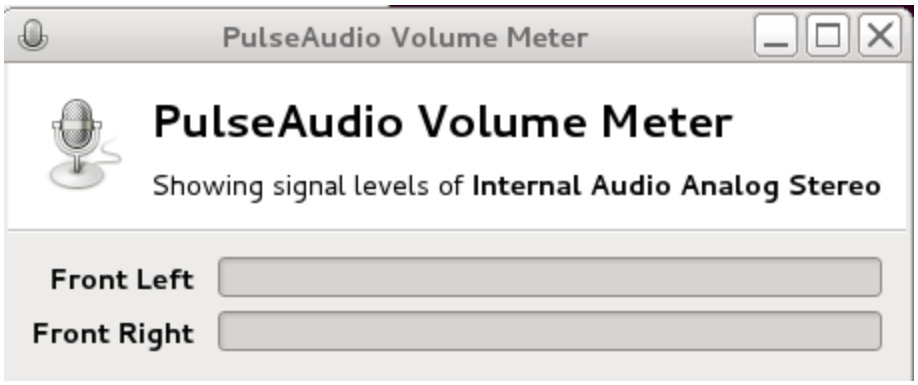



PulseAudio Manager [unix:/home/newmarch/.pulse/f8cf536f3b4117c93f6806310000000e-runtime/native]
PulseAudio Manager
View and control the internals of your sound server
Server Information
Devices
Clients
Modules
Sample Cache
Name
Description
Sinks
alsa_output.pci-0000_01_00.1.hdmi-stereo
HDMI Audio s
alsa_output.pci-0000_00_1b.0.analog-stereo
Internal Audio
alsa_output.usb-Creative_Technology_Ltd_SB_X-Fi_Surround_5.1_Pro_000003d0-00-Pro.analog-stereo
SB X-Fi Surro
Sources
alsa_output.pci-0000_01_00.1.hdmi-stereo.monitor
Monitor of HD
alsa_output.pci-0000_00_1b.0.analog-stereo.monitor
Monitor of Int
alsa_input.pci-0000_00_1b.0.analog-stereo
Internal Audio
alsa_output.usb-Creative_Technology_Ltd_SB_X-Fi_Surround_5.1_Pro_000003d0-00-Pro.analog-stereo.monitor
Monitor of SB
alsa_input.usb-Creative_Technology_Ltd_SB_X-Fi_Surround_5.1_Pro_000003d0-00-Pro.analog-stereo
SB X-Fi Surro
#9
Peak detect
Properties
Ready

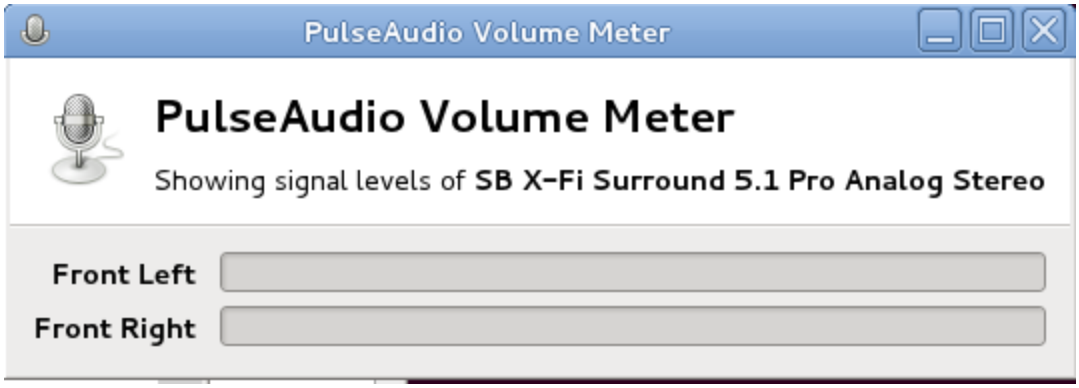

PulseAudio Manager [unix:/home/newmarch/.pulse/f8cf536f3b4117c93f6806310000000e-runtime/native]
PulseAudio Manager
View and control the internals of your sound server
Server Information
Devices
Clients
Modules
Sample Cache
Name
ConsoleKit Session /org/freedesktop/ConsoleKit/Session2
GNOME Volume Control Media Keys
XSMP Session on gnome-session as 10952eba8f2c13bbee1338812124670911000000017130044
GNOME Volume Control Applet
Metacity
Firefox
gnome-sound-applet
GNOME Volume Control Dialog
System Settings
PulseAudio Manager
Properties
Ready

pavumeter

`pavumeter` is a simple meter for showing input or output levels of the default devices. When run by `pavumeter` it shows the playback device as in

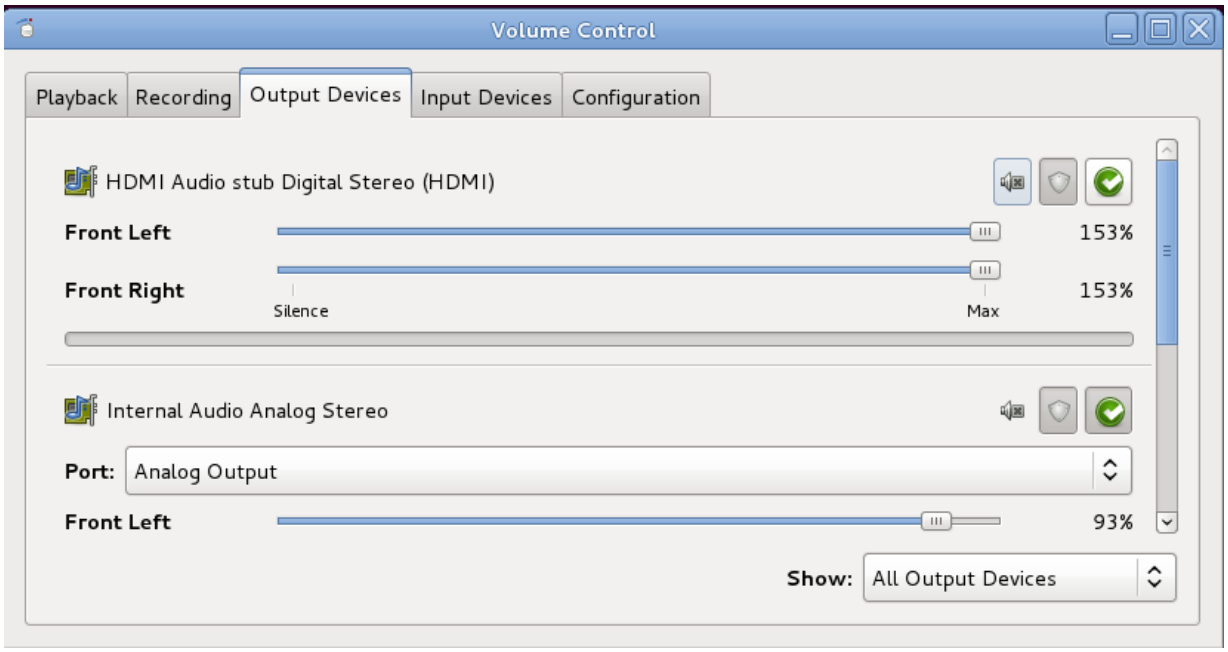


while if it is run by `pavumeter --record` it shows the record device as in



pavucontrol

`pavucontrol` allows you to control the input and ouput volumes of the different connected audio devices:



With these tabs, `pavucontrol` is a device-level mixer, able to control the overall volume to individual devices.

One of the special advantages of PulseAudio is that it can perform application-level mixing. If two audio sources write to the same PulseAudio device, the audio will be mixed to the output device. `pavucontrol` can show the multiple applications using the Playback tab, showing all applications or all streams currently being mixed. Each stream can have its channel volumes individually controlled.

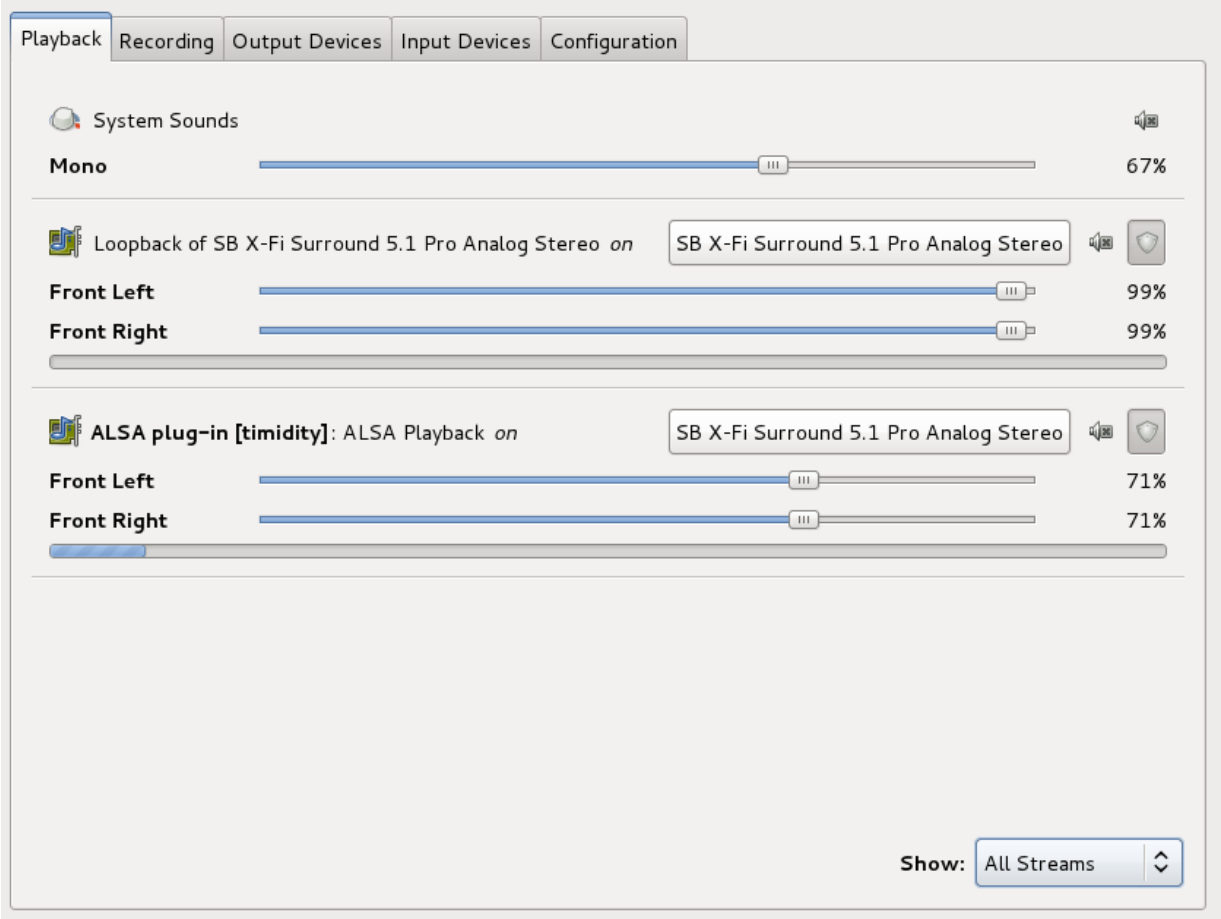
For example, Karaoke on the cheap can be done by setting the straight-through module for the microphone to speaker by

```
pactl load-module module-loopback
latency_msec=1
```

while the Karoake file is played by a Karaoke player such as `kmid` through `timidity` e.g.

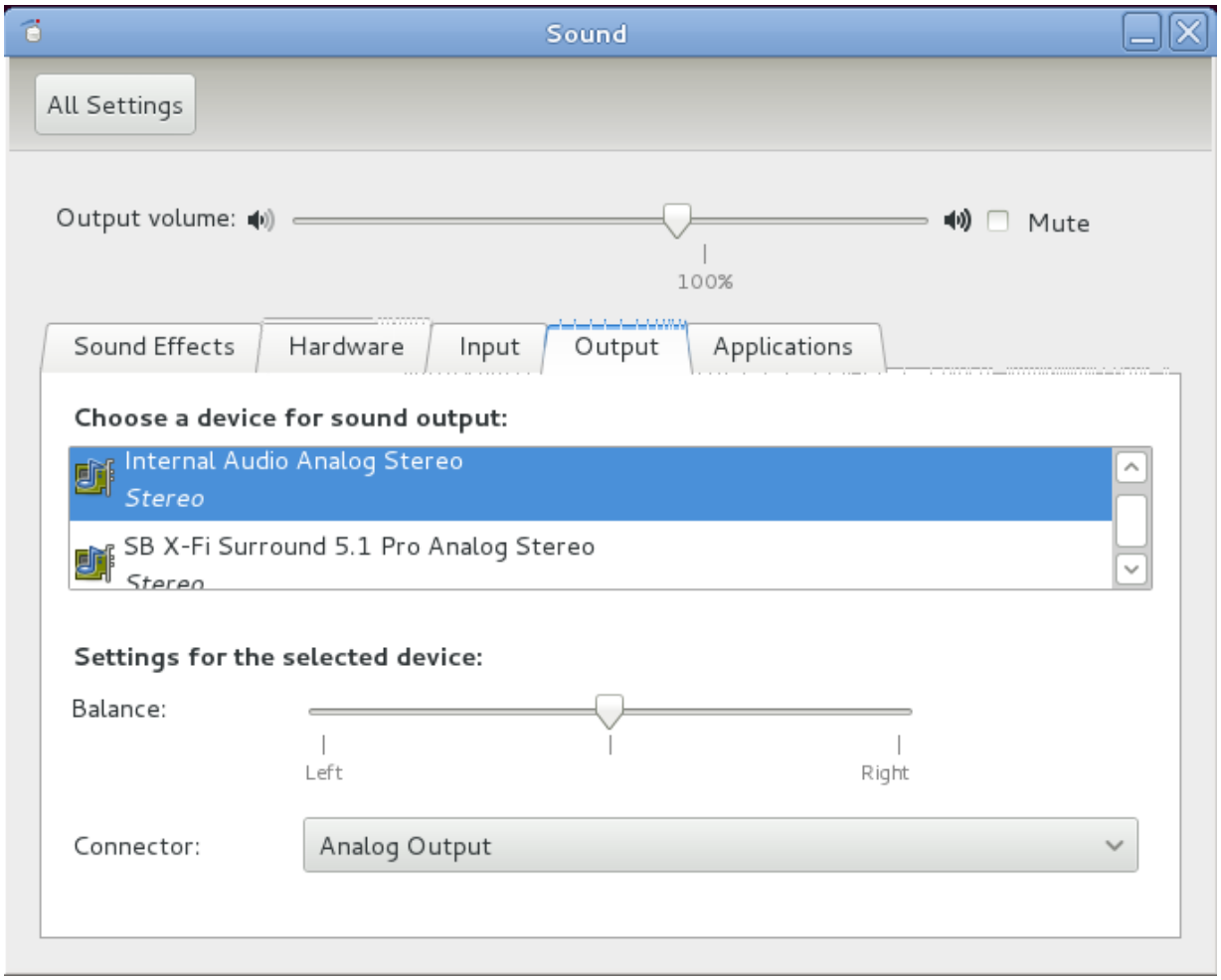
```
kmid nightsin.kar
```

While these two are running, relative volumes can be controlled by use of `pavucontrol`:



Gnome control center (sound)

The `gnome-control-center sound` allows full view and control of the attached sound devices, including selection of the default input and output devices. It looks like



parec/paplay/pacat

These are command line tools to record and playback sound files. They are all symbolic links to the same code, just differently named links. The default format is PCM s16. There are many options, but they don't always do quite what you want them to. For example, to play from the default record device to the default playback device with minimum latency,

```
pacat -r --latency-msec=1 | pacat -p --latency-
msec=1
```

This actually has a latency of about 50 msec.

pactl/pacmd

These two commands do basically the same thing. `pacmd` is the interactive version with more options. For example `pacmd` with the command `list-sinks` includes

```
name: <alsa_output.pci-0000_00_1b.0.analog-
stereo>
    driver: <module-alsa-card.c>
    flags: HARDWARE HW_MUTE_CTRL HW_VOLUME_CTRL
DECIBEL_VOLUME LATENCY FLAT_VOLUME
DYNAMIC_LATENCY
    state: SUSPENDED
    suspend cause: IDLE
    priority: 9959
    volume: 0:  93% 1:  93%
            0: -1.88 dB 1: -1.88 dB
            balance 0.00
    base volume: 100%
                 0.00 dB
    volume steps: 65537
    muted: no
    current latency: 0.00 ms
    max request: 0 KiB
    max rewind: 0 KiB
    monitor source: 1
    sample spec: s16le 2ch 44100Hz
    channel map: front-left,front-right
                 Stereo
    used by: 0
    linked by: 0
    configured latency: 0.00 ms; range is 16.00
```

```
.. 2000.00 ms
    card: 1 <alsa_card.pci-0000_00_1b.0>
    module: 5
    properties:
        alsa.resolution_bits = "16"
        device.api = "alsa"
        device.class = "sound"
        alsa.class = "generic"
        alsa.subclass = "generic-mix"
        alsa.name = "STAC92xx Analog"
        alsa.id = "STAC92xx Analog"
        alsa.subdevice = "0"
        alsa.subdevice_name = "subdevice #0"
        alsa.device = "0"
        alsa.card = "0"
        alsa.card_name = "HDA Intel PCH"
        alsa.long_card_name = "HDA Intel PCH at
0xe6e60000 irq 47"
        alsa.driver_name = "snd_hda_intel"
        device.bus_path = "pci-0000:00:1b.0"
        sysfs.path =
"/devices/pci0000:00/0000:00:1b.0/sound/card0"
        device.bus = "pci"
        device.vendor.id = "8086"
        device.vendor.name = "Intel
Corporation"
        device.product.id = "1c20"
```

```
        device.product.name = "6 Series/C200
Series Chipset Family High Definition Audio
Controller"
        device.form_factor = "internal"
        device.string = "front:0"
        device.buffering.buffer_size = "352800"
        device.buffering.fragment_size =
"176400"
        device.access_mode = "mmap+timer"
        device.profile.name = "analog-stereo"
        device.profile.description = "Analog
Stereo"
        device.description = "Internal Audio
Analog Stereo"
        alsa.mixer_name = "IDT 92HD90BXX"
        alsa.components =
"HDA:111d76e7,10280494,00100102"
        module-udev-detect.discovered = "1"
        device.icon_name = "audio-card-pci"
    ports:
        analog-output: Analog Output (priority
9900)
        analog-output-headphones: Analog
Headphones (priority 9000)
    active port: <analog-output>
```

Device names

PulseAudio uses its own naming conventions. The names of source devices (such as microphones) can be found using code from the PulseAudio FAQ :

```
pactl list | grep -A2 'Source #' | grep 'Name:
.*\.monitor$' | cut -d" " -f2
```

On my system this produces

```
alsa_output.pci-0000_01_00.1.hdmi-
stereo.monitor
alsa_output.pci-0000_00_1b.0.analog-
stereo.monitor
alsa_input.pci-0000_00_1b.0.analog-stereo
```

Similarly the output devices are found by

```
pactl list | grep -A2 'Sink #' | grep 'Name:
.*\.monitor$' | cut -d" " -f2
```

to give

```
alsa_output.pci-0000_01_00.1.hdmi-stereo
alsa_output.pci-0000_00_1b.0.analog-stereo
```

Loopback module

Using `pactl` you can load the module `module-loopback` by

```
pactl load-module module-loopback
latency_msec=1
```

When loaded, sound is internally routed from the input device to the output device. The latency is effectively zero.

If you load this module into, say. your laptop, be careful about unplugging speakers, microphones, etc. The internal speaker and microphone are close enough to set up a feedback loop. Unload module number N by

```
pactl unload-module N
```

(if you have forgotten the module number, just running `pactl` will list all modules so you can identify the loopback module.).

PulseAudio and ALSA

Output from `pacmd` shows PulseAudio uses ALSA. The relationship is deeper: the default ALSA device is "hw:0" but PulseAudio overrides that. In `/etc/asound.conf` is a hook to load `/etc/alsa/pulse-default.conf` and this contains

```
pcm.!default {
    type pulse
    hint {
        description "Default"
    }
}
```

which replaces the default device with a PulseAudio module.

Opening the default ALSA device will actually call into PulseAudio which will then call back into ALSA with the devices it chooses.

Programming with PulseAudio

The source for Pulse Audio and its documentation is PulseAudio 2.0 There are two ways of programming with PulseAudio: the "simple API" and the "asynchronous API". Both are described below.

Simple API

Pulse has a "simple" API and a far more complex asynchronous API. The simple API may be good enough for your needs.

The simple API has a small set of functions

```
pa_simple *     pa_simple_new (const char
*server, const char *name,
pa_stream_direction_t dir, const char *dev,
const char *stream_name, const pa_sample_spec
*ss, const pa_channel_map *map, const
pa_buffer_attr *attr, int *error)
     Create a new connection to the server.
void     pa_simple_free (pa_simple *s)
     Close and free the connection to the
server.
int     pa_simple_write (pa_simple *s, const
void *data, size_t bytes, int *error)
     Write some data to the server.
int     pa_simple_drain (pa_simple *s, int
*error)
     Wait until all data already written is
played by the daemon.
int     pa_simple_read (pa_simple *s, void
*data, size_t bytes, int *error)
     Read some data from the server.
pa_usec_t     pa_simple_get_latency (pa_simple
*s, int *error)
     Return the playback latency.
int     pa_simple_flush (pa_simple *s, int
*error)
     Flush the playback buffer.
```

Play a file

A program to play from a file to the default output device is from the PulseAudio site. The basic structure is

- Create a new playback stream (pa_simple_new)
- Read blocks from the file (read)...
- ...and write them to the stream (pa_simple_write)
- Finish by flushing the stream (pa_simple_drain)

The program is pacat-simple.c . Rather weirdly, it does a `dup2` to map the open file descriptor onto `stdin` and then reads from `stdin` . This isn't necessary - what not just read from the original file descriptor?

```
/***
 *   This file is part of PulseAudio.
 *
 *   PulseAudio is free software; you can
redistribute it and/or modify
 *   it under the terms of the GNU Lesser
General Public License as published
 *   by the Free Software Foundation; either
version 2.1 of the License,
 *   or (at your option) any later version.
 *
 *   PulseAudio is distributed in the hope that
it will be useful, but
 *   WITHOUT ANY WARRANTY; without even the
implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE. See the GNU
 *   General Public License for more details.
 *
 *   You should have received a copy of the GNU
Lesser General Public License
 *   along with PulseAudio; if not, write to
the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite
330, Boston, MA 02111-1307
 *   USA.
```

```
 ****/

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>

#include <pulse/simple.h>
#include <pulse/error.h>

#define BUFSIZE 1024

int main(int argc, char*argv[]) {

    // set to NULL for default output device
    char *device = "alsa_output.pci-
0000_00_1b.0.analog-stereo";

    /* The Sample format to use */
    static const pa_sample_spec ss = {
        .format = PA_SAMPLE_S16LE,
        .rate = 44100,
```

```
        .channels = 2
    };

    pa_simple *s = NULL;
    int ret = 1;
    int error;

    /* replace STDIN with the specified file if
needed */
    if (argc > 1) {
        int fd;

        if ((fd = open(argv[1], O_RDONLY)) < 0)
{
            fprintf(stderr, __FILE__": open()
failed: %s\n", strerror(errno));
            goto finish;
        }

        if (dup2(fd, STDIN_FILENO) < 0) {
            fprintf(stderr, __FILE__": dup2()
failed: %s\n", strerror(errno));
            goto finish;
        }

        close(fd);
    }
```

```
    /* Create a new playback stream */
    if (!(s = pa_simple_new(NULL, argv[0],
PA_STREAM_PLAYBACK, device, "playback", &ss,
NULL, NULL, &error))) {
        fprintf(stderr, __FILE__":
pa_simple_new() failed: %s\n",
pa_strerror(error));
        goto finish;
    }

    for (;;) {
        uint8_t buf[BUFSIZE];
        ssize_t r;

#if 1
        pa_usec_t latency;

        if ((latency = pa_simple_get_latency(s,
&error)) == (pa_usec_t) -1) {
            fprintf(stderr, __FILE__":
pa_simple_get_latency() failed: %s\n",
pa_strerror(error));
            goto finish;
        }

        fprintf(stderr, "%0.0f usec    \r",
```

```
(float)latency);
#endif

        /* Read some data ... */
        if ((r = read(STDIN_FILENO, buf,
sizeof(buf))) <= 0) {
            if (r == 0) /* EOF */
                break;

            fprintf(stderr, __FILE__": read()
failed: %s\n", strerror(errno));
            goto finish;
        }

        /* ... and play it */
        if (pa_simple_write(s, buf, (size_t) r,
&error) < 0) {
            fprintf(stderr, __FILE__":
pa_simple_write() failed: %s\n",
pa_strerror(error));
            goto finish;
        }
    }

    /* Make sure that every single sample was
played */
    if (pa_simple_drain(s, &error) < 0) {
```

```
        fprintf(stderr, __FILE__":
pa_simple_drain() failed: %s\n",
pa_strerror(error));
        goto finish;
    }

    ret = 0;

finish:

    if (s)
        pa_simple_free(s);

    return ret;
}
```

Record to a file

A program to record to a file from the default input device is from the Pulse Audio site parec-simple.c The basic structure is

- Create a new recording stream (pa_simple_new)
- Read blocks from the stream (pa_simple_read)...
- ...and write them to the output (write)
- Finish by releasing the stream (pa_simple_free)

Note that you need to tell PulseAudio the format to write the data, using a pa_sample_spec. Two channel, 44100hz and PCM 16 bit little-endian is chosen.

```
/***
  This file is part of PulseAudio.

  PulseAudio is free software; you can
redistribute it and/or modify
  it under the terms of the GNU Lesser General
Public License as published
  by the Free Software Foundation; either
version 2.1 of the License,
  or (at your option) any later version.

  PulseAudio is distributed in the hope that it
will be useful, but
  WITHOUT ANY WARRANTY; without even the
implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE. See the GNU
  General Public License for more details.

  You should have received a copy of the GNU
Lesser General Public License
  along with PulseAudio; if not, write to the
Free Software
  Foundation, Inc., 59 Temple Place, Suite 330,
Boston, MA 02111-1307
  USA.
```

```
***/

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>

#include <pulse/simple.h>
#include <pulse/error.h>

#define BUFSIZE 1024

/* A simple routine calling UNIX write() in a
loop */
static ssize_t loop_write(int fd, const
void*data, size_t size) {
    ssize_t ret = 0;

    while (size > 0) {
        ssize_t r;

        if ((r = write(fd, data, size)) < 0)
            return r;
```

```
        if (r == 0)
            break;

        ret += r;
        data = (const uint8_t*) data + r;
        size -= (size_t) r;
    }

    return ret;
}

int main(int argc, char*argv[]) {
    /* The sample type to use */
    static const pa_sample_spec ss = {
        .format = PA_SAMPLE_S16LE,
        .rate = 44100,
        .channels = 2
    };
    pa_simple *s = NULL;
    int ret = 1;
    int error;

    /* Create the recording stream */
    if (!(s = pa_simple_new(NULL, argv[0],
PA_STREAM_RECORD, NULL, "record", &ss, NULL,
NULL, &error))) {
```

```
        fprintf(stderr, __FILE__":
pa_simple_new() failed: %s\n",
pa_strerror(error));
        goto finish;
    }

    for (;;) {
        uint8_t buf[BUFSIZE];

        /* Record some data ... */
        if (pa_simple_read(s, buf, sizeof(buf),
&error) < 0) {
            fprintf(stderr, __FILE__":
pa_simple_read() failed: %s\n",
pa_strerror(error));
            goto finish;
        }

        /* And write it to STDOUT */
        if (loop_write(STDOUT_FILENO, buf,
sizeof(buf)) != sizeof(buf)) {
            fprintf(stderr, __FILE__": write()
failed: %s\n", strerror(errno));
            goto finish;
        }
    }
```

```
    ret = 0;

finish:

    if (s)
        pa_simple_free(s);

    return ret;
}
```

The output from this is a PCM s16 file. You can convert it to another format using Sox (e.g. sox -c 2 -r 44100 tmp.s16 tmp.wav), or import it as raw data into Audacity and play it directly.

How good are these for real-time audio? The first program can show the latency (turn the "#if 0" to "#if 1"). This code can also be copied into the second one. The results are not good:

- recording has a latency of 11 msecs on my laptop
- playback has a latency of 130 msecs!

Play from source to sink

You can combine the two programs to copy from the microphone to the speaker using a record and a playback stream. The program is pa-mic-2-speaker-simple.c :

```c
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>

#include <pulse/simple.h>
#include <pulse/error.h>

#define BUFSIZE 32

int main(int argc, char*argv[]) {

    /* The Sample format to use */
    static const pa_sample_spec ss = {
        .format = PA_SAMPLE_S16LE,
        .rate = 44100,
        .channels = 2
    };

    pa_simple *s_in, *s_out = NULL;
    int ret = 1;
```

```
    int error;


    /* Create a new playback stream */
    if (!(s_out = pa_simple_new(NULL, argv[0],
PA_STREAM_PLAYBACK, NULL, "playback", &ss,
NULL, NULL, &error))) {
        fprintf(stderr, __FILE__":
pa_simple_new() failed: %s\n",
pa_strerror(error));
        goto finish;
    }

      if (!(s_in = pa_simple_new(NULL, argv[0],
PA_STREAM_RECORD, NULL, "record", &ss, NULL,
NULL, &error))) {
        fprintf(stderr, __FILE__":
pa_simple_new() failed: %s\n",
pa_strerror(error));
        goto finish;
    }

    for (;;) {
        uint8_t buf[BUFSIZE];
        ssize_t r;

#if 1
```

```
        pa_usec_t latency;

        if ((latency =
pa_simple_get_latency(s_in, &error)) ==
(pa_usec_t) -1) {
            fprintf(stderr, __FILE__":
pa_simple_get_latency() failed: %s\n",
pa_strerror(error));
            goto finish;
        }

        fprintf(stderr, "In:  %0.0f usec
\r\n", (float)latency);

        if ((latency =
pa_simple_get_latency(s_out, &error)) ==
(pa_usec_t) -1) {
            fprintf(stderr, __FILE__":
pa_simple_get_latency() failed: %s\n",
pa_strerror(error));
            goto finish;
        }

        fprintf(stderr, "Out: %0.0f usec
\r\n", (float)latency);
#endif
```

```
        if (pa_simple_read(s_in, buf,
sizeof(buf), &error) < 0) {

            fprintf(stderr, __FILE__": read()
failed: %s\n", strerror(errno));
            goto finish;
        }

        /* ... and play it */
        if (pa_simple_write(s_out, buf,
sizeof(buf), &error) < 0) {
            fprintf(stderr, __FILE__":
pa_simple_write() failed: %s\n",
pa_strerror(error));
            goto finish;
        }
    }

    /* Make sure that every single sample was
played */
    if (pa_simple_drain(s_out, &error) < 0) {
        fprintf(stderr, __FILE__":
pa_simple_drain() failed: %s\n",
pa_strerror(error));
        goto finish;
    }
```

```
    ret = 0;

finish:

    if (s_in)
        pa_simple_free(s_in);
    if (s_out)
        pa_simple_free(s_out);

    return ret;
}
```

Try running this and you will discover that the the latency is noticeable and unsatisfactory.

Asynchronous API

The simple API is ... simple. By contrast, the asynchronous API is large and complex. There are also very few examples of using this API.

Nearly all interaction with this API is asynchronous. A call is made to the PulseAudio server and when the response is ready, a library invokes a callback function that you will have passed to it when making the library call. This avoids the need for user code to either block or make polling calls.

The essential structure is

- Create a PulseAudio mainloop (synchronous - pa_mainloop_new)
- Get the mainloop API object, a table of mainloop functions (synchronous - pa_mainloop_get_api)
- Get a context object to talk to the PulseAudio server (synchronous - pa_context_new)
- Establish a connection to the PulseAudio server. This is asynchronous - pa_context_connect
- Register a callback for context state changes from the server
- pa_context_set_state_callback

- Commence the event processing loop - (pa_mainloop_run
- Within the context state callback, determine what state has changed e.g. the connection has been established
- Within this callback, set up, for example, record or playback streams
- Establish further callbacks for these streams
- Within the stream callbacks, do more processing, such as saving a recording stream to file

Steps (1) - (7) will be common to most applications. The context state callback wil be called in response to changes in the server. These are state changes such as `PA_CONTEXT_CONNECTIN`, `PA_CONTEXT_SETTING_NAME` and so on. The change of relevance to most applications will be `PA_CONTEXT_READY`. This signifies that the application can make requests of the server in its steady state.

In step (8) the application will set its own behaviour. This is done by setting up further callback functions for various operations, such as listing devices or playing audio.

List of devices

The function `pa_context_get_sink_info_list` will set up a callback function to list source devices by

```
pa_context_get_sink_info_list(c, sinklist_cb,
NULL)
```

where `c` is the context, `sinklist_cb` is the application's callback and `NULL` is user data passed to the callback.

The callback is called as

```
void sinklist_cb(pa_context *c, const
pa_sink_info *i, int eol, void *userdata)
```

The parameter `eol` can take three values: negative means a failure of some kind; zero means a valid entry for `pa_sink_info`; positive means that there are no more valid entries in the list.

The structure `pa_sink_info` is defined as

```
struct {
  const char *  name;
  uint32_t     index;
  const char *     description;
  pa_sample_spec     sample_spec;
  pa_channel_map     channel_map;
  uint32_t     owner_module;
  pa_cvolume     volume;
  int     mute;
  uint32_t     monitor_source;
  const char *     monitor_source_name;
  pa_usec_t     latency;
  const char *     driver;
  pa_sink_flags_t     flags;
  pa_proplist *     proplist;
  pa_usec_t     configured_latency;
  pa_volume_t     base_volume;
  pa_sink_state_t     state;
  uint32_t     n_volume_steps;
  uint32_t     card;
  uint32_t     n_ports;
  pa_sink_port_info **     ports;
  pa_sink_port_info *     active_port;
  uint8_t     n_formats;
  pa_format_info **     formats;
} pa_sink_info
```

Further information about this structure is maintained in the Doxygen entry pa_sink_info Struct Reference

For information, the major fields are the `name` and the `description`. The `index` is an opaque index into some data structure and is used in many PulseAudio functions. The `proplist` is a map of general information that may contain interesting information. This can be retrieved by iterating through the map.

There is a similar callback and data structures for input devices.

A program to list input and output devices current when the application connects to the server is palist_devices.c:

```c
/**
 * palist_devices.c
 * Jan Newmarch
 */

#include <stdio.h>
#include <string.h>
#include <pulse/pulseaudio.h>

#define _(x) x

// quit when this reaches 2
int no_more_sources_or_sinks = 0;

int ret;

pa_context *context;

void show_error(char *s) {
    fprintf(stderr, "%s\n", s);
}

void print_properties(pa_proplist *props) {
    void *state = NULL;

    printf("  Properties are: \n");
```

```
    while (1) {
    char *key;
    if ((key = pa_proplist_iterate(props,
&state)) == NULL) {
        return;
    }
    char *value = pa_proplist_gets(props, key);
    printf("   key: %s, value: %s\n", key,
value);
    }
}



/**
 * print information about a sink
 */
void sinklist_cb(pa_context *c, const
pa_sink_info *i, int eol, void *userdata) {

    // If eol is set to a positive number,
you're at the end of the list
    if (eol > 0) {
    printf("**No more sinks\n");
    no_more_sources_or_sinks++;
    if (no_more_sources_or_sinks == 2)
        exit(0);
    return;
```

```
    }

    printf("Sink: name %s, description %s\n",
i->name, i->description);
    print_properties(i->proplist);
}

/**
 * print information about a source
 */
void sourcelist_cb(pa_context *c, const
pa_source_info *i, int eol, void *userdata) {
    if (eol > 0) {
    printf("**No more sources\n");
    no_more_sources_or_sinks++;
    if (no_more_sources_or_sinks == 2)
        exit(0);
    return;
    }

    printf("Source: name %s, description %s\n",
i->name, i->description);
    print_properties(i->proplist);
}


void context_state_cb(pa_context *c, void
```

```
*userdata) {

    switch (pa_context_get_state(c)) {
    case PA_CONTEXT_UNCONNECTED:
    case PA_CONTEXT_CONNECTING:
    case PA_CONTEXT_AUTHORIZING:
    case PA_CONTEXT_SETTING_NAME:
    break;

    case PA_CONTEXT_READY: {
    pa_operation *o;

    // set up a callback to tell us about
source devices
    if (!(o =
pa_context_get_source_info_list(c,
                            sourcelist_cb,
                            NULL
                              ))) {
        show_error(_("pa_context_subscribe()
failed"));
        return;
    }
    pa_operation_unref(o);

    // set up a callback to tell us about sink
devices
```

```
    if (!(o = pa_context_get_sink_info_list(c,
                                sinklist_cb,
                                NULL
                                  ))) {
        show_error(_("pa_context_subscribe()
failed"));
        return;
    }
    pa_operation_unref(o);

    break;
    }

    case PA_CONTEXT_FAILED:
    case PA_CONTEXT_TERMINATED:
    default:
    return;
    }
}

int main(int argc, char *argv[]) {

    // Define our pulse audio loop and
connection variables
    pa_mainloop *pa_ml;
    pa_mainloop_api *pa_mlapi;
```

```
    // Create a mainloop API and connection to
the default server
    pa_ml = pa_mainloop_new();
    pa_mlapi = pa_mainloop_get_api(pa_ml);
    context = pa_context_new(pa_mlapi, "Device
list");

    // This function connects to the pulse
server
    pa_context_connect(context, NULL, 0, NULL);

    // This function defines a callback so the
server will tell us its state.
    pa_context_set_state_callback(context,
context_state_cb, NULL);

    if (pa_mainloop_run(pa_ml, &ret) < 0) {
    printf("pa_mainloop_run() failed.");
    exit(1);
    }
}
```

On my laptop this gives (elided)

```
Source: name alsa_output.pci-0000_01_00.1.hdmi-
stereo.monitor, description Monitor of HDMI
Audio stub Digital Stereo (HDMI)
  Properties are:
   key: device.description, value: Monitor of
HDMI Audio stub Digital Stereo (HDMI)
   key: device.class, value: monitor
   key: alsa.card, value: 1
   key: alsa.card_name, value: HDA NVidia
   key: alsa.long_card_name, value: HDA NVidia
at 0xe5080000 irq 17
   key: alsa.driver_name, value: snd_hda_intel
   key: device.bus_path, value: pci-
0000:01:00.1
   key: sysfs.path, value:
/devices/pci0000:00/0000:00:01.0/0000:01:00.1/sou

   key: device.bus, value: pci
   key: device.vendor.id, value: 10de
   key: device.vendor.name, value: nVidia
Corporation
   key: device.product.id, value: 0e08
   key: device.product.name, value: HDMI Audio
stub
   key: device.string, value: 1
   key: module-udev-detect.discovered, value: 1
```

```
   key: device.icon_name, value: audio-card-pci
Source: name alsa_output.pci-
0000_00_1b.0.analog-stereo.monitor, description
Monitor of Internal Audio Analog Stereo
  Properties are:
   ...
Source: name alsa_input.pci-
0000_00_1b.0.analog-stereo, description
Internal Audio Analog Stereo
  Properties are:
  ...
Source: name alsa_output.usb-
Creative_Technology_Ltd_SB_X-
Fi_Surround_5.1_Pro_000003d0-00-Pro.analog-
stereo.monitor, description Monitor of SB X-Fi
Surround 5.1 Pro Analog Stereo
  Properties are:
  ...
Source: name alsa_input.usb-
Creative_Technology_Ltd_SB_X-
Fi_Surround_5.1_Pro_000003d0-00-Pro.analog-
stereo, description SB X-Fi Surround 5.1 Pro
Analog Stereo
  Properties are:
  ...
**No more sources
Sink: name alsa_output.pci-0000_01_00.1.hdmi-
```

```
stereo, description HDMI Audio stub Digital
Stereo (HDMI)
  Properties are:
   key: alsa.resolution_bits, value: 16
   key: device.api, value: alsa
   key: device.class, value: sound
   key: alsa.class, value: generic
   key: alsa.subclass, value: generic-mix
   key: alsa.name, value: HDMI 0
   key: alsa.id, value: HDMI 0
   key: alsa.subdevice, value: 0
   key: alsa.subdevice_name, value: subdevice
#0
   key: alsa.device, value: 3
   key: alsa.card, value: 1
   key: alsa.card_name, value: HDA NVidia
   key: alsa.long_card_name, value: HDA NVidia
at 0xe5080000 irq 17
   key: alsa.driver_name, value: snd_hda_intel
   key: device.bus_path, value: pci-
0000:01:00.1
   key: sysfs.path, value:
/devices/pci0000:00/0000:00:01.0/0000:01:00.1/sou

   key: device.bus, value: pci
   key: device.vendor.id, value: 10de
   key: device.vendor.name, value: nVidia
```

```
Corporation
   key: device.product.id, value: 0e08
   key: device.product.name, value: HDMI Audio
stub
   key: device.string, value: hdmi:1
   key: device.buffering.buffer_size, value:
352768
   key: device.buffering.fragment_size, value:
176384
   key: device.access_mode, value: mmap+timer
   key: device.profile.name, value: hdmi-stereo
   key: device.profile.description, value:
Digital Stereo (HDMI)
   key: device.description, value: HDMI Audio
stub Digital Stereo (HDMI)
   key: alsa.mixer_name, value: Nvidia GPU 1c
HDMI/DP
   key: alsa.components, value:
HDA:10de001c,10281494,00100100
   key: module-udev-detect.discovered, value: 1
   key: device.icon_name, value: audio-card-pci
Sink: name alsa_output.pci-0000_00_1b.0.analog-
stereo, description Internal Audio Analog
Stereo
  Properties are:
  ...
Sink: name alsa_output.usb-
```

```
Creative_Technology_Ltd_SB_X-
Fi_Surround_5.1_Pro_000003d0-00-Pro.analog-
stereo, description SB X-Fi Surround 5.1 Pro
Analog Stereo
  Properties are:
  ...
**No more sinks
```

An alternative program with the same effect is PulseAudio: An Async Example To Get Device Lists by Igor Brezac and Eric Connell. It doesn't follow quite as complex a route as the above, as it only queries the server for its devices. However, it uses its own state machine to track where in the callback process it is!

Monitoring ongoing changes: new sources and sinks

The last program listed the source and sink devices registered with PulseAudio at the time a connection to the server was established. However, when a new device is connected or an existing device is disconnected, PulseAudio registers a changes in the context and this can also be monitored by callbacks.

The key to doing this is to *subscribe* to context changes by `pa_context_subscribe`. This takes a context, a mask of subscription events and user data. Possible values of the mask are described at Subscription event mask and include `PA_SUBSCRIPTION_MASK_SINK` for changes in sinks and `PA_SUBSCRIPTION_MASK_SINK_INPUT` for sink input events.

Setting the callback function to monitor these changes is a bit odd. The function `pa_context_subscribe` takes a callback function of type `pa_context_success_cb` but this doesn't contain information about what caused the callback. Instead, it is better to first call `pa_context_set_subscribe_callback` which takes a callback function of type `pa_context_subscribe_cb_t` which *does* get passed such information and then use `NULL` for the callback in `pa_context_subscribe`!

Within a `pa_context_subscribe_cb_t` subscription callback, the cause of the callback can be examined and appropriate code called. If a new subscription to a sink is found, then information about the sink can be found by `pa_context_get_sink_info_by_index` which takes another callback! after chasing through all these callbacks, you can eventually get information about new devices.

Note that the callback function used by `pa_context_get_sink_info_list` and the callback function used by `pa_context_get_sink_info_by_index` are the same - the callback is called once per sink device regardless of whether it is a singleton or one of a list of devices.

A program to list devices on connection and also to list changes as devices are connected or disconnected is palist_devices_ongoing.c:

```
/**
 * palist_clients.c
 * Jan Newmarch
 */

/***
    This file is based on pacat.c and
pavuctl.c,  part of PulseAudio.

    pacat.c:
    Copyright 2004-2006 Lennart Poettering
    Copyright 2006 Pierre Ossman
<ossman@cendio.se> for Cendio AB

    PulseAudio is free software; you can
redistribute it and/or modify
    it under the terms of the GNU Lesser
General Public License as published
    by the Free Software Foundation; either
version 2.1 of the License,
    or (at your option) any later version.

    PulseAudio is distributed in the hope that
it will be useful, but
    WITHOUT ANY WARRANTY; without even the
implied warranty of
```

```
    MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE. See the GNU
    General Public License for more details.

    You should have received a copy of the GNU
Lesser General Public License
    along with PulseAudio; if not, write to the
Free Software
    Foundation, Inc., 59 Temple Place, Suite
330, Boston, MA 02111-1307
    USA.
***/



#include <stdio.h>
#include <string.h>
#include <pulse/pulseaudio.h>

#define _(x) x

int ret;

pa_context *context;

void show_error(char *s) {
    fprintf(stderr, "%s\n", s);
}
```

```
void print_properties(pa_proplist *props) {
    void *state = NULL;

    printf("  Properties are: \n");
    while (1) {
    char *key;
    if ((key = pa_proplist_iterate(props,
&state)) == NULL) {
        return;
    }
    char *value = pa_proplist_gets(props, key);
    printf("   key: %s, value: %s\n", key,
value);
    }
}

/**
 * print information about a sink
 */
void sink_cb(pa_context *c, const pa_sink_info
*i, int eol, void *userdata) {

    // If eol is set to a positive number,
you're at the end of the list
    if (eol > 0) {
    return;
```

```
    }

    printf("Sink: name %s, description %s\n",
i->name, i->description);
    // print_properties(i->proplist);
}

/**
 * print information about a source
 */
void source_cb(pa_context *c, const
pa_source_info *i, int eol, void *userdata) {
    if (eol > 0) {
    return;
    }

    printf("Source: name %s, description %s\n",
i->name, i->description);
    // print_properties(i->proplist);
}

void subscribe_cb(pa_context *c,
pa_subscription_event_type_t t, uint32_t index,
void *userdata) {

    switch (t &
PA_SUBSCRIPTION_EVENT_FACILITY_MASK) {
```

```
    case PA_SUBSCRIPTION_EVENT_SINK:
    if ((t & PA_SUBSCRIPTION_EVENT_TYPE_MASK)
== PA_SUBSCRIPTION_EVENT_REMOVE)
        printf("Removing sink index %d\n",
index);
    else {
        pa_operation *o;
        if (!(o =
pa_context_get_sink_info_by_index(c, index,
sink_cb, NULL))) {

show_error(_("pa_context_get_sink_info_by_index()
 failed"));
        return;
        }
        pa_operation_unref(o);
    }
    break;

    case PA_SUBSCRIPTION_EVENT_SOURCE:
    if ((t & PA_SUBSCRIPTION_EVENT_TYPE_MASK)
== PA_SUBSCRIPTION_EVENT_REMOVE)
        printf("Removing source index %d\n",
index);
    else {
        pa_operation *o;
```

```
        if (!(o =
pa_context_get_source_info_by_index(c, index,
source_cb, NULL))) {

show_error(_("pa_context_get_source_info_by_index
 failed"));
        return;
        }
        pa_operation_unref(o);
    }
    break;
    }
}

void context_state_cb(pa_context *c, void
*userdata) {

    switch (pa_context_get_state(c)) {
    case PA_CONTEXT_UNCONNECTED:
    case PA_CONTEXT_CONNECTING:
    case PA_CONTEXT_AUTHORIZING:
    case PA_CONTEXT_SETTING_NAME:
    break;

    case PA_CONTEXT_READY: {
    pa_operation *o;
```

```
    if (!(o =
pa_context_get_source_info_list(c,
                                source_cb,
                                NULL
                                ))) {
        show_error(_("pa_context_subscribe()
failed"));
        return;
    }
    pa_operation_unref(o);

    if (!(o = pa_context_get_sink_info_list(c,
                            sink_cb,
                            NULL
                            ))) {
        show_error(_("pa_context_subscribe()
failed"));
        return;
    }
    pa_operation_unref(o);

    pa_context_set_subscribe_callback(c,
subscribe_cb, NULL);

    if (!(o = pa_context_subscribe(c,
(pa_subscription_mask_t)
```

```
(PA_SUBSCRIPTION_MASK_SINK|

PA_SUBSCRIPTION_MASK_SOURCE), NULL, NULL))) {
        show_error(_("pa_context_subscribe()
failed"));
        return;
    }
    pa_operation_unref(o);

    break;
    }

    case PA_CONTEXT_FAILED:
    case PA_CONTEXT_TERMINATED:
    default:
    return;
    }
}

int main(int argc, char *argv[]) {

    // Define our pulse audio loop and
connection variables
    pa_mainloop *pa_ml;
    pa_mainloop_api *pa_mlapi;
    pa_operation *pa_op;
    pa_time_event *time_event;
```

```
    // Create a mainloop API and connection to
the default server
    pa_ml = pa_mainloop_new();
    pa_mlapi = pa_mainloop_get_api(pa_ml);
    context = pa_context_new(pa_mlapi, "Device
list");

    // This function connects to the pulse
server
    pa_context_connect(context, NULL, 0, NULL);

    // This function defines a callback so the
server will tell us its state.
    pa_context_set_state_callback(context,
context_state_cb, NULL);


    if (pa_mainloop_run(pa_ml, &ret) < 0) {
    printf("pa_mainloop_run() failed.");
    exit(1);
    }
}
```

Record a stream

If you download the source for PulseAudio from FreeDesktop.org you will find a program `pacat.c` in the `utils` directory. This program uses some of the "private" API and will not compile using the "public" libraries. It also has all the bells and whistles that you would expect from a production program. I've taken this and stripped out the complexities so that you can find your way into this API. The file is parec.c:

```
/**
 * parec.c
 * Jan Newmarch
 */

/***
  This file is based on pacat.c,  part of
PulseAudio.

  pacat.c:
  Copyright 2004-2006 Lennart Poettering
  Copyright 2006 Pierre Ossman
<ossman@cendio.se> for Cendio AB

  PulseAudio is free software; you can
redistribute it and/or modify
  it under the terms of the GNU Lesser General
Public License as published
  by the Free Software Foundation; either
version 2.1 of the License,
  or (at your option) any later version.

  PulseAudio is distributed in the hope that it
will be useful, but
  WITHOUT ANY WARRANTY; without even the
implied warranty of
```

```
  MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE. See the GNU
  General Public License for more details.

  You should have received a copy of the GNU
Lesser General Public License
  along with PulseAudio; if not, write to the
Free Software
  Foundation, Inc., 59 Temple Place, Suite 330,
Boston, MA 02111-1307
  USA.
***/


#include <stdio.h>
#include <string.h>
#include <pulse/pulseaudio.h>

#define CLEAR_LINE "\n"
#define _(x) x

// From pulsecore/macro.h
#define pa_memzero(x,l) (memset((x), 0, (l)))
#define pa_zero(x) (pa_memzero(&(x),
sizeof(x)))

int fdout;
```

```
char *fname = "tmp.s16";

int verbose = 1;
int ret;

pa_context *context;

static pa_sample_spec sample_spec = {
  .format = PA_SAMPLE_S16LE,
  .rate = 44100,
  .channels = 2
};

static pa_stream *stream = NULL;

/* This is my builtin card. Use paman to find
yours
   or set it to NULL to get the default device
*/
static char *device = "alsa_input.pci-
0000_00_1b.0.analog-stereo";

static pa_stream_flags_t flags = 0;

void stream_state_callback(pa_stream *s, void
*userdata) {
  assert(s);
```

```
  switch (pa_stream_get_state(s)) {
  case PA_STREAM_CREATING:
    // The stream has been created, so
    // let's open a file to record to
    printf("Creating stream\n");
    fdout = creat(fname,  0711);
    break;

  case PA_STREAM_TERMINATED:
    close(fdout);
    break;

  case PA_STREAM_READY:

    // Just for info: no functionality in this
branch
    if (verbose) {
      const pa_buffer_attr *a;
      char cmt[PA_CHANNEL_MAP_SNPRINT_MAX],
sst[PA_SAMPLE_SPEC_SNPRINT_MAX];

      printf("Stream successfully created.");

      if (!(a = pa_stream_get_buffer_attr(s)))
    printf("pa_stream_get_buffer_attr() failed:
%s",
```

```
pa_strerror(pa_context_errno(pa_stream_get_contex

      else {
    printf("Buffer metrics: maxlength=%u,
fragsize=%u", a->maxlength, a->fragsize);

      }

      printf("Connected to device %s (%u,
%ssuspended).",
         pa_stream_get_device_name(s),
         pa_stream_get_device_index(s),
         pa_stream_is_suspended(s) ? "" : "not
");
    }

    break;

  case PA_STREAM_FAILED:
  default:
    printf("Stream error: %s",
pa_strerror(pa_context_errno(pa_stream_get_contex

    exit(1);
  }
}
```

```
/************ Stream callbacks **************/

/* This is called whenever new data is
available */
static void stream_read_callback(pa_stream *s,
size_t length, void *userdata) {

  assert(s);
  assert(length > 0);

  // Copy the data from the server out to a
file
  fprintf(stderr, "Can read %d\n", length);


  while (pa_stream_readable_size(s) > 0) {
    const void *data;
    size_t length;

    // peek actually creates and fills the data
vbl
    if (pa_stream_peek(s, &data, &length) < 0)
{
      fprintf(stderr, "Read failed\n");
      exit(1);
      return;
    }
```

```
    fprintf(stderr, "Writing %d\n", length);
    write(fdout, data, length);

    // swallow the data peeked at before
    pa_stream_drop(s);
  }
}



// This callback gets called when our context
changes state.  We really only
// care about when it's ready or if it has
failed
void state_cb(pa_context *c, void *userdata) {
  pa_context_state_t state;
  int *pa_ready = userdata;

  printf("State changed\n");
  state = pa_context_get_state(c);
  switch  (state) {
    // There are just here for reference
  case PA_CONTEXT_UNCONNECTED:
  case PA_CONTEXT_CONNECTING:
  case PA_CONTEXT_AUTHORIZING:
  case PA_CONTEXT_SETTING_NAME:
    break;
  case PA_CONTEXT_FAILED:
```

```
  case PA_CONTEXT_TERMINATED:
    *pa_ready = 2;
    break;
  case PA_CONTEXT_READY: {
    pa_buffer_attr buffer_attr;

    if (verbose)
      printf("Connection established.%s\n",
CLEAR_LINE);

    if (!(stream = pa_stream_new(c,
"JanCapture", &sample_spec, NULL))) {
      printf("pa_stream_new() failed: %s",
pa_strerror(pa_context_errno(c)));
      exit(1);
    }

    // Watch for changes in the stream state to
create the output file
    pa_stream_set_state_callback(stream,
stream_state_callback, NULL);

    // Watch for changes in the stream's read
state to write to the output file
    pa_stream_set_read_callback(stream,
stream_read_callback, NULL);
```

```
    // Set properties of the record buffer
    pa_zero(buffer_attr);
    buffer_attr.maxlength = (uint32_t) -1;
    buffer_attr.prebuf = (uint32_t) -1;
    buffer_attr.fragsize = buffer_attr.tlength
= (uint32_t) -1;
    buffer_attr.minreq = (uint32_t) -1;

    // and start recording
    if (pa_stream_connect_record(stream,
device, &buffer_attr, flags) < 0) {
      printf("pa_stream_connect_record()
failed: %s", pa_strerror(pa_context_errno(c)));
      exit(1);
    }
    break;
  }
  default:
    return;
  }
}

int main(int argc, char *argv[]) {

  // Define our pulse audio loop and connection
variables
  pa_mainloop *pa_ml;
```

```
  pa_mainloop_api *pa_mlapi;
  pa_operation *pa_op;
  pa_time_event *time_event;

  // Create a mainloop API and connection to
the default server
  pa_ml = pa_mainloop_new();
  pa_mlapi = pa_mainloop_get_api(pa_ml);
  context = pa_context_new(pa_mlapi, "test");

  // This function connects to the pulse server
  pa_context_connect(context, NULL, 0, NULL);

  // This function defines a callback so the
server will tell us its state.
  pa_context_set_state_callback(context,
state_cb, NULL);

  if (pa_mainloop_run(pa_ml, &ret) < 0) {
    printf("pa_mainloop_run() failed.");
    exit(1);
  }
}
```

Play a file

Recording an input stream is done within a stream read callback by the call `pa_stream_peek`. Similarly, playing an output stream is done by a stream write callback by the call `pa_stream_write`.

In the following program the callback is set within the PA_CONTEXT_READY branch of the context state change callback. The stream write callback is passed the number of bytes the consuming stream is prepared to receive, so read that number of bytes from the file and write them to the stream.

Care has to be taken at the end of file. There may be unplayed material in PulseAudio's output buffers. This needs to be drained before the program can exit. This is done by the function `pa_stream_drain`. On end of file, first set the stream write callback to null so that the output stream doesn't keep calling for more input, and then drain the stream. A stream drain complete callback will be called on completion of this, so the program can then exit (or do something else).

In this program we include many more callbacks than in earlier ones, to show the range of features that can be monitored.

The program is pacat2.c:

```
/**
 * pacat2.c
 * Jan Newmarch
 */



/***
  This file is based on pacat.c,  part of
PulseAudio.

  pacat.c:
  Copyright 2004-2006 Lennart Poettering
  Copyright 2006 Pierre Ossman
<ossman@cendio.se> for Cendio AB

  PulseAudio is free software; you can
redistribute it and/or modify
  it under the terms of the GNU Lesser General
Public License as published
  by the Free Software Foundation; either
version 2.1 of the License,
  or (at your option) any later version.

  PulseAudio is distributed in the hope that it
will be useful, but
  WITHOUT ANY WARRANTY; without even the
```

```
implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE. See the GNU
  General Public License for more details.

  You should have received a copy of the GNU
Lesser General Public License
  along with PulseAudio; if not, write to the
Free Software
  Foundation, Inc., 59 Temple Place, Suite 330,
Boston, MA 02111-1307
  USA.
***/


#include <stdio.h>
#include <string.h>
#include <pulse/pulseaudio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>

// ???
#define CLEAR_LINE "\n"
```

```
// From pulsecore/macro.h JN
#define pa_memzero(x,l) (memset((x), 0, (l)))
#define pa_zero(x) (pa_memzero(&(x),
sizeof(x)))

int verbose = 1;
int ret;

static pa_volume_t volume = PA_VOLUME_NORM;
static int volume_is_set = 0;

static int fdin;

static pa_sample_spec sample_spec = {
  .format = PA_SAMPLE_S16LE,
  .rate = 44100,
  .channels = 2
};

static pa_stream *stream = NULL;
static pa_channel_map channel_map;
static pa_proplist *proplist = NULL;

// Define our pulse audio loop and connection
variables
static pa_mainloop *mainloop;
static pa_mainloop_api *mainloop_api;
```

```
static pa_operation *pa_op;
static pa_context *context = NULL;

static void *buffer = NULL;
static size_t buffer_length = 0, buffer_index =
0;

static pa_io_event* stdio_event = NULL;

// Get device name from e.g. paman
//static char *device = "alsa_output.pci-
0000_00_1b.0.analog-stereo";
// Use default device
static char *device = NULL;

static pa_stream_flags_t flags = 0;

static size_t latency = 0, process_time = 0;
static int32_t latency_msec = 1,
process_time_msec = 0;

static int raw = 1;


/* Connection draining complete */
static void
context_drain_complete(pa_context*c, void
```

```
*userdata) {
  pa_context_disconnect(c);
}

static void stream_drain_complete(pa_stream*s,
int success, void *userdata) {
  pa_operation *o = NULL;

  if (!success) {
    printf("Failed to drain stream: %s",
pa_strerror(pa_context_errno(context)));
    exit(1);
  }

  if (verbose)
    printf("Playback stream drained.");

  pa_stream_disconnect(stream);
  pa_stream_unref(stream);
  stream = NULL;

  if (!(o = pa_context_drain(context,
context_drain_complete, NULL)))
    pa_context_disconnect(context);
  else {
    pa_operation_unref(o);
    if (verbose)
```

```
      printf("Draining connection to server.");
  }
}



/* Start draining */
static void start_drain(void) {
  printf("Draining\n");
  if (stream) {
    pa_operation *o;

    pa_stream_set_write_callback(stream, NULL,
NULL);

    if (!(o = pa_stream_drain(stream,
stream_drain_complete, NULL))) {
      //printf("pa_stream_drain(): %s",
pa_strerror(pa_context_errno(context)));
      exit(1);
      return;
    }

    pa_operation_unref(o);
  } else
    exit(0);
}
```

```
/* Write some data to the stream */
static void do_stream_write(size_t length) {
  size_t l;
  assert(length);

  printf("do stream write: Writing %d to
stream\n", length);

  if (!buffer || !buffer_length) {
    buffer = pa_xmalloc(length);
    buffer_length = length;
    buffer_index = 0;
    //printf("  return without writing\n");
    //return;

  }

  while (buffer_length > 0) {
    l = read(fdin, buffer + buffer_index,
buffer_length);
    if (l <= 0) {
      start_drain();
      return;
    }
    if (pa_stream_write(stream, (uint8_t*)
buffer + buffer_index, l, NULL, 0,
PA_SEEK_RELATIVE) < 0) {
```

```
      printf("pa_stream_write() failed: %s",
pa_strerror(pa_context_errno(context)));
      exit(1);
      return;
    }
    buffer_length -= l;
    buffer_index += l;

    if (!buffer_length) {
      pa_xfree(buffer);
      buffer = NULL;
      buffer_index = buffer_length = 0;
    }
  }
}


void stream_state_callback(pa_stream *s, void
*userdata) {
  assert(s);

  switch (pa_stream_get_state(s)) {
  case PA_STREAM_CREATING:
    break;
  case PA_STREAM_TERMINATED:
    break;
```

```
  case PA_STREAM_READY:

    if (verbose) {
      const pa_buffer_attr *a;
      char cmt[PA_CHANNEL_MAP_SNPRINT_MAX],
sst[PA_SAMPLE_SPEC_SNPRINT_MAX];

      printf("Stream successfully created.\n");

      if (!(a = pa_stream_get_buffer_attr(s)))
    printf("pa_stream_get_buffer_attr() failed:
%s\n",
pa_strerror(pa_context_errno(pa_stream_get_contex

      else {
    printf("Buffer metrics: maxlength=%u,
fragsize=%u\n", a->maxlength, a->fragsize);

      }
      /*
    printf("Using sample spec '%s', channel map
'%s'.",
    pa_sample_spec_snprint(sst, sizeof(sst),
pa_stream_get_sample_spec(s)),
    pa_channel_map_snprint(cmt, sizeof(cmt),
pa_stream_get_channel_map(s)));
      */
```

```
      printf("Connected to device %s (%u,
%ssuspended).\n",
         pa_stream_get_device_name(s),
         pa_stream_get_device_index(s),
         pa_stream_is_suspended(s) ? "" : "not
");
    }

    break;

  case PA_STREAM_FAILED:
  default:
    printf("Stream error: %s",
pa_strerror(pa_context_errno(pa_stream_get_contex

    exit(1); //quit(1);
  }
}


/*********** Stream callbacks **************/


static void stream_success(pa_stream *s, int
succes, void *userdata) {
  printf("Succeded\n");
```

```
}


static void stream_suspended_callback(pa_stream
*s, void *userdata) {
  assert(s);

  if (verbose) {
    if (pa_stream_is_suspended(s))
      fprintf(stderr, "Stream device
suspended.%s \n", CLEAR_LINE);
    else
      fprintf(stderr, "Stream device resumed.%s
\n", CLEAR_LINE);
  }
}

static void stream_underflow_callback(pa_stream
*s, void *userdata) {
  assert(s);

  if (verbose)
    fprintf(stderr, "Stream underrun.%s \n",
CLEAR_LINE);
}

static void stream_overflow_callback(pa_stream
```

```
*s, void *userdata) {
  assert(s);

  if (verbose)
    fprintf(stderr, "Stream overrun.%s \n",
CLEAR_LINE);
}

static void stream_started_callback(pa_stream
*s, void *userdata) {
  assert(s);

  if (verbose)
    fprintf(stderr, "Stream started.%s \n",
CLEAR_LINE);
}

static void stream_moved_callback(pa_stream *s,
void *userdata) {
  assert(s);

  if (verbose)
    fprintf(stderr, "Stream moved to device %s
(%u, %ssuspended).%s \n",
pa_stream_get_device_name(s),
pa_stream_get_device_index(s),
pa_stream_is_suspended(s) ? "" : "not ",
```

```
CLEAR_LINE);
}

static void
stream_buffer_attr_callback(pa_stream *s, void
*userdata) {
  assert(s);

  if (verbose)
    fprintf(stderr, "Stream buffer attributes
changed.%s \n",  CLEAR_LINE);
}

static void stream_event_callback(pa_stream *s,
const char *name, pa_proplist *pl, void
*userdata) {
  char *t;

  assert(s);
  assert(name);
  assert(pl);

  t = pa_proplist_to_string_sep(pl, ", ");
  fprintf(stderr, "Got event '%s', properties
'%s'\n", name, t);
  pa_xfree(t);
}
```

```
/* This is called whenever new data may be
written to the stream */
static void stream_write_callback(pa_stream *s,
size_t length, void *userdata) {
  //assert(s);
  //assert(length > 0);

  printf("Stream write callback: Ready to write
%d bytes\n", length);

  printf("  do stream write from stream write
callback\n");
  do_stream_write(length);

 }


// This callback gets called when our context
changes state.  We really only
// care about when it's ready or if it has
failed
void state_cb(pa_context *c, void *userdata) {
  pa_context_state_t state;
  int *pa_ready = userdata;
```

```
  printf("State changed\n");
  state = pa_context_get_state(c);
  switch  (state) {
    // There are just here for reference
  case PA_CONTEXT_UNCONNECTED:
  case PA_CONTEXT_CONNECTING:
  case PA_CONTEXT_AUTHORIZING:
  case PA_CONTEXT_SETTING_NAME:
    break;
  case PA_CONTEXT_FAILED:
  case PA_CONTEXT_TERMINATED:
    *pa_ready = 2;
    break;
  case PA_CONTEXT_READY: {
    pa_buffer_attr buffer_attr;

    if (verbose)
      printf("Connection established.%s\n",
CLEAR_LINE);

    if (!(stream = pa_stream_new(c,
"JanPlayback", &sample_spec, NULL))) {
      printf("pa_stream_new() failed: %s",
pa_strerror(pa_context_errno(c)));
      exit(1); // goto fail;
    }
```

```
    pa_stream_set_state_callback(stream,
stream_state_callback, NULL);

    pa_stream_set_write_callback(stream,
stream_write_callback, NULL);

    //pa_stream_set_read_callback(stream,
stream_read_callback, NULL);

    pa_stream_set_suspended_callback(stream,
stream_suspended_callback, NULL);
    pa_stream_set_moved_callback(stream,
stream_moved_callback, NULL);
    pa_stream_set_underflow_callback(stream,
stream_underflow_callback, NULL);
    pa_stream_set_overflow_callback(stream,
stream_overflow_callback, NULL);

    pa_stream_set_started_callback(stream,
stream_started_callback, NULL);

    pa_stream_set_event_callback(stream,
stream_event_callback, NULL);
    pa_stream_set_buffer_attr_callback(stream,
stream_buffer_attr_callback, NULL);
```

```
    pa_zero(buffer_attr);
    buffer_attr.maxlength = (uint32_t) -1;
    buffer_attr.prebuf = (uint32_t) -1;

    pa_cvolume cv;





    if (pa_stream_connect_playback(stream,
NULL, &buffer_attr, flags,
                    NULL,
                    NULL) < 0) {
      printf("pa_stream_connect_playback()
failed: %s", pa_strerror(pa_context_errno(c)));
      exit(1); //goto fail;
    } else {
      printf("Set playback callback\n");
    }

    pa_stream_trigger(stream, stream_success,
NULL);
    break;
  }
  default:
    return;
```

```
  }
}


int main(int argc, char *argv[]) {


  struct stat st;
  off_t size;
  ssize_t nread;

  // We'll need these state variables to keep
track of our requests
  int state = 0;
  int pa_ready = 0;

  if (argc != 2) {
    fprintf(stderr, "Usage: %s file\n",
argv[0]);
    exit(1);
  }
  // slurp the whole file into buffer
  if ((fdin = open(argv[1],  O_RDONLY)) == -1)
{
    perror("open");
    exit(1);
  }
```

```
  // Create a mainloop API and connection to
the default server
  mainloop = pa_mainloop_new();
  mainloop_api = pa_mainloop_get_api(mainloop);
  context = pa_context_new(mainloop_api,
"test");

  // This function connects to the pulse server
  pa_context_connect(context, NULL, 0, NULL);
  printf("Connecting\n");

  // This function defines a callback so the
server will tell us it's state.
  // Our callback will wait for the state to be
ready.  The callback will
  // modify the variable to 1 so we know when
we have a connection and it's
  // ready.
  // If there's an error, the callback will set
pa_ready to 2
  pa_context_set_state_callback(context,
state_cb, &pa_ready);


  if (pa_mainloop_run(mainloop, &ret) < 0) {
    printf("pa_mainloop_run() failed.");
```

```
      exit(1); // goto quit
    }




  }
```

With the latency set to the default, the number of bytes that can be written on each callback is 65470 bytes. This gives a minimum latency of 65470/44100 secs, or about 1500 msecs. With the latency and process time both set to 1msec, the buffer size is about 1440 bytes, for a latency of 32 msecs.

Play a file using I/O callbacks

Writing a file to an output stream is simple: read from a file into a buffer, and keep emptying the buffer by writing to the stream. Reading from a file is straightforward: use the standard Unix `read` function. You request a read of a number of bytes, and the `read` function returns the number of bytes actually read. This was discussed in the last section.

The program in the PulseAudio distribution uses a more complex system. It uses I/O-ready callbacks to pass some handling to an I/O callback. This makes use of two functions:

- `pa_stream_writable_size` tells how many bytes can be written to the stream
- `pa_stream_write` writes a number of bytes to a stream

The logic becomes: fill a buffer by reading from the file, and at the same time write as many bytes as possible from the buffer to the stream, upto the limit of the buffer size or however many bytes the stream can take, whichever is smaller.

In PulseAudio this is done asynchronously, using callback functions. The two relevant functions are

- The function `pa_stream_set_write_callback()` registers a callback that will be called whenever the stream is ready to be written to. Registering the callback looks like

```
pa_stream_set_write_callback(stream,
stream_write_callback, NULL)
```

The callback is passed the stream to write to (s) and the number of bytes that can be written (length):

```
void stream_write_callback(pa_stream *s, size_t
length, void *userdata)
```

- A callback to read from files is registered by one of the functions kept in the `mainloop_api` table. The registering function is `io_new` and is passed a Unix file descriptor for the file and the callback function. Reigstering the callback looks like

```
mainloop_api->io_new(mainloop_api,
               fdin,
               PA_IO_EVENT_INPUT,
               stdin_callback, NULL))
```

The callback is passed the file descriptor (fd) to read from:

```
void stdin_callback(pa_mainloop_api
*mainloop_api, pa_io_event *stdio_event,
                     int fd, pa_io_event_flags_t
f, void *userdata)
```

(Note: the PulseAudio code does a `dup2` from the source file's descriptor to `STDIN_FILENO` - which matches the name of the function. I can't see the point of that, and their code uses `fd` anyway.)

When should these callbacks be registered? The stream write callback can be registered at any time after the stream has been created which is done by `pa_stream_new`. For the stdin callback, I could only get it to work properly by registering it once the stream was ready i.e. in the `PA_STREAM_READY` branch of the stream state callback function.

So after all that, what is the logic of the program?

- In the stdin callback:
- if the buffer has stuff in it, then just return - no point in adding any more
- if the buffer is empty, then query the stream to see how much can be written to it
- if the stream says no more, then just read something into the buffer and return
- if the stream can be written to, then read from the file into the buffer and write it to the stream
- In the stream write callback:

- if the buffer is non-empty, write its contents to the stream

The program to play from a file presently looks like pacat.c:

```
/***
  This file is based on pacat.c,  part of
PulseAudio.

  pacat.c:
  Copyright 2004-2006 Lennart Poettering
  Copyright 2006 Pierre Ossman
<ossman@cendio.se> for Cendio AB

  PulseAudio is free software; you can
redistribute it and/or modify
  it under the terms of the GNU Lesser General
Public License as published
  by the Free Software Foundation; either
version 2.1 of the License,
  or (at your option) any later version.

  PulseAudio is distributed in the hope that it
will be useful, but
  WITHOUT ANY WARRANTY; without even the
implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE. See the GNU
  General Public License for more details.

  You should have received a copy of the GNU
```

```
Lesser General Public License
  along with PulseAudio; if not, write to the
Free Software
  Foundation, Inc., 59 Temple Place, Suite 330,
Boston, MA 02111-1307
  USA.
***/



#include <stdio.h>
#include <string.h>
#include <pulse/pulseaudio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>

#define CLEAR_LINE "\n"

// From pulsecore/macro.h JN
#define pa_memzero(x,l) (memset((x), 0, (l)))
#define pa_zero(x) (pa_memzero(&(x),
sizeof(x)))

int verbose = 1;
```

```
int ret;

static pa_volume_t volume = PA_VOLUME_NORM;
static int volume_is_set = 0;

static int fdin;

static pa_sample_spec sample_spec = {
  .format = PA_SAMPLE_S16LE,
  .rate = 44100,
  .channels = 2
};

static pa_stream *stream = NULL;
static pa_channel_map channel_map;
static pa_proplist *proplist = NULL;

// Define our pulse audio loop and connection
variables
static pa_mainloop *mainloop;
static pa_mainloop_api *mainloop_api;
static pa_operation *pa_op;
static pa_context *context = NULL;

static void *buffer = NULL;
static size_t buffer_length = 0, buffer_index =
0;
```

```
static pa_io_event* stdio_event = NULL;


static char *device = "alsa_output.pci-
0000_00_1b.0.analog-stereo";

static pa_stream_flags_t flags = 0;

static size_t latency = 0, process_time = 0;
static int32_t latency_msec = 0,
process_time_msec = 0;

static int raw = 1;



/* Write some data to the stream */
static void do_stream_write(size_t length) {
  size_t l;
  assert(length);

  printf("do stream write: Writing %d to
stream\n", length);

  if (!buffer || !buffer_length) {
    printf("  return without writing\n");
```

```
    return;
  }

  l = length;
  if (l > buffer_length)
    l = buffer_length;
  printf("  writing %d\n", l);
  if (pa_stream_write(stream, (uint8_t*) buffer
+ buffer_index, l, NULL, 0, PA_SEEK_RELATIVE) <
0) {
    printf("pa_stream_write() failed: %s",
pa_strerror(pa_context_errno(context)));
    exit(1);
    return;
  }

  buffer_length -= l;
  buffer_index += l;

  if (!buffer_length) {
    pa_xfree(buffer);
    buffer = NULL;
    buffer_index = buffer_length = 0;
  }

}
```

```
/* Connection draining complete */
static void
context_drain_complete(pa_context*c, void
*userdata) {
  pa_context_disconnect(c);
}

static void stream_drain_complete(pa_stream*s,
int success, void *userdata) {
  pa_operation *o = NULL;

  if (!success) {
    printf("Failed to drain stream: %s",
pa_strerror(pa_context_errno(context)));
    exit(1);
  }

  if (verbose)
    printf("Playback stream drained.");

  pa_stream_disconnect(stream);
  pa_stream_unref(stream);
  stream = NULL;

  if (!(o = pa_context_drain(context,
context_drain_complete, NULL)))
    pa_context_disconnect(context);
```

```
  else {
    pa_operation_unref(o);
    if (verbose)
      printf("Draining connection to server.");
  }
}



/* Start draining */
static void start_drain(void) {
  printf("Draining\n");
  if (stream) {
    pa_operation *o;

    pa_stream_set_write_callback(stream, NULL,
NULL);

    if (!(o = pa_stream_drain(stream,
stream_drain_complete, NULL))) {
      //printf("pa_stream_drain(): %s",
pa_strerror(pa_context_errno(context)));
      exit(1);
      return;
    }

    pa_operation_unref(o);
  } else
```

```
    exit(0);
}


/* New data on STDIN **/
static void stdin_callback(pa_mainloop_api
*mainloop_api, pa_io_event *stdio_event, int
fd, pa_io_event_flags_t f, void *userdata) {
  size_t l, w = 0;
  ssize_t r;

  printf("In stdin callback\n");
  //pa_assert(a == mainloop_api);
  // pa_assert(e);
  // pa_assert(stdio_event == e);

  if (buffer) {
    mainloop_api->io_enable(stdio_event,
PA_IO_EVENT_NULL);
    printf("  Buffer isn't null\n");
    return;
  }

  if (!stream || pa_stream_get_state(stream) !=
PA_STREAM_READY || !(l = w =
pa_stream_writable_size(stream)))
    l = 4096;
```

```
  buffer = pa_xmalloc(l);

  if ((r = read(fd, buffer, l)) <= 0) {
    if (r == 0) {
      if (verbose)
    printf("Got EOF.\n");

      start_drain();

    } else {
      printf("read() failed: %s\n",
strerror(errno));
      exit(1);
    }

    mainloop_api->io_free(stdio_event);
    stdio_event = NULL;
    return;
  }
  printf("  Read %d\n", r);

  buffer_length = (uint32_t) r;
  buffer_index = 0;

  if (w) {
    printf("  do stream write from stdin
```

```
callback\n");
    do_stream_write(w);
  }
}



void stream_state_callback(pa_stream *s, void
*userdata) {
  assert(s);

  switch (pa_stream_get_state(s)) {
  case PA_STREAM_CREATING:
    break;
  case PA_STREAM_TERMINATED:
    break;

  case PA_STREAM_READY:

    if (verbose) {
      const pa_buffer_attr *a;
      char cmt[PA_CHANNEL_MAP_SNPRINT_MAX],
sst[PA_SAMPLE_SPEC_SNPRINT_MAX];

      printf("Stream successfully created.\n");

      if (!(a = pa_stream_get_buffer_attr(s)))
```

```
    printf("pa_stream_get_buffer_attr() failed:
%s\n",
pa_strerror(pa_context_errno(pa_stream_get_contex

      else {
    printf("Buffer metrics: maxlength=%u,
fragsize=%u\n", a->maxlength, a->fragsize);

      }
      /*
    printf("Using sample spec '%s', channel map
'%s'.",
    pa_sample_spec_snprint(sst, sizeof(sst),
pa_stream_get_sample_spec(s)),
    pa_channel_map_snprint(cmt, sizeof(cmt),
pa_stream_get_channel_map(s)));
      */

      printf("Connected to device %s (%u,
%ssuspended).\n",
         pa_stream_get_device_name(s),
         pa_stream_get_device_index(s),
         pa_stream_is_suspended(s) ? "" : "not
");
    }

    // TRY HERE???
```

```
    if (!(stdio_event = mainloop_api-
>io_new(mainloop_api,
                           fdin, // STDIN_FILENO,
                           PA_IO_EVENT_INPUT,
                           stdin_callback,
NULL))) {
      printf("io_new() failed.");
      exit(1);
    }

    break;

  case PA_STREAM_FAILED:
  default:
    printf("Stream error: %s",
pa_strerror(pa_context_errno(pa_stream_get_contex

    exit(1); //quit(1);
  }
}


/*********** Stream callbacks **************/


static void stream_read_callback(pa_stream *s,
```

```
size_t length, void *userdata) {
  printf("Raedy to read\n");
}


static void stream_success(pa_stream *s, int
succes, void *userdata) {
  printf("Succeded\n");
}


static void stream_suspended_callback(pa_stream
*s, void *userdata) {
  assert(s);

  if (verbose) {
    if (pa_stream_is_suspended(s))
      fprintf(stderr, "Stream device
suspended.%s \n", CLEAR_LINE);
    else
      fprintf(stderr, "Stream device resumed.%s
\n", CLEAR_LINE);
  }
}

static void stream_underflow_callback(pa_stream
*s, void *userdata) {
```

```
  assert(s);

  if (verbose)
    fprintf(stderr, "Stream underrun.%s \n",
CLEAR_LINE);
}

static void stream_overflow_callback(pa_stream
*s, void *userdata) {
  assert(s);

  if (verbose)
    fprintf(stderr, "Stream overrun.%s \n",
CLEAR_LINE);
}

static void stream_started_callback(pa_stream
*s, void *userdata) {
  assert(s);

  if (verbose)
    fprintf(stderr, "Stream started.%s \n",
CLEAR_LINE);
}

static void stream_moved_callback(pa_stream *s,
void *userdata) {
```

```
  assert(s);

  if (verbose)
    fprintf(stderr, "Stream moved to device %s
(%u, %ssuspended).%s \n",
pa_stream_get_device_name(s),
pa_stream_get_device_index(s),
pa_stream_is_suspended(s) ? "" : "not ",
CLEAR_LINE);
}

static void
stream_buffer_attr_callback(pa_stream *s, void
*userdata) {
  assert(s);

  if (verbose)
    fprintf(stderr, "Stream buffer attributes
changed.%s \n",  CLEAR_LINE);
}

static void stream_event_callback(pa_stream *s,
const char *name, pa_proplist *pl, void
*userdata) {
  char *t;

  assert(s);
```

```
  assert(name);
  assert(pl);

  t = pa_proplist_to_string_sep(pl, ", ");
  fprintf(stderr, "Got event '%s', properties
'%s'\n", name, t);
  pa_xfree(t);
}



/* This is called whenever new data may be
written to the stream */
static void stream_write_callback(pa_stream *s,
size_t length, void *userdata) {
  //assert(s);
  //assert(length > 0);

  printf("Stream write callback: Ready to write
%d bytes\n", length);

  if (raw) {
    // assert(!sndfile);

    if (stdio_event)
      mainloop_api->io_enable(stdio_event,
PA_IO_EVENT_INPUT);
```

```
    if (!buffer)
      return;
    printf("  do stream write from stream write
callback\n");
    do_stream_write(length);

  }
}

// This callback gets called when our context
changes state.  We really only
// care about when it's ready or if it has
failed
void state_cb(pa_context *c, void *userdata) {
  pa_context_state_t state;
  int *pa_ready = userdata;

  printf("State changed\n");
  state = pa_context_get_state(c);
  switch  (state) {
    // There are just here for reference
  case PA_CONTEXT_UNCONNECTED:
  case PA_CONTEXT_CONNECTING:
  case PA_CONTEXT_AUTHORIZING:
  case PA_CONTEXT_SETTING_NAME:
    break;
  case PA_CONTEXT_FAILED:
```

```
  case PA_CONTEXT_TERMINATED:
    *pa_ready = 2;
    break;
  case PA_CONTEXT_READY: {
    pa_buffer_attr buffer_attr;

    if (verbose)
      printf("Connection established.%s\n",
CLEAR_LINE);

    if (!(stream = pa_stream_new(c,
"JanPlayback", &sample_spec, NULL))) {
      printf("pa_stream_new() failed: %s",
pa_strerror(pa_context_errno(c)));
      exit(1); // goto fail;
    }

    pa_stream_set_state_callback(stream,
stream_state_callback, NULL);

    pa_stream_set_write_callback(stream,
stream_write_callback, NULL);

    pa_stream_set_read_callback(stream,
stream_read_callback, NULL);

    pa_stream_set_suspended_callback(stream,
```

```
stream_suspended_callback, NULL);
    pa_stream_set_moved_callback(stream,
stream_moved_callback, NULL);
    pa_stream_set_underflow_callback(stream,
stream_underflow_callback, NULL);
    pa_stream_set_overflow_callback(stream,
stream_overflow_callback, NULL);

    pa_stream_set_started_callback(stream,
stream_started_callback, NULL);

    pa_stream_set_event_callback(stream,
stream_event_callback, NULL);
    pa_stream_set_buffer_attr_callback(stream,
stream_buffer_attr_callback, NULL);



    pa_zero(buffer_attr);
    buffer_attr.maxlength = (uint32_t) -1;
    buffer_attr.prebuf = (uint32_t) -1;

    buffer_attr.fragsize = buffer_attr.tlength
= (uint32_t) -1;
    buffer_attr.minreq = (uint32_t) -1;

    pa_cvolume cv;
```

```
    if (pa_stream_connect_playback(stream,
NULL, &buffer_attr, flags,
                    NULL,
                    NULL) < 0) {
      printf("pa_stream_connect_playback()
failed: %s", pa_strerror(pa_context_errno(c)));
      exit(1); //goto fail;
    } else {
      printf("Set playback callback\n");
    }

    pa_stream_trigger(stream, stream_success,
NULL);
    break;
  }
  default:
    return;
  }
}


int main(int argc, char *argv[]) {


  struct stat st;
  off_t size;
```

```
  ssize_t nread;

  // We'll need these state variables to keep
track of our requests
  int state = 0;
  int pa_ready = 0;

  if (argc != 2) {
    fprintf(stderr, "Usage: %s file\n",
argv[0]);
    exit(1);
  }
  // slurp the whole file into buffer
  if ((fdin = open(argv[1],  O_RDONLY)) == -1)
{
    perror("open");
    exit(1);
  }

  // Create a mainloop API and connection to
the default server
  mainloop = pa_mainloop_new();
  mainloop_api = pa_mainloop_get_api(mainloop);
  context = pa_context_new(mainloop_api,
"test");

  // This function connects to the pulse server
```

```
  pa_context_connect(context, NULL, 0, NULL);
  printf("Connecting\n");

  // This function defines a callback so the
server will tell us it's state.
  // Our callback will wait for the state to be
ready.  The callback will
  // modify the variable to 1 so we know when
we have a connection and it's
  // ready.
  // If there's an error, the callback will set
pa_ready to 2
  pa_context_set_state_callback(context,
state_cb, &pa_ready);


  if (pa_mainloop_run(mainloop, &ret) < 0) {
    printf("pa_mainloop_run() failed.");
    exit(1); // goto quit
  }
}
```

Controlling latency

Managing latency is described at Software/PulseAudio/Documentation/Developer/Clients/Latency Control . In brief:

> In your code you then have to do the following when calling pa_stream_connect_playback() resp. pa_stream_connect_record():
>
> - Pass PA_STREAM_ADJUST_LATENCY in the flags parameter. Only if this flag is set PA will reconfigure the low-level device's buffer size and adjust it to the latency you specify.
> - Pass a pa_buffer_attr struct in the buffer_attr parameter. In the fields of this struct make sure to initialize every single field to (uint32_t) -1, with the exception of tlength (for playback) resp. fragsize (for recording). Initialize those to the latency you want to achieve. Use pa_usec_to_bytes(&ss, ...) to convert the latency from a time unit to bytes.

The extra code is:

```
// Set properties of the record buffer
    pa_zero(buffer_attr);
    buffer_attr.maxlength = (uint32_t) -1;
    buffer_attr.prebuf = (uint32_t) -1;

    if (latency_msec > 0) {
      buffer_attr.fragsize =
buffer_attr.tlength =
pa_usec_to_bytes(latency_msec *
PA_USEC_PER_MSEC, &sample_spec);
      flags |= PA_STREAM_ADJUST_LATENCY;
    } else if (latency > 0) {
      buffer_attr.fragsize =
buffer_attr.tlength = (uint32_t) latency;
      flags |= PA_STREAM_ADJUST_LATENCY;
    } else
      buffer_attr.fragsize =
buffer_attr.tlength = (uint32_t) -1;

    if (process_time_msec > 0) {
      buffer_attr.minreq =
pa_usec_to_bytes(process_time_msec *
PA_USEC_PER_MSEC, &sample_spec);
    } else if (process_time > 0)
      buffer_attr.minreq = (uint32_t)
process_time;
```

```
    else
      buffer_attr.minreq = (uint32_t) -1;
```

PulseAudio also has mechanisms to estimate the latency of the devices. It uses information from timing events. A timer event callback has to be declared, as in

```
pa_context_rttime_new(context, pa_rtclock_now()
+ TIME_EVENT_USEC, time_event_callback, NULL))
```

The timer event callback is a "single shot" calback. It installs a stream update timer callback and sets up another timer callback:

```
void time_event_callback(pa_mainloop_api *m,
                 pa_time_event *e, const struct
timeval *t,
                 void *userdata) {
    if (stream && pa_stream_get_state(stream)
== PA_STREAM_READY) {
        pa_operation *o;
        if (!(o =
pa_stream_update_timing_info(stream,
stream_update_timing_callback, NULL)))
      1;
//pa_log(_("pa_stream_update_timing_info()
failed: %s"),
pa_strerror(pa_context_errno(context)));
        else
            pa_operation_unref(o);
    }

    pa_context_rttime_restart(context, e,
pa_rtclock_now() + TIME_EVENT_USEC);
```

The stream update timer callback can then estimate the latency:

```
void stream_update_timing_callback(pa_stream
*s, int success, void *userdata) {
    pa_usec_t l, usec;
    int negative = 0;

    // pa_assert(s);

    fprintf(stderr, "Update timing\n");

    if (!success ||
        pa_stream_get_time(s, &usec) < 0 ||
        pa_stream_get_latency(s, &l, &negative)
< 0) {
        fprintf(stderr, "Failed to get
latency\n");
        return;
    }

    fprintf(stderr, _("Time: %0.3f sec;
Latency: %0.0f usec."),
            (float) usec / 1000000,
            (float) l * (negative?-1.0f:1.0f));
    fprintf(stderr, "        \r");
}
```

With latency left to PulseAudio by setting fragsize and tlength to -1, I got:

```
Time: 0.850 sec; Latency: 850365 usec.
Time: 0.900 sec; Latency: 900446 usec.
Time: 0.951 sec; Latency: 950548 usec.
Time: 1.001 sec; Latency: 1000940 usec.
Time: 1.051 sec; Latency: 50801 usec.
Time: 1.101 sec; Latency: 100934 usec.
Time: 1.151 sec; Latency: 151007 usec.
Time: 1.201 sec; Latency: 201019 usec.
Time: 1.251 sec; Latency: 251150 usec.
Time: 1.301 sec; Latency: 301160 usec.
Time: 1.351 sec; Latency: 351218 usec.
Time: 1.401 sec; Latency: 401329 usec.
Time: 1.451 sec; Latency: 451400 usec.
Time: 1.501 sec; Latency: 501465 usec.
Time: 1.551 sec; Latency: 551587 usec.
Time: 1.602 sec; Latency: 601594 usec.
```

With them set to 1 msec, I got:

```
Time: 1.599 sec; Latency: 939 usec.
Time: 1.649 sec; Latency: 1105 usec.
Time: 1.699 sec; Latency: -158 usec.
Time: 1.750 sec; Latency: 1020 usec.
Time: 1.800 sec; Latency: 397 usec.
Time: 1.850 sec; Latency: -52 usec.
Time: 1.900 sec; Latency: 1827 usec.
Time: 1.950 sec; Latency: 529 usec.
Time: 2.000 sec; Latency: -90 usec.
Time: 2.050 sec; Latency: 997 usec.
Time: 2.100 sec; Latency: 436 usec.
Time: 2.150 sec; Latency: 866 usec.
Time: 2.200 sec; Latency: 406 usec.
Time: 2.251 sec; Latency: 1461 usec.
Time: 2.301 sec; Latency: 107 usec.
Time: 2.351 sec; Latency: 1257 usec.
```

The program to do all this is parec-latency.c:

```c
/* parec-latency.c */


#include <stdio.h>
#include <string.h>
#include <pulse/pulseaudio.h>

#define CLEAR_LINE "\n"
#define _(x) x

#define TIME_EVENT_USEC 50000

// From pulsecore/macro.h
#define pa_memzero(x,l) (memset((x), 0, (l)))
#define pa_zero(x) (pa_memzero(&(x),
sizeof(x)))

int fdout;
char *fname = "tmp.pcm";

int verbose = 1;
int ret;

pa_context *context;

static pa_sample_spec sample_spec = {
```

```
  .format = PA_SAMPLE_S16LE,
  .rate = 44100,
  .channels = 2
};

static pa_stream *stream = NULL;

/* This is my builtin card. Use paman to find
yours
   or set it to NULL to get the default device
*/
static char *device = "alsa_input.pci-
0000_00_1b.0.analog-stereo";

static pa_stream_flags_t flags = 0;

static size_t latency = 0, process_time = 0;
static int32_t latency_msec = 0,
process_time_msec = 0;

void stream_state_callback(pa_stream *s, void
*userdata) {
  assert(s);

  switch (pa_stream_get_state(s)) {
  case PA_STREAM_CREATING:
    // The stream has been created, so
```

```
    // let's open a file to record to
    printf("Creating stream\n");
    fdout = creat(fname,  0711);
    break;

  case PA_STREAM_TERMINATED:
    close(fdout);
    break;

  case PA_STREAM_READY:

    // Just for info: no functionality in this
branch
    if (verbose) {
      const pa_buffer_attr *a;
      char cmt[PA_CHANNEL_MAP_SNPRINT_MAX],
sst[PA_SAMPLE_SPEC_SNPRINT_MAX];

      printf("Stream successfully created.");

      if (!(a = pa_stream_get_buffer_attr(s)))
    printf("pa_stream_get_buffer_attr() failed:
%s",
pa_strerror(pa_context_errno(pa_stream_get_contex

      else {
    printf("Buffer metrics: maxlength=%u,
```

```
fragsize=%u", a->maxlength, a->fragsize);

      }

      printf("Connected to device %s (%u,
%ssuspended).",
         pa_stream_get_device_name(s),
         pa_stream_get_device_index(s),
         pa_stream_is_suspended(s) ? "" : "not
");
    }

    break;

  case PA_STREAM_FAILED:
  default:
    printf("Stream error: %s",
pa_strerror(pa_context_errno(pa_stream_get_contex

    exit(1);
  }
}


/* Show the current latency */
static void
stream_update_timing_callback(pa_stream *s, int
```

```
success, void *userdata) {
    pa_usec_t l, usec;
    int negative = 0;

    // pa_assert(s);

    fprintf(stderr, "Update timing\n");

    if (!success ||
        pa_stream_get_time(s, &usec) < 0 ||
        pa_stream_get_latency(s, &l, &negative)
< 0) {
        // pa_log(_("Failed to get latency"));
        //pa_log(_("Failed to get latency:
%s"), pa_strerror(pa_context_errno(context)));
        // quit(1);
        return;
    }

    fprintf(stderr, _("Time: %0.3f sec;
Latency: %0.0f usec.\n"),
            (float) usec / 1000000,
            (float) l * (negative?-1.0f:1.0f));
    //fprintf(stderr, "        \r");
}

static void time_event_callback(pa_mainloop_api
```

```
*m,
                pa_time_event *e, const struct
timeval *t,
                void *userdata) {
    if (stream && pa_stream_get_state(stream)
== PA_STREAM_READY) {
        pa_operation *o;
        if (!(o =
pa_stream_update_timing_info(stream,
stream_update_timing_callback, NULL)))
      1;
//pa_log(_("pa_stream_update_timing_info()
failed: %s"),
pa_strerror(pa_context_errno(context)));
        else
            pa_operation_unref(o);
    }

    pa_context_rttime_restart(context, e,
pa_rtclock_now() + TIME_EVENT_USEC);
}


void get_latency(pa_stream *s) {
  pa_usec_t latency;
  int neg;
  pa_timing_info *timing_info;
```

```
  timing_info = pa_stream_get_timing_info(s);

  if (pa_stream_get_latency(s, &latency, &neg)
!= 0) {
    fprintf(stderr, __FILE__":
pa_stream_get_latency() failed\n");
    return;
  }

  fprintf(stderr, "%0.0f usec    \r",
(float)latency);
}


/*********** Stream callbacks **************/

/* This is called whenever new data is
available */
static void stream_read_callback(pa_stream *s,
size_t length, void *userdata) {

  assert(s);
  assert(length > 0);

  // Copy the data from the server out to a
file
```

```
  //fprintf(stderr, "Can read %d\n", length);


  while (pa_stream_readable_size(s) > 0) {
    const void *data;
    size_t length;

    //get_latency(s);

    // peek actually creates and fills the data
vbl
    if (pa_stream_peek(s, &data, &length) < 0)
{
      fprintf(stderr, "Read failed\n");
      exit(1);
      return;
    }
    fprintf(stderr, "Writing %d\n", length);
    write(fdout, data, length);

    // swallow the data peeked at before
    pa_stream_drop(s);
  }
}


// This callback gets called when our context
```

```
changes state.  We really only
// care about when it's ready or if it has
failed
void state_cb(pa_context *c, void *userdata) {
  pa_context_state_t state;
  int *pa_ready = userdata;

  printf("State changed\n");
  state = pa_context_get_state(c);
  switch  (state) {
    // There are just here for reference
  case PA_CONTEXT_UNCONNECTED:
  case PA_CONTEXT_CONNECTING:
  case PA_CONTEXT_AUTHORIZING:
  case PA_CONTEXT_SETTING_NAME:
    break;
  case PA_CONTEXT_FAILED:
  case PA_CONTEXT_TERMINATED:
    *pa_ready = 2;
    break;
  case PA_CONTEXT_READY: {
    pa_buffer_attr buffer_attr;

    if (verbose)
      printf("Connection established.%s\n",
CLEAR_LINE);
```

```
    if (!(stream = pa_stream_new(c,
"JanCapture", &sample_spec, NULL))) {
      printf("pa_stream_new() failed: %s",
pa_strerror(pa_context_errno(c)));
      exit(1);
    }

    // Watch for changes in the stream state to
create the output file
    pa_stream_set_state_callback(stream,
stream_state_callback, NULL);

    // Watch for changes in the stream's read
state to write to the output file
    pa_stream_set_read_callback(stream,
stream_read_callback, NULL);

    // timing info
    pa_stream_update_timing_info(stream,
stream_update_timing_callback, NULL);

    // Set properties of the record buffer
    pa_zero(buffer_attr);
    buffer_attr.maxlength = (uint32_t) -1;
    buffer_attr.prebuf = (uint32_t) -1;

    if (latency_msec > 0) {
```

```
      buffer_attr.fragsize =
buffer_attr.tlength =
pa_usec_to_bytes(latency_msec *
PA_USEC_PER_MSEC, &sample_spec);
      flags |= PA_STREAM_ADJUST_LATENCY;
    } else if (latency > 0) {
      buffer_attr.fragsize =
buffer_attr.tlength = (uint32_t) latency;
      flags |= PA_STREAM_ADJUST_LATENCY;
    } else
      buffer_attr.fragsize =
buffer_attr.tlength = (uint32_t) -1;

    if (process_time_msec > 0) {
      buffer_attr.minreq =
pa_usec_to_bytes(process_time_msec *
PA_USEC_PER_MSEC, &sample_spec);
    } else if (process_time > 0)
      buffer_attr.minreq = (uint32_t)
process_time;
    else
      buffer_attr.minreq = (uint32_t) -1;

    flags |= PA_STREAM_INTERPOLATE_TIMING;

    get_latency(stream);
```

```
    // and start recording
    if (pa_stream_connect_record(stream,
device, &buffer_attr, flags) < 0) {
      printf("pa_stream_connect_record()
failed: %s", pa_strerror(pa_context_errno(c)));
      exit(1);
    }
    break;
  }
  default:
    return;
  }
}

int main(int argc, char *argv[]) {

  // Define our pulse audio loop and connection
variables
  pa_mainloop *pa_ml;
  pa_mainloop_api *pa_mlapi;
  pa_operation *pa_op;
  pa_time_event *time_event;

  // Create a mainloop API and connection to
the default server
  pa_ml = pa_mainloop_new();
  pa_mlapi = pa_mainloop_get_api(pa_ml);
```

```
  context = pa_context_new(pa_mlapi, "test");

  // This function connects to the pulse server
  pa_context_connect(context, NULL, 0, NULL);

  // This function defines a callback so the
server will tell us its state.
  pa_context_set_state_callback(context,
state_cb, NULL);

  if (!(time_event =
pa_context_rttime_new(context, pa_rtclock_now()
+ TIME_EVENT_USEC, time_event_callback, NULL)))
{
    //pa_log(_("pa_context_rttime_new()
failed."));
    //goto quit;
  }



  if (pa_mainloop_run(pa_ml, &ret) < 0) {
    printf("pa_mainloop_run() failed.");
    exit(1);
  }
}
```

Play microphone to speaker

Combining what we have so far, we get pa-mic-2-speaker.c:

```c
/*
 * Copy from microphone to speaker
 * pa-mic-2-speaker.c
 */


#include <stdio.h>
#include <string.h>
#include <pulse/pulseaudio.h>

#define CLEAR_LINE "\n"
#define BUFF_LEN 4096

// From pulsecore/macro.h
#define pa_memzero(x,l) (memset((x), 0, (l)))
#define pa_zero(x) (pa_memzero(&(x),
sizeof(x)))

static void *buffer = NULL;
static size_t buffer_length = 0, buffer_index =
0;


int verbose = 1;
int ret;
```

```
static pa_sample_spec sample_spec = {
  .format = PA_SAMPLE_S16LE,
  .rate = 44100,
  .channels = 2
};

static pa_stream *istream = NULL,
                 *ostream = NULL;

// This is my builtin card. Use paman to find
yours
//static char *device = "alsa_input.pci-
0000_00_1b.0.analog-stereo";
static char *idevice = NULL;
static char *odevice = NULL;

static pa_stream_flags_t flags = 0;

static size_t latency = 0, process_time = 0;
static int32_t latency_msec = 1,
process_time_msec = 0;

void stream_state_callback(pa_stream *s, void
*userdata) {
  assert(s);

  switch (pa_stream_get_state(s)) {
```

```
  case PA_STREAM_CREATING:
    // The stream has been created, so
    // let's open a file to record to
    printf("Creating stream\n");
    // fdout = creat(fname,  0711);
    buffer = pa_xmalloc(BUFF_LEN);
    buffer_length = BUFF_LEN;
    buffer_index = 0;
    break;

  case PA_STREAM_TERMINATED:
    // close(fdout);
    break;

  case PA_STREAM_READY:

    // Just for info: no functionality in this
branch
    if (verbose) {
      const pa_buffer_attr *a;
      char cmt[PA_CHANNEL_MAP_SNPRINT_MAX],
sst[PA_SAMPLE_SPEC_SNPRINT_MAX];

      printf("Stream successfully created.");

      if (!(a = pa_stream_get_buffer_attr(s)))
    printf("pa_stream_get_buffer_attr() failed:
```

```
%s",
pa_strerror(pa_context_errno(pa_stream_get_contex

      else {
    printf("Buffer metrics: maxlength=%u,
fragsize=%u", a->maxlength, a->fragsize);

      }

      printf("Connected to device %s (%u,
%ssuspended).",
         pa_stream_get_device_name(s),
         pa_stream_get_device_index(s),
         pa_stream_is_suspended(s) ? "" : "not
");
    }

    break;

  case PA_STREAM_FAILED:
  default:
    printf("Stream error: %s",
pa_strerror(pa_context_errno(pa_stream_get_contex

    exit(1);
  }
}
```

```
/*********** Stream callbacks **************/

/* This is called whenever new data is
available */
static void stream_read_callback(pa_stream *s,
size_t length, void *userdata) {

  assert(s);
  assert(length > 0);

  // Copy the data from the server out to a
file
  fprintf(stderr, "Can read %d\n", length);

  while (pa_stream_readable_size(s) > 0) {
    const void *data;
    size_t length, lout;

    // peek actually creates and fills the data
vbl
    if (pa_stream_peek(s, &data, &length) < 0)
{
      fprintf(stderr, "Read failed\n");
      exit(1);
      return;
```

```
    }

    fprintf(stderr, "read %d\n", length);
    lout =  pa_stream_writable_size(ostream);
    fprintf(stderr, "Writable: %d\n", lout);
    if (lout == 0) {
      fprintf(stderr, "can't write, zero
writable\n");
      return;
    }
    if (lout < length) {
      fprintf(stderr, "Truncating read\n");
      length = lout;
  }

  if (pa_stream_write(ostream, (uint8_t*) data,
length, NULL, 0, PA_SEEK_RELATIVE) < 0) {
    fprintf(stderr, "pa_stream_write()
failed\n");
    exit(1);
    return;
  }


    // STICK OUR CODE HERE TO WRITE OUT
    //fprintf(stderr, "Writing %d\n", length);
    //write(fdout, data, length);
```

```
    // swallow the data peeked at before
    pa_stream_drop(s);
  }
}



/* This is called whenever new data may be
written to the stream */
// We don't actually write anything this time
static void stream_write_callback(pa_stream *s,
size_t length, void *userdata) {
  //assert(s);
  //assert(length > 0);

  printf("Stream write callback: Ready to write
%d bytes\n", length);
 }


// This callback gets called when our context
changes state.  We really only
// care about when it's ready or if it has
failed
void state_cb(pa_context *c, void *userdata) {
  pa_context_state_t state;
```

```
  int *pa_ready = userdata;

  printf("State changed\n");
  state = pa_context_get_state(c);
  switch  (state) {
    // There are just here for reference
  case PA_CONTEXT_UNCONNECTED:
  case PA_CONTEXT_CONNECTING:
  case PA_CONTEXT_AUTHORIZING:
  case PA_CONTEXT_SETTING_NAME:
    break;
  case PA_CONTEXT_FAILED:
  case PA_CONTEXT_TERMINATED:
    *pa_ready = 2;
    break;
  case PA_CONTEXT_READY: {
    pa_buffer_attr buffer_attr;

    if (verbose)
      printf("Connection established.%s\n",
CLEAR_LINE);

    if (!(istream = pa_stream_new(c,
"JanCapture", &sample_spec, NULL))) {
      printf("pa_stream_new() failed: %s",
pa_strerror(pa_context_errno(c)));
      exit(1);
```

```
    }

    if (!(ostream = pa_stream_new(c,
"JanPlayback", &sample_spec, NULL))) {
      printf("pa_stream_new() failed: %s",
pa_strerror(pa_context_errno(c)));
      exit(1);
    }

    // Watch for changes in the stream state to
create the output file
    pa_stream_set_state_callback(istream,
stream_state_callback, NULL);

    // Watch for changes in the stream's read
state to write to the output file
    pa_stream_set_read_callback(istream,
stream_read_callback, NULL);

    pa_stream_set_write_callback(ostream,
stream_write_callback, NULL);



    // Set properties of the record buffer
    pa_zero(buffer_attr);
    buffer_attr.maxlength = (uint32_t) -1;
```

```
    buffer_attr.prebuf = (uint32_t) -1;

    if (latency_msec > 0) {
      buffer_attr.fragsize =
buffer_attr.tlength =
pa_usec_to_bytes(latency_msec *
PA_USEC_PER_MSEC, &sample_spec);
      flags |= PA_STREAM_ADJUST_LATENCY;
    } else if (latency > 0) {
      buffer_attr.fragsize =
buffer_attr.tlength = (uint32_t) latency;
      flags |= PA_STREAM_ADJUST_LATENCY;
    } else
      buffer_attr.fragsize =
buffer_attr.tlength = (uint32_t) -1;

    if (process_time_msec > 0) {
      buffer_attr.minreq =
pa_usec_to_bytes(process_time_msec *
PA_USEC_PER_MSEC, &sample_spec);
    } else if (process_time > 0)
      buffer_attr.minreq = (uint32_t)
process_time;
    else
      buffer_attr.minreq = (uint32_t) -1;

    // and start recording
```

```
    if (pa_stream_connect_record(istream,
idevice, &buffer_attr, flags) < 0) {
      printf("pa_stream_connect_record()
failed: %s", pa_strerror(pa_context_errno(c)));
      exit(1);
    }


    if (pa_stream_connect_playback(ostream,
odevice, &buffer_attr, flags,
                      NULL,
                      NULL) < 0) {
      printf("pa_stream_connect_playback()
failed: %s", pa_strerror(pa_context_errno(c)));
      exit(1); //goto fail;
    } else {
      printf("Set playback callback\n");
    }
    break;
  }
  default:
    return;
  }
}

int main(int argc, char *argv[]) {
```

```
  // Define our pulse audio loop and connection
variables
  pa_mainloop *pa_ml;
  pa_mainloop_api *pa_mlapi;
  pa_operation *pa_op;
  pa_context *pa_ctx;

  // Create a mainloop API and connection to
the default server
  pa_ml = pa_mainloop_new();
  pa_mlapi = pa_mainloop_get_api(pa_ml);
  pa_ctx = pa_context_new(pa_mlapi, "test");

  // This function connects to the pulse server
  pa_context_connect(pa_ctx, NULL, 0, NULL);

  // This function defines a callback so the
server will tell us its state.
  pa_context_set_state_callback(pa_ctx,
state_cb, NULL);

  if (pa_mainloop_run(pa_ml, &ret) < 0) {
    printf("pa_mainloop_run() failed.");
    exit(1);
  }
}
```

When the latency is set to 1 msec for everything, the actual latency is about 16-28 msec. I couldn't detect it.

Setting the volume on devices

Each device can have its input or output volume controlled by PulseAudio. The principal calls for sinks are `pa_context_set_sink_volume_by_name` and `pa_context_set_sink_volume_by_index` with similar calls for sources.

These calls make use of a structure `pa_cvolume`. This stucture can be manipulated using calls such as

- `pa_cvolume_init`
- `pa_cvolume_set`
- `pa_cvolume_mute`

In the following program we set the volume on a particular device by reading integer values from stdin and using these to set the value. Such a loop should probably best take place in a separate thread to the PulseAudio framework. Rather than

introducing application threading here, we make use of an alternative set of PulseAudio calls which set up a separate thread for the PulseAudio main loop. These calls are

- `pa_threaded_mainloop` instead of `pa_mainloop`
- `pa_threaded_mainloop_get_api` instead of `pa_mainloop_get_api`
- `pa_threaded_mainloop_start` instead of `pa_mainloop_start`

The threaded calls allow us to start PulseAudio in its own thread, and leave the current thread for reading volume values. This gives the relatively simple program pavolume.c:

```c
/**
 * pavolume.c
 * Jan Newmarch
 */

#include <stdio.h>
#include <string.h>
#include <pulse/pulseaudio.h>

#define _(x) x

char *device = "alsa_output.pci-
0000_00_1b.0.analog-stereo";

int ret;

pa_context *context;

void show_error(char *s) {
    fprintf(stderr, "%s\n", s);
}


void volume_cb(pa_context *c, int success, void
*userdata) {
    if (success)
```

```
    printf("Volume set\n");
    else
    printf("Volume not set\n");
}

void context_state_cb(pa_context *c, void
*userdata) {

    switch (pa_context_get_state(c)) {
    case PA_CONTEXT_UNCONNECTED:
    case PA_CONTEXT_CONNECTING:
    case PA_CONTEXT_AUTHORIZING:
    case PA_CONTEXT_SETTING_NAME:
    break;

    case PA_CONTEXT_READY: {
    pa_operation *o;

    break;
    }

    case PA_CONTEXT_FAILED:
    case PA_CONTEXT_TERMINATED:
    default:
    return;
    }
}
```

```
int main(int argc, char *argv[]) {
    long volume = 0;
    char buf[128];
    struct pa_cvolume v;

    // Define our pulse audio loop and
connection variables
    pa_threaded_mainloop *pa_ml;
    pa_mainloop_api *pa_mlapi;

    // Create a mainloop API and connection to
the default server
    //pa_ml = pa_mainloop_new();
    pa_ml = pa_threaded_mainloop_new();
    pa_mlapi =
pa_threaded_mainloop_get_api(pa_ml);
    context = pa_context_new(pa_mlapi, "Voulme
control");

    // This function connects to the pulse
server
    pa_context_connect(context, NULL, 0, NULL);

    // This function defines a callback so the
server will tell us its state.
    pa_context_set_state_callback(context,
```

```
context_state_cb, NULL);

    pa_threaded_mainloop_start(pa_ml);
    printf("Enter volume for device %s\n");

    pa_cvolume_init(&v);
    while (1) {
        puts("Enter an integer 0-65536\n");
        fgets(buf, 128, stdin);
        volume = atoi(buf);
    pa_cvolume_set(&v, 2, volume);
    pa_context_set_sink_volume_by_name(context,
                        device,
                        &v,
                        volume_cb,
                        NULL
                        );
    }
}
```

Listing clients

PulseAudio is a server that talks to devices at the bottom layer and to clients at the top layer. The clients are producers and consumers of audio. One of the roles of PulseAudio is to mix

signals from different source clients to shared output devices. In order to do this, PulseAudio keeps track of registrations by clients, and can make these available to *other* clients by suitbale callbacks.

The program palist_clients.c is very similar to the program palist_devices.c. The principal difference is that when the context changes state to `PA_CONTEXT_READY` the application subscribes to `PA_SUBSCRIPTION_MASK_CLIENT` instead of `(PA_SUBSCRIPTION_MASK_SINK|PA_SUBSCRIPTION_MASK_SOURCE)` and the subscription callback asks for `pa_context_get_client_info` instead of `pa_context_get_source_info`.

The program palist_clients.c is

```c
/**
 * palist_clients.c
 * Jan Newmarch
 */

#include <stdio.h>
#include <string.h>
#include <pulse/pulseaudio.h>

#define CLEAR_LINE "\n"
#define _(x) x

// From pulsecore/macro.h
//#define pa_memzero(x,l) (memset((x), 0, (l)))
//#define pa_zero(x) (pa_memzero(&(x),
sizeof(x)))

int ret;

pa_context *context;

void show_error(char *s) {
    /* stub */
}

void print_properties(pa_proplist *props) {
```

```
    void *state = NULL;

    printf("  Properties are: \n");
    while (1) {
    char *key;
    if ((key = pa_proplist_iterate(props,
&state)) == NULL) {
        return;
    }
    char *value = pa_proplist_gets(props, key);
    printf("   key %s, value %s\n", key,
value);
    }
}

void add_client_cb(pa_context *context, const
pa_client_info *i, int eol, void *userdata) {

    if (eol < 0) {
        if (pa_context_errno(context) ==
PA_ERR_NOENTITY)
            return;

        show_error(_("Client callback
failure"));
        return;
    }
```

```
    if (eol > 0) {
        return;
    }

    printf("Found a new client index %d name %s
eol %d\n", i->index, i->name, eol);
    print_properties(i->proplist);
}

void remove_client_cb(pa_context *context,
const pa_client_info *i, int eol, void
*userdata) {

    if (eol < 0) {
        if (pa_context_errno(context) ==
PA_ERR_NOENTITY)
            return;

        show_error(_("Client callback
failure"));
        return;
    }

    if (eol > 0) {
        return;
    }
```

```
    printf("Removing a client index %d name
%s\n", i->index, i->name);
    print_properties(i->proplist);
}

void subscribe_cb(pa_context *c,
pa_subscription_event_type_t t, uint32_t index,
void *userdata) {

    switch (t &
PA_SUBSCRIPTION_EVENT_FACILITY_MASK) {

    case PA_SUBSCRIPTION_EVENT_CLIENT:
    if ((t & PA_SUBSCRIPTION_EVENT_TYPE_MASK)
== PA_SUBSCRIPTION_EVENT_REMOVE) {
        printf("Remove event at index %d\n",
index);
        pa_operation *o;
        if (!(o = pa_context_get_client_info(c,
index, remove_client_cb, NULL))) {

show_error(_("pa_context_get_client_info()
failed"));
        return;
        }
        pa_operation_unref(o);
```

```
    } else {
        pa_operation *o;
        if (!(o = pa_context_get_client_info(c,
index, add_client_cb, NULL))) {

show_error(_("pa_context_get_client_info()
failed"));
        return;
        }
        pa_operation_unref(o);
    }
    break;
    }
}

void context_state_cb(pa_context *c, void
*userdata) {

    switch (pa_context_get_state(c)) {
    case PA_CONTEXT_UNCONNECTED:
    case PA_CONTEXT_CONNECTING:
    case PA_CONTEXT_AUTHORIZING:
    case PA_CONTEXT_SETTING_NAME:
    break;

    case PA_CONTEXT_READY: {
```

```
    pa_operation *o;

    pa_context_set_subscribe_callback(c,
subscribe_cb, NULL);

    if (!(o = pa_context_subscribe(c,
(pa_subscription_mask_t)

(PA_SUBSCRIPTION_MASK_CLIENT), NULL, NULL))) {
        show_error(_("pa_context_subscribe()
failed"));
        return;
    }
    pa_operation_unref(o);

    if (!(o =
pa_context_get_client_info_list(context,
                            add_client_cb,
                            NULL
    ) )) {
        show_error(_("pa_context_subscribe()
failed"));
        return;
    }
    pa_operation_unref(o);
```

```
    break;
    }

    case PA_CONTEXT_FAILED:
    return;

    case PA_CONTEXT_TERMINATED:
    default:
    return;
    }
}




void stream_state_callback(pa_stream *s, void
*userdata) {
    assert(s);

    switch (pa_stream_get_state(s)) {
    case PA_STREAM_CREATING:
    break;

    case PA_STREAM_TERMINATED:
    break;

    case PA_STREAM_READY:
```

```
    break;

    case PA_STREAM_FAILED:
    default:
    printf("Stream error: %s",
pa_strerror(pa_context_errno(pa_stream_get_contex

    exit(1);
    }
}

int main(int argc, char *argv[]) {

    // Define our pulse audio loop and
connection variables
    pa_mainloop *pa_ml;
    pa_mainloop_api *pa_mlapi;
    pa_operation *pa_op;
    pa_time_event *time_event;

    // Create a mainloop API and connection to
the default server
    pa_ml = pa_mainloop_new();
    pa_mlapi = pa_mainloop_get_api(pa_ml);
    context = pa_context_new(pa_mlapi, "test");
```

```
    // This function connects to the pulse
server
    pa_context_connect(context, NULL, 0, NULL);

    // This function defines a callback so the
server will tell us its state.
    //pa_context_set_state_callback(context,
state_cb, NULL);
    pa_context_set_state_callback(context,
context_state_cb, NULL);


    if (pa_mainloop_run(pa_ml, &ret) < 0) {
    printf("pa_mainloop_run() failed.");
    exit(1);
    }
}
```

The output on my system is (elided)

```
Found a new client index 0 name ConsoleKit
Session /org/freedesktop/ConsoleKit/Session2
eol 0
  Properties are:
   key application.name, value ConsoleKit
Session /org/freedesktop/ConsoleKit/Session2
   key console-kit.session, value
/org/freedesktop/ConsoleKit/Session2
Found a new client index 4 name XSMP Session on
gnome-session as
1057eba7239ba1ec3d1363598095985901000000018790044
 eol 0
  Properties are:
   key application.name, value XSMP Session on
gnome-session as
1057eba7239ba1ec3d1363598095985901000000018790044

   key xsmp.vendor, value gnome-session
   key xsmp.client.id, value
1057eba7239ba1ec3d1363598095985901000000018790044

Found a new client index 5 name GNOME Volume
Control Media Keys eol 0
  Properties are:
   ...
Found a new client index 7 name GNOME Volume
```

```
Control Applet eol 0
  Properties are:
   ...
Found a new client index 53 name Metacity eol 0
  Properties are:
  ...
Found a new client index 54 name Firefox eol 0
  Properties are:
    ...
Found a new client index 248 name PulseAudio
Volume Control eol 0
  Properties are:
    ...
Found a new client index 341 name test eol 0
  Properties are:
    ...
```

Listing client sources and sinks

Clients can act as sources: programs such as `mplayer` and `vlc` do just that, sending streams to PulseAudio. Other clients can act as sinks. The clients themselves are monitored by the previous program. To monitor their *activity* you set the mask on `pa_subscribe_callback` to `(PA_SUBSCRIPTION_MASK_CLIENT |`

`PA_SUBSCRIPTION_MASK_SINK_INPUT | PA_SUBSCRIPTION_MASK_SOURCE_OUTPUT)` . Within the subscription callback you make calls to `pa_context_get_sink_input_info` within the `PA_SUBSCRIPTION_EVENT_SINK_INPUT` branch and similarly for the source output.

The sink input callback is passed the structure `pa_sink_input_info` . This contains the familiar `name` and `index` fields but also has an integer field `client` . This links the sink input back to the index of the client responsible for the sink. In the following program we list all the clients as well, so that these links can followed visually. Programatically, PulseAudio makes you keep much information (such as what clients have what indices) yourself: this is ignored here.

The program to list clients and monitor changes in their input and output streams is pamonitor_clients.c:

```c
/**
 * pamonitor_clients.c
 * Jan Newmarch
 */



#include <stdio.h>
#include <string.h>
#include <pulse/pulseaudio.h>

#define CLEAR_LINE "\n"
#define _(x) x

// From pulsecore/macro.h
#define pa_memzero(x,l) (memset((x), 0, (l)))
#define pa_zero(x) (pa_memzero(&(x),
sizeof(x)))

int ret;

pa_context *context;

void show_error(char *s) {
    /* stub */
}
```

```
void add_client_cb(pa_context *context, const
pa_client_info *i, int eol, void *userdata) {

    if (eol < 0) {
        if (pa_context_errno(context) ==
PA_ERR_NOENTITY)
            return;

        show_error(_("Client callback
failure"));
        return;
    }

    if (eol > 0) {
        return;
    }

    printf("Found a new client index %d name %s
eol %d\n", i->index, i->name, eol);
}

void remove_client_cb(pa_context *context,
const pa_client_info *i, int eol, void
*userdata) {

    if (eol < 0) {
```

```
        if (pa_context_errno(context) ==
PA_ERR_NOENTITY)
            return;

        show_error(_("Client callback
failure"));
        return;
    }

    if (eol > 0) {
        return;
    }

    printf("Removing a client index %d name
%s\n", i->index, i->name);
}

void sink_input_cb(pa_context *c, const
pa_sink_input_info *i, int eol, void *userdata)
{
    if (eol < 0) {
        if (pa_context_errno(context) ==
PA_ERR_NOENTITY)
            return;

        show_error(_("Sink input callback
failure"));
```

```
        return;
    }

    if (eol > 0) {
        return;
    }
    printf("Sink input found index %d name %s
for client %d\n", i->index, i->name, i-
>client);
}

void source_output_cb(pa_context *c, const
pa_source_output_info *i, int eol, void
*userdata) {
    if (eol < 0) {
        if (pa_context_errno(context) ==
PA_ERR_NOENTITY)
            return;

        show_error(_("Source output callback
failure"));
        return;
    }

    if (eol > 0) {
        return;
    }
```

```
    printf("Source output found index %d name
%s for client %d\n", i->index, i->name, i-
>client);
}


void subscribe_cb(pa_context *c,
pa_subscription_event_type_t t, uint32_t index,
void *userdata) {

    switch (t &
PA_SUBSCRIPTION_EVENT_FACILITY_MASK) {

    case PA_SUBSCRIPTION_EVENT_CLIENT:
    if ((t & PA_SUBSCRIPTION_EVENT_TYPE_MASK)
== PA_SUBSCRIPTION_EVENT_REMOVE) {
        printf("Remove event at index %d\n",
index);
        pa_operation *o;
        if (!(o = pa_context_get_client_info(c,
index, remove_client_cb, NULL))) {

show_error(_("pa_context_get_client_info()
failed"));
        return;
        }
        pa_operation_unref(o);
```

```
    } else {
        pa_operation *o;
        if (!(o = pa_context_get_client_info(c,
index, add_client_cb, NULL))) {

show_error(_("pa_context_get_client_info()
failed"));
        return;
        }
        pa_operation_unref(o);
    }
    break;

    case PA_SUBSCRIPTION_EVENT_SINK_INPUT:
    if ((t & PA_SUBSCRIPTION_EVENT_TYPE_MASK)
== PA_SUBSCRIPTION_EVENT_REMOVE)
        printf("Removing sink input %d\n",
index);
    else {
        pa_operation *o;
        if (!(o =
pa_context_get_sink_input_info(context, index,
sink_input_cb, NULL))) {

show_error(_("pa_context_get_sink_input_info()
failed"));
```

```
        return;
        }
        pa_operation_unref(o);
    }
    break;

    case PA_SUBSCRIPTION_EVENT_SOURCE_OUTPUT:
    if ((t & PA_SUBSCRIPTION_EVENT_TYPE_MASK)
== PA_SUBSCRIPTION_EVENT_REMOVE)
        printf("Removing source output %d\n",
index);
    else {
        pa_operation *o;
        if (!(o =
pa_context_get_source_output_info(context,
index, source_output_cb, NULL))) {

show_error(_("pa_context_get_sink_input_info()
failed"));
        return;
        }
        pa_operation_unref(o);
    }
    break;
    }
}
```

```
void context_state_cb(pa_context *c, void
*userdata) {

    switch (pa_context_get_state(c)) {
    case PA_CONTEXT_UNCONNECTED:
    case PA_CONTEXT_CONNECTING:
    case PA_CONTEXT_AUTHORIZING:
    case PA_CONTEXT_SETTING_NAME:
    break;

    case PA_CONTEXT_READY: {
    pa_operation *o;

    pa_context_set_subscribe_callback(c,
subscribe_cb, NULL);

    if (!(o = pa_context_subscribe(c,
(pa_subscription_mask_t)

(PA_SUBSCRIPTION_MASK_CLIENT |

PA_SUBSCRIPTION_MASK_SINK_INPUT |

PA_SUBSCRIPTION_MASK_SOURCE_OUTPUT), NULL,
NULL))) {
        show_error(_("pa_context_subscribe()
failed"));
```

```
        return;
    }
    pa_operation_unref(o);

    if (!(o =
pa_context_get_client_info_list(context,
                                add_client_cb,
                                NULL
    ) )) {
        show_error(_("pa_context_subscribe()
failed"));
        return;
    }
    pa_operation_unref(o);

    break;
    }

    case PA_CONTEXT_FAILED:
    return;

    case PA_CONTEXT_TERMINATED:
    default:
    // Gtk::Main::quit();
    return;
    }
}
```

```
void stream_state_callback(pa_stream *s, void
*userdata) {
    assert(s);

    switch (pa_stream_get_state(s)) {
    case PA_STREAM_CREATING:
    break;

    case PA_STREAM_TERMINATED:
    break;

    case PA_STREAM_READY:
    break;

    case PA_STREAM_FAILED:
    default:
    printf("Stream error: %s",
pa_strerror(pa_context_errno(pa_stream_get_contex

    exit(1);
    }
}
```

```c
int main(int argc, char *argv[]) {

    // Define our pulse audio loop and
connection variables
    pa_mainloop *pa_ml;
    pa_mainloop_api *pa_mlapi;
    pa_operation *pa_op;
    pa_time_event *time_event;

    // Create a mainloop API and connection to
the default server
    pa_ml = pa_mainloop_new();
    pa_mlapi = pa_mainloop_get_api(pa_ml);
    context = pa_context_new(pa_mlapi, "test");

    // This function connects to the pulse
server
    pa_context_connect(context, NULL, 0, NULL);

    // This function defines a callback so the
server will tell us its state.
    //pa_context_set_state_callback(context,
state_cb, NULL);
    pa_context_set_state_callback(context,
context_state_cb, NULL);
```

```
    if (pa_mainloop_run(pa_ml, &ret) < 0) {
    printf("pa_mainloop_run() failed.");
    exit(1);
    }
}
```

The output on my system is

```
Found a new client index 0 name ConsoleKit
Session /org/freedesktop/ConsoleKit/Session2
eol 0
Found a new client index 4 name XSMP Session on
gnome-session as
1057eba7239ba1ec3d1363598095985901000000018790044
 eol 0
Found a new client index 5 name GNOME Volume
Control Media Keys eol 0
Found a new client index 7 name GNOME Volume
Control Applet eol 0
Found a new client index 53 name Metacity eol 0
Found a new client index 54 name Firefox eol 0
Found a new client index 248 name PulseAudio
Volume Control eol 0
Found a new client index 342 name test eol 0
```

Controlling the volume of a sink client

One of the significant features of PulseAudio is that not only can it mix streams to a device, but it can also control the volume of each stream. This is in addition to the volume control of each device. In `pavucontrol` you can see this under the Playback tab, where the volume of playback clients can be adjusted.

Programatically this is done by calling `pa_context_set_sink_input_volume` with parameters the index of the sink input and the volume. In the following program we follow what we did in the pavolume_client.c program where we set PulseAudio to run in a separate thread and input values for the volume in the main thread. A slight difference is that we have to wait for a client to start up a sink input, which we do by sleeping until the sink input callback assigns a non-zero value to the `sink_index` variable. Crude, yes. In a program such as `pavucontrol` the GUI runs in separate threads anyway and we do not need to resort to such simple tricks.

The program is pavolume_sink.c . If you play a file using e.g. `mplayer` then its volume can be adjusted by this program.

```c
/**
 * pavolume_sink.c
 * Jan Newmarch
 */

#include <stdio.h>
#include <string.h>
#include <pulse/pulseaudio.h>

#define CLEAR_LINE "\n"
#define _(x) x

int ret;

// sink we will control volume on when it is
non-zero
int sink_index = 0;
int sink_num_channels;

pa_context *context;

void show_error(char *s) {
    /* stub */
}

void sink_input_cb(pa_context *c, const
```

```
pa_sink_input_info *i, int eol, void *userdata)
{
    if (eol < 0) {
        if (pa_context_errno(context) ==
PA_ERR_NOENTITY)
            return;

        show_error(_("Sink input callback
failure"));
        return;
    }

    if (eol > 0) {
        return;
    }
    printf("Sink input found index %d name %s
for client %d\n", i->index, i->name, i-
>client);
    sink_num_channels = i-
>channel_map.channels;
    sink_index = i->index;
}

void volume_cb(pa_context *c, int success, void
*userdata) {
    if (success)
    printf("Volume set\n");
```

```
    else
    printf("Volume not set\n");
}

void subscribe_cb(pa_context *c,
pa_subscription_event_type_t t, uint32_t index,
void *userdata) {

    switch (t &
PA_SUBSCRIPTION_EVENT_FACILITY_MASK) {

    case PA_SUBSCRIPTION_EVENT_SINK_INPUT:
    if ((t & PA_SUBSCRIPTION_EVENT_TYPE_MASK)
== PA_SUBSCRIPTION_EVENT_REMOVE)
        printf("Removing sink input %d\n",
index);
    else {
        pa_operation *o;
        if (!(o =
pa_context_get_sink_input_info(context, index,
sink_input_cb, NULL))) {

show_error(_("pa_context_get_sink_input_info()
failed"));
        return;
        }
        pa_operation_unref(o);
```

```
    }
    break;
    }
}

void context_state_cb(pa_context *c, void
*userdata) {

    switch (pa_context_get_state(c)) {
    case PA_CONTEXT_UNCONNECTED:
    case PA_CONTEXT_CONNECTING:
    case PA_CONTEXT_AUTHORIZING:
    case PA_CONTEXT_SETTING_NAME:
    break;

    case PA_CONTEXT_READY: {
    pa_operation *o;

    pa_context_set_subscribe_callback(c,
subscribe_cb, NULL);

    if (!(o = pa_context_subscribe(c,
(pa_subscription_mask_t)

(PA_SUBSCRIPTION_MASK_SINK_INPUT), NULL,
NULL))) {
        show_error(_("pa_context_subscribe()
```

```
failed"));
        return;
    }
    break;
    }

    case PA_CONTEXT_FAILED:
    return;

    case PA_CONTEXT_TERMINATED:
    default:
    // Gtk::Main::quit();
    return;
    }
}

void stream_state_callback(pa_stream *s, void
*userdata) {
    assert(s);

    switch (pa_stream_get_state(s)) {
    case PA_STREAM_CREATING:
    break;

    case PA_STREAM_TERMINATED:
    break;
```

```
    case PA_STREAM_READY:
    break;

    case PA_STREAM_FAILED:
    default:
    printf("Stream error: %s",
pa_strerror(pa_context_errno(pa_stream_get_contex

    exit(1);
    }
}

int main(int argc, char *argv[]) {

    // Define our pulse audio loop and
connection variables
    pa_threaded_mainloop *pa_ml;
    pa_mainloop_api *pa_mlapi;
    pa_operation *pa_op;
    pa_time_event *time_event;
    long volume = 0;
    char buf[128];
    struct pa_cvolume v;

    // Create a mainloop API and connection to
the default server
    pa_ml = pa_threaded_mainloop_new();
```

```
    pa_mlapi =
pa_threaded_mainloop_get_api(pa_ml);
    context = pa_context_new(pa_mlapi, "test");

    // This function connects to the pulse
server
    pa_context_connect(context, NULL, 0, NULL);

    // This function defines a callback so the
server will tell us its state.
    //pa_context_set_state_callback(context,
state_cb, NULL);
    pa_context_set_state_callback(context,
context_state_cb, NULL);


    pa_threaded_mainloop_start(pa_ml);

    /* wait till there is a sink */
    while (sink_index == 0) {
    sleep(1);
    }

    printf("Enter volume for sink %d\n",
sink_index);
    pa_cvolume_init(&v);
    while (1) {
```

```
        puts("Enter an integer 0-65536");
        fgets(buf, 128, stdin);
        volume = atoi(buf);
    pa_cvolume_set(&v, sink_num_channels,
volume);
    pa_context_set_sink_input_volume(context,
                                     sink_index,
                                     &v,
                                     volume_cb,
                                     NULL
                                     );
    }
}
```

Conclusion

This chapter has looked at PulseAudio. This is currently the standard sound system for consumer Linux. There are a number of utilities for exploring PulseAudio. There are two APIs: the simple API and the asynchronous API. The chapter has looked at playing and recording using these APIs. Some other aspects of PulseAudio were also examined.

Latency is not a goal, and it is not designed for real-time audio. However, you can request that the latency be made small, and if PulseAudio can do it will give you reasonable performance. However, PulseAudio makes no guarantees about latency, so if a maximum latency is critical then PulseAudio may not be suitable.

PulseAudio is presently built on top of ALSA and usually interacts by making itself the default ALSA plugin.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

Jack is intended for professional audio, connecting many devices in arbitrary configurations. All software components run synchronously, with effectively zero latency.

Introduction

The article Knowing Jack gives a gentle introduction to Jack.

Resources

- The Jack API
- Knowing Jack
- ArchLinux Pro Audio
- Gentoo Jack
- Paul Davis talk on Jack architecture
- Knowing Jack by Dave Phillips in Linux Magazine

Starting Jack

The Jack server is `jackd`. It has one required parameter which is the sound backend such as ALSA. The minimal command is

```
jackd -dalsa
```

if you are using a normal Linux distro such as Fedora or Ubuntu, this will quite likely fail if the PulseAudio system is running. This may need to be stopped, or at least paused while you run Jack. See the previous chapter for stopping PulseAudio, To pause it, I usually run this in a terminal window:

```
pasuspender cat
```

This will pause PulseAudio until `cat` terminates, which it will do when you enter ctrl-d.

`jackd` will try to start using the Linux real-time scheduler. If you want to run without it, use the option

```
jackd --no-realtime -dalsa
```

If you want to run with the realtime scheduler, there are several ways:

- Run the server from the root user

  ```
  sudo jackd -dalsa
  ```

- Add a user to the `audio` and `jackuser` group by e.g.

  ```
  useradd -G audio newmarch
  useradd -G jackuser newmarch
  ```

  (You will need to logout and back in before this takes effect.)

Note that if you run the server as the root user, then you will not be able to connect to it from clients that are not in the `jackuser` group.

No apparent systemd or upstart scripts exist for Jack, but there are instructions about starting Jack at boot time from Gentoo jack :

```
#!/sbin/runscript
 # This programm will be used by init in order
to launch jackd with the privileges
 # and id of the user defined into
/etc/conf.d/jackd

 depend() {
    need alsasound
 }

 start() {
    if ! test -f "${JACKDHOME}/.jackdrc"; then
        eerror "You must start and configure
jackd before launch it. Sorry."
        eerror "You can use qjackctl for that."
        return 1
    else JACKDOPTS=$(cat
"${JACKDHOME}/.jackdrc"|sed -e
's\/usr/bin/jackd \\')
    fi

    if [ -e /var/run/jackd.pid ]; then
         rm /var/run/jackd.pid
    fi

    ebegin "Starting JACK Daemon"
```

```
    env HOME="${JACKDHOME}" start-stop-daemon -
-start \
        --quiet --background \
        --make-pidfile --pidfile
/var/run/jackd.pid \
        -c ${JACKDUSER} \
        -x /usr/bin/jackd -- ${JACKDOPTS}
>${LOG}

    sleep 2
    if ! pgrep -u ${JACKDUSER} jackd >
/dev/null; then
         eerror "JACK daemon can't be started!
Check logfile: ${LOG}"
     fi
     eend $?
 }

 stop() {
     ebegin "Stopping JACK daemon -- please
wait"
     start-stop-daemon --stop --pidfile
/var/run/jackd.pid &>/dev/null
     eend $?
 }

 restart() {
```

```
      svc_stop
      while `pgrep -u ${JACKDUSER} jackd
>/dev/null`; do
          sleep 1
      done
      svc_start
 }
```

File: /etc/conf.d/jackd:

```
# owner of jackd process (Must be an existing
user.)
 JACKDUSER="dom"

 # .jackdrc location for that user (Must be
existing, JACKDUSER can use
 # qjackctl in order to create it.)
 JACKDHOME="/home/${JACKDUSER}"

 # logfile (/dev/null for nowhere)
 LOG=/var/log/jackd.log
```

Create and save those 2 files. Don't forget to adjust JACKDUSER to the wanted user name (the same as yours I guess).

We need to make /etc/init.d/jackd executable:

```
# chmod +x /etc/init.d/jackd
```

Adding the script into the default run-level:

```
# rc-update add jackd default
```

Before restarting your system or starting this script, you must be sure that jackd is configured for $JACKUSER or jackd will fail. This is because the script will read /home/${USER}/.jackdrc. If this file doesn't exist, the easiest way to create it is to run QJackCtl as explained above.

Note on Realtime: Due to a limitation in the implementation of start-stop-daemon, it is not possible to start jackd in realtime mode as a non-root user by this method if using pam_limits. start-stop-daemon does not implement support for pam_sessions, meaning that changes to limits.conf have no effect in this context.

User tools

There is really only one tool that you need to use with Jack: `qjackctl`. This gives a graphical view of which Jack applications are playing and allows you to link inputs and outputs.

A simple tutorial on using qjackctl is HowToQjackCtlConnections It is actually amazingly simple to use: click on a source, link it to a destination by clicking on that. A line will be shown linking them - and that's all you have to do. Many Jack applications will do this for you, so you just observe the results.

Applications using Jack

There are many pieces of software using Jack, described in Applications using JACK

mplayer

To run `mplayer` using Jack, add the option `-ao jack`:

```
mplayer -ao jack 54154.mp3
```

`mplayer` used in this way will connect to the Jack `system` output device. To output to another Jack application such as `jack-rack`, append the output application to the audio output command

```
mplayer -ao jack:port=jack_rack 54154.mp3
```

VLC

VLC will play to Jack output if the Jack module is included. This is available as a downloadable Debian package `vlc-plugin-jack`. You can check if you have it by seeing if jack is listed as a module in `vlc --list` shows ALSA but not Jack.

Play a file using Jack by e.g.

```
vlc --aout jack 54154.mp3
```

You should be able to connect to a particular Jack application using the option `--jack-connect-regex <string>`.

TiMidity

TiMidity is a MIDI player discussed later. It can play to Jack output devices by

```
timidity -Oj 54154.mid
```

Jack-supplied programs

Jack comes with a large number of clients:

```
jack_alias                    jack_midisine
jack_bufsize                  jack_monitor_client
jack_connect                  jack_multiple_metro
jack_control                  jack_net_master
jack_cpu                      jack_net_slave
jack_cpu_load                 jack_netsource
jackd                         jack_rec
jackdbus                      jack_samplerate
jack_disconnect               jack_server_control
jack_evmon                    jack_session_notify
jack_freewheel                jack_showtime
jack_iodelay                  jack_simple_client
jack_latent_client
jack_simple_session_client
jack_load                     jack_test
jack_lsp                      jack_thru
jack_metro                    jack_transport
jack_midi_dump                jack_unload
jack_midi_latency_test        jack_wait
jack_midiseq                  jack_zombie
```

For many of these the source code is available in the Jack source code distribution and there is a `man` page for each one.

Running, say, `jack_thru` connects the system capture ports to the `jack_thru` input ports and the `jack_thru` output ports to the system playback ports. You can then do things such as disconnect ports using "client:port" for the port name as in

```
jack_disconnect jack_thru:output_1
system:playback_1
```

These command line tools allow you to do the same kind of actions as `qjackctl`

Other Jack programs

The page Applications using JACK lists many applications using Jack.

The page Jack MIDI Apps at linuxaudio.org lists many MIDI applications using Jack

Using a different soundcard

The default ALSA device for Jack will be hw:0. If you wish to use a different soundcard then you can spcecify this when starting Jack as in

```
jackd -dalsa -dhw:0
```

I have a USB Sound Blaster card, which requires some extra parameters

```
jackd -dalsa -dhw:2 -r 48000 -S
```

This doesn't work great - I get a regular "ticking" sound.

Without the `-S` (16-bit) flag I just get the cryptic

```
ALSA: cannot set hardware parameters for
playback
```

Alternatively, I can run

```
jackd -dalsa -dplughw:2 -r 48000
```

When I start it this way, Jack advises against using ALSA plug devices but it works best so far.

How can I use multiple soundcards with JACK?

See How can I use multiple soundcards with JACK?

Mixing audio

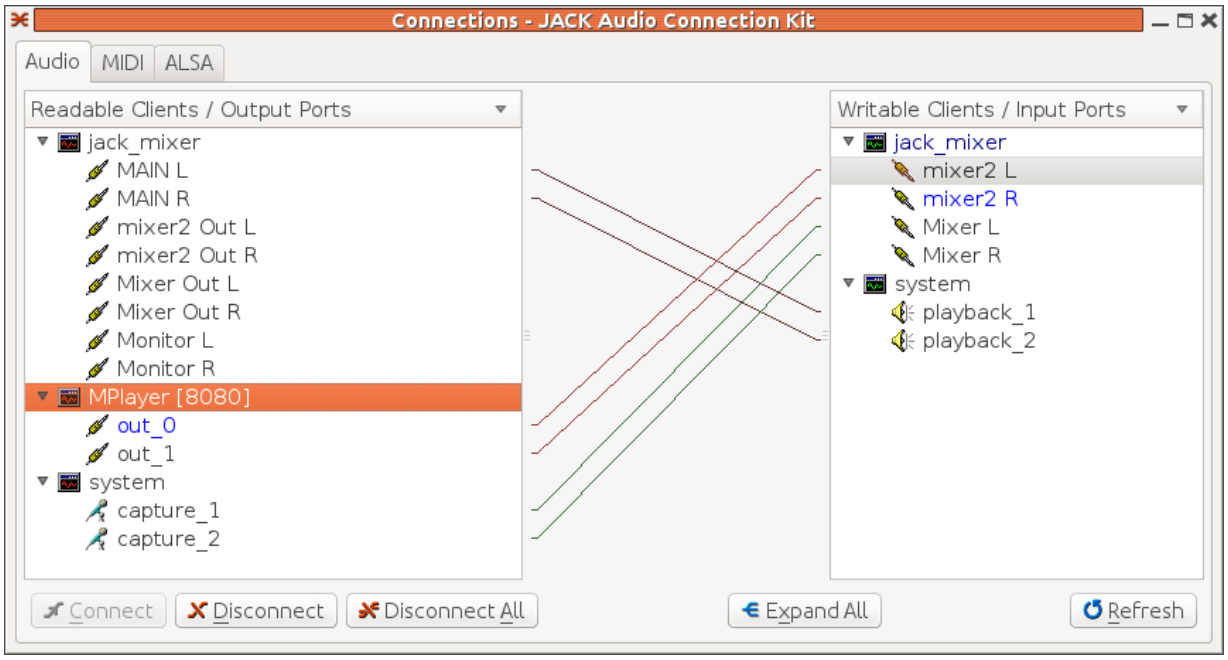
If two output ports from two different sources are connected to the same input port, then Jack will mix them for you. This allows you to sing along to your favourite MP3 file with no effort:

- Connect the microphone capture ports to the playback ports - avoid setting up a feedback loop between your laptop's microphone and speakers by e.g plugging in headphones
- Start a player such as `mplayer` which will also connect to the playback ports by e.g.

  ```
  mplayer -ao jack <MP3 file >
  ```

- Start singing

Of course, there is no volume control on each source. You can insert a mixer such as jack_mixer (maybe in your distro too), and then use that to control the volume of each source, as shown in this `qjackctl` screen:


Connections - JACK Audio Connection Kit
Audio
MIDI
ALSA
Readable Clients / Output Ports
jack_mixer
MAIN L
MAIN R
mixer2 Out L
mixer2 Out R
Mixer Out L
Mixer Out R
Monitor L
Monitor R
MPlayer [8080]
out_0
out_1
system
capture_1
capture_2
Writable Clients / Input Ports
jack_mixer
mixer2 L
mixer2 R
Mixer L
Mixer R
system
playback_1
playback_2
Connect
Disconnect
Disconnect All
Expand All
Refresh

Writing Audio Applications With JACK

See Writing Audio Applications With JACK - A tutorial/journal

The design of Jack is discussed at The JACK Audio Connection Kit by its primary author Paul Davis. The goals are

- Jack should allow streaming of low-latency, high-bandwidth data between independent applications.
- Although not a requirement, Jack should support any streaming data type, not just audio.
- In an active Jack setup, there will be one server and one or more Jack plugins. It will be possible to run multiple Jack servers, but each server will form an independent Jack setup. Jack will not define any interfaces between Jack servers.
- Applications connected using Jack may have their own graphical interfaces. Jack will not make any specifications as to different GUI toolkits or libraries. As a consequence of this requirement, different parts of a running Jack setup may be spread across multiple processes.
- Jack should provide full, sample accurate synchronation (ie. totally synchronous execution of all client plugins)

- To represent audio data, Jack should use 32 bit IEEE floats, normalized to value range [-1,1].
- Only noninterleaved audio streams will be supported.
- One Jack client may consume or produce multiple data streams.
- The Jack API should be specified in ANSI C. There are no restrictions on how servers and clients are to be implemented.
- It should be possible to connect already running applications.
- It should be possible to add or remove Jack clients while the server is running.

To pick the eyes out of this, the principal goals are

- Jack should allow streaming of low-latency, high-bandwidth data between independent applications.
- Jack should provide full, sample accurate synchronation (ie. totally synchronous execution of all client plugins)

The second is guaranteed by the Jack framework. The first is supplied by the Jack framework - as long as the applications are coded correctly.

Under the hood Jack uses fast Linux (Unix) pipelines to stream data from one aplication to another. Within each Jack application is a realtime loop which takes data off the input pipe and sends data to the output pipe. To avoid latency delays, there should essentially be no (or as little as possible) processing between reading and writing data - the ideal would be to pass pointer data from input to output, or at most to just do a `memcpy` .

So how can processing be done? Copy the data read to another data structure and pass processing off to another thread, or copy data processed in another thread to the ouput pipe. Anything else will cause latency which may become noticeable. In particular, certain system calls are essentially banned: `malloc` can cause swapping; `sleep` is an obvious no-no; `read/write` etc can cause disk I/O; `pthread_cond_wait` will ... wait.

Jack applications are inherently multi-threaded. In a Linux world this means Posix threads, and fortunately there is a book PThreads Primer by Bil Lewis and Daniel J Berg to tell you all about Posix threads!

There are mechanisms to set up a Jack application:

- Open a connection to a Jack server: `jack_client_open`
- Examine the status of the connection and bail out if needed

- Install a process callback handler to manage I/O: `jack_set_process_callback`
- Install a shutdown callback: `jack_on_shutdown`
- Register input and output ports with the Jack server: `jack_port_register`. Note that each port only carries a mono channel, so for stereo you will get two input ports, etc. This does *not* as yet link them to the pipelines
- Activate the ports i.e. tell Jack to start its processing thread: `jack_activate`
- Connect the ports to the pipelines: `jack_connect`
- Sit there in some way - for a text client just sleep in a loop. A GUI client might have a GUI processing loop

Libraries

The following examples need to be linked to various libraries. So far, they need to be

```
LIBS = -ljack -ljackserver -lasound -lm -
lpthread -lsndfile
```

Port information

Jack uses ports which carry mono 32-bit data. Each port has a name as a string, and poperties such as input and output. Once a connection to a Jack server has been made, queries for ports known to the server can be made using `jack_get_ports`. If the arguments are `NULL` or zero then al ports are returned, or patterns can be used to restrict the port names returned. Once a port name is found it can be turned into a `jack_port_t` and its properties can be queried.

A program to do this is `listports.c`:

```c
/** @file delay.c
 *
 * @brief This client delays one channel by
4096 framse.
 */

#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <signal.h>
#ifndef WIN32
#include <unistd.h>
#endif
#include <jack/jack.h>

jack_client_t *client;

void print_port_info(char *name) {
    printf("Port name is %s\n", name);
    jack_port_t *port =    jack_port_by_name
(client, name);
    if (port == NULL) {
    printf("No port by name %s\n", name);
    return;
```

```
    }
    printf("  Type is %s\n",
jack_port_type(port));

    int flags = jack_port_flags(port);
    if (flags & JackPortIsInput)
    printf("  Is an input port\n");
    else
    printf("  Is an output port\n");
    char **connections =
jack_port_get_connections(port);
    char **c = connections;
    printf("  Connected to:\n");
    while ((c != NULL) && (*c != NULL)) {
    printf("    %s\n", *c++);
    }
    if (connections != NULL)
    jack_free(connections);
}

int
main ( int argc, char *argv[] )
{
    int i;
    const char **ports;
    const char *client_name;
    const char *server_name = NULL;
```

```
    jack_options_t options = JackNullOption;
    jack_status_t status;

    if ( argc >= 2 )        /* client name
specified? */
    {
        client_name = argv[1];
        if ( argc >= 3 )    /* server name
specified? */
        {
            server_name = argv[2];
            options |= JackServerName;
        }
    }
    else              /* use basename of
argv[0] */
    {
        client_name = strrchr ( argv[0], '/' );
        if ( client_name == 0 )
        {
            client_name = argv[0];
        }
        else
        {
            client_name++;
        }
    }
```

```
    /* open a client connection to the JACK
server */

    client = jack_client_open ( client_name,
options, &status, server_name );
    if ( client == NULL )
    {
        fprintf ( stderr, "jack_client_open()
failed, "
                  "status = 0x%2.0x\n", status
);
        if ( status & JackServerFailed )
        {
            fprintf ( stderr, "Unable to
connect to JACK server\n" );
        }
        exit ( 1 );
    }
    if ( status & JackServerStarted )
    {
        fprintf ( stderr, "JACK server
started\n" );
    }
    if ( status & JackNameNotUnique )
    {
        client_name = jack_get_client_name (
```

```
client );
        fprintf ( stderr, "unique name `%s'
assigned\n", client_name );
    }

    if ( jack_activate ( client ) )
    {
        fprintf ( stderr, "cannot activate
client" );
        exit ( 1 );
    }

     ports = jack_get_ports ( client, NULL,
NULL, 0 );
    if ( ports == NULL )
    {
        fprintf ( stderr, "no ports\n" );
        exit ( 1 );
    }
    char **p = ports;
    while (*p != NULL)
    print_port_info(*p++);
    jack_free(ports);

    jack_client_close ( client );
    exit ( 0 );
}
```

Copy input to output

The Jack source code distribution has an "example clients" subdirectory. Included in there is a client `thru_client.c` which just copies input to output. The processing heart of this example is the function `process`. This takes a number of frames available on both input and output as parameter. This loops through the (stereo) channels, gets corresponding input and output buffers (for input and output pipelines) and copies data from input to corresponding output.

The code is

```
/** @file thru_client.c
 *
 * @brief This simple through client
demonstrates the basic features of JACK
 * as they would be used by many applications.
 */

#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <signal.h>
#ifndef WIN32
#include <unistd.h>
#endif
#include <jack/jack.h>

jack_port_t **input_ports;
jack_port_t **output_ports;
jack_client_t *client;

static void signal_handler ( int sig )
{
    jack_client_close ( client );
    fprintf ( stderr, "signal received, exiting
```

```
...\n" );
    exit ( 0 );
}

/**
 * The process callback for this JACK
application is called in a
 * special realtime thread once for each audio
cycle.
 *
 * This client follows a simple rule: when the
JACK transport is
 * running, copy the input port to the output.
When it stops, exit.
 */

int
process ( jack_nframes_t nframes, void *arg )
{
    int i;
    jack_default_audio_sample_t *in, *out;
    for ( i = 0; i < 2; i++ )
    {
        in = jack_port_get_buffer (
input_ports[i], nframes );
        out = jack_port_get_buffer (
output_ports[i], nframes );
```

```
        memcpy ( out, in, nframes * sizeof (
jack_default_audio_sample_t ) );
    }
    return 0;
}

/**
 * JACK calls this shutdown_callback if the
server ever shuts down or
 * decides to disconnect the client.
 */
void
jack_shutdown ( void *arg )
{
    free ( input_ports );
    free ( output_ports );
    exit ( 1 );
}

int
main ( int argc, char *argv[] )
{
    int i;
    const char **ports;
    const char *client_name;
    const char *server_name = NULL;
    jack_options_t options = JackNullOption;
```

```
    jack_status_t status;

    if ( argc >= 2 )        /* client name
specified? */
    {
        client_name = argv[1];
        if ( argc >= 3 )    /* server name
specified? */
        {
            server_name = argv[2];
            options |= JackServerName;
        }
    }
    else              /* use basename of
argv[0] */
    {
        client_name = strrchr ( argv[0], '/' );
        if ( client_name == 0 )
        {
            client_name = argv[0];
        }
        else
        {
            client_name++;
        }
    }
```

```
    /* open a client connection to the JACK
server */

    client = jack_client_open ( client_name,
options, &status, server_name );
    if ( client == NULL )
    {
        fprintf ( stderr, "jack_client_open()
failed, "
                  "status = 0x%2.0x\n", status
);
        if ( status & JackServerFailed )
        {
            fprintf ( stderr, "Unable to
connect to JACK server\n" );
        }
        exit ( 1 );
    }
    if ( status & JackServerStarted )
    {
        fprintf ( stderr, "JACK server
started\n" );
    }
    if ( status & JackNameNotUnique )
    {
        client_name = jack_get_client_name (
client );
```

```
        fprintf ( stderr, "unique name `%s'
assigned\n", client_name );
    }

    /* tell the JACK server to call `process()'
whenever
       there is work to be done.
    */

    jack_set_process_callback ( client,
process, 0 );

    /* tell the JACK server to call
`jack_shutdown()' if
       it ever shuts down, either entirely, or
if it
       just decides to stop calling us.
    */

    jack_on_shutdown ( client, jack_shutdown, 0
);

    /* create two ports pairs*/
    input_ports = ( jack_port_t** ) calloc ( 2,
sizeof ( jack_port_t* ) );
    output_ports = ( jack_port_t** ) calloc (
2, sizeof ( jack_port_t* ) );
```

```
    char port_name[16];
    for ( i = 0; i < 2; i++ )
    {
        sprintf ( port_name, "input_%d", i + 1
);
        input_ports[i] = jack_port_register (
client, port_name, JACK_DEFAULT_AUDIO_TYPE,
JackPortIsInput, 0 );
        sprintf ( port_name, "output_%d", i + 1
);
        output_ports[i] = jack_port_register (
client, port_name, JACK_DEFAULT_AUDIO_TYPE,
JackPortIsOutput, 0 );
        if ( ( input_ports[i] == NULL ) || (
output_ports[i] == NULL ) )
        {
            fprintf ( stderr, "no more JACK
ports available\n" );
            exit ( 1 );
        }
    }

    /* Tell the JACK server that we are ready
to roll.  Our
     * process() callback will start running
now. */
```

```
    if ( jack_activate ( client ) )
    {
        fprintf ( stderr, "cannot activate
client" );
        exit ( 1 );
    }

    /* Connect the ports.  You can't do this
before the client is
     * activated, because we can't make
connections to clients
     * that aren't running.  Note the confusing
(but necessary)
     * orientation of the driver backend ports:
playback ports are
     * "input" to the backend, and capture
ports are "output" from
     * it.
     */

    ports = jack_get_ports ( client, NULL,
NULL, JackPortIsPhysical|JackPortIsOutput );
    if ( ports == NULL )
    {
        fprintf ( stderr, "no physical capture
ports\n" );
```

```
        exit ( 1 );
    }

    for ( i = 0; i < 2; i++ )
        if ( jack_connect ( client, ports[i],
jack_port_name ( input_ports[i] ) ) )
            fprintf ( stderr, "cannot connect
input ports\n" );

    free ( ports );

    ports = jack_get_ports ( client, NULL,
NULL, JackPortIsPhysical|JackPortIsInput );
    if ( ports == NULL )
    {
        fprintf ( stderr, "no physical playback
ports\n" );
        exit ( 1 );
    }

    for ( i = 0; i < 2; i++ )
        if ( jack_connect ( client,
jack_port_name ( output_ports[i] ), ports[i] )
)
            fprintf ( stderr, "cannot connect
input ports\n" );
```

```
    free ( ports );

    /* install a signal handler to properly
quits jack client */
#ifdef WIN32
    signal ( SIGINT, signal_handler );
    signal ( SIGABRT, signal_handler );
    signal ( SIGTERM, signal_handler );
#else
    signal ( SIGQUIT, signal_handler );
    signal ( SIGTERM, signal_handler );
    signal ( SIGHUP, signal_handler );
    signal ( SIGINT, signal_handler );
#endif

    /* keep running until the transport stops
*/

    while (1)
    {
#ifdef WIN32
        Sleep ( 1000 );
#else
        sleep ( 1 );
#endif
    }
```

```
    jack_client_close ( client );
    exit ( 0 );
}
```

Delaying audio

While this book is not about audio effects, we can easily introduce one effect - latency - by just delaying sounds. Now this - and any time consuming actions - are against the spirit (and implementation!) of Jack, so it can only be done in co-operation with the Jack model.

The simplest idea is just to throw in `sleep` commands at the right places. This would assume that calls to the `process` callback happen asynchronously but they don't - they happen synchronously within the Jack processing thread. Activities which cost time aren't allowed. If you try it, you will will end up with lots of xruns at best, seizures of Jack at worst.

In this case the solution is straightforward: keep a buffer in which previous inputs are kept, and read older entries out of this buffer when output is requested. A "big enough" wrap-around array will do this, where old entries are read out and new entries read in.

The following program `delay.c` will copy the left channel in real time, but delay the left channel by 4096 samples:

```
/** @file delay.c
 *
 * @brief This client delays one channel by
4096 framse.
 */

#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <signal.h>
#ifndef WIN32
#include <unistd.h>
#endif
#include <jack/jack.h>

jack_port_t **input_ports;
jack_port_t **output_ports;
jack_client_t *client;

#define SIZE 8192
#define DELAY 4096
jack_default_audio_sample_t buffer[SIZE];
int idx, delay_idx;
```

```
static void signal_handler ( int sig )
{
    jack_client_close ( client );
    fprintf ( stderr, "signal received, exiting
...\n" );
    exit ( 0 );
}

static void copy2out(
jack_default_audio_sample_t *out,
              jack_nframes_t nframes) {
    if (delay_idx + nframes < SIZE) {
    memcpy(out, buffer + delay_idx,
           nframes * sizeof (
jack_default_audio_sample_t ) );
    } else {
    int frames_to_end = SIZE - delay_idx;
    int overflow = delay_idx + nframes - SIZE;
    memcpy(out, buffer + delay_idx,
           frames_to_end * sizeof (
jack_default_audio_sample_t ) );
    memcpy(out, buffer, overflow *
sizeof(jack_default_audio_sample_t));
    }
    delay_idx = (delay_idx + nframes) % SIZE;
}
```

```
static void copy2buffer(
jack_default_audio_sample_t *in,
              jack_nframes_t nframes) {
    if (idx + nframes < SIZE) {
    memcpy(buffer + idx, in,
           nframes * sizeof (
jack_default_audio_sample_t ) );
    } else {
    int frames_to_end = SIZE - idx;
    int overflow = idx + nframes - SIZE;
    memcpy(buffer + idx, in,
           frames_to_end * sizeof (
jack_default_audio_sample_t ) );
    memcpy(buffer, in, overflow *
sizeof(jack_default_audio_sample_t));
    }
    idx = (idx + nframes) % SIZE;
}

/**
 * The process callback for this JACK
application is called in a
 * special realtime thread once for each audio
cycle.
 *
 * This client follows a simple rule: when the
JACK transport is
```

```
 * running, copy the input port to the output.
When it stops, exit.
 */

int
process ( jack_nframes_t nframes, void *arg )
{
    int i;
    jack_default_audio_sample_t *in, *out;

    in = jack_port_get_buffer ( input_ports[0],
nframes );
    out = jack_port_get_buffer (
output_ports[0], nframes );
    memcpy ( out, in, nframes * sizeof (
jack_default_audio_sample_t ) );

    in = jack_port_get_buffer ( input_ports[1],
nframes );
    out = jack_port_get_buffer (
output_ports[1], nframes );
    copy2out(out, nframes);
    copy2buffer(in, nframes);

    return 0;
}
```

```
/**
 * JACK calls this shutdown_callback if the
server ever shuts down or
 * decides to disconnect the client.
 */
void
jack_shutdown ( void *arg )
{
    free ( input_ports );
    free ( output_ports );
    exit ( 1 );
}

int
main ( int argc, char *argv[] )
{
    int i;
    const char **ports;
    const char *client_name;
    const char *server_name = NULL;
    jack_options_t options = JackNullOption;
    jack_status_t status;

    if ( argc >= 2 )        /* client name
specified? */
    {
        client_name = argv[1];
```

```
        if ( argc >= 3 )    /* server name
specified? */
        {
            server_name = argv[2];
            options |= JackServerName;
        }
    }
    else              /* use basename of
argv[0] */
    {
        client_name = strrchr ( argv[0], '/' );
        if ( client_name == 0 )
        {
            client_name = argv[0];
        }
        else
        {
            client_name++;
        }
    }

    /* open a client connection to the JACK
server */

    client = jack_client_open ( client_name,
options, &status, server_name );
    if ( client == NULL )
```

```
    {
        fprintf ( stderr, "jack_client_open()
failed, "
                  "status = 0x%2.0x\n", status
);
        if ( status & JackServerFailed )
        {
            fprintf ( stderr, "Unable to
connect to JACK server\n" );
        }
        exit ( 1 );
    }
    if ( status & JackServerStarted )
    {
        fprintf ( stderr, "JACK server
started\n" );
    }
    if ( status & JackNameNotUnique )
    {
        client_name = jack_get_client_name (
client );
        fprintf ( stderr, "unique name `%s'
assigned\n", client_name );
    }

    /* tell the JACK server to call `process()'
whenever
```

```
       there is work to be done.
    */

    jack_set_process_callback ( client,
process, 0 );

    /* tell the JACK server to call
`jack_shutdown()' if
       it ever shuts down, either entirely, or
if it
       just decides to stop calling us.
    */

    jack_on_shutdown ( client, jack_shutdown, 0
);

    /* create two ports pairs*/
    input_ports = ( jack_port_t** ) calloc ( 2,
sizeof ( jack_port_t* ) );
    output_ports = ( jack_port_t** ) calloc (
2, sizeof ( jack_port_t* ) );

    char port_name[16];
    for ( i = 0; i < 2; i++ )
    {
        sprintf ( port_name, "input_%d", i + 1
);
```

```
        input_ports[i] = jack_port_register (
client, port_name, JACK_DEFAULT_AUDIO_TYPE,
JackPortIsInput, 0 );
        sprintf ( port_name, "output_%d", i + 1
);
        output_ports[i] = jack_port_register (
client, port_name, JACK_DEFAULT_AUDIO_TYPE,
JackPortIsOutput, 0 );
        if ( ( input_ports[i] == NULL ) || (
output_ports[i] == NULL ) )
        {
            fprintf ( stderr, "no more JACK
ports available\n" );
            exit ( 1 );
        }
    }

    bzero(buffer, SIZE * sizeof (
jack_default_audio_sample_t ));
    delay_idx = 0;
    idx = DELAY;

    /* Tell the JACK server that we are ready
to roll.  Our
     * process() callback will start running
now. */
```

```
    if ( jack_activate ( client ) )
    {
        fprintf ( stderr, "cannot activate
client" );
        exit ( 1 );
    }

    /* Connect the ports.  You can't do this
before the client is
     * activated, because we can't make
connections to clients
     * that aren't running.  Note the confusing
(but necessary)
     * orientation of the driver backend ports:
playback ports are
     * "input" to the backend, and capture
ports are "output" from
     * it.
     */

    ports = jack_get_ports ( client, NULL,
NULL, JackPortIsPhysical|JackPortIsOutput );
    if ( ports == NULL )
    {
        fprintf ( stderr, "no physical capture
ports\n" );
        exit ( 1 );
```

```
    }

    for ( i = 0; i < 2; i++ )
        if ( jack_connect ( client, ports[i],
jack_port_name ( input_ports[i] ) ) )
            fprintf ( stderr, "cannot connect
input ports\n" );

    free ( ports );

    ports = jack_get_ports ( client, NULL,
NULL, JackPortIsPhysical|JackPortIsInput );
    if ( ports == NULL )
    {
        fprintf ( stderr, "no physical playback
ports\n" );
        exit ( 1 );
    }

    for ( i = 0; i < 2; i++ )
        if ( jack_connect ( client,
jack_port_name ( output_ports[i] ), ports[i] )
)
            fprintf ( stderr, "cannot connect
input ports\n" );

    free ( ports );
```

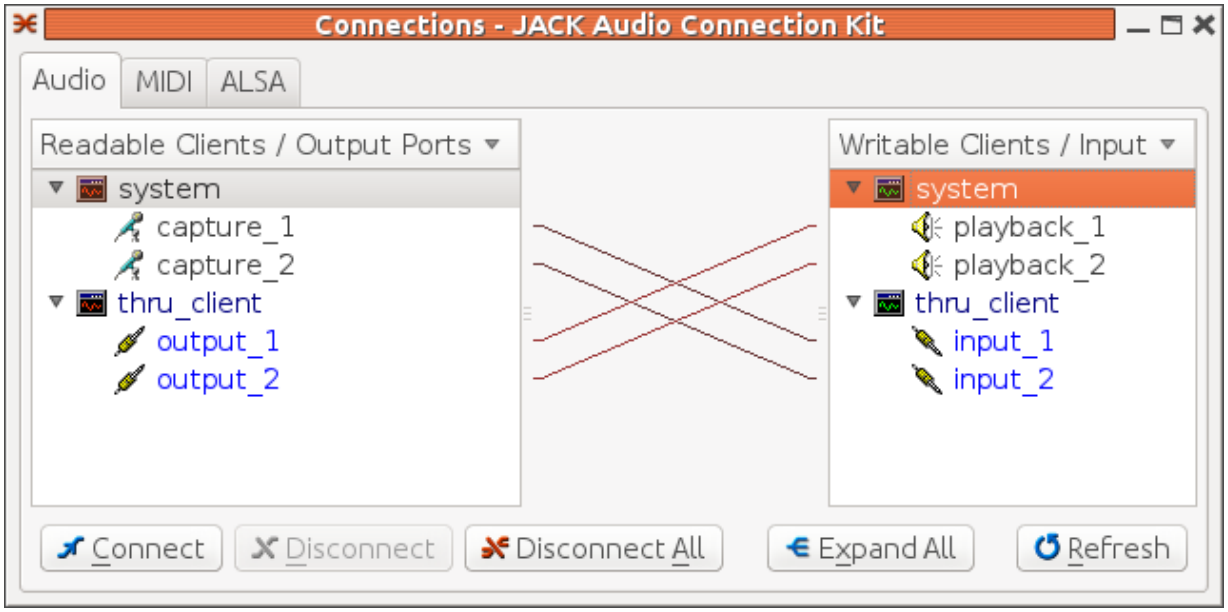
```
    /* install a signal handler to properly
quits jack client */
#ifdef WIN32
    signal ( SIGINT, signal_handler );
    signal ( SIGABRT, signal_handler );
    signal ( SIGTERM, signal_handler );
#else
    signal ( SIGQUIT, signal_handler );
    signal ( SIGTERM, signal_handler );
    signal ( SIGHUP, signal_handler );
    signal ( SIGINT, signal_handler );
#endif

    /* keep running until the transport stops
*/

    while (1)
    {
#ifdef WIN32
        Sleep ( 1000 );
#else
        sleep ( 1 );
#endif
    }

    jack_client_close ( client );
```

```
    exit ( 0 );
}
```

Audacity with Jack

Audacity is Jack aware. You can use it to capture and display Jack streams. But that doesn't mean that for the user it plays in a *nice* way! With a running Jack system, starting Audacity registers it with Jack - but there are no input nor output ports. These only show up when you start a record session with Audacity. It then establishes its own links within Jack.

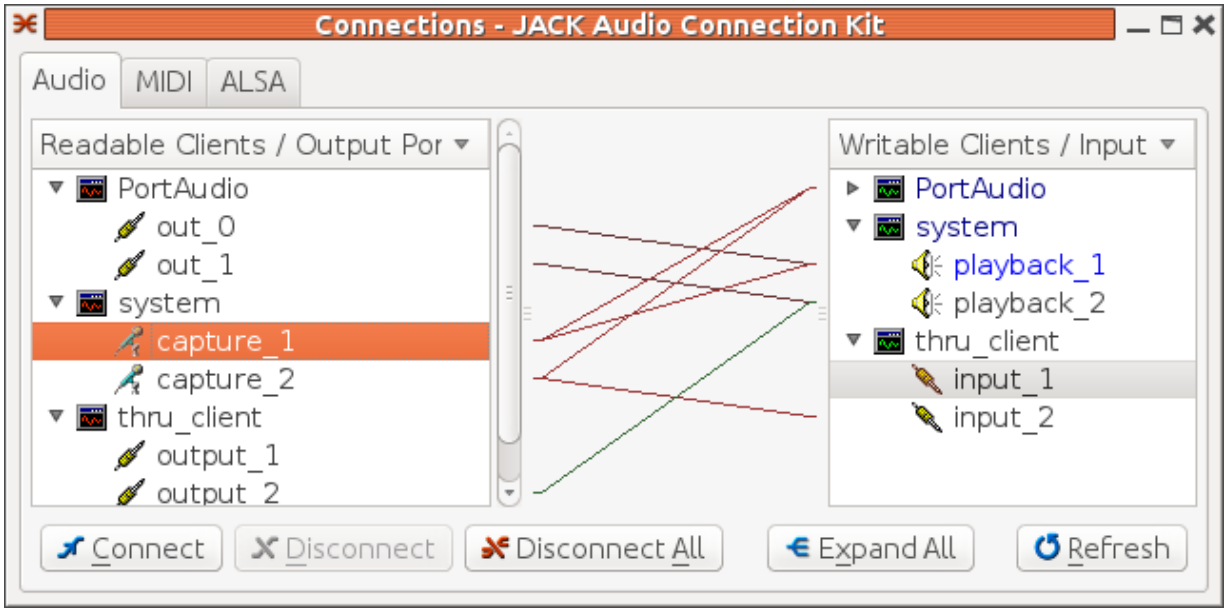
For example, with `thru_client` as the only client within Jack, `qjackctl` shows the connections



In this, the capture devices are connected to the `thru_client` inputs, and the `thru_client` outputs are connected to the playback outputs.

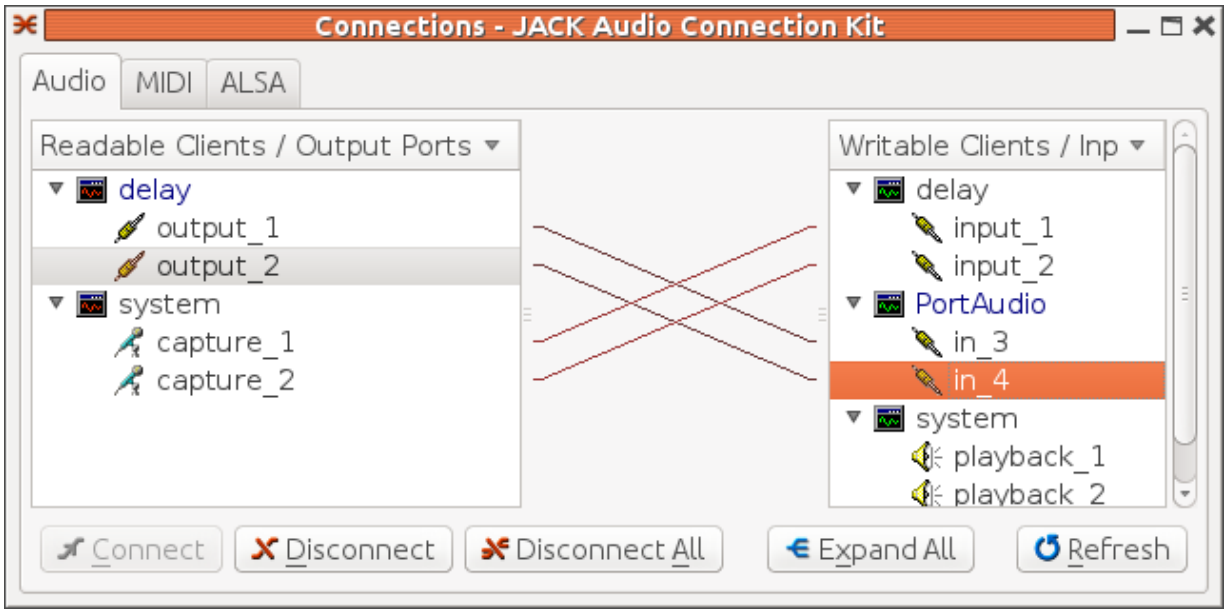
Just starting Audacity but not recording anything makes no changes to this connection graph.

But when Audacity starts recording with `thru_client` already running, `qjackctl` shows the links established as in this figure:



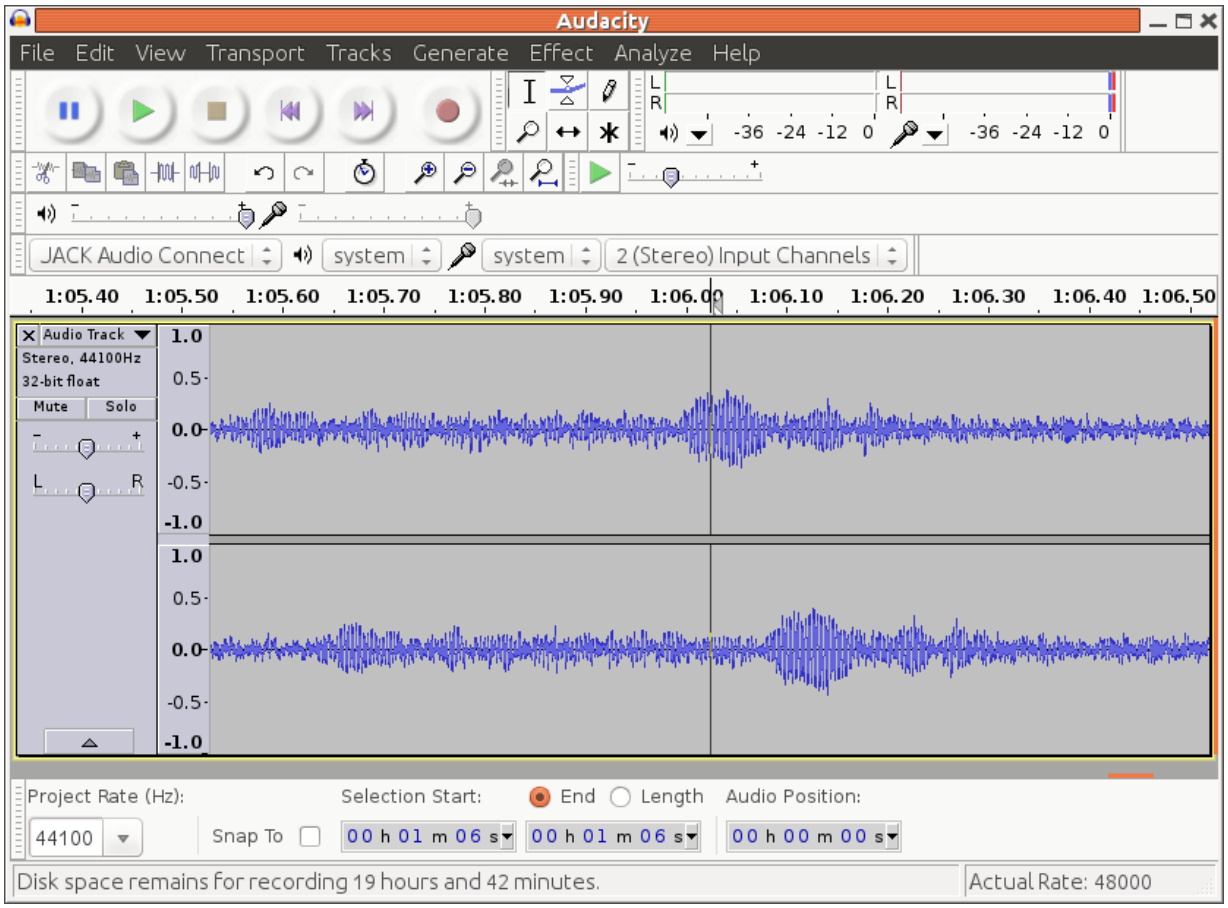
This is a lot messier: Audacity shows as Port Audio devices and the capture devices are linked to the Port Audio inputs while the Port Audio outputs are linked to the playback devices. The existing `thru_client` links are basically discarded. To set up your desired situation, these have to be relinked as needed.

To demonstrate the effects of delaying one channel, start Jack, start `delay` and then start Audacity. Relink the ports according to the following `qjackctl` figure:



That is, capture ports are linked to `delay` input ports, `delay` output ports are linked to Port Audio (Audacity) input ports and Port Audio output ports are linked to playback ports.

The waveforms captured by Audacity clearly show the delay on the left channel compared to the right:


Audacity
File Edit View Transport Tracks Generate Effect Analyze Help
JACK Audio Connect
system
system
2 (Stereo) Input Channels
1:05.40 1:05.50 1:05.60 1:05.70 1:05.80 1:05.90 1:06.00 1:06.10 1:06.20 1:06.30 1:06.40 1:06.50
Audio Track
Stereo, 44100Hz
32-bit float
Mute
Solo
Project Rate (Hz):
44100
Snap To
Selection Start:
End
Length
Audio Position:
00 h 01 m 06 s
00 h 01 m 06 s
00 h 00 m 00 s
Disk space remains for recording 19 hours and 42 minutes.
Actual Rate: 48000

Play a sine wave

The copy example does not show the detail of what is in the buffers: the contents are `jack_default_audio_sample_t` 's. What these are is described in the macro `JACK_DEFAULT_AUDIO_TYPE` with default value "32 bit float mono audio"

To do anything more than simply pass audio through, you need to handle the data in this format. The example program `simple_client.c` fills an array with 32-bit floating point sine curve values. On each call to `process` it copies data from the sine curve array into the output buffers. The increment into the sine curve array is different for left and right channels to give a different note on each channel.

Note that the calculation of the sine curve array is *not* done within the `process` function - that would be too slow and would cause latency.

The program is

```c
/** @file simple_client.c
 *
 * @brief This simple client demonstrates the
basic features of JACK
 * as they would be used by many applications.
 */

#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <signal.h>
#ifndef WIN32
#include <unistd.h>
#endif
#include <jack/jack.h>

jack_port_t *output_port1, *output_port2;
jack_client_t *client;

#ifndef M_PI
#define M_PI  (3.14159265)
#endif

#define TABLE_SIZE   (200)
```

```
typedef struct
{
    float sine[TABLE_SIZE];
    int left_phase;
    int right_phase;
}
paTestData;

static void signal_handler(int sig)
{
    jack_client_close(client);
    fprintf(stderr, "signal received, exiting
...\n");
    exit(0);
}

/**
 * The process callback for this JACK
application is called in a
 * special realtime thread once for each audio
cycle.
 *
 * This client follows a simple rule: when the
JACK transport is
 * running, copy the input port to the output.
When it stops, exit.
 */
```

```
int
process (jack_nframes_t nframes, void *arg)
{
    jack_default_audio_sample_t *out1, *out2;
    paTestData *data = (paTestData*)arg;
    int i;

    out1 =
(jack_default_audio_sample_t*)jack_port_get_buffe
 (output_port1, nframes);
    out2 =
(jack_default_audio_sample_t*)jack_port_get_buffe
 (output_port2, nframes);

    for( i=0; i<nframes; i++ )
    {
        out1[i] = data->sine[data->left_phase];
/* left */
        out2[i] = data->sine[data-
>right_phase];  /* right */
        data->left_phase += 1;
        if( data->left_phase >= TABLE_SIZE )
data->left_phase -= TABLE_SIZE;
        data->right_phase += 3; /* higher pitch
so we can distinguish left and right. */
        if( data->right_phase >= TABLE_SIZE )
```

```
data->right_phase -= TABLE_SIZE;
    }

    return 0;
}

/**
 * JACK calls this shutdown_callback if the
server ever shuts down or
 * decides to disconnect the client.
 */
void
jack_shutdown (void *arg)
{
    exit (1);
}

int
main (int argc, char *argv[])
{
    const char **ports;
    const char *client_name;
    const char *server_name = NULL;
    jack_options_t options = JackNullOption;
    jack_status_t status;
    paTestData data;
    int i;
```

```
    if (argc >= 2) {        /* client name
specified? */
        client_name = argv[1];
        if (argc >= 3) {    /* server name
specified? */
            server_name = argv[2];
            int my_option = JackNullOption |
JackServerName;
            options =
(jack_options_t)my_option;
        }
    } else {            /* use basename of
argv[0] */
        client_name = strrchr(argv[0], '/');
        if (client_name == 0) {
            client_name = argv[0];
        } else {
            client_name++;
        }
    }

    for( i=0; i<TABLE_SIZE; i++ )
    {
        data.sine[i] = 0.2 * (float) sin(
((double)i/(double)TABLE_SIZE) * M_PI * 2. );
    }
```

```
    data.left_phase = data.right_phase = 0;


    /* open a client connection to the JACK
server */

    client = jack_client_open (client_name,
options, &status, server_name);
    if (client == NULL) {
        fprintf (stderr, "jack_client_open()
failed, "
             "status = 0x%2.0x\n", status);
        if (status & JackServerFailed) {
            fprintf (stderr, "Unable to connect
to JACK server\n");
        }
        exit (1);
    }
    if (status & JackServerStarted) {
        fprintf (stderr, "JACK server
started\n");
    }
    if (status & JackNameNotUnique) {
        client_name =
jack_get_client_name(client);
        fprintf (stderr, "unique name `%s'
assigned\n", client_name);
```

```
    }

    /* tell the JACK server to call `process()'
whenever
       there is work to be done.
    */

    jack_set_process_callback (client, process,
&data);

    /* tell the JACK server to call
`jack_shutdown()' if
       it ever shuts down, either entirely, or
if it
       just decides to stop calling us.
    */

    jack_on_shutdown (client, jack_shutdown,
0);

    /* create two ports */

    output_port1 = jack_port_register (client,
"output1",
                         JACK_DEFAULT_AUDIO_TYPE,
                         JackPortIsOutput, 0);
```

```
    output_port2 = jack_port_register (client,
"output2",
                        JACK_DEFAULT_AUDIO_TYPE,
                        JackPortIsOutput, 0);

    if ((output_port1 == NULL) || (output_port2
== NULL)) {
        fprintf(stderr, "no more JACK ports
available\n");
        exit (1);
    }

    /* Tell the JACK server that we are ready
to roll.  Our
     * process() callback will start running
now. */

    if (jack_activate (client)) {
        fprintf (stderr, "cannot activate
client");
        exit (1);
    }

    /* Connect the ports.  You can't do this
before the client is
     * activated, because we can't make
connections to clients
```

```
     * that aren't running.  Note the confusing
(but necessary)
     * orientation of the driver backend ports:
playback ports are
     * "input" to the backend, and capture
ports are "output" from
     * it.
     */

    ports = jack_get_ports (client, NULL, NULL,

JackPortIsPhysical|JackPortIsInput);
    if (ports == NULL) {
        fprintf(stderr, "no physical playback
ports\n");
        exit (1);
    }

    if (jack_connect (client, jack_port_name
(output_port1), ports[0])) {
        fprintf (stderr, "cannot connect output
ports\n");
    }

    if (jack_connect (client, jack_port_name
(output_port2), ports[1])) {
        fprintf (stderr, "cannot connect output
```

```
ports\n");
    }

    free (ports);

    /* install a signal handler to properly
quits jack client */
#ifdef WIN32
    signal(SIGINT, signal_handler);
    signal(SIGABRT, signal_handler);
    signal(SIGTERM, signal_handler);
#else
    signal(SIGQUIT, signal_handler);
    signal(SIGTERM, signal_handler);
    signal(SIGHUP, signal_handler);
    signal(SIGINT, signal_handler);
#endif

    /* keep running until the Ctrl+C */

    while (1) {
    #ifdef WIN32
        Sleep(1000);
    #else
        sleep (1);
    #endif
    }
```

```
    jack_client_close (client);
    exit (0);
}
```

Saving input to disk

Disk I/O cannot be performed within the Jack processing loop: it is just too slow. To save input to file requires use of a separate thread to manage disk I/O and pass control between the Jack and disk threads.

The program `capture_client.c` from the examples does this:

```
/*
    Copyright (C) 2001 Paul Davis
    Copyright (C) 2003 Jack O'Quin

    This program is free software; you can
redistribute it and/or modify
    it under the terms of the GNU General
Public License as published by
    the Free Software Foundation; either
version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope
that it will be useful,
    but WITHOUT ANY WARRANTY; without even the
implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.  See the
    GNU General Public License for more
details.

    You should have received a copy of the GNU
General Public License
    along with this program; if not, write to
the Free Software
    Foundation, Inc., 675 Mass Ave, Cambridge,
```

```
MA 02139, USA.

    * 2002/08/23 - modify for libsndfile 1.0.0
<andy@alsaplayer.org>
    * 2003/05/26 - use ringbuffers - joq
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sndfile.h>
#include <pthread.h>
#include <signal.h>
#include <getopt.h>
#include <jack/jack.h>
#include <jack/ringbuffer.h>

typedef struct _thread_info {
    pthread_t thread_id;
    SNDFILE *sf;
    jack_nframes_t duration;
    jack_nframes_t rb_size;
    jack_client_t *client;
    unsigned int channels;
    int bitdepth;
```

```
    char *path;
    volatile int can_capture;
    volatile int can_process;
    volatile int status;
} jack_thread_info_t;

/* JACK data */
unsigned int nports;
jack_port_t **ports;
jack_default_audio_sample_t **in;
jack_nframes_t nframes;
const size_t sample_size =
sizeof(jack_default_audio_sample_t);

/* Synchronization between process thread and
disk thread. */
#define DEFAULT_RB_SIZE 16384        /*
ringbuffer size in frames */
jack_ringbuffer_t *rb;
pthread_mutex_t disk_thread_lock =
PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t  data_ready =
PTHREAD_COND_INITIALIZER;
long overruns = 0;
jack_client_t *client;

static void signal_handler(int sig)
```

```
{
    jack_client_close(client);
    fprintf(stderr, "signal received, exiting
...\n");
    exit(0);
}

static void *
disk_thread (void *arg)
{
    jack_thread_info_t *info =
(jack_thread_info_t *) arg;
    static jack_nframes_t total_captured = 0;
    jack_nframes_t samples_per_frame = info-
>channels;
    size_t bytes_per_frame = samples_per_frame
* sample_size;
    void *framebuf = malloc (bytes_per_frame);

    pthread_setcanceltype
(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
    pthread_mutex_lock (&disk_thread_lock);

    info->status = 0;

    while (1) {
```

```
        /* Write the data one frame at a time.
This is
         * inefficient, but makes things
simpler. */
        while (info->can_capture &&
               (jack_ringbuffer_read_space (rb)
>= bytes_per_frame)) {

            jack_ringbuffer_read (rb, framebuf,
bytes_per_frame);

            if (sf_writef_float (info->sf,
framebuf, 1) != 1) {
                char errstr[256];
                sf_error_str (0, errstr, sizeof
(errstr) - 1);
                fprintf (stderr,
                     "cannot write sndfile
(%s)\n",
                     errstr);
                info->status = EIO; /* write
failed */
                goto done;
            }

            if (++total_captured >= info-
>duration) {
```

```
                printf ("disk thread
finished\n");
                goto done;
            }
        }

        /* wait until process() signals more
data */
        pthread_cond_wait (&data_ready,
&disk_thread_lock);
    }

 done:
    pthread_mutex_unlock (&disk_thread_lock);
    free (framebuf);
    return 0;
}

static int
process (jack_nframes_t nframes, void *arg)
{
    int chn;
    size_t i;
    jack_thread_info_t *info =
(jack_thread_info_t *) arg;

    /* Do nothing until we're ready to begin.
```

```
*/
    if ((!info->can_process) || (!info-
>can_capture))
        return 0;

    for (chn = 0; chn < nports; chn++)
        in[chn] = jack_port_get_buffer
(ports[chn], nframes);

    /* Sndfile requires interleaved data.  It
is simpler here to
     * just queue interleaved samples to a
single ringbuffer. */
    for (i = 0; i < nframes; i++) {
        for (chn = 0; chn < nports; chn++) {
            if (jack_ringbuffer_write (rb,
(void *) (in[chn]+i),
                                sample_size)
                < sample_size)
                overruns++;
        }
    }

    /* Tell the disk thread there is work to
do.  If it is already
     * running, the lock will not be available.
We can't wait
```

```
     * here in the process() thread, but we
don't need to signal
     * in that case, because the disk thread
will read all the
     * data queued before waiting again. */
    if (pthread_mutex_trylock
(&disk_thread_lock) == 0) {
        pthread_cond_signal (&data_ready);
        pthread_mutex_unlock
(&disk_thread_lock);
    }

    return 0;
}

static void
jack_shutdown (void *arg)
{
    fprintf(stderr, "JACK shut down, exiting
...\n");
    exit(1);
}

static void
setup_disk_thread (jack_thread_info_t *info)
{
    SF_INFO sf_info;
```

```
    int short_mask;

    sf_info.samplerate = jack_get_sample_rate
(info->client);
    sf_info.channels = info->channels;

    switch (info->bitdepth) {
        case 8: short_mask = SF_FORMAT_PCM_U8;
              break;
        case 16: short_mask = SF_FORMAT_PCM_16;
             break;
        case 24: short_mask = SF_FORMAT_PCM_24;
             break;
        case 32: short_mask = SF_FORMAT_PCM_32;
             break;
        default: short_mask = SF_FORMAT_PCM_16;
             break;
    }
    sf_info.format = SF_FORMAT_WAV|short_mask;

    if ((info->sf = sf_open (info->path,
SFM_WRITE, &sf_info)) == NULL) {
        char errstr[256];
        sf_error_str (0, errstr, sizeof
(errstr) - 1);
        fprintf (stderr, "cannot open sndfile
\"%s\" for output (%s)\n", info->path, errstr);
```

```
        jack_client_close (info->client);
        exit (1);
    }

    info->duration *= sf_info.samplerate;
    info->can_capture = 0;

    pthread_create (&info->thread_id, NULL,
disk_thread, info);
}

static void
run_disk_thread (jack_thread_info_t *info)
{
    info->can_capture = 1;
    pthread_join (info->thread_id, NULL);
    sf_close (info->sf);
    if (overruns > 0) {
        fprintf (stderr,
             "jackrec failed with %ld
overruns.\n", overruns);
        fprintf (stderr, " try a bigger buffer
than -B %"
             PRIu32 ".\n", info->rb_size);
        info->status = EPIPE;
    }
}
```

```
static void
setup_ports (int sources, char *source_names[],
jack_thread_info_t *info)
{
    unsigned int i;
    size_t in_size;

    /* Allocate data structures that depend on
the number of ports. */
    nports = sources;
    ports = (jack_port_t **) malloc (sizeof
(jack_port_t *) * nports);
    in_size =  nports * sizeof
(jack_default_audio_sample_t *);
    in = (jack_default_audio_sample_t **)
malloc (in_size);
    rb = jack_ringbuffer_create (nports *
sample_size * info->rb_size);

    /* When JACK is running realtime,
jack_activate() will have
     * called mlockall() to lock our pages into
memory.  But, we
     * still need to touch any newly allocated
pages before
     * process() starts using them.  Otherwise,
```

```
a page fault could
     * create a delay that would force JACK to
shut us down. */
    memset(in, 0, in_size);
    memset(rb->buf, 0, rb->size);

    for (i = 0; i < nports; i++) {
        char name[64];

        sprintf (name, "input%d", i+1);

        if ((ports[i] = jack_port_register
(info->client, name, JACK_DEFAULT_AUDIO_TYPE,
JackPortIsInput, 0)) == 0) {
            fprintf (stderr, "cannot register
input port \"%s\"!\n", name);
            jack_client_close (info->client);
            exit (1);
        }
    }

    for (i = 0; i < nports; i++) {
        if (jack_connect (info->client,
source_names[i], jack_port_name (ports[i]))) {
            fprintf (stderr, "cannot connect
input port %s to %s\n", jack_port_name
(ports[i]), source_names[i]);
```

```
            jack_client_close (info->client);
            exit (1);
        }
    }

    info->can_process = 1;        /* process()
can start, now */
}

int
main (int argc, char *argv[])
{
    jack_thread_info_t thread_info;
    int c;
    int longopt_index = 0;
    extern int optind, opterr;
    int show_usage = 0;
    char *optstring = "d:f:b:B:h";
    struct option long_options[] = {
        { "help", 0, 0, 'h' },
        { "duration", 1, 0, 'd' },
        { "file", 1, 0, 'f' },
        { "bitdepth", 1, 0, 'b' },
        { "bufsize", 1, 0, 'B' },
        { 0, 0, 0, 0 }
    };
```

```
    memset (&thread_info, 0, sizeof
(thread_info));
    thread_info.rb_size = DEFAULT_RB_SIZE;
    opterr = 0;

    while ((c = getopt_long (argc, argv,
optstring, long_options, &longopt_index)) !=
-1) {
        switch (c) {
        case 1:
            /* getopt signals end of '-'
options */
            break;

        case 'h':
            show_usage++;
            break;
        case 'd':
            thread_info.duration = atoi
(optarg);
            break;
        case 'f':
            thread_info.path = optarg;
            break;
        case 'b':
            thread_info.bitdepth = atoi
(optarg);
```

```
            break;
        case 'B':
            thread_info.rb_size = atoi
(optarg);
            break;
        default:
            fprintf (stderr, "error\n");
            show_usage++;
            break;
        }
    }

    if (show_usage || thread_info.path == NULL
|| optind == argc) {
        fprintf (stderr, "usage: jackrec -f
filename [ -d second ] [ -b bitdepth ] [ -B
bufsize ] port1 [ port2 ... ]\n");
        exit (1);
    }

    if ((client = jack_client_open ("jackrec",
JackNullOption, NULL)) == 0) {
        fprintf (stderr, "JACK server not
running?\n");
        exit (1);
    }
```

```
    thread_info.client = client;
    thread_info.channels = argc - optind;
    thread_info.can_process = 0;

    setup_disk_thread (&thread_info);

    jack_set_process_callback (client, process,
&thread_info);
    jack_on_shutdown (client, jack_shutdown,
&thread_info);

    if (jack_activate (client)) {
        fprintf (stderr, "cannot activate
client");
    }

    setup_ports (argc - optind, &argv[optind],
&thread_info);

     /* install a signal handler to properly
quits jack client */
    signal(SIGQUIT, signal_handler);
    signal(SIGTERM, signal_handler);
    signal(SIGHUP, signal_handler);
    signal(SIGINT, signal_handler);

    run_disk_thread (&thread_info);
```

```
    jack_client_close (client);

    jack_ringbuffer_free (rb);

    exit (0);
}
```

Interacting with ALSA devices

Jack will eventually get its input from, and send its output to, devices. Currently, they are most likely to be ALSA devices. Consequently there must be a bridge between Jack processing and ALSA input and output. This will involve all the complexity of ALSA programming.

Fortunately there are Jack clients that do this. The Jack framework will talk to these as specified on starting the Jack server

```
jackd -dalsa
```

If you do want to worry, the examples directory contains ALSA examples. The program `alsa_in.c.ok` brings an ALSA input device into the Jack world

```
/** @file simple_client.c
 *
 * @brief This simple client demonstrates the
basic features of JACK
 * as they would be used by many applications.
 */

#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>

#include <math.h>

#include <jack/jack.h>
#include <jack/jslist.h>
#include "memops.h"

#include "alsa/asoundlib.h"

#include <samplerate.h>

// Here are the lists of the jack ports...
```

```
JSList       *capture_ports = NULL;
JSList       *capture_srcs = NULL;
JSList       *playback_ports = NULL;
JSList       *playback_srcs = NULL;
jack_client_t *client;

snd_pcm_t *alsa_handle;

int jack_sample_rate;
int jack_buffer_size;

int quit = 0;
double resample_mean = 1.0;
double static_resample_factor = 1.0;
double resample_lower_limit = 0.25;
double resample_upper_limit = 4.0;

double *offset_array;
double *window_array;
int offset_differential_index = 0;

double offset_integral = 0;

// ---------------------------------------------
---------- commandline parameters

int sample_rate = 0;                /* stream
```

```
rate */
int num_channels = 2;                /* count
of channels */
int period_size = 1024;
int num_periods = 2;

int target_delay = 0;        /* the delay which
the program should try to approach. */
int max_diff = 0;        /* the diff value,
when a hard readpointer skip should occur */
int catch_factor = 100000;
int catch_factor2 = 10000;
double pclamp = 15.0;
double controlquant = 10000.0;
int smooth_size = 256;
int good_window=0;
int verbose = 0;
int instrument = 0;
int samplerate_quality = 2;

// Debug stuff:

volatile float output_resampling_factor = 1.0;
volatile int output_new_delay = 0;
volatile float output_offset = 0.0;
volatile float output_integral = 0.0;
volatile float output_diff = 0.0;
```

```
snd_pcm_uframes_t real_buffer_size;
snd_pcm_uframes_t real_period_size;

// buffers

char *tmpbuf;
char *outbuf;
float *resampbuf;

// format selection, and corresponding
functions from memops in a nice set of structs.

typedef struct alsa_format {
    snd_pcm_format_t format_id;
    size_t sample_size;
    void (*jack_to_soundcard) (char *dst,
jack_default_audio_sample_t *src, unsigned long
nsamples, unsigned long dst_skip,
dither_state_t *state);
    void (*soundcard_to_jack)
(jack_default_audio_sample_t *dst, char *src,
unsigned long nsamples, unsigned long
src_skip);
    const char *name;
} alsa_format_t;
```

```
alsa_format_t formats[] = {
    { SND_PCM_FORMAT_FLOAT_LE, 4,
sample_move_dS_floatLE,
sample_move_floatLE_sSs, "float" },
    { SND_PCM_FORMAT_S32, 4,
sample_move_d32u24_sS, sample_move_dS_s32u24,
"32bit" },
    { SND_PCM_FORMAT_S24_3LE, 3,
sample_move_d24_sS, sample_move_dS_s24, "24bit
- real" },
    { SND_PCM_FORMAT_S24, 4,
sample_move_d24_sS, sample_move_dS_s24, "24bit"
},
    { SND_PCM_FORMAT_S16, 2,
sample_move_d16_sS, sample_move_dS_s16, "16bit"
}
};
#define NUMFORMATS
(sizeof(formats)/sizeof(formats[0]))
int format=0;

// Alsa stuff... i dont want to touch this
bullshit in the next years.... please...

static int xrun_recovery(snd_pcm_t *handle, int
err) {
//    printf( "xrun !!!.... %d\n", err );
```

```
    if (err == -EPIPE) {    /* under-run */
        err = snd_pcm_prepare(handle);
        if (err < 0)
            printf("Can't recovery from
underrun, prepare failed: %s\n",
snd_strerror(err));
        return 0;
    } else if (err == -EAGAIN) {
        while ((err = snd_pcm_resume(handle))
== -EAGAIN)
            usleep(100);    /* wait until the
suspend flag is released */
        if (err < 0) {
            err = snd_pcm_prepare(handle);
            if (err < 0)
                printf("Can't recovery from
suspend, prepare failed: %s\n",
snd_strerror(err));
        }
        return 0;
    }
    return err;
}

static int set_hwformat( snd_pcm_t *handle,
snd_pcm_hw_params_t *params )
{
```

```
    int i;
    int err;

    for( i=0; i<NUMFORMATS; i++ ) {
        /* set the sample format */
        err =
snd_pcm_hw_params_set_format(handle, params,
formats[i].format_id);
        if (err == 0) {
            format = i;
            return 0;
        }
    }

    return err;
}

static int set_hwparams(snd_pcm_t *handle,
snd_pcm_hw_params_t *params, snd_pcm_access_t
access, int rate, int channels, int period, int
nperiods ) {
    int err, dir=0;
    unsigned int buffer_time;
    unsigned int period_time;
    unsigned int rrate;
    unsigned int rchannels;
```

```
    /* choose all parameters */
    err = snd_pcm_hw_params_any(handle,
params);
    if (err < 0) {
        printf("Broken configuration for
playback: no configurations available: %s\n",
snd_strerror(err));
        return err;
    }
    /* set the interleaved read/write format */
    err = snd_pcm_hw_params_set_access(handle,
params, access);
    if (err < 0) {
        printf("Access type not available for
playback: %s\n", snd_strerror(err));
        return err;
    }

    /* set the sample format */
    err = set_hwformat(handle, params);
    if (err < 0) {
        printf("Sample format not available for
playback: %s\n", snd_strerror(err));
        return err;
    }
    /* set the count of channels */
    rchannels = channels;
```

```
    err =
snd_pcm_hw_params_set_channels_near(handle,
params, &rchannels);
    if (err < 0) {
        printf("Channels count (%i) not
available for record: %s\n", channels,
snd_strerror(err));
        return err;
    }
    if (rchannels != channels) {
        printf("WARNING: chennel count does not
match (requested %d got %d)\n", channels,
rchannels);
        num_channels = rchannels;
    }
    /* set the stream rate */
    rrate = rate;
    err =
snd_pcm_hw_params_set_rate_near(handle, params,
&rrate, 0);
    if (err < 0) {
        printf("Rate %iHz not available for
playback: %s\n", rate, snd_strerror(err));
        return err;
    }
    if (rrate != rate) {
        printf("WARNING: Rate doesn't match
```

```
(requested %iHz, get %iHz)\n", rate, rrate);
        sample_rate = rrate;
    }
    /* set the buffer time */

    buffer_time = 1000000*
(uint64_t)period*nperiods/rate;
    err =
snd_pcm_hw_params_set_buffer_time_near(handle,
params, &buffer_time, &dir);
    if (err < 0) {
        printf("Unable to set buffer time %i
for playback: %s\n",
1000000*period*nperiods/rate,
snd_strerror(err));
        return err;
    }
    err = snd_pcm_hw_params_get_buffer_size(
params, &real_buffer_size );
    if (err < 0) {
        printf("Unable to get buffer size back:
%s\n", snd_strerror(err));
        return err;
    }
    if( real_buffer_size != nperiods * period )
{
        printf( "WARNING: buffer size does not
```

```
match: (requested %d, got %d)\n", nperiods *
period, (int) real_buffer_size );
    }
    /* set the period time */
    period_time = 1000000*
(uint64_t)period/rate;
    err =
snd_pcm_hw_params_set_period_time_near(handle,
params, &period_time, &dir);
    if (err < 0) {
        printf("Unable to set period time %i
for playback: %s\n", 1000000*period/rate,
snd_strerror(err));
        return err;
    }
    err =
snd_pcm_hw_params_get_period_size(params,
&real_period_size, NULL );
    if (err < 0) {
        printf("Unable to get period size back:
%s\n", snd_strerror(err));
        return err;
    }
    if( real_period_size != period ) {
        printf( "WARNING: period size does not
match: (requested %i, got %i)\n", period,
(int)real_period_size );
```

```
    }
    /* write the parameters to device */
    err = snd_pcm_hw_params(handle, params);
    if (err < 0) {
        printf("Unable to set hw params for
playback: %s\n", snd_strerror(err));
        return err;
    }
    return 0;
}

static int set_swparams(snd_pcm_t *handle,
snd_pcm_sw_params_t *swparams, int period) {
    int err;

    /* get the current swparams */
    err = snd_pcm_sw_params_current(handle,
swparams);
    if (err < 0) {
        printf("Unable to determine current
swparams for capture: %s\n",
snd_strerror(err));
        return err;
    }
    /* start the transfer when the buffer is
full */
    err =
```

```
snd_pcm_sw_params_set_start_threshold(handle,
swparams, period );
    if (err < 0) {
        printf("Unable to set start threshold
mode for capture: %s\n", snd_strerror(err));
        return err;
    }
    err =
snd_pcm_sw_params_set_stop_threshold(handle,
swparams, -1 );
    if (err < 0) {
        printf("Unable to set start threshold
mode for capture: %s\n", snd_strerror(err));
        return err;
    }
    /* allow the transfer when at least
period_size samples can be processed */
    err =
snd_pcm_sw_params_set_avail_min(handle,
swparams, 2*period );
    if (err < 0) {
        printf("Unable to set avail min for
capture: %s\n", snd_strerror(err));
        return err;
    }
    /* align all transfers to 1 sample */
    err =
```

```
snd_pcm_sw_params_set_xfer_align(handle,
swparams, 1);
    if (err < 0) {
        printf("Unable to set transfer align
for capture: %s\n", snd_strerror(err));
        return err;
    }
    /* write the parameters to the playback
device */
    err = snd_pcm_sw_params(handle, swparams);
    if (err < 0) {
        printf("Unable to set sw params for
capture: %s\n", snd_strerror(err));
        return err;
    }
    return 0;
}

// ok... i only need this function to
communicate with the alsa bloat api...

static snd_pcm_t *open_audiofd( char
*device_name, int capture, int rate, int
channels, int period, int nperiods ) {
  int err;
  snd_pcm_t *handle;
  snd_pcm_hw_params_t *hwparams;
```

```
  snd_pcm_sw_params_t *swparams;

  snd_pcm_hw_params_alloca(&hwparams);
  snd_pcm_sw_params_alloca(&swparams);

  if ((err = snd_pcm_open(&(handle),
device_name, capture ? SND_PCM_STREAM_CAPTURE :
SND_PCM_STREAM_PLAYBACK, SND_PCM_NONBLOCK )) <
0) {
      printf("Capture open error: %s\n",
snd_strerror(err));
      return NULL;
  }

  if ((err = set_hwparams(handle,
hwparams,SND_PCM_ACCESS_RW_INTERLEAVED, rate,
channels, period, nperiods )) < 0) {
      printf("Setting of hwparams failed:
%s\n", snd_strerror(err));
      return NULL;
  }
  if ((err = set_swparams(handle, swparams,
period)) < 0) {
      printf("Setting of swparams failed:
%s\n", snd_strerror(err));
      return NULL;
  }
```

```
  snd_pcm_start( handle );
  snd_pcm_wait( handle, 200 );

  return handle;
}

double hann( double x )
{
    return 0.5 * (1.0 - cos( 2*M_PI * x ) );
}

/**
 * The process callback for this JACK
application.
 * It is called by JACK at the appropriate
times.
 */
int process (jack_nframes_t nframes, void *arg)
{

    int rlen;
    int err;
    snd_pcm_sframes_t delay = target_delay;
    int put_back_samples=0;
    int i;
```

```
    delay = snd_pcm_avail( alsa_handle );

    delay -= jack_frames_since_cycle_start(
client );
    // Do it the hard way.
    // this is for compensating xruns etc...

    if( delay > (target_delay+max_diff) ) {

    output_new_delay = (int) delay;

    while ((delay-target_delay) > 0) {
        snd_pcm_uframes_t to_read = ((delay-
target_delay) > 512) ? 512 : (delay-
target_delay);
        snd_pcm_readi( alsa_handle, tmpbuf,
to_read );
        delay -= to_read;
    }

    delay = target_delay;

    // Set the resample_rate... we need to
adjust the offset integral, to do this.
    // first look at the PI controller, this
code is just a special case, which should never
execute once
```

```
    // everything is swung in.
    offset_integral = - (resample_mean -
static_resample_factor) * catch_factor *
catch_factor2;
    // Also clear the array. we are beginning a
new control cycle.
    for( i=0; i<smooth_size; i++ )
        offset_array[i] = 0.0;
    }
    if( delay < (target_delay-max_diff) ) {
    snd_pcm_rewind( alsa_handle, target_delay -
delay );
    output_new_delay = (int) delay;
    delay = target_delay;

    // Set the resample_rate... we need to
adjust the offset integral, to do this.
    offset_integral = - (resample_mean -
static_resample_factor) * catch_factor *
catch_factor2;
    // Also clear the array. we are beginning a
new control cycle.
    for( i=0; i<smooth_size; i++ )
        offset_array[i] = 0.0;
    }
    /* ok... now we should have target_delay +-
max_diff on the alsa side.
```

```
     *
     * calculate the number of frames, we want
to get.
     */

    double offset = delay - target_delay;

    // Save offset.
    offset_array[(offset_differential_index++)%
smooth_size ] = offset;

    // Build the mean of the windowed offset
array
    // basically fir lowpassing.
    double smooth_offset = 0.0;
    for( i=0; i<smooth_size; i++ )
        smooth_offset +=
            offset_array[ (i +
offset_differential_index-1) % smooth_size] *
window_array[i];
    smooth_offset /= (double) smooth_size;

    // this is the integral of the
smoothed_offset
    offset_integral += smooth_offset;

    // Clamp offset.
```

```
    // the smooth offset still contains
unwanted noise
    // which would go straigth onto the
resample coeff.
    // it only used in the P component and the
I component is used for the fine tuning
anyways.
    if( fabs( smooth_offset ) < pclamp )
        smooth_offset = 0.0;

    // ok. now this is the PI controller.
    // u(t) = K * ( e(t) + 1/T \int e(t') dt' )
    // K = 1/catch_factor and T = catch_factor2
    double current_resample_factor =
static_resample_factor - smooth_offset /
(double) catch_factor - offset_integral /
(double) catch_factor / (double)catch_factor2;

    // now quantize this value around
resample_mean, so that the noise which is in
the integral component doesnt hurt.
    current_resample_factor = floor(
(current_resample_factor - resample_mean) *
controlquant + 0.5 ) / controlquant +
resample_mean;

    // Output "instrumentatio" gonna change
```

```
that to real instrumentation in a few.
    output_resampling_factor = (float)
current_resample_factor;
    output_diff = (float) smooth_offset;
    output_integral = (float) offset_integral;
    output_offset = (float) offset;

    // Clamp a bit.
    if( current_resample_factor <
resample_lower_limit ) current_resample_factor
= resample_lower_limit;
    if( current_resample_factor >
resample_upper_limit ) current_resample_factor
= resample_upper_limit;

    // Now Calculate how many samples we need.
    rlen = ceil( ((double)nframes) /
current_resample_factor )+2;
    assert( rlen > 2 );

    // Calculate resample_mean so we can init
ourselves to saner values.
    resample_mean = 0.9999 * resample_mean +
0.0001 * current_resample_factor;

    // get the data...
again:
```

```
    err = snd_pcm_readi(alsa_handle, outbuf,
rlen);
    if( err < 0 ) {
    printf( "err = %d\n", err );
    if (xrun_recovery(alsa_handle, err) < 0) {
        //printf("Write error: %s\n",
snd_strerror(err));
        //exit(EXIT_FAILURE);
    }
    goto again;
    }
    if( err != rlen ) {
    //printf( "read = %d\n", rlen );
    }

    /*
     * render jack ports to the outbuf...
     */

    int chn = 0;
    JSList *node = capture_ports;
    JSList *src_node = capture_srcs;
    SRC_DATA src;

    while ( node != NULL)
    {
    jack_port_t *port = (jack_port_t *) node-
```

```
>data;
    float *buf = jack_port_get_buffer (port,
nframes);

    SRC_STATE *src_state = src_node->data;

    formats[format].soundcard_to_jack(
resampbuf, outbuf + format[formats].sample_size
* chn, rlen,
num_channels*format[formats].sample_size );

    src.data_in = resampbuf;
    src.input_frames = rlen;

    src.data_out = buf;
    src.output_frames = nframes;
    src.end_of_input = 0;

    src.src_ratio = current_resample_factor;

    src_process( src_state, &src );

    put_back_samples = rlen-
src.input_frames_used;

    src_node = jack_slist_next (src_node);
    node = jack_slist_next (node);
```

```
    chn++;
    }

    // Put back the samples libsamplerate did
not consume.
    //printf( "putback = %d\n",
put_back_samples );
    snd_pcm_rewind( alsa_handle,
put_back_samples );

    return 0;
}

/**
 * the latency callback.
 * sets up the latencies on the ports.
 */

void
latency_cb (jack_latency_callback_mode_t mode,
void *arg)
{
    jack_latency_range_t range;
    JSList *node;

    range.min = range.max = target_delay;
```

```
    if (mode == JackCaptureLatency) {
        for (node = capture_ports; node; node =
jack_slist_next (node)) {
            jack_port_t *port = node->data;
            jack_port_set_latency_range (port,
mode, &range);
        }
    } else {
        for (node = playback_ports; node; node
= jack_slist_next (node)) {
            jack_port_t *port = node->data;
            jack_port_set_latency_range (port,
mode, &range);
        }
    }
}



/**
 * Allocate the necessary jack ports...
 */

void alloc_ports( int n_capture, int n_playback
) {

    int port_flags = JackPortIsOutput;
    int chn;
```

```
    jack_port_t *port;
    char buf[32];

    capture_ports = NULL;
    for (chn = 0; chn < n_capture; chn++)
    {
    snprintf (buf, sizeof(buf) - 1,
"capture_%u", chn+1);

    port = jack_port_register (client, buf,
        JACK_DEFAULT_AUDIO_TYPE,
        port_flags, 0);

    if (!port)
    {
        printf( "jacknet_client: cannot
register port for %s", buf);
        break;
    }

    capture_srcs = jack_slist_append(
capture_srcs, src_new( 4-samplerate_quality, 1,
NULL ) );
    capture_ports = jack_slist_append
(capture_ports, port);
    }
```

```
    port_flags = JackPortIsInput;

    playback_ports = NULL;
    for (chn = 0; chn < n_playback; chn++)
    {
    snprintf (buf, sizeof(buf) - 1,
"playback_%u", chn+1);

    port = jack_port_register (client, buf,
        JACK_DEFAULT_AUDIO_TYPE,
        port_flags, 0);

    if (!port)
    {
        printf( "jacknet_client: cannot
register port for %s", buf);
        break;
    }

    playback_srcs = jack_slist_append(
playback_srcs, src_new( 4-samplerate_quality,
1, NULL ) );
    playback_ports = jack_slist_append
(playback_ports, port);
    }
}
```

```
/**
 * This is the shutdown callback for this JACK
application.
 * It is called by JACK if the server ever
shuts down or
 * decides to disconnect the client.
 */

void jack_shutdown (void *arg) {

    exit (1);
}

/**
 * be user friendly.
 * be user friendly.
 * be user friendly.
 */

void printUsage() {
fprintf(stderr, "usage: alsa_out [options]\n"
        "\n"
        "  -j <jack name> - client name\n"
        "  -d <alsa_device> \n"
        "  -c <channels> \n"
        "  -p <period_size> \n"
        "  -n <num_period> \n"
```

```
        "  -r <sample_rate> \n"
        "  -q <sample_rate quality [0..4]\n"
        "  -m <max_diff> \n"
        "  -t <target_delay> \n"
        "  -i  turns on instrumentation\n"
        "  -v  turns on printouts\n"
        "\n");
}



/**
 * the main function....
 */

void
sigterm_handler( int signal )
{
    quit = 1;
}



int main (int argc, char *argv[]) {
    char jack_name[30] = "alsa_in";
    char alsa_device[30] = "hw:0";

    extern char *optarg;
    extern int optind, optopt;
```

```
    int errflg=0;
    int c;

    while ((c = getopt(argc, argv,
"ivj:r:c:p:n:d:q:m:t:f:F:C:Q:s:")) != -1) {
    switch(c) {
        case 'j':
        strcpy(jack_name,optarg);
        break;
        case 'r':
        sample_rate = atoi(optarg);
        break;
        case 'c':
        num_channels = atoi(optarg);
        break;
        case 'p':
        period_size = atoi(optarg);
        break;
        case 'n':
        num_periods = atoi(optarg);
        break;
        case 'd':
        strcpy(alsa_device,optarg);
        break;
        case 't':
        target_delay = atoi(optarg);
        break;
```

```
        case 'q':
        samplerate_quality = atoi(optarg);
        break;
        case 'm':
        max_diff = atoi(optarg);
        break;
        case 'f':
        catch_factor = atoi(optarg);
        break;
        case 'F':
        catch_factor2 = atoi(optarg);
        break;
        case 'C':
        pclamp = (double) atoi(optarg);
        break;
        case 'Q':
        controlquant = (double) atoi(optarg);
        break;
        case 'v':
        verbose = 1;
        break;
        case 'i':
        instrument = 1;
        break;
        case 's':
        smooth_size = atoi(optarg);
        break;
```

```
        case ':':
        fprintf(stderr,
            "Option -%c requires an operand\n",
optopt);
        errflg++;
        break;
        case '?':
        fprintf(stderr,
            "Unrecognized option: -%c\n",
optopt);
        errflg++;
    }
    }
    if (errflg) {
    printUsage();
    exit(2);
    }

    if( (samplerate_quality < 0) ||
(samplerate_quality > 4) ) {
    fprintf (stderr, "invalid samplerate
quality\n");
    return 1;
    }
    if ((client = jack_client_open (jack_name,
0, NULL)) == 0) {
    fprintf (stderr, "jack server not running?
```

```
\n");
    return 1;
    }

    /* tell the JACK server to call `process()'
whenever
       there is work to be done.
       */

    jack_set_process_callback (client, process,
0);

    /* tell the JACK server to call
`jack_shutdown()' if
       it ever shuts down, either entirely, or
if it
       just decides to stop calling us.
       */

    jack_on_shutdown (client, jack_shutdown,
0);

    if (jack_set_latency_callback)
        jack_set_latency_callback (client,
latency_cb, 0);

    // get jack sample_rate
```

```
    jack_sample_rate = jack_get_sample_rate(
client );

    if( !sample_rate )
    sample_rate = jack_sample_rate;

    // now open the alsa fd...
    alsa_handle = open_audiofd( alsa_device, 1,
sample_rate, num_channels, period_size,
num_periods);
    if( alsa_handle == 0 )
    exit(20);

    printf( "selected sample format: %s\n",
formats[format].name );

    static_resample_factor = (double)
jack_sample_rate / (double) sample_rate;
    resample_lower_limit =
static_resample_factor * 0.25;
    resample_upper_limit =
static_resample_factor * 4.0;
    resample_mean = static_resample_factor;

    offset_array = malloc( sizeof(double) *
smooth_size );
```

```
    if( offset_array == NULL ) {
        fprintf( stderr, "no memory for
offset_array !!!\n" );
        exit(20);
    }
    window_array = malloc( sizeof(double) *
smooth_size );
    if( window_array == NULL ) {
        fprintf( stderr, "no memory for
window_array !!!\n" );
        exit(20);
    }
    int i;
    for( i=0; i<smooth_size; i++ ) {
        offset_array[i] = 0.0;
        window_array[i] = hann( (double) i /
((double) smooth_size - 1.0) );
    }

    jack_buffer_size = jack_get_buffer_size(
client );
    // Setup target delay and max_diff for the
normal user, who does not play with them...
    if( !target_delay )
    target_delay = (num_periods*period_size /
2) + jack_buffer_size/2;
```

```
    if( !max_diff )
    max_diff = num_periods*period_size -
target_delay ;

    if( max_diff > target_delay ) {
        fprintf( stderr, "target_delay (%d)
cant be smaller than max_diff(%d)\n",
target_delay, max_diff );
        exit(20);
    }
    if( (target_delay+max_diff) >
(num_periods*period_size) ) {
        fprintf( stderr, "target_delay+max_diff
(%d) cant be bigger than buffersize(%d)\n",
target_delay+max_diff, num_periods*period_size
);
        exit(20);
    }
    // alloc input ports, which are blasted out
to alsa...
    alloc_ports( num_channels, 0 );

    outbuf = malloc( num_periods * period_size
* formats[format].sample_size * num_channels );
    resampbuf = malloc( num_periods *
period_size * sizeof( float ) );
    tmpbuf = malloc( 512 *
```

```
formats[format].sample_size * num_channels );

    if ((outbuf == NULL) || (resampbuf == NULL)
|| (tmpbuf == NULL))
    {
        fprintf( stderr, "no memory for
buffers.\n" );
        exit(20);
    }

    memset( tmpbuf, 0, 512 *
formats[format].sample_size * num_channels);

    /* tell the JACK server that we are ready
to roll */

    if (jack_activate (client)) {
    fprintf (stderr, "cannot activate client");
    return 1;
    }

    signal( SIGTERM, sigterm_handler );
    signal( SIGINT, sigterm_handler );

    if( verbose ) {
        while(!quit) {
            usleep(500000);
```

```
            if( output_new_delay ) {
                printf( "delay = %d\n",
output_new_delay );
                output_new_delay = 0;
            }
            printf( "res: %f, \tdiff = %f,
\toffset = %f \n", output_resampling_factor,
output_diff, output_offset );
        }
    } else if( instrument ) {
        printf( "#
n\tresamp\tdiff\toffseti\tintegral\n");
        int n=0;
        while(!quit) {
            usleep(1000);
            printf( "%d\t%f\t%f\t%f\t%f\n",
n++, output_resampling_factor, output_diff,
output_offset, output_integral );
        }
    } else {
        while(!quit)
        {
            usleep(500000);
            if( output_new_delay ) {
                printf( "delay = %d\n",
output_new_delay );
                output_new_delay = 0;
```

```
            }
        }
    }

    jack_deactivate( client );
    jack_client_close (client);
    exit (0);
}
```

Conclusion

This chapter has considered using Jack from a user viewpoint and also looked at programming Jack clients.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

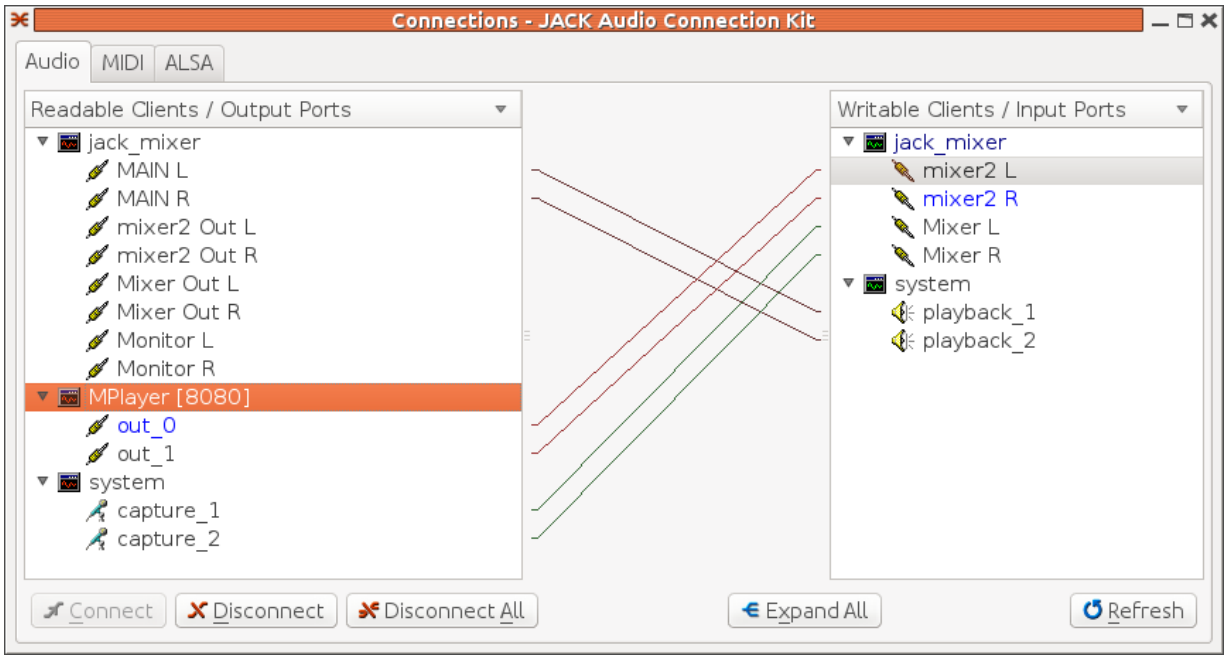
A complex sound system may consist of multiple sound sources, multiple filters and multiple outputs. If they have to be set up afresh each time they are used, then there can be errors, wasted time, etc. Session management attempts to solve these problems.

Resources

- Dave Phillips has A brief survey of Linux audio session managers
- Session Management Overview contains links to many resources
- LADI Session Handler
- the Jack session API at trac
- JACK-AUDIO-CONNECTION-KIT - Session API for clients
- How to build LADISH-aware apps at Level 1 app examples
- and also see LADI Session Handler
- Non Session Management API

Session management issues

Whenever there are multiple modules linked in some way, there can be a need to manage the modules and their linkages. These arise quite quickly in the Jack environment, which is designed for multiple linkages. It doesn't take a very complex arrangement of Jack modules for management to become tedious. For example, consider the mixer session of the previous chapter:



To set this up from the beginning requires

- Start `jackd` and `qjackctl`
- Start `jack_mixer`
- Open up two new sets of input ports on the mixer

- Connect the MAIN mixer output ports to the playback ports
- Connect the microphone ports to one set of mixer input ports
- Start `mplayer`, which automatically connects to the playback ports
- Disconnect the `mplayer` output ports from the playback ports and reconnect them to the other set of mixer input ports

You don't want to do this every time you play a song!

The LADISH session manager identifies different levels of control of applications by session managers. Removing the explicit references to particular managers and frameworks, they are:

- level 0 - an application is not linked to a session handling library. User has to save application projects manually or rely on autosave support from application.
- level 1 - an application is not linked to a session handling library. Application saves when particular messages or signals are received.
- level 2, an application is linked to a session management library. Limited interaction with session handler because of limitations in the session manager

- level 3 - an application is linked to a sophisticated session manager. Full interaction with the session handler

As Dave Phillips points out, "The use of these levels is an attempt to sort and regulate the various possible conditions for any Linux audio application. Those conditions include the degree of JACK compliance, any WINE or DOS requirements, network operation, the multiplicity of existing APIs, and so forth. "

The current batch of session management frameworks used for Linux audio includes

- LASH
- Jack session management
- LADISH
- Non-session manager
- Chino

The existence of multiple managers means that most applications will only support the protocols of only one or at most a few. If you choose a particular manager then you will be restricted to the applications you can run under its control.

jack_connect

The programs `jack_connect` and `jack_disconnect` canbe used to reconfigure connections between clients. For example, the MIDI player `TiMidity` will connect its output ports to the first available Jack input ports, which are generally the system ports connected to the sound card. If you wish to connect `TiMidity` to, say, `jack-rack` then its output ports have to be first disconnected and then connected to the correct ones. On the other hand, `jack-rack` does not connect to anything by default so may need to be connected to the system ports. This is done by e.g.

```
jack_disconnect TiMidity:port_1
system:playback_1
jack_disconnect TiMidity:port_2
system:playback_2

jack_connect TiMidity:port_1 jack_rack:in_1
jack_connect TiMidity:port_2 jack_rack:in_2

jack_connect jack_rack:out_1 system:playback_1
jack_connect jack_rack:out_2 system:playback_2
```

LASH

This was the earliest successful session mananager for Linux audio but has since fallen out of use. It does not seem to be in the Ubuntu repositories any more.

One of the applications requiring LASH is `jack_mixer`. Even worse, it uses the Python `LASH` module from the `python-lash.2.7.4-0ubuntu` package. The only copy I can find requires a version of Python less than 2.7 and the installed version of Python is 2.7.4. This is an application which at present will not benefit from current session management tools - while it might run as Level 1 with LASH, it can only run at Level 0 with other session managers.

So there are Jack applications which require LASH for session management but no such support seems to exist any more.

Jack sessions

A list of Jack session aware applications as at 2010 is at apps supporting jack-session and Jack Session

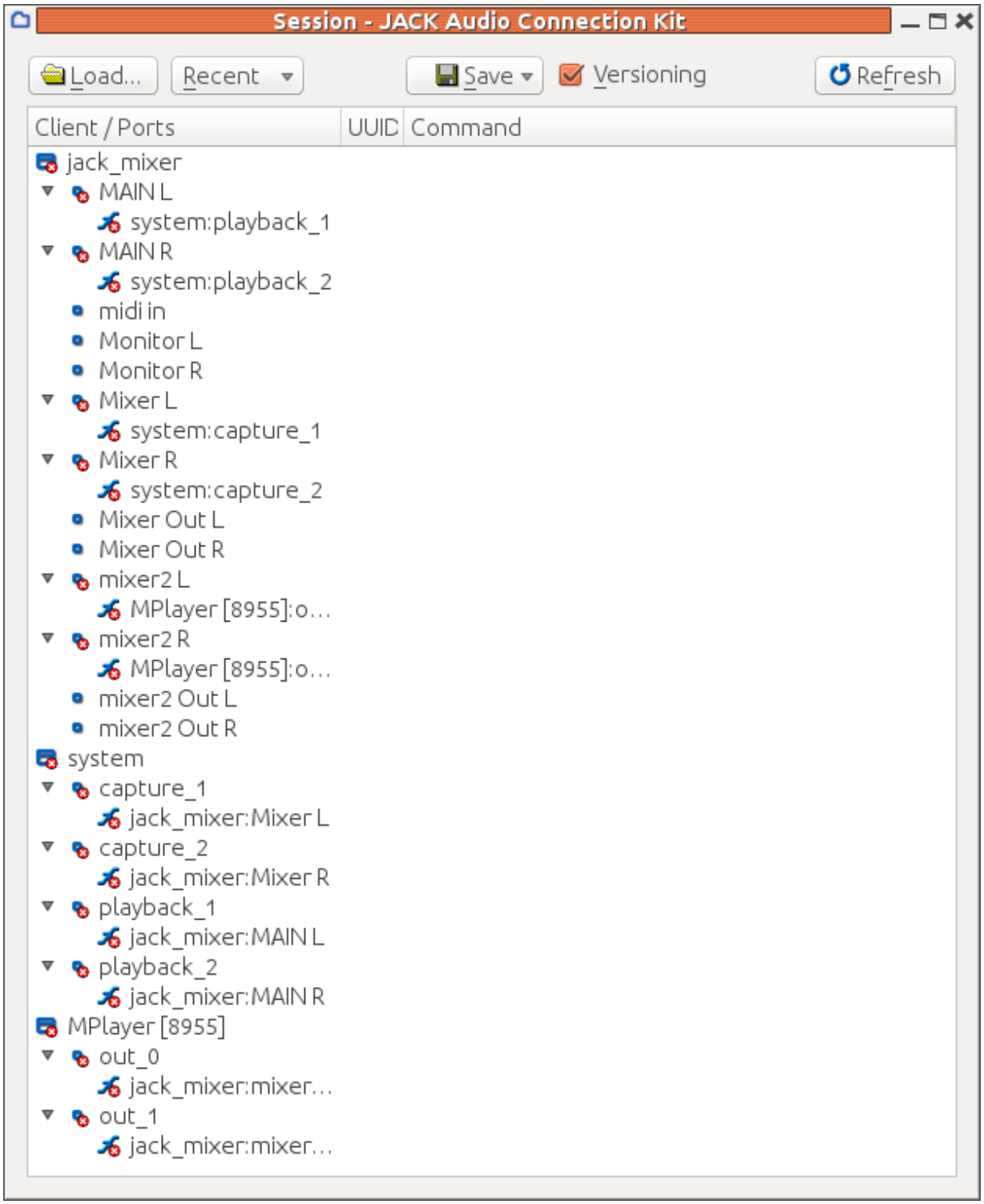
`qjackctl` has a session manager which will allow you to save and restore sessions. You save a session by clicking on the Session button and then choosing a session name and directory. It stores the session information as an XML file in whatever directory you save it. For the session above, this looks like

```
<!DOCTYPE qjackctlSession>
<session name="session2">
 <client name="jack_mixer">
  <port type="out" name="MAIN L">
   <connect port="playback_1" client="system"/>
  </port>
  <port type="out" name="MAIN R">
   <connect port="playback_2" client="system"/>
  </port>
  <port type="in" name="midi in"/>
  <port type="out" name="Monitor L"/>
  <port type="out" name="Monitor R"/>
  <port type="in" name="Mixer L">
   <connect port="capture_1" client="system"/>
  </port>
  <port type="in" name="Mixer R">
   <connect port="capture_2" client="system"/>
  </port>
  <port type="out" name="Mixer Out L"/>
  <port type="out" name="Mixer Out R"/>
  <port type="in" name="mixer2 L">
   <connect port="out_0" client="MPlayer
[8955]"/>
  </port>
  <port type="in" name="mixer2 R">
   <connect port="out_1" client="MPlayer
```

```
[8955]"/>
  </port>
  <port type="out" name="mixer2 Out L"/>
  <port type="out" name="mixer2 Out R"/>
 </client>
 <client name="system">
  <port type="out" name="capture_1">
   <connect port="Mixer L"
client="jack_mixer"/>
  </port>
  <port type="out" name="capture_2">
   <connect port="Mixer R"
client="jack_mixer"/>
  </port>
  <port type="in" name="playback_1">
   <connect port="MAIN L" client="jack_mixer"/>
  </port>
  <port type="in" name="playback_2">
   <connect port="MAIN R" client="jack_mixer"/>
  </port>
 </client>
 <client name="MPlayer [8955]">
  <port type="out" name="out_0">
   <connect port="mixer2 L"
client="jack_mixer"/>
  </port>
  <port type="out" name="out_1">
```

```
    <connect port="mixer2 R"
 client="jack_mixer"/>
   </port>
  </client>
 </session>
```

On loading the session, it looks like


Session - JACK Audio Connection Kit
Load...
Recent
Save
Versioning
Refresh
Client / Ports
UUID
Command
jack_mixer
MAIN L
system:playback_1
MAIN R
system:playback_2
midi in
Monitor L
Monitor R
Mixer L
system:capture_1
Mixer R
system:capture_2
Mixer Out L
Mixer Out R
mixer2 L
MPlayer [8955]:o...
mixer2 R
MPlayer [8955]:o...
mixer2 Out L
mixer2 Out R
system
capture_1
jack_mixer:Mixer L
capture_2
jack_mixer:Mixer R
playback_1
jack_mixer:MAIN L
playback_2
jack_mixer:MAIN R
MPlayer [8955]
out_0
jack_mixer:mixer...
out_1
jack_mixer:mixer...

There are many red crosses - restoring a session doesn't start these particular applications. If you re-start `jack_mixer` by hand, then it establishes the links between its MAIN output ports and system playback ports, and several of the red crosses disappear. But it doesn't create the extra ports that we created earlier. We need to repeat the work of creating new input ports with the right names and then `qjackctl` does re-establish the connections, and more red crosses disappear.

If we run `mplayer` again, it just establishes its own default connections to the playback ports and has to be re-mapped by hand. It doesn't even seem to meet Level 0, as `qjackctl` doesn't remap its connections automatically.

The issue here is that the `mplayer` and `jack_mixer` do not talk the Jack session management protocol. The session manager does reset any connections made by some applications, but not all of them. An example is given later of adding Jack session management to an application and then it will be restarted and reconnected properly.

LADISH

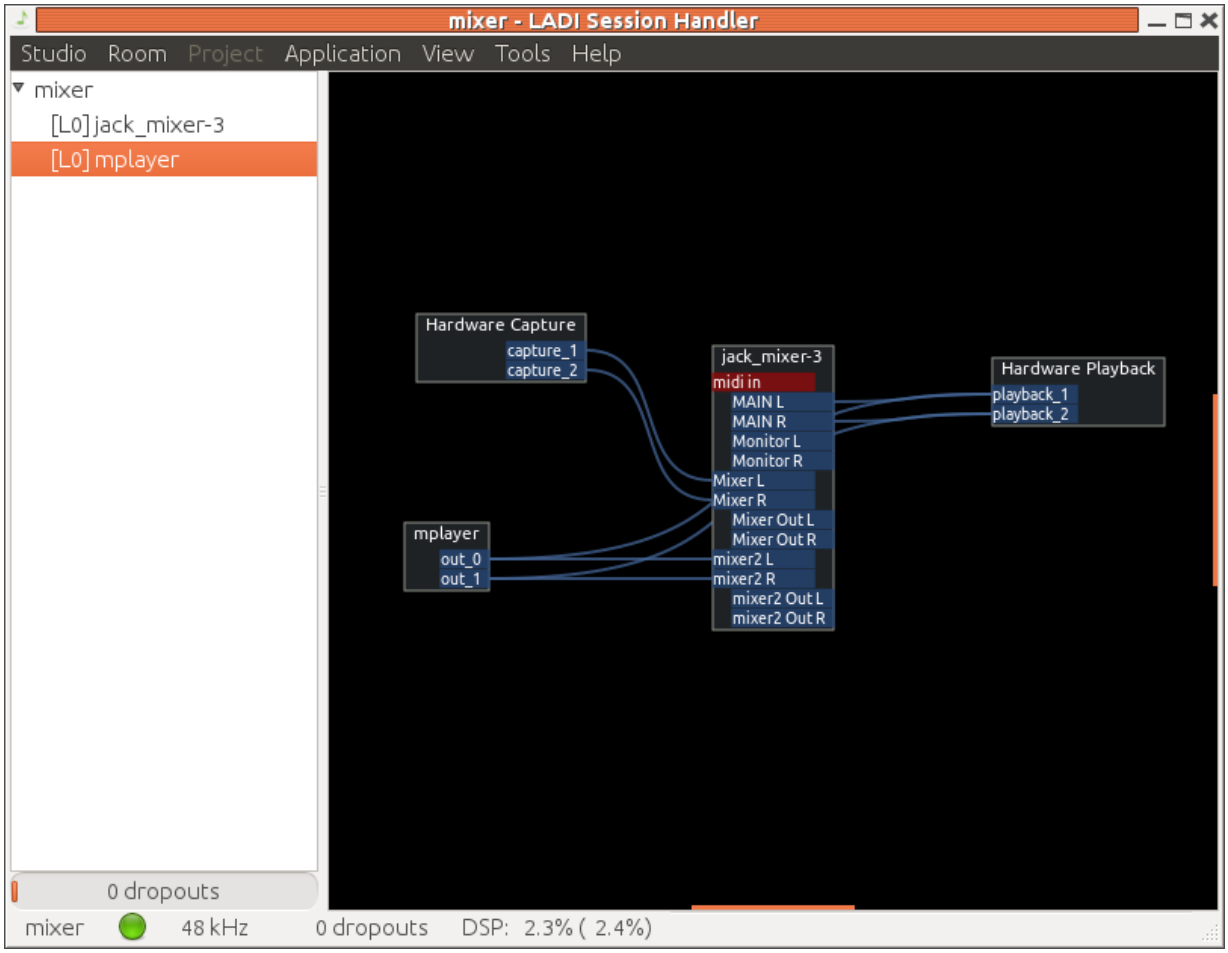
LADISH is designed as the successor to LASH and is available in the repositories.

LADISH can start, stop and configure sessions. In particular, it can setup different Jack configurations. This means that you do not start Jack and then start LADISH, but the other way around: start the GUI tool `gladish`, configure Jack and then start a session. The process is described in the LADI Session Handler Wiki - follow it, in particular connecting Jack to, say, ALSA. Otherwise you will get no sound! See also The LADI Session Handler by the Penguin Producer.

Once set up, start a new Studio and then start applications from its Applications menu. To run `mplayer` you need to give the full command such as

```
mplayer -ao jack 54154.mp3
```

You can start `jack_mixer` from the Applications menu and then add two new sets of input ports, as in the Jack chapter. After reconnecting them, you end with a connection graph like



Connection graphs are stored as an XML file in `$HOME/.ladish`. For example, the above graph is stored as

```
<?xml version="1.0"?>
<!--
ladish Studio configuration.
-->
<!-- Sun Sep 29 10:49:54 2013 -->
<studio>
  <jack>
    <conf>
      <parameter
path="/engine/driver">alsa</parameter>
      <parameter path="/engine/client-
timeout">500</parameter>
      <parameter path="/engine/port-
max">64</parameter>
    </conf>
    <clients>
      <client name="system" uuid="5ef937c6-
46f7-45cd-8441-8ff6e2aee4eb">
        <ports>
          <port name="capture_1"
uuid="9432f206-44c3-45cb-8024-3ba7160962bc" />
          <port name="capture_2"
uuid="3c9acf5c-c91d-4692-add2-e3defb7c508a" />
          <port name="playback_1"
uuid="95c68011-dab9-401c-8904-b3d149e20570" />
          <port name="playback_2"
```

uuid="5b8e9215-3ff4-4973-8c0b-1eb5ab7ccc9b" />
        </ports>
      </client>
      <client name="jack_mixer-3"
uuid="4538833e-d7e7-47d0-8a43-67ee25d17898">
        <ports>
          <port name="midi in" uuid="17d04191-
f59d-4d16-970c-55030162aae7" />
          <port name="MAIN L" uuid="9d986401-
c303-4f35-89b7-a32e10120ce4" />
          <port name="MAIN R" uuid="fae94d01-
00ef-449d-8e05-f95df84c5357" />
          <port name="Monitor L"
uuid="1758d824-75cd-46b3-8e53-82c6be1ca200" />
          <port name="Monitor R"
uuid="d14815e9-d3bc-457b-8e4f-29ad29ea36f7" />
          <port name="Mixer L" uuid="07d388ed-
d00a-4ee0-92aa-3ae79200e11e" />
          <port name="Mixer R" uuid="d1eb3400-
75ce-422d-b9b8-b7e670f95428" />
          <port name="Mixer Out L"
uuid="fad2a77e-6146-4919-856f-b6f7befdb84d" />
          <port name="Mixer Out R"
uuid="920c5d12-9f62-46aa-b191-52bfbb94065d" />
          <port name="mixer2 L" uuid="c2b96996-
9cd1-41dd-a750-192bb5717438" />
          <port name="mixer2 R" uuid="3de52738-

d7e8-4733-bf08-3ea2b6372a4c" />
          <port name="mixer2 Out L"
uuid="4e08eba4-a0c1-4e76-9dff-c14f76d5328e" />
          <port name="mixer2 Out R"
uuid="9d2f79a5-e2d0-484b-b094-98ef7a4f61a7" />
        </ports>
      </client>
      <client name="mplayer" uuid="66e0d45f-
2e21-4fbf-ac34-5d3658ee018a">
        <ports>
          <port name="out_0" uuid="83152a6e-
e6f6-4357-93ce-020ba58b7d00" />
          <port name="out_1" uuid="55a05594-
174d-48a5-805b-96d2c9e77cf1" />
        </ports>
      </client>
    </clients>
  </jack>
  <clients>
    <client name="Hardware Capture"
uuid="47c1cd18-7b21-4389-bec4-6e0658e1d6b1"
naming="app">
      <ports>
        <port name="capture_1" uuid="9432f206-
44c3-45cb-8024-3ba7160962bc" type="audio"
direction="output" />
        <port name="capture_2" uuid="3c9acf5c-

c91d-4692-add2-e3defb7c508a" type="audio"
direction="output" />
      </ports>
      <dict>
        <key
name="http://ladish.org/ns/canvas/x">1364.000000<

        <key
name="http://ladish.org/ns/canvas/y">1083.000000<

      </dict>
    </client>
    <client name="Hardware Playback"
uuid="b2a0bb06-28d8-4bfe-956e-eb24378f9629"
naming="app">
      <ports>
        <port name="playback_1" uuid="95c68011-
dab9-401c-8904-b3d149e20570" type="audio"
direction="input" />
        <port name="playback_2" uuid="5b8e9215-
3ff4-4973-8c0b-1eb5ab7ccc9b" type="audio"
direction="input" />
      </ports>
      <dict>
        <key
name="http://ladish.org/ns/canvas/x">1745.000000<

```
        <key
name="http://ladish.org/ns/canvas/y">1112.000000<

      </dict>
    </client>
    <client name="jack_mixer-3" uuid="4b198f0f-
5a77-4486-9f54-f7ec044d9bf2" naming="app"
app="98729282-8b18-4bcf-b929-41bc53f2b4ed">
      <ports>
        <port name="midi in" uuid="17d04191-
f59d-4d16-970c-55030162aae7" type="midi"
direction="input" />
        <port name="MAIN L" uuid="9d986401-
c303-4f35-89b7-a32e10120ce4" type="audio"
direction="output" />
        <port name="MAIN R" uuid="fae94d01-
00ef-449d-8e05-f95df84c5357" type="audio"
direction="output" />
        <port name="Monitor L" uuid="1758d824-
75cd-46b3-8e53-82c6be1ca200" type="audio"
direction="output" />
        <port name="Monitor R" uuid="d14815e9-
d3bc-457b-8e4f-29ad29ea36f7" type="audio"
direction="output" />
        <port name="Mixer L" uuid="07d388ed-
d00a-4ee0-92aa-3ae79200e11e" type="audio"
direction="input" />
```

```
        <port name="Mixer R" uuid="d1eb3400-
75ce-422d-b9b8-b7e670f95428" type="audio"
direction="input" />
        <port name="Mixer Out L"
uuid="fad2a77e-6146-4919-856f-b6f7befdb84d"
type="audio" direction="output" />
        <port name="Mixer Out R"
uuid="920c5d12-9f62-46aa-b191-52bfbb94065d"
type="audio" direction="output" />
        <port name="mixer2 L" uuid="c2b96996-
9cd1-41dd-a750-192bb5717438" type="audio"
direction="input" />
        <port name="mixer2 R" uuid="3de52738-
d7e8-4733-bf08-3ea2b6372a4c" type="audio"
direction="input" />
        <port name="mixer2 Out L"
uuid="4e08eba4-a0c1-4e76-9dff-c14f76d5328e"
type="audio" direction="output" />
        <port name="mixer2 Out R"
uuid="9d2f79a5-e2d0-484b-b094-98ef7a4f61a7"
type="audio" direction="output" />
      </ports>
      <dict>
        <key
name="http://ladish.org/ns/canvas/x">1560.000000<


        <key
```

name="http://ladish.org/ns/canvas/y">1104.000000<

      </dict>
    </client>
    <client name="mplayer" uuid="2f15cfec-7f6d-41b4-80e8-e1ae80c3be9e" naming="app" app="7a9be17b-eb40-4be3-a9dc-82f36bbceeeb">
      <ports>
        <port name="out_0" uuid="83152a6e-e6f6-4357-93ce-020ba58b7d00" type="audio" direction="output" />
        <port name="out_1" uuid="55a05594-174d-48a5-805b-96d2c9e77cf1" type="audio" direction="output" />
      </ports>
      <dict>
        <key name="http://ladish.org/ns/canvas/x">1350.000000<

        <key name="http://ladish.org/ns/canvas/y">1229.000000<

      </dict>
    </client>
  </clients>
  <connections>
    <connection port1="9432f206-44c3-45cb-8024-

3ba7160962bc" port2="07d388ed-d00a-4ee0-92aa-3ae79200e11e" />
    <connection port1="3c9acf5c-c91d-4692-add2-e3defb7c508a" port2="d1eb3400-75ce-422d-b9b8-b7e670f95428" />
    <connection port1="fad2a77e-6146-4919-856f-b6f7befdb84d" port2="95c68011-dab9-401c-8904-b3d149e20570" />
    <connection port1="920c5d12-9f62-46aa-b191-52bfbb94065d" port2="5b8e9215-3ff4-4973-8c0b-1eb5ab7ccc9b" />
    <connection port1="83152a6e-e6f6-4357-93ce-020ba58b7d00" port2="c2b96996-9cd1-41dd-a750-192bb5717438" />
    <connection port1="55a05594-174d-48a5-805b-96d2c9e77cf1" port2="3de52738-d7e8-4733-bf08-3ea2b6372a4c" />
  </connections>
  <applications>
    <application name="jack_mixer-3" uuid="98729282-8b18-4bcf-b929-41bc53f2b4ed" terminal="false" level="0" autorun="true">jack_mixer</application>
    <application name="mplayer" uuid="7a9be17b-eb40-4be3-a9dc-82f36bbceeeb" terminal="true" level="0" autorun="true">mplayer -ao jack %2Fhome%2Fhttpd%2Fhtml%2FLinuxSound%2FKaraoke%2FS

```
  </applications>
</studio>
```

The full command to restart `mplayer` is stored in this file, as are all the connections made.

On stopping and restarting a session, `mplayer` is started with the same MP3 file, but has the default connections. It ignores the connections of the LADISH session. Similarly, `jack_mixer` is restarted, but the additional ports have to be recreated by hand - this is not a LADISH aware application so it runs at Level 0. However, once created, the LADISH reconnections are made okay.

A list of LADISH-aware applications is at ladish: LADI Session Handler

From the user's viewpoint, the difference between these session managers are

- Jack applications can be started in any manner and will be picked up by the Jack session manager. However, any specific command-line parameters will be lost

- Applications need to be started by the LADISH session manager in order to be managed by it. However, it can record command line parameters and restart the application using them

From the developer's viewpoint (see later), the difference between these session managers are

- Jack session aware applications can be started in any manner and will encode the command line required to restart them in the program
-

Non-session manager

List of NSM applications at NSM - Non Session Management

Non Session Management API

Jack session API

See the session API at trac.

Applications that can be managed by Jack Sessions (JS) may be JS aware at Level 1 or JS unaware. For the unaware ones, the best that can be done is for the session manager to maybe start and stop them. For the JS aware applications, they must be set up to

- Register with a JS manager
- Respond to messages from the JS manager
- Be startable with session information

Response to a JS message will generally

- Save the application's state into a file, where the directory is given by the session manager
- Reply to the session manager with a command that can be used to restart the application, with enough information that it can restore its state (typically the name of the file in which it stored its state information)

JS aware clients identify themselves to the session manager by a UUID (unique universal identifier). It doesn't seem to matter what this is or how it is generated: the client application just makes it up as long as it is an integer represented as a string. This is passed to the session manager when registering, but should also be passed back to the client when it is restarted by the session manager. This is done by a command line argument to the application, and the format of the command line is also up to the client.

A simple case might be two options `-u` for UUID and `-f` for saved state file. This would be parsed using `getopt` by

```
int main(int argc, char **argv) {
  int c;
  char *file = NULL;
  char *uuid = "13";
  while ((c = getopt (argc, argv, "f:u:")) !=
-1)
    switch (c) {
      case 'u':
        uuid = optarg;
        break;
      case 'f':
        file = optarg;
        break;
      ...
    }
  }
  ...
}
```

The application could then restore its state using the information it has previously stored in the state file, and then register again with a session manager by

```
jack_client *client;
client = jack_client_open("myapp",
JackSessionID, NULL, uuid);
jack_set_session_callback(client,
session_callback, NULL);
```

The callback function `session_callback` is invoked whenever the session manager needs to communicate with the application. It takes a `jack_session_event` and whatever was passed as the last argument to `jack_set_session_callback`.

The job of the callback is then to save state information, pass information back to the session manager and perhaps exit (from trac - the session API):

```
int session_callback(jack_session_event_t *ev)
{
  char filename[256];
  char command[256];

  snprintf(filename, sizeof(filename),
"%smyfile.state", ev->session_dir);
  snprintf(command,  sizeof(command),
           "my_app -u %s -f
${SESSION_DIR}myfile.state", ev->client_uuid);
  your_save_function(filename);
  ev->command_line = strdup(command);
  jack_session_reply(jack_client, ev);
  if(ev->type == JackSessionSaveAndQuit)
      quit();
  jack_session_event_free(ev);
  return 0;
}
```

trac suggests that if this is run in a multi-threaded environment such as GTK, it should be run when other threads are idle e.g. by `g_idel_add` .

WE can illustrate this with the `delay` program from the Jack chapter. Adding in the extra code gives a revised `delay.c`. I have enclosed the extra code with `#ifdef JACK_SESSION` for ease in seeing the changes.

```
/** @file delay.c
 *
 * @brief This client delays one channel by
4096 framse.
 */

#define JACK_SESSION

#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <signal.h>
#ifndef WIN32
#include <unistd.h>
#endif
#include <jack/jack.h>

#ifdef JACK_SESSION
#include <jack/session.h>
#endif

jack_port_t **input_ports;
jack_port_t **output_ports;
jack_client_t *client;
```

```
#define SIZE 8192
#define DELAY 4096
jack_default_audio_sample_t buffer[SIZE];
int idx, delay_idx;

static void signal_handler ( int sig )
{
    jack_client_close ( client );
    fprintf ( stderr, "signal received, exiting
...\n" );
    exit ( 0 );
}

static void copy2out(
jack_default_audio_sample_t *out,
              jack_nframes_t nframes) {
    if (delay_idx + nframes < SIZE) {
    memcpy(out, buffer + delay_idx,
           nframes * sizeof (
jack_default_audio_sample_t ) );
    } else {
    int frames_to_end = SIZE - delay_idx;
    int overflow = delay_idx + nframes - SIZE;
    memcpy(out, buffer + delay_idx,
           frames_to_end * sizeof (
jack_default_audio_sample_t ) );
```

```
    memcpy(out, buffer, overflow *
sizeof(jack_default_audio_sample_t));
    }
    delay_idx = (delay_idx + nframes) % SIZE;
}

static void copy2buffer(
jack_default_audio_sample_t *in,
             jack_nframes_t nframes) {
    if (idx + nframes < SIZE) {
    memcpy(buffer + idx, in,
           nframes * sizeof (
jack_default_audio_sample_t ) );
    } else {
    int frames_to_end = SIZE - idx;
    int overflow = idx + nframes - SIZE;
    memcpy(buffer + idx, in,
           frames_to_end * sizeof (
jack_default_audio_sample_t ) );
    memcpy(buffer, in, overflow *
sizeof(jack_default_audio_sample_t));
    }
    idx = (idx + nframes) % SIZE;
}

/**
 * The process callback for this JACK
```

```
application is called in a
 * special realtime thread once for each audio
cycle.
 *
 * This client follows a simple rule: when the
JACK transport is
 * running, copy the input port to the output.
When it stops, exit.
 */

int
process ( jack_nframes_t nframes, void *arg )
{
    int i;
    jack_default_audio_sample_t *in, *out;

    in = jack_port_get_buffer ( input_ports[0],
nframes );
    out = jack_port_get_buffer (
output_ports[0], nframes );
    memcpy ( out, in, nframes * sizeof (
jack_default_audio_sample_t ) );

    in = jack_port_get_buffer ( input_ports[1],
nframes );
    out = jack_port_get_buffer (
output_ports[1], nframes );
```

```
    copy2out(out, nframes);
    copy2buffer(in, nframes);

    return 0;
}

/**
 * JACK calls this shutdown_callback if the
server ever shuts down or
 * decides to disconnect the client.
 */
void
jack_shutdown ( void *arg ) {
    free ( input_ports );
    free ( output_ports );
    exit ( 1 );
}

#ifdef JACK_SESSION
/*
 * Callback function for JS
 */
void session_callback(jack_session_event_t *ev,
void *args) {
    char command[256];

    snprintf(command,  sizeof(command),
```

```
"/home/httpd/html/LinuxSound/Sampled/SessionManag
 -u %s",
         ev->client_uuid);
    ev->flags = JackSessionNeedTerminal;
    ev->command_line = strdup(command);
    jack_session_reply(client, ev);

    if(ev->type == JackSessionSaveAndQuit)
         jack_shutdown(NULL);

    jack_session_event_free(ev);
}
#endif

int main ( int argc, char *argv[] ) {
    int i;
    const char **ports;
    const char *client_name;
    const char *server_name = NULL;
    jack_status_t status;

#ifdef JACK_SESSION
    /*
     * Extra code for JS
     */
    int c;
```

```
    char *uuid = "13";
    while ((c = getopt (argc, argv, "u:")) !=
-1)
    switch (c) {
    case 'u':
        uuid = optarg;
        break;
    }
    printf("UUID is %s\n", uuid);
#endif

    client_name = strrchr ( argv[0], '/' );
    if ( client_name == 0 ) {
    client_name = argv[0];
    }
    else {
    client_name++;
    }

    /* open a client connection to the JACK
server */
    /* Changed args for JS */

#ifdef JACK_SESSION
    client = jack_client_open ( client_name,
JackSessionID, &status, uuid);
#else
```

```
    client = jack_client_open ( client_name,
JackNullOption, &status);
#endif
    if ( client == NULL )
    {
        fprintf ( stderr, "jack_client_open()
failed, "
              "status = 0x%2.0x\n", status );
        if ( status & JackServerFailed )
        {
            fprintf ( stderr, "Unable to
connect to JACK server\n" );
        }
        exit ( 1 );
    }
    if ( status & JackServerStarted )
    {
        fprintf ( stderr, "JACK server
started\n" );
    }
    if ( status & JackNameNotUnique )
    {
        client_name = jack_get_client_name (
client );
        fprintf ( stderr, "unique name `%s'
assigned\n", client_name );
    }
```

```
#ifdef JACK_SESSION
    /* Set callback function for JS
     */
    jack_set_session_callback(client,
session_callback, NULL);
#endif

    /* tell the JACK server to call `process()'
whenever
       there is work to be done.
    */
    jack_set_process_callback ( client,
process, 0 );

    /* tell the JACK server to call
`jack_shutdown()' if
       it ever shuts down, either entirely, or
if it
       just decides to stop calling us.
    */

    jack_on_shutdown ( client, jack_shutdown, 0
);

    /* create two ports pairs*/
    input_ports = ( jack_port_t** ) calloc ( 2,
```

```
sizeof ( jack_port_t* ) );
    output_ports = ( jack_port_t** ) calloc (
2, sizeof ( jack_port_t* ) );

    char port_name[16];
    for ( i = 0; i < 2; i++ )
    {
        sprintf ( port_name, "input_%d", i + 1
);
        input_ports[i] = jack_port_register (
client, port_name, JACK_DEFAULT_AUDIO_TYPE,
JackPortIsInput, 0 );
        sprintf ( port_name, "output_%d", i + 1
);
        output_ports[i] = jack_port_register (
client, port_name, JACK_DEFAULT_AUDIO_TYPE,
JackPortIsOutput, 0 );
        if ( ( input_ports[i] == NULL ) || (
output_ports[i] == NULL ) )
        {
            fprintf ( stderr, "no more JACK
ports available\n" );
            exit ( 1 );
        }
    }

    bzero(buffer, SIZE * sizeof (
```

```
jack_default_audio_sample_t ));
    delay_idx = 0;
    idx = DELAY;

    /* Tell the JACK server that we are ready
to roll.  Our
     * process() callback will start running
now. */

    if ( jack_activate ( client ) )
    {
        fprintf ( stderr, "cannot activate
client" );
        exit ( 1 );
    }

    /* Connect the ports.  You can't do this
before the client is
     * activated, because we can't make
connections to clients
     * that aren't running.  Note the confusing
(but necessary)
     * orientation of the driver backend ports:
playback ports are
     * "input" to the backend, and capture
ports are "output" from
     * it.
```

```
     */

    ports = jack_get_ports ( client, NULL,
NULL, JackPortIsPhysical|JackPortIsOutput );
    if ( ports == NULL )
    {
        fprintf ( stderr, "no physical capture
ports\n" );
        exit ( 1 );
    }

    for ( i = 0; i < 2; i++ )
    if ( jack_connect ( client, ports[i],
jack_port_name ( input_ports[i] ) ) )
        fprintf ( stderr, "cannot connect input
ports\n" );

    free ( ports );

    ports = jack_get_ports ( client, NULL,
NULL, JackPortIsPhysical|JackPortIsInput );
    if ( ports == NULL )
    {
        fprintf ( stderr, "no physical playback
ports\n" );
        exit ( 1 );
    }
```

```
    for ( i = 0; i < 2; i++ )
    if ( jack_connect ( client, jack_port_name
( output_ports[i] ), ports[i] ) )
        fprintf ( stderr, "cannot connect input
ports\n" );

    free ( ports );

    /* install a signal handler to properly
quits jack client */
#ifdef WIN32
    signal ( SIGINT, signal_handler );
    signal ( SIGABRT, signal_handler );
    signal ( SIGTERM, signal_handler );
#else
    signal ( SIGQUIT, signal_handler );
    signal ( SIGTERM, signal_handler );
    signal ( SIGHUP, signal_handler );
    signal ( SIGINT, signal_handler );
#endif

    /* keep running until the transport stops
*/

    while (1)
    {
```

```
#ifdef WIN32
        Sleep ( 1000 );
#else
        sleep ( 1 );
#endif
    }

    jack_client_close ( client );
    exit ( 0 );
}
```

LADISH API

If an application is Jack session aware, then the LADISH GUI tool `gladish` can manage the applicaiton as a Level 1 application. In other words, `gladish` can manage Jack session and LADISH clients equally. In that sense, there is no need to additionally add LADISH awareness to an application unless you prefer the LADISH way of managing sessions.

How to build LADISH-aware apps at Level 1 app examples and also see LADI Session Handler

Non-session management API

See Non Session Management API

Conclusion

This chapter has looked at some of the session management systems. The situation is not particularly satisfactory and there is substantial room for improvement.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

This chapter covers the essentials of programming sampled data using the Java Sound API

Introduction

Java Sound has been around since early days of Java. It deals with both sampled and MIDI data. There are many resources available for Java Sound:

- The Java™ Platform, Standard Edition 7 API Specification is the reference point for all of the standard Java APIs, including `javax.sound.sampled`.
- The Java Tutorials - Trail: Sound give a good overview of both the sampled and MIDI packages.
- The Java Sound Resources: FAQ: Audio Programming answers many questions about Java Sound
- The Sound Group is "comprised of those developers designing, implementing and maintaining the various OpenJDK sound components". There's your hook into finding out more about the ongoing development of Java Sound in the Open Source community.

Key Java Sound classes

- The `AudioSystem` class is the entry point for all sampled audio classes.
- The `AudioFormat` class specifies information about the format, such as sampling rate.
- The `AudioInputStream` class supplies an input stream from the target line of a mixer
- The `Mixer` class represents an audio device
- The `SourceDataLine` class represents an input line to a device
- The `TargetDataLine` class represents an output line from a device

Information about devices

Each device is represented by a `Mixer` object. Ask the `AudioSystem` for a list of these. Each mixer has a set of target (output) lines and source (input lines). Query each mixer about these separately. The program is DeviceInfo.java:

```
import javax.sound.sampled.*;

public class DeviceInfo {

    public static void main(String[] args)
throws Exception {

    Mixer.Info[] minfoSet =
AudioSystem.getMixerInfo();
    System.out.println("Mixers:");
    for (Mixer.Info minfo: minfoSet) {
        System.out.println("   " +
minfo.toString());

        Mixer m = AudioSystem.getMixer(minfo);
        System.out.println("    Mixer: " +
m.toString());
        System.out.println("      Source
lines");
        Line.Info[] slines =
m.getSourceLineInfo();
        for (Line.Info s: slines) {
        System.out.println("        " +
s.toString());
        }
```

```
        Line.Info[] tlines =
m.getTargetLineInfo();
        System.out.println("      Target
lines");
        for (Line.Info t: tlines) {
        System.out.println("        " +
t.toString());
        }
    }
    }
}
```

A part of the output on my system is

```
Mixers:
   PulseAudio Mixer, version 0.02
      Source lines
        interface SourceDataLine supporting 42
audio formats, and buffers of 0 to 1000000
bytes
        interface Clip supporting 42 audio
formats, and buffers of 0 to 1000000 bytes
      Target lines
        interface TargetDataLine supporting 42
audio formats, and buffers of 0 to 1000000
bytes
   default [default], version 1.0.24
      Source lines
        interface SourceDataLine supporting 512
audio formats, and buffers of at least 32 bytes
        interface Clip supporting 512 audio
formats, and buffers of at least 32 bytes
      Target lines
        interface TargetDataLine supporting 512
audio formats, and buffers of at least 32 bytes
   PCH [plughw:0,0], version 1.0.24
      Source lines
        interface SourceDataLine supporting 24
audio formats, and buffers of at least 32 bytes
        interface Clip supporting 24 audio
```

```
formats, and buffers of at least 32 bytes
      Target lines
        interface TargetDataLine supporting 24
audio formats, and buffers of at least 32 bytes
   NVidia [plughw:1,3], version 1.0.24
      Source lines
        interface SourceDataLine supporting 96
audio formats, and buffers of at least 32 bytes
        interface Clip supporting 96 audio
formats, and buffers of at least 32 bytes
      Target lines
   NVidia [plughw:1,7], version 1.0.24
      Source lines
        interface SourceDataLine supporting 96
audio formats, and buffers of at least 32 bytes
        interface Clip supporting 96 audio
formats, and buffers of at least 32 bytes
      Target lines
   NVidia [plughw:1,8], version 1.0.24
      Source lines
        interface SourceDataLine supporting 96
audio formats, and buffers of at least 32 bytes
        interface Clip supporting 96 audio
formats, and buffers of at least 32 bytes
      Target lines
```

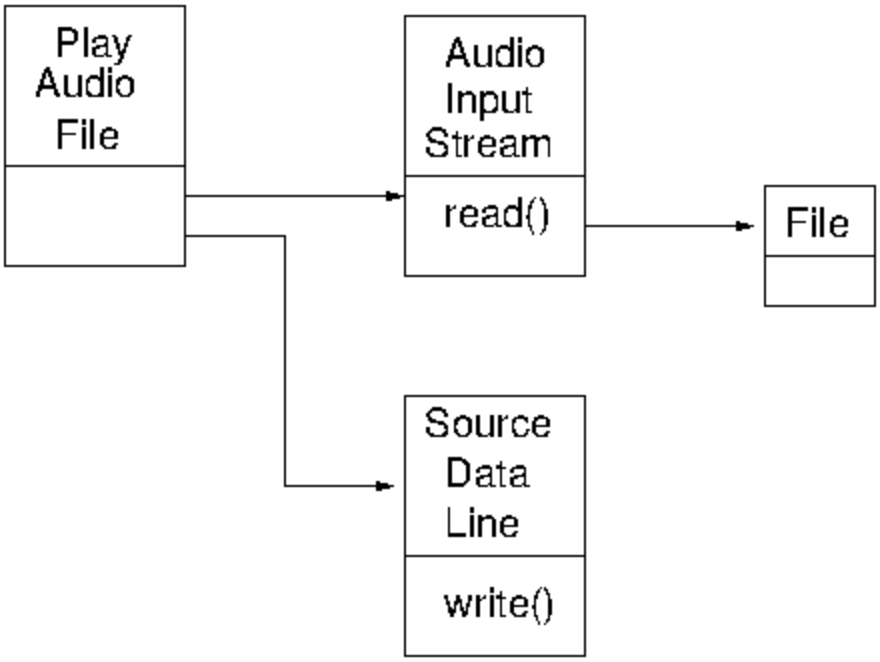
This shows both PulseAudio and ALSA mixers. Further queries could show what the supported formats are, for example.

Playing audio from a file

To play from a file, appropriate objects must be created for reading from the file and for writing to the output device. These are

- An `AudioInputStream` is requested from the `AudioSystem` . It is created with the filename as parameter.
- A source data line is created for the output. The nomenclature may be confusing: the program produces *output* , but this is *input* to the dataline. So the dataline must be a source for the output device. The creation of a dataline is a multi-step process:
- First create a `AudioFormat` object to specify parameters for the dataline
- Create a `DataLine.Info` for a source dataline with the audion format
- Request a source dataline from the `AudioSystem` which will handle the `DataLine.Info`

Following these steps, data can then be read from the input stream and written to the dataline. The UML class diagram for the relevant classes is


Play
Audio
File
Audio
Input
Stream
read()
File
Source
Data
Line
write()

```
import java.io.File;
import java.io.IOException;

import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.FloatControl;
import
javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.SourceDataLine;

public class PlayAudioFile {
    /** Plays audio from given file names. */
    public static void main(String [] args) {
    // Check for given sound file names.
    if (args.length < 1) {
        System.out.println("Usage: java Play
<sound file names>*");
        System.exit(0);
    }

    // Process arguments.
    for (int i = 0; i < args.length; i++)
        playAudioFile(args[i]);
```

```
    // Must exit explicitly since audio creates
non-daemon threads.
    System.exit(0);
    } // main

    public static void playAudioFile(String
fileName) {
    File soundFile = new File(fileName);

    try {
        // Create a stream from the given file.
        // Throws IOException or
UnsupportedAudioFileException
        AudioInputStream audioInputStream =
AudioSystem.getAudioInputStream(soundFile);
        //
AudioSystem.getAudioInputStream(inputStream);
// alternate audio stream from inputstream
        playAudioStream(audioInputStream);
    } catch (Exception e) {
        System.out.println("Problem with file "
+ fileName + ":");
        e.printStackTrace();
    }
    } // playAudioFile

    /** Plays audio from the given audio input
```

```
stream. */
    public static void
playAudioStream(AudioInputStream
audioInputStream) {
    // Audio format provides information like
sample rate, size, channels.
    AudioFormat audioFormat =
audioInputStream.getFormat();
    System.out.println("Play input audio
format=" + audioFormat);

    // Open a data line to play our type of
sampled audio.
    // Use SourceDataLine for play and
TargetDataLine for record.
    DataLine.Info info = new
DataLine.Info(SourceDataLine.class,
audioFormat);
    if (!AudioSystem.isLineSupported(info)) {

System.out.println("Play.playAudioStream does
not handle this type of audio on this
system.");
        return;
    }

    try {
```

```
        // Create a SourceDataLine for play
back (throws LineUnavailableException).
        SourceDataLine dataLine =
(SourceDataLine) AudioSystem.getLine(info);
        // System.out.println("SourceDataLine
class=" + dataLine.getClass());

        // The line acquires system resources
(throws LineAvailableException).
        dataLine.open(audioFormat);

        // Adjust the volume on the output
line.

if(dataLine.isControlSupported(FloatControl.Type.
 {
        FloatControl volume = (FloatControl)
dataLine.getControl(FloatControl.Type.MASTER_GAIN

        volume.setValue(6.0F);
        }

        // Allows the line to move data in and
out to a port.
        dataLine.start();

        // Create a buffer for moving data from
```

```
the audio stream to the line.
        int bufferSize = (int)
audioFormat.getSampleRate() *
audioFormat.getFrameSize();
        byte [] buffer = new byte[ bufferSize
];

        // Move the data until done or there is
an error.
        try {
        int bytesRead = 0;
        while (bytesRead >= 0) {
            bytesRead =
audioInputStream.read(buffer, 0,
buffer.length);
            if (bytesRead >= 0) {
            //
System.out.println("Play.playAudioStream bytes
read=" + bytesRead +
            // ", frame size=" +
audioFormat.getFrameSize() + ", frames read=" +
bytesRead / audioFormat.getFrameSize());
            // Odd sized sounds throw an
exception if we don't write the same amount.
            int framesWritten =
dataLine.write(buffer, 0, bytesRead);
            }
```

```
        } // while
        } catch (IOException e) {
        e.printStackTrace();
        }


System.out.println("Play.playAudioStream
draining line.");
        // Continues data line I/O until its
buffer is drained.
        dataLine.drain();


System.out.println("Play.playAudioStream
closing line.");
        // Closes the data line, freeing any
resources such as the audio device.
        dataLine.close();
    } catch (LineUnavailableException e) {
        e.printStackTrace();
    }
    } // playAudioStream
} // PlayAudioFile
```

Recording audio to a file

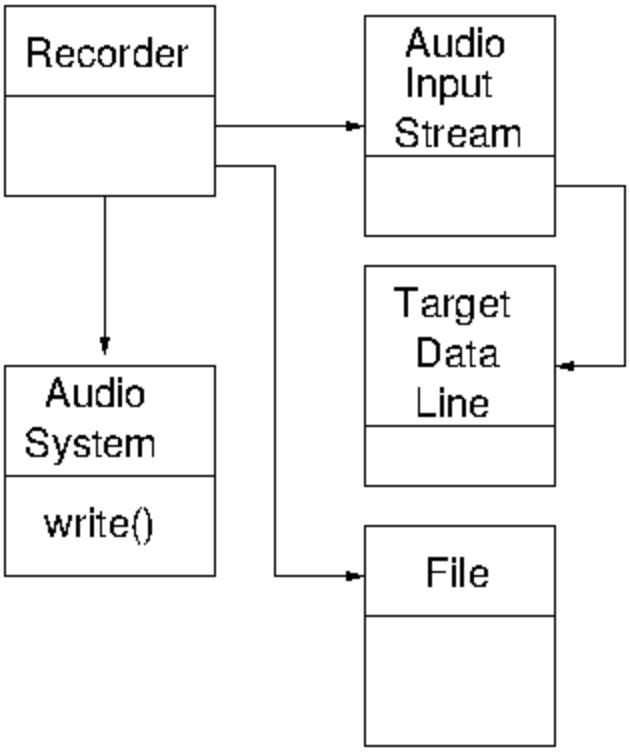
Most of the work to do this is in preparation of an audio input stream. Once that is done the method `write` of `AudioSystem` will copy input from the audio input stream to the output file.

To prepare the audio input stream:

- Create an `AudioFormat` object describing the parameters of the input
- The microphone *produces* audio. So it needs a `TargetDataLine`. So create a `DataLine.Info` for a target dataline.
- Ask the `AudioSystem` for a line satisfying the information
- Wrap the line in an `AudioInputStream`

The output is just a Java `File`.

Then use the `AudioSystem` function `write()` to copy the stream to the file. The UML class diagram is



The program is:

```
import javax.sound.sampled.*;
import java.io.File;

/**
 * Sample audio recorder
 */
public class Recorder extends Thread
{
    /**
     * The TargetDataLine that we’ll use to
read data from
     */
    private TargetDataLine line;

    /**
     * The audio format type that we’ll encode
the audio data with
     */
    private AudioFileFormat.Type targetType =
AudioFileFormat.Type.WAVE;

    /**
     * The AudioInputStream that we’ll read the
audio data from
     */
    private AudioInputStream inputStream;
```

```
    /**
     * The file that we're going to write data
out to
     */
    private File file;

    /**
     * Creates a new Audio Recorder
     */
    public Recorder(String outputFilename)
    {
    try {
        // Create an AudioFormat that specifies
how the recording will be performed
        // In this example we'll 44.1Khz, 16-
bit, stereo
        AudioFormat audioFormat = new
AudioFormat(

AudioFormat.Encoding.PCM_SIGNED,    // Encoding
technique
                                44100.0F,
// Sample Rate
                                16,
// Number of bits in each channel
                                2,
```

```
// Number of channels (2=stereo)
                                4,
// Number of bytes in each frame
                                44100.0F,
// Number of frames per second
                                false);
// Big-endian (true) or little-
        // endian (false)

        // Create our TargetDataLine that will
be used to read audio data by first
        // creating a DataLine instance for our
audio format type
        DataLine.Info info = new
DataLine.Info(TargetDataLine.class,
audioFormat);

        // Next we ask the AudioSystem to
retrieve a line that matches the
        // DataLine Info
        this.line =
(TargetDataLine)AudioSystem.getLine(info);

        // Open the TargetDataLine with the
specified format
        this.line.open(audioFormat);
```

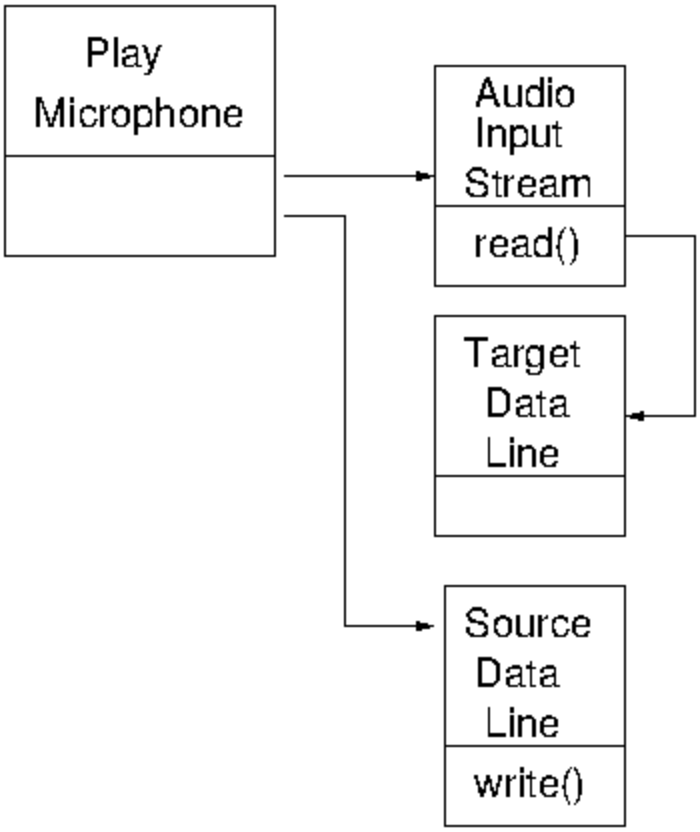
```
        // Create an AudioInputStream that we
can use to read from the line
        this.inputStream = new
AudioInputStream(this.line);

        // Create the output file
        this.file = new File(outputFilename);
    }
    catch(Exception e) {
        e.printStackTrace();
    }
    }

    public void startRecording() {
        // Start the TargetDataLine
        this.line.start();

        // Start our thread
        start();
    }

    public void stopRecording() {
        // Stop and close the TargetDataLine
        this.line.stop();
        this.line.close();
    }
```

```
    public void run() {
        try {
        // Ask the AudioSystem class to write
audio data from the audio input stream
        // to our file in the specified data
type (PCM 44.1Khz, 16-bit, stereo)
        AudioSystem.write(this.inputStream,
this.targetType, this.file);
    }
        catch(Exception e) {
        e.printStackTrace();
    }
    }

    public static void main(String[] args) {
        if (args.length == 0) {
        System.out.println("Usage: Recorder
<filename>");
        System.exit(0);
    }

        try {
        // Create a recorder that writes WAVE
data to the specified filename
        Recorder r = new Recorder(args[0]);
        System.out.println("Press ENTER to
start recording");
```

```
        System.in.read();

        // Start the recorder
        r.startRecording();

        System.out.println("Press ENTER to stop
recording");
        System.in.read();

        // Stop the recorder
        r.stopRecording();

        System.out.println("Recording
complete");
    }
        catch(Exception e) {
        e.printStackTrace();
    }
    }

}
```

Play microphone to speaker

This is a combination of the previous two programs. An `AudioInputStream` is prepared for reading from the microphone. A `SourceDataLine` is prepared for writing to the speaker. The data is copied from the first to the second by reading from the audio input stream and writing to the source data line. The UML class diagram is



The program is:

```
import java.io.File;
import java.io.IOException;


import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.Line;
import javax.sound.sampled.Line.Info;
import javax.sound.sampled.TargetDataLine;
import javax.sound.sampled.FloatControl;
import
javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.SourceDataLine;

public class PlayMicrophone {
    private static final int FRAMES_PER_BUFFER
= 1024;

    public static void main(String[] args)
throws Exception {
    /*
     *    We check that there is exactely one
command-line
     *    argument. If not, we display the
```

```
usage message and
     *    exit.
     */

    new PlayMicrophone().playAudio();
    }



    private void out(String strMessage)
    {
    System.out.println(strMessage);
    }

  //This method creates and returns an
  // AudioFormat object for a given set of
format
  // parameters.  If these parameters don't
work
  // well for you, try some of the other
  // allowable parameter values, which are
shown
  // in comments following the declarations.
  private  AudioFormat getAudioFormat(){
    float sampleRate = 44100.0F;
    //8000,11025,16000,22050,44100
    int sampleSizeInBits = 16;
```

```
    //8,16
    int channels = 1;
    //1,2
    boolean signed = true;
    //true,false
    boolean bigEndian = false;
    //true,false
    return new AudioFormat(sampleRate,
                           sampleSizeInBits,
                           channels,
                           signed,
                           bigEndian);
  }//end getAudioFormat

    public void playAudio() throws Exception {
    AudioFormat audioFormat;
    TargetDataLine targetDataLine;

    audioFormat = getAudioFormat();
    DataLine.Info dataLineInfo =
        new DataLine.Info(
                  TargetDataLine.class,
                  audioFormat);
    targetDataLine = (TargetDataLine)
        AudioSystem.getLine(dataLineInfo);

    /*
```

```
    Line.Info lines[] =
AudioSystem.getTargetLineInfo(dataLineInfo);
    for (int n = 0; n < lines.length; n++) {
        System.out.println("Target " +
lines[n].toString() + " " +
lines[n].getLineClass());
    }
    targetDataLine = (TargetDataLine)
        AudioSystem.getLine(lines[0]);
    */

    targetDataLine.open(audioFormat,
                audioFormat.getFrameSize() *
FRAMES_PER_BUFFER);
    targetDataLine.start();

    playAudioStream(new
AudioInputStream(targetDataLine));

    /*
    File soundFile = new File( fileName );

    try {
        // Create a stream from the given file.
        // Throws IOException or
UnsupportedAudioFileException
        AudioInputStream audioInputStream =
```

```
AudioSystem.getAudioInputStream( soundFile );
        // AudioSystem.getAudioInputStream(
inputStream ); // alternate audio stream from
inputstream
        playAudioStream( audioInputStream );
    } catch ( Exception e ) {
        System.out.println( "Problem with file
" + fileName + ":" );
        e.printStackTrace();
    }
    */
    } // playAudioFile

    /** Plays audio from the given audio input
stream. */
    public void playAudioStream(
AudioInputStream audioInputStream ) {
    // Audio format provides information like
sample rate, size, channels.
    AudioFormat audioFormat =
audioInputStream.getFormat();
    System.out.println( "Play input audio
format=" + audioFormat );

    // Open a data line to play our type of
sampled audio.
    // Use SourceDataLine for play and
```

```
TargetDataLine for record.
    DataLine.Info info = new DataLine.Info(
SourceDataLine.class, audioFormat );

    Line.Info lines[] =
AudioSystem.getSourceLineInfo(info);
    for (int n = 0; n < lines.length; n++) {
        System.out.println("Source " +
lines[n].toString() + " " +
lines[n].getLineClass());
    }

    if ( !AudioSystem.isLineSupported( info ) )
{
        System.out.println(
"Play.playAudioStream does not handle this type
of audio on this system." );
        return;
    }

    try {
        // Create a SourceDataLine for play
back (throws LineUnavailableException).
        SourceDataLine dataLine =
(SourceDataLine) AudioSystem.getLine( info );
        // System.out.println( "SourceDataLine
class=" + dataLine.getClass() );
```

```
        // The line acquires system resources
(throws LineAvailableException).
        dataLine.open( audioFormat,
               audioFormat.getFrameSize() *
FRAMES_PER_BUFFER);

        // Adjust the volume on the output
line.
        if( dataLine.isControlSupported(
FloatControl.Type.MASTER_GAIN ) ) {
        FloatControl volume = (FloatControl)
dataLine.getControl(
FloatControl.Type.MASTER_GAIN );
        volume.setValue( 6.0F );
        }

        // Allows the line to move data in and
out to a port.
        dataLine.start();

        // Create a buffer for moving data from
the audio stream to the line.
        int bufferSize = (int)
audioFormat.getSampleRate() *
audioFormat.getFrameSize();
        bufferSize =
```

```
audioFormat.getFrameSize() * FRAMES_PER_BUFFER;
        System.out.println("Buffer size: " +
bufferSize);
        byte [] buffer = new byte[ bufferSize
];

        // Move the data until done or there is
an error.
        try {
        int bytesRead = 0;
        while ( bytesRead >= 0 ) {
            bytesRead = audioInputStream.read(
buffer, 0, buffer.length );
            if ( bytesRead >= 0 ) {
            System.out.println(
"Play.playAudioStream bytes read=" + bytesRead
+
            ", frame size=" +
audioFormat.getFrameSize() + ", frames read=" +
bytesRead / audioFormat.getFrameSize() );
            // Odd sized sounds throw an
exception if we don't write the same amount.
            int framesWritten = dataLine.write(
buffer, 0, bytesRead );
            }
        } // while
        } catch ( IOException e ) {
```

```
        e.printStackTrace();
        }

        System.out.println(
"Play.playAudioStream draining line." );
        // Continues data line I/O until its
buffer is drained.
        dataLine.drain();

        System.out.println(
"Play.playAudioStream closing line." );
        // Closes the data line, freeing any
resources such as the audio device.
        dataLine.close();
    } catch ( LineUnavailableException e ) {
        e.printStackTrace();
    }
    } // playAudioStream

}
```

Where does javaSound get its devices from?

The first program in this chapter showed a list of mixer devices and their attributes. How does Java get this information? In this section we look at JDK 1.7 - OpenJDK will probably be similar. You will need the Java source from Oracle to track through this. Alternatively, move on...

The file `jre/lib/resources.jar` contains a list of resources used by the JRE runtime. This is a zip file, and contains the file `META-INF/services/javax.sound.sampled.spi.MixerProvider`. On my system the contents of this file are

```
# last mixer is default mixer
com.sun.media.sound.PortMixerProvider
com.sun.media.sound.DirectAudioDeviceProvider
```

The class `com.sun.media.sound.PortMixerProvider` is in the file `java/media/src/share/native/com/sun/media/sound/PortMixerProvider.java` on my system. It extends

`MixerProvider` and implements methods such as `Mixer.Info[] getMixerInfo` . This class stores the device information.

The bulk of the work done done by this class is actually performed by native methods in the C file `java/media/src/share/native/com/sun/media/sound/PortMixerProvider.c` which implements the two methods `nGetNumDevices` and `nNewPortMixerInfo` used by the `PortMixerProvider` class. Unfortunately, there's ot much joy to be found in this C file, as it just makes calls to the C functions `PORT_GetPortMixerCount` and `PORT_GetPortMixerDescription` .

There are three files containing these functions

```
java/media/src/windows/native/com/sun/media/sound

java/media/src/solaris/native/com/sun/media/sound

java/media/src/solaris/native/com/sun/media/sound
```

In the file `PLATFORM_API_LinuxOS_ALSA_Ports.c` you will see the function calls to ALSA as described in the ALSA chapter . These calls fill in information about the ALSA devices for use by JavaSound.

Conclusion

The Java Sound API is well documented. We have shown four simple programs here, but more complex ones are possible. The link to the underlying sound system is briefly discussed.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

This chapter only includes pointers to material on GStreamer at present

Resources

- Introducing GStreamer by Marcelo Lira
- GStreamer by Gnome Project
- Multipurpose multimedia processing with GStreamer by Maciej Katafiasz
- GStreamer FAQ
- A playbin is a highlevel, automatic, audio and video player
- GStreamer Transcoding
- GStreamer plugins reference

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

Fromlibao Overview"Libao is designed to make it easy to do simple audio output using various audio devices and libraries. For this reason, complex audio control features are missing and will probably never be added. However, if you just want to be able to open whatever audio device is available and play sound, libao should be just fine. "

Resources

- libao Documentation

Examples

```
/*
 *
 * ao_example.c
 *
 *     Written by Stan Seibert - July 2001
 *
 * Legal Terms:
 *
 *     This source file is released into the
public domain.  It is
 *     distributed without any warranty;
without even the implied
 *     warranty * of merchantability or fitness
for a particular
 *     purpose.
 *
 * Function:
 *
 *     This program opens the default driver
and plays a 440 Hz tone for
 *     one second.
 *
 * Compilation command line (for Linux
systems):
 *
 *     gcc -lao -ldl -lm -o ao_example
```

```c
ao_example.c
 *
 */

#include <stdio.h>
#include <ao/ao.h>
#include <math.h>

#define BUF_SIZE 4096

int main(int argc, char **argv)
{
    ao_device *device;
    ao_sample_format format;
    int default_driver;
    char *buffer;
    int buf_size;
    int sample;
    float freq = 440.0;
    int i;

    /* -- Initialize -- */

    fprintf(stderr, "libao example program\n");

    ao_initialize();
```

```
    /* -- Setup for default driver -- */

    default_driver = ao_default_driver_id();

        memset(&format, 0, sizeof(format));
    format.bits = 16;
    format.channels = 2;
    format.rate = 44100;
    format.byte_format = AO_FMT_LITTLE;

    /* -- Open driver -- */
    device = ao_open_live(default_driver,
&format, NULL /* no options */);
    if (device == NULL) {
        fprintf(stderr, "Error opening
device.\n");
        return 1;
    }

    /* -- Play some stuff -- */
    buf_size = format.bits/8 * format.channels
* format.rate;
    buffer = calloc(buf_size,
            sizeof(char));

    for (i = 0; i < format.rate; i++) {
        sample = (int)(0.75 * 32768.0 *
```

```
            sin(2 * M_PI * freq * ((float)
i/format.rate)));

        /* Put the same stuff in left and right
channel */
        buffer[4*i] = buffer[4*i+2] = sample &
0xff;
        buffer[4*i+1] = buffer[4*i+3] = (sample
>> 8) & 0xff;
    }
    ao_play(device, buffer, buf_size);

    /* -- Close and shutdown -- */
    ao_close(device);

    ao_shutdown();

  return (0);
}
```

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

FromA FFmpeg Tutorial For Beginners"FFmpeg is a complete, cross platform command line tool capable of recording, converting and streaming digital audio and video in various formats. It can be used to do most of our multimedia tasks quickly and easily say, audio compression, audio/video format conversion, extract images from a video and a lot more."FFmpeg consists of a set of command line tools and a set of libraries that can be used especially for transforming audio (and video) files from one format to another. It can work on both containers and on codecs. It is not designed for playing or recording audio, more for a general-purpose conversion tool.The version of FFmpeg discussed in this book is v0.8 as in the Fedora 16 repositories. The lastest version is1.1.3and several things have changed since then.

Resources

- FFmpeg home page
- Using libavformat and libavcodec
- A FFmpeg Tutorial For Beginners by shredder12 describes ffmpeg command line tool
- ffmpeg Documentation is extensive documentation for the `ffmpeg` command
- An ffmpeg and SDL Tutorial with updated code here

FFmpeg command line tools

The principal FFmpeg tool is `ffmpeg` itself. The simplest use is as a converter from one format to another as in

```
ffmpeg -i file.ogg file.mp3
```

which will convert an Ogg container of Vorbis codec data to an MPEG container of MP2 codec data.

Internally, `ffmpeg` uses a pipeline of modules

```
_______                 _______________
__________                _______________
________
|       |               |              |
|         |               |              |
|        |
| input |  demuxer   | encoded data |   decoder
| decoded |  encoder   | encoded data |  muxer
| output |
| file  | ---------> | packets      |  --------
-> | frames  | ---------> | packets      | ----
---> | file   |
|_______|               |_______________|
|__________|               |_______________|
|________|
```

(Figure from ffmpeg Documentation .) The muxer/demuxer, decoder/encoder can all be set using options if the defaults are not appropriate.

Other commands are

- `ffprobe` to give information about a file
- `ffplay` a simple media player
- `ffserver` a media server

libavformat or libavdecode

This example is taken from blinking bill and plays almost any file (Ogg Vorbis, AVI, MP3, etc).

```
/**
 * Program copyright blinking blip, 2011
 * Downloaded from
 *
http://blinkingblip.wordpress.com/2011/10/08/deco
and-playing-an-audio-stream-using-libavcodec-
libavformat-and-libao/
 */

#include <stdio.h>
#include <libavcodec/avcodec.h>
#include <libavformat/avformat.h>
#include <libavutil/avutil.h>
#include <ao/ao.h>

void die(const char* message) {
    fprintf(stderr, "%s\n", message);
    exit(1);
}

int main(int argc, char* argv[]) {
    if (argc < 2) {
        die("Usage: play file");
    }

    const char* input_filename = argv[1];
```

```
    // This call is necessarily done once in
your app to initialize
    // libavformat to register all the muxers,
demuxers and protocols.
    av_register_all();

    // A media container
    AVFormatContext* container = 0;

    if (avformat_open_input(&container,
input_filename, NULL, NULL) < 0) {
        die("Could not open file");
    }

    if (av_find_stream_info(container) < 0) {
        die("Could not find file info");
    }

    int stream_id = -1;

    // To find the first audio stream. This
process may not be necessary
    // if you can gurarantee that the container
contains only the desired
    // audio stream
    int i;
```

```
    for (i = 0; i < container->nb_streams; i++)
{
        if (container->streams[i]->codec-
>codec_type == AVMEDIA_TYPE_AUDIO) {
            stream_id = i;
            break;
        }
    }

    if (stream_id == -1) {
        die("Could not find an audio stream");
    }

    /*
    // Extract some metadata
    AVDictionary* metadata = container-
>metadata;

    const char* artist = av_dict_get(metadata,
"artist", NULL, 0)->value;
    const char* title = av_dict_get(metadata,
"title", NULL, 0)->value;

    fprintf(stdout, "Playing: %s - %s\n",
artist, title);
    */
```

```
    // Find the apropriate codec and open it
    AVCodecContext* codec_context = container-
>streams[stream_id]->codec;

    AVCodec* codec =
avcodec_find_decoder(codec_context->codec_id);

    if (!avcodec_open(codec_context, codec) <
0) {
        die("Could not find open the needed
codec");
    }

    // To initalize libao for playback
    ao_initialize();

    int driver = ao_default_driver_id();

    // The format of the decoded PCM samples
    ao_sample_format sample_format;
    sample_format.bits = 16;
    sample_format.channels = 2;
    sample_format.rate = 44100;
    sample_format.byte_format = AO_FMT_NATIVE;
    sample_format.matrix = 0;

    ao_device* device = ao_open_live(driver,
```

```
&sample_format, NULL);

    AVPacket packet;
    int buffer_size =
AVCODEC_MAX_AUDIO_FRAME_SIZE;
    int8_t
buffer[AVCODEC_MAX_AUDIO_FRAME_SIZE];

    while (1) {

        buffer_size =
AVCODEC_MAX_AUDIO_FRAME_SIZE;

        // Read one packet into `packet`
        if (av_read_frame(container, &packet) <
0) {
            break;  // End of stream. Done
decoding.
        }

        // Decodes from `packet` into the
buffer
        if
(avcodec_decode_audio3(codec_context,
(int16_t*)buffer, &buffer_size, &packet) < 1) {
            break;  // Error in decoding
        }
```

```
        // Send the buffer contents to the
audio device
        ao_play(device, (char*)buffer,
buffer_size);
    }

    av_close_input_file(container);

    ao_shutdown();

    fprintf(stdout, "Done playing.
Exiting...");

    return 0;
}
```

The example reads frames from a container file, decodes them and then passes the PCM data to `libao` for playing. It could hardly be simpler!

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

OpenMAX and OpenSL are open standards from the Khronos group that are designed for low capability devices. Vendors of cards are expected to produce implementations. There is little by way of general Linux implementations, but Broadcom have implemented one of the specifications (OpenMAX IL), and their chip is used in the Raspberry Pi. Other specifications (OpenMAX AL and OpenSL ES) are expected to be implemented in Android devices, accessible by the Native Development Kit (NDK).

Resources

- The OpenMAX Integration Layer standard - eLinux
- Khronos home page gives the specifications for free download - they are quite well done and readable.
- LIM OpenMAX Implementation is a Linux implementation. Download of 1.0 is okay from lim-omx-1.0.tar.gz .
- OpenMAX IL Bellagio package Source, DEB package and RPM available
- Texas Instruments OpenMax Development Guide
- OpenMAX (Open Media Acceleration)

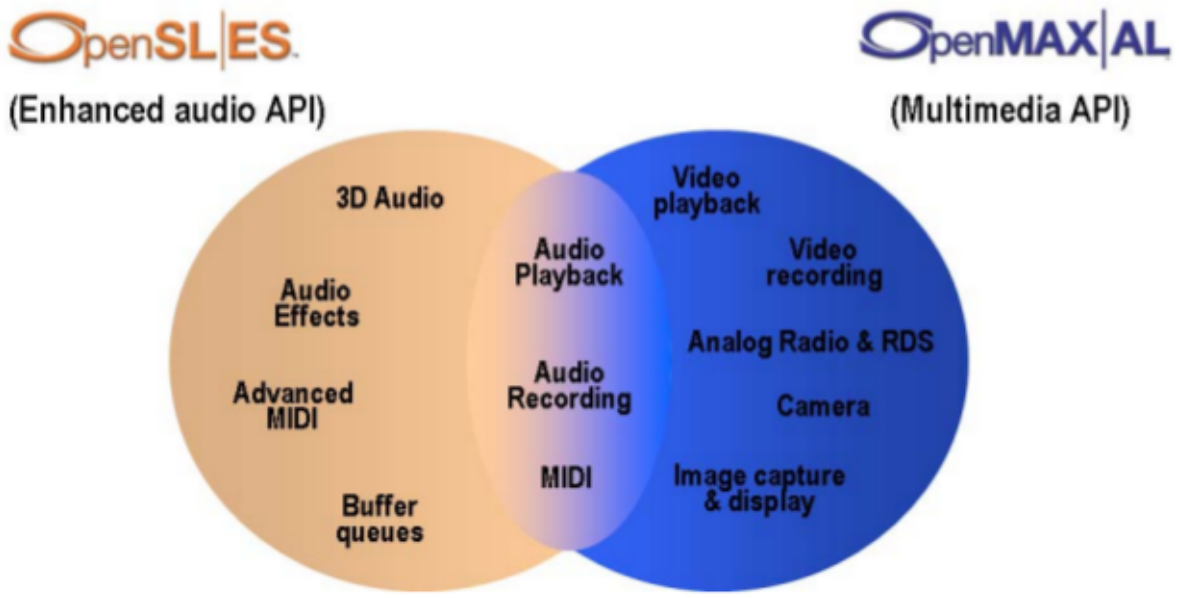
Quotes

- By jamesh OpenMAX is a complete and utter nightmare to use btw
- By dom I have written a fair bit of openmax client code and find it very hard. You have to get an awful lot right before you get anything useful out. Just lots of OMX_ErrorInvalidState, and OMX_ErrorBadParameter messages if you are lucky. Nothing happening at all if you are not...
- By Twinkletoes I'm from a DirectShow background, and I thought that was badly documented... then I met OpenMax. Lots of ppts saying lovely things about it but no documentation or code examples I can find.

OpenMAX AL and OpenSL ES

"The Khronos Group is a not for profit industry consortium creating open standards for the authoring and acceleration of parallel computing, graphics, dynamic media, computer vision and sensor processing on a wide variety of platforms and devices". For this book, the relevant standards that they produce are OpenSL (not to be confused with OpenGL - a graphics specification) and OpenMAX.

OpenSL and OpenMAX are intended for slightly different uses, but have an overlap of functionality. From the OpenSL ES and OpenMAX Al specifications this is shown by the following figure:



OpenSL is intended for complex audio systems while OpenMAX is intended for general multi-media systems. Either can be used separately, or if needed both can be used - in the overlapping area the functionality is the same (even though the function calls are distinct).

Each of these is further divided into *profiles* . For OpenSL the profiles are

- Phone
- Music
- Game

with optional functionality in

- Audio recording
- Haptics
- Macroscopic behavior in 3D audio

A full chart of features in each profile is given in the specification.

OpenMAX has profiles

- Media playback
- Media recording
- Effects and controls

with optional functionality in

- MIDI
- Analog radio
- LED array
- Vibration device

A full chart of features in each profile is given in the specification.

Just to complicate things a little, OpenSL only has one specification, OpenSL ES (no, I don't know what ES stands for). On the other hand, OpenMAX has two specified layers, OpenMAX IL (interface layer) and OpenMAX AL (application layer). The APIs for OpenSL ES and OpenMAX AL mirror each other in the common area. The API for OpenMAX IL is at a considerably lower level than the API for OpenMAX AL, and is discussed in a later section.

The specifications give some examples on how the different profiles might be mixed:

- Example 1: an audio-only application will have no need for video and image functionality and therefore would likely pick one of the OpenSL ES profiles, depending on the use cases of interest.
- Example 2: a media playback and recording application would use the OpenMAX AL 1.0 Media Player/Recorder profile.
- Example 3: An advanced multimedia/game application that needs audio/video/image playback and recording as well as advanced audio features like 3D audio and effects would use both the Media Player/Recorder profile of OpenMAX AL 1.0 and the Game profile of OpenSL ES 1.0.

Linux

It is expected that OpenMAX and OpenSL will find their major uses in situations where the CPU and memory are limited, but where there is the possibility of hardware acceleration of video and audio by e.g. a GPU. Consequently, the situation for general Linux systems is poor. Bellagio has an implementation of OpenMAX IL. LIM has an implementation of both OpenMAX AL and IL.

I have not been able to find any OpenSL ES implementations for general Linux.

Raspberry Pi

The Raspberry Pi supports OpenMAX IL from the Broadcom card included. If you download the firmware master from the Raspberry Pi download site, a number of examples are included such as firmware-master/documentation/ilcomponents/audio_mixer.html. These are not "pure" OpenMAX IL: they make use of a Broadcom convenience package and also use some vendor extension APIs beyond the OpenMAX IL standard.

Android

You write Android apps in Java. But like most Java systems, code written in C or C++ can be included as "native" methods. The native code is linked by a framework called JNI (Java Native Interface). For the Android, Google have standardised a number of libraries in the NDK (Native Development Kit). This includes both OpenSL ES and OpenMAX AL. Informaton about these is included in the Android NDK .

Documents in the NDK such as android-ndk-r8d/docs/opensles/index.html give good overviews of using these, although using the JNI to build applications is not for the faint of heart. In addition, the documentation inlcudes comments like "In particular, use of OpenSL ES does not guarantee a lower audio latency, higher scheduling priority, etc. than what the platform generally provides". From Android 4.2 on, lower latency will be possible using OpenSL ES. See the Performance section in opensles/index.html

See android-ndk-r8d/samples/native-audio for an example using Java + JNI + OpenSL ES.

OpenSL ES examples

These examples are taken from the appendices of the OpenSL ES specification. They have not been tested.

The programming model for OpenSL ES is that of *interfaces* and *objects* . The programming *language* is C, so one of the common mechanisms for writing O/O style code in C is used (see e.g. Object-Oriented Programming In C). Interfaces are represented by structures containing function pointers for methods. An object which implements the interface is of the interface type, with the methods as particular functions. Calling a method involves calling the appropriate function in the structure. The "this" object is always passed in as the first parameter of the function as in

```
(*object) -> method(object, ...)
```

Every object has to be "realised" before it can be used.

There does not seem to be any inheritance model, but then that isn't necessary here.

Note that every function returns a value that can be tested for sucess or fail. Objects that are created are passed as address parameters for filling. The application is responsible for freeing any memory allocated during this by calling `Destroy()` on the object.

An OpenSL (or OpenMAX AL) application is started by creating an "engine" using `slCreateEngine()` . This is the only global function in the library. From then on, objects are obtained from the engine or other objects by a call to `GetInterface()` or `CreateXYZ` on the engine or object. For example,

```
(*engine) -> CreateOutputMix(engine,
&output_mix...);
(*output_mix) -> GetInterface(output_mix,
SL_IID_VOLUME, &volume, ...);
```

Play audio from a buffer

The model for playing audio is not very complex, just verbose. It consists of

- Initialise the system
- Prepare a data source

- Prepare an output sink
- Set up a callback function to be called when the player needs more data
- Create and start a playback engine
- Clean up

Initialise the system

```
SLresult res;
    SLObjectItf sl;
    SLEngineOption EngineOption[] = {
    (SLuint32) SL_ENGINEOPTION_THREADSAFE,
    (SLuint32) SL_BOOLEAN_TRUE};
    res = slCreateEngine( &sl, 1, EngineOption,
0, NULL, NULL);
    CheckErr(res);
    /* Realizing the SL Engine in synchronous
mode. */
    res = (*sl)->Realize(sl, SL_BOOLEAN_FALSE);
CheckErr(res);
```

Prepare a data source

```
/* Local storage for Audio data */
    SLint16 pcmData[AUDIO_DATA_STORAGE_SIZE];
    /* put something nito pcmData ... not done
here */

    /* Setup the data source structure for the
buffer queue */
    bufferQueue.locatorType =
SL_DATALOCATOR_BUFFERQUEUE;
    bufferQueue.numBuffers = 4; /* Four buffers
in our buffer queue */
    /* Setup the format of the content in the
buffer queue */
    pcm.formatType = SL_DATAFORMAT_PCM;
    pcm.numChannels = 2;
    pcm.samplesPerSec = SL_SAMPLINGRATE_44_1;
    pcm.bitsPerSample =
SL_PCMSAMPLEFORMAT_FIXED_16;
    pcm.containerSize = 16;
    pcm.channelMask = SL_SPEAKER_FRONT_LEFT |
SL_SPEAKER_FRONT_RIGHT;
    pcm.endianness = SL_BYTEORDER_LITTLEENDIAN;
    audioSource.pFormat    = (void *)&pcm;
    audioSource.pLocator    = (void
*)&bufferQueue;
```

Prepare an output sink

```
for (i=0;i<MAX_NUMBER_INTERFACES;i++)
    {
        required[i] = SL_BOOLEAN_FALSE;
        iidArray[i] = SL_IID_NULL;
    }
    // Create Output Mix object to be used by
player
    res = (*EngineItf)-
>CreateOutputMix(EngineItf, &OutputMix, 1,
                     iidArray, required);
CheckErr(res);
    // Realizing the Output Mix object in
synchronous mode.
    res = (*OutputMix)->Realize(OutputMix,
SL_BOOLEAN_FALSE);
    CheckErr(res);
```

Set up a callback function to be called when the player needs more data

The callback in this case uses a data structure to hold all of the data and pointers to the start and current position of the data. This is for this application, and is not a part of OpenSL

```
/* Structure for passing information to
callback function */
typedef struct CallbackCntxt_ {
    SLPlayItf playItf;
    SLint16* pDataBase;
    // Base adress of local audio data storage
    SLint16* pData;
    // Current adress of local audio data
storage
    SLuint32 size;
} CallbackCntxt;
```

Then we can set up the output sink and callback data

```
/* Setup the data sink structure */
    locator_outputmix.locatorType =
SL_DATALOCATOR_OUTPUTMIX;
    locator_outputmix.outputMix = OutputMix;
    audioSink.pLocator = (void
*)&locator_outputmix;
    audioSink.pFormat = NULL;
    /* Initialize the context for Buffer queue
callbacks */
    cntxt.pDataBase = (void*)&pcmData;
    cntxt.pData = cntxt.pDataBase;
    cntxt.size = sizeof(pcmData);
```

Create and start a playback engine

```
/* Set arrays required[] and iidArray[] for
SEEK interface
       (PlayItf is implicit) */
    required[0] =    SL_BOOLEAN_TRUE;
    iidArray[0] =    SL_IID_BUFFERQUEUE;
    /* Create the music player */
    res = (*EngineItf)-
>CreateAudioPlayer(EngineItf, &player,
                       &audioSource, &audioSink,
1, iidArray, required); CheckErr(res);
    /* Realizing the player in synchronous
mode. */
    res = (*player)->Realize(player,
SL_BOOLEAN_FALSE); CheckErr(res);
    /* Get seek and play interfaces */
    res = (*player)->GetInterface(player,
SL_IID_PLAY, (void*)&playItf);
    CheckErr(res);
    res = (*player)->GetInterface(player,
SL_IID_BUFFERQUEUE,
                  (void*)&bufferQueueItf);
CheckErr(res);
    /* Setup to receive buffer queue event
callbacks */
    res = (*bufferQueueItf)-
>RegisterCallback(bufferQueueItf,
```

```
                                BufferQueueCallback,
NULL); CheckErr(res);
    /* Enqueue a few buffers to get the ball
rolling */
    res = (*bufferQueueItf)-
>Enqueue(bufferQueueItf, cntxt.pData,
                        2 *
AUDIO_DATA_BUFFER_SIZE); /* Size given in
bytes. */
    CheckErr(res);
    cntxt.pData += AUDIO_DATA_BUFFER_SIZE;
    res = (*bufferQueueItf)-
>Enqueue(bufferQueueItf, cntxt.pData,
                        2 *
AUDIO_DATA_BUFFER_SIZE); /* Size given in
bytes. */
    CheckErr(res);
    cntxt.pData += AUDIO_DATA_BUFFER_SIZE;
    res = (*bufferQueueItf)-
>Enqueue(bufferQueueItf, cntxt.pData,
                        2 *
AUDIO_DATA_BUFFER_SIZE); /* Size given in
bytes. */
    CheckErr(res);
    cntxt.pData += AUDIO_DATA_BUFFER_SIZE;
    /* Play the PCM samples using a buffer
queue */
```

```
    res = (*playItf)->SetPlayState( playItf,
SL_PLAYSTATE_PLAYING );
    CheckErr(res);
    /* Wait until the PCM data is done playing,
the buffer queue callback
       will continue to queue buffers until the
entire PCM data has been
       played. This is indicated by waiting for
the count member of the
       SLBufferQueueState to go to zero.
    */
    res = (*bufferQueueItf)-
>GetState(bufferQueueItf, &state);
    CheckErr(res);
    while(state.count)
    {
        (*bufferQueueItf)-
>GetState(bufferQueueItf, &state);
    }
```

Clean up

```
/* Make sure player is stopped */
    res = (*playItf)->SetPlayState(playItf,
SL_PLAYSTATE_STOPPED);
    CheckErr(res);
    /* Destroy the player */
    (*player)->Destroy(player);
    /* Destroy Output Mix object */
    (*OutputMix)->Destroy(OutputMix);
```

Full program

The full code is

```
#include <stdio.h>
#include <stdlib.h>
#include "OpenSLES.h"
#define SLEEP(x) /* Client system sleep
function to sleep x milliseconds
            would replace SLEEP macro */
#define MAX_NUMBER_INTERFACES 3
#define MAX_NUMBER_OUTPUT_DEVICES 6
/* Local storage for Audio data in 16 bit words
*/
#define AUDIO_DATA_STORAGE_SIZE 4096
/* Audio data buffer size in 16 bit words. 8
data segments are used in
   this simple example */
#define AUDIO_DATA_BUFFER_SIZE 4096/8
/* Checks for error. If any errors exit the
application! */
void CheckErr( SLresult res )
{
    if ( res != SL_RESULT_SUCCESS )
    {
        // Debug printing to be placed here
        exit(1);
    }
}
/* Structure for passing information to
```

```
callback function */
typedef struct CallbackCntxt_ {
    SLPlayItf playItf;
    SLint16* pDataBase;
    // Base adress of local audio data storage
    SLint16* pData;
    // Current adress of local audio data
storage
    SLuint32 size;
} CallbackCntxt;
/* Local storage for Audio data */
SLint16 pcmData[AUDIO_DATA_STORAGE_SIZE];
/* Callback for Buffer Queue events */
void BufferQueueCallback(
            SLBufferQueueItf queueItf,
            void *pContext)
{
    SLresult res;
    CallbackCntxt *pCntxt =
(CallbackCntxt*)pContext;
    if(pCntxt->pData < (pCntxt->pDataBase +
pCntxt->size))
    {
        res = (*queueItf)->Enqueue(queueItf,
(void*) pCntxt->pData,
                        2 *
AUDIO_DATA_BUFFER_SIZE); /* Size given in
```

```
bytes. */
        CheckErr(res);
        /* Increase data pointer by buffer size
*/
        pCntxt->pData +=
AUDIO_DATA_BUFFER_SIZE;
    }
}
/* Play some music from a buffer queue */
void TestPlayMusicBufferQueue( SLObjectItf sl )
{
    SLEngineItf EngineItf;
    SLint32 numOutputs = 0;
    SLuint32 deviceID = 0;
    SLresult res;
    SLDataSource audioSource;
    SLDataLocator_BufferQueue bufferQueue;
    SLDataFormat_PCM pcm;
    SLDataSink audioSink;
    SLDataLocator_OutputMix locator_outputmix;
    SLObjectItf player;
    SLPlayItf playItf;
    SLBufferQueueItf bufferQueueItf;
    SLBufferQueueState state;
    SLObjectItf OutputMix;
    SLVolumeItf volumeItf;
    int i;
```

```
    SLboolean required[MAX_NUMBER_INTERFACES];
    SLInterfaceID
iidArray[MAX_NUMBER_INTERFACES];
    /* Callback context for the buffer queue
callback function */
    CallbackCntxt cntxt;
    /* Get the SL Engine Interface which is
implicit */
    res = (*sl)->GetInterface(sl,
SL_IID_ENGINE, (void*)&EngineItf);
    CheckErr(res);
    /* Initialize arrays required[] and
iidArray[] */
    for (i=0;i<MAX_NUMBER_INTERFACES;i++)
    {
        required[i] = SL_BOOLEAN_FALSE;
        iidArray[i] = SL_IID_NULL;
    }
    // Set arrays required[] and iidArray[] for
VOLUME interface
    required[0] = SL_BOOLEAN_TRUE;
    iidArray[0] = SL_IID_VOLUME;
    // Create Output Mix object to be used by
player
    res = (*EngineItf)-
>CreateOutputMix(EngineItf, &OutputMix, 1,
                     iidArray, required);
```

```
CheckErr(res);
    // Realizing the Output Mix object in
synchronous mode.
    res = (*OutputMix)->Realize(OutputMix,
SL_BOOLEAN_FALSE);
    CheckErr(res);
    res = (*OutputMix)->GetInterface(OutputMix,
SL_IID_VOLUME,
                        (void*)&volumeItf);
CheckErr(res);
    /* Setup the data source structure for the
buffer queue */
    bufferQueue.locatorType =
SL_DATALOCATOR_BUFFERQUEUE;
    bufferQueue.numBuffers = 4; /* Four buffers
in our buffer queue */
    /* Setup the format of the content in the
buffer queue */
    pcm.formatType = SL_DATAFORMAT_PCM;
    pcm.numChannels = 2;
    pcm.samplesPerSec = SL_SAMPLINGRATE_44_1;
    pcm.bitsPerSample =
SL_PCMSAMPLEFORMAT_FIXED_16;
    pcm.containerSize = 16;
    pcm.channelMask = SL_SPEAKER_FRONT_LEFT |
SL_SPEAKER_FRONT_RIGHT;
    pcm.endianness = SL_BYTEORDER_LITTLEENDIAN;
```

```
    audioSource.pFormat    = (void *)&pcm;
    audioSource.pLocator    = (void
*)&bufferQueue;
    /* Setup the data sink structure */
    locator_outputmix.locatorType =
SL_DATALOCATOR_OUTPUTMIX;
    locator_outputmix.outputMix = OutputMix;
    audioSink.pLocator = (void
*)&locator_outputmix;
    audioSink.pFormat = NULL;
    /* Initialize the context for Buffer queue
callbacks */
    cntxt.pDataBase = (void*)&pcmData;
    cntxt.pData = cntxt.pDataBase;
    cntxt.size = sizeof(pcmData);
    /* Set arrays required[] and iidArray[] for
SEEK interface
       (PlayItf is implicit) */
    required[0] =    SL_BOOLEAN_TRUE;
    iidArray[0] =    SL_IID_BUFFERQUEUE;
    /* Create the music player */
    res = (*EngineItf)-
>CreateAudioPlayer(EngineItf, &player,
                       &audioSource, &audioSink,
1, iidArray, required); CheckErr(res);
    /* Realizing the player in synchronous
mode. */
```

```
    res = (*player)->Realize(player,
SL_BOOLEAN_FALSE); CheckErr(res);
    /* Get seek and play interfaces */
    res = (*player)->GetInterface(player,
SL_IID_PLAY, (void*)&playItf);
    CheckErr(res);
    res = (*player)->GetInterface(player,
SL_IID_BUFFERQUEUE,
                  (void*)&bufferQueueItf);
CheckErr(res);
    /* Setup to receive buffer queue event
callbacks */
    res = (*bufferQueueItf)-
>RegisterCallback(bufferQueueItf,
                          BufferQueueCallback,
NULL); CheckErr(res);
    /* Before we start set volume to -3dB
(-300mB) */
    res = (*volumeItf)-
>SetVolumeLevel(volumeItf, -300);
CheckErr(res);
    /* Enqueue a few buffers to get the ball
rolling */
    res = (*bufferQueueItf)-
>Enqueue(bufferQueueItf, cntxt.pData,
                      2 *
AUDIO_DATA_BUFFER_SIZE); /* Size given in
```

```
bytes. */
    CheckErr(res);
    cntxt.pData += AUDIO_DATA_BUFFER_SIZE;
    res = (*bufferQueueItf)-
>Enqueue(bufferQueueItf, cntxt.pData,
                        2 *
AUDIO_DATA_BUFFER_SIZE); /* Size given in
bytes. */
    CheckErr(res);
    cntxt.pData += AUDIO_DATA_BUFFER_SIZE;
    res = (*bufferQueueItf)-
>Enqueue(bufferQueueItf, cntxt.pData,
                        2 *
AUDIO_DATA_BUFFER_SIZE); /* Size given in
bytes. */
    CheckErr(res);
    cntxt.pData += AUDIO_DATA_BUFFER_SIZE;
    /* Play the PCM samples using a buffer
queue */
    res = (*playItf)->SetPlayState( playItf,
SL_PLAYSTATE_PLAYING );
    CheckErr(res);
    /* Wait until the PCM data is done playing,
the buffer queue callback
       will continue to queue buffers until the
entire PCM data has been
       played. This is indicated by waiting for
```

```
the count member of the
       SLBufferQueueState to go to zero.
    */
    res = (*bufferQueueItf)-
>GetState(bufferQueueItf, &state);
    CheckErr(res);
    while(state.count)
    {
        (*bufferQueueItf)-
>GetState(bufferQueueItf, &state);
        /* should sleep be called here? */
    }
    /* Make sure player is stopped */
    res = (*playItf)->SetPlayState(playItf,
SL_PLAYSTATE_STOPPED);
    CheckErr(res);
    /* Destroy the player */
    (*player)->Destroy(player);
    /* Destroy Output Mix object */
    (*OutputMix)->Destroy(OutputMix);
}
int sl_main( void )
{
    SLresult res;
    SLObjectItf sl;
    SLEngineOption EngineOption[] = {
    (SLuint32) SL_ENGINEOPTION_THREADSAFE,
```

```
    (SLuint32) SL_BOOLEAN_TRUE};
    res = slCreateEngine( &sl, 1, EngineOption,
0, NULL, NULL);
    CheckErr(res);
    /* Realizing the SL Engine in synchronous
mode. */
    res = (*sl)->Realize(sl, SL_BOOLEAN_FALSE);
CheckErr(res);
    TestPlayMusicBufferQueue(sl);
    /* Shutdown OpenSL ES */
    (*sl)->Destroy(sl);
    exit(0);
}
```

Record audio

The ability to record audio in OpenSL ES is an option for the implementation. Consequently, we cannot guarantee that there is any microphone or similar device. The main difference in this program from the previous one is that checks need to be performed first on the input devices:

```
res = (*sl)->GetInterface(sl,
SL_IID_AUDIOIODEVICECAPABILITIES,

(void*)&AudioIODeviceCapabilitiesItf);
CheckErr(res);
    numInputs = MAX_NUMBER_INPUT_DEVICES;
    res = (*AudioIODeviceCapabilitiesItf)-
>GetAvailableAudioInputs(

AudioIODeviceCapabilitiesItf, &numInputs,
InputDeviceIDs); CheckErr(res);
    /* Search for either earpiece microphone or
headset microphone input
       device - with a preference for the
latter */
    for (i=0;i < numInputs; i++)
    {
        res = (*AudioIODeviceCapabilitiesItf)-

>QueryAudioInputCapabilities(AudioIODeviceCapabil

                          InputDeviceIDs[i],
&AudioInputDescriptor); CheckErr(res);

if((AudioInputDescriptor.deviceConnection ==
        SL_DEVCONNECTION_ATTACHED_WIRED)&&
```

```
            (AudioInputDescriptor.deviceScope ==
SL_DEVSCOPE_USER)&&
            (AudioInputDescriptor.deviceLocation
==
        SL_DEVLOCATION_HEADSET))
        {
            mic_deviceID = InputDeviceIDs[i];
            mic_available = SL_BOOLEAN_TRUE;
            break;
        }
        else
if((AudioInputDescriptor.deviceConnection ==
             SL_DEVCONNECTION_INTEGRATED)&&
            (AudioInputDescriptor.deviceScope
==
             SL_DEVSCOPE_USER)&&

(AudioInputDescriptor.deviceLocation ==
             SL_DEVLOCATION_HANDSET))
        {
            mic_deviceID = InputDeviceIDs[i];
            mic_available = SL_BOOLEAN_TRUE;
            break;
        }
    }
    /* If neither of the preferred input audio
devices is available, no
```

```
        point in continuing */
    if (!mic_available) {
    /* Appropriate error message here */
    exit(1);
    }
```

The other major differences are that the engine is used to create an audio recorder rather than a player, and that the recorded sounds are saved to a file.

```c
#include <stdio.h>
#include <stdlib.h>
#include "OpenSLES.h"
#define MAX_NUMBER_INTERFACES 5
#define MAX_NUMBER_INPUT_DEVICES 3
#define POSITION_UPDATE_PERIOD 1000 /* 1 sec */
/* Checks for error. If any errors exit the
application! */
void CheckErr( SLresult res )
{
    if ( res != SL_RESULT_SUCCESS )
    {
        // Debug printing to be placed here
        exit(1);
    }
}
void RecordEventCallback(SLRecordItf caller,
             void *pContext,
             SLuint32 recordevent)

{
    /* Callback code goes here */
}
/*
 * Test recording of audio from a microphone
into a specified file
```

```
 */
void TestAudioRecording(SLObjectItf sl)
{
    SLObjectItf
    recorder;
    SLRecordItf
    recordItf;
    SLEngineItf
    EngineItf;
    SLAudioIODeviceCapabilitiesItf
AudioIODeviceCapabilitiesItf;
    SLAudioInputDescriptor
    AudioInputDescriptor;
    SLresult
    res;
    SLDataSource audioSource;
    SLDataLocator_IODevice locator_mic;
    SLDeviceVolumeItf devicevolumeItf;
    SLDataSink audioSink;
    SLDataLocator_URI uri;
    SLDataFormat_MIME mime;
    int i;
    SLboolean required[MAX_NUMBER_INTERFACES];
    SLInterfaceID
iidArray[MAX_NUMBER_INTERFACES];
    SLuint32
InputDeviceIDs[MAX_NUMBER_INPUT_DEVICES];
```

```
    SLint32
    numInputs = 0;
    SLboolean mic_available = SL_BOOLEAN_FALSE;
    SLuint32 mic_deviceID = 0;
    /* Get the SL Engine Interface which is
implicit */
    res = (*sl)->GetInterface(sl,
SL_IID_ENGINE, (void*)&EngineItf);
    CheckErr(res);
    /* Get the Audio IO DEVICE CAPABILITIES
interface, which is also
       implicit */
    res = (*sl)->GetInterface(sl,
SL_IID_AUDIOIODEVICECAPABILITIES,

(void*)&AudioIODeviceCapabilitiesItf);
CheckErr(res);
    numInputs = MAX_NUMBER_INPUT_DEVICES;
    res = (*AudioIODeviceCapabilitiesItf)-
>GetAvailableAudioInputs(

AudioIODeviceCapabilitiesItf, &numInputs,
InputDeviceIDs); CheckErr(res);
    /* Search for either earpiece microphone or
headset microphone input
       device - with a preference for the
latter */
```

```
    for (i=0;i<numInputs; i++)
    {
        res = (*AudioIODeviceCapabilitiesItf)-

>QueryAudioInputCapabilities(AudioIODeviceCapabil

                          InputDeviceIDs[i],
&AudioInputDescriptor); CheckErr(res);

if((AudioInputDescriptor.deviceConnection ==
        SL_DEVCONNECTION_ATTACHED_WIRED)&&
           (AudioInputDescriptor.deviceScope ==
SL_DEVSCOPE_USER)&&
           (AudioInputDescriptor.deviceLocation
==
        SL_DEVLOCATION_HEADSET))
        {
            mic_deviceID = InputDeviceIDs[i];
            mic_available = SL_BOOLEAN_TRUE;
            break;
        }
        else
if((AudioInputDescriptor.deviceConnection ==
             SL_DEVCONNECTION_INTEGRATED)&&
            (AudioInputDescriptor.deviceScope
==
             SL_DEVSCOPE_USER)&&
```

```
(AudioInputDescriptor.deviceLocation ==
             SL_DEVLOCATION_HANDSET))
        {
            mic_deviceID = InputDeviceIDs[i];
            mic_available = SL_BOOLEAN_TRUE;
            break;
        }
    }
    /* If neither of the preferred input audio
devices is available, no
       point in continuing */
    if (!mic_available) {
    /* Appropriate error message here */
    exit(1);
    }
    /* Initialize arrays required[] and
iidArray[] */
    for (i=0;i<MAX_NUMBER_INTERFACES;i++)
    {
        required[i] = SL_BOOLEAN_FALSE;
        iidArray[i] = SL_IID_NULL;
    }
    /* Get the optional DEVICE VOLUME interface
from the engine */
    res = (*sl)->GetInterface(sl,
SL_IID_DEVICEVOLUME,
```

```
                    (void*)&devicevolumeItf);
CheckErr(res);
    /* Set recording volume of the microphone
to -3 dB */
    res = (*devicevolumeItf)-
>SetVolume(devicevolumeItf, mic_deviceID, -
                    300); CheckErr(res);
    /* Setup the data source structure */
    locator_mic.locatorType    =
SL_DATALOCATOR_IODEVICE;
    locator_mic.deviceType    =
SL_IODEVICE_AUDIOINPUT;
    locator_mic.deviceID    = mic_deviceID;
    locator_mic.device            = NULL;
    audioSource.pLocator     = (void
*)&locator_mic;
    audioSource.pFormat         = NULL;
    /* Setup the data sink structure */
    uri.locatorType    = SL_DATALOCATOR_URI;
    uri.URI    = (SLchar *)
"file:///recordsample.wav";
    mime.formatType    = SL_DATAFORMAT_MIME;
    mime.mimeType    = (SLchar *) "audio/x-
wav";
    mime.containerType    =
SL_CONTAINERTYPE_WAV;
    audioSink.pLocator    = (void *)&uri;
```

```
    audioSink.pFormat    = (void *)&mime;
    /* Create audio recorder */
    res = (*EngineItf)-
>CreateAudioRecorder(EngineItf, &recorder,
                         &audioSource,
&audioSink, 0, iidArray, required);
CheckErr(res);
    /* Realizing the recorder in synchronous
mode. */
    res = (*recorder)->Realize(recorder,
SL_BOOLEAN_FALSE); CheckErr(res);
    /* Get the RECORD interface - it is an
implicit interface */
    res = (*recorder)->GetInterface(recorder,
SL_IID_RECORD,
                    (void*)&recordItf);
CheckErr(res);
    /* Setup to receive position event
callbacks */
    res = (*recordItf)-
>RegisterCallback(recordItf,
RecordEventCallback,
                     NULL);
    CheckErr(res);
    /* Set notifications to occur after every
second - may be useful in
       updating a recording progress bar */
```

```
    res = (*recordItf)-
>SetPositionUpdatePeriod( recordItf,

POSITION_UPDATE_PERIOD); CheckErr(res);
    res = (*recordItf)->SetCallbackEventsMask(
recordItf,

SL_RECORDEVENT_HEADATNEWPOS); CheckErr(res);
    /* Set the duration of the recording - 30
seconds (30,000
       milliseconds) */
    res = (*recordItf)-
>SetDurationLimit(recordItf, 30000);
CheckErr(res);
    /* Record the audio */
    res = (*recordItf)-
>SetRecordState(recordItf,SL_RECORDSTATE_RECORDIN

    /* Destroy the recorder object */
    (*recorder)->Destroy(recorder);
}
int sl_main( void )
{
    SLresult
    res;
    SLObjectItf sl;
    /* Create OpenSL ES engine in thread-safe
```

```
mode */
    SLEngineOption EngineOption[] = {(SLuint32)

SL_ENGINEOPTION_THREADSAFE, (SLuint32)
SL_BOOLEAN_TRUE};
    res = slCreateEngine( &sl, 1, EngineOption,
0, NULL, NULL);
    CheckErr(res);
    /* Realizing the SL Engine in synchronous
mode. */
    res = (*sl)->Realize(sl, SL_BOOLEAN_FALSE);
CheckErr(res);
    TestAudioRecording(sl);
    /* Shutdown OpenSL ES */
    (*sl)->Destroy(sl);
    exit(0);
}
```

OpenMAX AL example

The following example OpenMAX_AL_playback.c is from the OpenMAX AL specification. It has not been tested by me!

The functionality of this program for the audio component overlaps that of the OpenSL_playback.c program given earlier. The structure is very similar. The most notable difference is that all functions and constants that begin with `SL` for OpenSL ES now begin with `XA` for OpenMAX AL.

This program also allows some user interaction to pause/start the playback as well as checking the loudspeaker availability.

The code is

```c
/*
 * OpenMAX AL - Audio/Video Playback Example
 */
#include <stdio.h>
#include <stdlib.h>
#include "OpenMAXAL.h"
#define MAX_NUMBER_INTERFACES 5
#define MAX_NUMBER_OUTPUT_DEVICES 3
#define POSITION_UPDATE_PERIOD 1000 /* 1 sec */
/* Checks for error. If any errors exit the
application! */
void CheckErr (XAresult res)
{
    if (res != XA_RESULT_SUCCESS)
    {
        /* Debug printing to be placed here */
        exit(1);
    }
}
void PlayEventCallback (
            XAPlayItf caller,
            void *pContext,
            XAuint32 playevent)
{
    /* Callback code goes here */
}
```

```
/*
 * Test audio/video playback from a 3GPP file.
 *
 * NOTE: For the purposes of this example, the
implementation is assumed
 * to support the requisite audio and video
codecs. Therefore, video and
 * audio decoder capabilities are NOT checked
in this example.
 */
void TestAudioVideoPlayback (XAObjectItf
engine)
{
    XAObjectItf    player;
    XAObjectItf    OutputMix;
    XAPlayItf    playItf;
    XAEngineItf    EngineItf;
    XAAudioIODeviceCapabilitiesItf
AudioIODeviceCapabilitiesItf;
    XAAudioOutputDescriptor
AudioOutputDescriptor;
    XAresult    res;
    XADataSink     audioSink;
    XADataSink  videoSink;
    XADataLocator_OutputMix
locator_outputmix;
    XADataLocator_NativeDisplay
```

```
locator_displayregion;
    XAVolumeItf    volumeItf;
    XADataSource    avSource;
    XADataLocator_URI uri;
    XADataFormat_MIME mime;
    int i;
    char c;
    XAboolean
required[MAX_NUMBER_INTERFACES];
    XAInterfaceID
iidArray[MAX_NUMBER_INTERFACES];
    XAuint32
OutputDeviceIDs[MAX_NUMBER_OUTPUT_DEVICES];
    XAint32    numOutputs    = 0;
    XAboolean     hfs_available    =
XA_BOOLEAN_FALSE;
    XAboolean    hfs_default    =
XA_BOOLEAN_FALSE;
    XAuint32    hfs_deviceID    = 0;
    XANativeHandle    nativeWindowHandle =
NULL;
    XANativeHandle nativeDisplayHandle = NULL;
    /* Get the XA Engine Interface, which is
implicit */
    res = (*engine)->GetInterface(engine,
                    XA_IID_ENGINE, (void*)
&EngineItf); CheckErr(res);
```

```
    /* Get the Audio IO DEVICE CAPABILITIES
interface, which is also
     * implicit */
    res = (*engine)->GetInterface(engine,

XA_IID_AUDIOIODEVICECAPABILITIES,
                  (void*)
&AudioIODeviceCapabilitiesItf); CheckErr(res);
    numOutputs = MAX_NUMBER_OUTPUT_DEVICES;
    res = (*AudioIODeviceCapabilitiesItf)->

GetAvailableAudioOutputs(AudioIODeviceCapabilitie

                 &numOutputs, OutputDeviceIDs);
CheckErr(res);
    /* Search for integrated handsfree
loudspeaker */
    for (i = 0; i < numOutputs; i++)
    {
        res = (*AudioIODeviceCapabilitiesItf)->

QueryAudioOutputCapabilities(AudioIODeviceCapabil

                         OutputDeviceIDs[i],
&AudioOutputDescriptor);
        CheckErr(res);
        if
```

```
((AudioOutputDescriptor.deviceConnection ==
         XA_DEVCONNECTION_INTEGRATED) &&
        (AudioOutputDescriptor.deviceScope ==
         XA_DEVSCOPE_ENVIRONMENT) &&
        (AudioOutputDescriptor.deviceLocation
==
         XA_DEVLOCATION_HANDSET))
        {
            hfs_deviceID = OutputDeviceIDs[i];
            hfs_available = XA_BOOLEAN_TRUE;
            break;
        }
    }
    /* If preferred output audio device is not
available, no point in
     * continuing */
    if (!hfs_available)
    {
        /* Appropriate error message here */
        exit(1);
    }
    numOutputs = MAX_NUMBER_OUTPUT_DEVICES;
    res = (*AudioIODeviceCapabilitiesItf)->

GetDefaultAudioDevices(AudioIODeviceCapabilitiesI
```

```
XA_DEFAULTDEVICEID_AUDIOOUTPUT, &numOutputs,
                    OutputDeviceIDs);
    CheckErr(res);
    /* Check whether Default Output devices
include the handsfree
     * loudspeaker */
    for (i = 0; i < numOutputs; i++)
    {
        if (OutputDeviceIDs[i] == hfs_deviceID)
        {
            hfs_default = XA_BOOLEAN_TRUE;
            break;
        }
    }
    /* Expect handsfree loudspeaker to be set
as one of the default
     * output devices */
    if (!hfs_default)
    {
        /* Debug printing to be placed here */
        exit(1);
    }
    /* Initialize arrays required[] and
iidArray[] */
    for (i = 0; i < MAX_NUMBER_INTERFACES; i++)
    {
        required[i] = XA_BOOLEAN_FALSE;
```

```
        iidArray[i] = XA_IID_NULL;
    }
    /* Set arrays required[] and iidArray[] for
VOLUME interface */
    required[0] = XA_BOOLEAN_TRUE;
    iidArray[0] = XA_IID_VOLUME;
    /* Create Output Mix object to be used by
player */
    res = (*EngineItf)-
>CreateOutputMix(EngineItf,
                    &OutputMix, 1, iidArray,
required); CheckErr(res);
    /* Realizing the Output Mix object in
synchronous mode
       res = (*OutputMix)->Realize(OutputMix,
       XA_BOOLEAN_FALSE); CheckErr(res);
    */
    /* Get the volume interface on the output
mix */
    res = (*OutputMix)->GetInterface(OutputMix,
                     XA_IID_VOLUME,
(void*)&volumeItf); CheckErr(res);
    /* Setup the audio/video data source
structure */
    uri.locatorType    = XA_DATALOCATOR_URI;
    uri.URI    = (XAchar *)
"file:///avmedia.3gp";
```

```
    mime.formatType    = XA_DATAFORMAT_MIME;
    mime.mimeType    = (XAchar *) "video/3gpp";
    mime.containerType = XA_CONTAINERTYPE_3GPP;
/* provided as a hint to
                         * the player */
    avSource.pLocator = (void*) &uri;
    avSource.pFormat = (void*) &mime;
    /* Setup the audio data sink structure */
    locator_outputmix.locatorType =
XA_DATALOCATOR_OUTPUTMIX;
    locator_outputmix.outputMix    = OutputMix;
    audioSink.pLocator    = (void*)
&locator_outputmix;
    audioSink.pFormat
    = NULL;
    /* Set nativeWindowHandle and
nativeDisplayHandle to
     * platform-specific values here */
    /* nativeWindowHandle = <a platform-
specific value>; */
    /* nativeDisplayHandle = <a platform-
specific value>; */
    /* Setup the video data sink structure */
    locator_displayregion.locatorType =
XA_DATALOCATOR_NATIVEDISPLAY;
    locator_displayregion.hWindow    =
nativeWindowHandle;
```

```
    locator_displayregion.hDisplay    =
nativeDisplayHandle;
    videoSink.pLocator    = (void*)
&locator_displayregion;
    videoSink.pFormat    = NULL;
    /* Create the media player. pBankSrc is
NULL as we have a non-MIDI
     * data source */
    res = (*EngineItf)-
>CreateMediaPlayer(EngineItf,
                       &player, &avSource, NULL,
&audioSink, &videoSink, NULL, NULL,
                       1, iidArray, required);
CheckErr(res);
    /* Realizing the player in synchronous mode
*/
    res = (*player)->Realize(player,
XA_BOOLEAN_FALSE); CheckErr(res);
    /* Get play interface */
    res = (*player)->GetInterface(player,
                  XA_IID_PLAY, (void*)
&playItf); CheckErr(res);
    /* Setup to receive position event
callbacks */
    res = (*playItf)->RegisterCallback(playItf,
                        PlayEventCallback,
NULL); CheckErr(res);
```

```
    /* Set notifications to occur after every 1
second - might be useful
     * in updating a progress bar */
    res = (*playItf)-
>SetPositionUpdatePeriod(playItf,

POSITION_UPDATE_PERIOD); CheckErr(res);
    res = (*playItf)-
>SetCallbackEventsMask(playItf,

XA_PLAYEVENT_HEADATNEWPOS); CheckErr(res);
    /* Before we start, set volume to -3dB
(-300mB) */
    res = (*volumeItf)-
>SetVolumeLevel(volumeItf, -300);
CheckErr(res);
    /* Play the media */
    res = (*playItf)->SetPlayState(playItf,
                    XA_PLAYSTATE_PLAYING);
CheckErr(res);
    while ((c = getchar()) != 'q')
    {
        XAuint32 playState;
        switch(c)
        {
        case '1':
            /* Play the media - if it is not
```

```
already playing */
            res = (*playItf)-
>GetPlayState(playItf,
                            &playState);
CheckErr(res);
            if (playState !=
XA_PLAYSTATE_PLAYING)
            {
                res = (*playItf)-
>SetPlayState(playItf,

XA_PLAYSTATE_PLAYING); CheckErr(res);
                455
                }
            break;
        case '2':
            /* Pause the media - if it is
playing */
            res = (*playItf)-
>GetPlayState(playItf,
                            &playState);
CheckErr(res);
            if (playState ==
XA_PLAYSTATE_PLAYING)
            {
                res = (*playItf)-
>SetPlayState(playItf,
```

```
XA_PLAYSTATE_PAUSED); CheckErr(res);
            }
            break;
        default:
            break;
        }
    }
    /* Stop the media playback */
    res = (*playItf)->SetPlayState(playItf,
                    XA_PLAYSTATE_STOPPED);
CheckErr(res);
    /* Destroy the player object */
    (*player)->Destroy(player);
    /* Destroy the output mix object */
    (*OutputMix)->Destroy(OutputMix);
}
int xa_main (void)
{
    XAresult    res;
    XAObjectItf engine;
    /* Create OpenMAX AL engine in thread-safe
mode */
    XAEngineOption EngineOption[] = {
    (XAuint32) XA_ENGINEOPTION_THREADSAFE,
    (XAuint32) XA_BOOLEAN_TRUE};
    res = xaCreateEngine(&engine,
```

```
                1, EngineOption, 0, NULL, NULL);
CheckErr(res);
    /* Realizing the AL Engine in synchronous
mode */
    res = (*engine)->Realize(engine,
XA_BOOLEAN_FALSE); CheckErr(res);
    TestAudioVideoPlayback(engine);
    /* Shutdown OpenMAX AL engine */
    (*engine)->Destroy(engine);
    exit(0);
}
```

Further examples are given in the OpenMAX AL specification.

OpenMAX IL

Implementations

There are three implementations that I have been able to access.

Raspberry Pi

The Raspberry Pi has a Broadcom GPU (graphics processing unit) and Broadcom support OpenMAX IL. The include files needed to build applications are in `/opt/vc/include/IL`, `/opt/vc/include` and `/opt/vc/include/interface/vcos/pthreads`. The libraries that need to be linked are in the `/opt/vc/lib` directory and are `openmaxil` and `bcm_host`.

The Broadcom libraries need additional code to be called as well as standard OpenMAX IL functions. In addition, there are a number of (legal) extensions to OpenMAX IL that are not found in the specification or in other implementations. These are

described in `/opt/vc/include/IL/OMX_Broadcom.h`. For these reasons I define `RASPBERRY_PI` to allow these to be dealt with.

The compile line for e.g. `listcomponents.c` is

```
cc -g -DRASPBERRY_PI -I /opt/vc/include/IL -I
/opt/vc/include \
   -I /opt/vc/include/interface/vcos/pthreads \
   -o listcomponents listcomponents.c \
   -L /opt/vc/lib -l openmaxil -l bcm_host
```

The Broadcom implementation is closed source. It appears to be a thin wrapper around their GPU API, and they will not release any details of that API. This means that you cannot extend the set of components, or the codecs supported, since there are no details of how to build new components. While the set of components is reasonable, at present there is no support for codecs other than PCM, and there is no support of non-GPU hardware such as USB soundcards.

OtherCrashOverride says he has managed to get the Broadcom components running under the LIM implementation, but I haven't confirmed that yet.

Bellagio

The Bellagio library does not require additional code or have any extensions. There are a few minor bugs, so I define `BELLAGIO` to handle them. I built from source, but didn't install, so the includes and libraries are in a funny place. My compile line is

```
cc  -g -DBELLAGIO -I ../libomxil-bellagio-
0.9.3/include/ \
    -o listcomponents listcomponents.c \
    -L ../libomxil-bellagio-0.9.3/src/.libs -l
omxil-bellagio
```

and at run time

```
export LD_LIBRARY_PATH=../libomxil-bellagio-
0.9.3/src/.libs/
./listcomponents
```

The Bellagio code is open source.

LIM

Downloading the 1.1 version was a hassle because the 1.1 download uses a Git repo that has disappeared (as of Feb, 2013). Instead you have to run

```
git clone
git://limoa.git.sourceforge.net/gitroot/limoa/lim
components
  git clone
git://limoa.git.sourceforge.net/gitroot/limoa/lim
core
  git clone
git://limoa.git.sourceforge.net/gitroot/limoa/lim
plugins
  git clone
git://limoa.git.sourceforge.net/gitroot/limoa/lim

  git clone
git://limoa.git.sourceforge.net/gitroot/limoa/man
```

"You have to copy the root.mk in build to a top level folder containing all the code and rename it Makefile. The root.readme file has build instructions." Thanks to OtherCrashOverride for these instructions.

Building the library had some minor hiccups. I had to comment out a couple of lines from one video file as it referred to non-existent structure fields, and had to remove `-Werrors` from one `Makefile.am` as otherwise a warning about an unused variable would abort the compile.

The library build puts files in a new directory in my HOME. I have found some minor bugs in the implementation so far. My compile line is

```
cc -g -DLIM -I ../../lim-omx-1.1/LIM/limoi-core/include/ \
   -o listcomponents listcomponents.c \
   -L /home/newmarch/osm-build/lib/ -l limoa -l limoi-core
```

and at runtime,

```
export LD_LIBRARY_PATH=/home/newmarch/osm-build/lib/
./listcomponents
```

The LIM code is open source.

Hardware supported versions

A list of hardware supported versions is at OpenMAX IL Conformant Products .

OpenMAX IL concepts

The OpenMAX IL API is quite distinct from that of OpenMAX AL. The basic concept is of a *Component* , which is an audio/video (or other) processing unit of some type, such as a volume control, a mixer, an output device. Each Component has zero or more input and output *ports* , and each port can have one or more *buffers* that carry data.

OpenMAX IL is typically meant for use by an A/V framework of some kind, such as OpenMAX AL. In addition to OpenMAX AL, there is curently a GStreamer plugin that uses OpenMAX IL underneath. But one can also build standalone applications where direct calls are made into the OpenMAX IL API. Collectively, these are all known as *IL clients* .

The OpenMAX IL API is difficult to work with directly. Error messages are frequently quite useless and threads will block without explanation until everything is *exactly* right - and silently blocking doesn't give you any clues about what isn't right. In

addition, the examples I have to work with don't follow the specification exactly correctly which can lead to much wasted time.

OpenMAX IL components use buffers to carry data. A component will usually process data from an input buffer and place it on an output buffer. This processing is not visible to the API, and so allows vendors to implement components in hardware or software, built on top of other A/V components, etc. OpenMAX IL gives mechanisms for setting and getting parameters of components, for calling standard functions on the components, or for getting data in and out of components.

While some of the OpenMAX IL calls are synchronous, those that require possibly substantial amounts of processing are asynchronous, communicating the results through callback functions. This leads naturally to a multi-threaded processing model, although OpenMAX IL does not visibly use any thread libraries and should be agnostic to how an IL client uses threads. The Bellagio examples use pthreads while the Broadcom examples for the Raspberry Pi use Broadcom's VideoCore O/S (vcos) threads.

There are two mechanisms for getting data into and out of components. The first is where the IL client makes calls on the component. All components are required to support this mechanism. The second is where a *tunnel* is set up between two components for data to flow along a shared buffer. A component is not required to support this mechanism.

OpenMAX IL components

OpenMAX IL in 1.1.2 lists a number of standard components, including (for audio) a decoder, an encoder, a mixer, a reader, a renderer, a writer, a capturer and a processor. An IL client gets such a component by calling `OMX_GetHandle()`, passing in the name of the component. This is a problem: the components do not have a standard name. The 1.1.2 specification says:

> Since components are requested by name, a naming convention is defined. OpenMAX IL component names are zero terminated strings with the following format: “OMX..”. For example:

OMX.CompanyABC.MP3Decoder.productXYZ

No standardization among component names is dictated across different vendors.

The Bellagio library (you need the source package to see these files) lists in its README only two audio components:

- OMX audio volume control
- OMX audio mixer component

and their names (from the example test files) are "OMX.st.volume.component" and "OMX.st.audio.mixer" respectively. The company behind Bellagio is STMicroelectronics which explains the "st".

The Broadcom OpenMAX IL implementation used on the Raspberry Pi is much better documented. If you download the firmware-master file for the Raspberry Pi it lists the IL components in the documentation/ilcomponents directory. This lists the components audio_capture, audio_decode, audio_encode, audio_lowpower, audio_mixer, audio_processor, audio_render and audio_splitter.

Many of the openMAX IL function calls in the Broadcom examples are buried in Broadcom convenience functions such as

```
ilclient_create_component(st->client, &st-
>audio_render,
                          "audio_render",

ILCLIENT_ENABLE_INPUT_BUFFERS |
ILCLIENT_DISABLE_ALL_PORTS);
```

which wraps around `OMX_GetHandle()`. But at least the `ilclient.h` states "Component names as provided are automatically prefixed with 'OMX.broadcom.' before passing to the IL core.". So we can conclude that the real names are e.g. "OMX.broadcom.audio_render" etc.

There is a simple way of programmatically getting the supported components. First initialise the OpenMAX system by `OMX_init()` and then make calls to `OMX_ComponentNameEnum()`. For successive index values it returns a unique name each time, until it finally returns an error value of `OMX_ErrorNoMore`.

Each component may support a number of *roles*. These are given by `OMX_GetRolesOfComponent`. The 1.1 specification lists classes of audio components and associated roles in section

8.6 "Standard Audio Components". The LIM library matches these, while Bellagio and Broadcom do not.

The program is listcomponents.c:

```c
#include <stdio.h>
#include <stdlib.h>

#include <OMX_Core.h>

#ifdef RASPBERRY_PI
#include <bcm_host.h>
#endif

OMX_ERRORTYPE err;

//extern OMX_COMPONENTREGISTERTYPE
OMX_ComponentRegistered[];

void listroles(char *name) {
    int n;
    OMX_U32 numRoles;
    OMX_U8 *roles[32];

    /* get the number of roles by passing in a
NULL roles param */
    err = OMX_GetRolesOfComponent(name,
&numRoles, NULL);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Getting roles failed\n",
0);
```

```
    exit(1);
    }
    printf("  Num roles is %d\n", numRoles);
    if (numRoles > 32) {
    printf("Too many roles to list\n");
    return;
    }

    /* now get the roles */
    for (n = 0; n < numRoles; n++) {
    roles[n] = malloc(OMX_MAX_STRINGNAME_SIZE);
    }
    err = OMX_GetRolesOfComponent(name,
&numRoles, roles);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Getting roles failed\n",
0);
    exit(1);
    }
    for (n = 0; n < numRoles; n++) {
    printf("    role: %s\n", roles[n]);
    free(roles[n]);
    }

    /* This is in version 1.2
    for (i = 0; OMX_ErrorNoMore != err; i++) {
    err = OMX_RoleOfComponentEnum(role, name,
```

```
i);
    if (OMX_ErrorNone == err) {
        printf("   Role of omponent is %s\n",
role);
    }
    }
    */
}

int main(int argc, char** argv) {

    int i;
    unsigned char
name[OMX_MAX_STRINGNAME_SIZE];

# ifdef RASPBERRY_PI
    bcm_host_init();
# endif

    err = OMX_Init();
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "OMX_Init() failed\n", 0);
    exit(1);
    }

    err = OMX_ErrorNone;
    for (i = 0; OMX_ErrorNoMore != err; i++) {
```

```
    err = OMX_ComponentNameEnum(name,
OMX_MAX_STRINGNAME_SIZE, i);
    if (OMX_ErrorNone == err) {
        printf("Component is %s\n", name);
        listroles(name);
    }
    }
    printf("No more components\n");

    /*
    i= 0 ;
    while (1) {
    printf("Component %s\n",
OMX_ComponentRegistered[i++]);
    }
    */
    exit(0);
}
```

The output from the Bellagio library is

```
Component is OMX.st.clocksrc
  Num roles is 1
    role: clocksrc
Component is OMX.st.clocksrc
  Num roles is 1
    role: clocksrc
Component is OMX.st.video.scheduler
  Num roles is 1
    role: video.scheduler
Component is OMX.st.video.scheduler
  Num roles is 1
    role: video.scheduler
Component is OMX.st.volume.component
  Num roles is 1
    role: volume.component
Component is OMX.st.volume.component
  Num roles is 1
    role: volume.component
Component is OMX.st.audio.mixer
  Num roles is 1
    role: audio.mixer
Component is OMX.st.audio.mixer
  Num roles is 1
    role: audio.mixer
Component is OMX.st.clocksrc
  Num roles is 1
```

```
    role: clocksrc
Component is OMX.st.clocksrc
  Num roles is 1
    role: clocksrc
Component is OMX.st.video.scheduler
  Num roles is 1
    role: video.scheduler
Component is OMX.st.video.scheduler
  Num roles is 1
    role: video.scheduler
Component is OMX.st.volume.component
  Num roles is 1
    role: volume.component
Component is OMX.st.volume.component
  Num roles is 1
    role: volume.component
Component is OMX.st.audio.mixer
  Num roles is 1
    role: audio.mixer
Component is OMX.st.audio.mixer
  Num roles is 1
    role: audio.mixer
No more components
```

which is not quite correct: the OpenMAX IL specification says that each somponent must appear once only, not repeated.

The Raspberry Pi reports a large number of components but does not define a role for any of them:

```
Component is OMX.broadcom.audio_capture
  Num roles is 0
Component is OMX.broadcom.audio_decode
  Num roles is 0
Component is OMX.broadcom.audio_encode
  Num roles is 0
Component is OMX.broadcom.audio_render
  Num roles is 0
Component is OMX.broadcom.audio_mixer
  Num roles is 0
Component is OMX.broadcom.audio_splitter
  Num roles is 0
Component is OMX.broadcom.audio_processor
  Num roles is 0
Component is OMX.broadcom.camera
  Num roles is 0
Component is OMX.broadcom.clock
  Num roles is 0
Component is OMX.broadcom.coverage
  Num roles is 0
Component is OMX.broadcom.egl_render
  Num roles is 0
Component is OMX.broadcom.image_fx
  Num roles is 0
Component is OMX.broadcom.image_decode
  Num roles is 0
```

```
Component is OMX.broadcom.image_encode
  Num roles is 0
Component is OMX.broadcom.image_read
  Num roles is 0
Component is OMX.broadcom.image_write
  Num roles is 0
Component is OMX.broadcom.read_media
  Num roles is 0
Component is OMX.broadcom.resize
  Num roles is 0
Component is OMX.broadcom.source
  Num roles is 0
Component is OMX.broadcom.text_scheduler
  Num roles is 0
Component is OMX.broadcom.transition
  Num roles is 0
Component is OMX.broadcom.video_decode
  Num roles is 0
Component is OMX.broadcom.video_encode
  Num roles is 0
Component is OMX.broadcom.video_render
  Num roles is 0
Component is OMX.broadcom.video_scheduler
  Num roles is 0
Component is OMX.broadcom.video_splitter
  Num roles is 0
Component is OMX.broadcom.visualisation
```

```
  Num roles is 0
Component is OMX.broadcom.write_media
  Num roles is 0
Component is OMX.broadcom.write_still
  Num roles is 0
No more components
```

The output from LIM is

```
Component is OMX.limoi.alsa_sink
  Num roles is 1
    role: audio_renderer.pcm
Component is OMX.limoi.clock
  Num roles is 1
    role: clock.binary
Component is OMX.limoi.ffmpeg.decode.audio
  Num roles is 8
    role: audio_decoder.aac
    role: audio_decoder.adpcm
    role: audio_decoder.amr
    role: audio_decoder.mp3
    role: audio_decoder.ogg
    role: audio_decoder.pcm
    role: audio_decoder.ra
    role: audio_decoder.wma
Component is OMX.limoi.ffmpeg.decode.video
  Num roles is 7
    role: video_decoder.avc
    role: video_decoder.h263
    role: video_decoder.mjpeg
    role: video_decoder.mpeg2
    role: video_decoder.mpeg4
    role: video_decoder.rv
    role: video_decoder.wmv
Component is OMX.limoi.ffmpeg.demux
```

```
  Num roles is 1
    role: container_demuxer.all
Component is OMX.limoi.ffmpeg.encode.audio
  Num roles is 2
    role: audio_encoder.aac
    role: audio_encoder.mp3
Component is OMX.limoi.ffmpeg.encode.video
  Num roles is 2
    role: video_encoder.h263
    role: video_encoder.mpeg4
Component is OMX.limoi.ffmpeg.mux
  Num roles is 1
    role: container_muxer.all
Component is OMX.limoi.ogg_dec
  Num roles is 1
    role: audio_decoder_with_framing.ogg
Component is OMX.limoi.sdl.renderer.video
  Num roles is 1
    role: iv_renderer.yuv.overlay
Component is OMX.limoi.video_scheduler
  Num roles is 1
    role: video_scheduler.binary
No more components
```

Getting information about an IL component

We will next look at how to get information about the OpenMAX IL system and any component that we use. All IL clients must initialise OpenMAX IL by calling `OMX_Init()`. Nearly all functions return error values, and the style used by Bellagio is

```
err = OMX_Init();
  if(err != OMX_ErrorNone) {
      fprintf(stderr, "OMX_Init() failed\n",
0);
      exit(1);
  }
```

This looks like a reasonable style to me, so I follow it in the sequel.

The next requirement is to get a *handle* to a component. This requires the vendor's name for the component, which can be found using the `listcomponents.c` program above. The function `OMX_GetHandle` takes some parameters including a set of *callback* functions. These are needed to track behaviour of

the application, but are not needed for the example in this section. This code shows how to get a handle to the Bellagio Volume component:

```
OMX_HANDLETYPE handle;
  OMX_CALLBACKTYPE callbacks;
  OMX_ERRORTYPE err;

  err = OMX_GetHandle(&handle,
"OMX.st.volume.component", NULL /*appPriv */,
&callbacks);
  if(err != OMX_ErrorNone) {
      fprintf(stderr, "OMX_GetHandle failed\n",
0);
      exit(1);
  }
```

The component has ports and the ports have channels. Getting and setting information about these is done by the functions `OMX_GetParameter()` , `OMX_SetParameter()` , `OMX_GetConfig()` and `OMX_GetConfig()` . The ...Parameter calls are made before the component is "loaded", ...Config calls are made after it is loaded.

C is not an O/O language and this is an ordinary function call (well, actually it's a macro). In an O/O language it would be a method of an object taking another object as parameter as in `component.method(object)`. In OpenMAX IL the Get/Set function takes the calling "object" as first parameter

- the component, an indicator of what type of "object" the method's parameter is - an index into possible "object" types, and a structure for the parameter object. The index values are related to structures in Table 4-2 "Audio Coding Types by Index" of the 1.1 specification.

The calls take a (pointer to a) structure for filling in or extracting values. The structures are all normalised so that they share common fields such as the size of the structure. In Bellagio examples, this is done by a macro `setHeader()`. The structure passed in to get port information is usually a generic structure of type `OMX_PORT_PARAM_TYPE`. Some fields can be accessed directly; some need a typecast to a more specialised type; and some buried down in unions and have to be extracted.

Ports are labelled by integer indices. There are different ports for different functions, such as audio, image, video and other. To get information about the starting value for audio ports, use:

```
setHeader(&param, sizeof(OMX_PORT_PARAM_TYPE));
  err = OMX_GetParameter(handle,
OMX_IndexParamAudioInit, &param);
  if(err != OMX_ErrorNone){
      fprintf(stderr, "Error in getting
OMX_PORT_PARAM_TYPE parameter\n", 0);
    exit(1);
  }
  printf("Audio ports start on %d\n",

((OMX_PORT_PARAM_TYPE)param).nStartPortNumber);
  printf("There are %d open ports\n",
     ((OMX_PORT_PARAM_TYPE)param).nPorts);
```

The macro `setHeader` justs fills in header information such as version numbers, and the size of the data structure.

Particular ports may now be queried about their capablilies. We can query for the type of the port (audio or otherwise), the direction (input or output) and information about the MIME type supported.

```
OMX_PARAM_PORTDEFINITIONTYPE sPortDef;

  setHeader(&sPortDef,
sizeof(OMX_PARAM_PORTDEFINITIONTYPE));
  sPortDef.nPortIndex = 0;
  err = OMX_GetParameter(handle,
OMX_IndexParamPortDefinition, &sPortDef);
  if(err != OMX_ErrorNone){
      fprintf(stderr, "Error in getting
OMX_PORT_PARAM_TYPE parameter\n", 0);
    exit(1);
  }
  if (sPortDef.eDomain == OMX_PortDomainAudio)
{
      printf("Is an audio port\n");
  } else {
      printf("Is other device port\n");
  }

  if (sPortDef.eDir == OMX_DirInput) {
      printf("Port is an input port\n");
  } else {
      printf("Port is an output port\n");
  }

  /* the Audio Port info is buried in a union
```

```
format.audio within the struct */
  printf("Port min buffers %d,  mimetype %s,
encoding %d\n",
     sPortDef.nBufferCountMin,
     sPortDef.format.audio.cMIMEType,
     sPortDef.format.audio.eEncoding);
```

The Bellagio library returns "raw/audio" for the MIME type supported by its volume control component. This is not a valid MIME type as listed by IANA MIME Media Types , though. The value returned from the encoding is zero, corresponding to `OMX_AUDIO_CodingUnused` which also does not seem to be correct.

If we try the same program on the Raspberry Pi component `audio_render` and on the LIM component `OMX.limoi.alsa_sink` we get NULL for the MIME type but an encoding value of 2 which is `OMX_AUDIO_CodingPCM` . PCM has a MIME type of `audio/L16` so NULL seems inappropriate.

An OpenMAX IL library allows a port to be queried for the data types it supports. This is done by querying for a `OMX_AUDIO_PARAM_PORTFORMATTYPE` object using the index `OMX_IndexParamAudioPortFormat` . According to the

specification, for each index from zero upwards a call to `GetParameter()` should return an encoding such as `OMX_AUDIO_CodingPCM` or `OMX_AUDIO_CodingMp3` until there are no more supported formats, on which it returns `OMX_ErrorNoMore`.

The Bellagio code returns a value of `OMX_AUDIO_CodingUnused` which is not correct. The LIM code does not set a value at all, so you just get garbage. The Broadcom implementation works okay, but as discussed below returns values that are not actually supported. So there is limited value in this call...

This code tests this:

```
void getSupportedAudioFormats(int indentLevel,
int portNumber) {
    OMX_AUDIO_PARAM_PORTFORMATTYPE
sAudioPortFormat;

    setHeader(&sAudioPortFormat,
sizeof(OMX_AUDIO_PARAM_PORTFORMATTYPE));
    sAudioPortFormat.nIndex = 0;
    sAudioPortFormat.nPortIndex = portNumber;

    printf("Supported audio formats are:\n");
    for(;;) {
        err = OMX_GetParameter(handle,
OMX_IndexParamAudioPortFormat,
&sAudioPortFormat);
        if (err == OMX_ErrorNoMore) {
        printf("No more formats supported\n");
        return;
        }

    /* This shouldn't occur, but does with
Broadcom library */
    if (sAudioPortFormat.eEncoding ==
OMX_AUDIO_CodingUnused) {
         printf("No coding format returned\n");
         return;
```

```
    }

    switch (sAudioPortFormat.eEncoding) {
    case OMX_AUDIO_CodingPCM:
        printf("Supported encoding is PCM\n");
        break;
    case OMX_AUDIO_CodingVORBIS:
        printf("Supported encoding is Ogg
Vorbis\n");
        break;
    case OMX_AUDIO_CodingMP3:
        printf("Supported encoding is MP3\n");
        break;
#ifdef RASPBERRY_PI
    case OMX_AUDIO_CodingFLAC:
        printf("Supported encoding is FLAC\n");
        break;
    case OMX_AUDIO_CodingDDP:
        printf("Supported encoding is DDP\n");
        break;
    case OMX_AUDIO_CodingDTS:
        printf("Supported encoding is DTS\n");
        break;
    case OMX_AUDIO_CodingWMAPRO:
        printf("Supported encoding is
WMAPRO\n");
        break;
```

```
#endif
    case OMX_AUDIO_CodingAAC:
        printf("Supported encoding is AAC\n");
        break;
    case OMX_AUDIO_CodingWMA:
        printf("Supported encoding is WMA\n");
        break;
    case OMX_AUDIO_CodingRA:
        printf("Supported encoding is RA\n");
        break;
    case OMX_AUDIO_CodingAMR:
        printf("Supported encoding is AMR\n");
        break;
    case OMX_AUDIO_CodingEVRC:
        printf("Supported encoding is EVRC\n");
        break;
    case OMX_AUDIO_CodingG726:
        printf("Supported encoding is G726\n");
        break;
    case OMX_AUDIO_CodingMIDI:
        printf("Supported encoding is MIDI\n");
        break;
    case OMX_AUDIO_CodingATRAC3:
        printf("Supported encoding is
ATRAC3\n");
        break;
    case OMX_AUDIO_CodingATRACX:
```

```
        printf("Supported encoding is
ATRACX\n");
        break;
    case OMX_AUDIO_CodingATRACAAL:
        printf("Supported encoding is
ATRACAAL\n");
        break;
        /*
    case OMX_AUDIO_Coding:
        printf("Supported encoding is \n");
        break;
        */
    default:
        printf("Supported encoding is %d\n",
          sAudioPortFormat.eEncoding);
    }
        sAudioPortFormat.nIndex++;
    }
}
```

Note that the code contains enum values such as `OMX_AUDIO_CodingATRAC3` which are specific to the Broadcom library. These are legal values according to an OpenMAX IL extension mechanism, but of course are not portable values.

The Bellagio library incorrectly returns `OMX_AUDIO_CodingUnused` for every index value.

The Broadcom library can return lots of values. For example, for the `audio_decode` component it returns

```
Supported audio formats are:
      Supported encoding is MP3
      Supported encoding is PCM
      Supported encoding is AAC
      Supported encoding is WMA
      Supported encoding is Ogg Vorbis
      Supported encoding is RA
      Supported encoding is AMR
      Supported encoding is EVRC
      Supported encoding is G726
      Supported encoding is FLAC
      Supported encoding is DDP
      Supported encoding is DTS
      Supported encoding is WMAPRO
      Supported encoding is ATRAC3
      Supported encoding is ATRACX
      Supported encoding is ATRACAAL
      Supported encoding is MIDI
      No more formats supported
```

Regrettably, none of these are actually supported except for PCM. According to jamesh in "OMX_AllocateBuffer fails for audio decoder component":

> The way it works is that the component passes back success for all the codecs it can *potentially* support. (i.e. all the codecs we've ever had going). That is then constrained by what codecs are actually installed. It would be better to run time detect which codecs are present, but that code has never been written since its never been required. It's also unlikely ever to be done as Broadcom no longer support audio codecs in this way - they have moved off the Videocore to the host CPU since they are now powerful enough to handle any audio decoding task

That's kind of sad, really.

Putting all the bits together gives the program info.c:

```
/**
   Based on code
   Copyright (C) 2007-2009 STMicroelectronics
   Copyright (C) 2007-2009 Nokia Corporation
and/or its subsidiary(-ies).
   under the LGPL
*/

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/stat.h>

#include <OMX_Core.h>
#include <OMX_Component.h>
#include <OMX_Types.h>
#include <OMX_Audio.h>

#ifdef RASPBERRY_PI
#include <bcm_host.h>
#endif

OMX_ERRORTYPE err;
```

```
OMX_HANDLETYPE handle;
OMX_VERSIONTYPE specVersion, compVersion;

OMX_CALLBACKTYPE callbacks;

#define indent {int n = 0; while (n++ <
indentLevel*2) putchar(' ');}

static void setHeader(OMX_PTR header, OMX_U32
size) {
    /* header->nVersion */
    OMX_VERSIONTYPE* ver = (OMX_VERSIONTYPE*)
(header + sizeof(OMX_U32));
    /* header->nSize */
    *((OMX_U32*)header) = size;

    /* for 1.2
       ver->s.nVersionMajor =
OMX_VERSION_MAJOR;
       ver->s.nVersionMinor =
OMX_VERSION_MINOR;
       ver->s.nRevision = OMX_VERSION_REVISION;
       ver->s.nStep = OMX_VERSION_STEP;
    */
    ver->s.nVersionMajor =
specVersion.s.nVersionMajor;
    ver->s.nVersionMinor =
```

```
specVersion.s.nVersionMinor;
    ver->s.nRevision = specVersion.s.nRevision;
    ver->s.nStep = specVersion.s.nStep;
}

void printState() {
    OMX_STATETYPE state;
    err = OMX_GetState(handle, &state);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error on getting
state\n");
    exit(1);
    }
    switch (state) {
    case OMX_StateLoaded: fprintf(stderr,
"StateLoaded\n"); break;
    case OMX_StateIdle: fprintf(stderr,
"StateIdle\n"); break;
    case OMX_StateExecuting: fprintf(stderr,
"StateExecuting\n"); break;
    case OMX_StatePause: fprintf(stderr,
"StatePause\n"); break;
    case OMX_StateWaitForResources:
fprintf(stderr, "StateWiat\n"); break;
    default:  fprintf(stderr, "State
unknown\n"); break;
    }
```

```
}

OMX_ERRORTYPE setEncoding(int portNumber,
OMX_AUDIO_CODINGTYPE encoding) {
    OMX_PARAM_PORTDEFINITIONTYPE sPortDef;

    setHeader(&sPortDef,
sizeof(OMX_PARAM_PORTDEFINITIONTYPE));
    sPortDef.nPortIndex = portNumber;
    sPortDef.nPortIndex = portNumber;
    err = OMX_GetParameter(handle,
OMX_IndexParamPortDefinition, &sPortDef);
    if(err != OMX_ErrorNone){
        fprintf(stderr, "Error in getting
OMX_PORT_DEFINITION_TYPE parameter\n",
 0);
        exit(1);
    }

    sPortDef.format.audio.eEncoding = encoding;
    sPortDef.nBufferCountActual =
sPortDef.nBufferCountMin;

    err = OMX_SetParameter(handle,
OMX_IndexParamPortDefinition, &sPortDef);
    return err;
}
```

```
void getPCMInformation(int indentLevel, int
portNumber) {
    /* assert: PCM is a supported mode */
    OMX_AUDIO_PARAM_PCMMODETYPE sPCMMode;

    /* set it into PCM format before asking for
PCM info */
    if (setEncoding(portNumber,
OMX_AUDIO_CodingPCM) != OMX_ErrorNone) {
    fprintf(stderr, "Error in setting coding to
PCM\n");
    return;
    }

    setHeader(&sPCMMode,
sizeof(OMX_AUDIO_PARAM_PCMMODETYPE));
    sPCMMode.nPortIndex = portNumber;
    err = OMX_GetParameter(handle,
OMX_IndexParamAudioPcm, &sPCMMode);
    if(err != OMX_ErrorNone){
    indent printf("PCM mode unsupported\n");
    } else {
    indent printf("  PCM default sampling rate
%d\n", sPCMMode.nSamplingRate);
    indent printf("  PCM default bits per
sample %d\n", sPCMMode.nBitPerSample);
```

```
    indent printf("  PCM default number of
channels %d\n", sPCMMode.nChannels);
    }

    /*
    setHeader(&sAudioPortFormat,
sizeof(OMX_AUDIO_PARAM_PORTFORMATTYPE));
    sAudioPortFormat.nIndex = 0;
    sAudioPortFormat.nPortIndex = portNumber;
    */



}
void getMP3Information(int indentLevel, int
portNumber) {
    /* assert: MP3 is a supported mode */
    OMX_AUDIO_PARAM_MP3TYPE sMP3Mode;

    /* set it into MP3 format before asking for
MP3 info */
    if (setEncoding(portNumber,
OMX_AUDIO_CodingMP3) != OMX_ErrorNone) {
    fprintf(stderr, "Error in setting coding to
MP3\n");
    return;
    }
```

```
    setHeader(&sMP3Mode,
sizeof(OMX_AUDIO_PARAM_MP3TYPE));
    sMP3Mode.nPortIndex = portNumber;
    err = OMX_GetParameter(handle,
OMX_IndexParamAudioMp3, &sMP3Mode);
    if(err != OMX_ErrorNone){
    indent printf("MP3 mode unsupported\n");
    } else {
    indent printf("  MP3 default sampling rate
%d\n", sMP3Mode.nSampleRate);
    indent printf("  MP3 default bits per
sample %d\n", sMP3Mode.nBitRate);
    indent printf("  MP3 default number of
channels %d\n", sMP3Mode.nChannels);
    }
}


void getSupportedAudioFormats(int indentLevel,
int portNumber) {
    OMX_AUDIO_PARAM_PORTFORMATTYPE
sAudioPortFormat;

    setHeader(&sAudioPortFormat,
sizeof(OMX_AUDIO_PARAM_PORTFORMATTYPE));
    sAudioPortFormat.nIndex = 0;
    sAudioPortFormat.nPortIndex = portNumber;
```

```
#ifdef LIM
    printf("LIM doesn't set audio formats
properly\n");
    return;
#endif

    indent printf("Supported audio formats
are:\n");
    for(;;) {
        err = OMX_GetParameter(handle,
OMX_IndexParamAudioPortFormat,
&sAudioPortFormat);
        if (err == OMX_ErrorNoMore) {
        indent printf("No more formats
supported\n");
        return;
        }

    /* This shouldn't occur, but does with
Broadcom library */
    if (sAudioPortFormat.eEncoding ==
OMX_AUDIO_CodingUnused) {
         indent printf("No coding format
returned\n");
         return;
    }
```

```
    switch (sAudioPortFormat.eEncoding) {
    case OMX_AUDIO_CodingPCM:
        indent printf("Supported encoding is
PCM\n");
        getPCMInformation(indentLevel+1,
portNumber);
        break;
    case OMX_AUDIO_CodingVORBIS:
        indent printf("Supported encoding is
Ogg Vorbis\n");
        break;
    case OMX_AUDIO_CodingMP3:
        indent printf("Supported encoding is
MP3\n");
        getMP3Information(indentLevel+1,
portNumber);
        break;
#ifdef RASPBERRY_PI
    case OMX_AUDIO_CodingFLAC:
        indent printf("Supported encoding is
FLAC\n");
        break;
    case OMX_AUDIO_CodingDDP:
        indent printf("Supported encoding is
DDP\n");
        break;
```

```
    case OMX_AUDIO_CodingDTS:
        indent printf("Supported encoding is
DTS\n");
        break;
    case OMX_AUDIO_CodingWMAPRO:
        indent printf("Supported encoding is
WMAPRO\n");
        break;
    case OMX_AUDIO_CodingATRAC3:
        indent printf("Supported encoding is
ATRAC3\n");
        break;
    case OMX_AUDIO_CodingATRACX:
        indent printf("Supported encoding is
ATRACX\n");
        break;
    case OMX_AUDIO_CodingATRACAAL:
        indent printf("Supported encoding is
ATRACAAL\n");
        break;
#endif
    case OMX_AUDIO_CodingAAC:
        indent printf("Supported encoding is
AAC\n");
        break;
    case OMX_AUDIO_CodingWMA:
        indent printf("Supported encoding is
```

```
WMA\n");
        break;
    case OMX_AUDIO_CodingRA:
        indent printf("Supported encoding is
RA\n");
        break;
    case OMX_AUDIO_CodingAMR:
        indent printf("Supported encoding is
AMR\n");
        break;
    case OMX_AUDIO_CodingEVRC:
        indent printf("Supported encoding is
EVRC\n");
        break;
    case OMX_AUDIO_CodingG726:
        indent printf("Supported encoding is
G726\n");
        break;
    case OMX_AUDIO_CodingMIDI:
        indent printf("Supported encoding is
MIDI\n");
        break;

        /*
    case OMX_AUDIO_Coding:
        indent printf("Supported encoding is
\n");
```

```
        break;
        */
    default:
        indent printf("Supported encoding is
not PCM or MP3 or Vorbis, is 0x%X\n",
          sAudioPortFormat.eEncoding);
    }
        sAudioPortFormat.nIndex++;
    }
}



void getAudioPortInformation(int indentLevel,
int nPort, OMX_PARAM_PORTDEFINITIONTYPE
sPortDef) {
    indent printf("Port %d requires %d
buffers\n", nPort, sPortDef.nBufferCountMin);
    indent printf("Port %d has min buffer size
%d bytes\n", nPort, sPortDef.nBufferSize);

    if (sPortDef.eDir == OMX_DirInput) {
    indent printf("Port %d is an input port\n",
nPort);
    } else {
    indent printf("Port %d is an output
port\n",  nPort);
```

```
    }
    switch (sPortDef.eDomain) {
    case OMX_PortDomainAudio:
    indent printf("Port %d is an audio port\n",
nPort);
    indent printf("Port mimetype %s\n",
           sPortDef.format.audio.cMIMEType);

    switch (sPortDef.format.audio.eEncoding) {
    case OMX_AUDIO_CodingPCM:
        indent printf("Port encoding is
PCM\n");
        break;
    case OMX_AUDIO_CodingVORBIS:
        indent printf("Port encoding is Ogg
Vorbis\n");
        break;
    case OMX_AUDIO_CodingMP3:
        indent printf("Port encoding is
MP3\n");
        break;
    default:
        indent printf("Port encoding is not PCM
or MP3 or Vorbis, is %d\n",
           sPortDef.format.audio.eEncoding);
    }
    getSupportedAudioFormats(indentLevel+1,
```

```
nPort);

    break;
    /* could put other port types here */
    default:
    indent printf("Port %d is not an audio
port\n",  nPort);
    }
}

void getAllAudioPortsInformation(int
indentLevel) {
    OMX_PORT_PARAM_TYPE param;
    OMX_PARAM_PORTDEFINITIONTYPE sPortDef;

    int startPortNumber;
    int nPorts;
    int n;

    setHeader(&param,
sizeof(OMX_PORT_PARAM_TYPE));

    err = OMX_GetParameter(handle,
OMX_IndexParamAudioInit, &param);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error in getting audio
OMX_PORT_PARAM_TYPE parameter\n", 0);
```

```
    return;
    }
    indent printf("Audio ports:\n");
    indentLevel++;

    startPortNumber = param.nStartPortNumber;
    nPorts = param.nPorts;
    if (nPorts == 0) {
    indent printf("No ports of this type\n");
    return;
    }

    indent printf("Ports start on %d\n",
startPortNumber);
    indent printf("There are %d open ports\n",
nPorts);

    for (n = 0; n < nPorts; n++) {
    setHeader(&sPortDef,
sizeof(OMX_PARAM_PORTDEFINITIONTYPE));
    sPortDef.nPortIndex = startPortNumber + n;
    err = OMX_GetParameter(handle,
OMX_IndexParamPortDefinition, &sPortDef);
    if(err != OMX_ErrorNone){
        fprintf(stderr, "Error in getting
OMX_PORT_DEFINITION_TYPE parameter\n", 0);
        exit(1);
```

```
    }
    getAudioPortInformation(indentLevel+1,
startPortNumber + n, sPortDef);
    }
}

void getAllVideoPortsInformation(int
indentLevel) {
    OMX_PORT_PARAM_TYPE param;
    int startPortNumber;
    int nPorts;
    int n;

    setHeader(&param,
sizeof(OMX_PORT_PARAM_TYPE));

    err = OMX_GetParameter(handle,
OMX_IndexParamVideoInit, &param);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error in getting video
OMX_PORT_PARAM_TYPE parameter\n", 0);
    return;
    }
    printf("Video ports:\n");
    indentLevel++;

    startPortNumber = param.nStartPortNumber;
```

```
    nPorts = param.nPorts;
    if (nPorts == 0) {
    indent printf("No ports of this type\n");
    return;
    }

    indent printf("Ports start on %d\n",
startPortNumber);
    indent printf("There are %d open ports\n",
nPorts);
}

void getAllImagePortsInformation(int
indentLevel) {
    OMX_PORT_PARAM_TYPE param;
    int startPortNumber;
    int nPorts;
    int n;

    setHeader(&param,
sizeof(OMX_PORT_PARAM_TYPE));

    err = OMX_GetParameter(handle,
OMX_IndexParamVideoInit, &param);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error in getting image
OMX_PORT_PARAM_TYPE parameter\n", 0);
```

```
    return;
    }
    printf("Image ports:\n");
    indentLevel++;

    startPortNumber = param.nStartPortNumber;
    nPorts = param.nPorts;
    if (nPorts == 0) {
    indent printf("No ports of this type\n");
    return;
    }

    indent printf("Ports start on %d\n",
startPortNumber);
    indent printf("There are %d open ports\n",
nPorts);
}

void getAllOtherPortsInformation(int
indentLevel) {
    OMX_PORT_PARAM_TYPE param;
    int startPortNumber;
    int nPorts;
    int n;

    setHeader(&param,
sizeof(OMX_PORT_PARAM_TYPE));
```

```
    err = OMX_GetParameter(handle,
OMX_IndexParamVideoInit, &param);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error in getting other
OMX_PORT_PARAM_TYPE parameter\n", 0);
    exit(1);
    }
    printf("Other ports:\n");
    indentLevel++;

    startPortNumber = param.nStartPortNumber;
    nPorts = param.nPorts;
    if (nPorts == 0) {
    indent printf("No ports of this type\n");
    return;
    }

    indent printf("Ports start on %d\n",
startPortNumber);
    indent printf("There are %d open ports\n",
nPorts);
}

int main(int argc, char** argv) {

    OMX_PORT_PARAM_TYPE param;
```

```
    OMX_PARAM_PORTDEFINITIONTYPE sPortDef;
    OMX_AUDIO_PORTDEFINITIONTYPE sAudioPortDef;
    OMX_AUDIO_PARAM_PORTFORMATTYPE
sAudioPortFormat;
    OMX_AUDIO_PARAM_PCMMODETYPE sPCMMode;

#ifdef RASPBERRY_PI
    char *componentName =
"OMX.broadcom.audio_mixer";
#endif
#ifdef LIM
    char *componentName =
"OMX.limoi.alsa_sink";
#else
    char *componentName =
"OMX.st.volume.component";
#endif
    unsigned char name[128]; /* spec says 128
is max name length */
    OMX_UUIDTYPE uid;
    int startPortNumber;
    int nPorts;
    int n;

    /* ovveride component name by command line
argument */
    if (argc == 2) {
```

```
    componentName = argv[1];
    }

# ifdef RASPBERRY_PI
    bcm_host_init();
# endif

    err = OMX_Init();
    if(err != OMX_ErrorNone) {
    fprintf(stderr, "OMX_Init() failed\n", 0);
    exit(1);
    }
    /** Ask the core for a handle to the volume
control component
     */
    err = OMX_GetHandle(&handle, componentName,
NULL /*app private data */, &callbacks);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "OMX_GetHandle failed\n",
0);
    exit(1);
    }
    err = OMX_GetComponentVersion(handle, name,
&compVersion, &specVersion, &uid);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "OMX_GetComponentVersion
failed\n", 0);
```

```
    exit(1);
    }
    printf("Component name: %s version %d.%d,
Spec version %d.%d\n",
       name, compVersion.s.nVersionMajor,
       compVersion.s.nVersionMinor,
       specVersion.s.nVersionMajor,
       specVersion.s.nVersionMinor);

    /** Get  ports information */
    getAllAudioPortsInformation(0);
    getAllVideoPortsInformation(0);
    getAllImagePortsInformation(0);
    getAllOtherPortsInformation(0);

    exit(0);
}
```

The Makefile for the Bellagio version is

```
INCLUDES=-I ../libomxil-bellagio-0.9.3/include/
LIBS=-L ../libomxil-bellagio-0.9.3/src/.libs -l
omxil-bellagio
CFLAGS = -g

info: info.c
        cc $(FLAGS) $(INCLUDES) -o info info.c
$(LIBS)
```

The output using the Bellagio implementation is

```
Component name: OMX.st.volume.component version
1.1, Spec version 1.1
Audio ports:
  Ports start on 0
  There are 2 open ports
    Port 0 requires 2 buffers
    Port 0 is an input port
    Port 0 is an audio port
    Port mimetype raw/audio
    Port encoding is not PCM or MP3 or Vorbis,
is 0
      Supported audio formats are:
      No coding format returned
    Port 1 requires 2 buffers
    Port 1 is an output port
    Port 1 is an audio port
    Port mimetype raw/audio
    Port encoding is not PCM or MP3 or Vorbis,
is 0
      Supported audio formats are:
      No coding format returned
Video ports:
  No ports of this type
Image ports:
  No ports of this type
```

```
Other ports:
  No ports of this type
```

The Makefile for the Raspberry Pi is

```
INCLUDES=-I /opt/vc/include/IL -I
/opt/vc/include -I
/opt/vc/include/interface/vcos/pthreads
CFLAGS=-g -DRASPBERRY_PI
LIBS=-L /opt/vc/lib -l openmaxil -l bcm_host

info: info.c
    cc $(CFLAGS) $(INCLUDES) -o info info.c
$(LIBS)
```

The output on the Raspberry Pi for the audio_render component is

```
Audio ports:
  Ports start on 100
  There are 1 open ports
    Port 100 requires 1 buffers
    Port 100 is an input port
    Port 100 is an audio port
    Port mimetype (null)
    Port encoding is PCM
      Supported audio formats are:
      Supported encoding is PCM
          PCM default sampling rate 44100
          PCM default bits per sample 16
          PCM default number of channels 2
      Supported encoding is DDP
      No more formats supported
Video ports:
  No ports of this type
Image ports:
  No ports of this type
Other ports:
  No ports of this type
```

The Makefile for LIM is

```
INCLUDES=-I ../../lim-omx-1.1/LIM/limoi-
core/include/
#LIBS=-L ../../lim-omx-1.1/LIM/limoi-
base/src/.libs -l limoi-base
LIBS = -L /home/newmarch/osm-build/lib/ -l
limoa -l limoi-core
CFLAGS = -g -DLIM

info: info.c
    cc $(CFLAGS) $(INCLUDES) -o info info.c
$(LIBS)
```

The output on LIM for the alsa_sink component is

```
Component name: OMX.limoi.alsa_sink version
0.0, Spec version 1.1
Audio ports:
  Ports start on 0
  There are 1 open ports
    Port 0 requires 2 buffers
    Port 0 is an input port
    Port 0 is an audio port
    Port mimetype (null)
    Port encoding is PCM
LIM doesn't set audio formats properly
Error in getting video OMX_PORT_PARAM_TYPE
parameter
Error in getting image OMX_PORT_PARAM_TYPE
parameter
Error in getting other OMX_PORT_PARAM_TYPE
parameter
```

The LIM implementation throws errors when the component does not support a mode (here an audio component does not support video, image or other). This is against the 1.1 specification which says

```
"All standard components shall support the
following parameters:
  o OMX_IndexParamPortDefinition
  o OMX_IndexParamCompBufferSupplier
  o OMX_IndexParamAudioInit
  o OMX_IndexParamImageInit
  o OMX_IndexParamVideoInit
  o OMX_IndexParamOtherInit"
```

I suppose you could argue that an `alsa_sink` component isn't a standard one, so it is allowed. Well, okay...

Playing PCM audio files

Playing audio to an output device requires use of an "audio_render" device. This is one of the standard devices in the 1.1 specification, and is included in the Broadcom Raspberry Pi library but not in the Bellagio library. LIM has a component "alsa_sink" which plays the same role.

The structure of a program to play audio is

- initialise the library and audio render component
- Continually fill input buffers and ask the component to empty the buffers

- Capture events from the component saying that a buffer has been emptied in order to schedule re-filling the buffer and requesting it to be emptied
- Clean up on completion

Note that the Raspberry Pi audio render component will *only* play PCM data and that the LIM alsa_sink component only plays back at 44,100hz.

State

Initialising the component is a multi-step process that depends on the state of the component. Components are created in the `Loaded` state. They transition from one state to another through an `OMX_SendCommand(handle, OMX_CommandStateSet, <next state>, <param>)`. The next state from `Loaded` should be `Idle` and from there to `Executing`. There are other states which we need not be concerned about.

Requests to change state are asynchronous. The send command returns immediately (well, within 5 milliseconds). When the actual change of state occurs an event handler callback function is called.

Threads

Some commands require a component to be in a particular state. Requests to put a component into a state are asynchronous. So a request can be made by a client but then the client might have to wait until the state change has occurred. This is best done by the client suspending operation of its thread until woken up by the state change occurring in the event handler.

Linux/Unix has standardised on the Posix pthreads library for managing multiple threads. For our purposes we use two parts from this library: the ability to place a *mutex* around critical sections, and the ability to suspend/wake up threads based on *conditions* . Pthreads are covered in many places, with a short and good tutorial by Blaise Barney at POSIX Threads Programming .

The functions and data we use are

```
pthread_mutex_t mutex;
OMX_STATETYPE currentState = OMX_StateLoaded;
pthread_cond_t stateCond;

void waitFor(OMX_STATETYPE state) {
    pthread_mutex_lock(&mutex);
    while (currentState != state)
    pthread_cond_wait(&stateCond, &mutex);
    fprintf(stderr, "Wait successfully
completed\n");
    pthread_mutex_unlock(&mutex);
}

void wakeUp(OMX_STATETYPE newState) {
    pthread_mutex_lock(&mutex);
    currentState = newState;
    pthread_cond_signal(&stateCond);
    pthread_mutex_unlock(&mutex);
}
pthread_mutex_t empty_mutex;
int emptyState = 0;
OMX_BUFFERHEADERTYPE* pEmptyBuffer;
pthread_cond_t emptyStateCond;

void waitForEmpty() {
    pthread_mutex_lock(&empty_mutex);
```

```
    while (emptyState == 1)
    pthread_cond_wait(&emptyStateCond,
&empty_mutex);
    emptyState = 1;
    pthread_mutex_unlock(&empty_mutex);
}

void wakeUpEmpty(OMX_BUFFERHEADERTYPE* pBuffer)
{
    pthread_mutex_lock(&empty_mutex);
    emptyState = 0;
    pEmptyBuffer = pBuffer;
    pthread_cond_signal(&emptyStateCond);
    pthread_mutex_unlock(&empty_mutex);
}

void mutex_init() {
    int n = pthread_mutex_init(&mutex, NULL);
    if ( n != 0) {
    fprintf(stderr, "Can't init state
mutex\n");
    }
    n = pthread_mutex_init(&empty_mutex, NULL);
    if ( n != 0) {
    fprintf(stderr, "Can't init empty
mutex\n");
```

```
    }
}
```

Hungarian notation

Hungarian notation was invented by Charles Simonyi to add type or functional information to variable, structure and field names. A form was heavily used in the Microsoft Windows SDK. A simplified form is used in OpenMAX IL by prefixing variables, fields, etc including the following:

- 'n' prefixes a number of some kind
- 'p' prefixes a pointer
- 's' prefixes a structure or a string
- 'c' prefixes a callback function

The value of such conventions is highly debatable...

Callbacks

There are two types of callback functions relevant to this example: event callbacks which occur on changes of state and some other events, and empty buffer callbacks which occur when a component has emptied an input buffer. These are registered by

```
OMX_CALLBACKTYPE callbacks  = { .EventHandler =
cEventHandler,

.EmptyBufferDone = cEmptyBufferDone,
};
err = OMX_GetHandle(&handle, componentName,
NULL /*app private data */, &callbacks);
```

Component resources

Each component has a number of ports that have to be configured. The ports are some of the component's *resources* . Each port starts off as *enabled* , but may be set to *disabled* by `OMX_SendCommand(handle, OMX_CommandPortDisable, <port number>, NULL)` .

Enabled ports can have *buffers* allocated for transfer of data into and out of the component. This can be done in two ways: `OMX_AllocateBuffer` asks the component to perform the allocation for the client, while with `OMX_UseBuffer` the client hands a buffer to the component. As there may be buffer memory alignment issues, I prefer to let the component do the allocation.

Here is a tricky part. In order to allocate or use buffers on a component, a request must be made to transition from `Loaded` state to `Idle`. So a call to `OMX_SendCommand(handle, OMX_CommandStateSet, OMX_StateIdle, <param>)`. must be made before buffers are allocated. *But* the transition to `Idle` will not take place until each port is either disabled or all buffers for it are allocated.

This last step cost me nearly a week of head scratching. The `audio_render` component has two ports: an input audio port and a time update port. While I had configured the audio port correctly, I had not disabled the time port because I hadn't realised it had one. Consequently the transition to `Idle` never took place... Code to handle this is

```
setHeader(&param, sizeof(OMX_PORT_PARAM_TYPE));
    err = OMX_GetParameter(handle,
OMX_IndexParamOtherInit, &param);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error in getting
OMX_PORT_PARAM_TYPE parameter\n", 0);
    exit(1);
    }
    startPortNumber =
((OMX_PORT_PARAM_TYPE)param).nStartPortNumber;
    nPorts =
((OMX_PORT_PARAM_TYPE)param).nPorts;
    printf("Other has %d ports\n", nPorts);
    /* and disable it */
    err = OMX_SendCommand(handle,
OMX_CommandPortDisable, startPortNumber, NULL);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error on setting port to
disabled\n");
    exit(1);
    }
```

Setting parameters for the audio port is

```
/** Get audio port information */
    setHeader(&param,
sizeof(OMX_PORT_PARAM_TYPE));
    err = OMX_GetParameter(handle,
OMX_IndexParamAudioInit, &param);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error in getting
OMX_PORT_PARAM_TYPE parameter\n", 0);
    exit(1);
    }
    startPortNumber =
((OMX_PORT_PARAM_TYPE)param).nStartPortNumber;
    nPorts =
((OMX_PORT_PARAM_TYPE)param).nPorts;
    if (nPorts > 1) {
    fprintf(stderr, "Render device has more
than one port\n");
    exit(1);
    }

    setHeader(&sPortDef,
sizeof(OMX_PARAM_PORTDEFINITIONTYPE));
    sPortDef.nPortIndex = startPortNumber;
    err = OMX_GetParameter(handle,
OMX_IndexParamPortDefinition, &sPortDef);
    if(err != OMX_ErrorNone){
```

```
    fprintf(stderr, "Error in getting
OMX_PORT_DEFINITION_TYPE parameter\n", 0);
    exit(1);
    }
    if (sPortDef.eDomain !=
OMX_PortDomainAudio) {
    fprintf(stderr, "Port %d is not an audio
port\n", startPortNumber);
    exit(1);
    }

    if (sPortDef.eDir != OMX_DirInput) {
    fprintf(stderr, "Port is not an input
port\n");
    exit(1);
    }
    if (sPortDef.format.audio.eEncoding ==
OMX_AUDIO_CodingPCM) {
    printf("Port encoding is PCM\n");
    }    else {
    printf("Port has unknown encoding\n");
    }

    /* create minimum number of buffers for the
port */
    nBuffers = sPortDef.nBufferCountActual =
sPortDef.nBufferCountMin;
```

```
    printf("Number of bufers is %d\n",
nBuffers);
    err = OMX_SetParameter(handle,
OMX_IndexParamPortDefinition, &sPortDef);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error in setting
OMX_PORT_PARAM_TYPE parameter\n", 0);
    exit(1);
    }

    /* call to put state into idle before
allocating buffers */
    err = OMX_SendCommand(handle,
OMX_CommandStateSet, OMX_StateIdle, NULL);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error on setting state to
idle\n");
    exit(1);
    }

    err = OMX_SendCommand(handle,
OMX_CommandPortEnable, startPortNumber, NULL);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error on setting port to
enabled\n");
    exit(1);
    }
```

```
    nBufferSize = sPortDef.nBufferSize;
    printf("%d buffers of size is %d\n",
nBuffers, nBufferSize);

    inBuffers = malloc(nBuffers *
sizeof(OMX_BUFFERHEADERTYPE *));
    if (inBuffers == NULL) {
    fprintf(stderr, "Can't allocate
buffers\n");
    exit(1);
    }
    for (n = 0; n < nBuffers; n++) {
        err = OMX_AllocateBuffer(handle,
inBuffers+n, startPortNumber, NULL,
                                nBufferSize);
    if (err != OMX_ErrorNone) {
        fprintf(stderr, "Error on
AllocateBuffer in 1%i\n", err);
          exit(1);
    }

    }

    waitFor(OMX_StateIdle);
    /* try setting the encoding to PCM mode */
    setHeader(&sPCMMode,
```

```
sizeof(OMX_AUDIO_PARAM_PCMMODETYPE));
    sPCMMode.nPortIndex = startPortNumber;
    err = OMX_GetParameter(handle,
OMX_IndexParamAudioPcm, &sPCMMode);
    if(err != OMX_ErrorNone){
    printf("PCM mode unsupported\n");
    exit(1);
    } else {
    printf("PCM mode supported\n");
    printf("PCM sampling rate %d\n",
sPCMMode.nSamplingRate);
    printf("PCM nChannels %d\n",
sPCMMode.nChannels);
    }
```

Setting the output device

OpenMAX has a standard audio render component. But what device does it render to? The inbuilt sound card? A USB sound card? That is not a part of OpenMAX IL - there isn't even a way to list the audio devices - only the audio components.

OpenMAX has an extension mechanism which can be used by an OpenMAX implementor to answer questions like this. The Broadcom core implementation has extension types

`OMX_CONFIG_BRCMAUDIODESTINATIONTYPE` (and `OMX_CONFIG_BRCMAUDIOSOURCETYPE`) which can be used to set the audio destination (source) device. Code to do this is

```
void setOutputDevice(const char *name) {
   int32_t success = -1;
   OMX_CONFIG_BRCMAUDIODESTINATIONTYPE arDest;

   if (name && strlen(name) <
sizeof(arDest.sName)) {
       setHeader(&arDest,
sizeof(OMX_CONFIG_BRCMAUDIODESTINATIONTYPE));
       strcpy((char *)arDest.sName, name);

       err = OMX_SetParameter(handle,
OMX_IndexConfigBrcmAudioDestination, &arDest);
       if (err != OMX_ErrorNone) {
       fprintf(stderr, "Error on setting audio
destination\n");
       exit(1);
       }
   }
}
```

Here is where it descends into murkiness again: the header file

states that the default value of `sName` is "local" but doesn't give any other values. The Raspberry Pi forums say that this refers to the 3.5mm analog audio out, and that HDMI is chosen by using the value "hdmi". No other values are documented, and it seems that the Broadcom OpenMAX IL does not support any other audio devices: in particular, USB audio devices are not supported by the current Broadcom OpenMAX IL components for either input or output. So you can't use OpenMAX IL for say audio capture on the Raspberry Pi since it has no Broadcom supported audio input.

Main loop

Playing the audio file once all the ports are set up consists of filling buffers, waiting for them to empty and then re-filling them until the data is finished. There are two possible styles:

- fill the buffers once in the main loop and then continue to fill and empty them in the empty buffer callbacks
- in the main loop, fill and empty the buffers continually, waiting between each fill for the buffer to empty

The Bellagio example use the first technique. However, the 1.2 specification says that "...the IL client shall not call IL core or component functions from within an IL callback context" so this

is not a good technique. The Raspberry Pi examples use the second technique, but use a non-standard call to find the latency and delay for that time. It is better to just set up more pthreads conditions and block on those.

This leads to a main loop of

```
emptyState = 1;
    for (;;) {
    int data_read = read(fd, inBuffers[0]-
>pBuffer, nBufferSize);
    inBuffers[0]->nFilledLen = data_read;
    inBuffers[0]->nOffset = 0;
    filesize -= data_read;
    if (data_read <= 0) {
        fprintf(stderr, "In the %s no more
input data available\n", __func__);
        inBuffers[0]->nFilledLen=0;
        inBuffers[0]->nFlags =
OMX_BUFFERFLAG_EOS;
        bEOS=OMX_TRUE;
        err = OMX_EmptyThisBuffer(handle,
inBuffers[0]);
        break;
    }
    if(!bEOS) {
        fprintf(stderr, "Emptying again buffer
%p %d bytes, %d to go\n", inBuffers[0],
data_read, filesize);
        err = OMX_EmptyThisBuffer(handle,
inBuffers[0]);
    }else {
        fprintf(stderr, "In %s Dropping Empty
```

```
This buffer to Audio Dec\n", __func__);
    }
    waitForEmpty();
    printf("Waited for empty\n");
    }

    printf("Buffers emptied\n");
```

Complete program

The complete program is

```
/**
   Based on code
   Copyright (C) 2007-2009 STMicroelectronics
   Copyright (C) 2007-2009 Nokia Corporation
and/or its subsidiary(-ies).
   under the LGPL
*/

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/stat.h>
#include <pthread.h>

#include <OMX_Core.h>
#include <OMX_Component.h>
#include <OMX_Types.h>
#include <OMX_Audio.h>

#ifdef RASPBERRY_PI
#include <bcm_host.h>
#include <IL/OMX_Broadcom.h>
#endif
```

```
OMX_ERRORTYPE err;
OMX_HANDLETYPE handle;
OMX_VERSIONTYPE specVersion, compVersion;

int fd = 0;
unsigned int filesize;
static OMX_BOOL bEOS=OMX_FALSE;

OMX_U32 nBufferSize;
int nBuffers;

pthread_mutex_t mutex;
OMX_STATETYPE currentState = OMX_StateLoaded;
pthread_cond_t stateCond;

void waitFor(OMX_STATETYPE state) {
    pthread_mutex_lock(&mutex);
    while (currentState != state)
    pthread_cond_wait(&stateCond, &mutex);
    pthread_mutex_unlock(&mutex);
}

void wakeUp(OMX_STATETYPE newState) {
    pthread_mutex_lock(&mutex);
    currentState = newState;
    pthread_cond_signal(&stateCond);
```

```
    pthread_mutex_unlock(&mutex);
}

pthread_mutex_t empty_mutex;
int emptyState = 0;
OMX_BUFFERHEADERTYPE* pEmptyBuffer;
pthread_cond_t emptyStateCond;

void waitForEmpty() {
    pthread_mutex_lock(&empty_mutex);
    while (emptyState == 1)
    pthread_cond_wait(&emptyStateCond,
&empty_mutex);
    emptyState = 1;
    pthread_mutex_unlock(&empty_mutex);
}

void wakeUpEmpty(OMX_BUFFERHEADERTYPE* pBuffer)
{
    pthread_mutex_lock(&empty_mutex);
    emptyState = 0;
    pEmptyBuffer = pBuffer;
    pthread_cond_signal(&emptyStateCond);
    pthread_mutex_unlock(&empty_mutex);
}

void mutex_init() {
```

```
    int n = pthread_mutex_init(&mutex, NULL);
    if ( n != 0) {
    fprintf(stderr, "Can't init state
mutex\n");
    }
    n = pthread_mutex_init(&empty_mutex, NULL);
    if ( n != 0) {
    fprintf(stderr, "Can't init empty
mutex\n");
    }
}

static void display_help() {
    fprintf(stderr, "Usage: render
input_file");
}



/** Gets the file descriptor's size
 * @return the size of the file. If size cannot
be computed
 * (i.e. stdin, zero is returned)
 */
static int getFileSize(int fd) {

    struct stat input_file_stat;
    int err;
```

```
    /* Obtain input file length */
    err = fstat(fd, &input_file_stat);
    if(err){
    fprintf(stderr, "fstat failed",0);
    exit(-1);
    }
    return input_file_stat.st_size;
}

OMX_ERRORTYPE cEventHandler(
                OMX_HANDLETYPE hComponent,
                OMX_PTR pAppData,
                OMX_EVENTTYPE eEvent,
                OMX_U32 Data1,
                OMX_U32 Data2,
                OMX_PTR pEventData) {

    fprintf(stderr, "Hi there, I am in the %s
callback\n", __func__);
    if(eEvent == OMX_EventCmdComplete) {
    if (Data1 == OMX_CommandStateSet) {
        fprintf(stderr, "Component State
changed in ", 0);
        switch ((int)Data2) {
        case OMX_StateInvalid:
        fprintf(stderr, "OMX_StateInvalid\n",
```

```
0);
        break;
        case OMX_StateLoaded:
        fprintf(stderr, "OMX_StateLoaded\n",
0);
        break;
        case OMX_StateIdle:
        fprintf(stderr, "OMX_StateIdle\n",0);
        break;
        case OMX_StateExecuting:
        fprintf(stderr,
"OMX_StateExecuting\n",0);
        break;
        case OMX_StatePause:
        fprintf(stderr, "OMX_StatePause\n",0);
        break;
        case OMX_StateWaitForResources:
        fprintf(stderr,
"OMX_StateWaitForResources\n",0);
        break;
        }
        wakeUp((int) Data2);
    } else  if (Data1 == OMX_CommandPortEnable)
{

    } else if (Data1 == OMX_CommandPortDisable)
{
```

```
    }
    } else if(eEvent == OMX_EventBufferFlag) {
    if((int)Data2 == OMX_BUFFERFLAG_EOS) {

    }
    } else {
    fprintf(stderr, "Param1 is %i\n",
(int)Data1);
    fprintf(stderr, "Param2 is %i\n",
(int)Data2);
    }

    return OMX_ErrorNone;
}

OMX_ERRORTYPE cEmptyBufferDone(
                   OMX_HANDLETYPE hComponent,
                   OMX_PTR pAppData,
                   OMX_BUFFERHEADERTYPE*
pBuffer) {

    fprintf(stderr, "Hi there, I am in the %s
callback.\n", __func__);
    if (bEOS) {
    fprintf(stderr, "Buffers emptied,
exiting\n");
```

```
    }
    wakeUpEmpty(pBuffer);
    fprintf(stderr, "Exiting callback\n");

    return OMX_ErrorNone;
}

OMX_CALLBACKTYPE callbacks  = { .EventHandler =
cEventHandler,
                .EmptyBufferDone =
cEmptyBufferDone,
};

void printState() {
    OMX_STATETYPE state;
    err = OMX_GetState(handle, &state);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error on getting
state\n");
    exit(1);
    }
    switch (state) {
    case OMX_StateLoaded: fprintf(stderr,
"StateLoaded\n"); break;
    case OMX_StateIdle: fprintf(stderr,
"StateIdle\n"); break;
    case OMX_StateExecuting: fprintf(stderr,
```

```
"StateExecuting\n"); break;
    case OMX_StatePause: fprintf(stderr,
"StatePause\n"); break;
    case OMX_StateWaitForResources:
fprintf(stderr, "StateWiat\n"); break;
    default:  fprintf(stderr, "State
unknown\n"); break;
    }
}


static void setHeader(OMX_PTR header, OMX_U32
size) {
    /* header->nVersion */
    OMX_VERSIONTYPE* ver = (OMX_VERSIONTYPE*)
(header + sizeof(OMX_U32));
    /* header->nSize */
    *((OMX_U32*)header) = size;

    /* for 1.2
       ver->s.nVersionMajor =
OMX_VERSION_MAJOR;
       ver->s.nVersionMinor =
OMX_VERSION_MINOR;
       ver->s.nRevision = OMX_VERSION_REVISION;
       ver->s.nStep = OMX_VERSION_STEP;
    */
```

```
    ver->s.nVersionMajor =
specVersion.s.nVersionMajor;
    ver->s.nVersionMinor =
specVersion.s.nVersionMinor;
    ver->s.nRevision = specVersion.s.nRevision;
    ver->s.nStep = specVersion.s.nStep;
}

/**
 * Disable unwanted ports, or we can't
transition to Idle state
 */
void disablePort(OMX_INDEXTYPE paramType) {
    OMX_PORT_PARAM_TYPE param;
    int nPorts;
    int startPortNumber;
    int n;

    setHeader(&param,
sizeof(OMX_PORT_PARAM_TYPE));
    err = OMX_GetParameter(handle, paramType,
&param);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error in getting
OMX_PORT_PARAM_TYPE parameter\n", 0);
    exit(1);
    }
```

```
    startPortNumber =
((OMX_PORT_PARAM_TYPE)param).nStartPortNumber;
    nPorts =
((OMX_PORT_PARAM_TYPE)param).nPorts;
    if (nPorts > 0) {
    fprintf(stderr, "Other has %d ports\n",
nPorts);
    /* and disable it */
    for (n = 0; n < nPorts; n++) {
        err = OMX_SendCommand(handle,
OMX_CommandPortDisable, n + startPortNumber,
NULL);
        if (err != OMX_ErrorNone) {
        fprintf(stderr, "Error on setting port
to disabled\n");
        exit(1);
        }
    }
    }
}

#ifdef RASPBERRY_PI
/* For the RPi name can be "hdmi" or "local" */
void setOutputDevice(const char *name) {
   int32_t success = -1;
   OMX_CONFIG_BRCMAUDIODESTINATIONTYPE arDest;
```

```
   if (name && strlen(name) <
sizeof(arDest.sName)) {
       setHeader(&arDest,
sizeof(OMX_CONFIG_BRCMAUDIODESTINATIONTYPE));
       strcpy((char *)arDest.sName, name);

       err = OMX_SetParameter(handle,
OMX_IndexConfigBrcmAudioDestination, &arDest);
       if (err != OMX_ErrorNone) {
       fprintf(stderr, "Error on setting audio
destination\n");
       exit(1);
       }
   }
}
#endif

void setPCMMode(int startPortNumber) {
    OMX_AUDIO_PARAM_PCMMODETYPE sPCMMode;

    setHeader(&sPCMMode,
sizeof(OMX_AUDIO_PARAM_PCMMODETYPE));
    sPCMMode.nPortIndex = startPortNumber;
    sPCMMode.nSamplingRate = 48000;
    sPCMMode.nChannels;

    err = OMX_SetParameter(handle,
```

```
OMX_IndexParamAudioPcm, &sPCMMode);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "PCM mode unsupported\n");
    return;
    } else {
    fprintf(stderr, "PCM mode supported\n");
    fprintf(stderr, "PCM sampling rate %d\n",
sPCMMode.nSamplingRate);
    fprintf(stderr, "PCM nChannels %d\n",
sPCMMode.nChannels);
    }
}

int main(int argc, char** argv) {

    OMX_PORT_PARAM_TYPE param;
    OMX_PARAM_PORTDEFINITIONTYPE sPortDef;
    OMX_AUDIO_PORTDEFINITIONTYPE sAudioPortDef;
    OMX_AUDIO_PARAM_PORTFORMATTYPE
sAudioPortFormat;
    OMX_AUDIO_PARAM_PCMMODETYPE sPCMMode;
    OMX_BUFFERHEADERTYPE **inBuffers;

#ifdef RASPBERRY_PI
    char *componentName =
"OMX.broadcom.audio_render";
#endif
```

```
#ifdef LIM
    char *componentName =
"OMX.limoi.alsa_sink";
#endif
    unsigned char
name[OMX_MAX_STRINGNAME_SIZE];
    OMX_UUIDTYPE uid;
    int startPortNumber;
    int nPorts;
    int n;

# ifdef RASPBERRY_PI
    bcm_host_init();
# endif

    fprintf(stderr, "Thread id is %p\n",
pthread_self());
    if(argc < 2){
    display_help();
    exit(1);
    }

    fd = open(argv[1], O_RDONLY);
    if(fd < 0){
    perror("Error opening input file\n");
    exit(1);
    }
```

```
    filesize = getFileSize(fd);



    err = OMX_Init();
    if(err != OMX_ErrorNone) {
    fprintf(stderr, "OMX_Init() failed\n", 0);
    exit(1);
    }
    /** Ask the core for a handle to the audio
render component
     */
    err = OMX_GetHandle(&handle, componentName,
NULL /*app private data */, &callbacks);
    if(err != OMX_ErrorNone) {
    fprintf(stderr, "OMX_GetHandle failed\n",
0);
    exit(1);
    }
    err = OMX_GetComponentVersion(handle, name,
&compVersion, &specVersion, &uid);
    if(err != OMX_ErrorNone) {
    fprintf(stderr, "OMX_GetComponentVersion
failed\n", 0);
    exit(1);
    }

    /** disable other ports */
```

```
    disablePort(OMX_IndexParamOtherInit);

    /** Get audio port information */
    setHeader(&param,
sizeof(OMX_PORT_PARAM_TYPE));
    err = OMX_GetParameter(handle,
OMX_IndexParamAudioInit, &param);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error in getting
OMX_PORT_PARAM_TYPE parameter\n", 0);
    exit(1);
    }
    startPortNumber =
((OMX_PORT_PARAM_TYPE)param).nStartPortNumber;
    nPorts =
((OMX_PORT_PARAM_TYPE)param).nPorts;
    if (nPorts > 1) {
    fprintf(stderr, "Render device has more
than one port\n");
    exit(1);
    }

    /* Get and check port information */
    setHeader(&sPortDef,
sizeof(OMX_PARAM_PORTDEFINITIONTYPE));
    sPortDef.nPortIndex = startPortNumber;
    err = OMX_GetParameter(handle,
```

```
OMX_IndexParamPortDefinition, &sPortDef);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error in getting
OMX_PORT_DEFINITION_TYPE parameter\n", 0);
    exit(1);
    }
    if (sPortDef.eDomain !=
OMX_PortDomainAudio) {
    fprintf(stderr, "Port %d is not an audio
port\n", startPortNumber);
    exit(1);
    }

    if (sPortDef.eDir != OMX_DirInput) {
    fprintf(stderr, "Port is not an input
port\n");
    exit(1);
    }
    if (sPortDef.format.audio.eEncoding ==
OMX_AUDIO_CodingPCM) {
    fprintf(stderr, "Port encoding is PCM\n");
    }    else {
    fprintf(stderr, "Port has unknown
encoding\n");
    }

    /* Create minimum number of buffers for the
```

```
port */
    nBuffers = sPortDef.nBufferCountActual =
sPortDef.nBufferCountMin;
    fprintf(stderr, "Number of bufers is %d\n",
nBuffers);
    err = OMX_SetParameter(handle,
OMX_IndexParamPortDefinition, &sPortDef);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error in setting
OMX_PORT_PARAM_TYPE parameter\n", 0);
    exit(1);
    }
    if (sPortDef.bEnabled) {
    fprintf(stderr, "Port is enabled\n");
    } else {
    fprintf(stderr, "Port is not enabled\n");
    }

    /* call to put state into idle before
allocating buffers */
    err = OMX_SendCommand(handle,
OMX_CommandStateSet, OMX_StateIdle, NULL);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error on setting state to
idle\n");
    exit(1);
    }
```

```
    err = OMX_SendCommand(handle,
OMX_CommandPortEnable, startPortNumber, NULL);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error on setting port to
enabled\n");
    exit(1);
    }

    /* Configure buffers for the port */
    nBufferSize = sPortDef.nBufferSize;
    fprintf(stderr, "%d buffers of size is
%d\n", nBuffers, nBufferSize);

    inBuffers = malloc(nBuffers *
sizeof(OMX_BUFFERHEADERTYPE *));
    if (inBuffers == NULL) {
    fprintf(stderr, "Can't allocate
buffers\n");
    exit(1);
    }

    for (n = 0; n < nBuffers; n++) {
    err = OMX_AllocateBuffer(handle,
inBuffers+n, startPortNumber, NULL,
                    nBufferSize);
    if (err != OMX_ErrorNone) {
```

```
        fprintf(stderr, "Error on
AllocateBuffer in 1%i\n", err);
        exit(1);
    }
    }
    /* Make sure we've reached Idle state */
    waitFor(OMX_StateIdle);

    /* Now try to switch to Executing state */
    err = OMX_SendCommand(handle,
OMX_CommandStateSet, OMX_StateExecuting, NULL);
    if(err != OMX_ErrorNone){
    exit(1);
    }

    /* One buffer is the minimum for Broadcom
component, so use that */
    pEmptyBuffer = inBuffers[0];
    emptyState = 1;
    /* Fill and empty buffer */
    for (;;) {
    int data_read = read(fd, pEmptyBuffer-
>pBuffer, nBufferSize);
    pEmptyBuffer->nFilledLen = data_read;
    pEmptyBuffer->nOffset = 0;
    filesize -= data_read;
    if (data_read <= 0) {
```

```
        fprintf(stderr, "In the %s no more
input data available\n", __func__);
        pEmptyBuffer->nFilledLen=0;
        pEmptyBuffer->nFlags =
OMX_BUFFERFLAG_EOS;
        bEOS=OMX_TRUE;
    }
    fprintf(stderr, "Emptying again buffer %p
%d bytes, %d to go\n", pEmptyBuffer, data_read,
filesize);
    err = OMX_EmptyThisBuffer(handle,
pEmptyBuffer);
    waitForEmpty();
    fprintf(stderr, "Waited for empty\n");
    if (bEOS) {
        fprintf(stderr, "Exiting loop\n");
        break;
    }
    }
    fprintf(stderr, "Buffers emptied\n");
    exit(0);
}
```

Ogg Vorbis

A typical use of an audio system is to play files containing audio data to some output device. Ogg Vorbis is an open, patent free system that defines the Vorbis codec to compress raw data into smaller streams, and the Ogg container format to frame the data and make it easier for applications to manage.

An "Ogg Vorbis" file is an Ogg container of Vorbis encoded data. Playing such a file consists of reading the file, extracting the Vorbis stream from the Ogg frames, decoding the compressed Vorbis stream into, say, a PCM stream and then handing that to some renderer that will play the PCM stream on an output device.

Decoding an Ogg Vorbis file to a PCM stream

There are no standard OpenMAX IL components to decode Ogg frame data. The `audio_decode` component *may* decode a Vorbis stream, but to use that would first require extracting the Vorbis stream from the Ogg frames. We consider how to do that in a later section.

The LIM package has a non-standard component `OMX.limoi.ogg_dec` which plays the role `audio_decoder_with_framing.ogg`. This can directly manage Ogg frames of Vorbis data, and decode it into a PCM stream.

The OpenMAX IL specification shows in Figure 2.5 that in an "in-context" framework where the component runs in the same thread as the application, then the callback must complete before a call to OMX_FillThis Buffer or to OMX_EmptyThisBuffer returns. So the calback should not make further calls to OMX_FillThis Buffer or to OMX_EmptyThisBuffer or it will just build up a long stack of uncompleted calls. This won't happen if they run in separate threads, so the calls coud be okay.

The Broadcom, LIM and Bellagio packages all use separate threads, and all their examples make re-entrant calls. It's easier to write the code that way. So we follow that, as we don't have any example packages where such calls should not be made.

With those provisos in mind, the code is fairly straightforward: when an input buffer is emptied, fill it and empty it again, and when an output buffer is filled, empty it and fill it again. The program is `ogg_decode.c`. it reads from an Ogg Vorbis file and saves in a PCM file.

```
/**
   Based on code
   Copyright (C) 2007-2009 STMicroelectronics
   Copyright (C) 2007-2009 Nokia Corporation
and/or its subsidiary(-ies).
   under the LGPL
*/

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/stat.h>
#include <pthread.h>

#include <OMX_Core.h>
#include <OMX_Component.h>
#include <OMX_Types.h>
#include <OMX_Audio.h>


#ifdef LIM
    //char *componentName =
"OMX.limoi.ffmpeg.decode.audio";
```

```
    char *componentName = "OMX.limoi.ogg_dec";
#endif

OMX_ERRORTYPE err;
OMX_HANDLETYPE handle;
OMX_VERSIONTYPE specVersion, compVersion;

int inFd = 0;
int outFd = 0;
unsigned int filesize;
static OMX_BOOL bEOS=OMX_FALSE;

OMX_U32 inBufferSize;
OMX_U32 outBufferSize;
int numInBuffers;
int numOutBuffers;

pthread_mutex_t mutex;
OMX_STATETYPE currentState = OMX_StateLoaded;
pthread_cond_t stateCond;

void waitFor(OMX_STATETYPE state) {
    pthread_mutex_lock(&mutex);
    while (currentState != state)
    pthread_cond_wait(&stateCond, &mutex);
    fprintf(stderr, "Wait successfully
completed\n");
```

```
    pthread_mutex_unlock(&mutex);
}

void wakeUp(OMX_STATETYPE newState) {
    pthread_mutex_lock(&mutex);
    currentState = newState;
    pthread_cond_signal(&stateCond);
    pthread_mutex_unlock(&mutex);
}
pthread_mutex_t empty_mutex;
int emptyState = 0;
OMX_BUFFERHEADERTYPE* pEmptyBuffer;
pthread_cond_t emptyStateCond;

void waitForEmpty() {
    pthread_mutex_lock(&empty_mutex);
    while (emptyState == 1)
    pthread_cond_wait(&emptyStateCond,
&empty_mutex);
    emptyState = 1;
    pthread_mutex_unlock(&empty_mutex);
}

void wakeUpEmpty(OMX_BUFFERHEADERTYPE* pBuffer)
{
    pthread_mutex_lock(&empty_mutex);
    emptyState = 0;
```

```
    pEmptyBuffer = pBuffer;
    pthread_cond_signal(&emptyStateCond);
    pthread_mutex_unlock(&empty_mutex);
}

void mutex_init() {
    int n = pthread_mutex_init(&mutex, NULL);
    if ( n != 0) {
    fprintf(stderr, "Can't init state
mutex\n");
    }
    n = pthread_mutex_init(&empty_mutex, NULL);
    if ( n != 0) {
    fprintf(stderr, "Can't init empty
mutex\n");
    }
}

static void display_help() {
    fprintf(stderr, "Usage: ogg_decode
input_file output_file");
}



/** Gets the file descriptor's size
 * @return the size of the file. If size cannot
be computed
```

```
 * (i.e. stdin, zero is returned)
 */
static int getFileSize(int fd) {

    struct stat input_file_stat;
    int err;

    /* Obtain input file length */
    err = fstat(fd, &input_file_stat);
    if(err){
    fprintf(stderr, "fstat failed",0);
    exit(-1);
    }
    return input_file_stat.st_size;
}

OMX_ERRORTYPE cEventHandler(
                OMX_HANDLETYPE hComponent,
                OMX_PTR pAppData,
                OMX_EVENTTYPE eEvent,
                OMX_U32 Data1,
                OMX_U32 Data2,
                OMX_PTR pEventData) {

    fprintf(stderr, "Hi there, I am in the %s
callback\n", __func__);
    if(eEvent == OMX_EventCmdComplete) {
```

```
    if (Data1 == OMX_CommandStateSet) {
        fprintf(stderr, "Component State
changed in ", 0);
        switch ((int)Data2) {
        case OMX_StateInvalid:
        fprintf(stderr, "OMX_StateInvalid\n",
0);
        break;
        case OMX_StateLoaded:
        fprintf(stderr, "OMX_StateLoaded\n",
0);
        break;
        case OMX_StateIdle:
        fprintf(stderr, "OMX_StateIdle\n",0);
        break;
        case OMX_StateExecuting:
        fprintf(stderr,
"OMX_StateExecuting\n",0);
        break;
        case OMX_StatePause:
        fprintf(stderr, "OMX_StatePause\n",0);
        break;
        case OMX_StateWaitForResources:
        fprintf(stderr,
"OMX_StateWaitForResources\n",0);
        break;
        }
```

```
        wakeUp((int) Data2);
    } else  if (Data1 == OMX_CommandPortEnable)
{

    } else if (Data1 == OMX_CommandPortDisable)
{

    }
    } else if(eEvent == OMX_EventBufferFlag) {
    if((int)Data2 == OMX_BUFFERFLAG_EOS) {

    }
    } else {
    fprintf(stderr, "Event is %i\n",
(int)eEvent);
    fprintf(stderr, "Param1 is %i\n",
(int)Data1);
    fprintf(stderr, "Param2 is %i\n",
(int)Data2);
    }

    return OMX_ErrorNone;
}

OMX_ERRORTYPE cEmptyBufferDone(
                    OMX_HANDLETYPE hComponent,
                    OMX_PTR pAppData,
```

```
                    OMX_BUFFERHEADERTYPE*
pBuffer) {

    printf("Hi there, I am in the %s callback
with buffer %p.\n", __func__, pBuffer);
    if (bEOS) {
    printf("Buffers emptied, exiting\n");
    wakeUp(OMX_StateLoaded);
    exit(0);
    }

    int data_read = read(inFd, pBuffer-
>pBuffer, inBufferSize);
    if (data_read <= 0) {
    bEOS = 1;
    }
    printf("  filled buffer with %d\n",
data_read);
    pBuffer->nFilledLen = data_read;
    err = OMX_EmptyThisBuffer(handle, pBuffer);

    return OMX_ErrorNone;
}

OMX_ERRORTYPE cFillBufferDone(
                  OMX_HANDLETYPE hComponent,
                  OMX_PTR pAppData,
```

```
                    OMX_BUFFERHEADERTYPE*
pBuffer) {

    fprintf(stderr, "Hi there, I am in the %s
callback with buffer %pn", __func__, pBuffer);
    if (bEOS) {
    fprintf(stderr, "Buffers filled,
exiting\n");
    }
    fprintf(stderr, " writing %d bytes\n",
pBuffer->nFilledLen);
    write(outFd, pBuffer->pBuffer, pBuffer-
>nFilledLen);
    pBuffer->nFilledLen = 0;
    err = OMX_FillThisBuffer(handle, pBuffer);

    return OMX_ErrorNone;
}


OMX_CALLBACKTYPE callbacks  = { .EventHandler =
cEventHandler,
                .EmptyBufferDone =
cEmptyBufferDone,
                .FillBufferDone =
cFillBufferDone
};
```

```
void printState() {
    OMX_STATETYPE state;
    err = OMX_GetState(handle, &state);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error on getting
state\n");
    exit(1);
    }
    switch (state) {
    case OMX_StateLoaded: fprintf(stderr,
"StateLoaded\n"); break;
    case OMX_StateIdle: fprintf(stderr,
"StateIdle\n"); break;
    case OMX_StateExecuting: fprintf(stderr,
"StateExecuting\n"); break;
    case OMX_StatePause: fprintf(stderr,
"StatePause\n"); break;
    case OMX_StateWaitForResources:
fprintf(stderr, "StateWiat\n"); break;
    default:  fprintf(stderr, "State
unknown\n"); break;
    }
}


static void setHeader(OMX_PTR header, OMX_U32
```

```
size) {
    /* header->nVersion */
    OMX_VERSIONTYPE* ver = (OMX_VERSIONTYPE*)
(header + sizeof(OMX_U32));
    /* header->nSize */
    *((OMX_U32*)header) = size;

    /* for 1.2
       ver->s.nVersionMajor =
OMX_VERSION_MAJOR;
       ver->s.nVersionMinor =
OMX_VERSION_MINOR;
       ver->s.nRevision = OMX_VERSION_REVISION;
       ver->s.nStep = OMX_VERSION_STEP;
    */
    ver->s.nVersionMajor =
specVersion.s.nVersionMajor;
    ver->s.nVersionMinor =
specVersion.s.nVersionMinor;
    ver->s.nRevision = specVersion.s.nRevision;
    ver->s.nStep = specVersion.s.nStep;
}

void setPCMMode(int portNumber) {
    OMX_AUDIO_PARAM_PCMMODETYPE pcm;
    int num_channels = 2;//1;
    int sample_rate = 0;//44100;
```

```
    int bit_depth = 0;//16;

    setHeader(&pcm,
sizeof(OMX_AUDIO_PARAM_PCMMODETYPE));
    pcm.nPortIndex = portNumber;

    err = OMX_GetParameter(handle,
OMX_IndexParamAudioPcm, &pcm);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error getting PCM mode on
port %d\n", portNumber);
    exit(1);
    }

    err = OMX_SetParameter(handle,
OMX_IndexParamAudioPcm, &pcm);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error setting PCM mode on
port %d\n", portNumber);
    exit(1);
    }
}

void setEncoding(int portNumber,
OMX_AUDIO_CODINGTYPE encoding) {
    OMX_AUDIO_PARAM_PORTFORMATTYPE
sAudioPortFormat;
```

```
    setHeader(&sAudioPortFormat,
sizeof(OMX_AUDIO_PARAM_PORTFORMATTYPE));
    sAudioPortFormat.nIndex = 0;
    sAudioPortFormat.nPortIndex = portNumber;

    err = OMX_GetParameter(handle,
OMX_IndexParamAudioPortFormat,
&sAudioPortFormat);
    if (err == OMX_ErrorNoMore) {
    printf("Can't get format\n");
    return;
    }
    sAudioPortFormat.eEncoding = encoding;
//OMX_AUDIO_CodingPcm;
    err = OMX_SetParameter(handle,
OMX_IndexParamAudioPortFormat,
&sAudioPortFormat);
    if (err == OMX_ErrorNoMore) {
    printf("Can't set format\n");
    return;
    }
    printf("Set format on port %d to PCM\n",
portNumber);
}
```

```
int setNumBuffers(int portNumber) {
    /* Get and check input port information */
    OMX_PARAM_PORTDEFINITIONTYPE sPortDef;
    int nBuffers;

    setHeader(&sPortDef,
sizeof(OMX_PARAM_PORTDEFINITIONTYPE));
    sPortDef.nPortIndex = portNumber;
    err = OMX_GetParameter(handle,
OMX_IndexParamPortDefinition, &sPortDef);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error in getting
OMX_PORT_DEFINITION_TYPE parameter\n", 0);
    exit(1);
    }

    /* Create minimum number of buffers for the
port */
    nBuffers = sPortDef.nBufferCountActual =
sPortDef.nBufferCountMin;
    fprintf(stderr, "Minimum number of buffers
is %d\n", nBuffers);
    err = OMX_SetParameter(handle,
OMX_IndexParamPortDefinition, &sPortDef);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error in setting
OMX_PORT_PARAM_TYPE parameter\n", 0);
```

```
    exit(1);
    }
    return nBuffers;
}

void createMinBuffers(int portNumber, OMX_U32
*pBufferSize, OMX_BUFFERHEADERTYPE
***pppBuffers) {
    OMX_PARAM_PORTDEFINITIONTYPE sPortDef;
    OMX_BUFFERHEADERTYPE **ppBuffers;
    int n;
    int nBuffers;

    setHeader(&sPortDef,
sizeof(OMX_PARAM_PORTDEFINITIONTYPE));
    sPortDef.nPortIndex = portNumber;
    err = OMX_GetParameter(handle,
OMX_IndexParamPortDefinition, &sPortDef);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error in getting
OMX_PORT_DEFINITION_TYPE parameter\n", 0);
    exit(1);
    }

    *pBufferSize = sPortDef.nBufferSize;
    nBuffers = sPortDef.nBufferCountMin;
    fprintf(stderr, "Port %d has %d buffers of
```

```
size is %d\n", portNumber, nBuffers,
*pBufferSize);

    ppBuffers = malloc(nBuffers *
sizeof(OMX_BUFFERHEADERTYPE *));
    if (ppBuffers == NULL) {
    fprintf(stderr, "Can't allocate
buffers\n");
    exit(1);
    }

    for (n = 0; n < nBuffers; n++) {
    err = OMX_AllocateBuffer(handle,
ppBuffers+n, portNumber, NULL,
                    *pBufferSize);
    if (err != OMX_ErrorNone) {
        fprintf(stderr, "Error on
AllocateBuffer is %d\n", err);
        exit(1);
    }
    }
    *pppBuffers = ppBuffers;
}

int main(int argc, char** argv) {

    OMX_PORT_PARAM_TYPE param;
```

```
    OMX_PARAM_PORTDEFINITIONTYPE sPortDef;
    OMX_AUDIO_PORTDEFINITIONTYPE sAudioPortDef;
    OMX_AUDIO_PARAM_PORTFORMATTYPE
sAudioPortFormat;
    OMX_AUDIO_PARAM_PCMMODETYPE sPCMMode;
    OMX_BUFFERHEADERTYPE **inBuffers;
    OMX_BUFFERHEADERTYPE **outBuffers;
    OMX_AUDIO_PARAM_MP3TYPE sMP3Mode;



    unsigned char
name[OMX_MAX_STRINGNAME_SIZE];
    OMX_UUIDTYPE uid;
    int startPortNumber;
    int nPorts;
    int n;

    fprintf(stderr, "Thread id is %p\n",
pthread_self());
    if(argc < 2){
    display_help();
    exit(1);
    }

    inFd = open(argv[1], O_RDONLY);
    if(inFd < 0){
    perror("Error opening input file\n");
```

```
    exit(1);
    }
    filesize = getFileSize(inFd);

    outFd = open(argv[2], (O_WRONLY | O_CREAT),
0644);
    if(outFd < 0){
    perror("Error opening output file\n");
    exit(1);
    }

    err = OMX_Init();
    if(err != OMX_ErrorNone) {
    fprintf(stderr, "OMX_Init() failed\n", 0);
    exit(1);
    }
    /** Ask the core for a handle to the audio
render component
     */
    err = OMX_GetHandle(&handle, componentName,
NULL /*app private data */, &callbacks);
    if(err != OMX_ErrorNone) {
    fprintf(stderr, "OMX_GetHandle failed\n",
0);
    exit(1);
    }
    err = OMX_GetComponentVersion(handle, name,
```

```
&compVersion, &specVersion, &uid);
    if(err != OMX_ErrorNone) {
    fprintf(stderr, "OMX_GetComponentVersion
failed\n", 0);
    exit(1);
    }

    /** no other ports to disable */

    /** Get audio port information */
    setHeader(&param,
sizeof(OMX_PORT_PARAM_TYPE));
    err = OMX_GetParameter(handle,
OMX_IndexParamAudioInit, &param);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error in getting
OMX_PORT_PARAM_TYPE parameter\n", 0);
    exit(1);
    }
    startPortNumber =
((OMX_PORT_PARAM_TYPE)param).nStartPortNumber;
    nPorts =
((OMX_PORT_PARAM_TYPE)param).nPorts;
    if (nPorts != 2) {
    fprintf(stderr, "Decode device has wrong
number of ports: %d\n", nPorts);
    exit(1);
```

```
    }

    setEncoding(startPortNumber,
OMX_AUDIO_CodingVORBIS);
    setEncoding(startPortNumber+1,
OMX_AUDIO_CodingPCM);

    printState();;

    numInBuffers =
setNumBuffers(startPortNumber);
    numOutBuffers =
setNumBuffers(startPortNumber+1);

    /* call to put state into idle before
allocating buffers */
    err = OMX_SendCommand(handle,
OMX_CommandStateSet, OMX_StateIdle, NULL);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error on setting state to
idle\n");
    exit(1);
    }

    err = OMX_SendCommand(handle,
OMX_CommandPortEnable, startPortNumber, NULL);
    if (err != OMX_ErrorNone) {
```

```
    fprintf(stderr, "Error on setting port to
enabled\n");
    exit(1);
    }
    err = OMX_SendCommand(handle,
OMX_CommandPortEnable, startPortNumber+1,
NULL);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error on setting port to
enabled\n");
    exit(1);
    }

    createMinBuffers(startPortNumber,
&inBufferSize, &inBuffers);
    createMinBuffers(startPortNumber+1,
&outBufferSize, &outBuffers);



    /* Make sure we've reached Idle state */
    waitFor(OMX_StateIdle);
    fprintf(stderr, "Reached Idle state\n");
    //exit(0);

    /* Now try to switch to Executing state */
    err = OMX_SendCommand(handle,
OMX_CommandStateSet, OMX_StateExecuting, NULL);
```

```
    if(err != OMX_ErrorNone){
    exit(1);
    }

    /* no buffers emptied yet */
    pEmptyBuffer = NULL;

    /* fill and empty the input buffers */

    for (n = 0; n < numInBuffers; n++) {
    //for (n = 0; n < 2; n++) {
    int data_read = read(inFd, inBuffers[n]-
>pBuffer, inBufferSize);
    inBuffers[n]->nFilledLen = data_read;
    printf("Read %d into buffer\n", data_read);
    if (data_read <= 0) {
        printf("In the %s no more input data
available\n", __func__);
        inBuffers[n]->nFilledLen = 0;
        inBuffers[n]->nFlags =
OMX_BUFFERFLAG_EOS;
        bEOS=OMX_TRUE;
    }
    }

    /* empty and fill the output buffers */
    //for (n = 0; n < numOutBuffers; n++) {
```

```
    for (n = 0; n < 2; n++) {
    outBuffers[n]->nFilledLen = 0;
    err = OMX_FillThisBuffer(handle,
outBuffers[n]);
    if (err != OMX_ErrorNone) {
        fprintf(stderr, "Error on filling
buffer\n");
        exit(1);
    }
    }

    for (n = 0; n < numInBuffers; n++) {
    //for (n = 0; n < 2; n++) {
    err = OMX_EmptyThisBuffer(handle,
inBuffers[n]);
    if (err != OMX_ErrorNone) {
        fprintf(stderr, "Error on emptying
buffer\n");
        exit(1);
    }
    }


    pEmptyBuffer = inBuffers[0];
    emptyState = 1;

    waitFor(OMX_StateLoaded);
```

```
    fprintf(stderr, "Buffers emptied\n");
    exit(0);
}
```

Playing an Ogg Vorbis file

We shall use the LIM components and the LIM OpenMAX IL implementation to demonstrate this. There are no appropriate components in the Bellagio or Broadcom cores.

This problem requires the use and interaction between *two* components. In this case, it will be the `OMX.limoi.ogg_dec` and the `OMX.limoi.alsa_sink` components. The `OMX.limoi.ogg_dec` component will take data from an Ogg Vorbis stream, extract the Vorbis data and decode it to PCM. The `OMX.limoi.alsa_sink` component can render this PCM stream to an ALSA output device.

Using the `info` program, we can establish the following characteristics of each component:

- **`OMX.limoi.ogg_dec`** :

```
Component name: OMX.limoi.ogg_dec version
0.0, Spec version 1.1
Audio ports:
Ports start on 0
There are 2 open ports
  Port 0 requires 4 buffers
  Port 0 has min buffer size 4096 bytes
  Port 0 is an input port
  Port 0 is an audio port
  Port mimetype (null)
  Port encoding is Ogg Vorbis
LIM doesn't set audio formats properly
  Port 1 requires 4 buffers
  Port 1 has min buffer size 196608 bytes
  Port 1 is an output port
  Port 1 is an audio port
  Port mimetype (null)
  Port encoding is PCM
LIM doesn't set audio formats properly
Error in getting video OMX_PORT_PARAM_TYPE
parameter
Error in getting image OMX_PORT_PARAM_TYPE
parameter
Error in getting other OMX_PORT_PARAM_TYPE
parameter
```

- **`OMX.limoi.alsa_sink`** :

```
Component name: OMX.limoi.alsa_sink version
0.0, Spec version 1.1
Audio ports:
Ports start on 0
There are 1 open ports
  Port 0 requires 2 buffers
  Port 0 has min buffer size 32768 bytes
  Port 0 is an input port
  Port 0 is an audio port
  Port mimetype (null)
  Port encoding is PCM
LIM doesn't set audio formats properly
Error in getting video OMX_PORT_PARAM_TYPE
parameter
Error in getting image OMX_PORT_PARAM_TYPE
parameter
Error in getting other OMX_PORT_PARAM_TYPE
parameter
```

The decoder has a minimum of four input and four output buffers. The renderer only requires two input buffers, and of course has no output buffers. We will need to take the output from the decoder buffers and put them into the renderer's input

buffers. It isn't totally clear to me whether or not you are required to use all of the buffers, but you can add extra buffers. It would be easier coding if we had the same number of buffers at each stage!

Should this be an aside? If you look at the LIM component "OMX.limoi.ffmpeg.decode.video" it has a minimum of 64 buffers. But if you look at the example LIM code `limoi-core/test/decode.c` you see the following:

```
port_def.nBufferCountActual = 2;
        port_def.nBufferCountMin = 2;
        port_def.nBufferSize = bytes;
        OMX_SetParameter(comp,
OMX_IndexParamPortDefinition, &port_def);
```

Now this is *not* correct! The 1.1 specification in section 3.1.2.12.1 says

> `nBufferCountMin` is a *read-only* [emphasis added] field that specifies the minimum number of buffers that the port requires. The component shall define this non-zero default value.

`nBufferSize` is a *read-only* [emphasis added] field that specifies the minimum size in bytes for buffers that are allocated for this port.

and this example resets these read-only fields.

So what I've done is to make the number of buffers configurable by a #define and set this number in all of the components ports.

The callback logic I use is

- Decode EmptyBuffer Done: refill the buffer from the input file and empty it
- Decode FillBufferDone: set the corresponding alsa_sink input buffer to point to the decoder output buffer data and call EmptyThis Buffer on the alsa_sink
- Alsa Sink EmptyBufferDone: fill the corresponding output decode buffer

Thus the decoder output and renderer input buffers are linked by the IL client, while the decoder input buffer is emptied by the decoder component independently. In an earlier version I tried to link all three buffers, but this always eventually deadlocked with a decoder input buffer not emptying properly.

The code is play_ogg.c

```
/**
   Contains code
   Copyright (C) 2007-2009 STMicroelectronics
   Copyright (C) 2007-2009 Nokia Corporation
and/or its subsidiary(-ies).
   under the LGPL

   and code Copyright (C) 2010 SWOAG Technology
<www.swoag.com> under the LGPL
*/

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/stat.h>
#include <pthread.h>

#include <OMX_Core.h>
#include <OMX_Component.h>
#include <OMX_Types.h>
#include <OMX_Audio.h>
```

```
/* we will use 4 of each port's buffers */
#define NUM_BUFFERS_USED 4

#ifdef LIM
char *decodeComponentName =
"OMX.limoi.ogg_dec";
char *renderComponentName =
"OMX.limoi.alsa_sink";
#endif

OMX_HANDLETYPE decodeHandle;
OMX_HANDLETYPE renderHandle;
OMX_VERSIONTYPE specVersion, compVersion;

int inFd = 0;

static OMX_BOOL bEOS=OMX_FALSE;

OMX_U32 inDecodeBufferSize;
OMX_U32 outDecodeBufferSize;
int numInDecodeBuffers;
int numOutDecodeBuffers;

OMX_U32 inRenderBufferSize;
int numInRenderBuffers;

OMX_BUFFERHEADERTYPE
```

```
*inDecodeBuffers[NUM_BUFFERS_USED];
OMX_BUFFERHEADERTYPE
*outDecodeBuffers[NUM_BUFFERS_USED];
OMX_BUFFERHEADERTYPE
*inRenderBuffers[NUM_BUFFERS_USED];

pthread_mutex_t mutex;
OMX_STATETYPE currentState = OMX_StateLoaded;
pthread_cond_t stateCond;

void waitFor(OMX_STATETYPE state) {
    pthread_mutex_lock(&mutex);
    while (currentState != state)
    pthread_cond_wait(&stateCond, &mutex);
    printf("Wait successfully completed\n");
    pthread_mutex_unlock(&mutex);
}

void wakeUp(OMX_STATETYPE newState) {
    pthread_mutex_lock(&mutex);
    currentState = newState;
    pthread_cond_signal(&stateCond);
    pthread_mutex_unlock(&mutex);
}
pthread_mutex_t empty_mutex;
int emptyState = 0;
OMX_BUFFERHEADERTYPE* pEmptyBuffer;
```

```
pthread_cond_t emptyStateCond;

void waitForEmpty() {
    pthread_mutex_lock(&empty_mutex);
    while (emptyState == 1)
    pthread_cond_wait(&emptyStateCond,
&empty_mutex);
    emptyState = 1;
    pthread_mutex_unlock(&empty_mutex);
}

void wakeUpEmpty(OMX_BUFFERHEADERTYPE* pBuffer)
{
    pthread_mutex_lock(&empty_mutex);
    emptyState = 0;
    pEmptyBuffer = pBuffer;
    pthread_cond_signal(&emptyStateCond);
    pthread_mutex_unlock(&empty_mutex);
}

void mutex_init() {
    int n = pthread_mutex_init(&mutex, NULL);
    if ( n != 0) {
    fprintf(stderr, "Can't init state
mutex\n");
    }
    n = pthread_mutex_init(&empty_mutex, NULL);
```

```
    if ( n != 0) {
    fprintf(stderr, "Can't init empty
mutex\n");
    }
}

static void display_help() {
    fprintf(stderr, "Usage: play_ogg
input_file");
}


static void setHeader(OMX_PTR header, OMX_U32
size) {
    /* header->nVersion */
    OMX_VERSIONTYPE* ver = (OMX_VERSIONTYPE*)
(header + sizeof(OMX_U32));
    /* header->nSize */
    *((OMX_U32*)header) = size;

    /* for 1.2
       ver->s.nVersionMajor =
OMX_VERSION_MAJOR;
       ver->s.nVersionMinor =
OMX_VERSION_MINOR;
       ver->s.nRevision = OMX_VERSION_REVISION;
       ver->s.nStep = OMX_VERSION_STEP;
```

```
    */
    ver->s.nVersionMajor =
specVersion.s.nVersionMajor;
    ver->s.nVersionMinor =
specVersion.s.nVersionMinor;
    ver->s.nRevision = specVersion.s.nRevision;
    ver->s.nStep = specVersion.s.nStep;
}

static void reconfig_component_port
(OMX_HANDLETYPE comp, int port) {
    OMX_PARAM_PORTDEFINITIONTYPE port_def;

    port_def.nSize =
sizeof(OMX_PARAM_PORTDEFINITIONTYPE);
    port_def.nPortIndex = port;
    if (OMX_ErrorNone != OMX_GetParameter
        (comp, OMX_IndexParamPortDefinition,
&port_def))
    {
        fprintf(stderr, "unable to get port %d
definition.\n", port);
        exit(-1);
    } else
    {
        port_def.nBufferCountActual =
NUM_BUFFERS_USED;
```

```
        port_def.nBufferCountMin =
NUM_BUFFERS_USED;

        OMX_SetParameter(comp,
OMX_IndexParamPortDefinition, &port_def);
    }
}

OMX_ERRORTYPE cEventHandler(
                OMX_HANDLETYPE hComponent,
                OMX_PTR pAppData,
                OMX_EVENTTYPE eEvent,
                OMX_U32 Data1,
                OMX_U32 Data2,
                OMX_PTR pEventData) {
    OMX_ERRORTYPE err;

    printf("Hi there, I am in the %s
callback\n", __func__);
    if(eEvent == OMX_EventCmdComplete) {
    if (Data1 == OMX_CommandStateSet) {
        printf("Component State changed in ",
0);
        switch ((int)Data2) {
        case OMX_StateInvalid:
        printf("OMX_StateInvalid\n", 0);
        break;
```

```
        case OMX_StateLoaded:
        printf("OMX_StateLoaded\n", 0);
        break;
        case OMX_StateIdle:
        printf("OMX_StateIdle\n",0);
        break;
        case OMX_StateExecuting:
        printf("OMX_StateExecuting\n",0);
        break;
        case OMX_StatePause:
        printf("OMX_StatePause\n",0);
        break;
        case OMX_StateWaitForResources:

printf("OMX_StateWaitForResources\n",0);
        break;
        }
        wakeUp((int) Data2);
    } else  if (Data1 == OMX_CommandPortEnable)
{
        printf("OMX State Port enabled\n");
    } else if (Data1 == OMX_CommandPortDisable)
{
        printf("OMX State Port disabled\n");
    }
    } else if(eEvent == OMX_EventBufferFlag) {
    if((int)Data2 == OMX_BUFFERFLAG_EOS) {
```

```
    }
    } else if(eEvent == OMX_EventError) {
    printf("Event is Error\n");
    } else  if(eEvent == OMX_EventMark) {
    printf("Event is Buffer Mark\n");
    } else  if(eEvent ==
OMX_EventPortSettingsChanged) {
    /* See 1.1 spec, section 8.9.3.1 Playback
Use Case */
    OMX_PARAM_PORTDEFINITIONTYPE sPortDef;

    setHeader(&sPortDef,
sizeof(OMX_PARAM_PORTDEFINITIONTYPE));
    sPortDef.nPortIndex = Data1;

    err = OMX_GetConfig(hComponent,
OMX_IndexParamPortDefinition, &sPortDef);
    if(err != OMX_ErrorNone){
        fprintf(stderr, "Error in getting
OMX_PORT_DEFINITION_TYPE parameter\n", 0);
        exit(1);
    }
    printf("Event is Port Settings Changed on
port %d in component %p\n",
            Data1, hComponent);
    printf("Port has %d buffers of size is
```

```
%d\n",  sPortDef.nBufferCountMin,
sPortDef.nBufferSize);
    } else {
    printf("Event is %i\n", (int)eEvent);
    printf("Param1 is %i\n", (int)Data1);
    printf("Param2 is %i\n", (int)Data2);
    }

    return OMX_ErrorNone;
}

OMX_ERRORTYPE cDecodeEmptyBufferDone(
                    OMX_HANDLETYPE hComponent,
                    OMX_PTR pAppData,
                    OMX_BUFFERHEADERTYPE*
pBuffer) {
    int n;
    OMX_ERRORTYPE err;

    printf("Hi there, I am in the %s callback,
buffer %p.\n", __func__, pBuffer);
    if (bEOS) {
    printf("Buffers emptied, exiting\n");
    wakeUp(OMX_StateLoaded);
    exit(0);
    }
    printf("  left in buffer %d\n", pBuffer-
```

```
>nFilledLen);

    /* put data into the buffer, and then empty
it */
    int data_read = read(inFd, pBuffer-
>pBuffer, inDecodeBufferSize);
    if (data_read <= 0) {
    bEOS = 1;
    }
    printf("  filled buffer %p with %d\n",
pBuffer, data_read);
    pBuffer->nFilledLen = data_read;

    err = OMX_EmptyThisBuffer(decodeHandle,
pBuffer);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error on emptying decode
buffer\n");
    exit(1);
    }

    return OMX_ErrorNone;
}

OMX_ERRORTYPE cDecodeFillBufferDone(
                  OMX_HANDLETYPE hComponent,
                  OMX_PTR pAppData,
```

```
                    OMX_BUFFERHEADERTYPE*
pBuffer) {
    OMX_BUFFERHEADERTYPE* pRenderBuffer;
    int n;
    OMX_ERRORTYPE err;

    printf("Hi there, I am in the %s callback,
buffer %p with %d bytes.\n",
       __func__, pBuffer, pBuffer->nFilledLen);
    if (bEOS) {
    printf("Buffers filled, exiting\n");
    }

    /* find matching render buffer */
    for (n = 0; n < NUM_BUFFERS_USED; n++) {
    if (pBuffer == outDecodeBuffers[n]) {
        pRenderBuffer = inRenderBuffers[n];
        break;
    }
    }

    pRenderBuffer->nFilledLen = pBuffer-
>nFilledLen;
    /* We don't attempt to refill the decode
output buffer until we
       have emptied the render input buffer. So
we can just use the
```

```
        decode output buffer for the render
input buffer.
        Avoids us copying the data across
buffers.
    */
    pRenderBuffer->pBuffer = pBuffer->pBuffer;

    err = OMX_EmptyThisBuffer(renderHandle,
pRenderBuffer);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error on emptying
buffer\n");
    exit(1);
    }

    return OMX_ErrorNone;
}


OMX_CALLBACKTYPE decodeCallbacks  = {
.EventHandler = cEventHandler,
                .EmptyBufferDone =
cDecodeEmptyBufferDone,
                .FillBufferDone =
cDecodeFillBufferDone
};
```

```
OMX_ERRORTYPE cRenderEmptyBufferDone(
                    OMX_HANDLETYPE hComponent,
                    OMX_PTR pAppData,
                    OMX_BUFFERHEADERTYPE*
pBuffer) {
    OMX_BUFFERHEADERTYPE *inDecodeBuffer;
    OMX_BUFFERHEADERTYPE *outDecodeBuffer;
    int n;
    OMX_ERRORTYPE err;

    printf("Hi there, I am in the %s callback,
buffer %p.\n", __func__, pBuffer);

    /* find matching buffer indices */
    for (n = 0; n < NUM_BUFFERS_USED; n++) {
    if (pBuffer == inRenderBuffers[n]) {
        outDecodeBuffer = outDecodeBuffers[n];
        break;
    }
    }

    /* and fill the corresponding output buffer
*/
    outDecodeBuffer->nFilledLen = 0;
    err = OMX_FillThisBuffer(decodeHandle,
outDecodeBuffer);
```

```
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error on filling
buffer\n");
    exit(1);
    }

    return OMX_ErrorNone;
}



OMX_CALLBACKTYPE renderCallbacks  = {
.EventHandler = cEventHandler,
                .EmptyBufferDone =
cRenderEmptyBufferDone,
                .FillBufferDone = NULL
};

void printState(OMX_HANDLETYPE handle) {
    OMX_STATETYPE state;
    OMX_ERRORTYPE err;

    err = OMX_GetState(handle, &state);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error on getting
state\n");
    exit(1);
    }
```

```
    switch (state) {
    case OMX_StateLoaded:
printf("StateLoaded\n"); break;
    case OMX_StateIdle: printf("StateIdle\n");
break;
    case OMX_StateExecuting:
printf("StateExecuting\n"); break;
    case OMX_StatePause:
printf("StatePause\n"); break;
    case OMX_StateWaitForResources:
printf("StateWiat\n"); break;
    default:  printf("State unknown\n"); break;
    }
}

void setEncoding(OMX_HANDLETYPE handle, int
portNumber, OMX_AUDIO_CODINGTYPE encoding) {
    OMX_AUDIO_PARAM_PORTFORMATTYPE
sAudioPortFormat;
    OMX_ERRORTYPE err;


    setHeader(&sAudioPortFormat,
sizeof(OMX_AUDIO_PARAM_PORTFORMATTYPE));
    sAudioPortFormat.nIndex = 0;
    sAudioPortFormat.nPortIndex = portNumber;
```

```
    err = OMX_GetParameter(handle,
OMX_IndexParamAudioPortFormat,
&sAudioPortFormat);
    if (err == OMX_ErrorNoMore) {
    printf("Can't get format\n");
    return;
    }
    sAudioPortFormat.eEncoding = encoding;
    err = OMX_SetParameter(handle,
OMX_IndexParamAudioPortFormat,
&sAudioPortFormat);
    if (err == OMX_ErrorNoMore) {
    printf("Can't set format\n");
    return;
    }
    printf("Set format on port %d\n",
portNumber);
}

/*
   *pBuffer is set to non-zero if a particular
buffer size is required by the client
 */
void createBuffers(OMX_HANDLETYPE handle, int
portNumber,
              OMX_U32 *pBufferSize,
OMX_BUFFERHEADERTYPE **ppBuffers) {
```

```
    OMX_PARAM_PORTDEFINITIONTYPE sPortDef;
    int n;
    int nBuffers;
    OMX_ERRORTYPE err;

    setHeader(&sPortDef,
sizeof(OMX_PARAM_PORTDEFINITIONTYPE));
    sPortDef.nPortIndex = portNumber;
    err = OMX_GetParameter(handle,
OMX_IndexParamPortDefinition, &sPortDef);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error in getting
OMX_PORT_DEFINITION_TYPE parameter\n", 0);
    exit(1);
    }

    /* if no size pre-allocated, use the
minimum */
    if (*pBufferSize == 0) {
    *pBufferSize = sPortDef.nBufferSize;
    } else {
    sPortDef.nBufferSize = *pBufferSize;
    }

    nBuffers = NUM_BUFFERS_USED;
    printf("Port %d has %d buffers of size is
%d\n", portNumber, nBuffers, *pBufferSize);
```

```
    for (n = 0; n < nBuffers; n++) {
    err = OMX_AllocateBuffer(handle,
ppBuffers+n, portNumber, NULL,
                    *pBufferSize);
    if (err != OMX_ErrorNone) {
        fprintf(stderr, "Error on
AllocateBuffer is %d\n", err);
        exit(1);
    }
    }
}

int main(int argc, char** argv) {

    OMX_PORT_PARAM_TYPE param;
    OMX_PARAM_PORTDEFINITIONTYPE sPortDef;
    OMX_AUDIO_PORTDEFINITIONTYPE sAudioPortDef;
    OMX_AUDIO_PARAM_PORTFORMATTYPE
sAudioPortFormat;
    OMX_AUDIO_PARAM_PCMMODETYPE sPCMMode;
    OMX_ERRORTYPE err;

    OMX_AUDIO_PARAM_MP3TYPE sMP3Mode;


    unsigned char
```

```
name[OMX_MAX_STRINGNAME_SIZE];
    OMX_UUIDTYPE uid;
    int startDecodePortNumber;
    int nDecodePorts;
    int startRenderPortNumber;
    int nRenderPorts;
    int n;

    printf("Thread id is %p\n",
pthread_self());
    if(argc < 1){
    display_help();
    exit(1);
    }

    inFd = open(argv[1], O_RDONLY);
    if(inFd < 0){
    fprintf(stderr, "Error opening input file
\"%s\"\n", argv[1]);
    exit(1);
    }

    err = OMX_Init();
    if(err != OMX_ErrorNone) {
    fprintf(stderr, "OMX_Init() failed\n", 0);
    exit(1);
    }
```

```
   /** Ask the core for a handle to the decode
component
     */
    err = OMX_GetHandle(&decodeHandle,
decodeComponentName,
            NULL /*app private data */,
&decodeCallbacks);
    if(err != OMX_ErrorNone) {
    fprintf(stderr, "OMX_GetHandle failed\n",
0);
    exit(1);
    }
    err = OMX_GetComponentVersion(decodeHandle,
name, &compVersion, &specVersion, &uid);
    if(err != OMX_ErrorNone) {
    fprintf(stderr, "OMX_GetComponentVersion
failed\n", 0);
    exit(1);
    }

    /** Ask the core for a handle to the render
component
     */
    err = OMX_GetHandle(&renderHandle,
renderComponentName,
            NULL /*app private data */,
```

```
&renderCallbacks);
    if(err != OMX_ErrorNone) {
    fprintf(stderr, "OMX_GetHandle failed\n",
0);
    exit(1);
    }

    /** no other ports to disable */

    /** Get audio port information */
    /* Decode component */
    setHeader(&param,
sizeof(OMX_PORT_PARAM_TYPE));
    err = OMX_GetParameter(decodeHandle,
OMX_IndexParamAudioInit, &param);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error in getting
OMX_PORT_PARAM_TYPE parameter\n", 0);
    exit(1);
    }
    startDecodePortNumber =
((OMX_PORT_PARAM_TYPE)param).nStartPortNumber;
    nDecodePorts =
((OMX_PORT_PARAM_TYPE)param).nPorts;
    if (nDecodePorts != 2) {
    fprintf(stderr, "Decode device has wrong
number of ports: %d\n", nDecodePorts);
```

```
    exit(1);
    }

    setEncoding(decodeHandle,
startDecodePortNumber, OMX_AUDIO_CodingVORBIS);
    setEncoding(decodeHandle,
startDecodePortNumber+1, OMX_AUDIO_CodingPCM);

    printState(decodeHandle);;

    reconfig_component_port(decodeHandle,
startDecodePortNumber);
    reconfig_component_port(decodeHandle,
startDecodePortNumber+1);

    /* call to put decoder state into idle
before allocating buffers */
    err = OMX_SendCommand(decodeHandle,
OMX_CommandStateSet, OMX_StateIdle, NULL);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error on setting state to
idle\n");
    exit(1);
    }

    /* ensure decoder ports are enabled */
    err = OMX_SendCommand(decodeHandle,
```

```
OMX_CommandPortEnable, startDecodePortNumber,
NULL);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error on setting port to
enabled\n");
    exit(1);
    }
    err = OMX_SendCommand(decodeHandle,
OMX_CommandPortEnable, startDecodePortNumber+1,
NULL);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error on setting port to
enabled\n");
    exit(1);
    }

    /* use default buffer sizes */
    inDecodeBufferSize = outDecodeBufferSize =
0;
    createBuffers(decodeHandle,
startDecodePortNumber, &inDecodeBufferSize,
inDecodeBuffers);
    createBuffers(decodeHandle,
startDecodePortNumber+1, &outDecodeBufferSize,
outDecodeBuffers);
```

```
    /* Make sure we've reached Idle state */
    waitFor(OMX_StateIdle);
    printf("Reached Idle state\n");

    /* Now try to switch to Executing state */
    err = OMX_SendCommand(decodeHandle,
OMX_CommandStateSet, OMX_StateExecuting, NULL);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error changing to
Executing state\n");
    exit(1);
    }
    /* end decode setting */


    /* Render component */
    setHeader(&param,
sizeof(OMX_PORT_PARAM_TYPE));
    err = OMX_GetParameter(renderHandle,
OMX_IndexParamAudioInit, &param);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error in getting
OMX_PORT_PARAM_TYPE parameter\n", 0);
    exit(1);
    }
    startRenderPortNumber =
((OMX_PORT_PARAM_TYPE)param).nStartPortNumber;
```

```
    nRenderPorts =
((OMX_PORT_PARAM_TYPE)param).nPorts;
    if (nRenderPorts != 1) {
    fprintf(stderr, "Render device has wrong
number of ports: %d\n", nRenderPorts);
    exit(1);
    }

    setEncoding(renderHandle,
startRenderPortNumber, OMX_AUDIO_CodingPCM);

    printState(renderHandle);

     /* call to put state into idle before
allocating buffers */
    err = OMX_SendCommand(renderHandle,
OMX_CommandStateSet, OMX_StateIdle, NULL);
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error on setting state to
idle\n");
    exit(1);
    }

    /* ensure port is enabled */
    err = OMX_SendCommand(renderHandle,
OMX_CommandPortEnable, startRenderPortNumber,
NULL);
```

```
    if (err != OMX_ErrorNone) {
    fprintf(stderr, "Error on setting port to
enabled\n");
    exit(1);
    }

    reconfig_component_port(renderHandle,
startRenderPortNumber);

    /* set render buffer size to decoder out
buffer size */
    inRenderBufferSize = outDecodeBufferSize;
    createBuffers(renderHandle,
startRenderPortNumber, &inRenderBufferSize,
inRenderBuffers);


    /* Make sure we've reached Idle state */
    waitFor(OMX_StateIdle);
    printf("Reached Idle state\n");

    /* Now try to switch renderer to Executing
state */
    err = OMX_SendCommand(renderHandle,
OMX_CommandStateSet, OMX_StateExecuting, NULL);
    if(err != OMX_ErrorNone){
    fprintf(stderr, "Error changing render to
```

```
executing state\n");
    exit(1);
    }
    /* end render setting */


    /* debugging: print buffer pointers */
    for (n = 0; n < NUM_BUFFERS_USED; n++)
    printf("In decode buffer %d is %p\n", n,
inDecodeBuffers[n]);
    for (n = 0; n < NUM_BUFFERS_USED; n++)
    printf("Out decode buffer %d is %p\n", n,
outDecodeBuffers[n]);
    for (n = 0; n < NUM_BUFFERS_USED; n++)
    printf("In render buffer %d is %p\n", n,
inRenderBuffers[n]);


    /* no buffers emptied yet */
    pEmptyBuffer = NULL;

    /* load  the decoder input buffers */
    for (n = 0; n < NUM_BUFFERS_USED; n++) {
    int data_read = read(inFd,
inDecodeBuffers[n]->pBuffer,
inDecodeBufferSize);
    inDecodeBuffers[n]->nFilledLen = data_read;
```

```
    printf("Read %d into buffer %p\n",
data_read, inDecodeBuffers[n]);
    if (data_read <= 0) {
        printf("In the %s no more input data
available\n", __func__);
        inDecodeBuffers[n]->nFilledLen = 0;
        inDecodeBuffers[n]->nFlags =
OMX_BUFFERFLAG_EOS;
        bEOS=OMX_TRUE;
    }
    }

    /* fill the decoder output buffers */
    for (n = 0; n < NUM_BUFFERS_USED; n++) {
    outDecodeBuffers[n]->nFilledLen = 0;
    err = OMX_FillThisBuffer(decodeHandle,
outDecodeBuffers[n]);
    if (err != OMX_ErrorNone) {
        fprintf(stderr, "Error on filling
buffer\n");
        exit(1);
    }
    }

    /* empty the decoder input bufers */
    for (n = 0; n < NUM_BUFFERS_USED; n++) {
    err = OMX_EmptyThisBuffer(decodeHandle,
```

```
inDecodeBuffers[n]);
    if (err != OMX_ErrorNone) {
        fprintf(stderr, "Error on emptying
buffer\n");
        exit(1);
    }
    }

    pEmptyBuffer = inDecodeBuffers[0];
    emptyState = 1;

    waitFor(OMX_StateLoaded);
    printf("Buffers emptied\n");
    exit(0);
}
```

Conclusion

The Khronos group has produced specifications for audio and video in low capability systems. These are currently used by Android and by the Raspberry Pi. This chapter has given an introductory overview of these specifications and some example programs and will be expanded later.

```
/*
 * OpenMAX AL - Audio/Video Playback Example
 */
#include <stdio.h>
#include <stdlib.h>
#include "OpenMAXAL.h"
#define MAX_NUMBER_INTERFACES
5
#define MAX_NUMBER_OUTPUT_DEVICES 3
#define POSITION_UPDATE_PERIOD
1000 /* 1 sec */
/* Checks for error. If any errors exit the
application! */
void CheckErr (XAresult res)
{
    if (res != XA_RESULT_SUCCESS)
    {
        /* Debug printing to be placed here */
        exit(1);
    }
}
void PlayEventCallback (
            XAPlayItf caller,
            void *
            pContext,
            XAuint32 playevent)
```

```
{
    /* Callback code goes here */
}
/*
 * Test audio/video playback from a 3GPP file.
 *
 * NOTE: For the purposes of this example, the
implementation is assumed
 * to support the requisite audio and video
codecs. Therefore, video and
 * audio decoder capabilities are NOT checked
in this example.
 */
void TestAudioVideoPlayback (XAObjectItf
engine)
{
    XAObjectItf
    player;
    XAObjectItf
    OutputMix;
    XAPlayItf
    playItf;
    XAEngineItf
    EngineItf;
    XAAudioIODeviceCapabilitiesItf
AudioIODeviceCapabilitiesItf;
    XAAudioOutputDescriptor
```

```
    AudioOutputDescriptor;
    XAresult
    res;
    XADataSink
    XADataSink
    XADataLocator_OutputMix
    XADataLocator_NativeDisplay
    XAVolumeItf
    audioSink;
    videoSink;
    locator_outputmix;
    locator_displayregion;
    volumeItf;
    XADataSource
    avSource;
    XADataLocator_URI uri;
    XADataFormat_MIME mime;
    int i;
    char c;
    XAboolean
    required[MAX_NUMBER_INTERFACES];
    XAInterfaceID
iidArray[MAX_NUMBER_INTERFACES];
    XAuint32
    XAint32
    XAboolean
    XAboolean
```

```
    XAuint32
    XANativeHandle
    OutputDeviceIDs[MAX_NUMBER_OUTPUT_DEVICES];
    numOutputs
    = 0;
    hfs_available
    = XA_BOOLEAN_FALSE;
    hfs_default
    = XA_BOOLEAN_FALSE;
    hfs_deviceID
    = 0;
    nativeWindowHandle = NULL;
    XANativeHandle nativeDisplayHandle = NULL;
    /* Get the XA Engine Interface, which is
implicit */
    res = (*engine)->GetInterface(engine,
                   XA_IID_ENGINE, (void*)
&EngineItf); CheckErr(res);
    /* Get the Audio IO DEVICE CAPABILITIES
interface, which is also
     * implicit */
    res = (*engine)->GetInterface(engine,

XA_IID_AUDIOIODEVICECAPABILITIES,
                   (void*)
&AudioIODeviceCapabilitiesItf); CheckErr(res);
    numOutputs = MAX_NUMBER_OUTPUT_DEVICES;
```

```
    res = (*AudioIODeviceCapabilitiesItf)->

GetAvailableAudioOutputs(AudioIODeviceCapabilitie

                 &numOutputs, OutputDeviceIDs);
CheckErr(res);
    /* Search for integrated handsfree
loudspeaker */
    for (i = 0; i < numOutputs; i++)
    {
        res = (*AudioIODeviceCapabilitiesItf)->

QueryAudioOutputCapabilities(AudioIODeviceCapabil

                         OutputDeviceIDs[i],
&AudioOutputDescriptor);
        CheckErr(res);
        if
((AudioOutputDescriptor.deviceConnection ==
         XA_DEVCONNECTION_INTEGRATED) &&
        (AudioOutputDescriptor.deviceScope ==
         XA_DEVSCOPE_ENVIRONMENT) &&
        (AudioOutputDescriptor.deviceLocation
==
         XA_DEVLOCATION_HANDSET))
        {
            hfs_deviceID = OutputDeviceIDs[i];
```

```
            hfs_available = XA_BOOLEAN_TRUE;
            break;
        }
    }
    /* If preferred output audio device is not
available, no point in
     * continuing */
    if (!hfs_available)
    {
        /* Appropriate error message here */
        exit(1);
    }
    numOutputs = MAX_NUMBER_OUTPUT_DEVICES;
    res = (*AudioIODeviceCapabilitiesItf)->

GetDefaultAudioDevices(AudioIODeviceCapabilitiesI



XA_DEFAULTDEVICEID_AUDIOOUTPUT, &numOutputs,
                  OutputDeviceIDs);
    CheckErr(res);
    /* Check whether Default Output devices
include the handsfree
     * loudspeaker */
    for (i = 0; i < numOutputs; i++)
    {
        if (OutputDeviceIDs[i] == hfs_deviceID)
```

```
        {
            hfs_default = XA_BOOLEAN_TRUE;
            break;
        }
    }
    /* Expect handsfree loudspeaker to be set
as one of the default
     * output devices */
    if (!hfs_default)
    {
        /* Debug printing to be placed here */
        exit(1);
    }
    /* Initialize arrays required[] and
iidArray[] */
    for (i = 0; i < MAX_NUMBER_INTERFACES; i++)
    {
        required[i] = XA_BOOLEAN_FALSE;
        iidArray[i] = XA_IID_NULL;
    }
    /* Set arrays required[] and iidArray[] for
VOLUME interface */
    required[0] = XA_BOOLEAN_TRUE;
    iidArray[0] = XA_IID_VOLUME;
    /* Create Output Mix object to be used by
player */
    res = (*EngineItf)-
```

```
>CreateOutputMix(EngineItf,
                    &OutputMix, 1, iidArray,
required); CheckErr(res);
    /* Realizing the Output Mix object in
synchronous mode
       res = (*OutputMix)->Realize(OutputMix,
       XA_BOOLEAN_FALSE); CheckErr(res);
    */
    /* Get the volume interface on the output
mix */
    res = (*OutputMix)->GetInterface(OutputMix,
                     XA_IID_VOLUME,
(void*)&volumeItf); CheckErr(res);
    /* Setup the audio/video data source
structure */
    uri.locatorType
    = XA_DATALOCATOR_URI;
    uri.URI
    = (XAchar *) "file:///avmedia.3gp";
    mime.formatType
    = XA_DATAFORMAT_MIME;
    mime.mimeType
    = (XAchar *) "video/3gpp";
    mime.containerType = XA_CONTAINERTYPE_3GPP;
/* provided as a hint to
                         * the player */
    avSource.pLocator = (void*) &uri;
```

```
    avSource.pFormat = (void*) &mime;
    /* Setup the audio data sink structure */
    locator_outputmix.locatorType =
XA_DATALOCATOR_OUTPUTMIX;
    locator_outputmix.outputMix
    = OutputMix;
    audioSink.pLocator
    = (void*) &locator_outputmix;
    audioSink.pFormat
    = NULL;
    /* Set nativeWindowHandle and
nativeDisplayHandle to
     * platform-specific values here */
    /* nativeWindowHandle = <a platform-
specific value>; */
    /* nativeDisplayHandle = <a platform-
specific value>; */
    /* Setup the video data sink structure */
    locator_displayregion.locatorType =
XA_DATALOCATOR_NATIVEDISPLAY;
    locator_displayregion.hWindow
    = nativeWindowHandle;
    locator_displayregion.hDisplay
    = nativeDisplayHandle;
    videoSink.pLocator
    = (void*) &locator_displayregion;
    videoSink.pFormat
```

```
    = NULL;
    /* Create the media player. pBankSrc is
NULL as we have a non-MIDI
     * data source */
    res = (*EngineItf)-
>CreateMediaPlayer(EngineItf,
                        &player, &avSource, NULL,
&audioSink, &videoSink, NULL, NULL,
                        1, iidArray, required);
CheckErr(res);
    /* Realizing the player in synchronous mode
*/
    res = (*player)->Realize(player,
XA_BOOLEAN_FALSE); CheckErr(res);
    /* Get play interface */
    res = (*player)->GetInterface(player,
                   XA_IID_PLAY, (void*)
&playItf); CheckErr(res);
    /* Setup to receive position event
callbacks */
    res = (*playItf)->RegisterCallback(playItf,
                         PlayEventCallback,
NULL); CheckErr(res);
    /* Set notifications to occur after every 1
second - might be useful
     * in updating a progress bar */
    res = (*playItf)-
```

```
>SetPositionUpdatePeriod(playItf,

POSITION_UPDATE_PERIOD); CheckErr(res);
    res = (*playItf)-
>SetCallbackEventsMask(playItf,

XA_PLAYEVENT_HEADATNEWPOS); CheckErr(res);
    /* Before we start, set volume to -3dB
(-300mB) */
    res = (*volumeItf)-
>SetVolumeLevel(volumeItf, -300);
CheckErr(res);
    /* Play the media */
    res = (*playItf)->SetPlayState(playItf,
                    XA_PLAYSTATE_PLAYING);
CheckErr(res);
    while ((c = getchar()) != 'q')
    {
        XAuint32 playState;
        switch(c)
        {
        case '1':
            /* Play the media - if it is not
already playing */
            res = (*playItf)-
>GetPlayState(playItf,
                            &playState);
```

```
CheckErr(res);
            if (playState !=
XA_PLAYSTATE_PLAYING)
            {
                res = (*playItf)-
>SetPlayState(playItf,

XA_PLAYSTATE_PLAYING); CheckErr(res);
                455
                }
            break;
        case '2':
            /* Pause the media - if it is
playing */
            res = (*playItf)-
>GetPlayState(playItf,
                            &playState);
CheckErr(res);
            if (playState ==
XA_PLAYSTATE_PLAYING)
            {
                res = (*playItf)-
>SetPlayState(playItf,

XA_PLAYSTATE_PAUSED); CheckErr(res);
            }
            break;
```

```
        default:
            break;
        }
    }
    /* Stop the media playback */
    res = (*playItf)->SetPlayState(playItf,
                    XA_PLAYSTATE_STOPPED);
CheckErr(res);
    /* Destroy the player object */
    (*player)->Destroy(player);
    /* Destroy the output mix object */
    (*OutputMix)->Destroy(OutputMix);
}
int xa_main (void)
{
    XAresult
    res;
    XAObjectItf engine;
    /* Create OpenMAX AL engine in thread-safe
mode */
    XAEngineOption EngineOption[] = {
    (XAuint32) XA_ENGINEOPTION_THREADSAFE,
    (XAuint32) XA_BOOLEAN_TRUE};
    res = xaCreateEngine(&engine,
             1, EngineOption, 0, NULL, NULL);
CheckErr(res);
    /* Realizing the AL Engine in synchronous
```

```
mode */
    res = (*engine)->Realize(engine,
XA_BOOLEAN_FALSE); CheckErr(res);
    TestAudioVideoPlayback(engine);
    /* Shutdown OpenMAX AL engine */
    (*engine)->Destroy(engine);
    exit(0);
}
```

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

PART 2: DIGITAL SIGNAL PROCESSING

LADSPA (Linux Audio Plug-Ins) is a set of plugins that may be used by applictions to add effects such as delays and filters.

Resources

- Linux Audio Plug-Ins: A Look Into LADSPA by Dave Phillips
- Linux Audio Developer's Simple Plugin API (LADSPA)

Files

All files

Introduction

LADPSA is a set of plugins that may be used by applications. It is designed with simplicity in mind, so is only capable of a limited number of effects. Nevertheless, these can be quite wide-ranging and are sufficient for a large variety of applications.

User level tools

LADSPA plugins live in a directory defaulting to `/usr/lib/ladspa`. This can be controlled by the environment variable `LADSPA_PATH`. This directory will contain a set of `.so` files as LADSPA plugins.

Each plugin contains information about itself, and the set of plugins can be inspected by running the command line tool `listplugins`. By installing just LADPSA, the default plugins are

```
/usr/lib/ladspa/amp.so:
    Mono Amplifier (1048/amp_mono)
    Stereo Amplifier (1049/amp_stereo)
/usr/lib/ladspa/delay.so:
    Simple Delay Line (1043/delay_5s)
/usr/lib/ladspa/filter.so:
    Simple Low Pass Filter (1041/lpf)
    Simple High Pass Filter (1042/hpf)
/usr/lib/ladspa/sine.so:
    Sine Oscillator (Freq:audio, Amp:audio)
(1044/sine_faaa)
    Sine Oscillator (Freq:audio, Amp:control)
(1045/sine_faac)
    Sine Oscillator (Freq:control, Amp:audio)
(1046/sine_fcaa)
    Sine Oscillator (Freq:control, Amp:control)
(1047/sine_fcac)
/usr/lib/ladspa/noise.so:
    White Noise Source (1050/noise_white)
```

More detailed information about each plugin can be found from the tool `analyseplugin`. For example for the `amp` plugin,

```
$analyseplugin amp

Plugin Name: "Mono Amplifier"
Plugin Label: "amp_mono"
Plugin Unique ID: 1048
Maker: "Richard Furse (LADSPA example plugins)"
Copyright: "None"
Must Run Real-Time: No
Has activate() Function: No
Has deactivate() Function: No
Has run_adding() Function: No
Environment: Normal or Hard Real-Time
Ports:    "Gain" input, control, 0 to ...,
default 1, logarithmic
    "Input" input, audio
    "Output" output, audio

Plugin Name: "Stereo Amplifier"
Plugin Label: "amp_stereo"
Plugin Unique ID: 1049
Maker: "Richard Furse (LADSPA example plugins)"
Copyright: "None"
Must Run Real-Time: No
Has activate() Function: No
Has deactivate() Function: No
Has run_adding() Function: No
```

```
Environment: Normal or Hard Real-Time
Ports:    "Gain" input, control, 0 to ...,
default 1, logarithmic
    "Input (Left)" input, audio
    "Output (Left)" output, audio
    "Input (Right)" input, audio
    "Output (Right)" output, audio
```

A simple test of each plugin can be performed using `applyplugin`. When run with no arguments it gives a usage message:

```
$applyplugin
Usage:    applyplugin [flags] <input Wave file>
<output Wave file>
    <LADSPA plugin file name> <plugin label>
<Control1> <Control2>...
    [<LADSPA plugin file name> <plugin label>
<Control1> <Control2>...]...
Flags:    -s<seconds>  Add seconds of silence
after end of input file.
```

This takes an input and an output WAV file as first and second parameters. The next ones are the names of the `.so` file and the plugin label chosen. This is followed by values of the controls.

For the `amp` plugin, the file name is `amp.so`, the stereo plugin is `amp_stereo` and there is only one control for gain as a value between zero and one. To halve the volume of a file containing stereo WAV data,

```
applyplugin 54154.wav tmp.wav amp.so amp_stereo
0.5
```

The type LADSPA_Descriptor

Coomunication between an application and a LADSPA plugin takes place through a data structure of type `LADSPA_Descriptor`. This has fields that contain all of the information that is shown by `listplugins` and `analyseplugins`. In addition, it contains fields that control memory layout, whether or not it supports hard realtime, etc.

- **`unsigned long UniqueID`** : Each plugin must have a unique ID within the LADSPA system.
- **`const char * Label`** : This is the label used to refer to the plugin within the LADSPA system
- **`const char * Name`** : This is the "user friendly" name of the plugin. For example, the `amp` file (shown later) contains two plugins, the mono amplifier has ID 1048, label "amp_mono" and name "Mono Amplifier", while the stereo amplifier has ID 1049, label "amp_stereo" and name "Stereo Amplifier"
- **`const char * Maker, * Copyright`** : Obvious
- **`unsigned long PortCount`** : This indicates the number of ports (input AND output) present on the plugin

- **`const LADSPA_PortDescriptor * PortDescriptors`** : This member indicates an array of port descriptors. Valid indices vary from 0 to PortCount-1
- **`const char * const * PortNames`** : This member indicates an array of null-terminated strings describing ports. For example, the mono amplifer has two input ports and one output port labelled "Gain", "Input" and "Output". The Input port has PortDescriptor `(LADSPA_PORT_INPUT | LADSPA_PORT_AUDIO)` while the Output port has PortDescriptor `(LADSPA_PORT_OUTPUT | LADSPA_PORT_AUDIO)`
- **`LADSPA_PortRangeHint * PortRangeHints`** : This is an array of type `LADSPA_PortRangeHint` , one element for each port. This allows the plugin to pass information such as whether it has a value that is bounded above or below, and if so what is that bound, as to whether it should be treated as a Boolean value, and so on. These hints could be used by, say, a GUI to give a visual control display for the plugin.

Additionally, it contains fields that are function pointers, which are called by the LADSPA runtime to initialise the plugin, handle data and clean up. These fields are

- **`instantiate`** : This takes the sample rate as parameter. It is responsible for general instantiation of the plugin, setting

local parameters, allocating memory, etc. It returns a pointer to a plugin-specific data structure containing all of the information relating to that plugin. This pointer will be passed as the first parameter to the other functions so that they can retrieve information for this plugin.

- **`connect_port`** : This takes three parameters, the second and third being the port number and the address on which data will be readable/writable respectively. The plugin is expected to read/write data from the LADSPA runtime using this address only for each port. it will be called before `run` or `run_adding` .
- **`activate/deactivate`** : These may be called to re-initialise the plugin state. They may be `NULL` .
- **`run`** : This function is where all of the plugin'a real work is done. Its second parameter is the number of samples that are ready to read/write.
- **`cleanup`** : Obvious

Other function fields are normally set to `NULL` .

Loading a plugin

An application can load a plugin by calling `loadLADSPAPluginLibrary` with one parameter which is the name of the plugin file. Note that there is no LADSPA library - LADPSA suuplies a header file `ladspa.h` and the distribution may include a file `load.c` which implements `loadLADSPAPluginLibrary` (it searches the directories in the `LADSPA_PATH`).

When a plugin is loaded by `dlopen` , the function `_init` is called with no parameters. This may be used to setup the plugin and build, for example, the `LADSPA_Descriptor` .

A DLL must have an entry point that you can hook into. For LADSPA, each plugin must define a function `LADSPA_Descriptor * ladspa_descriptor(unsigned long Index)` . The values for indices 0, 1, ... are the `LADSPA_Descriptor` 's for each of the plugins included in the file.

The amp program

The code for the `amp` program is

```c
/* amp.c

   Free software by Richard W.E. Furse. Do with
as you will. No
   warranty.

   This LADSPA plugin provides simple mono and
stereo amplifiers.

   This file has poor memory protection.
Failures during malloc() will
   not recover nicely. */

/*****************************************************


#include <stdlib.h>
#include <string.h>

/*****************************************************


#include "ladspa.h"

/*****************************************************
```

```
/* The port numbers for the plugin: */

#define AMP_CONTROL 0
#define AMP_INPUT1  1
#define AMP_OUTPUT1 2
#define AMP_INPUT2  3
#define AMP_OUTPUT2 4

/*****************************************************


/* The structure used to hold port connection
information and state
   (actually gain controls require no further
state). */

typedef struct {

  /* Ports:
     ------ */

  LADSPA_Data * m_pfControlValue;
  LADSPA_Data * m_pfInputBuffer1;
  LADSPA_Data * m_pfOutputBuffer1;
  LADSPA_Data * m_pfInputBuffer2;  /* (Not used
for mono) */
```

```
  LADSPA_Data * m_pfOutputBuffer2; /* (Not used
for mono) */

} Amplifier;

/*************************************************



/* Construct a new plugin instance. */
LADSPA_Handle
instantiateAmplifier(const LADSPA_Descriptor *
Descriptor,
             unsigned long
SampleRate) {
  return malloc(sizeof(Amplifier));
}

/*************************************************



/* Connect a port to a data location. */
void
connectPortToAmplifier(LADSPA_Handle Instance,
               unsigned long Port,
               LADSPA_Data * DataLocation) {

  Amplifier * psAmplifier;
```

```
  psAmplifier = (Amplifier *)Instance;
  switch (Port) {
  case AMP_CONTROL:
    psAmplifier->m_pfControlValue =
DataLocation;
    break;
  case AMP_INPUT1:
    psAmplifier->m_pfInputBuffer1 =
DataLocation;
    break;
  case AMP_OUTPUT1:
    psAmplifier->m_pfOutputBuffer1 =
DataLocation;
    break;
  case AMP_INPUT2:
    /* (This should only happen for stereo.) */
    psAmplifier->m_pfInputBuffer2 =
DataLocation;
    break;
  case AMP_OUTPUT2:
    /* (This should only happen for stereo.) */
    psAmplifier->m_pfOutputBuffer2 =
DataLocation;
    break;
  }
}
```

```
/*****************************************************


void
runMonoAmplifier(LADSPA_Handle Instance,
         unsigned long SampleCount) {

  LADSPA_Data * pfInput;
  LADSPA_Data * pfOutput;
  LADSPA_Data fGain;
  Amplifier * psAmplifier;
  unsigned long lSampleIndex;

  psAmplifier = (Amplifier *)Instance;

  pfInput = psAmplifier->m_pfInputBuffer1;
  pfOutput = psAmplifier->m_pfOutputBuffer1;
  fGain = *(psAmplifier->m_pfControlValue);

  for (lSampleIndex = 0; lSampleIndex <
SampleCount; lSampleIndex++)
    *(pfOutput++) = *(pfInput++) * fGain;
}

/*****************************************************
```

```
void
runStereoAmplifier(LADSPA_Handle Instance,
           unsigned long SampleCount) {

  LADSPA_Data * pfInput;
  LADSPA_Data * pfOutput;
  LADSPA_Data fGain;
  Amplifier * psAmplifier;
  unsigned long lSampleIndex;

  psAmplifier = (Amplifier *)Instance;

  fGain = *(psAmplifier->m_pfControlValue);

  pfInput = psAmplifier->m_pfInputBuffer1;
  pfOutput = psAmplifier->m_pfOutputBuffer1;
  for (lSampleIndex = 0; lSampleIndex <
SampleCount; lSampleIndex++)
    *(pfOutput++) = *(pfInput++) * fGain;

  pfInput = psAmplifier->m_pfInputBuffer2;
  pfOutput = psAmplifier->m_pfOutputBuffer2;
  for (lSampleIndex = 0; lSampleIndex <
SampleCount; lSampleIndex++)
    *(pfOutput++) = *(pfInput++) * fGain;
}
```

```
/*****************************************************


/* Throw away a simple delay line. */
void
cleanupAmplifier(LADSPA_Handle Instance) {
  free(Instance);
}

/*****************************************************


LADSPA_Descriptor * g_psMonoDescriptor = NULL;
LADSPA_Descriptor * g_psStereoDescriptor =
NULL;

/*****************************************************


/* _init() is called automatically when the
plugin library is first
   loaded. */
void
_init() {

  char ** pcPortNames;
```

```
  LADSPA_PortDescriptor * piPortDescriptors;
  LADSPA_PortRangeHint * psPortRangeHints;

  g_psMonoDescriptor
    = (LADSPA_Descriptor
*)malloc(sizeof(LADSPA_Descriptor));
  g_psStereoDescriptor
    = (LADSPA_Descriptor
*)malloc(sizeof(LADSPA_Descriptor));

  if (g_psMonoDescriptor) {

    g_psMonoDescriptor->UniqueID
      = 1048;
    g_psMonoDescriptor->Label
      = strdup("amp_mono");
    g_psMonoDescriptor->Properties
      = LADSPA_PROPERTY_HARD_RT_CAPABLE;
    g_psMonoDescriptor->Name
      = strdup("Mono Amplifier");
    g_psMonoDescriptor->Maker
      = strdup("Richard Furse (LADSPA example
plugins)");
    g_psMonoDescriptor->Copyright
      = strdup("None");
    g_psMonoDescriptor->PortCount
      = 3;
```

```
    piPortDescriptors
      = (LADSPA_PortDescriptor *)calloc(3,
sizeof(LADSPA_PortDescriptor));
    g_psMonoDescriptor->PortDescriptors
      = (const LADSPA_PortDescriptor
*)piPortDescriptors;
    piPortDescriptors[AMP_CONTROL]
      = LADSPA_PORT_INPUT |
LADSPA_PORT_CONTROL;
    piPortDescriptors[AMP_INPUT1]
      = LADSPA_PORT_INPUT | LADSPA_PORT_AUDIO;
    piPortDescriptors[AMP_OUTPUT1]
      = LADSPA_PORT_OUTPUT | LADSPA_PORT_AUDIO;
    pcPortNames
      = (char **)calloc(3, sizeof(char *));
    g_psMonoDescriptor->PortNames
      = (const char **)pcPortNames;
    pcPortNames[AMP_CONTROL]
      = strdup("Gain");
    pcPortNames[AMP_INPUT1]
      = strdup("Input");
    pcPortNames[AMP_OUTPUT1]
      = strdup("Output");
    psPortRangeHints = ((LADSPA_PortRangeHint
*)
            calloc(3,
sizeof(LADSPA_PortRangeHint)));
```

```
    g_psMonoDescriptor->PortRangeHints
      = (const LADSPA_PortRangeHint
*)psPortRangeHints;

psPortRangeHints[AMP_CONTROL].HintDescriptor
      = (LADSPA_HINT_BOUNDED_BELOW
     | LADSPA_HINT_LOGARITHMIC
     | LADSPA_HINT_DEFAULT_1);
    psPortRangeHints[AMP_CONTROL].LowerBound
      = 0;
    psPortRangeHints[AMP_INPUT1].HintDescriptor
      = 0;

psPortRangeHints[AMP_OUTPUT1].HintDescriptor
      = 0;
    g_psMonoDescriptor->instantiate
      = instantiateAmplifier;
    g_psMonoDescriptor->connect_port
      = connectPortToAmplifier;
    g_psMonoDescriptor->activate
      = NULL;
    g_psMonoDescriptor->run
      = runMonoAmplifier;
    g_psMonoDescriptor->run_adding
      = NULL;
    g_psMonoDescriptor->set_run_adding_gain
      = NULL;
```

```
    g_psMonoDescriptor->deactivate
      = NULL;
    g_psMonoDescriptor->cleanup
      = cleanupAmplifier;
  }

  if (g_psStereoDescriptor) {

    g_psStereoDescriptor->UniqueID
      = 1049;
    g_psStereoDescriptor->Label
      = strdup("amp_stereo");
    g_psStereoDescriptor->Properties
      = LADSPA_PROPERTY_HARD_RT_CAPABLE;
    g_psStereoDescriptor->Name
      = strdup("Stereo Amplifier");
    g_psStereoDescriptor->Maker
      = strdup("Richard Furse (LADSPA example
plugins)");
    g_psStereoDescriptor->Copyright
      = strdup("None");
    g_psStereoDescriptor->PortCount
      = 5;
    piPortDescriptors
      = (LADSPA_PortDescriptor *)calloc(5,
sizeof(LADSPA_PortDescriptor));
    g_psStereoDescriptor->PortDescriptors
```

```
      = (const LADSPA_PortDescriptor
*)piPortDescriptors;
    piPortDescriptors[AMP_CONTROL]
      = LADSPA_PORT_INPUT |
LADSPA_PORT_CONTROL;
    piPortDescriptors[AMP_INPUT1]
      = LADSPA_PORT_INPUT | LADSPA_PORT_AUDIO;
    piPortDescriptors[AMP_OUTPUT1]
      = LADSPA_PORT_OUTPUT | LADSPA_PORT_AUDIO;
    piPortDescriptors[AMP_INPUT2]
      = LADSPA_PORT_INPUT | LADSPA_PORT_AUDIO;
    piPortDescriptors[AMP_OUTPUT2]
      = LADSPA_PORT_OUTPUT | LADSPA_PORT_AUDIO;
    pcPortNames
      = (char **)calloc(5, sizeof(char *));
    g_psStereoDescriptor->PortNames
      = (const char **)pcPortNames;
    pcPortNames[AMP_CONTROL]
      = strdup("Gain");
    pcPortNames[AMP_INPUT1]
      = strdup("Input (Left)");
    pcPortNames[AMP_OUTPUT1]
      = strdup("Output (Left)");
    pcPortNames[AMP_INPUT2]
      = strdup("Input (Right)");
    pcPortNames[AMP_OUTPUT2]
      = strdup("Output (Right)");
```

```
    psPortRangeHints = ((LADSPA_PortRangeHint
*)
            calloc(5,
sizeof(LADSPA_PortRangeHint)));
    g_psStereoDescriptor->PortRangeHints
      = (const LADSPA_PortRangeHint
*)psPortRangeHints;

psPortRangeHints[AMP_CONTROL].HintDescriptor
      = (LADSPA_HINT_BOUNDED_BELOW
     | LADSPA_HINT_LOGARITHMIC
     | LADSPA_HINT_DEFAULT_1);
    psPortRangeHints[AMP_CONTROL].LowerBound
      = 0;
    psPortRangeHints[AMP_INPUT1].HintDescriptor
      = 0;

psPortRangeHints[AMP_OUTPUT1].HintDescriptor
      = 0;
    psPortRangeHints[AMP_INPUT2].HintDescriptor
      = 0;

psPortRangeHints[AMP_OUTPUT2].HintDescriptor
      = 0;
    g_psStereoDescriptor->instantiate
      = instantiateAmplifier;
    g_psStereoDescriptor->connect_port
```

```
      = connectPortToAmplifier;
    g_psStereoDescriptor->activate
      = NULL;
    g_psStereoDescriptor->run
      = runStereoAmplifier;
    g_psStereoDescriptor->run_adding
      = NULL;
    g_psStereoDescriptor->set_run_adding_gain
      = NULL;
    g_psStereoDescriptor->deactivate
      = NULL;
    g_psStereoDescriptor->cleanup
      = cleanupAmplifier;
  }
}

/*************************************************


void
deleteDescriptor(LADSPA_Descriptor *
psDescriptor) {
  unsigned long lIndex;
  if (psDescriptor) {
    free((char *)psDescriptor->Label);
    free((char *)psDescriptor->Name);
    free((char *)psDescriptor->Maker);
```

```
    free((char *)psDescriptor->Copyright);
    free((LADSPA_PortDescriptor *)psDescriptor-
>PortDescriptors);
    for (lIndex = 0; lIndex < psDescriptor-
>PortCount; lIndex++)
      free((char *)(psDescriptor-
>PortNames[lIndex]));
    free((char **)psDescriptor->PortNames);
    free((LADSPA_PortRangeHint *)psDescriptor-
>PortRangeHints);
    free(psDescriptor);
  }
}

/*****************************************************



/* _fini() is called automatically when the
library is unloaded. */
void
_fini() {
  deleteDescriptor(g_psMonoDescriptor);
  deleteDescriptor(g_psStereoDescriptor);
}

/*****************************************************
```

```
/* Return a descriptor of the requested plugin
type. There are two
   plugin types available in this library (mono
and stereo). */
const LADSPA_Descriptor *
ladspa_descriptor(unsigned long Index) {
  /* Return the requested descriptor or null if
the index is out of
     range. */
  switch (Index) {
  case 0:
    return g_psMonoDescriptor;
  case 1:
    return g_psStereoDescriptor;
  default:
    return NULL;
  }
}

/*****************************************************



/* EOF */
```

The analysePlugin client

The `analysePlugin` client loads a plugin file and then does a dump of all the information given about each plugin in the file. It is really just a long-winded plough through all the possibilities, printing out information. Its code is:

```
/* analyseplugin.c

   Free software by Richard W.E. Furse. Do with
as you will. No
   warranty. */

/*****************************************************


#include <dlfcn.h>
#include <math.h>
#include <stdio.h>
#include <string.h>

/*****************************************************


#include "ladspa.h"

#include "utils.h"

/*****************************************************


/* Returns 0 if all goes well, otherwise
returns 1. Label may be null
```

```
   indicating `all plugins.' */
static int
analysePlugin(const char * pcPluginFilename,
          const char * pcPluginLabel,
          const int bVerbose) {

  LADSPA_Descriptor_Function
pfDescriptorFunction;
  const LADSPA_Descriptor * psDescriptor;
  unsigned long lPluginIndex;
  unsigned long lPortIndex;
  unsigned long lSpaceIndex;
  unsigned long lSpacePadding1;
  unsigned long lSpacePadding2;
  unsigned long lLength;
  void * pvPluginHandle;
  LADSPA_PortRangeHintDescriptor
iHintDescriptor;
  LADSPA_Data fBound;
  LADSPA_Data fDefault;

  pvPluginHandle =
loadLADSPAPluginLibrary(pcPluginFilename);

  dlerror();
  pfDescriptorFunction
    =
```

```
(LADSPA_Descriptor_Function)dlsym(pvPluginHandle,
 "ladspa_descriptor");
  if (!pfDescriptorFunction) {
    const char * pcError = dlerror();
    if (pcError)
      fprintf(stderr,
          "Unable to find ladspa_descriptor()
function in plugin file "
          "\"%s\": %s.\n"
          "Are you sure this is a LADSPA plugin
file?\n",
          pcPluginFilename,
          pcError);
    return 1;
  }

  lSpacePadding1 = 0;
  lSpacePadding2 = 0;
  if (!bVerbose) {
    for (lPluginIndex = 0;; lPluginIndex++) {
      psDescriptor =
pfDescriptorFunction(lPluginIndex);
      if (!psDescriptor)
    break;
      if (pcPluginLabel != NULL)
    if (strcmp(pcPluginLabel, psDescriptor-
>Label) != 0)
```

```
      continue;

      lLength = strlen(psDescriptor->Label);
      if (lSpacePadding1 < lLength)
    lSpacePadding1 = lLength;

      lLength = (long)(log10(psDescriptor-
>UniqueID)) + 1;
      if (lSpacePadding2 < lLength)
    lSpacePadding2 = lLength;
    }
    lSpacePadding1 += 2;
    lSpacePadding2 += 2;
  }

  for (lPluginIndex = 0;; lPluginIndex++) {

    psDescriptor =
pfDescriptorFunction(lPluginIndex);
    if (!psDescriptor)
      break;
    if (pcPluginLabel != NULL)
      if (strcmp(pcPluginLabel, psDescriptor-
>Label) != 0)
    continue;

    if (!bVerbose) {
```

```
      printf("%s", psDescriptor->Label);
      for (lSpaceIndex = strlen(psDescriptor-
>Label);
       lSpaceIndex < lSpacePadding1;
       lSpaceIndex++)
    putchar(' ');
      printf("%lu", psDescriptor->UniqueID);
      for (lSpaceIndex = (long)
(log10(psDescriptor->UniqueID)) + 1;
       lSpaceIndex < lSpacePadding2;
       lSpaceIndex++)
    putchar(' ');
      puts(psDescriptor->Name);
    }
    else {

      putchar('\n');

      printf("Plugin Name: \"%s\"\n",
psDescriptor->Name);
      printf("Plugin Label: \"%s\"\n",
psDescriptor->Label);
      printf("Plugin Unique ID: %lu\n",
psDescriptor->UniqueID);
      printf("Maker: \"%s\"\n", psDescriptor-
>Maker);
      printf("Copyright: \"%s\"\n",
```

```
psDescriptor->Copyright);

      printf("Must Run Real-Time: ");
      if (LADSPA_IS_REALTIME(psDescriptor-
>Properties))
    printf("Yes\n");
      else
    printf("No\n");

      printf("Has activate() Function: ");
      if (psDescriptor->activate != NULL)
    printf("Yes\n");
      else
    printf("No\n");
      printf("Has deactivate() Function: ");
      if (psDescriptor->deactivate != NULL)
    printf("Yes\n");
      else
    printf("No\n");
      printf("Has run_adding() Function: ");
      if (psDescriptor->run_adding != NULL)
    printf("Yes\n");
      else
    printf("No\n");

      if (psDescriptor->instantiate == NULL)
    printf("ERROR: PLUGIN HAS NO INSTANTIATE
```

```
FUNCTION.\n");
      if (psDescriptor->connect_port == NULL)
    printf("ERROR: PLUGIN HAS NO CONNECT_PORT
FUNCTION.\n");
      if (psDescriptor->run == NULL)
    printf("ERROR: PLUGIN HAS NO RUN
FUNCTION.\n");
      if (psDescriptor->run_adding != NULL
      && psDescriptor->set_run_adding_gain ==
NULL)
    printf("ERROR: PLUGIN HAS RUN_ADDING
FUNCTION BUT "
           "NOT SET_RUN_ADDING_GAIN.\n");
      if (psDescriptor->run_adding == NULL
      && psDescriptor->set_run_adding_gain !=
NULL)
    printf("ERROR: PLUGIN HAS
SET_RUN_ADDING_GAIN FUNCTION BUT "
           "NOT RUN_ADDING.\n");
      if (psDescriptor->cleanup == NULL)
    printf("ERROR: PLUGIN HAS NO CLEANUP
FUNCTION.\n");

      printf("Environment: ");
      if
(LADSPA_IS_HARD_RT_CAPABLE(psDescriptor-
>Properties))
```

```
    printf("Normal or Hard Real-Time\n");
      else
    printf("Normal\n");

      if
(LADSPA_IS_INPLACE_BROKEN(psDescriptor-
>Properties))
    printf("This plugin cannot use in-place
processing. "
           "It will not work with all
hosts.\n");

      printf("Ports:");

      if (psDescriptor->PortCount == 0)
    printf("\tERROR: PLUGIN HAS NO PORTS.\n");

      for (lPortIndex = 0;
       lPortIndex < psDescriptor->PortCount;
       lPortIndex++) {

    printf("\t\"%s\" ", psDescriptor-
>PortNames[lPortIndex]);

    if (LADSPA_IS_PORT_INPUT
        (psDescriptor-
>PortDescriptors[lPortIndex])
```

```
        && LADSPA_IS_PORT_OUTPUT
        (psDescriptor-
>PortDescriptors[lPortIndex]))
      printf("ERROR: INPUT AND OUTPUT");
    else if (LADSPA_IS_PORT_INPUT
        (psDescriptor-
>PortDescriptors[lPortIndex]))
      printf("input");
    else if (LADSPA_IS_PORT_OUTPUT
        (psDescriptor-
>PortDescriptors[lPortIndex]))
      printf("output");
    else
      printf("ERROR: NEITHER INPUT NOR
OUTPUT");

    if (LADSPA_IS_PORT_CONTROL
        (psDescriptor-
>PortDescriptors[lPortIndex])
        && LADSPA_IS_PORT_AUDIO
        (psDescriptor-
>PortDescriptors[lPortIndex]))
      printf(", ERROR: CONTROL AND AUDIO");
    else if (LADSPA_IS_PORT_CONTROL
        (psDescriptor-
>PortDescriptors[lPortIndex]))
      printf(", control");
```

```
    else if (LADSPA_IS_PORT_AUDIO
         (psDescriptor-
>PortDescriptors[lPortIndex]))
      printf(", audio");
    else
      printf(", ERROR: NEITHER CONTROL NOR
AUDIO");

    iHintDescriptor
      = psDescriptor-
>PortRangeHints[lPortIndex].HintDescriptor;

    if
(LADSPA_IS_HINT_BOUNDED_BELOW(iHintDescriptor)
        ||
LADSPA_IS_HINT_BOUNDED_ABOVE(iHintDescriptor))
{
      printf(", ");
      if
(LADSPA_IS_HINT_BOUNDED_BELOW(iHintDescriptor))
{
        fBound = psDescriptor-
>PortRangeHints[lPortIndex].LowerBound;
        if
(LADSPA_IS_HINT_SAMPLE_RATE(iHintDescriptor) &&
fBound != 0)
          printf("%g*srate", fBound);
```

```
        else
          printf("%g", fBound);
      }
      else
        printf("...");
      printf(" to ");
      if
(LADSPA_IS_HINT_BOUNDED_ABOVE(iHintDescriptor))
{
        fBound = psDescriptor-
>PortRangeHints[lPortIndex].UpperBound;
        if
(LADSPA_IS_HINT_SAMPLE_RATE(iHintDescriptor) &&
fBound != 0)
          printf("%g*srate", fBound);
        else
          printf("%g", fBound);
      }
      else
        printf("...");
    }

    if
(LADSPA_IS_HINT_TOGGLED(iHintDescriptor)) {
      if ((iHintDescriptor
           | LADSPA_HINT_DEFAULT_0
           | LADSPA_HINT_DEFAULT_1)
```

```
            != (LADSPA_HINT_TOGGLED
            | LADSPA_HINT_DEFAULT_0
            | LADSPA_HINT_DEFAULT_1))
        printf(", ERROR: TOGGLED INCOMPATIBLE
WITH OTHER HINT");
      else
        printf(", toggled");
    }

    switch (iHintDescriptor &
LADSPA_HINT_DEFAULT_MASK) {
    case LADSPA_HINT_DEFAULT_NONE:
      break;
    case LADSPA_HINT_DEFAULT_MINIMUM:
      fDefault = psDescriptor-
>PortRangeHints[lPortIndex].LowerBound;
      if
(LADSPA_IS_HINT_SAMPLE_RATE(iHintDescriptor) &&
fDefault != 0)
        printf(", default %g*srate", fDefault);
      else
        printf(", default %g", fDefault);
      break;
    case LADSPA_HINT_DEFAULT_LOW:
      if
(LADSPA_IS_HINT_LOGARITHMIC(iHintDescriptor)) {
        fDefault
```

```
          = exp(log(psDescriptor-
>PortRangeHints[lPortIndex].LowerBound)
            * 0.75
            + log(psDescriptor-
>PortRangeHints[lPortIndex].UpperBound)
            * 0.25);
      }
      else {
        fDefault
          = (psDescriptor-
>PortRangeHints[lPortIndex].LowerBound
         * 0.75
         + psDescriptor-
>PortRangeHints[lPortIndex].UpperBound
         * 0.25);
      }
      if
(LADSPA_IS_HINT_SAMPLE_RATE(iHintDescriptor) &&
fDefault != 0)
        printf(", default %g*srate", fDefault);
      else
        printf(", default %g", fDefault);
      break;
    case LADSPA_HINT_DEFAULT_MIDDLE:
      if
(LADSPA_IS_HINT_LOGARITHMIC(iHintDescriptor)) {
        fDefault
```

```
          = sqrt(psDescriptor-
>PortRangeHints[lPortIndex].LowerBound
             * psDescriptor-
>PortRangeHints[lPortIndex].UpperBound);
      }
      else {
        fDefault
          = 0.5 * (psDescriptor-
>PortRangeHints[lPortIndex].LowerBound
               + psDescriptor-
>PortRangeHints[lPortIndex].UpperBound);
      }
      if
(LADSPA_IS_HINT_SAMPLE_RATE(iHintDescriptor) &&
fDefault != 0)
        printf(", default %g*srate", fDefault);
      else
        printf(", default %g", fDefault);
      break;
    case LADSPA_HINT_DEFAULT_HIGH:
      if
(LADSPA_IS_HINT_LOGARITHMIC(iHintDescriptor)) {
        fDefault
          = exp(log(psDescriptor-
>PortRangeHints[lPortIndex].LowerBound)
            * 0.25
            + log(psDescriptor-
```

```
>PortRangeHints[lPortIndex].UpperBound)
            * 0.75);
      }
      else {
        fDefault
          = (psDescriptor-
>PortRangeHints[lPortIndex].LowerBound
         * 0.25
         + psDescriptor-
>PortRangeHints[lPortIndex].UpperBound
         * 0.75);
      }
      if
(LADSPA_IS_HINT_SAMPLE_RATE(iHintDescriptor) &&
fDefault != 0)
        printf(", default %g*srate", fDefault);
      else
        printf(", default %g", fDefault);
      break;
    case LADSPA_HINT_DEFAULT_MAXIMUM:
      fDefault = psDescriptor-
>PortRangeHints[lPortIndex].UpperBound;
      if
(LADSPA_IS_HINT_SAMPLE_RATE(iHintDescriptor) &&
fDefault != 0)
        printf(", default %g*srate", fDefault);
      else
```

```
        printf(", default %g", fDefault);
      break;
    case LADSPA_HINT_DEFAULT_0:
      printf(", default 0");
      break;
    case LADSPA_HINT_DEFAULT_1:
      printf(", default 1");
      break;
    case LADSPA_HINT_DEFAULT_100:
      printf(", default 100");
      break;
    case LADSPA_HINT_DEFAULT_440:
      printf(", default 440");
      break;
    default:
      printf(", UNKNOWN DEFAULT CODE");
      /* (Not necessarily an error - may be a
newer version.) */
      break;
    }

    if
(LADSPA_IS_HINT_LOGARITHMIC(iHintDescriptor))
      printf(", logarithmic");

    if
(LADSPA_IS_HINT_INTEGER(iHintDescriptor))
```

```
      printf(", integer");

    putchar('\n');
      }
    }
  }

  if (bVerbose)
    putchar('\n');

  unloadLADSPAPluginLibrary(pvPluginHandle);

  return(0);
}

/*****************************************************



int
main(const int iArgc, const char ** ppcArgv) {

  const char * pcPluginName = NULL;
  const char * pcPluginLabel = NULL;
  int bVerbose = 1;

  /* Check for a flag, but only at the start.
Cannot get use getopt()
```

```
     as it gets thoroughly confused when faced
with negative numbers
     on the command line. */
  switch (iArgc) {
  case 2:
    if (strcmp(ppcArgv[1], "-h") != 0) {
      pcPluginName = ppcArgv[1];
      pcPluginLabel = NULL;
    }
    break;
  case 3:
    if (strcmp(ppcArgv[1], "-l") == 0) {
      pcPluginName = ppcArgv[2];
      pcPluginLabel = NULL;
      bVerbose = 0;
    }
    else {
      pcPluginName = ppcArgv[1];
      pcPluginLabel = ppcArgv[2];
    }
    break;
  case 4:
    if (strcmp(ppcArgv[1], "-l") == 0) {
      pcPluginName = ppcArgv[2];
      pcPluginLabel = ppcArgv[3];
      bVerbose = 0;
    }
```

```
    break;
  }

  if (!pcPluginName) {
    fprintf(stderr,
        "Usage:\tanalyseplugin [flags] <LADSPA
plugin file name> "
        "[<plugin label>].\n"
        "Flags:"
        "-l  Produce a summary list rather than
a verbose report.\n"
            "Note that the LADSPA_PATH
environment variable is used "
            "to help find plugins.\n");
    return(1);
  }

  return analysePlugin(pcPluginName,
pcPluginLabel, bVerbose);
}

/*****************************************************



/* EOF */
```

A mono amplifier client

The `applyplugin` program shows how clients can use LADSPA plugins in a general way. Unfortunately, the necessary generality makes it harder to see what is being done. In this section we just look at a simple client that uses the `amp_mono` plugin to halve the volume of a file.

From running `analyseplugin` on the `amp.so` file we can find that it contains a mono and stereo plugin. In the following program the `main` function loads the plugin file, gets a handle to the `ladspa_descriptor` structure and then looks through the descriptors until it finds the `amp_mono` plugin.

We know there are three ports: control, input and output, so we look through the list of ports to assign indices and connect the relevant arrays to the plugin descriptor. The control port only needs the address of a float value which is the amount of amplification that will occur.

The `run_plugin` function then just loops, reading samples from the imput file, applying the plugin and writing to the output file. I've used the libsndfile library to simplify reading and

writing files in whatever format they are in. I've also used the `load.c` file from the LADSPA package to simplify loading the plugin library.

The program is `mono_amp.c`:

```c
#include <stdlib.h>
#include <stdio.h>
#include <ladspa.h>
#include <dlfcn.h>
#include <sndfile.h>

#include "utils.h"

const LADSPA_Descriptor * psDescriptor;
LADSPA_Descriptor_Function
pfDescriptorFunction;
LADSPA_Handle handle;

// choose the mono plugin from the amp file
char *pcPluginFilename = "amp.so";
char *pcPluginLabel = "amp_mono";

long lInputPortIndex = -1;
long lOutputPortIndex = -1;

SNDFILE* pInFile;
SNDFILE* pOutFile;

// for the amplifier, the sample rate doesn't
really matter
#define SAMPLE_RATE 44100
```

```
// the buffer size isn't really important
either
#define BUF_SIZE 2048
LADSPA_Data pInBuffer[BUF_SIZE];
LADSPA_Data pOutBuffer[BUF_SIZE];

// How much we are amplifying the sound by
LADSPA_Data control = 0.5f;

char *pInFilePath = "/home/local/antialize-
wkhtmltopdf-7cb5810/scripts/static-build/linux-
local/qts/demos/mobile/quickhit/plugins/LevelTemp

char *pOutFilePath = "tmp.wav";

void open_files() {
    // using libsndfile functions for easy
read/write
    SF_INFO sfinfo;

    sfinfo.format = 0;
    pInFile = sf_open(pInFilePath, SFM_READ,
&sfinfo);
    if (pInFile == NULL) {
    perror("can't open input file");
    exit(1);
```

```
    }

    pOutFile = sf_open(pOutFilePath, SFM_WRITE,
&sfinfo);
    if (pOutFile == NULL) {
    perror("can't open output file");
    exit(1);
    }
}

sf_count_t fill_input_buffer() {
    return sf_read_float(pInFile, pInBuffer,
BUF_SIZE);
}

void empty_output_buffer(sf_count_t numread) {
    sf_write_float(pOutFile, pOutBuffer,
numread);
}

void run_plugin() {
    sf_count_t numread;

    open_files();

    // it's NULL for the amp plugin
    if (psDescriptor->activate != NULL)
```

```
    psDescriptor->activate(handle);

    while ((numread = fill_input_buffer()) > 0)
{
    printf("Num read %d\n", numread);
    psDescriptor->run(handle, numread);
    empty_output_buffer(numread);
    }
}

int main(int argc, char *argv[]) {
    int lPluginIndex;

    void *pvPluginHandle =
loadLADSPAPluginLibrary(pcPluginFilename);
    dlerror();

    pfDescriptorFunction
    =
(LADSPA_Descriptor_Function)dlsym(pvPluginHandle,
 "ladspa_descriptor");
    if (!pfDescriptorFunction) {
    const char * pcError = dlerror();
    if (pcError)
        fprintf(stderr,
            "Unable to find ladspa_descriptor()
function in plugin file "
```

```
            "\"%s\": %s.\n"
            "Are you sure this is a LADSPA
plugin file?\n",
            pcPluginFilename,
            pcError);
    return 1;
    }

    for (lPluginIndex = 0;; lPluginIndex++) {
    psDescriptor =
pfDescriptorFunction(lPluginIndex);
    if (!psDescriptor)
        break;
    if (pcPluginLabel != NULL) {
        if (strcmp(pcPluginLabel, psDescriptor-
>Label) != 0)
        continue;
    }
    // got mono_amp

    handle = psDescriptor-
>instantiate(psDescriptor, SAMPLE_RATE);
    if (handle == NULL) {
        fprintf(stderr, "Can't instantiate
plugin %s\n", pcPluginLabel);
        exit(1);
    }
```

```
    // get ports
    int lPortIndex;
    printf("Num ports %lu\n", psDescriptor-
>PortCount);
    for (lPortIndex = 0;
         lPortIndex < psDescriptor->PortCount;
         lPortIndex++) {
        if (LADSPA_IS_PORT_INPUT
        (psDescriptor-
>PortDescriptors[lPortIndex])
        && LADSPA_IS_PORT_AUDIO
        (psDescriptor-
>PortDescriptors[lPortIndex])) {
        printf("input %d\n", lPortIndex);
        lInputPortIndex = lPortIndex;

        psDescriptor->connect_port(handle,
                        lInputPortIndex,
pInBuffer);
        } else if (LADSPA_IS_PORT_OUTPUT
               (psDescriptor-
>PortDescriptors[lPortIndex])
               && LADSPA_IS_PORT_AUDIO
               (psDescriptor-
>PortDescriptors[lPortIndex])) {
        printf("output %d\n", lPortIndex);
```

```
        lOutputPortIndex = lPortIndex;

        psDescriptor->connect_port(handle,
                        lOutputPortIndex,
pOutBuffer);
        }

        if (LADSPA_IS_PORT_CONTROL
        (psDescriptor-
>PortDescriptors[lPortIndex])) {
        printf("control %d\n", lPortIndex);
        psDescriptor->connect_port(handle,
                        lPortIndex, &control);
        }
    }
    // we've got what we wanted, get out of
this loop
    break;
    }

    if ((psDescriptor == NULL) ||
    (lInputPortIndex == -1) ||
    (lOutputPortIndex == -1)) {
    fprintf(stderr, "Can't find plugin
information\n");
    exit(1);
    }
```

```
    run_plugin();

    exit(0);
}
```

it is run just by calling `mono_amp`, no arguments.

A stereo amplifer with GUI

The `amp` file contains a stereo amplifier as well as a mono amplifier. This causes several differences to managing the plugin. there are now two input ports and two output ports, but still only one control port for the amplification factor. We need an array of input ports and an array of output ports. This just adds a little complexity.

The major difference is in handling the streams: `libsndfile` returns *frames* of sound, with the two channels of a stereo signal interleaved. These have to be split out into separate channels for each input port, and then the two output ports have to interleaved back together.

Adding a GUI such as GTK is fairly straightforward. The following code just shows a slider to control the volume. The GUI code and the LADSPA code must obviously run in different (POSIX) threads. There is really only one tricky point: the control value is not supposed to change during execution of the `run` function. This could be protected by locks, but in this case

that is too heavyweight: just keep a copy of the control as modified by the slider, and bring that across before each call to `run`.

The code is written to use GTK v3 and is:

```c
#include <gtk/gtk.h>

#include <stdlib.h>
#include <stdio.h>
#include <ladspa.h>
#include <dlfcn.h>
#include <sndfile.h>

#include "utils.h"

gint count = 0;
char buf[5];

pthread_t ladspa_thread;

const LADSPA_Descriptor * psDescriptor;
LADSPA_Descriptor_Function
pfDescriptorFunction;
LADSPA_Handle handle;

// choose the mono plugin from the amp file
char *pcPluginFilename = "amp.so";
char *pcPluginLabel = "amp_stereo";

long lInputPortIndex = -1;
long lOutputPortIndex = -1;
```

```
int inBufferIndex = 0;
int outBufferIndex = 0;

SNDFILE* pInFile;
SNDFILE* pOutFile;

// for the amplifier, the sample rate doesn't
really matter
#define SAMPLE_RATE 44100

// the buffer size isn't really important
either
#define BUF_SIZE 2048
LADSPA_Data pInStereoBuffer[2*BUF_SIZE];
LADSPA_Data pOutStereoBuffer[2*BUF_SIZE];
LADSPA_Data pInBuffer[2][BUF_SIZE];
LADSPA_Data pOutBuffer[2][BUF_SIZE];

// How much we are amplifying the sound by
// We aren't allowed to change the control
values
// during execution of run(). We could put a
lock
// around run() or simpler, change the value of
// control only outside of run()
LADSPA_Data control;
```

```
LADSPA_Data pre_control = 0.2f;

char *pInFilePath =
"/home/newmarch/Music/karaoke/nights/nightsinwhit
0.wav";
char *pOutFilePath = "tmp.wav";

void open_files() {
    // using libsndfile functions for easy
read/write
    SF_INFO sfinfo;

    sfinfo.format = 0;
    pInFile = sf_open(pInFilePath, SFM_READ,
&sfinfo);
    if (pInFile == NULL) {
    perror("can't open input file");
    exit(1);
    }

    pOutFile = sf_open(pOutFilePath, SFM_WRITE,
&sfinfo);
    if (pOutFile == NULL) {
    perror("can't open output file");
    exit(1);
    }
}
```

```
sf_count_t fill_input_buffer() {
    int numread = sf_read_float(pInFile,
pInStereoBuffer, 2*BUF_SIZE);

    // split frames into samples for each
channel
    int n;
    for (n = 0; n < numread; n += 2) {
    pInBuffer[0][n/2] = pInStereoBuffer[n];
    pInBuffer[1][n/2] = pInStereoBuffer[n+1];
    }
    return numread/2;
}

void empty_output_buffer(sf_count_t numread) {
    // combine output samples back into frames
    int n;
    for (n = 0; n < 2*numread; n += 2) {
    pOutStereoBuffer[n] = pOutBuffer[0][n/2];
    pOutStereoBuffer[n+1] = pOutBuffer[1][n/2];
    }

    sf_write_float(pOutFile, pOutStereoBuffer,
2*numread);
}
```

```
gpointer run_plugin(gpointer args) {
    sf_count_t numread;

    // it's NULL for the amp plugin
    if (psDescriptor->activate != NULL)
    psDescriptor->activate(handle);

    while ((numread = fill_input_buffer()) > 0)
{
    // reset control outside of run()
    control = pre_control;

    psDescriptor->run(handle, numread);
    empty_output_buffer(numread);
    usleep(1000);
    }
    printf("Plugin finished!\n");
}

void setup_ladspa() {
    int lPluginIndex;

    void *pvPluginHandle =
loadLADSPAPluginLibrary(pcPluginFilename);
    dlerror();

    pfDescriptorFunction
```

```
    =
(LADSPA_Descriptor_Function)dlsym(pvPluginHandle,
 "ladspa_descriptor");
    if (!pfDescriptorFunction) {
    const char * pcError = dlerror();
    if (pcError)
        fprintf(stderr,
            "Unable to find ladspa_descriptor()
function in plugin file "
            "\"%s\": %s.\n"
            "Are you sure this is a LADSPA
plugin file?\n",
            pcPluginFilename,
            pcError);
    exit(1);
    }

    for (lPluginIndex = 0;; lPluginIndex++) {
    psDescriptor =
pfDescriptorFunction(lPluginIndex);
    if (!psDescriptor)
        break;
    if (pcPluginLabel != NULL) {
        if (strcmp(pcPluginLabel, psDescriptor-
>Label) != 0)
        continue;
    }
```

```
    // got mono_amp

    handle = psDescriptor-
>instantiate(psDescriptor, SAMPLE_RATE);
    if (handle == NULL) {
        fprintf(stderr, "Can't instantiate
plugin %s\n", pcPluginLabel);
        exit(1);
    }

    // get ports
    int lPortIndex;
    printf("Num ports %lu\n", psDescriptor-
>PortCount);
    for (lPortIndex = 0;
         lPortIndex < psDescriptor->PortCount;
         lPortIndex++) {
        if (LADSPA_IS_PORT_AUDIO
        (psDescriptor-
>PortDescriptors[lPortIndex])) {
        if (LADSPA_IS_PORT_INPUT
            (psDescriptor-
>PortDescriptors[lPortIndex])) {
            printf("input %d\n", lPortIndex);
            lInputPortIndex = lPortIndex;

            psDescriptor->connect_port(handle,
```

```
                                lInputPortIndex,
pInBuffer[inBufferIndex++]);
        } else if (LADSPA_IS_PORT_OUTPUT
                (psDescriptor-
>PortDescriptors[lPortIndex])) {
            printf("output %d\n", lPortIndex);
            lOutputPortIndex = lPortIndex;

            psDescriptor->connect_port(handle,
                                lOutputPortIndex,
pOutBuffer[outBufferIndex++]);
        }
        }

        if (LADSPA_IS_PORT_CONTROL
        (psDescriptor-
>PortDescriptors[lPortIndex])) {
        printf("control %d\n", lPortIndex);
        psDescriptor->connect_port(handle,
                        lPortIndex, &control);
        }
    }
    // we've got what we wanted, get out of
this loop
    break;
    }
```

```
    if ((psDescriptor == NULL) ||
    (lInputPortIndex == -1) ||
    (lOutputPortIndex == -1)) {
    fprintf(stderr, "Can't find plugin
information\n");
    exit(1);
    }

    open_files();

    pthread_create(&ladspa_thread, NULL,
run_plugin, NULL);
}

void slider_change(GtkAdjustment *adj,
gpointer data)
{
    count++;

    pre_control =
gtk_adjustment_get_value(adj);
    //gtk_label_set_text(GTK_LABEL(label),
buf);
}

int main(int argc, char** argv) {
```

```
    //GtkWidget *label;
    GtkWidget *window;
    GtkWidget *frame;
    GtkWidget *slider;
    GtkAdjustment *adjustment;

    setup_ladspa();

    gtk_init(&argc, &argv);

    window =
gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_position(GTK_WINDOW(window),
GTK_WIN_POS_CENTER);

gtk_window_set_default_size(GTK_WINDOW(window),
250, 80);
    gtk_window_set_title(GTK_WINDOW(window),
"Volume");

    frame = gtk_fixed_new();
    gtk_container_add(GTK_CONTAINER(window),
frame);

    adjustment = gtk_adjustment_new(1.0,
                                    0.0,
                                    2.0,
```

```
                                    0.1,
                                    1.0,
                                    0.0);
    slider =
gtk_scale_new(GTK_ORIENTATION_HORIZONTAL,
             adjustment);
    gtk_widget_set_size_request(slider, 240,
5);
    gtk_fixed_put(GTK_FIXED(frame), slider, 5,
20);



    //label = gtk_label_new("0");
    //gtk_fixed_put(GTK_FIXED(frame), label,
190, 58);

    gtk_widget_show_all(window);

    g_signal_connect(window, "destroy",
             G_CALLBACK (gtk_main_quit), NULL);

    g_signal_connect(adjustment, "value-
changed",
             G_CALLBACK(slider_change), NULL);

    gtk_main();
```

```
    return 0;
}
```

It is run just by calling `stereo_amp`, no arguments.

Conclusion

LADSPA is a commonly used framework for audio effects plugins. This chapter has considered the programming model and also some command line tools.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

PART 3: DIVERSIONS

This chapter has nothing to do with sound. However, many sound applications use a GUI environment playing a video (or similar), possibly with text overlays such as subtitles. This diversion is about programming the video side of this, using FFMpeg, Gtk, Cairo and Pango. I assume that you are familiar with the concepts of widgets, events and event handlers and so on that underlie all current GUI frameworks.

Introduction

Videos often accompany audio. Karaoke oftens overlays the video with lyrics. Building an application to include video as well as audio takes us into the realm of graphical user interfaces (GUIs). This is a complex area in its own right, and deserves (and has!) many books, including my own from many years back on the X Window System and Motif.

Motif lost its status as a major GUI for Linux/Unix systems a long time ago. There are many alternatives now, including Gtk (Gimp toolkit), tcl/Tk, Java Swing, KDE, XFCE, ... They each have their own adherents, domains of use, quirks, idiosyncrasies, ... No single GUI will satisfy everyone.

In this chapter I deal with Gtk. The reasons are threefold:

- It has a C library. It also has a Python library which is nice and I might use it one day. Most importantly it is *not* C++ based - C++ is one of my least favourite languages. I once came across a quote (source lost) that "C++ is a laboratory experiment that escaped" and I completely agree with that assessment

- It has good support for i18n (internationalisation). I want to be able to play Chinese Karaoke files, so this is important to me
- It is not Java-based. Don't get me wrong, I really like Java and have been programming in it for years. The MIDI API is pretty good, and of course everything else such as i18n is great. But for MIDI it is a CPU hog and is unusable on low powered devices such as the Raspberry Pi, and generally the audio/video API has not progressed in years

Nevertheless, as I struggled to get my head around Gtk, versions 2.0 versus 3.0, Cairo, Pango, Glib, etc. I thought it might have been easier just to fix the Java MIDI engine! This was not a pleasant experience, as the sequel will show :-(.

FFMpeg

To play Mpeg files, OGV files or similar, a decoder is required. The main contenders seem to be GStreamer and FFMpeg. For no particular reason I chose FFMpeg.

The following program reads from a video file and stores the first five frames to disk. It is taken directly from An ffmpeg and SDL Tutorial by Stephen Dranger. The program is `play_video.c` :

```
#include <libavcodec/avcodec.h>
#include <libavformat/avformat.h>
#include <libswscale/swscale.h>

/* Requires
   libavcodec-dev
   libavformat-dev
   libswscale
*/

void SaveFrame(AVFrame *pFrame, int width, int
height, int iFrame) {
    FILE *pFile;
    char szFilename[32];
    int  y;

    // Open file
    sprintf(szFilename, "frame%d.ppm", iFrame);
    pFile=fopen(szFilename, "wb");
    if(pFile==NULL)
    return;

    // Write header
    fprintf(pFile, "P6\n%d %d\n255\n", width,
height);
```

```
    // Write pixel data
    for(y=0; y<height; y++)
    fwrite(pFrame->data[0]+y*pFrame-
>linesize[0], 1, width*3, pFile);

    // Close file
    fclose(pFile);
}

main(int argc, char **argv) {
    AVFormatContext *pFormatCtx = NULL;
    int i, videoStream;
    AVCodecContext *pCodecCtx = NULL;
    AVCodec *pCodec = NULL;
    AVFrame *pFrame = NULL;
    AVFrame *pFrameRGB = NULL;
    AVPacket packet;
    int frameFinished;
    int numBytes;
    uint8_t *buffer = NULL;

    AVDictionary *optionsDict = NULL;
    struct SwsContext *sws_ctx = NULL;


    if(argc < 2) {
    printf("Please provide a movie file\n");
```

```
    return -1;
    }
    // Register all formats and codecs
    av_register_all();

    // Open video file
    if(avformat_open_input(&pFormatCtx,
argv[1], NULL, NULL)!=0)
    return -1; // Couldn't open file

    // Retrieve stream information
    if(avformat_find_stream_info(pFormatCtx,
NULL)<0)
    return -1; // Couldn't find stream
information

    // Dump information about file onto
standard error
    av_dump_format(pFormatCtx, 0, argv[1], 0);

    // Find the first video stream
    videoStream=-1;
    for(i=0; i<pFormatCtx->nb_streams; i++)
    if(pFormatCtx->streams[i]->codec-
>codec_type==AVMEDIA_TYPE_VIDEO) {
        videoStream=i;
        break;
```

```
    }
    if(videoStream==-1)
    return -1; // Didn't find a video stream

    // Get a pointer to the codec context for
the video stream
    pCodecCtx=pFormatCtx->streams[videoStream]-
>codec;

    // Find the decoder for the video stream
    pCodec=avcodec_find_decoder(pCodecCtx-
>codec_id);
    if(pCodec==NULL) {
    fprintf(stderr, "Unsupported codec!\n");
    return -1; // Codec not found
    }
    // Open codec
    if(avcodec_open2(pCodecCtx, pCodec,
&optionsDict)<0)
    return -1; // Could not open codec

    // Allocate video frame
    pFrame=avcodec_alloc_frame();

    // Allocate an AVFrame structure
    pFrameRGB=avcodec_alloc_frame();
    if(pFrameRGB==NULL)
```

```
    return -1;

    // Determine required buffer size and
allocate buffer
    numBytes=avpicture_get_size(PIX_FMT_RGB24,
pCodecCtx->width,
                pCodecCtx->height);
    buffer=(uint8_t
*)av_malloc(numBytes*sizeof(uint8_t));


    sws_ctx =
    sws_getContext
    (
     pCodecCtx->width,
     pCodecCtx->height,
     pCodecCtx->pix_fmt,
     pCodecCtx->width,
     pCodecCtx->height,
     PIX_FMT_RGB24,
     SWS_BILINEAR,
     NULL,
     NULL,
     NULL
     );
```

```
    // Assign appropriate parts of buffer to
image planes in pFrameRGB
    // Note that pFrameRGB is an AVFrame, but
AVFrame is a superset
    // of AVPicture
    avpicture_fill((AVPicture *)pFrameRGB,
buffer, PIX_FMT_RGB24,
           pCodecCtx->width, pCodecCtx-
>height);

    // Read frames and save first five frames
to disk
    i=0;
    while(av_read_frame(pFormatCtx,
&packet)>=0) {
    // Is this a packet from the video stream?
    if(packet.stream_index==videoStream) {
        // Decode video frame
        avcodec_decode_video2(pCodecCtx,
pFrame, &frameFinished,
                  &packet);

        // Did we get a video frame?
        if(frameFinished) {
        // Convert the image from its native
format to RGB
        sws_scale
```

```
            (
             sws_ctx,
             (uint8_t const * const *)pFrame-
>data,
             pFrame->linesize,
             0,
             pCodecCtx->height,
             pFrameRGB->data,
             pFrameRGB->linesize
             );

        printf("Read frame\n");
        // Save the frame to disk
        if(++i<=5)
            SaveFrame(pFrameRGB, pCodecCtx-
>width, pCodecCtx->height,
                      i);
        else
            break;
        }
    }

    // Free the packet that was allocated by
av_read_frame
    av_free_packet(&packet);
    }
```

```
    // Free the RGB image
    av_free(buffer);
    av_free(pFrameRGB);

    // Free the YUV frame
    av_free(pFrame);

    // Close the codec
    avcodec_close(pCodecCtx);

    // Close the video file
    avformat_close_input(&pFormatCtx);

    return 0;

}
```

Basic Gtk

Gtk is a fairly standard GUI toolkit. Simple programs are described in many tutorials such as First programs in GTK+ Refer to such tutorials for the basics in Gtk programming.

We just include the following example without explanation which uses three child widgets, two buttons and one label. The label will hold an integer number. The buttons will increase or decrease this number.

```c
#include <gtk/gtk.h>

    gint count = 0;
    char buf[5];

    void increase(GtkWidget *widget, gpointer
label)
    {
          count++;

        sprintf(buf, "%d", count);
        gtk_label_set_text(GTK_LABEL(label),
buf);
    }

    void decrease(GtkWidget *widget, gpointer
label)
    {
        count--;

        sprintf(buf, "%d", count);
        gtk_label_set_text(GTK_LABEL(label),
buf);
    }

    int main(int argc, char** argv) {
```

```
        GtkWidget *label;
        GtkWidget *window;
        GtkWidget *frame;
        GtkWidget *plus;
        GtkWidget *minus;

        gtk_init(&argc, &argv);

        window =
gtk_window_new(GTK_WINDOW_TOPLEVEL);

gtk_window_set_position(GTK_WINDOW(window),
GTK_WIN_POS_CENTER);

gtk_window_set_default_size(GTK_WINDOW(window),
250, 180);

gtk_window_set_title(GTK_WINDOW(window), "+-");

        frame = gtk_fixed_new();

gtk_container_add(GTK_CONTAINER(window),
frame);

        plus = gtk_button_new_with_label("+");
        gtk_widget_set_size_request(plus, 80,
```

```
35);
        gtk_fixed_put(GTK_FIXED(frame), plus,
50, 20);

        minus = gtk_button_new_with_label("-");
        gtk_widget_set_size_request(minus, 80,
35);
        gtk_fixed_put(GTK_FIXED(frame), minus,
50, 80);

        label = gtk_label_new("0");
        gtk_fixed_put(GTK_FIXED(frame), label,
190, 58);

        gtk_widget_show_all(window);

        g_signal_connect(window, "destroy",
        G_CALLBACK (gtk_main_quit), NULL);

        g_signal_connect(plus, "clicked",
        G_CALLBACK(increase), label);

        g_signal_connect(minus, "clicked",
        G_CALLBACK(decrease), label);

        gtk_main();
```

```
        return 0;
    }
```

Gtk, like every other GUI toolkit, has a large number of widgets. These are listed in the GTK+ 3 Reference Manual . This includes the widget GtkImage . As would be expected from the name, it can take a set of pixels from somewhere and build them into an image which can be displayed.

The following example is from CS 23 Software Design and Implementation Lecture notes GTK+ Programming and shows an image loaded from a file:

```
#include <gtk/gtk.h>

int main( int argc, char *argv[])
{
    GtkWidget *window, *image;

    gtk_init(&argc, &argv);

    window =
gtk_window_new(GTK_WINDOW_TOPLEVEL);

    gtk_window_set_position(GTK_WINDOW(window),
GTK_WIN_POS_CENTER);

gtk_window_set_default_size(GTK_WINDOW(window),
230, 150);
    gtk_window_set_title(GTK_WINDOW(window),
"Image");

gtk_window_set_resizable(GTK_WINDOW(window),
FALSE);


gtk_container_set_border_width(GTK_CONTAINER(wind
 2);
```

```
    image =
gtk_image_new_from_file("pic/60cm.gif");
    gtk_container_add(GTK_CONTAINER(window),
image);

    g_signal_connect(G_OBJECT(window),
"destroy", G_CALLBACK(gtk_main_quit), NULL);
    gtk_widget_show_all(window);
    gtk_main();

    return 0;
}
```

Versions of Gtk

Gtk currently (at July 2013) has major versions 2 and 3. The macro `GTK_MAJOR_VERSION` can be used to detect version 2 or 3. However, Gtk also depends on a number of other libraries and it can get very confusing working out which documentation pages you should be looking at. Here is a list of the principal libraries and their primary API pages:

- **Gtk 3** :
- Gdk 3
- Cairo 1
- Pango 1
- Gdk Pixbuf 2
- Glib 2
- Freetype 2
- **Gtk 2** :
- Gdk 2
- Cairo 1
- Pango 1
- Glib 2
- Freetype 2

Displaying the video using Gtk

We want to take the images produced by FFMpeg as `AVFrame`'s and display them in a `GtkImage`. We don't want to use code that reads from a file, because reading and writing files at 30 frames per second would be ludicrous. Instead we want some in-memory representation of the frames to load into the `GtkImage`.

Here is where we hit our first snag: the suitable in-memory representation changed in an incompatable way between Gtk 2.0 and Gtk 3.0. I'm only going to talk in the language of the X Window System since I don't know about other underlying systems such as Microsoft Windows.

See Migrating from GTK+ 2.x to GTK+ 3 for a description of some of the changes between these versions.

Pixmaps

The X Window System architecture model is a client-server model which has clients (applications) talking to servers (devices with graphic displays and input devices). At the lowest level

(Xlib) a client will send basic requests such as "draw a line from here to there" to the server. The server will draw the line using information on the server side such as current line thickness, colour, etc.

If you want to keep an array of pixels to represent an image, then this array is usually kept on the *X Window server* in a *pixmap* . Pixmaps can be created and modified by applications by sending messages across the wire from the client to the server. Even a simple modification such as changing the value of a single pixel involves a network roundtrip and this can obviously become very expensive if done often.

Pixbufs

Pixbufs are *client* side equivalents of pixmaps. They can be manipulated by the client without round trips to the X server. This reduces time and network overheads in manipulating them. However, it means that information that would have been kept on the server now has to built and maintained on the client application side.

X, Wayland and Mir

The X Window System is nearly 30 years old. During that time it has evolved to meet changes in hardware and in software requirements, while still maintaining backward compatability.

Significant changes have occurred in hardware during this 30 years: multi-core systems are now prevalent and GPUs have brought changes in processing video. And generally the amount of memory (cache and RAM) means that memory is no longer such an issue.

At the same time, the software side has evolved. It is now common to make use of a "compositing window manager" such as Compiz so that you can have effects such as Wobbly Windows. This is not good for the X Window model: requests from the application go to the X server but then a requested image has to be passed to the compositing window manager which will perform its effects and then send images back to the X server. This is big increase in network traffic, in which the X server is now just playing the role of display rather than compositor.

Application libraries have now evolved so that much of the work that was formerly done by the X server can now be done on the application side by libraries such as cairo, pixman, freetype, fontconfig and pango.

All of these changes have led to proposals for new backend servers which live cooperatively in this evolved world. This has been sparked by the development of Wayland , but is a bit messed up by Ubuntu forking this to develop Mir . We won't buy into those arguments - just google for "mir and wayland"...

In a very simplistic sense, what it means here is that in future pixmaps are out while pixbufs are in.

Gtk 3.0

With Gtk 3.0, pixmaps no longer exist. You only have pixbufs, in the data structure `GdkPixbuf` . To display the FFMpeg decoded video we pick up after the image has been transcoded to the `picture_RGB` , translate it into a `GdkPixbuf` and create the `GtkImage`

```
pixbuf = gdk_pixbuf_new_from_data(picture_RGB-
>data[0], GDK_COLORSPACE_RGB,
                                        0, 8,
width, height,

picture_RGB->linesize[0],

pixmap_destroy_notify,
                                        NULL);
    gtk_image_set_from_pixbuf((GtkImage*)
image, pixbuf);
```

Gtk 2.0

Gtk 2.0 still had pixmaps, in the structure `GdkPixmap`. In theory it should be possible to have code similar to the Gtk 3.0 code using the function `GdkPixmap *gdk_pixmap_create_from_data(GdkDrawable *drawable, const gchar *data, gint width, gint height, gint depth, const GdkColor *fg, const GdkColor *bg)` which is documented in the GDK 2 Reference Manual at Bitmaps and Pixmaps and then call `void`

`gtk_image_set_from_pixmap(GtkImage *image, GdkPixmap *pixmap, GdkBitmap *mask)` documented in the Gtk 2.6 reference manual at GtkImage .

The only problem is that I couldn't get the function `gdk_pixmap_create_from_data` to work. No matter what argument I tried for the drawable, the call always barfed on either its type or its value. For example, a documented value is `NULL` but this always caused an assertion error ("should not be NULL").

So what does work? Well, all I could find was a bit of a mess of both pixmaps *and* pixbufs: create a pixbuf filled with video data, create a pixmap, write the pixbuf data into the pixmap, and then fill the image with the pixmap data:

```
pixbuf = gdk_pixbuf_new_from_data(picture_RGB-
>data[0], GDK_COLORSPACE_RGB,
                                                0,
8, width, height,

picture_RGB->linesize[0],

pixmap_destroy_notify,
                                          NULL);
    pixmap = gdk_pixmap_new(window->window,
width, height, -1);
    gdk_draw_pixbuf((GdkDrawable *) pixmap,
NULL,
                     pixbuf,
                     0, 0, 0, 0, wifth, height,
                     GDK_RGB_DITHER_NORMAL, 0,
0);

    gtk_image_set_from_pixmap((GtkImage*)
image, pixmap, NULL);
```

Threads and Gtk

The video will need to play in its own thread. Gtk will set up a GUI processing loop in its thread. Since this is Linux, we will use Posix `pthreads`. The video thread will need to be started explicitly by

```
pthread_t tid;
    pthread_create(&tid, NULL, play_background,
NULL);
```

where the function `play_background` calls the FFMpeg code to decode the video file. Note that the thread should not be started until the application has been realised, or it will attempt to draw into non-existent windows.

The Gtk thread will be started by the call to

```
gtk_widget_show (window);
```

That's straightforward enough. But now we have to handle the video thread making calls into the GUI thread in order to draw the image. The best document I have found on this is Is GTK+ thread safe? How do I write multi-threaded GTK+ applications?

Basically it states that code which can affect the Gtk thread should be enclosed by a `gdk_threads_enter() ... gdk_threads_leave()` pair.

That's okay for Gtk 2.0. What about Gtk 3.0? Ooops! Those calls are now deprecated. So what are you supposed to do? So far (as at 3 July, 2013) all that seems to exist are developer dialogs such as Re: deprecating gdk threads which states "We have never done a great job of explaining when gdk_threads_enter/leave are required, it seems - as a consequence, a good number of the critical sections I've seen marked throughout my jhbuild checkout are unnecessary. If your application doesn't call gdk_threads_init or gdk_threads_set_lock_functions, there's no need to use enter/leave. Libraries are a different story, of course". But who should call `gdk_threads_init` ? And what's this about libraries? I'll continue to use them until I know better.

The code

Finally, the code to play a video in a Gtk application which works with both Gtk 2.0 and Gtk 3.0! It is `gtk_play_video.c` :

```c
#include <gtk/gtk.h>
#include <gdk/gdkx.h>

#include <libavcodec/avcodec.h>
#include <libavformat/avformat.h>
#include <libswscale/swscale.h>

GtkWidget *image;
GtkWidget *window;

#if GTK_MAJOR_VERSION == 2
GdkPixmap *pixmap;
#endif

/* FFMpeg vbls */
AVFormatContext *pFormatCtx = NULL;
AVCodecContext *pCodecCtx = NULL;
int videoStream;
struct SwsContext *sws_ctx = NULL;
AVCodec *pCodec = NULL;

#define WIDTH 192
#define HEIGHT 144

static void pixmap_destroy_notify(guchar
```

```
*pixels,
                    gpointer data) {
    printf("Destroy pixmap - not sure how\n");
}

static gpointer play_background(gpointer args)
{
    /* based on code from

http://www.cs.dartmouth.edu/~xy/cs23/gtk.html

http://cdry.wordpress.com/2009/09/09/using-
custom-io-callbacks-with-ffmpeg/
    */

    int i;
    AVPacket packet;
    int frameFinished;
    AVFrame *pFrame = NULL;

    /* initialize packet, set data to NULL, let
the demuxer fill it */
    /*
http://ffmpeg.org/doxygen/trunk/doc_2examples_2de
example.html#a80 */
    av_init_packet(&packet);
    packet.data = NULL;
```

```
    packet.size = 0;

    int bytesDecoded;
    GdkPixbuf *pixbuf;
    AVFrame *picture_RGB;
    char *buffer;

    pFrame=avcodec_alloc_frame();

    i=0;
    picture_RGB = avcodec_alloc_frame();
    buffer = malloc
(avpicture_get_size(PIX_FMT_RGB24, WIDTH,
HEIGHT));
    avpicture_fill((AVPicture *)picture_RGB,
buffer, PIX_FMT_RGB24, WIDTH, HEIGHT);

    while(av_read_frame(pFormatCtx,
&packet)>=0) {
    if(packet.stream_index==videoStream) {
        usleep(33670);  // 29.7 frames per
second
        // Decode video frame
        avcodec_decode_video2(pCodecCtx,
pFrame, &frameFinished,
                  &packet);
```

```
        int width = pCodecCtx->width;
        int height = pCodecCtx->height;

        sws_ctx = sws_getContext(width, height,
pCodecCtx->pix_fmt, width, height,
                     PIX_FMT_RGB24,
SWS_BICUBIC, NULL, NULL, NULL);

        if (frameFinished) {
        printf("Frame %d\n", i++);

        sws_scale(sws_ctx,  (uint8_t const *
const *) pFrame->data, pFrame->linesize, 0,
height, picture_RGB->data, picture_RGB-
>linesize);

        printf("old width %d new width %d\n",
pCodecCtx->width, picture_RGB->width);
        pixbuf =
gdk_pixbuf_new_from_data(picture_RGB->data[0],
GDK_COLORSPACE_RGB,
                         0, 8, width, height,
                         picture_RGB-
>linesize[0], pixmap_destroy_notify,
                         NULL);
```

```
        /* start GTK thread lock for drawing */
        gdk_threads_enter();

#if  GTK_MAJOR_VERSION == 2
        /* this leaks memory somehow */
        pixmap = gdk_pixmap_new(window->window,
WIDTH, HEIGHT, -1);

        gdk_draw_pixbuf((GdkDrawable *) pixmap,
NULL,
                pixbuf,
                0, 0, 0, 0, WIDTH, HEIGHT,
                GDK_RGB_DITHER_NORMAL, 0, 0);

        gtk_image_set_from_pixmap((GtkImage*)
image, pixmap, NULL);

        //gtk_widget_queue_draw(image);
#elif GTK_MAJOR_VERSION == 3
        gtk_image_set_from_pixbuf((GtkImage*)
image, pixbuf);

        //gtk_widget_queue_draw(image);

#endif
        //g_object_unref(pixbuf);
```

```
        //sws_freeContext(sws_ctx);
        /* release GTK thread lock */
        gdk_threads_leave();
        }
    }
    av_free_packet(&packet);
    g_thread_yield();
    }

    printf("Video over!\n");
    exit(0);
}

/* Called when the windows are realized
 */
static void realize_cb (GtkWidget *widget,
gpointer data) {
    /* start the video playing in its own
thread */
    GThread *tid;
    tid = g_thread_new("video",
                play_background,
                NULL);
}

static gboolean delete_event( GtkWidget
*widget,
```

```
                                GdkEvent  *event,
                                gpointer   data )
{
    /* If you return FALSE in the "delete-
event" signal handler,
     * GTK will emit the "destroy" signal.
Returning TRUE means
     * you don't want the window to be
destroyed.
     * This is useful for popping up 'are you
sure you want to quit?'
     * type dialogs. */

    g_print ("delete event occurred\n");

    /* Change TRUE to FALSE and the main window
will be destroyed with
     * a "delete-event". */
    return TRUE;
}

/* Another callback */
static void destroy( GtkWidget *widget,
                     gpointer   data )
{
    gtk_main_quit ();
}
```

```
int main(int argc, char** argv)
{
    // Is this necessary?
    XInitThreads();

    int i;


    /* FFMpeg stuff */

    AVFrame *pFrame = NULL;
    AVPacket packet;

    AVDictionary *optionsDict = NULL;

    av_register_all();

    if(avformat_open_input(&pFormatCtx,
"/home/httpd/html/ComputersComputing/simpson.mpg"
 NULL, NULL)!=0)
    return -1; // Couldn't open file

    // Retrieve stream information
    if(avformat_find_stream_info(pFormatCtx,
NULL)<0)
    return -1; // Couldn't find stream
```

```
information

    // Dump information about file onto
standard error
    av_dump_format(pFormatCtx, 0, argv[1], 0);

    // Find the first video stream
    videoStream=-1;
    for(i=0; i<pFormatCtx->nb_streams; i++)
    if(pFormatCtx->streams[i]->codec-
>codec_type==AVMEDIA_TYPE_VIDEO) {
        videoStream=i;
        break;
    }
    if(videoStream==-1)
    return -1; // Didn't find a video stream

    for(i=0; i<pFormatCtx->nb_streams; i++)
    if(pFormatCtx->streams[i]->codec-
>codec_type==AVMEDIA_TYPE_AUDIO) {
        printf("Found an audio stream too\n");
        break;
    }


    // Get a pointer to the codec context for
the video stream
```

```
    pCodecCtx=pFormatCtx->streams[videoStream]-
>codec;

    // Find the decoder for the video stream
    pCodec=avcodec_find_decoder(pCodecCtx-
>codec_id);
    if(pCodec==NULL) {
    fprintf(stderr, "Unsupported codec!\n");
    return -1; // Codec not found
    }

    // Open codec
    if(avcodec_open2(pCodecCtx, pCodec,
&optionsDict)<0)
    return -1; // Could not open codec

    sws_ctx =
    sws_getContext
    (
     pCodecCtx->width,
     pCodecCtx->height,
     pCodecCtx->pix_fmt,
     pCodecCtx->width,
     pCodecCtx->height,
     PIX_FMT_YUV420P,
     SWS_BILINEAR,
     NULL,
```

```
     NULL,
     NULL
     );

    /* GTK stuff now */

     /* This is called in all GTK applications.
Arguments are parsed
     * from the command line and are returned
to the application. */
    gtk_init (&argc, &argv);

    /* create a new window */
    window = gtk_window_new
(GTK_WINDOW_TOPLEVEL);

    /* When the window is given the "delete-
event" signal (this is given
     * by the window manager, usually by the
"close" option, or on the
     * titlebar), we ask it to call the
delete_event () function
     * as defined above. The data passed to the
callback
     * function is NULL and is ignored in the
callback function. */
    g_signal_connect (window, "delete-event",
```

```
                G_CALLBACK (delete_event), NULL);

    /* Here we connect the "destroy" event to a
signal handler.
     * This event occurs when we call
gtk_widget_destroy() on the window,
     * or if we return FALSE in the "delete-
event" callback. */
    g_signal_connect (window, "destroy",
                G_CALLBACK (destroy), NULL);

    g_signal_connect (window, "realize",
G_CALLBACK (realize_cb), NULL);

    /* Sets the border width of the window. */
    gtk_container_set_border_width
(GTK_CONTAINER (window), 10);

    image = gtk_image_new();
    gtk_widget_show (image);

    /* This packs the button into the window (a
gtk container). */
    gtk_container_add (GTK_CONTAINER (window),
image);

    /* and the window */
```

```
    gtk_widget_show (window);

    /* All GTK applications must have a
gtk_main(). Control ends here
     * and waits for an event to occur (like a
key press or
     * mouse event). */
    gtk_main ();

    return 0;
}
```

Overlaying an image on top of an image

It is common in a movie on TV to see a fixed image layered on top of the video. Subtitles can be an example of dynamic images but may be text overlaid instead. This section just considers one image on top of another.

In Gtk 2.0 it is surprisingly easy: draw one pixbuf into a pixmap and then draw the overlay pixbuf into the same pixmap.

```
pixmap = gdk_pixmap_new(window->window, 720,
480, -1);

gdk_draw_pixbuf((GdkDrawable *) pixmap, NULL,
        pixbuf,
        0, 0, 0, 0, 720, 480,
        GDK_RGB_DITHER_NORMAL, 0, 0);

// overlay another pixbuf
gdk_draw_pixbuf((GdkDrawable *) pixmap, NULL,
                overlay_pixbuf,
                0, 0, 0, 0, overlay_width,
overlay_height,
                GDK_RGB_DITHER_NORMAL, 0, 0);

gtk_image_set_from_pixmap((GtkImage*) image,
pixmap, NULL);

gtk_widget_queue_draw(image);
```

Gtk 3.0 does not seem so straightforward as pixmaps have disappeared. Various pages suggest using Cairo surfaces instead and later sections will look at that. But the page on the The GdkPixbuf Structure suggests that - as long as you get the data types aligned - you can just write the pixels of the second image

into the pixbuf data of the first. The page (although old) Gdk-pixbuf is a useful tutorial on Gdk pixbufs. One of the details you have to get right is the *rowstride* of each image: the two-dimensional image is stored as a linear array of bytes and the rowstride tells how many bytes make up a row. Typically there are 3 or 4 bytes per pixel (for RGB or RGB+alpha) and these also need to be matched between the images.

The program `gtk_play_video_overlay.c` is

```
#include <gtk/gtk.h>
#include <gdk/gdkx.h>

#include <libavcodec/avcodec.h>
#include <libavformat/avformat.h>
#include <libswscale/swscale.h>

//#define OVERLAY_IMAGE
"/home/newmarch/jannewmarch.signature.png"
#define OVERLAY_IMAGE "jan-small.png"
#define VIDEO "short.mpg"
//#define VIDEO "video1.mp4"

GtkWidget *image;
GtkWidget *window;

#if GTK_MAJOR_VERSION == 2
GdkPixmap *pixmap;
#endif

/* FFMpeg vbls */
AVFormatContext *pFormatCtx = NULL;
AVCodecContext *pCodecCtx = NULL;
int videoStream;
struct SwsContext *sws_ctx = NULL;
AVCodec *pCodec = NULL;
```

```
static void pixmap_destroy_notify(guchar
*pixels,
                  gpointer data) {
    printf("Destroy pixmap - not sure how\n");
}

#if  GTK_MAJOR_VERSION == 3
static void overlay(GdkPixbuf *pixbuf,
GdkPixbuf *overlay_pixbuf,
             int height_offset, int
width_offset) {
    int overlay_width, overlay_height,
overlay_rowstride, overlay_n_channels;
    guchar *overlay_pixels, *overlay_p;
    guchar red, green, blue, alpha;
    int m, n;
    int rowstride, n_channels, width, height;
    guchar *pixels, *p;

    if (overlay_pixbuf == NULL) {
        return;
    }

    /* get stuff out of overlay pixbuf */
```

```
    overlay_n_channels =
gdk_pixbuf_get_n_channels (overlay_pixbuf);
    n_channels =
gdk_pixbuf_get_n_channels(pixbuf);
    printf("Overlay has %d channels,
destination has %d channels\n",
           overlay_n_channels, n_channels);
    overlay_width = gdk_pixbuf_get_width
(overlay_pixbuf);
    overlay_height = gdk_pixbuf_get_height
(overlay_pixbuf);

    overlay_rowstride =
gdk_pixbuf_get_rowstride (overlay_pixbuf);
    overlay_pixels = gdk_pixbuf_get_pixels
(overlay_pixbuf);

    rowstride = gdk_pixbuf_get_rowstride
(pixbuf);
    width = gdk_pixbuf_get_width (pixbuf);
    pixels = gdk_pixbuf_get_pixels (pixbuf);

    printf("Overlay: width %d str8ide %d\n",
overlay_width, overlay_rowstride);
    printf("Dest: width  str8ide %d\n",
rowstride);
```

```
    for (m = 0; m < overlay_width; m++) {
        for (n = 0; n < overlay_height; n++) {
            overlay_p = overlay_pixels + n *
overlay_rowstride + m * overlay_n_channels;
            red = overlay_p[0];
            green = overlay_p[1];
            blue = overlay_p[2];
            if (overlay_n_channels == 4)
        alpha = overlay_p[3];
        else
        alpha = 0;

            p = pixels + (n+height_offset) *
rowstride + (m+width_offset) * n_channels;
            p[0] = red;
            p[1] = green;
            p[2] = blue;
            if (n_channels == 4)
                p[3] = alpha;
        }
    }
}
#endif /* GTK_MAJOR_VERSION == 3 */

static void *play_background(void *args) {
    /* based on code from
```

```
http://www.cs.dartmouth.edu/~xy/cs23/gtk.html

http://cdry.wordpress.com/2009/09/09/using-
custom-io-callbacks-with-ffmpeg/
    */

    int i;
    AVPacket packet;
    int frameFinished;
    AVFrame *pFrame = NULL;

    int bytesDecoded;
    GdkPixbuf *pixbuf;
    GdkPixbuf *overlay_pixbuf;
    AVFrame *picture_RGB;
    char *buffer;

    GError *error = NULL;
    overlay_pixbuf =
gdk_pixbuf_new_from_file(OVERLAY_IMAGE,
&error);
    if (!overlay_pixbuf) {
    fprintf(stderr, "%s\n", error->message);
    g_error_free(error);
    exit(1);
    }
    int overlay_width =
```

```
gdk_pixbuf_get_width(overlay_pixbuf);
    int overlay_height =
gdk_pixbuf_get_height(overlay_pixbuf);

    pFrame=avcodec_alloc_frame();

    i=0;
    picture_RGB = avcodec_alloc_frame();
    buffer = malloc
(avpicture_get_size(PIX_FMT_RGB24, 720, 576));
    avpicture_fill((AVPicture *)picture_RGB,
buffer, PIX_FMT_RGB24, 720, 576);

    while(av_read_frame(pFormatCtx,
&packet)>=0) {
    if(packet.stream_index==videoStream) {
        usleep(33670);  // 29.7 frames per
second
        // Decode video frame
        avcodec_decode_video2(pCodecCtx,
pFrame, &frameFinished,
                  &packet);
        int width = pCodecCtx->width;
        int height = pCodecCtx->height;

        sws_ctx = sws_getContext(width, height,
pCodecCtx->pix_fmt, width, height,
```

```
                        PIX_FMT_RGB24,
SWS_BICUBIC, NULL, NULL, NULL);

        if (frameFinished) {
        printf("Frame %d\n", i++);

        sws_scale(sws_ctx,  (uint8_t const *
const *) pFrame->data, pFrame->linesize, 0,
height, picture_RGB->data, picture_RGB-
>linesize);

        printf("old width %d new width %d\n",
pCodecCtx->width, picture_RGB->width);
        pixbuf =
gdk_pixbuf_new_from_data(picture_RGB->data[0],
GDK_COLORSPACE_RGB,
                            0, 8, width, height,
                            picture_RGB-
>linesize[0], pixmap_destroy_notify,
                            NULL);



        /* start GTK thread lock for drawing */
        gdk_threads_enter();

#if  GTK_MAJOR_VERSION == 2
```

```
        pixmap = gdk_pixmap_new(window->window,
720, 480, -1);

        gdk_draw_pixbuf((GdkDrawable *) pixmap,
NULL,
                pixbuf,
                0, 0, 0, 0, 720, 480,
                GDK_RGB_DITHER_NORMAL, 0, 0);

        // overlay another pixbuf
        gdk_draw_pixbuf((GdkDrawable *) pixmap,
NULL,
                                overlay_pixbuf,
                                0, 0, 0, 0,
overlay_width, overlay_height,

GDK_RGB_DITHER_NORMAL, 0, 0);

        gtk_image_set_from_pixmap((GtkImage*)
image, pixmap, NULL);

        gtk_widget_queue_draw(image);
#elif GTK_MAJOR_VERSION == 3
        overlay(pixbuf, overlay_pixbuf, 300,
200);

        gtk_image_set_from_pixbuf((GtkImage*)
```

```
image, pixbuf);

        gtk_widget_queue_draw(image);
#endif
        //g_object_unref(pixbuf);
        //sws_freeContext(sws_ctx);
        /* release GTK thread lock */
        gdk_threads_leave();
        }
    }
    av_free_packet(&packet);
    }

    printf("Video over!\n");
    exit(0);
}

/* Called when the windows are realized
 */
static void realize_cb (GtkWidget *widget,
gpointer data) {
    /* start the video playing in its own
thread */
    pthread_t tid;
    pthread_create(&tid, NULL, play_background,
NULL);
}
```

```
static gboolean delete_event( GtkWidget
*widget,
                              GdkEvent  *event,
                              gpointer   data )
{
    /* If you return FALSE in the "delete-
event" signal handler,
     * GTK will emit the "destroy" signal.
Returning TRUE means
     * you don't want the window to be
destroyed.
     * This is useful for popping up 'are you
sure you want to quit?'
     * type dialogs. */

    g_print ("delete event occurred\n");

    /* Change TRUE to FALSE and the main window
will be destroyed with
     * a "delete-event". */
    return TRUE;
}

/* Another callback */
static void destroy( GtkWidget *widget,
                     gpointer   data )
```

```
{
    gtk_main_quit ();
}

int main(int argc, char** argv)
{
    // Is this necessary?
    XInitThreads();

    int i;


    /* FFMpeg stuff */

    AVFrame *pFrame = NULL;
    AVPacket packet;

    AVDictionary *optionsDict = NULL;

    av_register_all();

    if(avformat_open_input(&pFormatCtx, VIDEO,
NULL, NULL)!=0)
    return -1; // Couldn't open file

    // Retrieve stream information
    if(avformat_find_stream_info(pFormatCtx,
```

```
NULL)<0)
    return -1; // Couldn't find stream
information

    // Dump information about file onto
standard error
    av_dump_format(pFormatCtx, 0, argv[1], 0);

    // Find the first video stream
    videoStream=-1;
    for(i=0; i<pFormatCtx->nb_streams; i++)
    if(pFormatCtx->streams[i]->codec-
>codec_type==AVMEDIA_TYPE_VIDEO) {
        videoStream=i;
        break;
    }
    if(videoStream==-1)
    return -1; // Didn't find a video stream

    for(i=0; i<pFormatCtx->nb_streams; i++)
    if(pFormatCtx->streams[i]->codec-
>codec_type==AVMEDIA_TYPE_AUDIO) {
        printf("Found an audio stream too\n");
        break;
    }
```

```
    // Get a pointer to the codec context for
the video stream
    pCodecCtx=pFormatCtx->streams[videoStream]-
>codec;

    // Find the decoder for the video stream
    pCodec=avcodec_find_decoder(pCodecCtx-
>codec_id);
    if(pCodec==NULL) {
    fprintf(stderr, "Unsupported codec!\n");
    return -1; // Codec not found
    }

    // Open codec
    if(avcodec_open2(pCodecCtx, pCodec,
&optionsDict)<0)
    return -1; // Could not open codec

    sws_ctx =
    sws_getContext
    (
     pCodecCtx->width,
     pCodecCtx->height,
     pCodecCtx->pix_fmt,
     pCodecCtx->width,
     pCodecCtx->height,
     PIX_FMT_YUV420P,
```

```
     SWS_BILINEAR,
     NULL,
     NULL,
     NULL
     );

    /* GTK stuff now */

    /* This is called in all GTK applications.
Arguments are parsed
     * from the command line and are returned
to the application. */
    gtk_init (&argc, &argv);

    /* create a new window */
    window = gtk_window_new
(GTK_WINDOW_TOPLEVEL);

    /* When the window is given the "delete-
event" signal (this is given
     * by the window manager, usually by the
"close" option, or on the
     * titlebar), we ask it to call the
delete_event () function
     * as defined above. The data passed to the
callback
     * function is NULL and is ignored in the
```

```
callback function. */
    g_signal_connect (window, "delete-event",
              G_CALLBACK (delete_event), NULL);

    /* Here we connect the "destroy" event to a
signal handler.
     * This event occurs when we call
gtk_widget_destroy() on the window,
     * or if we return FALSE in the "delete-
event" callback. */
    g_signal_connect (window, "destroy",
              G_CALLBACK (destroy), NULL);

    g_signal_connect (window, "realize",
G_CALLBACK (realize_cb), NULL);

    /* Sets the border width of the window. */
    gtk_container_set_border_width
(GTK_CONTAINER (window), 10);

    image = gtk_image_new();
    gtk_widget_show (image);

    /* This packs the button into the window (a
gtk container). */
    gtk_container_add (GTK_CONTAINER (window),
image);
```

```
    /* and the window */
    gtk_widget_show (window);

    /* All GTK applications must have a
gtk_main(). Control ends here
     * and waits for an event to occur (like a
key press or
     * mouse event). */
    gtk_main ();

    return 0;
}
```

Alpha channel

An overlay image may have some "transparent" parts in it. You don't want such parts to be overlaid onto the underlying image. But such parts will need to have a value in the array of pixels. Even zero is a value - black! Some images will have another byte per pixel allocated as the "alpha channel". This has a value to show how "transparent" the pixel is. A value of 255 means not transparent at all, a value of zero means totally transparent.

The simplest way of combining a "transparent" pixel with the underlying pixel is simply to not do so: leave the underlying pixel untouched. More complex algorithms are pointed to by the Wikipedia Alpha compositing page.

Converting an image which doesn't have an alpha channel to one which does can be done using the function `gdk_pixbuf_add_alpha`. This can also be used to set the value of the alpha channel by matching against a colour. For example, the following should set the alpha value to 0 for any white pixels and to 255 for all others:

```
pixbuf = gdk_pixbuf_add_alpha(pixbuf, TRUE,
255, 255, 255);
```

Unfortunately it seems to want to leave an "edge" of pixels which should be marked as transparent.

With alpha marking in place, a simple test can be used in the overlay function as to whether or not to perform the overlay:

```
if (alpha < 128) {
    continue;
 }
```

It's not worth giving a complete program just for a couple of changed lines. It is gtk_play_video_overlay_alpha.c .

Using Cairo to draw on an image

With the disappearance of pixmaps from Gtk 3.0, Cairo is now the only real way of assembling multiple components into an image. General Cairo information is at Cairo documentation , a tutorial is at Cairo graphics tutorial and information about overlaying onto images is at Images in Cairo .

Cairo takes sources and a destination. The sources can be changed, and frequently are: from an image source, to a colour source, etc. The destination is where the drawn stuff ends up.

Destinations can be in memory or at a variety of backends. We want in an in-memory destination so that we can extract a pixbuf from it, with all operations done on the client side. We create a destination as a *surface* of type `cairo_surface_t` and set it into a *Cairo context* of type `cairo_t` by

```
cairo_surface_t *surface =
cairo_image_surface_create
(CAIRO_FORMAT_ARGB32,
                                width, height);
cairo_t *cr = cairo_create(surface);
```

The cairo context `cr` is then used to set sources, perform drawing, etc. At the end of this we will extract a pixmap from the `surface`.

The first step is to set the source to the pixbuf for each frame of the video and to `paint` this to the destination by

```
gdk_cairo_set_source_pixbuf(cr, pixbuf, 0, 0);
cairo_paint (cr);
```

We can overlay another image on top of this by changing the source to the overlay image and painting that:

```
gdk_cairo_set_source_pixbuf(cr, overlay_pixbuf,
300, 200);
cairo_paint (cr);
```

Note that Cairo will do any alpha blending that is required if the overlay has "transparent" pixels.

To draw the text, we need to reset the source to an RGB surface, set all the parameters for the text, and draw the text into the destination. This is done by

```
// white text
cairo_set_source_rgb(cr, 1.0, 1.0, 1.0);
// this is a standard font for Cairo
cairo_select_font_face (cr, "cairo:serif",
            CAIRO_FONT_SLANT_NORMAL,
            CAIRO_FONT_WEIGHT_BOLD);
cairo_set_font_size (cr, 20);
cairo_move_to(cr, 10.0, 50.0);
cairo_show_text (cr, "hello");
```

Finally we want to extract the fnal image from the destination and set it into the `GdkImage` for display. Here there is another difference between Gtk 2.0 and Gtk 3.0: Gtk 3.0 has a function `gdk_pixbuf_get_from_surface` which will return a `GdKPixbuf` ; Gtk 2.0 has no such function. We only look at the Gtk 3.0 version for now

```
pixbuf = gdk_pixbuf_get_from_surface(surface,
                                     0,
                                     0,
                                     width,
                                     height);

gtk_image_set_from_pixbuf((GtkImage*) image,
pixbuf);
```

The complete program is `gtk_play_video_cairo.c`

```c
#include <gtk/gtk.h>

#include <libavcodec/avcodec.h>
#include <libavformat/avformat.h>
#include <libswscale/swscale.h>

#define OVERLAY_IMAGE "jan-small.png"

GtkWidget *image;
GtkWidget *window;

#if GTK_MAJOR_VERSION == 2
GdkPixmap *pixmap;
#endif

/* FFMpeg vbls */
AVFormatContext *pFormatCtx = NULL;
AVCodecContext *pCodecCtx = NULL;
int videoStream;
struct SwsContext *sws_ctx = NULL;
AVCodec *pCodec = NULL;



static void pixmap_destroy_notify(guchar
*pixels,
```

```
                    gpointer data) {
    printf("Destroy pixmap - not sure how\n");
}

static void *play_background(void *args) {
    /* based on code from

http://www.cs.dartmouth.edu/~xy/cs23/gtk.html

http://cdry.wordpress.com/2009/09/09/using-
custom-io-callbacks-with-ffmpeg/
    */

    int i;
    AVPacket packet;
    int frameFinished;
    AVFrame *pFrame = NULL;

    int bytesDecoded;
    GdkPixbuf *pixbuf;
    GdkPixbuf *overlay_pixbuf;
    AVFrame *picture_RGB;
    char *buffer;

    GError *error = NULL;
    overlay_pixbuf =
gdk_pixbuf_new_from_file(OVERLAY_IMAGE,
```

```
&error);
    if (!overlay_pixbuf) {
    fprintf(stderr, "%s\n", error->message);
    g_error_free(error);
    exit(1);
    }

    // add an alpha layer for a white
background
    overlay_pixbuf =
gdk_pixbuf_add_alpha(overlay_pixbuf, TRUE, 255,
255, 255);

    int overlay_width =
gdk_pixbuf_get_width(overlay_pixbuf);
    int overlay_height =
gdk_pixbuf_get_height(overlay_pixbuf);

    pFrame=avcodec_alloc_frame();

    i=0;
    picture_RGB = avcodec_alloc_frame();
    buffer = malloc
(avpicture_get_size(PIX_FMT_RGB24, 720, 576));
    avpicture_fill((AVPicture *)picture_RGB,
buffer, PIX_FMT_RGB24, 720, 576);
```

```
    while(av_read_frame(pFormatCtx,
&packet)>=0) {
    if(packet.stream_index==videoStream) {
        usleep(33670);  // 29.7 frames per
second
        // Decode video frame
        avcodec_decode_video2(pCodecCtx,
pFrame, &frameFinished,
                  &packet);

        int width = pCodecCtx->width;
        int height = pCodecCtx->height;

        sws_ctx = sws_getContext(width, height,
pCodecCtx->pix_fmt, width, height,
                     PIX_FMT_RGB24,
SWS_BICUBIC, NULL, NULL, NULL);

        if (frameFinished) {
        printf("Frame %d\n", i++);

        sws_scale(sws_ctx,  (uint8_t const *
const *) pFrame->data, pFrame->linesize, 0,
height, picture_RGB->data, picture_RGB-
>linesize);

        printf("old width %d new width %d\n",
```

```
pCodecCtx->width, picture_RGB->width);
        pixbuf =
gdk_pixbuf_new_from_data(picture_RGB->data[0],
GDK_COLORSPACE_RGB,
                                0, 8, width, height,
                                picture_RGB-
>linesize[0], pixmap_destroy_notify,
                                NULL);

        // Create the destination surface
        cairo_surface_t *surface =
cairo_image_surface_create
(CAIRO_FORMAT_ARGB32,
                                            width,
height);
        cairo_t *cr = cairo_create(surface);

        // draw the background image
        gdk_cairo_set_source_pixbuf(cr, pixbuf,
0, 0);
        cairo_paint (cr);

        // overlay an image on top
        // alpha blending will be done by Cairo
        gdk_cairo_set_source_pixbuf(cr,
overlay_pixbuf, 300, 200);
        cairo_paint (cr);
```

```
        // draw some white text on top
        cairo_set_source_rgb(cr, 1.0, 1.0,
1.0);
        // this is a standard font for Cairo
        cairo_select_font_face (cr,
"cairo:serif",
                    CAIRO_FONT_SLANT_NORMAL,
                    CAIRO_FONT_WEIGHT_BOLD);
        cairo_set_font_size (cr, 20);
        cairo_move_to(cr, 10.0, 50.0);
        cairo_show_text (cr, "hello");

        /* start GTK thread lock for drawing */
        gdk_threads_enter();

#if  GTK_MAJOR_VERSION == 2
        int pwidth, pheight, stride;
        unsigned char *data;

        data = cairo_image_surface_get_data
(surface);
        pwidth = cairo_image_surface_get_width
(surface);
        pheight =
cairo_image_surface_get_height (surface);
        stride = cairo_image_surface_get_stride
```

```
(surface);

        // this function doesn't work properly
        // code doesn't work
        pixmap =
gdk_pixmap_create_from_data(NULL, data,
                            pwidth, pheight,
                            8, NULL, NULL);

        gtk_image_set_from_pixmap((GtkImage*)
image, pixmap, NULL);

        gtk_widget_queue_draw(image);
#elif GTK_MAJOR_VERSION == 3
        pixbuf =
gdk_pixbuf_get_from_surface(surface,
                                0,
                                0,
                                width,
                                height);

        gtk_image_set_from_pixbuf((GtkImage*)
image, pixbuf);

        //gtk_widget_queue_draw(image);
#endif
        /* reclaim memory */
```

```
        g_object_unref(pixbuf);
        //sws_freeContext(sws_ctx);
        cairo_surface_destroy(surface);
        cairo_destroy(cr);

        /* release GTK thread lock */
        gdk_threads_leave();
        }
    }
    av_free_packet(&packet);
    }

    printf("Video over!\n");
    exit(0);
}

/* Called when the windows are realized
 */
static void realize_cb (GtkWidget *widget,
gpointer data) {
    /* start the video playing in its own
thread */
    pthread_t tid;
    pthread_create(&tid, NULL, play_background,
NULL);
}
```

```
static gboolean delete_event( GtkWidget
*widget,
                              GdkEvent  *event,
                              gpointer   data )
{
    /* If you return FALSE in the "delete-
event" signal handler,
     * GTK will emit the "destroy" signal.
Returning TRUE means
     * you don't want the window to be
destroyed.
     * This is useful for popping up 'are you
sure you want to quit?'
     * type dialogs. */

    g_print ("delete event occurred\n");

    /* Change TRUE to FALSE and the main window
will be destroyed with
     * a "delete-event". */
    return TRUE;
}

/* Another callback */
static void destroy( GtkWidget *widget,
                     gpointer   data )
{
```

```
    gtk_main_quit ();
}

int main(int argc, char** argv)
{
    // Is this necessary?
    XInitThreads();

    int i;


    /* FFMpeg stuff */

    AVFrame *pFrame = NULL;
    AVPacket packet;

    AVDictionary *optionsDict = NULL;

    av_register_all();

    if(avformat_open_input(&pFormatCtx,
"short.mpg", NULL, NULL)!=0)
    return -1; // Couldn't open file

    // Retrieve stream information
    if(avformat_find_stream_info(pFormatCtx,
NULL)<0)
```

```
    return -1; // Couldn't find stream
information

    // Dump information about file onto
standard error
    av_dump_format(pFormatCtx, 0, argv[1], 0);

    // Find the first video stream
    videoStream=-1;
    for(i=0; i<pFormatCtx->nb_streams; i++)
    if(pFormatCtx->streams[i]->codec-
>codec_type==AVMEDIA_TYPE_VIDEO) {
        videoStream=i;
        break;
    }
    if(videoStream==-1)
    return -1; // Didn't find a video stream

    for(i=0; i<pFormatCtx->nb_streams; i++)
    if(pFormatCtx->streams[i]->codec-
>codec_type==AVMEDIA_TYPE_AUDIO) {
        printf("Found an audio stream too\n");
        break;
    }


    // Get a pointer to the codec context for
```

```
the video stream
    pCodecCtx=pFormatCtx->streams[videoStream]-
>codec;

    // Find the decoder for the video stream
    pCodec=avcodec_find_decoder(pCodecCtx-
>codec_id);
    if(pCodec==NULL) {
    fprintf(stderr, "Unsupported codec!\n");
    return -1; // Codec not found
    }

    // Open codec
    if(avcodec_open2(pCodecCtx, pCodec,
&optionsDict)<0)
    return -1; // Could not open codec

    sws_ctx =
    sws_getContext
    (
     pCodecCtx->width,
     pCodecCtx->height,
     pCodecCtx->pix_fmt,
     pCodecCtx->width,
     pCodecCtx->height,
     PIX_FMT_YUV420P,
     SWS_BILINEAR,
```

```
     NULL,
     NULL,
     NULL
     );

    /* GTK stuff now */

    /* This is called in all GTK applications.
Arguments are parsed
     * from the command line and are returned
to the application. */
    gtk_init (&argc, &argv);

    /* create a new window */
    window = gtk_window_new
(GTK_WINDOW_TOPLEVEL);

    /* When the window is given the "delete-
event" signal (this is given
     * by the window manager, usually by the
"close" option, or on the
     * titlebar), we ask it to call the
delete_event () function
     * as defined above. The data passed to the
callback
     * function is NULL and is ignored in the
callback function. */
```

```
    g_signal_connect (window, "delete-event",
              G_CALLBACK (delete_event), NULL);

    /* Here we connect the "destroy" event to a
signal handler.
     * This event occurs when we call
gtk_widget_destroy() on the window,
     * or if we return FALSE in the "delete-
event" callback. */
    g_signal_connect (window, "destroy",
              G_CALLBACK (destroy), NULL);

    g_signal_connect (window, "realize",
G_CALLBACK (realize_cb), NULL);

    /* Sets the border width of the window. */
    gtk_container_set_border_width
(GTK_CONTAINER (window), 10);

    image = gtk_image_new();
    gtk_widget_show (image);

    /* This packs the button into the window (a
gtk container). */
    gtk_container_add (GTK_CONTAINER (window),
image);
```

```
    /* and the window */
    gtk_widget_show (window);

    /* All GTK applications must have a
gtk_main(). Control ends here
     * and waits for an event to occur (like a
key press or
     * mouse event). */
    gtk_main ();

    return 0;
}
```

Drawing text using Pango

While Cairo can draw any form of text, the functions such as `cairo_show_text` do not have much flexibility. To draw in, say, multiple colours will involve much work. Pango is a library for handling all aspects of text. There is a Pango Reference Manual . A good tutorial is at The Pango connection: Part 2 .

The simplest way of colouring text (and some other effects) is to create the text marked up with HTML such as

```
gchar *markup_text = "<span
foreground=\"red\">hello </span><span
foreground=\"black\">world</span>";
```

which has "hello" in red and "world" in black. This is then parsed into the text itself "ed black" and a set of attribute markups.

```
gchar *markup_text = "<span
foreground=\"red\">hello </span><span
foreground=\"black\">world</span>";
PangoAttrList *attrs;
gchar *text;

pango_parse_markup (markup_text, -1,0, &attrs,
&text, NULL, NULL);
```

This can be rendered into a Cairo context by creating a `PangoLayout` from the Cairo context, laying out the text with its attributes in the Pango layout and then showing this layout in the Cairo context:

```
PangoLayout *layout;
PangoFontDescription *desc;

cairo_move_to(cr, 300.0, 50.0);
layout = pango_cairo_create_layout (cr);
pango_layout_set_text (layout, text, -1);
pango_layout_set_attributes(layout, attrs);
pango_cairo_update_layout (cr, layout);
pango_cairo_show_layout (cr, layout);
```

(Yes, there is a lot of jumping around between libraries in all of this!).

Just as before, once all content has been drawn into the Cairo context, it can be extracted as a pixbuf from the Cairo surface destination, set into the `GtkImage` and added to the Gtk event queue.

The complete program is `gtk_play_video_pango.c` :

```c
#include <gtk/gtk.h>

#include <libavcodec/avcodec.h>
#include <libavformat/avformat.h>
#include <libswscale/swscale.h>

#define OVERLAY_IMAGE "jan-small.png"

GtkWidget *image;
GtkWidget *window;

#if GTK_MAJOR_VERSION == 2
GdkPixmap *pixmap;
#endif

/* FFMpeg vbls */
AVFormatContext *pFormatCtx = NULL;
AVCodecContext *pCodecCtx = NULL;
int videoStream;
struct SwsContext *sws_ctx = NULL;
AVCodec *pCodec = NULL;



static void pixmap_destroy_notify(guchar
*pixels,
```

```
                    gpointer data) {
    printf("Destroy pixmap - not sure how\n");
}

static void *play_background(void *args) {
    /* based on code from

http://www.cs.dartmouth.edu/~xy/cs23/gtk.html

http://cdry.wordpress.com/2009/09/09/using-
custom-io-callbacks-with-ffmpeg/
    */

    int i;
    AVPacket packet;
    int frameFinished;
    AVFrame *pFrame = NULL;

    /* initialize packet, set data to NULL, let
the demuxer fill it */
    /*
http://ffmpeg.org/doxygen/trunk/doc_2examples_2de
example.html#a80 */
    av_init_packet(&packet);
    packet.data = NULL;
    packet.size = 0;
```

```
    int bytesDecoded;
    GdkPixbuf *pixbuf;
    GdkPixbuf *overlay_pixbuf;
    AVFrame *picture_RGB;
    char *buffer;

    // Pango marked up text, half red, half
black
    gchar *markup_text = "<span
foreground=\"red\">hello</span><span
foreground=\"black\">world</span>";
    PangoAttrList *attrs;
    gchar *text;

    pango_parse_markup (markup_text, -1,0,
&attrs, &text, NULL, NULL);



    GError *error = NULL;
    overlay_pixbuf =
gdk_pixbuf_new_from_file(OVERLAY_IMAGE,
&error);
    if (!overlay_pixbuf) {
    fprintf(stderr, "%s\n", error->message);
    g_error_free(error);
    exit(1);
    }
```

```
    // add an alpha layer for a white
background
    overlay_pixbuf =
gdk_pixbuf_add_alpha(overlay_pixbuf, TRUE, 255,
255, 255);

    int overlay_width =
gdk_pixbuf_get_width(overlay_pixbuf);
    int overlay_height =
gdk_pixbuf_get_height(overlay_pixbuf);

    pFrame=avcodec_alloc_frame();

    i=0;
    picture_RGB = avcodec_alloc_frame();
    buffer = malloc
(avpicture_get_size(PIX_FMT_RGB24, 720, 576));
    avpicture_fill((AVPicture *)picture_RGB,
buffer, PIX_FMT_RGB24, 720, 576);

    while(av_read_frame(pFormatCtx,
&packet)>=0) {
    if(packet.stream_index==videoStream) {
        usleep(33670);  // 29.7 frames per
second
        // Decode video frame
```

```
        avcodec_decode_video2(pCodecCtx,
pFrame, &frameFinished,
                  &packet);

        int width = pCodecCtx->width;
        int height = pCodecCtx->height;

        sws_ctx = sws_getContext(width, height,
pCodecCtx->pix_fmt, width, height,
                     PIX_FMT_RGB24,
SWS_BICUBIC, NULL, NULL, NULL);

        if (frameFinished) {
        printf("Frame %d\n", i++);

        sws_scale(sws_ctx,  (uint8_t const *
const *) pFrame->data, pFrame->linesize, 0,
height, picture_RGB->data, picture_RGB-
>linesize);

        printf("old width %d new width %d\n",
pCodecCtx->width, picture_RGB->width);
        pixbuf =
gdk_pixbuf_new_from_data(picture_RGB->data[0],
GDK_COLORSPACE_RGB,
                         0, 8, width, height,
                         picture_RGB-
```

```
>linesize[0], pixmap_destroy_notify,
                            NULL);

        // Create the destination surface
        cairo_surface_t *surface =
cairo_image_surface_create
(CAIRO_FORMAT_ARGB32,
                                        width,
height);
        cairo_t *cr = cairo_create(surface);

        // draw the background image
        gdk_cairo_set_source_pixbuf(cr, pixbuf,
0, 0);
        cairo_paint (cr);

        // overlay an image on top
        // alpha blending will be done by Cairo
        gdk_cairo_set_source_pixbuf(cr,
overlay_pixbuf, 300, 200);
        cairo_paint (cr);

        // draw some white text on top
        cairo_set_source_rgb(cr, 1.0, 1.0,
1.0);
        // this is a standard font for Cairo
        cairo_select_font_face (cr,
```

```
"cairo:serif",
                    CAIRO_FONT_SLANT_NORMAL,
                    CAIRO_FONT_WEIGHT_BOLD);
        cairo_set_font_size (cr, 20);
        cairo_move_to(cr, 10.0, 50.0);
        cairo_show_text (cr, "hello");

        // draw Pango text
        PangoLayout *layout;
        PangoFontDescription *desc;

        cairo_move_to(cr, 300.0, 50.0);
        layout = pango_cairo_create_layout
(cr);
        pango_layout_set_text (layout, text,
-1);
        pango_layout_set_attributes(layout,
attrs);
        pango_cairo_update_layout (cr, layout);
        pango_cairo_show_layout (cr, layout);


        /* start GTK thread lock for drawing */
        gdk_threads_enter();

#if  GTK_MAJOR_VERSION == 2
        int pwidth, pheight, stride;
```

```
        unsigned char *data;

        data = cairo_image_surface_get_data
(surface);
        pwidth = cairo_image_surface_get_width
(surface);
        pheight =
cairo_image_surface_get_height (surface);
        stride = cairo_image_surface_get_stride
(surface);

        // this function doesn't work properly
        // code doesn't work
        pixmap =
gdk_pixmap_create_from_data(NULL, data,
                            pwidth, pheight,
                            8, NULL, NULL);

        gtk_image_set_from_pixmap((GtkImage*)
image, pixmap, NULL);

        gtk_widget_queue_draw(image);
#elif GTK_MAJOR_VERSION == 3
        pixbuf =
gdk_pixbuf_get_from_surface(surface,
                                0,
                                0,
```

```
                                width,
                                height);

        gtk_image_set_from_pixbuf((GtkImage*)
image, pixbuf);

        gtk_widget_queue_draw(image);
#endif
        /* reclaim memory */
        g_object_unref(pixbuf);
        //sws_freeContext(sws_ctx);
        g_object_unref(layout);
        cairo_surface_destroy(surface);
        cairo_destroy(cr);

        /* release GTK thread lock */
        gdk_threads_leave();
        }
    }
    av_free_packet(&packet);
    }

    printf("Video over!\n");
    exit(0);
}

/* Called when the windows are realized
```

```
 */
static void realize_cb (GtkWidget *widget,
gpointer data) {
    /* start the video playing in its own
thread */
    pthread_t tid;
    pthread_create(&tid, NULL, play_background,
NULL);
}

static gboolean delete_event( GtkWidget
*widget,
                              GdkEvent  *event,
                              gpointer   data )
{
    /* If you return FALSE in the "delete-
event" signal handler,
     * GTK will emit the "destroy" signal.
Returning TRUE means
     * you don't want the window to be
destroyed.
     * This is useful for popping up 'are you
sure you want to quit?'
     * type dialogs. */

    g_print ("delete event occurred\n");
```

```
    /* Change TRUE to FALSE and the main window
will be destroyed with
     * a "delete-event". */
    return TRUE;
}

/* Another callback */
static void destroy( GtkWidget *widget,
                     gpointer   data )
{
    gtk_main_quit ();
}

int main(int argc, char** argv)
{
    // Is this necessary?
    XInitThreads();

    int i;



    /* FFMpeg stuff */

    AVFrame *pFrame = NULL;
    AVPacket packet;

    AVDictionary *optionsDict = NULL;
```

```
    av_register_all();

    if(avformat_open_input(&pFormatCtx,
"short.mpg", NULL, NULL)!=0)
    return -1; // Couldn't open file

    // Retrieve stream information
    if(avformat_find_stream_info(pFormatCtx,
NULL)<0)
    return -1; // Couldn't find stream
information

    // Dump information about file onto
standard error
    av_dump_format(pFormatCtx, 0, argv[1], 0);

    // Find the first video stream
    videoStream=-1;
    for(i=0; i<pFormatCtx->nb_streams; i++)
    if(pFormatCtx->streams[i]->codec-
>codec_type==AVMEDIA_TYPE_VIDEO) {
        videoStream=i;
        break;
    }
    if(videoStream==-1)
    return -1; // Didn't find a video stream
```

```
    for(i=0; i<pFormatCtx->nb_streams; i++)
    if(pFormatCtx->streams[i]->codec-
>codec_type==AVMEDIA_TYPE_AUDIO) {
        printf("Found an audio stream too\n");
        break;
    }


    // Get a pointer to the codec context for
the video stream
    pCodecCtx=pFormatCtx->streams[videoStream]-
>codec;

    // Find the decoder for the video stream
    pCodec=avcodec_find_decoder(pCodecCtx-
>codec_id);
    if(pCodec==NULL) {
    fprintf(stderr, "Unsupported codec!\n");
    return -1; // Codec not found
    }

    // Open codec
    if(avcodec_open2(pCodecCtx, pCodec,
&optionsDict)<0)
    return -1; // Could not open codec
```

```
    sws_ctx =
    sws_getContext
    (
     pCodecCtx->width,
     pCodecCtx->height,
     pCodecCtx->pix_fmt,
     pCodecCtx->width,
     pCodecCtx->height,
     PIX_FMT_YUV420P,
     SWS_BILINEAR,
     NULL,
     NULL,
     NULL
     );

    /* GTK stuff now */

    /* This is called in all GTK applications.
Arguments are parsed
     * from the command line and are returned
to the application. */
    gtk_init (&argc, &argv);

    /* create a new window */
    window = gtk_window_new
(GTK_WINDOW_TOPLEVEL);
```

```
    /* When the window is given the "delete-
event" signal (this is given
     * by the window manager, usually by the
"close" option, or on the
     * titlebar), we ask it to call the
delete_event () function
     * as defined above. The data passed to the
callback
     * function is NULL and is ignored in the
callback function. */
    g_signal_connect (window, "delete-event",
              G_CALLBACK (delete_event), NULL);

    /* Here we connect the "destroy" event to a
signal handler.
     * This event occurs when we call
gtk_widget_destroy() on the window,
     * or if we return FALSE in the "delete-
event" callback. */
    g_signal_connect (window, "destroy",
              G_CALLBACK (destroy), NULL);

    g_signal_connect (window, "realize",
G_CALLBACK (realize_cb), NULL);

    /* Sets the border width of the window. */
    gtk_container_set_border_width
```

```
(GTK_CONTAINER (window), 10);

    image = gtk_image_new();
    gtk_widget_show (image);

    /* This packs the button into the window (a
gtk container). */
    gtk_container_add (GTK_CONTAINER (window),
image);

    /* and the window */
    gtk_widget_show (window);

    /* All GTK applications must have a
gtk_main(). Control ends here
     * and waits for an event to occur (like a
key press or
     * mouse event). */
    gtk_main ();

    return 0;
}
```

Conclusion

Getting to grips with some aspects of the Gtk toolkit is not trivial. We will use some of this material in later chapters, which is why it has been pulled out of the sound sections of this book and placed in a "diversions" section. Those not interested in Linux sound may nevertheless find it useful.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

The Raspberry Pi (RPi) has a very underpowered CPU but a very powerful GPU. Using the GPU is not well documented. This chapter looks at drawing, rendering images, etc, using the GPU.

Resources

- openGL-RPi-tutorial
- [OpenMAX+GLES2 : Decode and render JPEG (with source code):

```
 jpeg_gles2]
(http://www.raspberrypi.org/forums/viewtopic.p
f=33&t=57721)
```

- Using OpenGL ES 2.0 on the Raspberry Pi without X windows
- KHR_image_base describes EGLImageKHR type
- OpenGL Programming/Modern OpenGL Tutorial Text Rendering 01
- Using OpenGL ES to Accelerate Apps with Legacy 2D GUIs
- glEGLImageTargetTexture2DOES
- Raspberry Pi VideoCore APIs
- OpenGL® ES 2.0 Programming Guide by Aaftab Munshi, Dan Ginsburg and Dave Shreiner
- opengles-book-samples

- TGA File Format Summary

Accessing the GPU

Most drawing using RPi's GPU seems to be based around OpenGL ES. In order to use this we need to get an EGLSurface upon which OpenGL ES calls can be made.

The EGLSurface is a generic type which has to be backed by a specific implementation for the device/hardware/etc which you want to draw into. The page Raspberry Pi VideoCore APIs gives a quick overview of the APIs involved. Basically, to get to the EGLSurface you have to use the Dispmanx window system, which is reportedly being deprecated but is still used in all the demos and code that I have seen.

The relevant Dispmanx calls are

```
DISPMANX_ELEMENT_HANDLE_T dispman_element;
    DISPMANX_DISPLAY_HANDLE_T dispman_display;
    DISPMANX_UPDATE_HANDLE_T dispman_update;
    VC_RECT_T dst_rect;
    VC_RECT_T src_rect;

   success = graphics_get_display_size(0 /* LCD
*/,
              &screen_width, &screen_height);
    assert( success >= 0 );

    dst_rect.x = 0;
    dst_rect.y = 0;
    dst_rect.width = screen_width;
    dst_rect.height = screen_height;

    src_rect.x = 0;
    src_rect.y = 0;
    src_rect.width = screen_width << 16;
    src_rect.height = screen_height << 16;

    dispman_display = vc_dispmanx_display_open(
0 /* LCD */);
    dispman_update = vc_dispmanx_update_start(
0 );
    dispman_element =
```

```
        vc_dispmanx_element_add(dispman_update, dispman_display,
                                0/*layer*/, &dst_rect, 0/*src*/,
                                &src_rect, DISPMANX_PROTECTION_NONE,
                                0 /*alpha*/, 0/*clamp*/, 0/*transform*/);
```

At the end of this you have a window stored in `dispman_element` that can be used as a native window object later.

Initialising EGL is done in the standard way of

- Get an EGL display
- Initialise the display
- Choose a frame buffer
- Configure the frame buffer
- Create a rendering context
- Create an EGL window surface

Apart from the last step, this follows standard EGL programming:

```
static const EGLint attribute_list[] =
        {
            EGL_RED_SIZE, 8,
            EGL_GREEN_SIZE, 8,
            EGL_BLUE_SIZE, 8,
            EGL_ALPHA_SIZE, 8,
            EGL_SURFACE_TYPE, EGL_WINDOW_BIT,
            EGL_NONE
        };

    static const EGLint context_attributes[] =
        {
            EGL_CONTEXT_CLIENT_VERSION, 2,
            EGL_NONE
        };

    EGLConfig config;

    // get an EGL display connection
   display =
eglGetDisplay(EGL_DEFAULT_DISPLAY);

    // initialize the EGL display connection
    result = eglInitialize(display, NULL,
NULL);
```

```
    // get an appropriate EGL frame buffer
configuration
    result = eglChooseConfig(display,
attribute_list, &config, 1, &num_config);
    assert(EGL_FALSE != result);

    // get an appropriate EGL frame buffer
configuration
    result = eglBindAPI(EGL_OPENGL_ES_API);
    assert(EGL_FALSE != result);


    // create an EGL rendering context
    context = eglCreateContext(display, config,
EGL_NO_CONTEXT, context_attributes);
    assert(context!=EGL_NO_CONTEXT);
```

The next step is to link the Dispmanx window to the EGL window surface. This uses a structure of type `EGL_DISPMANX_WINDOW_T` which is filled in from the Dispmanx information:

```
EGL_DISPMANX_WINDOW_T nativewindow;

   nativewindow.element = dispman_element;
   nativewindow.width = screen_width;
   nativewindow.height = screen_height;
   vc_dispmanx_update_submit_sync(
dispman_update );
```

The EGL surface is then created by

```
surface = eglCreateWindowSurface(display,
config, &nativewindow, NULL);

// connect the context to the surface
eglMakeCurrent(display, surface, surface,
context);
```

In this section we do the absolute minimum: having got an EGL surface talking to the GPU, we just use OpenGL ES calls to set the background of the buffer to red and then display the buffer by swapping EGL buffers:

```
glClearColor(1.0, 0.0, 0.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();

    eglSwapBuffers(display, surface);
```

The complete program is rectangle.c and just displays a large red square:

```c
/*
 * code stolen from openGL-RPi-tutorial-
master/encode_OGL/
 */

#include <stdio.h>
#include <assert.h>
#include <math.h>

#include <EGL/egl.h>
#include <EGL/eglext.h>
#include <GLES2/gl2.h>

typedef struct
{
    uint32_t screen_width;
    uint32_t screen_height;

    EGLDisplay display;
    EGLSurface surface;
    EGLContext context;
} CUBE_STATE_T;


CUBE_STATE_T state, *p_state = &state;
```

```
void init_ogl(CUBE_STATE_T *state)
{
    int32_t success = 0;
    EGLBoolean result;
    EGLint num_config;

    bcm_host_init();

    static EGL_DISPMANX_WINDOW_T nativewindow;

    DISPMANX_ELEMENT_HANDLE_T dispman_element;
    DISPMANX_DISPLAY_HANDLE_T dispman_display;
    DISPMANX_UPDATE_HANDLE_T dispman_update;
    VC_RECT_T dst_rect;
    VC_RECT_T src_rect;

    static const EGLint attribute_list[] =
    {
        EGL_RED_SIZE, 8,
        EGL_GREEN_SIZE, 8,
        EGL_BLUE_SIZE, 8,
        EGL_ALPHA_SIZE, 8,
        EGL_SURFACE_TYPE, EGL_WINDOW_BIT,
        EGL_NONE
    };

    static const EGLint context_attributes[] =
```

```
    {
        EGL_CONTEXT_CLIENT_VERSION, 2,
        EGL_NONE
    };

    EGLConfig config;

    // get an EGL display connection
    state->display =
eglGetDisplay(EGL_DEFAULT_DISPLAY);

    // initialize the EGL display connection
    result = eglInitialize(state->display,
NULL, NULL);

    // get an appropriate EGL frame buffer
configuration
    result = eglChooseConfig(state->display,
attribute_list, &config, 1, &num_config);
    assert(EGL_FALSE != result);

    // get an appropriate EGL frame buffer
configuration
    result = eglBindAPI(EGL_OPENGL_ES_API);
    assert(EGL_FALSE != result);
```

```
    // create an EGL rendering context
    state->context = eglCreateContext(state-
>display, config, EGL_NO_CONTEXT,
context_attributes);
    assert(state->context!=EGL_NO_CONTEXT);

    // create an EGL window surface
    success = graphics_get_display_size(0 /*
LCD */, &state->screen_width, &state-
>screen_height);
    assert( success >= 0 );

    state->screen_width = 1024;

    state->screen_height = 1024;

    dst_rect.x = 0;
    dst_rect.y = 0;
    dst_rect.width = state->screen_width;
    dst_rect.height = state->screen_height;

    src_rect.x = 0;
    src_rect.y = 0;
    src_rect.width = state->screen_width << 16;
    src_rect.height = state->screen_height <<
16;
```

```
    dispman_display = vc_dispmanx_display_open(
0 /* LCD */);
    dispman_update = vc_dispmanx_update_start(
0 );

    dispman_element =
    vc_dispmanx_element_add(dispman_update,
dispman_display,
                0/*layer*/, &dst_rect,
0/*src*/,
                &src_rect,
DISPMANX_PROTECTION_NONE,
                0 /*alpha*/, 0/*clamp*/,
0/*transform*/);

    nativewindow.element = dispman_element;
    nativewindow.width = state->screen_width;
    nativewindow.height = state->screen_height;
    vc_dispmanx_update_submit_sync(
dispman_update );

    state->surface = eglCreateWindowSurface(
state->display, config, &nativewindow, NULL );
    assert(state->surface != EGL_NO_SURFACE);

    // connect the context to the surface
    result = eglMakeCurrent(state->display,
```

```
state->surface, state->surface, state-
>context);
    assert(EGL_FALSE != result);
}

int
main(int argc, char *argv[])
{

    bcm_host_init();

    init_ogl(p_state);

    glClearColor(1.0, 0.0, 0.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();

    eglSwapBuffers(p_state->display, p_state-
>surface);

    while (1) {
    sleep(10);
    }

    return 0;
}
```

Compiling the program uses a horrendous mess of defines and libraries, probably not all of which are needed!

```
cc -g  -DUSE_OPENGL -DUSE_EGL -DIS_RPI -
DSTANDALONE -D__STDC_CONSTANT_MACROS -
D__STDC_LIMIT_MACROS -DTARGET_POSIX -D_LINUX -
fPIC -DPIC -D_REENTRANT -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -U_FORTIFY_SOURCE -Wall -
g -DHAVE_LIBOPENMAX=2 -DOMX -DOMX_SKIP64BIT -
ftree-vectorize -pipe -DUSE_EXTERNAL_OMX -
DHAVE_LIBBCM_HOST -DUSE_EXTERNAL_LIBBCM_HOST -
DUSE_VCHIQ_ARM -Wno-psabi -I/opt/vc/include/ -
I/opt/vc/include/interface/vcos/pthreads -
I/opt/vc/include/interface/vmcs_host/linux -I./
-I/opt/vc/src/hello_pi/libs/ilclient -
I/opt/vc/src/hello_pi/libs/vgfont -g -c
rectangle.c -o rectangle.o -Wno-deprecated-
declarations

cc -o rectangle -Wl,--whole-archive rectangle.o
-L/opt/vc/lib/ -lGLESv2 -lEGL -lbcm_host -lvcos
-lvchiq_arm -lpthread -lrt -
L/opt/vc/src/hello_pi/libs/vgfont -ldl -lm -
Wl,--no-whole-archive -rdynamic
```

However, it can then be run easily by

```
./rectangle
```

Drawing a triangle using OpenGL ES

The previous example did the absolute minimum of OpenGL calls, just clearing the background. In this setion we do a bit more, by using OpenGL calls to clear the background and repeatedly draw a triangle (the same one!) multiple times a second. The code is just adapted from Chapter 2 of the OpenGL ES 2.0 Programming Guide

The essential differences from the Programming Guide are

- We don't give any explanations of the OpenGL code - the Guide has an exhaustive description
- They hide the grubby details of building OpenGL programs in OS-specific modules such as `esUtils.c`, while we explicitly show the RPi stuff here

The program is triangle.c :

```
/*
 * code stolen from openGL-RPi-tutorial-
master/encode_OGL/
 * and from OpenGL® ES 2.0 Programming Guide
 */

#include <stdio.h>
#include <assert.h>
#include <math.h>
#include <sys/time.h>

#include <EGL/egl.h>
#include <EGL/eglext.h>
#include <GLES2/gl2.h>

typedef struct CUBE_STATE_T
{
    uint32_t width;
    uint32_t height;

    EGLDisplay display;
    EGLSurface surface;
    EGLContext context;

    EGL_DISPMANX_WINDOW_T nativewindow;
    void *user_data;
```

```
    void (*draw_func) (struct CUBE_STATE_T* );
} CUBE_STATE_T;

// from esUtil.h
#define TRUE 1
#define FALSE 0

typedef struct
{
    // Handle to a program object
    GLuint programObject;
} UserData;

///
// Create a shader object, load the shader
source, and
// compile the shader.
//
GLuint LoadShader(GLenum type, const char
*shaderSrc)
{
    GLuint shader;
    GLint compiled;
    // Create the shader object
    shader = glCreateShader(type);
    if(shader == 0)
    return 0;
```

```
    // Load the shader source
    glShaderSource(shader, 1, &shaderSrc,
NULL);
    // Compile the shader
    glCompileShader(shader);
    // Check the compile status
    glGetShaderiv(shader, GL_COMPILE_STATUS,
&compiled);
    if(!compiled)
    {
        GLint infoLen = 0;
        glGetShaderiv(shader,
GL_INFO_LOG_LENGTH, &infoLen);
        if(infoLen > 1)
        {
            char* infoLog = malloc(sizeof(char)
* infoLen);
            glGetShaderInfoLog(shader, infoLen,
NULL, infoLog);
            fprintf(stderr, "Error compiling
shader:\n%s\n", infoLog);
            free(infoLog);
        }
        glDeleteShader(shader);
        return 0;
    }
    return shader;
```

```
}
///
// Initialize the shader and program object
//
int Init(CUBE_STATE_T *p_state)
{
    UserData *user_data = p_state->user_data;
    GLbyte vShaderStr[] =
    "attribute vec4 vPosition;\n"
    "void main()\n"
    "{\n"
    "gl_Position = vPosition; \n"
    "}\n";
    GLbyte fShaderStr[] =
    "precision mediump float;\n"
    "void main()\n"
    "{\n"
    " gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
\n"
    "}\n";
    GLuint vertexShader;
    GLuint fragmentShader;
    GLuint programObject;
    GLint linked;

    // Load the vertex/fragment shaders
    vertexShader = LoadShader(GL_VERTEX_SHADER,
```

```
vShaderStr);
    fragmentShader =
LoadShader(GL_FRAGMENT_SHADER, fShaderStr);
    // Create the program object
    programObject = glCreateProgram();
    if(programObject == 0)
    return 0;
    glAttachShader(programObject,
vertexShader);
    glAttachShader(programObject,
fragmentShader);
    // Bind vPosition to attribute 0
    glBindAttribLocation(programObject, 0,
"vPosition");
    // Link the program
    glLinkProgram(programObject);
    // Check the link status
    glGetProgramiv(programObject,
GL_LINK_STATUS, &linked);
    if(!linked)
    {
        GLint infoLen = 0;
        glGetProgramiv(programObject,
GL_INFO_LOG_LENGTH, &infoLen);
        if(infoLen > 1)
        {
            char* infoLog = malloc(sizeof(char)
```

```
* infoLen);
            glGetProgramInfoLog(programObject,
infoLen, NULL, infoLog);
            fprintf(stderr, "Error linking
program:\n%s\n", infoLog);
            free(infoLog);
        }
        glDeleteProgram(programObject);
        return FALSE;
    }
    // Store the program object
    user_data->programObject = programObject;
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    return TRUE;
}

///
// Draw a triangle using the shader pair
created in Init()
//
void Draw(CUBE_STATE_T *p_state)
{
    UserData *user_data = p_state->user_data;
    GLfloat vVertices[] = {0.0f, 0.5f, 0.0f,
                -0.5f, -0.5f, 0.0f,
                0.5f, -0.5f, 0.0f};
    // Set the viewport
```

```
    glViewport(0, 0, p_state->width, p_state-
>height);
    // Clear the color buffer
    glClear(GL_COLOR_BUFFER_BIT);
    // Use the program object
    glUseProgram(user_data->programObject);
    // Load the vertex data
    glVertexAttribPointer(0, 3, GL_FLOAT,
GL_FALSE, 0, vVertices);
    glEnableVertexAttribArray(0);
    glDrawArrays(GL_TRIANGLES, 0, 3);
    eglSwapBuffers(p_state->display, p_state-
>surface);
}


CUBE_STATE_T state, *p_state = &state;

void init_ogl(CUBE_STATE_T *state, int width,
int height)
{
    int32_t success = 0;
    EGLBoolean result;
    EGLint num_config;

    bcm_host_init();
```

```
    DISPMANX_ELEMENT_HANDLE_T dispman_element;
    DISPMANX_DISPLAY_HANDLE_T dispman_display;
    DISPMANX_UPDATE_HANDLE_T dispman_update;
    VC_RECT_T dst_rect;
    VC_RECT_T src_rect;

    static const EGLint attribute_list[] =
    {
        EGL_RED_SIZE, 8,
        EGL_GREEN_SIZE, 8,
        EGL_BLUE_SIZE, 8,
        EGL_ALPHA_SIZE, 8,
        EGL_SURFACE_TYPE, EGL_WINDOW_BIT,
        EGL_NONE
    };

    static const EGLint context_attributes[] =
    {
        EGL_CONTEXT_CLIENT_VERSION, 2,
        EGL_NONE
    };

    EGLConfig config;

    // get an EGL display connection
    state->display =
eglGetDisplay(EGL_DEFAULT_DISPLAY);
```

```
    // initialize the EGL display connection
    result = eglInitialize(state->display,
NULL, NULL);

    // get an appropriate EGL frame buffer
configuration
    result = eglChooseConfig(state->display,
attribute_list, &config, 1, &num_config);
    assert(EGL_FALSE != result);

    // get an appropriate EGL frame buffer
configuration
    result = eglBindAPI(EGL_OPENGL_ES_API);
    assert(EGL_FALSE != result);


    // create an EGL rendering context
    state->context = eglCreateContext(state-
>display, config, EGL_NO_CONTEXT,
context_attributes);
    assert(state->context!=EGL_NO_CONTEXT);

    // create an EGL window surface
    success = graphics_get_display_size(0 /*
LCD */, &state->width, &state->height);
    assert( success >= 0 );
```

```
    state->width = width;
    state->height = height;

    dst_rect.x = 0;
    dst_rect.y = 0;
    dst_rect.width = state->width;
    dst_rect.height = state->height;

    src_rect.x = 0;
    src_rect.y = 0;
    src_rect.width = state->width << 16;
    src_rect.height = state->height << 16;

    dispman_display = vc_dispmanx_display_open(
0 /* LCD */);
    dispman_update = vc_dispmanx_update_start(
0 );

    dispman_element =
    vc_dispmanx_element_add(dispman_update,
dispman_display,
                0/*layer*/, &dst_rect,
0/*src*/,
                &src_rect,
DISPMANX_PROTECTION_NONE,
                0 /*alpha*/, 0/*clamp*/,
```

```
0/*transform*/);

    state->nativewindow.element =
dispman_element;
    state->nativewindow.width = state->width;
    state->nativewindow.height = state->height;
    vc_dispmanx_update_submit_sync(
dispman_update );

    state->surface = eglCreateWindowSurface(
state->display, config, &(state->nativewindow),
NULL );
    assert(state->surface != EGL_NO_SURFACE);

    // connect the context to the surface
    result = eglMakeCurrent(state->display,
state->surface, state->surface, state-
>context);
    assert(EGL_FALSE != result);
}

void esInitContext ( CUBE_STATE_T *p_state )
{
   if ( p_state != NULL )
   {
      memset( p_state, 0, sizeof( CUBE_STATE_T)
);
```

```
   }
}

void esRegisterDrawFunc(CUBE_STATE_T *p_state,
void (*draw_func) (CUBE_STATE_T* ) )
{
   p_state->draw_func = draw_func;
}

void  esMainLoop (CUBE_STATE_T *esContext )
{
    struct timeval t1, t2;
    struct timezone tz;
    float deltatime;
    float totaltime = 0.0f;
    unsigned int frames = 0;

    gettimeofday ( &t1 , &tz );

    while(1)
    {
        gettimeofday(&t2, &tz);
        deltatime = (float)(t2.tv_sec -
t1.tv_sec + (t2.tv_usec - t1.tv_usec) * 1e-6);
        t1 = t2;

        if (esContext->draw_func != NULL)
```

```
            esContext->draw_func(esContext);

        eglSwapBuffers(esContext->display,
esContext->surface);

        totaltime += deltatime;
        frames++;
        if (totaltime >  2.0f)
        {
            printf("%4d frames rendered in
%1.4f seconds -> FPS=%3.4f\n", frames,
totaltime, frames/totaltime);
            totaltime -= 2.0f;
            frames = 0;
        }
    }
}

int main(int argc, char *argv[])
{
    UserData user_data;

    bcm_host_init();
    esInitContext(p_state);
    init_ogl(p_state, 320, 240);

    p_state->user_data = &user_data;
```

```
    if(!Init(p_state))
    return 0;
    esRegisterDrawFunc(p_state, Draw);

    eglSwapBuffers(p_state->display, p_state-
>surface);
    esMainLoop(p_state);
}
```

Drawing an image using OpenGL ES

The OpenGL ES Programming Guide includes an example of drawing a simple image using an array of pixels. Realistically, it notes that the image data is more likely to be read from a file. In this section we fill in the details.

There are many, many different file formats, some lossy, some lossless, some compressed, some not, some with metadata, some without. In this section we just use TGA - an uncompressed format with enough useful metadata, simple to load.

TGA files can be created from e.g. JPEG files by using the `convert` utility from the Gimp drawing system. It is simple to convert a file: just give the appropriate file extensions:

```
convert image.jpg image.tga
```

(Later sections look at decompressing files using the GPU itself.)

A TGA file has a header section which gives the width and height of the image from which its size can be calculated. The default format will be RGB, as 24-bit pixels. Reading in such a

file is just a matter of locating the dimensions, malloc'ing the right size buffer, skipping to the start of the image data and reading it all in.

The default is for the origin of the image to be the bottom lefthand corner, with the y-axis growing up. OpenGL ES on the other hand has the origin in the top lefthand corner with the y-axis growing down. So the image will be upside down. This can be fixed by reading the data in differently, or by using an OpenGL ES reflection.

Getting a native window is done as before. Using the OpenGL ES Programming Guide example, we also need to substitute our image array for their 2x2 array, adjusting the image bounds. Apart from those, there is no real change. The program is image.c

```
/*
 * code stolen from openGL-RPi-tutorial-
master/encode_OGL/
 * and from OpenGL® ES 2.0 Programming Guide
 */

#include <stdio.h>
#include <assert.h>
#include <math.h>
#include <sys/time.h>
//#include "jpeg.h"

#include <EGL/egl.h>
#include <EGL/eglext.h>
#include <GLES2/gl2.h>


// from esUtil.h
#define TRUE 1
#define FALSE 0

typedef struct
{
    // Handle to a program object
    GLuint programObject;
```

```
    // Attribute locations
    GLint  positionLoc;
    GLint  texCoordLoc;

    // Sampler location
    GLint samplerLoc;

    // Texture handle
    GLuint textureId;

} UserData;

typedef struct CUBE_STATE_T
{
    uint32_t width;
    uint32_t height;

    EGLDisplay display;
    EGLSurface surface;
    EGLContext context;

    EGL_DISPMANX_WINDOW_T nativewindow;
    UserData *user_data;
    void (*draw_func) (struct CUBE_STATE_T* );
} CUBE_STATE_T;

char *image;
```

```
int tex;


char* esLoadTGA ( char *fileName, int *width,
int *height )
{
    char *buffer = NULL;
    FILE *f;
    unsigned char tgaheader[12];
    unsigned char attributes[6];
    unsigned int imagesize;

    f = fopen(fileName, "rb");
    if(f == NULL) return NULL;

    if(fread(&tgaheader, sizeof(tgaheader), 1,
f) == 0)
    {
        fclose(f);
        return NULL;
    }

    if(fread(attributes, sizeof(attributes), 1,
f) == 0)
    {
        fclose(f);
        return 0;
```

```
    }

    *width = attributes[1] * 256 +
attributes[0];
    *height = attributes[3] * 256 +
attributes[2];
    imagesize = attributes[4] / 8 * *width *
*height;
    //imagesize *= 4/3;
    printf("Origin bits: %d\n", attributes[5] &
030);
    printf("Pixel depth %d\n", attributes[4]);
    buffer = malloc(imagesize);
    if (buffer == NULL)
    {
        fclose(f);
        return 0;
    }

#if 1
    // invert - should be reflect, easier is
180 rotate
    int n = 1;
    while (n <= imagesize) {
    fread(&buffer[imagesize - n], 1, 1, f);
    n++;
    }
```

```
#else
    // as is - upside down
    if(fread(buffer, 1, imagesize, f) !=
imagesize)
    {
        free(buffer);
        return NULL;
    }
#endif
    fclose(f);
    return buffer;
}

///
// Create a simple width x height texture image
with four different colors
//
GLuint CreateSimpleTexture2D(int width, int
height )
{
   // Texture object handle
   GLuint textureId;

   // Use tightly packed data
   glPixelStorei ( GL_UNPACK_ALIGNMENT, 1 );

   // Generate a texture object
```

```
   glGenTextures ( 1, &textureId );

   // Bind the texture object
   glBindTexture ( GL_TEXTURE_2D, textureId );

   // Load the texture



   glTexImage2D ( GL_TEXTURE_2D, 0, GL_RGB,
          width, height,
          0, GL_RGB, GL_UNSIGNED_BYTE, image );

   // Set the filtering mode
   glTexParameteri ( GL_TEXTURE_2D,
GL_TEXTURE_MIN_FILTER, GL_NEAREST );
   glTexParameteri ( GL_TEXTURE_2D,
GL_TEXTURE_MAG_FILTER, GL_NEAREST );
   return textureId;
}



///
// Create a shader object, load the shader
source, and
// compile the shader.
//
```

```
GLuint LoadShader(GLenum type, const char
*shaderSrc)
{
    GLuint shader;
    GLint compiled;
    // Create the shader object
    shader = glCreateShader(type);
    if(shader == 0)
    return 0;
    // Load the shader source
    glShaderSource(shader, 1, &shaderSrc,
NULL);
    // Compile the shader
    glCompileShader(shader);
    // Check the compile status
    glGetShaderiv(shader, GL_COMPILE_STATUS,
&compiled);
    if(!compiled)
    {
        GLint infoLen = 0;
        glGetShaderiv(shader,
GL_INFO_LOG_LENGTH, &infoLen);
        if(infoLen > 1)
        {
            char* infoLog = malloc(sizeof(char)
* infoLen);
            glGetShaderInfoLog(shader, infoLen,
```

```
NULL, infoLog);
            fprintf(stderr, "Error compiling
shader:\n%s\n", infoLog);
            free(infoLog);
        }
        glDeleteShader(shader);
        return 0;
    }
    return shader;
}

GLuint LoadProgram ( const char *vertShaderSrc,
const char *fragShaderSrc )
{
   GLuint vertexShader;
   GLuint fragmentShader;
   GLuint programObject;
   GLint linked;

   // Load the vertex/fragment shaders
   vertexShader = LoadShader (
GL_VERTEX_SHADER, vertShaderSrc );
   if ( vertexShader == 0 )
      return 0;

   fragmentShader = LoadShader (
GL_FRAGMENT_SHADER, fragShaderSrc );
```

```
   if ( fragmentShader == 0 )
   {
      glDeleteShader( vertexShader );
      return 0;
   }

   // Create the program object
   programObject = glCreateProgram ( );

   if ( programObject == 0 )
      return 0;

   glAttachShader ( programObject, vertexShader
);
   glAttachShader ( programObject,
fragmentShader );

   // Link the program
   glLinkProgram ( programObject );

   // Check the link status
   glGetProgramiv ( programObject,
GL_LINK_STATUS, &linked );

   if ( !linked )
   {
      GLint infoLen = 0;
```

```
      glGetProgramiv ( programObject,
GL_INFO_LOG_LENGTH, &infoLen );

      if ( infoLen > 1 )
      {
         char* infoLog = malloc (sizeof(char) *
infoLen );

         glGetProgramInfoLog ( programObject,
infoLen, NULL, infoLog );
         fprintf (stderr, "Error linking
program:\n%s\n", infoLog );

         free ( infoLog );
      }

      glDeleteProgram ( programObject );
      return 0;
   }

   // Free up no longer needed shader resources
   glDeleteShader ( vertexShader );
   glDeleteShader ( fragmentShader );

   return programObject;
}
```

```
///
// Initialize the shader and program object
//
int Init(CUBE_STATE_T *p_state)
{

   p_state->user_data =
malloc(sizeof(UserData));
   UserData *userData = p_state->user_data;
   GLbyte vShaderStr[] =
      "attribute vec4 a_position;   \n"
      "attribute vec2 a_texCoord;   \n"
      "varying vec2 v_texCoord;     \n"
      "void main()                  \n"
      "{                            \n"
      "   gl_Position = a_position; \n"
      "   v_texCoord = a_texCoord;  \n"
      "}                            \n";

   GLbyte fShaderStr[] =
      "precision mediump float;
\n"
      "varying vec2 v_texCoord;
\n"
      "uniform sampler2D s_texture;
\n"
      "void main()
```

```
\n"
      "{
\n"
      "  gl_FragColor = texture2D( s_texture,
v_texCoord );\n"
      "}
\n";

   // Load the shaders and get a linked program
object
   userData->programObject = LoadProgram (
vShaderStr, fShaderStr );

   // Get the attribute locations
   userData->positionLoc = glGetAttribLocation
( userData->programObject, "a_position" );
   userData->texCoordLoc = glGetAttribLocation
( userData->programObject, "a_texCoord" );

   // Get the sampler location
   userData->samplerLoc = glGetUniformLocation
( userData->programObject, "s_texture" );
   // Load the texture
   userData->textureId = CreateSimpleTexture2D
(p_state->width, p_state->height);

   glClearColor ( 0.0f, 0.0f, 0.0f, 0.0f );
```

```
   return GL_TRUE;
}

///
// Draw triangles using the shader pair created
in Init()
//
void Draw(CUBE_STATE_T *p_state)
{
   UserData *userData = p_state->user_data;

   GLfloat vVertices[] = { -0.5f,  0.5f, 0.0f,
// Position 0
                            0.0f,  0.0f,
// TexCoord 0
                           -0.5f, -0.5f, 0.0f,
// Position 1
                            0.0f,  1.0f,
// TexCoord 1
                            0.5f, -0.5f, 0.0f,
// Position 2
                            1.0f,  1.0f,
// TexCoord 2
                            0.5f,  0.5f, 0.0f,
// Position 3
                            1.0f,  0.0f
// TexCoord 3
```

```
                            };

   GLushort indices[] = { 0, 1, 2, 0, 2, 3 };
   //GLushort indices[] = {1, 0, 3, 0, 2, 0, 1
};

   // Set the viewport
   glViewport ( 0, 0, p_state->width, p_state-
>height );

   // Clear the color buffer
   glClear ( GL_COLOR_BUFFER_BIT );

   // Use the program object
   glUseProgram ( userData->programObject );

   // Load the vertex position
   glVertexAttribPointer ( userData-
>positionLoc, 3, GL_FLOAT,
                            GL_FALSE, 5 *
sizeof(GLfloat), vVertices );
   // Load the texture coordinate
   glVertexAttribPointer ( userData-
>texCoordLoc, 2, GL_FLOAT,
                            GL_FALSE, 5 *
sizeof(GLfloat), &vVertices[3] );
```

```
   glEnableVertexAttribArray ( userData-
>positionLoc );
   glEnableVertexAttribArray ( userData-
>texCoordLoc );

   // Bind the texture
   glActiveTexture ( GL_TEXTURE0 );
   glBindTexture ( GL_TEXTURE_2D, userData-
>textureId );

   // Set the sampler texture unit to 0
   glUniform1i ( userData->samplerLoc, 0 );

   glDrawElements ( GL_TRIANGLES, 6,
GL_UNSIGNED_SHORT, indices );
   //glDrawElements ( GL_TRIANGLE_STRIP, 6,
GL_UNSIGNED_SHORT, indices );
}

CUBE_STATE_T state, *p_state = &state;

void init_ogl(CUBE_STATE_T *state, int width,
int height)
{
    int32_t success = 0;
    EGLBoolean result;
    EGLint num_config;
```

```
bcm_host_init();

DISPMANX_ELEMENT_HANDLE_T dispman_element;
DISPMANX_DISPLAY_HANDLE_T dispman_display;
DISPMANX_UPDATE_HANDLE_T dispman_update;
VC_RECT_T dst_rect;
VC_RECT_T src_rect;

static const EGLint attribute_list[] =
{
    EGL_RED_SIZE, 8,
    EGL_GREEN_SIZE, 8,
    EGL_BLUE_SIZE, 8,
    EGL_ALPHA_SIZE, 8,
    EGL_SURFACE_TYPE, EGL_WINDOW_BIT,
    EGL_NONE
};

static const EGLint context_attributes[] =
{
    EGL_CONTEXT_CLIENT_VERSION, 2,
    EGL_NONE
};

EGLConfig config;
```

```
    // get an EGL display connection
    state->display =
eglGetDisplay(EGL_DEFAULT_DISPLAY);

    // initialize the EGL display connection
    result = eglInitialize(state->display,
NULL, NULL);

    // get an appropriate EGL frame buffer
configuration
    result = eglChooseConfig(state->display,
attribute_list, &config, 1, &num_config);
    assert(EGL_FALSE != result);

    // get an appropriate EGL frame buffer
configuration
    result = eglBindAPI(EGL_OPENGL_ES_API);
    assert(EGL_FALSE != result);



    // create an EGL rendering context
    state->context = eglCreateContext(state-
>display, config, EGL_NO_CONTEXT,
context_attributes);
    assert(state->context!=EGL_NO_CONTEXT);

    // create an EGL window surface
```

```
    success = graphics_get_display_size(0 /*
LCD */, &state->width, &state->height);
    assert( success >= 0 );

    state->width = width;
    state->height = height;

    dst_rect.x = 0;
    dst_rect.y = 0;
    dst_rect.width = state->width;
    dst_rect.height = state->height;

    src_rect.x = 0;
    src_rect.y = 0;
    src_rect.width = state->width << 16;
    src_rect.height = state->height << 16;

    dispman_display = vc_dispmanx_display_open(
0 /* LCD */);
    dispman_update = vc_dispmanx_update_start(
0 );

    dispman_element =
    vc_dispmanx_element_add(dispman_update,
dispman_display,
                0/*layer*/, &dst_rect,
0/*src*/,
```

```
                &src_rect,
DISPMANX_PROTECTION_NONE,
                0 /*alpha*/, 0/*clamp*/,
0/*transform*/);

    state->nativewindow.element =
dispman_element;
    state->nativewindow.width = state->width;
    state->nativewindow.height = state->height;
    vc_dispmanx_update_submit_sync(
dispman_update );

    state->surface = eglCreateWindowSurface(
state->display, config, &(state->nativewindow),
NULL );
    assert(state->surface != EGL_NO_SURFACE);

    // connect the context to the surface
    result = eglMakeCurrent(state->display,
state->surface, state->surface, state-
>context);
    assert(EGL_FALSE != result);
}

void esInitContext ( CUBE_STATE_T *p_state )
{
   if ( p_state != NULL )
```

```
   {
      memset( p_state, 0, sizeof( CUBE_STATE_T)
);
   }
}

void esRegisterDrawFunc(CUBE_STATE_T *p_state,
void (*draw_func) (CUBE_STATE_T* ) )
{
   p_state->draw_func = draw_func;
}

void  esMainLoop (CUBE_STATE_T *esContext )
{
    struct timeval t1, t2;
    struct timezone tz;
    float deltatime;
    float totaltime = 0.0f;
    unsigned int frames = 0;

    gettimeofday ( &t1 , &tz );

    while(1)
    {
        gettimeofday(&t2, &tz);
        deltatime = (float)(t2.tv_sec -
t1.tv_sec + (t2.tv_usec - t1.tv_usec) * 1e-6);
```

```
        t1 = t2;

        if (esContext->draw_func != NULL)
            esContext->draw_func(esContext);

        eglSwapBuffers(esContext->display,
esContext->surface);

        totaltime += deltatime;
        frames++;
        if (totaltime >  2.0f)
        {
            printf("%4d frames rendered in
%1.4f seconds -> FPS=%3.4f\n", frames,
totaltime, frames/totaltime);
            totaltime -= 2.0f;
            frames = 0;
        }
    }
}

int main(int argc, char *argv[])
{
    UserData user_data;
    int width, height;

    image = esLoadTGA("jan.tga", &width,
```

```
&height);
    if (image == NULL) {
    fprintf(stderr, "No such image\n");
    exit(1);
    }
    fprintf(stderr, "Image is %d x %d\n",
width, height);

    bcm_host_init();
    esInitContext(p_state);

    init_ogl(p_state, width, height);

    p_state->user_data = &user_data;
    p_state->width = width;
    p_state->height = height;

    if(!Init(p_state))
    return 0;

    esRegisterDrawFunc(p_state, Draw);

    eglSwapBuffers(p_state->display, p_state-
>surface);
    esMainLoop(p_state);
}
```

The image used is of me .

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

The Raspberry Pi (RPi) has a very underpowered CPU but a very powerful GPU. Using the GPU is not well documented. This chapter looks at drawing, rendering images, etc, using the GPU.

Resources

- openGL-RPi-tutorial
- Decoding and Rendering Compressed Images with OpenMAX on Raspberry Pi: PiOmxTextures
- Raspberry Pi VideoCore APIs
- TGA File Format Summary
- VMCS-X OpenMAX IL Components is a link to OpenMAX IL components on the RPi
- TGA File Format Summary
-
-
-

Acknowledgements

In many ways this chapter is a walk-through of some of the examples in the RPi's `/opt/vc/src/hello_pi` directory. These are written by a variety of authors, too many to acknowledge individually. Thanks to all of you!

OpenMAX

We discussed OpenMAX in general in the chapter OpenMAX , applying it to sound on the RPi and other systems. OpenMAX is an *extremely* difficult API to work with - if you have any sense you will walk away right now. By dom: "I have written a fair bit of openmax client code and find it very hard. You have to get an awful lot right before you get anything useful out. Just lots of OMX_ErrorInvalidState, and OMX_ErrorBadParameter messages if you are lucky. Nothing happening at all if you are not..." dom is quite correct - I estimate that my productivity in dealing with this API is done from the proverbial 10 lines per day to less than 10 lines per week. And lots of that is staring at the screen in complete bewilderment as to why it isn't working, with no error messages or anything to tell me what is going on (or isn't).

Compilation

Just to add a wrinkle to OpenMAX's complexity, you have to be careful to get all the flags correct in compiling programs - or, guess what? - things don't work then either. I suspect it is things like word-alignment that are critical, but I've haven't checked OpenMAX in detail. Here are what seems to work:

```
INCLUDES = -DSTANDALONE -
D__STDC_CONSTANT_MACROS -D__STDC_LIMIT_MACROS -
DTARGET_POSIX -D_LINUX -fPIC \
    -DPIC -D_REENTRANT -D_LARGEFILE64_SOURCE -
D_FILE_OFFSET_BITS=64 -U_FORTIFY_SOURCE -Wall -
g \
    -DHAVE_LIBOPENMAX=2 -DOMX -DOMX_SKIP64BIT -
ftree-vectorize -pipe -DUSE_EXTERNAL_OMX -
DHAVE_LIBBCM_HOST \
    -DUSE_EXTERNAL_LIBBCM_HOST -DUSE_VCHIQ_ARM
-Wno-psabi -I/opt/vc/include/ -
I/opt/vc/include/interface/vcos/pthreads \
    -I/opt/vc/include/interface/vmcs_host/linux
-I./ -I/opt/vc/src/hello_pi/libs/ilclient -
I/opt/vc/src/hello_pi/libs/vgfont

LDFLAGS = -Wl,--whole-archive -lilclient -
L/opt/vc/lib/ -lopenmaxil -lbcm_host -lvcos -
lvchiq_arm \
    -lpthread -lrt -
L/opt/vc/src/hello_pi/libs/ilclient -
L/opt/vc/src/hello_pi/libs/vgfont \
    -Wl,--no-whole-archive -rdynamic
```

You are welcome to try reducing the number of flags - good luck!

State changes

Execution of an OpenMAX component is driven by a state-transition model. A component will start in the `Loaded` state, transition to the `Idle` state and then to the `Executing` state. At least, that is the idea. In practice, the state transitions will only occur for a component when conditions are *exactly* right for that component.

The hardest transition to get right is from `Loaded` to `Idle`. Any *enabled* component has ports which have to be set up and if that is not done exactly right then nothing will happen. Being *exactly right* means that all of the resources such as buffers have to be allocated and configured *correctly* , which is often not so easy. So most programmers seem to default to *disabling* all of a component's ports at the beginning in order that this transition will occur, and then try enabling them afterwards. [The dramatic over-emphasis in this section is the result of bitter experience, both by myself and - I am sure - all of the other OpenMAX programmers!]

The ilclient library

The RPI has a library `ilclient` intended to make things easier. This library is *not* directly portable - it relies on VideoCore VC threads. This could presumably be replaced. The primary assistance that it appears to give to me is:

- OpenMAX is inherently multi-threaded, relying heavily on callbacks called in various threads
- The `ilclient` library manages those callbacks for you in a generic manner
- Instead of having to do asynchronous programming, the library gives you a number of thread wait/synchronise calls, and you use these to "sit out" the asynchronous calls in the main thread
- The programming style is often: "make an asynchronous call and then wait for it to complete."

That sounds awful, essentially turning concurrent programming into sequential programming, but in fact it doesn't matter much: much of the execution flow is just waiting in one thread or another, and this just reduces the number of threads you have to

think of... It's also a protective style of programming: by call and wait at least you have some chance of pinning down where your program has silently ground to a halt.

For example, a port for a component may be in disabled state. You can't allocate buffers for it until a call has been made to enable it. Then buffers can be allocated. After that (assuming no other issues) the port should be able to transition to enabled state. So wait for that to occur, to ensure that it does. Typical code is

```
// enable output port of decoder
    OMX_SendCommand(decoder->imageDecoder-
>handle,
            OMX_CommandPortEnable,
            decoder->imageDecoder->outPort,
NULL);

    // and allocate a buffer
    int             ret =
OMX_AllocateBuffer(decoder->imageDecoder-
>handle,
                        &decoder-
>pOutputBufferHeader,
                        decoder->imageDecoder-
>
                        outPort,
                        NULL,
                        portdef.nBufferSize);
    printf("Output port buffer allocated\n");

    if (ret != OMX_ErrorNone) {
    perror("Eror allocating buffer");
    return OMXJPEG_ERROR_MEMORY;
    }

    // wait for enable to complete
```

```
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
                OMX_CommandPortEnable, 1,
                decoder->imageDecoder->outPort,
1, 0,
                TIMEOUT_MS);
    printf("Decoder output port enabled\n");
```

ilclient errors

Calls to the `ilclient` library will sometimes throw error messages such as this

```
assertion
failure:ilclient.c:747:ilclient_change_component_
 == OMX_ErrorNone

Program received signal SIGABRT, Aborted.
0xb6e41bfc in raise () from /lib/arm-linux-
gnueabihf/libc.so.6
```

This doesn't tell you where in your code the call was made, and gives a totally useless error message.

Okay (you may think), run it inside a debugger such as `gdb` and when it stops, ask for a backtrace:

```
(gdb) where
#0  0xb6e41bfc in raise () from /lib/arm-linux-
gnueabihf/libc.so.6
#1  0xb6e4597c in abort () from /lib/arm-linux-
gnueabihf/libc.so.6
#2  0xb6f825a4 in ?? () from
/opt/vc/lib/libvcos.so
#3  0xb6f825a4 in ?? () from
/opt/vc/lib/libvcos.so
Backtrace stopped: previous frame identical to
this frame (corrupt stack?)
(gdb)
```

Ho hum, no luck there! You need to run the program inside the debugger to figure out which `ilclient` call broke, stepping past each `ilclient` call until one of them breaks. In this case, it was a call to `ilclient_change_component_state` (about the fifth such call I had made). Then run the program again, this time stepping *into* the offending call. Then you can see the details of each call made: in my case I was trying to change state with insufficient resources set:

```
(gdb)
365
ilclient_change_component_state(decoder-
>imageRender->component,
(gdb) step
ilclient_change_component_state (comp=0x4b150,
state=OMX_StateExecuting)
    at ilclient.c:746
746       error = OMX_SendCommand(comp->comp,
OMX_CommandStateSet, state, NULL);
(gdb) next
747       vc_assert(error == OMX_ErrorNone);
(gdb) print error
$1 = OMX_ErrorInsufficientResources
```

That's very tedious. Welcome to OpenMAX programming.

Creating a component

The `ilclient` library has a convenience function to create a component, `ilclient_create_component`. This takes four parameters. The first is an `ILCLIENT_T` which an application gets by a call to `ilclient_init()`. The second is the address to store the component.

The third is a 'simple' form of the component's name. Now this isn't the name used in the OpenMAX specification. And it isn't the actual name of the component. For an image decoder object these are `image_decoder` and `OMX.broadcom.image_decode` respectively. The name used is formed by dropping `OMX.broadcom.` from the component's name, to give `image_decode`. The full list of Broadcom components is at VMCS-X OpenMAX IL Components and this lists the names used.

The fourth parameter is a bit-wise OR of several flags. A major one is `ILCLIENT_DISABLE_ALL_PORTS` which - as might be expected - disables all ports of the component, to make it easier to transition to idle state later. Two other significant flags are `ILCLIENT_ENABLE_INPUT_BUFFERS` and

`ILCLIENT_ENABLE_OUTPUT_BUFFERS`. Now these don't actually enable any buffers for the ports, create buffers, or apparently do anything really useful. However, if you don't enable the appropriate buffers here and later try to actually enable them using an `OMX_CommandPortEnable` command, then the library will throw a cryptic error message complaining about an illegal operation. I suppose that is trying to ensure that you only make valid calls for that component, but I wish the error was not so un-helpful.

A typical component creation will look like

```
ilclient_create_component(decoder->client,
            &decoder->imageDecoder->component,
            "image_decode",
            ILCLIENT_DISABLE_ALL_PORTS
            |
            ILCLIENT_ENABLE_INPUT_BUFFERS
            |
            ILCLIENT_ENABLE_OUTPUT_BUFFERS)
```

Decoding a JPEG image

OpenMAX on the RPi has a standard component `OMX.broadcom.image_decode` . The RPi documentation for it is here . It has two ports. The input port has (3) buffers to take a JPEG image, and one output port for the decoded image. The input buffers have a default size, and if the image is large you just cycle through them, filling and emptying each buffer in turn.

The program starts off fairly easily in `main` by reading the JPEG file into a byte-buffer of the right size. The call to `bcm_host_init` is required to initialise the Broadcom libraries. The JPEG image decoder is then created and asked to decode the image.

```
int
main(int argc, char *argv[])
{
    OPENMAX_JPEG_DECODER *pDecoder;
    char            *sourceImage;
    size_t           imageSize;
    int              s;
    if (argc < 2) {
    printf("Usage: %s <filename>\n", argv[0]);
    return -1;
    }
    FILE            *fp = fopen(argv[1], "rb");
    if (!fp) {
    printf("File %s not found.\n", argv[1]);
    }
    fseek(fp, 0L, SEEK_END);
    imageSize = ftell(fp);
    fseek(fp, 0L, SEEK_SET);
    sourceImage = malloc(imageSize);
    assert(sourceImage != NULL);
    s = fread(sourceImage, 1, imageSize, fp);
    assert(s == imageSize);
    fclose(fp);
    bcm_host_init();
    s = setupOpenMaxJpegDecoder(&pDecoder);
    assert(s == 0);
```

```
    s = decodeImage(pDecoder, sourceImage,
imageSize);
    assert(s == 0);
    cleanup(pDecoder);
    free(sourceImage);
    return 0;
}
```

The call to `setupOpenMaxJpegDecoder` builds some data structures and calls to `prepareImageDecoder` to initialise the decoder and `startupImageDecoder` to move it into executing state, so that it can then decode the image.

```
int
setupOpenMaxJpegDecoder(OPENMAX_JPEG_DECODER **
pDecoder)
{
    *pDecoder =
malloc(sizeof(OPENMAX_JPEG_DECODER));
    if (pDecoder[0] == NULL) {
    perror("malloc decoder");
    return OMXJPEG_ERROR_MEMORY;
    }
    memset(*pDecoder, 0,
sizeof(OPENMAX_JPEG_DECODER));

    if ((pDecoder[0]->client = ilclient_init())
== NULL) {
    perror("ilclient_init");
    return OMXJPEG_ERROR_ILCLIENT_INIT;
    }

    if (OMX_Init() != OMX_ErrorNone) {
    ilclient_destroy(pDecoder[0]->client);
    perror("OMX_Init");
    return OMXJPEG_ERROR_OMX_INIT;
    }
    // prepare the image decoder
    int             ret =
```

```
prepareImageDecoder(pDecoder[0]);
    if (ret != OMXJPEG_OK)
    return ret;

    ret = startupImageDecoder(pDecoder[0]);
    if (ret != OMXJPEG_OK)
    return ret;

    return OMXJPEG_OK;
}
```

The call to `prepareImageDecoder` creates the component and establishes its input and output port numbers (which should be 320 and 321 respectively by the OpenMAX specification). These two ports are disabled, but they are enabled to have buffers.

The function `startupImageDecoder` is a heavy-duty function. It has to establish the format that it will accept from the input file by

```
// set input image format
    OMX_IMAGE_PARAM_PORTFORMATTYPE
imagePortFormat;
    memset(&imagePortFormat, 0,
sizeof(OMX_IMAGE_PARAM_PORTFORMATTYPE));
    imagePortFormat.nSize =
sizeof(OMX_IMAGE_PARAM_PORTFORMATTYPE);
    imagePortFormat.nVersion.nVersion =
OMX_VERSION;
    imagePortFormat.nPortIndex = decoder-
>imageDecoder->inPort;
    imagePortFormat.eCompressionFormat =
OMX_IMAGE_CodingJPEG;
    OMX_SetParameter(decoder->imageDecoder-
>handle,
             OMX_IndexParamImagePortFormat,
&imagePortFormat);
```

Then it queries for the buffer requirements, building an `OMX_PARAM_PORTDEFINITIONTYPE portdef` structure and populating it with a get parameter call.

```
// get buffer requirements
    OMX_PARAM_PORTDEFINITIONTYPE portdef;
    portdef.nSize =
sizeof(OMX_PARAM_PORTDEFINITIONTYPE);
    portdef.nVersion.nVersion = OMX_VERSION;
    portdef.nPortIndex = decoder->imageDecoder-
>inPort;
    OMX_GetParameter(decoder->imageDecoder-
>handle,
             OMX_IndexParamPortDefinition,
&portdef);
```

Then we can make a call to enable the input port, allocate the input buffers and wait for the port to become enabled

```
// enable the port and setup the buffers
    OMX_SendCommand(decoder->imageDecoder-
>handle,
            OMX_CommandPortEnable,
            decoder->imageDecoder->inPort,
NULL);
    decoder->inputBufferHeaderCount =
portdef.nBufferCountActual;
    // allocate pointer array
    decoder->ppInputBufferHeader =
    (OMX_BUFFERHEADERTYPE **)
malloc(sizeof(void) *
                     decoder-
>inputBufferHeaderCount);
    // allocate each buffer
    int             i;
    for (i = 0; i < decoder-
>inputBufferHeaderCount; i++) {
    if (OMX_AllocateBuffer(decoder-
>imageDecoder->handle,
                   &decoder-
>ppInputBufferHeader[i],
                   decoder->imageDecoder-
>inPort,
                   (void *) i,
                   portdef.nBufferSize) !=
```

```
OMX_ErrorNone) {
        perror("Allocate decode buffer");
        return OMXJPEG_ERROR_MEMORY;
    }
    }
    // wait for port enable to complete - which
it should once buffers are
    // assigned
    int             ret =
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
                OMX_CommandPortEnable, 0,
                decoder->imageDecoder->inPort,
0,
                0, TIMEOUT_MS);
    if (ret != 0) {
    fprintf(stderr, "Did not get port enable
%d\n", ret);
    return OMXJPEG_ERROR_EXECUTING;
    }
```

Finally, we can move the component into executing state. But being paranoic about the behaviour of this API, we wait to ensure that it actually does make the state transition requested:

```
// start executing the decoder
    ret = OMX_SendCommand(decoder-
>imageDecoder-gt;handle,
              OMX_CommandStateSet,
OMX_StateExecuting, NULL);
    if (ret != 0) {
    fprintf(stderr, "Error starting image
decoder %x\n", ret);
    return OMXJPEG_ERROR_EXECUTING;
    }
    ret = ilclient_wait_for_event(decoder-
gt;imageDecoder-gt;component,
                  OMX_EventCmdComplete,
                  OMX_StateExecuting, 0, 0, 1,
0,
                  TIMEOUT_MS);
    if (ret != 0) {
    fprintf(stderr, "Did not receive executing
stat %d\n", ret);
    // return OMXJPEG_ERROR_EXECUTING;
    }
```

The function `decodeImage` starts up normally enough. It loads the input buffers in a circular fashion from the JPEG image loaded into the `sourceImage` array. Each buffer is emptied by

a call to `OMX_EmptyThisBuffer`. When the entire image has been loaded, the final buffer has the `OMX_BUFFERFLAG_EOS` flag set to indicate that the image is complete.

The snag with this component is that you don't know how large the decoded image will be until the decoder has done at least some work on it. So while the input buffers can be assigned statically up front, filled and emptied, we don't know what the size of the output buffer should be. Fortunately OpneMAX manages this by raising a `PortSettingsChanged` event when enough information is gained from the JPEG image to know the size of the decoded image.

If we were into concurrent programming, we would catch a `PortSettingsChanged` event in an event handling thread and work from there. The `ilclient` library tries to force a sequential mode of operation. So whenever a buffer is emptied, the application will go into a loop either waiting for a `PortSettingsChanged` event to occur and timing out after 5 milliseconds (I think) if it doesn't or exiting if the input buffer is empty. This code is a bit messy!

```
// wait for buffer to empty or port changed
event
    int             done = 0;
    while ((done == 0) || (decoder-
>pOutputBufferHeader == NULL)) {
        if (decoder->pOutputBufferHeader ==
NULL) {
        ret =
            ilclient_wait_for_event
            (decoder->imageDecoder->component,
             OMX_EventPortSettingsChanged,
             decoder->imageDecoder->outPort, 0,
0, 1, 0, 5);

        if (ret == 0) {
            portSettingsChanged(decoder);
        }
        } else {
        ret =
            ilclient_remove_event(decoder-
>imageDecoder->component,

OMX_EventPortSettingsChanged,
                      decoder->imageDecoder-
>outPort,
                      0, 0, 1);
```

```
        if (ret == 0)
            portSettingsChangedAgain(decoder);

        }

        // check to see if buffer is now empty
        if (pBufHeader->nFilledLen == 0)
        done = 1;

        if ((done == 0)
        || (decoder->pOutputBufferHeader ==
NULL)) {
        printf("Buffer is now size %d\n",
pBufHeader->nFilledLen);
        sleep(1);
        }
    }
```

If a `PortSettingsChanged` event occurs, the application calls `portSettingsChanged` in the main thread. This queries the (single) output port for the width and height values now set, switches the component to enabled state, allocates a buffer and waits for it to move to enabled state. We can also collect information about the format of the decoded image as well as other features such as size.

Once the output buffer has been created, we can make a call to fill the buffer. We should only have to do this once, so a flag `bFilled` is used to control this.

```
// fill the buffer if we have created the
buffer
    if (bFilled == 0) {
        if ((decoder->pOutputBufferHeader ==
NULL)) {
        portSettingsChanged(decoder);
        }
        OMX_PARAM_U32TYPE param;
        param.nSize =
sizeof(OMX_PARAM_U32TYPE);
        param.nVersion.nVersion = OMX_VERSION;
        param.nPortIndex = decoder-
>imageDecoder->outPort;

        OMX_GetParameter(decoder->imageDecoder-
>handle,
             OMX_IndexParamActiveStream,
&param);
        printf("Active stream %d\n",
param.nU32);

        printf("Trying to fill output
buffer\n");
        printState(decoder->imageDecoder-
>handle);
        ret = OMX_FillThisBuffer(decoder-
```

```
>imageDecoder->handle,
                      decoder-
>pOutputBufferHeader);

        if (ret != OMX_ErrorNone) {
        perror("Filling output buffer");
        fprintf(stderr, "Error code %x\n",
ret);
        return OMXJPEG_ERROR_MEMORY;
        }

        bFilled = 1;
    }
```

Once the input buffers have been filled and emptied the output buffer has been sized, allocated and a call made to fill it, the application has nothing to do but wait until the output buffer is filled. OpenMAX should generate a `OMX_BUFFERFLAG_EOS` when this happens, so we just wait

```
// wait for end of stream events
    int             ret =
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventBufferFlag,
                decoder->imageDecoder->outPort,
1,
                OMX_BUFFERFLAG_EOS, 1,
                0, 2);
    if (ret != 0) {
    fprintf(stderr, "No EOS event on image
decoder %d\n", ret);
    } else  {
    fprintf(stderr, "EOS event on image decoder
%d\n", ret);
    }
```

Ooops! the EOS event doesn't seem to get generated in practice (contrary to the specification) but the program seems to work anyway :-(.

At this point we can do something like save the decoded image to a file, or do further processing.

The final code is jpeg-decoder.c

```
/*
Copyright (c) 2012, Matt Ownby
                    Anthong Sale
Copyright (c) 2014: Jan Newmarch

All rights reserved.

Redistribution and use in source and binary
forms, with or without
modification, are permitted provided that the
following conditions are met:
    * Redistributions of source code must
retain the above copyright
      notice, this list of conditions and the
following disclaimer.
    * Redistributions in binary form must
reproduce the above copyright
      notice, this list of conditions and the
following disclaimer in the
      documentation and/or other materials
provided with the distribution.
    * Neither the name of the copyright holder
nor the
      names of its contributors may be used to
endorse or promote products
      derived from this software without
```

```
specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
*/

#include <stdio.h>
#include <assert.h>
#include "jpeg.h"
```

```
#define TIMEOUT_MS 2000

typedef struct _COMPONENT_DETAILS {
    COMPONENT_T    *component;
    OMX_HANDLETYPE  handle;
    int             inPort;
    int             outPort;
} COMPONENT_DETAILS;

struct _OPENMAX_JPEG_DECODER {
    ILCLIENT_T     *client;
    COMPONENT_DETAILS *imageDecoder;
#if IR
    COMPONENT_DETAILS *imageResizer;
#endif
    OMX_BUFFERHEADERTYPE **ppInputBufferHeader;
    int             inputBufferHeaderCount;
    OMX_BUFFERHEADERTYPE *pOutputBufferHeader;
};

int             bufferIndex = 0;    // index to
buffer array

void printState(OMX_HANDLETYPE handle) {
    OMX_STATETYPE state;
    OMX_ERRORTYPE err;
```

```
    err = OMX_GetState(handle, &state);
    if (err != OMX_ErrorNone) {
        fprintf(stderr, "Error on getting
state\n");
        exit(1);
    }
    switch (state) {
    case OMX_StateLoaded:
printf("StateLoaded\n"); break;
    case OMX_StateIdle: printf("StateIdle\n");
break;
    case OMX_StateExecuting:
printf("StateExecuting\n"); break;
    case OMX_StatePause:
printf("StatePause\n"); break;
    case OMX_StateWaitForResources:
printf("StateWiat\n"); break;
    default:  printf("State unknown\n"); break;
    }
}


int
portSettingsChanged(OPENMAX_JPEG_DECODER *
decoder)
{
```

```
    OMX_PARAM_PORTDEFINITIONTYPE portdef;

    // CLEANUP

    printf("Pport settings changed\n");
    // need to setup the input for the resizer
with the output of the
    // decoder
    portdef.nSize =
sizeof(OMX_PARAM_PORTDEFINITIONTYPE);
    portdef.nVersion.nVersion = OMX_VERSION;
    portdef.nPortIndex = decoder->imageDecoder-
>outPort;
    OMX_GetParameter(decoder->imageDecoder-
>handle,
             OMX_IndexParamPortDefinition,
&portdef);

    unsigned int    uWidth =
    (unsigned int)
portdef.format.image.nFrameWidth;
    unsigned int    uHeight =
    (unsigned int)
portdef.format.image.nFrameHeight;

    // enable ports
    OMX_SendCommand(decoder->imageDecoder-
```

```
>handle,
            OMX_CommandPortEnable,
            decoder->imageDecoder->outPort,
NULL);

    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
                OMX_CommandPortEnable, 1,
                decoder->imageDecoder->outPort,
1, 0,
                TIMEOUT_MS);
    printf("Decoder output port enabled\n");

    OMX_IMAGE_PARAM_PORTFORMATTYPE iportdef;

    printf("Port settings changed\n");

    // show some logging so user knows it's
working
    printf
    ("Width: %u Height: %u Output Color Format:
0x%x Buffer Size: %u\n",
     (unsigned int)
portdef.format.image.nFrameWidth,
     (unsigned int)
portdef.format.image.nFrameHeight,
```

```
     (unsigned int)
portdef.format.image.eColorFormat,
     (unsigned int) portdef.nBufferSize);
    fflush(stdout);


    // enable output port of decoder
    OMX_SendCommand(decoder->imageDecoder-
>handle,
            OMX_CommandPortEnable,
            decoder->imageDecoder->outPort,
NULL);


    int             ret =
OMX_AllocateBuffer(decoder->imageDecoder-
>handle,
                        &decoder-
>pOutputBufferHeader,
                        decoder->imageDecoder-
>
                        outPort,
                        NULL,
                        portdef.nBufferSize);
    printf("Output port buffer allocated\n");

    if (ret != OMX_ErrorNone) {
```

```
    perror("Eror allocating buffer");
    return OMXJPEG_ERROR_MEMORY;
    }


    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
                OMX_CommandPortEnable, 1,
                decoder->imageDecoder->outPort,
1, 0,
                TIMEOUT_MS);
    printf("Decoder output port rnabled\n");

    return OMXJPEG_OK;
}

int
portSettingsChangedAgain(OPENMAX_JPEG_DECODER *
decoder)
{
    printf("Port settings changed again\n");
    ilclient_disable_port(decoder-
>imageDecoder->component,
              decoder->imageDecoder->outPort);

    OMX_PARAM_PORTDEFINITIONTYPE portdef;
```

```
    // need to setup the input for the resizer
with the output of the
    // decoder
    portdef.nSize =
sizeof(OMX_PARAM_PORTDEFINITIONTYPE);
    portdef.nVersion.nVersion = OMX_VERSION;
    portdef.nPortIndex = decoder->imageDecoder-
>outPort;
    OMX_GetParameter(decoder->imageDecoder-
>handle,
             OMX_IndexParamPortDefinition,
&portdef);

    // enable output of decoder
    ilclient_enable_port(decoder->imageDecoder-
>component,
             decoder->imageDecoder->outPort);

    // need to wait for this event
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventPortSettingsChanged,
                decoder->imageDecoder->outPort,
1,
                0, 0, 0, TIMEOUT_MS);
    return OMXJPEG_OK;
```

```
}

int
prepareImageDecoder(OPENMAX_JPEG_DECODER *
decoder)
{
    decoder->imageDecoder =
malloc(sizeof(COMPONENT_DETAILS));
    if (decoder->imageDecoder == NULL) {
    perror("malloc image decoder");
    return OMXJPEG_ERROR_MEMORY;
    }

    int ret =
ilclient_create_component(decoder->client,
                      &decoder->imageDecoder-
>component,
                      "image_decode",
                      ILCLIENT_DISABLE_ALL_PORTS
                      |

ILCLIENT_ENABLE_INPUT_BUFFERS
                      |

ILCLIENT_ENABLE_OUTPUT_BUFFERS);

    if (ret != 0) {
```

```
    perror("image decode");
    return OMXJPEG_ERROR_CREATING_COMP;
    }
    // grab the handle for later use in OMX
calls directly
    decoder->imageDecoder->handle =
    ILC_GET_HANDLE(decoder->imageDecoder-
>component);

    // get and store the ports
    OMX_PORT_PARAM_TYPE port;
    port.nSize = sizeof(OMX_PORT_PARAM_TYPE);
    port.nVersion.nVersion = OMX_VERSION;

    OMX_GetParameter(decoder->imageDecoder-
>handle,
             OMX_IndexParamImageInit, &port);
    if (port.nPorts != 2) {
    return OMXJPEG_ERROR_WRONG_NO_PORTS;
    }
    decoder->imageDecoder->inPort =
port.nStartPortNumber;
    decoder->imageDecoder->outPort =
port.nStartPortNumber + 1;

   decoder->pOutputBufferHeader = NULL;
```

```
    return OMXJPEG_OK;
}

int
startupImageDecoder(OPENMAX_JPEG_DECODER *
decoder)
{
    // move to idle
    ilclient_change_component_state(decoder-
>imageDecoder->component,
                        OMX_StateIdle);

    // set input image format
    OMX_IMAGE_PARAM_PORTFORMATTYPE
imagePortFormat;
    memset(&imagePortFormat, 0,
sizeof(OMX_IMAGE_PARAM_PORTFORMATTYPE));
    imagePortFormat.nSize =
sizeof(OMX_IMAGE_PARAM_PORTFORMATTYPE);
    imagePortFormat.nVersion.nVersion =
OMX_VERSION;
    imagePortFormat.nPortIndex = decoder-
>imageDecoder->inPort;
    imagePortFormat.eCompressionFormat =
OMX_IMAGE_CodingJPEG;
    OMX_SetParameter(decoder->imageDecoder-
>handle,
```

```
                OMX_IndexParamImagePortFormat,
&imagePortFormat);

    // get buffer requirements
    OMX_PARAM_PORTDEFINITIONTYPE portdef;
    portdef.nSize =
sizeof(OMX_PARAM_PORTDEFINITIONTYPE);
    portdef.nVersion.nVersion = OMX_VERSION;
    portdef.nPortIndex = decoder->imageDecoder-
>inPort;
    OMX_GetParameter(decoder->imageDecoder-
>handle,
                OMX_IndexParamPortDefinition,
&portdef);

    // enable the port and setup the buffers
    OMX_SendCommand(decoder->imageDecoder-
>handle,
               OMX_CommandPortEnable,
               decoder->imageDecoder->inPort,
NULL);
    decoder->inputBufferHeaderCount =
portdef.nBufferCountActual;
    // allocate pointer array
    decoder->ppInputBufferHeader =
    (OMX_BUFFERHEADERTYPE **)
malloc(sizeof(void) *
```

```
                        decoder-
>inputBufferHeaderCount);
    // allocate each buffer
    int             i;
    for (i = 0; i < decoder-
>inputBufferHeaderCount; i++) {
    if (OMX_AllocateBuffer(decoder-
>imageDecoder->handle,
                   &decoder-
>ppInputBufferHeader[i],
                   decoder->imageDecoder-
>inPort,
                   (void *) i,
                   portdef.nBufferSize) !=
OMX_ErrorNone) {
        perror("Allocate decode buffer");
        return OMXJPEG_ERROR_MEMORY;
    }
    }
    // wait for port enable to complete - which
it should once buffers are
    // assigned
    int             ret =
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
                OMX_CommandPortEnable, 0,
```

```
                 decoder->imageDecoder->inPort,
0,
                 0, TIMEOUT_MS);
    if (ret != 0) {
    fprintf(stderr, "Did not get port enable
%d\n", ret);
    return OMXJPEG_ERROR_EXECUTING;
    }
    // start executing the decoder
    ret = OMX_SendCommand(decoder-
>imageDecoder->handle,
              OMX_CommandStateSet,
OMX_StateExecuting, NULL);
    if (ret != 0) {
    fprintf(stderr, "Error starting image
decoder %x\n", ret);
    return OMXJPEG_ERROR_EXECUTING;
    }
    ret = ilclient_wait_for_event(decoder-
>imageDecoder->component,
                  OMX_EventCmdComplete,
                  OMX_StateExecuting, 0, 0, 1,
0,
                  TIMEOUT_MS);
    if (ret != 0) {
    fprintf(stderr, "Did not receive executing
stat %d\n", ret);
```

```
    // return OMXJPEG_ERROR_EXECUTING;
    }

    return OMXJPEG_OK;
}

// this function run the boilerplate to setup
the openmax components;
int
setupOpenMaxJpegDecoder(OPENMAX_JPEG_DECODER **
pDecoder)
{
    *pDecoder =
malloc(sizeof(OPENMAX_JPEG_DECODER));
    if (pDecoder[0] == NULL) {
    perror("malloc decoder");
    return OMXJPEG_ERROR_MEMORY;
    }
    memset(*pDecoder, 0,
sizeof(OPENMAX_JPEG_DECODER));

    if ((pDecoder[0]->client = ilclient_init())
== NULL) {
    perror("ilclient_init");
    return OMXJPEG_ERROR_ILCLIENT_INIT;
    }
```

```
    if (OMX_Init() != OMX_ErrorNone) {
    ilclient_destroy(pDecoder[0]->client);
    perror("OMX_Init");
    return OMXJPEG_ERROR_OMX_INIT;
    }
    // prepare the image decoder
    int             ret =
prepareImageDecoder(pDecoder[0]);
    if (ret != OMXJPEG_OK)
    return ret;

    ret = startupImageDecoder(pDecoder[0]);
    if (ret != OMXJPEG_OK)
    return ret;

    return OMXJPEG_OK;
}

// this function passed the jpeg image buffer
in, and returns the decoded
// image
int
decodeImage(OPENMAX_JPEG_DECODER * decoder,
char *sourceImage,
        size_t imageSize)
{
    char           *sourceOffset = sourceImage;
```

```
// we store a seperate
                        // buffer ot image so
we
                        // can offset it
    size_t          toread = 0;    // bytes
left to read from buffer
    toread += imageSize;
    int             bFilled = 0;    // have we
filled our output
                    // buffer
    bufferIndex = 0;

    while (toread > 0) {
    // get next buffer from array
    OMX_BUFFERHEADERTYPE *pBufHeader =
        decoder-
>ppInputBufferHeader[bufferIndex];

    // step index and reset to 0 if required
    bufferIndex++;
    if (bufferIndex >= decoder-
>inputBufferHeaderCount)
        bufferIndex = 0;

    // work out the next chunk to load into the
decoder
    if (toread > pBufHeader->nAllocLen)
```

```
        pBufHeader->nFilledLen = pBufHeader-
>nAllocLen;
    else
        pBufHeader->nFilledLen = toread;

    toread = toread - pBufHeader->nFilledLen;

    // pass the bytes to the buffer
    memcpy(pBufHeader->pBuffer, sourceOffset,
pBufHeader->nFilledLen);
    printf("Read into buffer %d\n", pBufHeader-
>nFilledLen);

    // update the buffer pointer and set the
input flags

    sourceOffset = sourceOffset + pBufHeader-
>nFilledLen;
    pBufHeader->nOffset = 0;
    pBufHeader->nFlags = 0;
    if (toread <= 0) {
        pBufHeader->nFlags =
OMX_BUFFERFLAG_EOS;
    }
    // empty the current buffer
    printf("Emptying buffer\n");
    int             ret =
```

```
        OMX_EmptyThisBuffer(decoder-
>imageDecoder->handle,
                pBufHeader);

    if (ret != OMX_ErrorNone) {
        perror("Empty input buffer");
        fprintf(stderr, "return code %x\n",
ret);
        return OMXJPEG_ERROR_MEMORY;
    }
    // wait for buffer to empty or port changed
event
    int             done = 0;
    while ((done == 0) || (decoder-
>pOutputBufferHeader == NULL)) {
        if (decoder->pOutputBufferHeader ==
NULL) {
        ret =
            ilclient_wait_for_event
            (decoder->imageDecoder->component,
             OMX_EventPortSettingsChanged,
             decoder->imageDecoder->outPort, 0,
0, 1, 0, 5);

        if (ret == 0) {
            portSettingsChanged(decoder);
        }
```

```
        } else {
        ret =
            ilclient_remove_event(decoder-
>imageDecoder->component,

OMX_EventPortSettingsChanged,
                        decoder->imageDecoder-
>outPort,
                        0, 0, 1);
        if (ret == 0)
            portSettingsChangedAgain(decoder);

        }

        // check to see if buffer is now empty
        if (pBufHeader->nFilledLen == 0)
        done = 1;

        if ((done == 0)
        || (decoder->pOutputBufferHeader ==
NULL)) {
        printf("Buffer is now size %d\n",
pBufHeader->nFilledLen);
        sleep(1);
        }
    }
```

```
    // fill the buffer if we have created the
buffer
    if (bFilled == 0) {
        if ((decoder->pOutputBufferHeader ==
NULL)) {
        portSettingsChanged(decoder);
        }
        OMX_PARAM_U32TYPE param;
        param.nSize =
sizeof(OMX_PARAM_U32TYPE);
        param.nVersion.nVersion = OMX_VERSION;
        param.nPortIndex = decoder-
>imageDecoder->outPort;

        OMX_GetParameter(decoder->imageDecoder-
>handle,
             OMX_IndexParamActiveStream,
&param);
        printf("Active stream %d\n",
param.nU32);

        printf("Trying to fill output
buffer\n");
        printState(decoder->imageDecoder-
>handle);
        ret = OMX_FillThisBuffer(decoder-
>imageDecoder->handle,
```

```
                        decoder-
>pOutputBufferHeader);

        if (ret != OMX_ErrorNone) {
        perror("Filling output buffer");
        fprintf(stderr, "Error code %x\n",
ret);
        return OMXJPEG_ERROR_MEMORY;
        }

        bFilled = 1;
    }
    }

    // wait for buffer to fill
    /*
     * while(pBufHeader->nFilledLen == 0) {
sleep(5); }
     */

    // wait for end of stream events
    int             ret =
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventBufferFlag,
                decoder->imageDecoder->outPort,
1,
```

```
                    OMX_BUFFERFLAG_EOS, 1,
                    0, 2);
    if (ret != 0) {
    fprintf(stderr, "No EOS event on image
decoder %d\n", ret);
    } else  {
    fprintf(stderr, "EOS event on image decoder
%d\n", ret);
    }

    printf("Resized %d\n", decoder-
>pOutputBufferHeader->nFilledLen);
    FILE *fp = fopen("out", "w");
    int n;
    for (n = 0; n < decoder-
>pOutputBufferHeader->nFilledLen; n++) {
    fputc(decoder->pOutputBufferHeader-
>pBuffer[n], fp);
    }
    fclose(fp);
    printf("File written\n");

    return OMXJPEG_OK;
}

// this function cleans up the decoder.
void
```

```
cleanup(OPENMAX_JPEG_DECODER * decoder)
{
    // flush everything through
    OMX_SendCommand(decoder->imageDecoder-
>handle,
            OMX_CommandFlush, decoder-
>imageDecoder->outPort,
            NULL);
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
OMX_CommandFlush, 0,
                decoder->imageDecoder->outPort,
0, 0,
                TIMEOUT_MS);

    OMX_SendCommand(decoder->imageDecoder-
>handle, OMX_CommandPortDisable,
            decoder->imageDecoder->inPort,
NULL);

    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
OMX_CommandPortDisable,
                0, decoder->imageDecoder-
>outPort, 0, 0,
```

```
                 TIMEOUT_MS);

    int             i = 0;
    for (i = 0; i < decoder-
>inputBufferHeaderCount; i++) {
    OMX_BUFFERHEADERTYPE *vpBufHeader =
        decoder->ppInputBufferHeader[i];

    OMX_FreeBuffer(decoder->imageDecoder-
>handle,
               decoder->imageDecoder->inPort,
vpBufHeader);
    }

    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
OMX_CommandPortDisable,
                0, decoder->imageDecoder-
>inPort, 0, 0,
                TIMEOUT_MS);

    OMX_FreeBuffer(decoder->imageDecoder-
>handle,
           decoder->imageDecoder->outPort,
           decoder->pOutputBufferHeader);
```

```
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
OMX_CommandPortDisable,
                0, decoder->imageDecoder-
>outPort, 0, 0,
                TIMEOUT_MS);

    ilclient_change_component_state(decoder-
>imageDecoder->component,
                    OMX_StateIdle);


    ilclient_change_component_state(decoder-
>imageDecoder->component,
                    OMX_StateLoaded);


    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
OMX_CommandStateSet, 0,
                OMX_StateLoaded, 0, 0,
TIMEOUT_MS);

    OMX_Deinit();
```

```
    if (decoder->client != NULL) {
    ilclient_destroy(decoder->client);
    }
}

int
main(int argc, char *argv[])
{
    OPENMAX_JPEG_DECODER *pDecoder;
    char            *sourceImage;
    size_t           imageSize;
    int              s;
    if (argc < 2) {
    printf("Usage: %s <filename>\n", argv[0]);
    return -1;
    }
    FILE            *fp = fopen(argv[1], "rb");
    if (!fp) {
    printf("File %s not found.\n", argv[1]);
    }
    fseek(fp, 0L, SEEK_END);
    imageSize = ftell(fp);
    fseek(fp, 0L, SEEK_SET);
    sourceImage = malloc(imageSize);
    assert(sourceImage != NULL);
    s = fread(sourceImage, 1, imageSize, fp);
    assert(s == imageSize);
```

```
    fclose(fp);
    bcm_host_init();
    s = setupOpenMaxJpegDecoder(&pDecoder);
    assert(s == 0);
    s = decodeImage(pDecoder, sourceImage,
imageSize);
    assert(s == 0);
    cleanup(pDecoder);
    free(sourceImage);
    printf("Success\n");
    return 0;
}
```

Format of decoded image

There are two principal colour formats used for images: YUV and RGB. It is simple to translate from one to the other. But the Broadcom `image_decode` component won't do any translation: it will only give you YUV images. This is according to the OpenMAX specifications for a JPEG decoder. If you want RGB, you have to do it yourself or use another Broadcom component, the `resize` component. This is not a standard OpenMAX component (see below).

Tunnelling

If you want to connect two or more OpenMAX components then these two components have to be able to pass information between them. This means that the port formats must be the same between one component's output port and the next component's input port, and there must be a means of transferring from output buffers to input buffers.

OpenMAX supplies two methods to do this:

- Do it all yourself; or
- Use tunnelling

Doing it yourself is always supported. In this case the client has to work out compatable port formats, and co-ordinate buffers. For example, when one component fills a buffer the client could copy it to the next component's input buffer. This is inefficient. Or the client could use the first component's output buffer as the next component's input buffer. But that would mean that the shared buffer would not be free for the first component to refill until the second component had received a buffer-empty event. Messy.

The alternative is tunnelling where the client asks the components to do all this for them. The OpenMAX components support tunnelling, which makes life a lot easier.

Tunnelling is used to connect two ports. It can only be done when the two components are in the Loaded state or when the ports are disabled. The call `OMX_SetupTunnel` will then connect the two ports as in

```
// establish tunnel between decoder output and
resizer input
    OMX_SetupTunnel(decoder->imageDecoder-
>handle,
            decoder->imageDecoder->outPort,
            decoder->imageResizer->handle,
            decoder->imageResizer->inPort);
```

A typical use of this is to disable the relevant ports while they are in Idle state, and then to fill and empty the input buffers of the first component. This will trigger a `PortSettingsChanged` event on the first component's output port. At this point, with formats and things like image sizes determined, the two ports can be connected by a tunnel. There is no need to allocate buffers: the tunnel does that.

Decoding a JPEG image into RGB format

The OpenMAX component `image_resize` will decode a JPEG image, but only to YUV format. Broadcom has an additional non-standard component `resize` which can not only resize an uncompresssed image but also convert it to RGB format.

Matt Ownby and Anthong Sale did a marvellous job of unravelling the sequence of steps required to set up tunnelling between a decoder and resizer. Their logic is captured in the `portSettingsChanged` function. This adds tunnelling to the decoder program given earlier.

```
int
portSettingsChanged(OPENMAX_JPEG_DECODER *
decoder)
{
    OMX_PARAM_PORTDEFINITIONTYPE portdef;

    // need to setup the input for the resizer
with the output of the
    // decoder
    portdef.nSize =
sizeof(OMX_PARAM_PORTDEFINITIONTYPE);
    portdef.nVersion.nVersion = OMX_VERSION;
    portdef.nPortIndex = decoder->imageDecoder-
>outPort;
    OMX_GetParameter(decoder->imageDecoder-
>handle,
             OMX_IndexParamPortDefinition,
&portdef);

    unsigned int    uWidth =
    (unsigned int)
portdef.format.image.nFrameWidth;
    unsigned int    uHeight =
    (unsigned int)
portdef.format.image.nFrameHeight;
```

```
    // tell resizer input what the decoder
output will be providing
    portdef.nPortIndex = decoder->imageResizer-
>inPort;
    OMX_SetParameter(decoder->imageResizer-
>handle,
             OMX_IndexParamPortDefinition,
&portdef);

    // establish tunnel between decoder output
and resizer input
    OMX_SetupTunnel(decoder->imageDecoder-
>handle,
            decoder->imageDecoder->outPort,
            decoder->imageResizer->handle,
            decoder->imageResizer->inPort);

    // enable ports
    OMX_SendCommand(decoder->imageDecoder-
>handle,
            OMX_CommandPortEnable,
            decoder->imageDecoder->outPort,
NULL);
    OMX_SendCommand(decoder->imageResizer-
>handle,
            OMX_CommandPortEnable,
            decoder->imageResizer->inPort,
```

```
NULL);

    // put resizer in idle state (this allows
the outport of the decoder
    // to become enabled)
    OMX_SendCommand(decoder->imageResizer-
>handle,
            OMX_CommandStateSet, OMX_StateIdle,
NULL);

    // wait for state change complete
    ilclient_wait_for_event(decoder-
>imageResizer->component,
                OMX_EventCmdComplete,
                OMX_CommandStateSet, 1,
                OMX_StateIdle, 1, 0,
TIMEOUT_MS);

    // once the state changes, both ports
should become enabled and the
    // resizer
    // output should generate a settings
changed event
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
                OMX_CommandPortEnable, 1,
```

```
                decoder->imageDecoder->outPort,
1, 0,
                TIMEOUT_MS);
    ilclient_wait_for_event(decoder-
>imageResizer->component,
                OMX_EventCmdComplete,
OMX_CommandPortEnable, 1,
                decoder->imageResizer->inPort,
1, 0,
                TIMEOUT_MS);
    ilclient_wait_for_event(decoder-
>imageResizer->component,
                OMX_EventPortSettingsChanged,
                decoder->imageResizer->outPort,
1, 0, 1, 0,
                TIMEOUT_MS);

    ilclient_disable_port(decoder-
>imageResizer->component,
              decoder->imageResizer->outPort);

    // query output buffer requirements for
resizer
    portdef.nSize =
sizeof(OMX_PARAM_PORTDEFINITIONTYPE);
    portdef.nVersion.nVersion = OMX_VERSION;
    portdef.nPortIndex = decoder->imageResizer-
```

```
>outPort;
    OMX_GetParameter(decoder->imageResizer-
>handle,
             OMX_IndexParamPortDefinition,
&portdef);

    // change output color format and
dimensions to match input
    portdef.format.image.eCompressionFormat =
OMX_IMAGE_CodingUnused;
    portdef.format.image.eColorFormat =
OMX_COLOR_Format32bitABGR8888;
    portdef.format.image.nFrameWidth = uWidth;
    portdef.format.image.nFrameHeight =
uHeight;
    portdef.format.image.nStride = 0;
    portdef.format.image.nSliceHeight = 0;
    portdef.format.image.bFlagErrorConcealment
= OMX_FALSE;

    OMX_SetParameter(decoder->imageResizer-
>handle,
             OMX_IndexParamPortDefinition,
&portdef);

    // grab output requirements again to get
actual buffer size
```

```
    // requirement (and buffer count
requirement!)
    OMX_GetParameter(decoder->imageResizer-
>handle,
             OMX_IndexParamPortDefinition,
&portdef);

    // move resizer into executing state
    ilclient_change_component_state(decoder-
>imageResizer->component,
                     OMX_StateExecuting);
    // show some logging so user knows it's
working
    printf
    ("Width: %u Height: %u Output Color Format:
0x%x Buffer Size: %u\n",
     (unsigned int)
portdef.format.image.nFrameWidth,
     (unsigned int)
portdef.format.image.nFrameHeight,
     (unsigned int)
portdef.format.image.eColorFormat,
     (unsigned int) portdef.nBufferSize);
    fflush(stdout);

    // enable output port of resizer
    OMX_SendCommand(decoder->imageResizer-
```

```
>handle,
            OMX_CommandPortEnable,
            decoder->imageResizer->outPort,
NULL);

    int             ret =
OMX_AllocateBuffer(decoder->imageResizer-
>handle,
                        &decoder-
>pOutputBufferHeader,
                        decoder->imageResizer-
>
                        outPort,
                        NULL,
                        portdef.nBufferSize);
    if (ret != OMX_ErrorNone) {
    perror("Eror allocating buffer");
    return OMXJPEG_ERROR_MEMORY;
    }

    ilclient_wait_for_event(decoder-
>imageResizer->component,
                OMX_EventCmdComplete,
                OMX_CommandPortEnable, 1,
                decoder->imageResizer->outPort,
1, 0,
                TIMEOUT_MS);
```

```
    return OMXJPEG_OK;
}
```

The only change we make to their code is to add a function to dump the decoded and converted RGB image as a TGA file. The function `save_image_as_TGA` is called when the output buffer of the resizer is non-empty. Just a few wrinkles in this. The image has a height and width. The resizer will have aligned those up to a 16-byte multiple, the Stride and the SliceHeight. We have asked for a 32-bit format (`OMX_COLOR_Format32bitABGR8888`) and so we have to specify this. The TGA format is specified here .

The co-ordinate system between TGA and the Broadcom is inverted. We are lazy here because it simplifies the code, and save the image upside down. The source is almost completely from the RPi `/opt/vc/src/hello_pi/hello_jpeg/jpeg.c` program, here as decodejpeg2rgb.c :

```
/*
Copyright (c) 2012, Matt Ownby
                    Anthong Sale
Copyright (c) 2014, Jan Newmarch

All rights reserved.

Redistribution and use in source and binary
forms, with or without
modification, are permitted provided that the
following conditions are met:
    * Redistributions of source code must
retain the above copyright
      notice, this list of conditions and the
following disclaimer.
    * Redistributions in binary form must
reproduce the above copyright
      notice, this list of conditions and the
following disclaimer in the
      documentation and/or other materials
provided with the distribution.
    * Neither the name of the copyright holder
nor the
      names of its contributors may be used to
endorse or promote products
      derived from this software without
```

```
specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
*/

#include <stdio.h>
#include <assert.h>
#include "jpeg.h"
```

```
#define TIMEOUT_MS 2000

typedef struct _COMPONENT_DETAILS {
    COMPONENT_T    *component;
    OMX_HANDLETYPE  handle;
    int             inPort;
    int             outPort;
} COMPONENT_DETAILS;

struct _OPENMAX_JPEG_DECODER {
    ILCLIENT_T     *client;
    COMPONENT_DETAILS *imageDecoder;
    COMPONENT_DETAILS *imageResizer;
    OMX_BUFFERHEADERTYPE **ppInputBufferHeader;
    int             inputBufferHeaderCount;
    OMX_BUFFERHEADERTYPE *pOutputBufferHeader;
};

int             bufferIndex = 0;    // index to
buffer array

int
portSettingsChanged(OPENMAX_JPEG_DECODER *
decoder)
{
    OMX_PARAM_PORTDEFINITIONTYPE portdef;
```

```
    // need to setup the input for the resizer
with the output of the
    // decoder
    portdef.nSize =
sizeof(OMX_PARAM_PORTDEFINITIONTYPE);
    portdef.nVersion.nVersion = OMX_VERSION;
    portdef.nPortIndex = decoder->imageDecoder-
>outPort;
    OMX_GetParameter(decoder->imageDecoder-
>handle,
             OMX_IndexParamPortDefinition,
&portdef);

    unsigned int    uWidth =
    (unsigned int)
portdef.format.image.nFrameWidth;
    unsigned int    uHeight =
    (unsigned int)
portdef.format.image.nFrameHeight;

    // tell resizer input what the decoder
output will be providing
    portdef.nPortIndex = decoder->imageResizer-
>inPort;
    OMX_SetParameter(decoder->imageResizer-
>handle,
```

```
              OMX_IndexParamPortDefinition,
&portdef);

    // establish tunnel between decoder output
and resizer input
    OMX_SetupTunnel(decoder->imageDecoder-
>handle,
            decoder->imageDecoder->outPort,
            decoder->imageResizer->handle,
            decoder->imageResizer->inPort);

    // enable ports
    OMX_SendCommand(decoder->imageDecoder-
>handle,
            OMX_CommandPortEnable,
            decoder->imageDecoder->outPort,
NULL);
    OMX_SendCommand(decoder->imageResizer-
>handle,
            OMX_CommandPortEnable,
            decoder->imageResizer->inPort,
NULL);

    // put resizer in idle state (this allows
the outport of the decoder
    // to become enabled)
    OMX_SendCommand(decoder->imageResizer-
```

```
>handle,
            OMX_CommandStateSet, OMX_StateIdle,
NULL);

    // wait for state change complete
    ilclient_wait_for_event(decoder-
>imageResizer->component,
                OMX_EventCmdComplete,
                OMX_CommandStateSet, 1,
                OMX_StateIdle, 1, 0,
TIMEOUT_MS);

    // once the state changes, both ports
should become enabled and the
    // resizer
    // output should generate a settings
changed event
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
                OMX_CommandPortEnable, 1,
                decoder->imageDecoder->outPort,
1, 0,
                TIMEOUT_MS);
    ilclient_wait_for_event(decoder-
>imageResizer->component,
                OMX_EventCmdComplete,
```

```
OMX_CommandPortEnable, 1,
                decoder->imageResizer->inPort,
1, 0,
                TIMEOUT_MS);
    ilclient_wait_for_event(decoder-
>imageResizer->component,
                OMX_EventPortSettingsChanged,
                decoder->imageResizer->outPort,
1, 0, 1, 0,
                TIMEOUT_MS);

    ilclient_disable_port(decoder-
>imageResizer->component,
              decoder->imageResizer->outPort);

    // query output buffer requirements for
resizer
    portdef.nSize =
sizeof(OMX_PARAM_PORTDEFINITIONTYPE);
    portdef.nVersion.nVersion = OMX_VERSION;
    portdef.nPortIndex = decoder->imageResizer-
>outPort;
    OMX_GetParameter(decoder->imageResizer-
>handle,
             OMX_IndexParamPortDefinition,
&portdef);
```

```
    // change output color format and
dimensions to match input
    portdef.format.image.eCompressionFormat =
OMX_IMAGE_CodingUnused;
    portdef.format.image.eColorFormat =
OMX_COLOR_Format32bitABGR8888;
    portdef.format.image.nFrameWidth = uWidth;
    portdef.format.image.nFrameHeight =
uHeight;
    portdef.format.image.nStride = 0;
    portdef.format.image.nSliceHeight = 0;
    portdef.format.image.bFlagErrorConcealment
= OMX_FALSE;

    OMX_SetParameter(decoder->imageResizer-
>handle,
             OMX_IndexParamPortDefinition,
&portdef);

    // grab output requirements again to get
actual buffer size
    // requirement (and buffer count
requirement!)
    OMX_GetParameter(decoder->imageResizer-
>handle,
             OMX_IndexParamPortDefinition,
&portdef);
```

```
    // move resizer into executing state
    ilclient_change_component_state(decoder-
>imageResizer->component,
                        OMX_StateExecuting);

    // show some logging so user knows it's
working
    printf
    ("Width: %u Height: %u Output Color Format:
0x%x Buffer Size: %u\n",
     (unsigned int)
portdef.format.image.nFrameWidth,
     (unsigned int)
portdef.format.image.nFrameHeight,
     (unsigned int)
portdef.format.image.eColorFormat,
     (unsigned int) portdef.nBufferSize);
    fflush(stdout);

    // enable output port of resizer
    OMX_SendCommand(decoder->imageResizer-
>handle,
            OMX_CommandPortEnable,
            decoder->imageResizer->outPort,
NULL);
```

```
    int             ret =
OMX_AllocateBuffer(decoder->imageResizer-
>handle,
                        &decoder-
>pOutputBufferHeader,
                        decoder->imageResizer-
>
                        outPort,
                        NULL,
                        portdef.nBufferSize);
    if (ret != OMX_ErrorNone) {
    perror("Eror allocating buffer");
    return OMXJPEG_ERROR_MEMORY;
    }

    ilclient_wait_for_event(decoder-
>imageResizer->component,
                OMX_EventCmdComplete,
                OMX_CommandPortEnable, 1,
                decoder->imageResizer->outPort,
1, 0,
                TIMEOUT_MS);

    return OMXJPEG_OK;
}

int
```

```
portSettingsChangedAgain(OPENMAX_JPEG_DECODER *
decoder)
{
    ilclient_disable_port(decoder-
>imageDecoder->component,
                decoder->imageDecoder->outPort);
    ilclient_disable_port(decoder-
>imageResizer->component,
                decoder->imageResizer->inPort);

    OMX_PARAM_PORTDEFINITIONTYPE portdef;

    // need to setup the input for the resizer
with the output of the
    // decoder
    portdef.nSize =
sizeof(OMX_PARAM_PORTDEFINITIONTYPE);
    portdef.nVersion.nVersion = OMX_VERSION;
    portdef.nPortIndex = decoder->imageDecoder-
>outPort;
    OMX_GetParameter(decoder->imageDecoder-
>handle,
                OMX_IndexParamPortDefinition,
&portdef);

    // tell resizer input what the decoder
output will be providing
```

```
    portdef.nPortIndex = decoder->imageResizer-
>inPort;
    OMX_SetParameter(decoder->imageResizer-
>handle,
             OMX_IndexParamPortDefinition,
&portdef);

    // enable output of decoder and input of
resizer (ie enable tunnel)
    ilclient_enable_port(decoder->imageDecoder-
>component,
             decoder->imageDecoder->outPort);
    ilclient_enable_port(decoder->imageResizer-
>component,
             decoder->imageResizer->inPort);

    // need to wait for this event
    ilclient_wait_for_event(decoder-
>imageResizer->component,
                OMX_EventPortSettingsChanged,
                decoder->imageResizer->outPort,
1,
                0, 0, 0, TIMEOUT_MS);

    return OMXJPEG_OK;
}
```

```
int
prepareResizer(OPENMAX_JPEG_DECODER * decoder)
{
    decoder->imageResizer =
malloc(sizeof(COMPONENT_DETAILS));
    if (decoder->imageResizer == NULL) {
    perror("malloc image resizer");
    return OMXJPEG_ERROR_MEMORY;
    }

    int             ret =
ilclient_create_component(decoder->client,
                              &decoder->
                              imageResizer->
                              component,
                              "resize",

ILCLIENT_DISABLE_ALL_PORTS
                              |

ILCLIENT_ENABLE_INPUT_BUFFERS
                              |

ILCLIENT_ENABLE_OUTPUT_BUFFERS);
    if (ret != 0) {
    perror("image resizer");
    return OMXJPEG_ERROR_CREATING_COMP;
```

```
    }
    // grab the handle for later use
    decoder->imageResizer->handle =
    ILC_GET_HANDLE(decoder->imageResizer-
>component);

    // get and store the ports
    OMX_PORT_PARAM_TYPE port;
    port.nSize = sizeof(OMX_PORT_PARAM_TYPE);
    port.nVersion.nVersion = OMX_VERSION;

    OMX_GetParameter(ILC_GET_HANDLE(decoder-
>imageResizer->component),
             OMX_IndexParamImageInit, &port);
    if (port.nPorts != 2) {
    return OMXJPEG_ERROR_WRONG_NO_PORTS;
    }
    decoder->imageResizer->inPort =
port.nStartPortNumber;
    decoder->imageResizer->outPort =
port.nStartPortNumber + 1;

    decoder->pOutputBufferHeader = NULL;

    return OMXJPEG_OK;
}
```

```
int
prepareImageDecoder(OPENMAX_JPEG_DECODER *
decoder)
{
    decoder->imageDecoder =
malloc(sizeof(COMPONENT_DETAILS));
    if (decoder->imageDecoder == NULL) {
    perror("malloc image decoder");
    return OMXJPEG_ERROR_MEMORY;
    }

    int             ret =
ilclient_create_component(decoder->client,
                              &decoder->
                              imageDecoder->
                              component,
                              "image_decode",

ILCLIENT_DISABLE_ALL_PORTS
                              |

ILCLIENT_ENABLE_INPUT_BUFFERS);

    if (ret != 0) {
    perror("image decode");
    return OMXJPEG_ERROR_CREATING_COMP;
    }
```

```
    // grab the handle for later use in OMX
calls directly
    decoder->imageDecoder->handle =
    ILC_GET_HANDLE(decoder->imageDecoder-
>component);

    // get and store the ports
    OMX_PORT_PARAM_TYPE port;
    port.nSize = sizeof(OMX_PORT_PARAM_TYPE);
    port.nVersion.nVersion = OMX_VERSION;

    OMX_GetParameter(decoder->imageDecoder-
>handle,
             OMX_IndexParamImageInit, &port);
    if (port.nPorts != 2) {
    return OMXJPEG_ERROR_WRONG_NO_PORTS;
    }
    decoder->imageDecoder->inPort =
port.nStartPortNumber;
    decoder->imageDecoder->outPort =
port.nStartPortNumber + 1;

    return OMXJPEG_OK;
}

int
startupImageDecoder(OPENMAX_JPEG_DECODER *
```

```
decoder)
{
    // move to idle
    ilclient_change_component_state(decoder-
>imageDecoder->component,
                        OMX_StateIdle);

    // set input image format
    OMX_IMAGE_PARAM_PORTFORMATTYPE
imagePortFormat;
    memset(&imagePortFormat, 0,
sizeof(OMX_IMAGE_PARAM_PORTFORMATTYPE));
    imagePortFormat.nSize =
sizeof(OMX_IMAGE_PARAM_PORTFORMATTYPE);
    imagePortFormat.nVersion.nVersion =
OMX_VERSION;
    imagePortFormat.nPortIndex = decoder-
>imageDecoder->inPort;
    imagePortFormat.eCompressionFormat =
OMX_IMAGE_CodingJPEG;
    OMX_SetParameter(decoder->imageDecoder-
>handle,
                OMX_IndexParamImagePortFormat,
&imagePortFormat);

    // get buffer requirements
    OMX_PARAM_PORTDEFINITIONTYPE portdef;
```

```
    portdef.nSize =
sizeof(OMX_PARAM_PORTDEFINITIONTYPE);
    portdef.nVersion.nVersion = OMX_VERSION;
    portdef.nPortIndex = decoder->imageDecoder-
>inPort;
    OMX_GetParameter(decoder->imageDecoder-
>handle,
             OMX_IndexParamPortDefinition,
&portdef);

    // enable the port and setup the buffers
    OMX_SendCommand(decoder->imageDecoder-
>handle,
            OMX_CommandPortEnable,
            decoder->imageDecoder->inPort,
NULL);
    decoder->inputBufferHeaderCount =
portdef.nBufferCountActual;
    // allocate pointer array
    decoder->ppInputBufferHeader =
    (OMX_BUFFERHEADERTYPE **)
malloc(sizeof(void) *
                     decoder-
>inputBufferHeaderCount);
    // allocate each buffer
    int             i;
    for (i = 0; i < decoder-
```

```
>inputBufferHeaderCount; i++) {
    if (OMX_AllocateBuffer(decoder-
>imageDecoder->handle,
                   &decoder-
>ppInputBufferHeader[i],
                   decoder->imageDecoder-
>inPort,
                   (void *) i,
                   portdef.nBufferSize) !=
OMX_ErrorNone) {
        perror("Allocate decode buffer");
        return OMXJPEG_ERROR_MEMORY;
    }
    }
    // wait for port enable to complete - which
it should once buffers are
    // assigned
    int             ret =
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
                OMX_CommandPortEnable, 0,
                decoder->imageDecoder->inPort,
0,
                0, TIMEOUT_MS);
    if (ret != 0) {
    fprintf(stderr, "Did not get port enable
```

```
%d\n", ret);
    return OMXJPEG_ERROR_EXECUTING;
    }
    // start executing the decoder
    ret = OMX_SendCommand(decoder-
>imageDecoder->handle,
              OMX_CommandStateSet,
OMX_StateExecuting, NULL);
    if (ret != 0) {
    fprintf(stderr, "Error starting image
decoder %x\n", ret);
    return OMXJPEG_ERROR_EXECUTING;
    }
    ret = ilclient_wait_for_event(decoder-
>imageDecoder->component,
                  OMX_EventCmdComplete,
                  OMX_StateExecuting, 0, 0, 1,
0,
                  TIMEOUT_MS);
    if (ret != 0) {
    fprintf(stderr, "Did not receive executing
stat %d\n", ret);
    // return OMXJPEG_ERROR_EXECUTING;
    }

    return OMXJPEG_OK;
}
```

```
// this function run the boilerplate to setup
the openmax components;
int
setupOpenMaxJpegDecoder(OPENMAX_JPEG_DECODER **
pDecoder)
{
    *pDecoder =
malloc(sizeof(OPENMAX_JPEG_DECODER));
    if (pDecoder[0] == NULL) {
    perror("malloc decoder");
    return OMXJPEG_ERROR_MEMORY;
    }
    memset(*pDecoder, 0,
sizeof(OPENMAX_JPEG_DECODER));

    if ((pDecoder[0]->client = ilclient_init())
== NULL) {
    perror("ilclient_init");
    return OMXJPEG_ERROR_ILCLIENT_INIT;
    }

    if (OMX_Init() != OMX_ErrorNone) {
    ilclient_destroy(pDecoder[0]->client);
    perror("OMX_Init");
    return OMXJPEG_ERROR_OMX_INIT;
    }
```

```
    // prepare the image decoder
    int             ret =
prepareImageDecoder(pDecoder[0]);
    if (ret != OMXJPEG_OK)
    return ret;

    ret = prepareResizer(pDecoder[0]);
    if (ret != OMXJPEG_OK)
    return ret;

    ret = startupImageDecoder(pDecoder[0]);
    if (ret != OMXJPEG_OK)
    return ret;

    return OMXJPEG_OK;
}

void save_image_as_TGA(OPENMAX_JPEG_DECODER *
decoder) {
    OMX_PARAM_PORTDEFINITIONTYPE portdef;
    unsigned int uWidth, uHeight, nSize,
nStride, nSliceHeight;
#define TGA_HEADER_SIZE 18
    char header[TGA_HEADER_SIZE];

    // query output buffer requirements for
resizer
```

```
    portdef.nSize =
sizeof(OMX_PARAM_PORTDEFINITIONTYPE);
    portdef.nVersion.nVersion = OMX_VERSION;
    portdef.nPortIndex = decoder->imageResizer-
>outPort;
    OMX_GetParameter(decoder->imageResizer-
>handle,
             OMX_IndexParamPortDefinition,
&portdef);

    uWidth =
portdef.format.image.nFrameWidth;
    uHeight =
portdef.format.image.nFrameHeight;
    nSliceHeight =
portdef.format.image.nSliceHeight;
    nStride = portdef.format.image.nStride;
    nSize = portdef.nBufferSize;
    //nSize = portdef.nFilledLen;
    printf("Width is %d stride %d height is %d
slice height %d, size %d\n",
       uWidth, nStride, uHeight, nSliceHeight,
nSize);
    // image in decoder->pOutputBufferHeader


    memset(header, 0, TGA_HEADER_SIZE);
```

```
    // header[0] = nSize;
    header[0x2] = 2;
    header[0xD] = (nStride/4) >> 8;
    header[0xC] = (nStride/4) & 0xFF;
    //header[0xF] = uHeight >> 8;
    //header[0xE] = uHeight & 0xFF;
    header[0xF] = nSliceHeight >> 8;
    header[0xE] = nSliceHeight & 0xFF;
    header[0x10] = 32;

    FILE *fout = fopen("out.tga", "w");
    fwrite(header, sizeof(char),
TGA_HEADER_SIZE, fout);

    // This writes the image upside down but is
simple
    // Should do it by rows, from end to
beginning
    int nwrote;
    nwrote = fwrite(decoder-
>pOutputBufferHeader->pBuffer, sizeof(char),
       nSize, fout);
    printf("Written %d\n", nwrote);
    fclose(fout);
}

// this function passed the jpeg image buffer
```

```
in, and returns the decoded
// image
int
decodeImage(OPENMAX_JPEG_DECODER * decoder,
char *sourceImage,
        size_t imageSize)
{
    char            *sourceOffset = sourceImage;
// we store a seperate
                        // buffer ot image so
we
                        // can offset it
    size_t          toread = 0;    // bytes
left to read from buffer
    toread += imageSize;
    int             bFilled = 0;    // have we
filled our output
                    // buffer
    bufferIndex = 0;

    OMX_BUFFERHEADERTYPE *pBufHeader;
    while (toread > 0) {
    // get next buffer from array
    pBufHeader =
        decoder-
>ppInputBufferHeader[bufferIndex];
```

```
    // step index and reset to 0 if required
    bufferIndex++;
    if (bufferIndex >= decoder-
>inputBufferHeaderCount)
        bufferIndex = 0;

    // work out the next chunk to load into the
decoder
    if (toread > pBufHeader->nAllocLen)
        pBufHeader->nFilledLen = pBufHeader-
>nAllocLen;
    else
        pBufHeader->nFilledLen = toread;

    toread = toread - pBufHeader->nFilledLen;

    // pass the bytes to the buffer
    memcpy(pBufHeader->pBuffer, sourceOffset,
pBufHeader->nFilledLen);

    // update the buffer pointer and set the
input flags

    sourceOffset = sourceOffset + pBufHeader-
>nFilledLen;
    pBufHeader->nOffset = 0;
    pBufHeader->nFlags = 0;
```

```
    if (toread <= 0) {
        pBufHeader->nFlags =
OMX_BUFFERFLAG_EOS;
    }
    // empty the current buffer
    int             ret =
        OMX_EmptyThisBuffer(decoder-
>imageDecoder->handle,
                pBufHeader);

    if (ret != OMX_ErrorNone) {
        perror("Empty input buffer");
        fprintf(stderr, "return code %x\n",
ret);
        return OMXJPEG_ERROR_MEMORY;
    }
    // wait for buffer to empty or port changed
event
    int             done = 0;
    while ((done == 0) || (decoder-
>pOutputBufferHeader == NULL)) {
        if (decoder->pOutputBufferHeader ==
NULL) {
        ret =
            ilclient_wait_for_event
            (decoder->imageDecoder->component,
             OMX_EventPortSettingsChanged,
```

```
                decoder->imageDecoder->outPort, 0,
0, 1, 0, 5);

        if (ret == 0) {
            portSettingsChanged(decoder);
        }
        } else {
        ret =
            ilclient_remove_event(decoder-
>imageDecoder->component,

OMX_EventPortSettingsChanged,
                        decoder->imageDecoder-
>outPort,
                        0, 0, 1);
        if (ret == 0)
            portSettingsChangedAgain(decoder);

        }

        // check to see if buffer is now empty
        if (pBufHeader->nFilledLen == 0)
        done = 1;

        if ((done == 0)
        || (decoder->pOutputBufferHeader ==
NULL))
```

```
        sleep(1);
    }

    // fill the buffer if we have created the
buffer
    if (bFilled == 0) {
        if ((decoder->pOutputBufferHeader ==
NULL)) {
        portSettingsChanged(decoder);
        }
        ret = OMX_FillThisBuffer(decoder-
>imageResizer->handle,
                            decoder-
>pOutputBufferHeader);
        if (ret != OMX_ErrorNone) {
        perror("Filling output buffer");
        fprintf(stderr, "Error code %x\n",
ret);
        return OMXJPEG_ERROR_MEMORY;
        }

        bFilled = 1;
    }
    }

    // wait for buffer to fill
    /*
```

```
    while(pBufHeader->nFilledLen == 0) {
printf("Sleeping\n"); sleep(5); }
    */

    // wait for end of stream events
    int             ret =
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventBufferFlag,
                decoder->imageDecoder->outPort,
1,
                OMX_BUFFERFLAG_EOS, 1,
                0, 2);
    if (ret != 0) {
    fprintf(stderr, "No EOS event on image
decoder %d\n", ret);
    }
    ret = ilclient_wait_for_event(decoder-
>imageResizer->component,
                  OMX_EventBufferFlag,
                  decoder->imageResizer-
>outPort, 1,
                  OMX_BUFFERFLAG_EOS, 1, 0, 2);
    if (ret != 0) {
    fprintf(stderr, "No EOS event on image
resizer %d\n", ret);
    }
```

```
    printf("Filled buffer len is %d\n",
pBufHeader->nFilledLen);
    save_image_as_TGA(decoder);

    return OMXJPEG_OK;
}

// this function cleans up the decoder.
void
cleanup(OPENMAX_JPEG_DECODER * decoder)
{
    // flush everything through
    OMX_SendCommand(decoder->imageDecoder-
>handle,
            OMX_CommandFlush, decoder-
>imageDecoder->outPort,
            NULL);
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
OMX_CommandFlush, 0,
                decoder->imageDecoder->outPort,
0, 0,
                TIMEOUT_MS);
    OMX_SendCommand(decoder->imageResizer-
>handle, OMX_CommandFlush,
```

```
            decoder->imageResizer->inPort,
NULL);
    ilclient_wait_for_event(decoder-
>imageResizer->component,
                OMX_EventCmdComplete,
OMX_CommandFlush, 0,
                decoder->imageResizer->inPort,
1, 0,
                TIMEOUT_MS);

    OMX_SendCommand(decoder->imageDecoder-
>handle, OMX_CommandPortDisable,
            decoder->imageDecoder->inPort,
NULL);

    int             i = 0;
    for (i = 0; i < decoder-
>inputBufferHeaderCount; i++) {
    OMX_BUFFERHEADERTYPE *vpBufHeader =
        decoder->ppInputBufferHeader[i];

    OMX_FreeBuffer(decoder->imageDecoder-
>handle,
               decoder->imageDecoder->inPort,
vpBufHeader);
    }
```

```
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
OMX_CommandPortDisable,
                0, decoder->imageDecoder-
>inPort, 0, 0,
                TIMEOUT_MS);

    OMX_SendCommand(decoder->imageResizer-
>handle, OMX_CommandPortDisable,
            decoder->imageResizer->outPort,
NULL);

    OMX_FreeBuffer(decoder->imageResizer-
>handle,
           decoder->imageResizer->outPort,
           decoder->pOutputBufferHeader);

    ilclient_wait_for_event(decoder-
>imageResizer->component,
                OMX_EventCmdComplete,
OMX_CommandPortDisable,
                0, decoder->imageResizer-
>outPort, 0, 0,
                TIMEOUT_MS);

    OMX_SendCommand(decoder->imageDecoder-
```

```
>handle, OMX_CommandPortDisable,
            decoder->imageDecoder->outPort,
NULL);
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
OMX_CommandPortDisable,
                0, decoder->imageDecoder-
>outPort, 0, 0,
                TIMEOUT_MS);

    OMX_SendCommand(decoder->imageResizer-
>handle, OMX_CommandPortDisable,
            decoder->imageResizer->inPort,
NULL);
    ilclient_wait_for_event(decoder-
>imageResizer->component,
                OMX_EventCmdComplete,
OMX_CommandPortDisable,
                0, decoder->imageResizer-
>inPort, 0, 0,
                TIMEOUT_MS);

    OMX_SetupTunnel(decoder->imageDecoder-
>handle,
            decoder->imageDecoder->outPort,
NULL, 0);
```

```
    OMX_SetupTunnel(decoder->imageResizer-
>handle,
            decoder->imageResizer->inPort,
NULL, 0);

    ilclient_change_component_state(decoder-
>imageDecoder->component,
                    OMX_StateIdle);
    ilclient_change_component_state(decoder-
>imageResizer->component,
                    OMX_StateIdle);

    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
OMX_CommandStateSet, 0,
                OMX_StateIdle, 0, 0,
TIMEOUT_MS);
    ilclient_wait_for_event(decoder-
>imageResizer->component,
                OMX_EventCmdComplete,
OMX_CommandStateSet, 0,
                OMX_StateIdle, 0, 0,
TIMEOUT_MS);

    ilclient_change_component_state(decoder-
>imageDecoder->component,
```

```
                    OMX_StateLoaded);
    ilclient_change_component_state(decoder-
>imageResizer->component,
                    OMX_StateLoaded);

    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
OMX_CommandStateSet, 0,
                OMX_StateLoaded, 0, 0,
TIMEOUT_MS);
    ilclient_wait_for_event(decoder-
>imageResizer->component,
                OMX_EventCmdComplete,
OMX_CommandStateSet, 0,
                OMX_StateLoaded, 0, 0,
TIMEOUT_MS);

    OMX_Deinit();

    if (decoder->client != NULL) {
    ilclient_destroy(decoder->client);
    }
}

int
main(int argc, char *argv[])
```

```
{
    OPENMAX_JPEG_DECODER *pDecoder;
    char            *sourceImage;
    size_t           imageSize;
    int              s;
    if (argc < 2) {
    printf("Usage: %s <filename>\n", argv[0]);
    return -1;
    }
    FILE            *fp = fopen(argv[1], "rb");
    if (!fp) {
    printf("File %s not found.\n", argv[1]);
    }
    fseek(fp, 0L, SEEK_END);
    imageSize = ftell(fp);
    fseek(fp, 0L, SEEK_SET);
    sourceImage = malloc(imageSize);
    assert(sourceImage != NULL);
    s = fread(sourceImage, 1, imageSize, fp);
    assert(s == imageSize);
    fclose(fp);
    bcm_host_init();
    s = setupOpenMaxJpegDecoder(&pDecoder);
    assert(s == 0);
    s = decodeImage(pDecoder, sourceImage,
imageSize);
    assert(s == 0);
```

```
    cleanup(pDecoder);
    free(sourceImage);
    return 0;
}
```

Rendering a JPEG image

In order to display am image we need to *render* it on some output device. There is no specific component to render images. Broadcom has a `video_render` which will accept *both* a video stream or a single image. Now this component only has one input port, and that port is a *video* port. If we want to take an image from an output *image* port, we would have to play some format conversion games. Or, we can connect an image output port to a video input port by using a tunnel, and let the components sort it out themselves. This is much easier!

To render a JPEG image, we only need to use an `image_decode` and a `video_render` component. We need to set up the input port for the decoder as in the last example. The video render component has no output port so nothing needs to be done for that. The two ports which are connected by a tunnel do all necessary setups and conversions themselves. So this is actually simpler than the last program: load and empty input buffers with the JPEG image, setup the tunnel when port settings change, and ... that's it! Note that we still defer setting up the tunnel until the port settings have changed - otherwise the buffer sizes would be wrong.

The program is renderjpeg.c

```
/*
Copyright (c) 2012, Matt Ownby
                    Anthong Sale
Copyright (c) 2014, Jan Newmarch

All rights reserved.

Redistribution and use in source and binary
forms, with or without
modification, are permitted provided that the
following conditions are met:
    * Redistributions of source code must
retain the above copyright
      notice, this list of conditions and the
following disclaimer.
    * Redistributions in binary form must
reproduce the above copyright
      notice, this list of conditions and the
following disclaimer in the
      documentation and/or other materials
provided with the distribution.
    * Neither the name of the copyright holder
nor the
      names of its contributors may be used to
endorse or promote products
      derived from this software without
```

```
specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
*/

#include <stdio.h>
#include <assert.h>
#include "jpeg.h"
```

```
#define TIMEOUT_MS 2000

typedef struct _COMPONENT_DETAILS {
    COMPONENT_T    *component;
    OMX_HANDLETYPE  handle;
    int             inPort;
    int             outPort;
} COMPONENT_DETAILS;

struct _OPENMAX_JPEG_DECODER {
    ILCLIENT_T     *client;
    COMPONENT_DETAILS *imageDecoder;
    COMPONENT_DETAILS *imageRender;
    OMX_BUFFERHEADERTYPE **ppInputBufferHeader;
    int             inputBufferHeaderCount;
    OMX_BUFFERHEADERTYPE *pOutputBufferHeader;
};

int             bufferIndex = 0;    // index to
buffer array

int
portSettingsChanged(OPENMAX_JPEG_DECODER *
decoder)
{
    OMX_PARAM_PORTDEFINITIONTYPE portdef;
```

```
    int ret;

    printf("Port settings changed\n");

    // disable ports
    OMX_SendCommand(decoder->imageDecoder-
>handle,
            OMX_CommandPortDisable,
            decoder->imageDecoder->outPort,
NULL);
    OMX_SendCommand(decoder->imageRender-
>handle,
            OMX_CommandPortDisable,
            decoder->imageRender->inPort,
NULL);

    // state to idle
    OMX_SendCommand(decoder->imageDecoder-
>handle,
            OMX_CommandStateSet, OMX_StateIdle,
NULL);
     OMX_SendCommand(decoder->imageRender-
>handle,
            OMX_CommandStateSet, OMX_StateIdle,
NULL);

    // establish tunnel between decoder output
```

```
and render input
    ret = OMX_SetupTunnel(decoder-
>imageDecoder->handle,
            decoder->imageDecoder->outPort,
            decoder->imageRender->handle,
            decoder->imageRender->inPort);
    if (ret != OMX_ErrorNone) {
    fprintf(stderr, "Error setting up tunnel
%X\n", ret);
    return OMXJPEG_ERROR_MEMORY;
    } else {
    printf("Tunnel set up ok\n");
    }

    // wait for ports to become disabled
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
                OMX_CommandPortDisable, 1,
                decoder->imageDecoder->outPort,
1, 0,
                TIMEOUT_MS);
    ilclient_wait_for_event(decoder-
>imageRender->component,
                OMX_EventCmdComplete,
OMX_CommandPortDisable, 1,
                decoder->imageRender->inPort,
```

```
1, 0,
                TIMEOUT_MS);
    printf("Port disable complete\n");

    // enable ports
    OMX_SendCommand(decoder->imageDecoder-
>handle,
            OMX_CommandPortEnable,
            decoder->imageDecoder->outPort,
NULL);
    OMX_SendCommand(decoder->imageRender-
>handle,
            OMX_CommandPortEnable,
            decoder->imageRender->inPort,
NULL);

    // wait for state change complete
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
                OMX_CommandStateSet, 1,
                OMX_StateIdle, 1, 0,
TIMEOUT_MS);
    printf("Decoder in idle state\n");

    ilclient_wait_for_event(decoder-
>imageRender->component,
```

```
                OMX_EventCmdComplete,
                OMX_CommandStateSet, 1,
                OMX_StateIdle, 1, 0,
TIMEOUT_MS);
    printf("Render in idle state\n");

    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
                OMX_CommandPortEnable, 1,
                decoder->imageDecoder->outPort,
1, 0,
                TIMEOUT_MS);
    ilclient_wait_for_event(decoder-
>imageRender->component,
                OMX_EventCmdComplete,
OMX_CommandPortEnable, 1,
                decoder->imageRender->inPort,
1, 0,
                TIMEOUT_MS);
    printf("Ports re-enabled\n");

    return OMXJPEG_OK;
}

int
prepareRender(OPENMAX_JPEG_DECODER * decoder)
```

```
{
    decoder->imageRender =
malloc(sizeof(COMPONENT_DETAILS));
    if (decoder->imageRender == NULL) {
    perror("malloc image render");
    return OMXJPEG_ERROR_MEMORY;
    }

    int             ret =
ilclient_create_component(decoder->client,
                            &decoder->
                            imageRender->
                            component,
                            "video_render",

ILCLIENT_DISABLE_ALL_PORTS
                            |

ILCLIENT_ENABLE_INPUT_BUFFERS
                            );
    if (ret != 0) {
    perror("image render");
    return OMXJPEG_ERROR_CREATING_COMP;
    }
    // grab the handle for later use
    decoder->imageRender->handle =
    ILC_GET_HANDLE(decoder->imageRender-
```

```
>component);

    // get and store the ports
    OMX_PORT_PARAM_TYPE port;
    port.nSize = sizeof(OMX_PORT_PARAM_TYPE);
    port.nVersion.nVersion = OMX_VERSION;

    OMX_GetParameter(ILC_GET_HANDLE(decoder-
>imageRender->component),
             OMX_IndexParamVideoInit, &port);
    if (port.nPorts != 1) {
    return OMXJPEG_ERROR_WRONG_NO_PORTS;
    }
    decoder->imageRender->inPort =
port.nStartPortNumber;
    /*
    decoder->imageRender->outPort =
port.nStartPortNumber + 1;
    */

    decoder->pOutputBufferHeader = NULL;

    return OMXJPEG_OK;
}

int
prepareImageDecoder(OPENMAX_JPEG_DECODER *
```

```
decoder)
{
    decoder->imageDecoder =
malloc(sizeof(COMPONENT_DETAILS));
    if (decoder->imageDecoder == NULL) {
    perror("malloc image decoder");
    return OMXJPEG_ERROR_MEMORY;
    }

    int             ret =
ilclient_create_component(decoder->client,
                            &decoder->
                            imageDecoder->
                            component,
                            "image_decode",

ILCLIENT_DISABLE_ALL_PORTS
                            |

ILCLIENT_ENABLE_INPUT_BUFFERS);

    if (ret != 0) {
    perror("image decode");
    return OMXJPEG_ERROR_CREATING_COMP;
    }
    // grab the handle for later use in OMX
calls directly
```

```
    decoder->imageDecoder->handle =
    ILC_GET_HANDLE(decoder->imageDecoder-
>component);

    // get and store the ports
    OMX_PORT_PARAM_TYPE port;
    port.nSize = sizeof(OMX_PORT_PARAM_TYPE);
    port.nVersion.nVersion = OMX_VERSION;

    OMX_GetParameter(decoder->imageDecoder-
>handle,
             OMX_IndexParamImageInit, &port);
    if (port.nPorts != 2) {
    return OMXJPEG_ERROR_WRONG_NO_PORTS;
    }
    decoder->imageDecoder->inPort =
port.nStartPortNumber;
    decoder->imageDecoder->outPort =
port.nStartPortNumber + 1;

    return OMXJPEG_OK;
}

int
startupImageDecoder(OPENMAX_JPEG_DECODER *
decoder)
{
```

```
    // move to idle
    ilclient_change_component_state(decoder-
>imageDecoder->component,
                        OMX_StateIdle);

    // set input image format
    OMX_IMAGE_PARAM_PORTFORMATTYPE
imagePortFormat;
    memset(&imagePortFormat, 0,
sizeof(OMX_IMAGE_PARAM_PORTFORMATTYPE));
    imagePortFormat.nSize =
sizeof(OMX_IMAGE_PARAM_PORTFORMATTYPE);
    imagePortFormat.nVersion.nVersion =
OMX_VERSION;
    imagePortFormat.nPortIndex = decoder-
>imageDecoder->inPort;
    imagePortFormat.eCompressionFormat =
OMX_IMAGE_CodingJPEG;
    OMX_SetParameter(decoder->imageDecoder-
>handle,
                OMX_IndexParamImagePortFormat,
&imagePortFormat);

    // get buffer requirements
    OMX_PARAM_PORTDEFINITIONTYPE portdef;
    portdef.nSize =
sizeof(OMX_PARAM_PORTDEFINITIONTYPE);
```

```
    portdef.nVersion.nVersion = OMX_VERSION;
    portdef.nPortIndex = decoder->imageDecoder-
>inPort;
    OMX_GetParameter(decoder->imageDecoder-
>handle,
             OMX_IndexParamPortDefinition,
&portdef);

    // enable the port and setup the buffers
    OMX_SendCommand(decoder->imageDecoder-
>handle,
            OMX_CommandPortEnable,
            decoder->imageDecoder->inPort,
NULL);
    decoder->inputBufferHeaderCount =
portdef.nBufferCountActual;
    // allocate pointer array
    decoder->ppInputBufferHeader =
    (OMX_BUFFERHEADERTYPE **)
malloc(sizeof(void) *
                     decoder-
>inputBufferHeaderCount);
    // allocate each buffer
    int             i;
    for (i = 0; i < decoder-
>inputBufferHeaderCount; i++) {
    if (OMX_AllocateBuffer(decoder-
```

```
>imageDecoder->handle,
                    &decoder-
>ppInputBufferHeader[i],
                    decoder->imageDecoder-
>inPort,
                    (void *) i,
                    portdef.nBufferSize) !=
OMX_ErrorNone) {
        perror("Allocate decode buffer");
        return OMXJPEG_ERROR_MEMORY;
    }
    }
    // wait for port enable to complete - which
it should once buffers are
    // assigned
    int             ret =
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
                OMX_CommandPortEnable, 0,
                decoder->imageDecoder->inPort,
0,
                0, TIMEOUT_MS);
    if (ret != 0) {
    fprintf(stderr, "Did not get port enable
%d\n", ret);
    return OMXJPEG_ERROR_EXECUTING;
```

```
    }
    // start executing the decoder
    ret = OMX_SendCommand(decoder-
>imageDecoder->handle,
              OMX_CommandStateSet,
OMX_StateExecuting, NULL);
    if (ret != 0) {
    fprintf(stderr, "Error starting image
decoder %x\n", ret);
    return OMXJPEG_ERROR_EXECUTING;
    }
    ret = ilclient_wait_for_event(decoder-
>imageDecoder->component,
                  OMX_EventCmdComplete,
                  OMX_StateExecuting, 0, 0, 1,
0,
                  TIMEOUT_MS);
    if (ret != 0) {
    fprintf(stderr, "Did not receive executing
stat %d\n", ret);
    // return OMXJPEG_ERROR_EXECUTING;
    }

    return OMXJPEG_OK;
}

// this function run the boilerplate to setup
```

```
the openmax components;
int
setupOpenMaxJpegDecoder(OPENMAX_JPEG_DECODER **
pDecoder)
{
    *pDecoder =
malloc(sizeof(OPENMAX_JPEG_DECODER));
    if (pDecoder[0] == NULL) {
    perror("malloc decoder");
    return OMXJPEG_ERROR_MEMORY;
    }
    memset(*pDecoder, 0,
sizeof(OPENMAX_JPEG_DECODER));

    if ((pDecoder[0]->client = ilclient_init())
== NULL) {
    perror("ilclient_init");
    return OMXJPEG_ERROR_ILCLIENT_INIT;
    }

    if (OMX_Init() != OMX_ErrorNone) {
    ilclient_destroy(pDecoder[0]->client);
    perror("OMX_Init");
    return OMXJPEG_ERROR_OMX_INIT;
    }
    // prepare the image decoder
    int             ret =
```

```
prepareImageDecoder(pDecoder[0]);
    if (ret != OMXJPEG_OK)
    return ret;

    ret = prepareRender(pDecoder[0]);
    if (ret != OMXJPEG_OK)
    return ret;

    ret = startupImageDecoder(pDecoder[0]);
    if (ret != OMXJPEG_OK)
    return ret;

    return OMXJPEG_OK;
}

// this function passed the jpeg image buffer
in, and returns the decoded
// image
int
decodeImage(OPENMAX_JPEG_DECODER * decoder,
char *sourceImage,
        size_t imageSize)
{
    char            *sourceOffset = sourceImage;
// we store a seperate
                        // buffer ot image so
we
```

```
                            // can offset it
    size_t          toread = 0;    // bytes
left to read from buffer
    toread += imageSize;
    int             bFilled = 0;    // have we
filled our output
    int bTunnelCreated = 0;            //
buffer

    bufferIndex = 0;

    OMX_BUFFERHEADERTYPE *pBufHeader;
    while (toread > 0) {
    // get next buffer from array
    pBufHeader =
        decoder-
>ppInputBufferHeader[bufferIndex];

    // step index and reset to 0 if required
    bufferIndex++;
    if (bufferIndex >= decoder-
>inputBufferHeaderCount)
        bufferIndex = 0;

    // work out the next chunk to load into the
decoder
    if (toread > pBufHeader->nAllocLen)
```

```
        pBufHeader->nFilledLen = pBufHeader-
>nAllocLen;
    else
        pBufHeader->nFilledLen = toread;

    toread = toread - pBufHeader->nFilledLen;

    // pass the bytes to the buffer
    memcpy(pBufHeader->pBuffer, sourceOffset,
pBufHeader->nFilledLen);

    // update the buffer pointer and set the
input flags

    sourceOffset = sourceOffset + pBufHeader-
>nFilledLen;
    pBufHeader->nOffset = 0;
    pBufHeader->nFlags = 0;
    if (toread <= 0) {
        pBufHeader->nFlags =
OMX_BUFFERFLAG_EOS;
        printf("EOS added to stream\n");
    }
    printf("Input buffer called to empty %d\n",
           pBufHeader->nFilledLen);

    // empty the current buffer
```

```
    int                 ret =
        OMX_EmptyThisBuffer(decoder-
>imageDecoder->handle,
                pBufHeader);

    if (ret != OMX_ErrorNone) {
        fprintf(stderr, "Empty input buffer
return code %x\n", ret);
        return OMXJPEG_ERROR_MEMORY;
    }

    // wait for buffer to empty or port changed
event
    int                 done = 0;
    //while ((done == 0) || (decoder-
>pOutputBufferHeader == NULL)) {
    while (done == 0) {
        // if (decoder->pOutputBufferHeader ==
NULL) {
        if (! bTunnelCreated) {
        printf("Tunnel not yet created\n");
        ret =
            ilclient_wait_for_event
            (decoder->imageDecoder->component,
             OMX_EventPortSettingsChanged,
             decoder->imageDecoder->outPort, 0,
0, 1, 0, 5);
```

```
        if ((ret == 0) && ! bTunnelCreated) {
            portSettingsChanged(decoder);
            bTunnelCreated = 1;
        }
        }

        // check to see if buffer is now empty
        if (pBufHeader->nFilledLen == 0) {
        done = 1;
        printf("Input buffer finished
emptying\n");
        } else {
        printf("Still %d in input buffer\n",
               pBufHeader->nFilledLen);
        // is there an event we can wait on
instead of this?
        sleep(1);
        }
    }
    }

    // wait for end of stream events
    int             ret =
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventBufferFlag,
```

```
                decoder->imageDecoder->outPort,
1,
                OMX_BUFFERFLAG_EOS, 1,
                0, 2);
    if (ret != 0) {
    fprintf(stderr, "No EOS event on image
decoder %d\n", ret);
    } else {
    printf("EOS event on decoder\n");
    }
    return OMXJPEG_OK;
}

// this function cleans up the decoder.
void
cleanup(OPENMAX_JPEG_DECODER * decoder)
{
    // flush everything through
    OMX_SendCommand(decoder->imageDecoder-
>handle,
            OMX_CommandFlush, decoder-
>imageDecoder->outPort,
            NULL);
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
OMX_CommandFlush, 0,
```

```
                decoder->imageDecoder->outPort,
0, 0,
                TIMEOUT_MS);
    OMX_SendCommand(decoder->imageRender-
>handle, OMX_CommandFlush,
            decoder->imageRender->inPort,
NULL);
    ilclient_wait_for_event(decoder-
>imageRender->component,
                OMX_EventCmdComplete,
OMX_CommandFlush, 0,
                decoder->imageRender->inPort,
1, 0,
                TIMEOUT_MS);

    OMX_SendCommand(decoder->imageDecoder-
>handle, OMX_CommandPortDisable,
            decoder->imageDecoder->inPort,
NULL);

    int             i = 0;
    for (i = 0; i < decoder-
>inputBufferHeaderCount; i++) {
    OMX_BUFFERHEADERTYPE *vpBufHeader =
        decoder->ppInputBufferHeader[i];

    OMX_FreeBuffer(decoder->imageDecoder-
```

```
>handle,
                decoder->imageDecoder->inPort,
vpBufHeader);
    }

    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
OMX_CommandPortDisable,
                0, decoder->imageDecoder-
>inPort, 0, 0,
                TIMEOUT_MS);

    OMX_SendCommand(decoder->imageDecoder-
>handle, OMX_CommandPortDisable,
            decoder->imageDecoder->outPort,
NULL);
    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
OMX_CommandPortDisable,
                0, decoder->imageDecoder-
>outPort, 0, 0,
                TIMEOUT_MS);

    OMX_SendCommand(decoder->imageRender-
>handle, OMX_CommandPortDisable,
```

```
            decoder->imageRender->inPort,
NULL);
    ilclient_wait_for_event(decoder-
>imageRender->component,
                OMX_EventCmdComplete,
OMX_CommandPortDisable,
                0, decoder->imageRender-
>inPort, 0, 0,
                TIMEOUT_MS);

    OMX_SetupTunnel(decoder->imageDecoder-
>handle,
            decoder->imageDecoder->outPort,
NULL, 0);
    OMX_SetupTunnel(decoder->imageRender-
>handle,
            decoder->imageRender->inPort, NULL,
0);

    ilclient_change_component_state(decoder-
>imageDecoder->component,
                    OMX_StateIdle);
    ilclient_change_component_state(decoder-
>imageRender->component,
                    OMX_StateIdle);

    ilclient_wait_for_event(decoder-
```

```
>imageDecoder->component,
                OMX_EventCmdComplete,
OMX_CommandStateSet, 0,
                OMX_StateIdle, 0, 0,
TIMEOUT_MS);
    ilclient_wait_for_event(decoder-
>imageRender->component,
                OMX_EventCmdComplete,
OMX_CommandStateSet, 0,
                OMX_StateIdle, 0, 0,
TIMEOUT_MS);

    ilclient_change_component_state(decoder-
>imageDecoder->component,
                    OMX_StateLoaded);
    ilclient_change_component_state(decoder-
>imageRender->component,
                    OMX_StateLoaded);

    ilclient_wait_for_event(decoder-
>imageDecoder->component,
                OMX_EventCmdComplete,
OMX_CommandStateSet, 0,
                OMX_StateLoaded, 0, 0,
TIMEOUT_MS);
    ilclient_wait_for_event(decoder-
>imageRender->component,
```

```
                    OMX_EventCmdComplete,
OMX_CommandStateSet, 0,
                    OMX_StateLoaded, 0, 0,
TIMEOUT_MS);

    OMX_Deinit();

    if (decoder->client != NULL) {
    ilclient_destroy(decoder->client);
    }
}

int
main(int argc, char *argv[])
{
    OPENMAX_JPEG_DECODER *pDecoder;
    char            *sourceImage;
    size_t           imageSize;
    int              s;
    if (argc < 2) {
    printf("Usage: %s <filename>\n", argv[0]);
    return -1;
    }
    FILE            *fp = fopen(argv[1], "rb");
    if (!fp) {
    printf("File %s not found.\n", argv[1]);
    }
```

```
    fseek(fp, 0L, SEEK_END);
    imageSize = ftell(fp);
    fseek(fp, 0L, SEEK_SET);
    sourceImage = malloc(imageSize);
    assert(sourceImage != NULL);
    s = fread(sourceImage, 1, imageSize, fp);
    assert(s == imageSize);
    fclose(fp);
    printf("File size is %d\n", imageSize);
    bcm_host_init();
    s = setupOpenMaxJpegDecoder(&pDecoder);
    assert(s == 0);
    s = decodeImage(pDecoder, sourceImage,
imageSize);
    assert(s == 0);

    // the image will be displayed until the
program exits
    sleep(100);
    cleanup(pDecoder);
    free(sourceImage);
    return 0;
}
```

Rendering a JPEG image without tunnelling

Tunnelling gives an end-to-end means of decoding and rendering an image. If we wanted to do things in the middle - say, overlay some text - we would need to have access to the intermediate stages. In this section we repeat the previous section, but without using tunnelling.

We have to explicitly manage the communication between the decode and render components. For the decode component, we have to allocate a buffer. We don't want to copy data between decode and render buffers so the render component *uses* the decoder's output buffers. Then as the decoder fills its output buffer, the render component empties that same buffer.

There are multiple hiccups along the way

- Lots of information has to be copied from the decode port to the render port. We can't just use a `memcpy` because the decoder has a structure for an *image* while the render has a structure for a *video* , and these do not align.

```
// need to setup the input for the render
with the output of the
  // decoder
  portdef.nSize =
sizeof(OMX_PARAM_PORTDEFINITIONTYPE);
  portdef.nVersion.nVersion = OMX_VERSION;
  portdef.nPortIndex = decoder-
>imageDecoder->outPort;
  OMX_GetParameter(decoder->imageDecoder-
>handle,
           OMX_IndexParamPortDefinition,
&portdef);

  // Get default values of render
  rportdef.nSize =
sizeof(OMX_PARAM_PORTDEFINITIONTYPE);
  rportdef.nVersion.nVersion = OMX_VERSION;
  rportdef.nPortIndex = decoder-
>imageRender->inPort;
  rportdef.nBufferSize =
portdef.nBufferSize;

  ret = OMX_GetParameter(decoder-
>imageRender->handle,
             OMX_IndexParamPortDefinition,
&rportdef);
```

```
  // tell render input what the decoder
output will be providing
  //Copy some
  rportdef.format.video.nFrameWidth =
portdef.format.image.nFrameWidth;
  rportdef.format.video.nFrameHeight =
portdef.format.image.nFrameHeight;
  rportdef.format.video.nStride =
portdef.format.image.nStride;
  rportdef.format.video.nSliceHeight =
portdef.format.image.nSliceHeight;

  ret = OMX_SetParameter(decoder-
>imageRender->handle,
             OMX_IndexParamPortDefinition,
&rportdef);
```

- The decode component has one output buffer while the render component has three input buffers. We re-use one buffer and set the other two to `NULL`

```
ret = OMX_AllocateBuffer(decoder-
>imageDecoder->handle,
               &decoder-
>pOutputBufferHeader,
               decoder->imageDecoder->
               outPort,
               NULL,
               portdef.nBufferSize);

  // and share it with the renderer
  // which has 3 default buffers, 2 minimum
  decoder->ppRenderInputBufferHeader =
  (OMX_BUFFERHEADERTYPE **)
malloc(sizeof(void) *
                  decoder-
>renderInputBufferHeaderCount);

  ret = OMX_UseBuffer(decoder->imageRender-
>handle,
          &decoder-
>ppRenderInputBufferHeader[0],
          decoder->imageRender->inPort,
          NULL,
          rportdef.nBufferSize,
          decoder->pOutputBufferHeader-
>pBuffer);
```

```
  int n;
  for (n = 1; n < decoder-
>renderInputBufferHeaderCount; n++) {
  printState(decoder->imageRender->handle);
  ret = OMX_UseBuffer(decoder->imageRender-
>handle,
              &
amp;decoder->ppRenderInputBufferHeader[n],
              decoder->imageRender->inPort,
              NULL,
              0,
              NULL);
  }
```

- Even though we feed the size of the shared buffer into `UseBuffer` call, the field `nAllocLen` does not get set correctly (how do we know: because of an illegal parameter error, and then using the debugger to guess at what isn't right).

```
decoder->ppRenderInputBufferHeader[0]-
>nAllocLen =
  decoder->pOutputBufferHeader->nAllocLen;
```

Apart from that, it is the usual games of playing with state, enabling and disabling ports until it works

Conclusion

The RPi has a useful GPU. Programming it using OpenMAX is hard work, though. This chapter has covered some of the issues. The topic is explored more in my book Programming the Raspberry Pi's GPU

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

PART 4: MIDI

Just some links to resources for now

Overview of MIDI

There are many overviews of MIDI. Here are some

- Introduction to Computer Music: Volume One. Chapter Three: MIDI
- MIDI Manufacturers Association: Tutorials
- MIDI Basic Tutorials
- Ted's Linux MIDI Guide
- Standard MIDI-File Format Spec. 1.1

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

This chapter gives an overview of the principal tools used for playing MIDI files. It does not include MIDI editors, MIDI producers, etc

Resources

- Ted's Linux MIDI Guide

Sound fonts

The tools described in this chapter each include a software synthesizer. This produces audio as PCM data from the MID data fed to it. The MIDI data contains information about the instrument playing each note, and of course, every instrument sounds different. So the synthesizer must make use of mapping information from MIDI notes + instrument into PCM data.

The mapping is usually done using *soundfont* files. There are various formats for this, of which the primary one is the .sf2 format. Some synthesizers (such as Timidity) can also use Gravis UltraSound patches, which are recorded real instruments.

Many soundfont files have been created. See for example Links to SoundFonts and other similar files (although many of the links are broken).

- A common sound font is from FluidSynth, as `/usr/share/sounds/sf2/FluidR3_GM.sf2`. This file is nearly 150Mbytes - soundfonts are not small!
- JavaSound has a soundfont `soundbank-emg.sf2`. This is considerably smaller at 1.9Mb!

- Another popular soundfont is at GeneralUser_GS_1.44-MuseScore by S. Christian Collins. This is not so large, at 31Mb.
- A small soundfont is by Tim Brechbill at 6Mb (linked from MuseScore - soundfonts)
- A list of soundfonts is at TiMidity++ Configuration File Package v2004/8/3

Possibly surprisingly, using different soundfonts doesn't seem to make much difference to CPU usage - for FluidSynth they all use about 60-70% CPU on one song. They do, of course, sound different.

Timidity

Timidity is a "Software sound renderer (MIDI sequencer, MOD player)"

Its home page is Maemo.org

Timidity can be used to play MIDI files by giving them on the command line

```
timidity rehab.mid
```

The default soundfonts used by Timidity are Gravis UltraSound patches, from the `/usr/share/midi/freepats/` directory. These soundfonts are missing many instruments, so should be replaced by another such as the FluidSynth fonts. The settings are made in the configurations file `/etc/timidity/timidity.cfg`

TiMidity as a server

It can also be run as an ALSA server listening on a port (Using MIDI with UNIX):

```
timidity -iAD -B2,8 -Os1l -s 44100
```

The "-iAD" option runs it as a Daemon process in the background as an ALSA sequencer client. The "-B2,8" option selects the number of buffer fragments. The "-Os1l" option selects ALSA output as PCM. The "-s" option is the sample size. (For the Raspberry Pi, I found that `-B0,12` worked better than `-B2,8`).

In this mode, ALSA can send messages to it. The command

```
aconnect -0
```

will show output such as

```
client 14: 'Midi Through' [type=kernel]
      0 'Midi Through Port-0'

laptop:/home/httpd/html/LinuxSound/MIDI/Python/py
0.0.3$aconnect -o
      client 14: 'Midi Through' [type=kernel]
      0 'Midi Through Port-0'
      client 128: 'TiMidity' [type=user]
      0 'TiMidity port 0 '
      1 'TiMidity port 1 '
      2 'TiMidity port 2 '
      3 'TiMidity port 3 '
```

The "Midi Through" port is not useful but the Timidity ports are. MIDI files can then be played by an ALSA sequencer such as

```
aplaymidi -p128:0 rehab.mid
```

Setting Timidity output device

The default output for TiMidity can be changed using the `-O` option. TiMidity help (`timidity -h`) shows, for example,

```
Available output modes (-O, --output-mode
option):
      -Os          ALSA pcm device
      -Ow          RIFF WAVE file
      -Or          Raw waveform data
      -Ou          Sun audio file
      -Oa          AIFF file
      -Ol          List MIDI event
      -Om          Write MIDI file
      -OM          MOD -> MIDI file conversion
```

For some of these modes the device name can also be set, using the `-o` option. For example, to play a file using the `hw:2` ALSA device, use

```
timidity -Os -o hw:2 ...
```

TiMidity and Jack

TiMidity can be run with Jack output using the `-Oj` option. In a user-based environment such as Ubuntu you may need to stop or pause PulseAudio, start the Jack server and then run TiMidity. PulseAudion may be paused by e.g.

```
pasuspender cat
```

in one terminal. In another, start the Jack daemon using ALSA input and output

```
jackd -dalsa
```

In a third terminal, run TiMidity

```
timidity -Oj 54154.mid
```

The links may be shown graphically by also running `qjackctl`.

Rosegarden

"Rosegarden is a well-rounded audio and MIDI sequencer, score editor, and general-purpose music composition and editing environment"

Its home page is RoseGarden

It is not a standalone synthsesizer: it uses FluidSynth, for example.

GStreamer

GStreamer allows you to build "pipelines" that can be played using `gst-launch`. It can play MIDI files by e.g.

```
gst-launch filesrc location="rehab.mid" !
decodebin ! alsasink
```

FluidSynth

`fluidsynth` is a command line MIDI player. It runs under ALSA with a command line

```
fluidsynth -a alsa -l <soundfont> <files...>
```

The soundfont is set explicitly on the command line, so can be set to another soundfont.

`qsynth` is a GUI interface to `fluidsynth`.

You can use FluidSynth to convert MIDI files to WAV files by

```
fluidsynth -F out.wav
/usr/share/sounds/sf2/FluidR3_GM.sf2 myfile.mid
```

Fluidsynth as a server

Fluidsynth can be run as a server in the same way as TiMidity. Use

```
fluidsynth --server --audio-driver=alsa
/usr/share/sounds/sf2/FluidR3_GM.sf2
```

Then `a connect -o` will show the ports and it can be played to by e.g.

```
amidi -p 128:0 <midi-file>
```

Wild MIDI

The aim of this sequencer/synthesizer is to be *small* . It succeeds at this!.

Comparison

On playing the same song with the different systems, I observed the following CPU patterns:

- **Timidity + pulseaudio (with GUS or SF2 soundfonts)**: 12-20% CPU
- **FluidSynth + pulseaudio**: 65-72% CPU
- **WildMIDI**: 6% CPU
- **JavaSound**: 52-60%
- **GStreamer**: 15-20% CPU

VLC

VLC is a general purpose media player. There is a VLC module to handle MIDI files using FluidSynth. To get this working on a Debian system you first need to intall the `vlc-plugin-fluidsynth` package. Then in Advanced Options of VLC, choose Codecs-Audio Codecs-FluidSynth. You will need to set the soundfont, eg. to `/usr/share/sounds/sf2/FluidR3_GM.sf2`.

Conclusion

This chapter has looked at a variety of user level tools for manipulating MIDI. It has primarily includde players, but there are also a large number of MIDI editors, producers, etc.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

JavaSound has a very well developed MIDI system, with cleanly separated components such as Sequencers and Synthesizers, and also allows hooks for Meta events as well as ordinary MIDI events.

Introduction

Java Sound has been around since early days of Java. It deals with both sampled and MIDI data. This chapter considers programming using the MIDI API.

Resources

There are many resources available for Java Sound:

- The Java™ Platform, Standard Edition 7 API Specification is the reference point for all of the standard Java APIs, including `javax.sound.sampled`.
- The Java Tutorials - Trail: Sound give a good overview of both the sampled and MIDI packages.
- The Java Sound Resources: FAQ: Audio Programming answers many questions about Java Sound
- JavaTM Sound Programmer Guide is a full book from Oracle (formerly Sun MicroSystems) about Java Sound
- The Sound Group is "comprised of those developers designing, implementing and maintaining the various OpenJDK sound components". There's your hook into finding out more about the ongoing development of Java Sound in the Open Source community.
- Gervill S/W sound synthesizer source

Key JavaSound MIDI classes

- The `MidiSystem` class is the entry point for all MIDI classes.
- A `MidiDevice` includes synthesizers, sequencers, MIDI input ports, and MIDI output ports
- A `Transmitter` sends `MidiEvent` objects to a `Receiver`. A `Transmitter` is the *source* of MIDI events and a `Receiver` is a *consumer* of events.
- "A `Sequencer` is a device for capturing and playing back sequences of MIDI events. It has transmitters, because it typically sends the MIDI messages stored in the sequence to another device, such as a synthesizer or MIDI output port. It also has receivers, because it can capture MIDI messages and store them in a sequence." (Java Sound Programmers Manual: Chapter 8: Overview of the MIDI Package)
- "A `Synthesizer` is a device for generating sound. It's the only object in the javax.sound.midi package that produces audio data" (Java Sound Programmers Manual: Chapter 8: Overview of the MIDI Package)

Device Information

Device information is found by querying `MidiSystem` for its list of `DeviceInfo` objects. Each information object contains fields such as Name and Vendor. The actual device may be found using this information object by `MidiSystem.getMidiDevice(info)`. The device may then be queried for its receivers and transmitters and its type as sequencer or synthesizer.

One annoying part is that you cannot get a list of all the devices's transmitters and receivers, only those that are *open* . You can ask for the default transmitter and receiver which will implicitly open them. So you can see that the list may be empty before asking for the default, but will be non-empty afterwards if there is a default! if there are no defaults, a `MidiUnavailableException` exception will be thrown.

The program is:

```
import javax.sound.midi.*;
import java.util.*;

public class DeviceInfo {

    public static void main(String[] args)
throws Exception {
    MidiDevice.Info[] devices;

    /*
    MidiDevice.Info[] info = p.getDeviceInfo();
    for (int m = 0; m < info.length; m++) {
        System.out.println(info[m].toString());
    }
    */

    System.out.println("MIDI devices:");
    devices = MidiSystem.getMidiDeviceInfo();
    for (MidiDevice.Info info: devices) {
        System.out.println("    Name: " +
info.toString() +
                    ", Decription: " +
                    info.getDescription() +
                    ", Vendor: " +
                    info.getVendor());
        MidiDevice device =
```

```
MidiSystem.getMidiDevice(info);
        if (! device.isOpen()) {
        device.open();
        }
        if (device instanceof Sequencer) {
        System.out.println("        Device is a
sequencer");
        }
        if (device instanceof Synthesizer) {
        System.out.println("        Device is a
synthesizer");
        }
        System.out.println("        Open
receivers:");
        List<Receiver> receivers =
device.getReceivers();
        for (Receiver r: receivers) {
        System.out.println("            " +
r.toString());
        }
        try {
        System.out.println("\n        Default
receiver: " +

device.getReceiver().toString());

        System.out.println("\n        Open
```

```
receivers now:");
        receivers = device.getReceivers();
        for (Receiver r: receivers) {
            System.out.println("            " +
r.toString());
        }
        } catch(MidiUnavailableException e) {
        System.out.println("        No default
receiver");
        }

        System.out.println("\n        Open
transmitters:");
        List<Transmitter> transmitters =
device.getTransmitters();
        for (Transmitter t: transmitters) {
        System.out.println("            " +
t.toString());
        }
        try {
        System.out.println("\n        Default
transmitter: " +

device.getTransmitter().toString());

        System.out.println("\n        Open
transmitters now:");
```

```
        transmitters =
device.getTransmitters();
        for (Transmitter t: transmitters) {
            System.out.println("            " +
t.toString());
        }
        } catch(MidiUnavailableException e) {
        System.out.println("        No default
transmitter");
        }
        device.close();
    }



    Sequencer sequencer =
MidiSystem.getSequencer();
    System.out.println("Default system
sequencer is " +

sequencer.getDeviceInfo().toString() +
               " (" + sequencer.getClass() +
")");

    Synthesizer synthesizer =
MidiSystem.getSynthesizer();
    System.out.println("Default system
synthesizer is " +
```

```
synthesizer.getDeviceInfo().toString() +
                " (" + synthesizer.getClass() +
")");

    }
}
```

The output on my system is

```
MIDI devices:
    Name: Gervill, Decription: Software MIDI
Synthesizer, Vendor: OpenJDK
        Device is a synthesizer
        Open receivers:

        Default receiver:
com.sun.media.sound.SoftReceiver@72f2a824

        Open receivers now:

com.sun.media.sound.SoftReceiver@72f2a824

        Open transmitters:
        No default transmitter
    Name: Real Time Sequencer, Decription:
Software sequencer, Vendor: Oracle Corporation
        Device is a sequencer
        Open receivers:

        Default receiver:
com.sun.media.sound.RealTimeSequencer$SequencerRe



        Open receivers now:
```

```
com.sun.media.sound.RealTimeSequencer$SequencerRe



        Open transmitters:
        Default transmitter:
com.sun.media.sound.RealTimeSequencer$SequencerTr



        Open transmitters now:

com.sun.media.sound.RealTimeSequencer$SequencerTr

Default system sequencer is Real Time Sequencer
Default system synthesizer is Gervill
```

Dumping a MIDI file

These two programs from jsresources.org dump a MIDI file to the console. The `MidiSystem` creates a `Sequence` from a file. Each track of the sequence is looped through and each event within each track is examined. While it would be possible to print *in situ* , each event is passed to a `Receiver` object which in this case is `DumpReceiver` . That object could do anything, but in this case just prints the event to stdout.

DumpSequence.java is

```
/*
 *    DumpSequence.java
 *
 *    This file is part of jsresources.org
 */

/*
 * Copyright (c) 1999, 2000 by Matthias
Pfisterer
 * All rights reserved.
 *
 * Redistribution and use in source and binary
forms, with or without
 * modification, are permitted provided that
the following conditions
 * are met:
 *
 * - Redistributions of source code must retain
the above copyright notice,
 *   this list of conditions and the following
disclaimer.
 * - Redistributions in binary form must
reproduce the above copyright
 *   notice, this list of conditions and the
following disclaimer in the
 *   documentation and/or other materials
```

```
provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN
NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

```
/*
  |<---            this code is formatted to
fit into 80 columns             --->|
*/

import java.io.File;
import java.io.IOException;

import javax.sound.midi.MidiSystem;
import
javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.Sequence;
import javax.sound.midi.Track;
import javax.sound.midi.MidiEvent;
import javax.sound.midi.MidiMessage;
import javax.sound.midi.ShortMessage;
import javax.sound.midi.MetaMessage;
import javax.sound.midi.SysexMessage;
import javax.sound.midi.Receiver;




/**    <titleabbrev>DumpSequence</titleabbrev>
    <title>Displaying the content of a MIDI
file</title>
```

```
    <formalpara><title>Purpose</title>
    <para>Dumps the decoded content of a MIDI
file to the console.</para>
    </formalpara>

    <formalpara><title>Usage</title>
    <para>
    <cmdsynopsis><command>java
DumpSequence</command>
    <arg choice="plain"><replaceable
class="parameter">midifile</replaceable></arg>
    </cmdsynopsis>
    </para></formalpara>

    <formalpara><title>Parameters</title>
    <variablelist>
    <varlistentry>
    <term><replaceable
class="parameter">midifile</replaceable></term>
    <listitem><para>the filename of the MIDI
file that should be displayed</para></listitem>
    </varlistentry>
    </variablelist>
    </formalpara>

    <formalpara><title>Bugs,
limitations</title>
```

```
    <para>Meta and system common events are not
displayed in detail.</para>
    </formalpara>

    <formalpara><title>Source code</title>
    <para>
    <ulink
url="DumpSequence.java.html">DumpSequence.java</u

    <ulink
url="DumpReceiver.java.html">DumpReceiver.java</u

    </para>
    </formalpara>

*/
public class DumpSequence
{
    private static String[]    sm_astrKeyNames
= {"C", "C#", "D", "D#", "E", "F", "F#", "G",
"G#", "A", "A#", "B"};

    private static Receiver        sm_receiver
= new DumpReceiver(System.out, true);
```

```
    public static void main(String[] args)
    {
    /*
     *    We check that there is exactely one
command-line
     *    argument. If not, we display the
usage message and
     *    exit.
     */
    if (args.length != 1)
        {
        out("DumpSequence: usage:");
        out("\tjava DumpSequence <midifile>");
        System.exit(1);
        }
    /*
     *    Now, that we're shure there is an
argument, we take it as
     *    the filename of the soundfile we want
to play.
     */
    String    strFilename = args[0];
    File    midiFile = new File(strFilename);

    /*
     *    We try to get a Sequence object,
```

```
which the content
     *    of the MIDI file.
     */
    Sequence    sequence = null;
    try
        {
        sequence =
MidiSystem.getSequence(midiFile);
        }
    catch (InvalidMidiDataException e)
        {
        e.printStackTrace();
        System.exit(1);
        }
    catch (IOException e)
        {
        e.printStackTrace();
        System.exit(1);
        }

    /*
     *    And now, we output the data.
     */
    if (sequence == null)
        {
        out("Cannot retrieve Sequence.");
        }
```

```
    else
        {
        out("-----------------------------------
----------------------------------------");
        out("File: " + strFilename);
        out("-----------------------------------
----------------------------------------");
        out("Length: " +
sequence.getTickLength() + " ticks");
        out("Duration: " +
sequence.getMicrosecondLength() + "
microseconds");
        out("-----------------------------------
----------------------------------------");
        float    fDivisionType =
sequence.getDivisionType();
        String    strDivisionType = null;
        if (fDivisionType == Sequence.PPQ)
            {
            strDivisionType = "PPQ";
            }
        else if (fDivisionType ==
Sequence.SMPTE_24)
            {
            strDivisionType = "SMPTE, 24 frames
per second";
            }
```

```
        else if (fDivisionType ==
Sequence.SMPTE_25)
            {
            strDivisionType = "SMPTE, 25 frames
per second";
            }
        else if (fDivisionType ==
Sequence.SMPTE_30DROP)
            {
            strDivisionType = "SMPTE, 29.97
frames per second";
            }
        else if (fDivisionType ==
Sequence.SMPTE_30)
            {
            strDivisionType = "SMPTE, 30 frames
per second";
            }

        out("DivisionType: " +
strDivisionType);

        String    strResolutionType = null;
        if (sequence.getDivisionType() ==
Sequence.PPQ)
            {
            strResolutionType = " ticks per
```

```
beat";
            }
        else
            {
            strResolutionType = " ticks per
frame";
            }
        out("Resolution: " +
sequence.getResolution() + strResolutionType);
        out("-----------------------------------
----------------------------------------");
        Track[]    tracks =
sequence.getTracks();
        for (int nTrack = 0; nTrack <
tracks.length; nTrack++)
            {
            out("Track " + nTrack + ":");
            out("------------------------");
            Track    track = tracks[nTrack];
            for (int nEvent = 0; nEvent <
track.size(); nEvent++)
                {
                MidiEvent    event =
track.get(nEvent);
                output(event);
                }
            out("------------------------------
```

```
-----------------------------------------------
");
            }
        // TODO: getPatchList()
        }
    }


    public static void output(MidiEvent event)
    {
    MidiMessage    message =
event.getMessage();
    long        lTicks = event.getTick();
    sm_receiver.send(message, lTicks);
    }



    private static void out(String strMessage)
    {
    System.out.println(strMessage);
    }
}



/*** DumpSequence.java ***/
```

DmpReceiver.java is

```
/*
 *    DumpReceiver.java
 *
 *    This file is part of jsresources.org
 */

/*
 * Copyright (c) 1999 - 2001 by Matthias
Pfisterer
 * Copyright (c) 2003 by Florian Bomers
 * All rights reserved.
 *
 * Redistribution and use in source and binary
forms, with or without
 * modification, are permitted provided that
the following conditions
 * are met:
 *
 * - Redistributions of source code must retain
the above copyright notice,
 *   this list of conditions and the following
disclaimer.
 * - Redistributions in binary form must
reproduce the above copyright
 *   notice, this list of conditions and the
following disclaimer in the
```

```
 *   documentation and/or other materials
provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN
NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

```
import    java.io.PrintStream;

import    javax.sound.midi.MidiSystem;
import
javax.sound.midi.InvalidMidiDataException;
import    javax.sound.midi.Sequence;
import    javax.sound.midi.Track;
import    javax.sound.midi.MidiEvent;
import    javax.sound.midi.MidiMessage;
import    javax.sound.midi.ShortMessage;
import    javax.sound.midi.MetaMessage;
import    javax.sound.midi.SysexMessage;
import    javax.sound.midi.Receiver;



/**    Displays the file format information of
a MIDI file.
 */
public class DumpReceiver
    implements    Receiver
{

    public static long seByteCount = 0;
    public static long smByteCount = 0;
    public static long seCount = 0;
```

```
    public static long smCount = 0;

    private static final String[]
sm_astrKeyNames = {"C", "C#", "D", "D#", "E",
"F", "F#", "G", "G#", "A", "A#", "B"};

    private static final String[]
sm_astrKeySignatures = {"Cb", "Gb", "Db", "Ab",
"Eb", "Bb", "F", "C", "G", "D", "A", "E", "B",
"F#", "C#"};
    private static final String[]
SYSTEM_MESSAGE_TEXT =
    {
        "System Exclusive (should not be in
ShortMessage!)",
        "MTC Quarter Frame: ",
        "Song Position: ",
        "Song Select: ",
        "Undefined",
        "Undefined",
        "Tune Request",
        "End of SysEx (should not be in
ShortMessage!)",
        "Timing clock",
        "Undefined",
        "Start",
        "Continue",
```

```
        "Stop",
        "Undefined",
        "Active Sensing",
        "System Reset"
    };

    private static final String[]
QUARTER_FRAME_MESSAGE_TEXT =
    {
        "frame count LS: ",
        "frame count MS: ",
        "seconds count LS: ",
        "seconds count MS: ",
        "minutes count LS: ",
        "minutes count MS: ",
        "hours count LS: ",
        "hours count MS: "
    };

    private static final String[]
FRAME_TYPE_TEXT =
    {
        "24 frames/second",
        "25 frames/second",
        "30 frames/second (drop)",
        "30 frames/second (non-drop)",
    };
```

```
    private PrintStream        m_printStream;
    private boolean            m_bDebug;
    private boolean
m_bPrintTimeStampAsTicks;



    public DumpReceiver(PrintStream
printStream)
    {
        this(printStream, false);
    }


    public DumpReceiver(PrintStream
printStream,
                boolean bPrintTimeStampAsTicks)
    {
        m_printStream = printStream;
        m_bDebug = false;
        m_bPrintTimeStampAsTicks =
bPrintTimeStampAsTicks;
    }
```

```
    public void close()
    {
    }



    public void send(MidiMessage message, long
lTimeStamp)
    {
        String    strMessage = null;
        if (message instanceof ShortMessage)
        {
            strMessage =
decodeMessage((ShortMessage) message);
        }
        else if (message instanceof
SysexMessage)
        {
            strMessage =
decodeMessage((SysexMessage) message);
        }
        else if (message instanceof
MetaMessage)
        {
            strMessage =
decodeMessage((MetaMessage) message);
        }
```

```
        else
        {
            strMessage = "unknown message
type";
        }
        String    strTimeStamp = null;
        if (m_bPrintTimeStampAsTicks)
        {
            strTimeStamp = "tick " + lTimeStamp
+ ": ";
        }
        else
        {
            if (lTimeStamp == -1L)
            {
                strTimeStamp = "timestamp
[unknown]: ";
            }
            else
            {
                strTimeStamp = "timestamp " +
lTimeStamp + " us: ";
            }
        }
        m_printStream.println(strTimeStamp +
strMessage);
    }
```

```
    public String decodeMessage(ShortMessage
message)
    {
        String    strMessage = null;
        switch (message.getCommand())
        {
        case 0x80:
            strMessage = "note Off " +
getKeyName(message.getData1()) + " velocity: "
+ message.getData2();
            break;

        case 0x90:
            strMessage = "note On " +
getKeyName(message.getData1()) + " velocity: "
+ message.getData2();
            break;

        case 0xa0:
            strMessage = "polyphonic key
pressure " + getKeyName(message.getData1()) + "
pressure: " + message.getData2();
            break;
```

```
        case 0xb0:
            strMessage = "control change " +
message.getData1() + " value: " +
message.getData2();
            break;

        case 0xc0:
            strMessage = "program change " +
message.getData1();
            break;

        case 0xd0:
            strMessage = "key pressure " +
getKeyName(message.getData1()) + " pressure: "
+ message.getData2();
            break;

        case 0xe0:
            strMessage = "pitch wheel change "
+ get14bitValue(message.getData1(),
message.getData2());
            break;

        case 0xF0:
            strMessage =
SYSTEM_MESSAGE_TEXT[message.getChannel()];
            switch (message.getChannel())
```

```
            {
            case 0x1:
                int    nQType =
(message.getData1() & 0x70) >> 4;
                int    nQData =
message.getData1() & 0x0F;
                if (nQType == 7)
                {
                    nQData = nQData & 0x1;
                }
                strMessage +=
QUARTER_FRAME_MESSAGE_TEXT[nQType] + nQData;
                if (nQType == 7)
                {
                    int    nFrameType =
(message.getData1() & 0x06) >> 1;
                    strMessage += ", frame
type: " + FRAME_TYPE_TEXT[nFrameType];
                }
                break;

            case 0x2:
                strMessage +=
get14bitValue(message.getData1(),
message.getData2());
                break;
```

```
            case 0x3:
                strMessage +=
message.getData1();
                break;
            }
            break;

        default:
            strMessage = "unknown message:
status = " + message.getStatus() + ", byte1 = "
+ message.getData1() + ", byte2 = " +
message.getData2();
            break;
        }
        if (message.getCommand() != 0xF0)
        {
            int    nChannel =
message.getChannel() + 1;
            String    strChannel = "channel " +
nChannel + ": ";
            strMessage = strChannel +
strMessage;
        }
        smCount++;
        smByteCount+=message.getLength();
        return "["+getHexString(message)+"]
"+strMessage;
```

```
    }



    public String decodeMessage(SysexMessage
message)
    {
        byte[]    abData = message.getData();
        String    strMessage = null;
        // System.out.println("sysex status: "
+ message.getStatus());
        if (message.getStatus() ==
SysexMessage.SYSTEM_EXCLUSIVE)
        {
            strMessage = "Sysex message: F0" +
getHexString(abData);
        }
        else if (message.getStatus() ==
SysexMessage.SPECIAL_SYSTEM_EXCLUSIVE)
        {
            strMessage = "Continued Sysex
message F7" + getHexString(abData);
            seByteCount--; // do not count the
F7
        }
        seByteCount += abData.length + 1;
        seCount++; // for the status byte
```

```
        return strMessage;
    }



    public String decodeMessage(MetaMessage
message)
    {
        byte[]    abMessage =
message.getMessage();
        byte[]    abData = message.getData();
        int    nDataLength =
message.getLength();
        String    strMessage = null;
        //System.out.println("data array
length: " + abData.length);
        switch (message.getType())
        {
        case 0:
                int    nSequenceNumber;
                if (abData.length == 0)
                nSequenceNumber = 0;
                else
                nSequenceNumber = ((abData[0] &
0xFF) << 8) | (abData[1] & 0xFF);
            strMessage = "Sequence Number: " +
nSequenceNumber;
```

```
            break;

        case 1:
            String    strText = new
String(abData);
            strMessage = "Text Event: " +
strText;
            break;

        case 2:
            String    strCopyrightText = new
String(abData);
            strMessage = "Copyright Notice: " +
strCopyrightText;
            break;

        case 3:
            String    strTrackName = new
String(abData);
            strMessage = "Sequence/Track Name:
" +  strTrackName;
            break;

        case 4:
            String    strInstrumentName = new
String(abData);
            strMessage = "Instrument Name: " +
```

```
strInstrumentName;
            break;

        case 5:
            String    strLyrics = new
String(abData);
            if (strLyrics.equals("\r\n"))
                strLyrics = "\\n";
            strMessage = "Lyric: " + strLyrics;
            break;

        case 6:
            String    strMarkerText = new
String(abData);
            strMessage = "Marker: " +
strMarkerText;
            break;

        case 7:
            String    strCuePointText = new
String(abData);
            strMessage = "Cue Point: " +
strCuePointText;
            break;

        case 0x20:
            int    nChannelPrefix = abData[0] &
```

```
0xFF;
            strMessage = "MIDI Channel Prefix:
" + nChannelPrefix;
            break;

        case 0x2F:
            strMessage = "End of Track";
            break;

        case 0x51:
            int    nTempo = ((abData[0] & 0xFF)
<< 16)
                    | ((abData[1] & 0xFF) << 8)
                    | (abData[2] & 0xFF);
// tempo in microseconds per beat
            float bpm = convertTempo(nTempo);
            // truncate it to 2 digits after
dot
            bpm = (float)
(Math.round(bpm*100.0f)/100.0f);
            strMessage = "Set Tempo: "+bpm+"
bpm";
            break;

        case 0x54:
            // System.out.println("data array
length: " + abData.length);
```

```
            strMessage = "SMTPE Offset: "
                + (abData[0] & 0xFF) + ":"
                + (abData[1] & 0xFF) + ":"
                + (abData[2] & 0xFF) + "."
                + (abData[3] & 0xFF) + "."
                + (abData[4] & 0xFF);
            break;

        case 0x58:
            strMessage = "Time Signature: "
                + (abData[0] & 0xFF) + "/" + (1
<< (abData[1] & 0xFF))
                + ", MIDI clocks per metronome
tick: " + (abData[2] & 0xFF)
                + ", 1/32 per 24 MIDI clocks: "
+ (abData[3] & 0xFF);
            break;

        case 0x59:
            String    strGender = (abData[1] ==
1) ? "minor" : "major";
            strMessage = "Key Signature: " +
sm_astrKeySignatures[abData[0] + 7] + " " +
strGender;
            break;

        case 0x7F:
```

```
            // TODO: decode vendor code, dump
data in rows
            String    strDataDump =
getHexString(abData);
            strMessage = "Sequencer-Specific
Meta event: " + strDataDump;
            break;

        default:
            String    strUnknownDump =
getHexString(abData);
            strMessage = "unknown Meta event: "
+ strUnknownDump;
            break;

        }
        return strMessage;
    }



    public static String getKeyName(int
nKeyNumber)
    {
        if (nKeyNumber > 127)
        {
            return "illegal value";
```

```
        }
        else
        {
            int    nNote = nKeyNumber % 12;
            int    nOctave = nKeyNumber / 12;
            return sm_astrKeyNames[nNote] +
(nOctave - 1);
        }
    }


    public static int get14bitValue(int
nLowerPart, int nHigherPart)
    {
        return (nLowerPart & 0x7F) |
((nHigherPart & 0x7F) << 7);
    }



    private static int
signedByteToUnsigned(byte b)
    {
        return b & 0xFF;
    }

    // convert from microseconds per quarter
```

```
note to beats per minute and vice versa
    private static float convertTempo(float
value) {
        if (value <= 0) {
            value = 0.1f;
        }
        return 60000000.0f / value;
    }




    private static char hexDigits[] =
       {'0', '1', '2', '3',
        '4', '5', '6', '7',
        '8', '9', 'A', 'B',
        'C', 'D', 'E', 'F'};

    public static String getHexString(byte[]
aByte)
    {
        StringBuffer    sbuf = new
StringBuffer(aByte.length * 3 + 2);
        for (int i = 0; i < aByte.length; i++)
        {
            sbuf.append(' ');
            sbuf.append(hexDigits[(aByte[i] &
0xF0) >> 4]);
```

```
            sbuf.append(hexDigits[aByte[i] &
0x0F]);
            /*byte    bhigh = (byte) ((aByte[i]
&  0xf0) >> 4);
            sbuf.append((char) (bhigh > 9 ?
bhigh + 'A' - 10: bhigh + '0'));
            byte    blow = (byte) (aByte[i] &
0x0f);
            sbuf.append((char) (blow > 9 ? blow
+ 'A' - 10: blow + '0'));*/
        }
        return new String(sbuf);
    }

    private static String intToHex(int i) {
        return ""+hexDigits[(i & 0xF0) >> 4]
                 +hexDigits[i & 0x0F];
    }

    public static String
getHexString(ShortMessage sm)
    {
        // bug in J2SDK 1.4.1
        // return
getHexString(sm.getMessage());
        int status = sm.getStatus();
        String res = intToHex(sm.getStatus());
```

```
        // if one-byte message, return
        switch (status) {
            case 0xF6:              // Tune
Request
            case 0xF7:              // EOX
                // System real-time messages
            case 0xF8:              // Timing
Clock
            case 0xF9:              // Undefined
            case 0xFA:              // Start
            case 0xFB:              // Continue
            case 0xFC:              // Stop
            case 0xFD:              // Undefined
            case 0xFE:              // Active
Sensing
            case 0xFF: return res;
        }
        res += ' '+intToHex(sm.getData1());
        // if 2-byte message, return
        switch (status) {
            case 0xF1:              // MTC
Quarter Frame
            case 0xF3:              // Song
Select
                    return res;
        }
        switch (sm.getCommand()) {
```

```
            case 0xC0:
            case 0xD0:
                    return res;
        }
        // 3-byte messages left
        res += ' '+intToHex(sm.getData2());
        return res;
    }
}



/*** DumpReceiver.java ***/
```

There are several sites with legal free MIDI files. The file Amy Winehouse - Rehab gives the result

```
-----------------------------------------------
----------------------------
File: rehab.mid
-----------------------------------------------
----------------------------
Length: 251475 ticks
Duration: 216788738 microseconds
-----------------------------------------------
----------------------------
DivisionType: PPQ
Resolution: 480 ticks per beat
-----------------------------------------------
----------------------------
Track 0:
-----------------------
tick 0: Time Signature: 4/4, MIDI clocks per
metronome tick: 24, 1/32 per 24 MIDI clocks: 8
tick 0: Key Signature: C major
tick 0: SMTPE Offset: 32:0:0.0.0
tick 0: Set Tempo: 145.0 bpm
tick 0: End of Track
-----------------------------------------------
----------------------------
Track 1:
-----------------------
tick 0: Sequence/Track Name: amy winehouse -
```

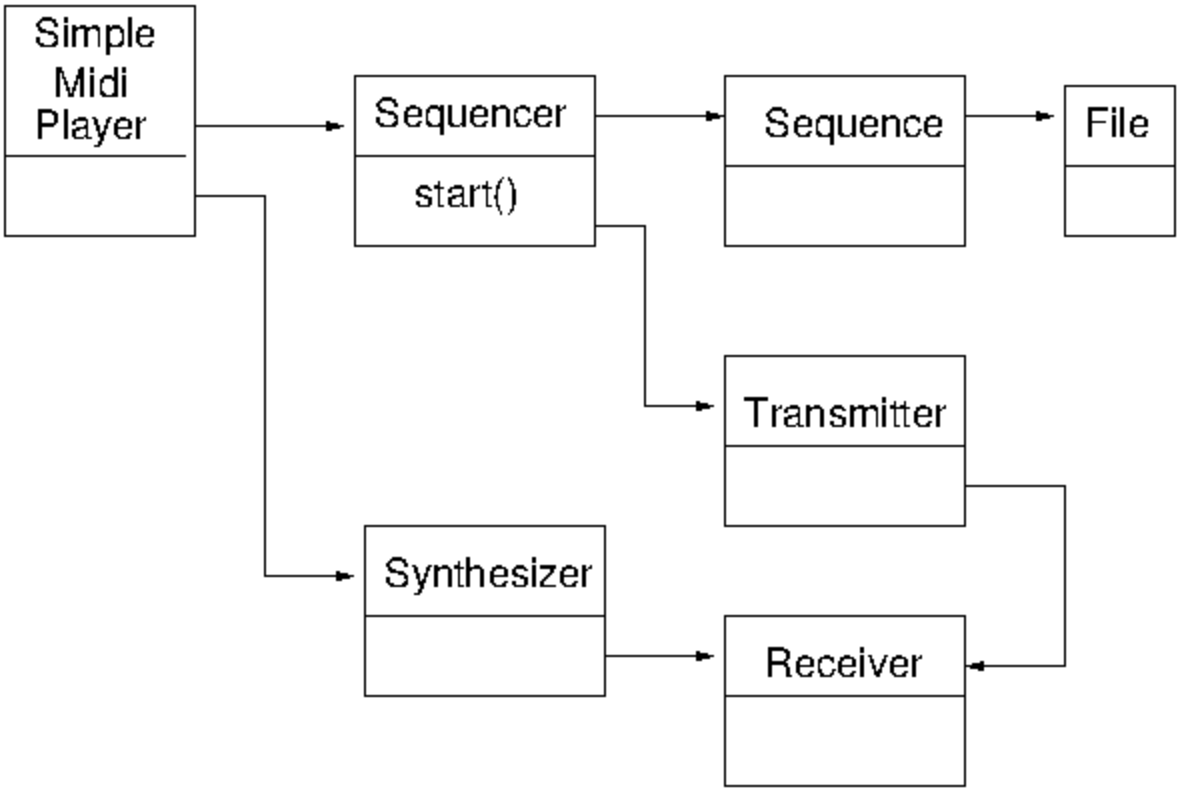
```
rehab
tick 0: Instrument Name: GM Device
tick 40: Sysex message: F0 7E 7F 09 01 F7
tick 40: End of Track
-----------------------------------------------
----------------------------
Track 2:
------------------------
tick 0: MIDI Channel Prefix: 1
tick 0: Sequence/Track Name: amy winehouse -
rehab
tick 0: Instrument Name: GM Device  2
tick 480: [B1 79 00] channel 2: control change
121 value: 0
tick 485: [B1 0A 40] channel 2: control change
10 value: 64
tick 490: [B1 5D 14] channel 2: control change
93 value: 20
tick 495: [B1 5B 00] channel 2: control change
91 value: 0
tick 500: [B1 0B 7F] channel 2: control change
11 value: 127
tick 505: [B1 07 69] channel 2: control change
7 value: 105
tick 510: [E1 00 40] channel 2: pitch wheel
change 8192
tick 515: [B1 00 00] channel 2: control change
```

```
0 value: 0
tick 520: [C1 22] channel 2: program change 34
...
```

Playing a MIDI file

To play a MIDI file, you create a `Sequence` from a `File`, using the `MidiSystem`. You also create a `Sequencer` from the `MidiSystem` and pass it the sequence. The sequencer will output MIDI messages through its `Transmitter`. This completes setup of the MIDI event generation side of the system.

The play side is constructed by getting a `Synthesizer` from the `MidiSystem`. The `Receiver` is found from the synthesizer and is given to the transmitter of MIDI events. Play commences by calling `start()` on the sequencer, which reads from the file and passes MIDI events to its transmitter. These are passed to the synthesizer's receiver and played. The UML class diagram for the relevant classes is



This code is from Playing an audio file (easy)

```
/*
 *    SimpleMidiPlayer.java
 *
 *    This file is part of jsresources.org
 */

/*
 * Copyright (c) 1999 - 2001 by Matthias
Pfisterer
 * All rights reserved.
 *
 * Redistribution and use in source and binary
forms, with or without
 * modification, are permitted provided that
the following conditions
 * are met:
 *
 * - Redistributions of source code must retain
the above copyright notice,
 *   this list of conditions and the following
disclaimer.
 * - Redistributions in binary form must
reproduce the above copyright
 *   notice, this list of conditions and the
following disclaimer in the
 *   documentation and/or other materials
```

```
provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN
NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

```
/*
|<---            this code is formatted to fit
into 80 columns             --->|
*/

import java.io.File;
import java.io.IOException;

import
javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.MidiSystem;
import
javax.sound.midi.MidiUnavailableException;
import javax.sound.midi.MetaEventListener;
import javax.sound.midi.MetaMessage;
import javax.sound.midi.Sequence;
import javax.sound.midi.Sequencer;
import javax.sound.midi.Synthesizer;
import javax.sound.midi.Receiver;
import javax.sound.midi.Transmitter;
import javax.sound.midi.MidiChannel;
import javax.sound.midi.ShortMessage;




/**
```

```
<titleabbrev>SimpleMidiPlayer</titleabbrev>
    <title>Playing a MIDI file (easy)</title>

    <formalpara><title>Purpose</title>
    <para>Plays a single MIDI file.</para>
</formalpara>

    <formalpara><title>Usage</title>
    <para>
    <cmdsynopsis>
    <command>java SimpleMidiPlayer</command>
    <arg choice="plain">
<replaceable>midifile</replaceable></arg>
    </cmdsynopsis>
    </para></formalpara>

    <formalpara><title>Parameters</title>
    <variablelist>
    <varlistentry>
    <term><option>
<replaceable>midifile</replaceable></option>
</term>
    <listitem><para>the name of the MIDI file
that should be
    played</para></listitem>
    </varlistentry>
    </variablelist>
```

```
    </formalpara>

    <formalpara><title>Bugs,
limitations</title>

    <para>This program always uses the default
Sequencer and the default
    Synthesizer to play on. For using non-
default sequencers,
    synthesizers or to play on an external MIDI
port, see
    <olink targetdoc="MidiPlayer"
    targetptr="MidiPlayer">MidiPlayer</olink>.
</para>
    </formalpara>

    <formalpara><title>Source code</title>
    <para>
    <ulink
url="SimpleMidiPlayer.java.html">SimpleMidiPlayer

    </para>
    </formalpara>

*/
public class SimpleMidiPlayer
{
```

```
    /*
      These variables are not really intended
to be static in a
      meaning of (good) design. They are used
by inner classes, so they
      can't just be automatic variables. There
were three possibilities:

      a) make them instance variables and
instantiate the object they
      belong to. This is clean (and is how you
should do it in a real
      application), but would have made the
example more complex.

      b) make them automatic final variables
inside main(). Design-wise,
      this is better than static, but automatic
final is something that
      is still like some black magic to me.

      c) make them static variables, as it is
done here. This is quite bad
      design, because if you have global
variables, you can't easily do
      the thing they are used for two times in
concurrency without risking
```

```
      indeterministic behaviour. However, it
makes the example easy to
      read.
     */
    private static Sequencer    sm_sequencer =
null;
    private static Synthesizer
sm_synthesizer = null;



    public static void main(String[] args)
    {
        /*
         *    We check if there is no command-
line argument at all
         *    or the first one is '-h'.
         *    If so, we display the usage
message and
         *    exit.
         */
        if (args.length == 0 ||
args[0].equals("-h"))
        {
            printUsageAndExit();
        }
```

```
        String    strFilename = args[0];
        File    midiFile = new
File(strFilename);

        /*
         *    We read in the MIDI file to a
Sequence object.
         *    This object is set at the
Sequencer later.
         */
        Sequence    sequence = null;
        try
        {
            sequence =
MidiSystem.getSequence(midiFile);
        }
        catch (InvalidMidiDataException e)
        {
            /*
             *    In case of an exception, we
dump the exception
             *    including the stack trace to
the console.
             *    Then, we exit the program.
             */
            e.printStackTrace();
            System.exit(1);
```

```
        }
        catch (IOException e)
        {
            /*
             *    In case of an exception, we
dump the exception
             *    including the stack trace to
the console.
             *    Then, we exit the program.
             */
            e.printStackTrace();
            System.exit(1);
        }

        /*
         *    Now, we need a Sequencer to play
the sequence.
         *    Here, we simply request the
default sequencer.
         *      With an argument of false, it
does not create
         *      a default syntesizer. With an
argument of true
         *      it is already linked to the
default synthesizer
         */
        try
```

```
        {
            sm_sequencer =
MidiSystem.getSequencer(false);
        }
        catch (MidiUnavailableException e)
        {
            e.printStackTrace();
            System.exit(1);
        }
        if (sm_sequencer == null)
        {
            out("SimpleMidiPlayer.main(): can't
get a Sequencer");
            System.exit(1);
        }

        /*
         *    There is a bug in the Sun
jdk1.3/1.4.
         *    It prevents correct termination
of the VM.
         *    So we have to exit ourselves.
         *    To accomplish this, we register a
Listener to the Sequencer.
         *    It is called when there are
"meta" events. Meta event
         *    47 is end of track.
```

```
         *
         *    Thanks to Espen Riskedal for
finding this trick.
         */
        sm_sequencer.addMetaEventListener(new
MetaEventListener()
            {
                public void meta(MetaMessage
event)
                {
                    if (event.getType() == 47)
                    {
                        sm_sequencer.close();
                        if (sm_synthesizer !=
null)
                        {

sm_synthesizer.close();
                        }
                        System.exit(0);
                    }
                }
            });

        /*
         *    The Sequencer is still a dead
object.
```

```
         *    We have to open() it to become
live.
         *    This is necessary to allocate
some ressources in
         *    the native part.
         */
        try
        {
            sm_sequencer.open();
        }
        catch (MidiUnavailableException e)
        {
            e.printStackTrace();
            System.exit(1);
        }

        /*
         *    Next step is to tell the
Sequencer which
         *    Sequence it has to play. In this
case, we
         *    set it as the Sequence object
created above.
         */
        try
        {
            sm_sequencer.setSequence(sequence);
```

```
        }
        catch (InvalidMidiDataException e)
        {
            e.printStackTrace();
            System.exit(1);
        }

        /*
         *    Now, we set up the destinations
the Sequence should be
         *    played on. Here, we try to use
the default
         *    synthesizer. With some Java Sound
implementations
         *    (Sun jdk1.3/1.4 and others
derived from this codebase),
         *    the default sequencer and the
default synthesizer
         *    are combined in one object. We
test for this
         *    condition, and if it's true,
nothing more has to
         *    be done. With other
implementations (namely Tritonus),
         *    sequencers and synthesizers are
always seperate
         *    objects. In this case, we have to
```

```
set up a link
         *    between the two objects manually.
         *
         *    By the way, you should never rely
on sequencers
         *    being synthesizers, too; this is
a highly non-
         *    portable programming style. You
should be able to
         *    rely on the other case working.
Alas, it is only
         *    partly true for the Sun
jdk1.3/1.4.
         */
        Receiver    synthReceiver = null;
        if (! (sm_sequencer instanceof
Synthesizer))
        {
            /*
             *    We try to get the default
synthesizer, open()
             *    it and chain it to the
sequencer with a
             *    Transmitter-Receiver pair.
             */
            try
            {
```

```
                sm_synthesizer =
MidiSystem.getSynthesizer();
                sm_synthesizer.open();
                    synthReceiver =
sm_synthesizer.getReceiver();
                Transmitter    seqTransmitter =
sm_sequencer.getTransmitter();

seqTransmitter.setReceiver(synthReceiver);
            }
            catch (MidiUnavailableException e)
            {
                e.printStackTrace();
            }
        }

        /*
         *    Now, we can start over.
         */
        sm_sequencer.start();

    try {
        Thread.sleep(5000);
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
```

```
    MidiChannel[] channels =
sm_synthesizer.getChannels();
    System.out.println("Num channels is " +
channels.length);
    for (int i = 0; i < channels.length; i++) {
        channels[i].controlChange(7, 1);
    }

    //    Receiver synthReceiver = null;
    try {
        synthReceiver =
MidiSystem.getReceiver();
    } catch (Exception e) {
    }
    ShortMessage volMessage = new
ShortMessage();
    int midiVolume = 100;
    for (Receiver rec:
sm_synthesizer.getReceivers()) {
        System.out.println("Setting vol on
recveiver " + rec.toString());
    for (int i = 0; i < channels.length; i++) {
        try {
        //
volMessage.setMessage(ShortMessage.CONTROL_CHANGE
 i, 123, midiVolume);
```

```
volMessage.setMessage(ShortMessage.CONTROL_CHANGE
 i, 7, midiVolume);
        } catch (InvalidMidiDataException e) {}
        synthReceiver.send(volMessage, -1);
        rec.send(volMessage, -1);
    }
    }



    }



    private static void printUsageAndExit()
    {
        out("SimpleMidiPlayer: usage:");
        out("\tjava SimpleMidiPlayer
<midifile>");
        System.exit(1);
    }



    private static void out(String strMessage)
    {
        System.out.println(strMessage);
    }
```

```
}



/*** SimpleMidiPlayer.java ***/
```

Playing a file to an external MIDI synthesizer

I have an Edirol Studio Canvas SD-20 synthesizer which I bought for a few hundred Australian dollars. This plugs into a PC through a USB port. Alsa recognises this by

```
$ amidi -l
Dir Device    Name
IO  hw:2,0,0  SD-20 Part A
IO  hw:2,0,1  SD-20 Part B
I   hw:2,0,2  SD-20 MIDI
```

The list of `MidiDevice.Info` device information lists `hw:2,0,0` twice, once for input and once for output, and similarly for the other values. The device information can be identified by the `toString` method, which returns values such as `"SD20 [hw:2,0,0]"`. From the device information the device can be found as before using `MidiSystem.getMidiDevice(info)`. The input and output devices can be distinguished by the number of

`maxOutputReceivers` it supports: zero means none, while any other value (including minus one!) means it has a MIDI receiver. Selecting an external receiver is done by code like

```
Receiver    synthReceiver = null;
        MidiDevice.Info[] devices;
        devices =
MidiSystem.getMidiDeviceInfo();

        for (MidiDevice.Info info: devices) {
            System.out.println("    Name: " +
info.toString() +
                        ", Decription: " +
                        info.getDescription() +
                        ", Vendor: " +
                        info.getVendor());
            if (info.toString().equals("SD20
[hw:2,0,0]")) {
            MidiDevice device =
MidiSystem.getMidiDevice(info);
            if (device.getMaxReceivers() != 0)
{
                try {
                device.open();
                System.out.println("  max
receivers: " + device.getMaxReceivers());
                receiver =
device.getReceiver();
                System.out.println("Found a
receiver");
```

```
                    break;
                    } catch(Exception e) {}
                }
                }
            }
```

Playing an audio file to my SD-20 is done by

```
/*
 *    ExternalMidiPlayer.java
 *
 *    This file adapted from SimpleMidiPlayer
of jsresources.org
 */

/*
 * Copyright (c) 1999 - 2001 by Matthias
Pfisterer
 * Copyright (c) 2015 Jan Newmarch
 * All rights reserved.
 *
 * Redistribution and use in source and binary
forms, with or without
 * modification, are permitted provided that
the following conditions
 * are met:
 *
 * - Redistributions of source code must retain
the above copyright notice,
 *   this list of conditions and the following
disclaimer.
 * - Redistributions in binary form must
reproduce the above copyright
 *   notice, this list of conditions and the
```

```
following disclaimer in the
 *   documentation and/or other materials
provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN
NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
```

```
 */

/*
|<---            this code is formatted to fit
into 80 columns             --->|
*/

import java.io.File;
import java.io.IOException;

import
javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.MidiSystem;
import
javax.sound.midi.MidiUnavailableException;
import javax.sound.midi.MetaEventListener;
import javax.sound.midi.MetaMessage;
import javax.sound.midi.Sequence;
import javax.sound.midi.Sequencer;
import javax.sound.midi.Synthesizer;
import javax.sound.midi.Receiver;
import javax.sound.midi.Transmitter;
import javax.sound.midi.MidiChannel;
import javax.sound.midi.ShortMessage;
import javax.sound.midi.MidiDevice;
import javax.sound.midi.MidiDeviceReceiver;
```

```
/**
<titleabbrev>SimpleMidiPlayer</titleabbrev>
    <title>Playing a MIDI file (easy)</title>

    <formalpara><title>Purpose</title>
    <para>Plays a single MIDI file.</para>
</formalpara>

    <formalpara><title>Usage</title>
    <para>
    <cmdsynopsis>
    <command>java SimpleMidiPlayer</command>
    <arg choice="plain">
<replaceable>midifile</replaceable></arg>
    </cmdsynopsis>
    </para></formalpara>

    <formalpara><title>Parameters</title>
    <variablelist>
    <varlistentry>
    <term><option>
<replaceable>midifile</replaceable></option>
</term>
    <listitem><para>the name of the MIDI file
that should be
```

```
    played</para></listitem>
    </varlistentry>
    </variablelist>
    </formalpara>

    <formalpara><title>Bugs,
limitations</title>

    <para>This program always uses the default
Sequencer and the default
    Synthesizer to play on. For using non-
default sequencers,
    synthesizers or to play on an external MIDI
port, see
    <olink targetdoc="MidiPlayer"
    targetptr="MidiPlayer">MidiPlayer</olink>.
</para>
    </formalpara>

    <formalpara><title>Source code</title>
    <para>
    <ulink
url="SimpleMidiPlayer.java.html">SimpleMidiPlayer

    </para>
    </formalpara>
```

```
*/
public class ExternalMidiPlayer
{
    /*
      These variables are not really intended
to be static in a
      meaning of (good) design. They are used
by inner classes, so they
      can't just be automatic variables. There
were three possibilities:

      a) make them instance variables and
instantiate the object they
      belong to. This is clean (and is how you
should do it in a real
      application), but would have made the
example more complex.

      b) make them automatic final variables
inside main(). Design-wise,
      this is better than static, but automatic
final is something that
      is still like some black magic to me.

      c) make them static variables, as it is
done here. This is quite bad
      design, because if you have global
```

```
variables, you can't easily do
      the thing they are used for two times in
concurrency without risking
      indeterministic behaviour. However, it
makes the example easy to
      read.
     */
    private static Sequencer    sm_sequencer =
null;
    private static Synthesizer
sm_synthesizer = null;
        private static Receiver receiver =
null;



    public static void main(String[] args)
throws MidiUnavailableException
    {
        /*
         *    We check if there is no command-
line argument at all
         *    or the first one is '-h'.
         *    If so, we display the usage
message and
         *    exit.
         */
        if (args.length == 0 ||
```

```
args[0].equals("-h"))
        {
            printUsageAndExit();
        }

        String    strFilename = args[0];
        File    midiFile = new
File(strFilename);

        /*
         *    We read in the MIDI file to a
Sequence object.
         *    This object is set at the
Sequencer later.
         */
        Sequence    sequence = null;
        try
        {
            sequence =
MidiSystem.getSequence(midiFile);
        }
        catch (InvalidMidiDataException e)
        {
            /*
             *    In case of an exception, we
dump the exception
             *    including the stack trace to
```

```
the console.
             *    Then, we exit the program.
             */
            e.printStackTrace();
            System.exit(1);
        }
        catch (IOException e)
        {
            /*
             *    In case of an exception, we
dump the exception
             *    including the stack trace to
the console.
             *    Then, we exit the program.
             */
            e.printStackTrace();
            System.exit(1);
        }

        /*
         *    Now, we need a Sequencer to play
the sequence.
         *    Here, we simply request the
default sequencer.
         *      With an argument of false, it
does not create
         *      a default syntesizer. With an
```

```
argument of true
         *      it is already linked to the
default synthesizer
         */
        try
        {
            sm_sequencer =
MidiSystem.getSequencer(false);
        }
        catch (MidiUnavailableException e)
        {
            e.printStackTrace();
            System.exit(1);
        }
        if (sm_sequencer == null)
        {
            out("SimpleMidiPlayer.main(): can't
get a Sequencer");
            System.exit(1);
        }

        /*
         *    There is a bug in the Sun
jdk1.3/1.4.
         *    It prevents correct termination
of the VM.
         *    So we have to exit ourselves.
```

```
         *    To accomplish this, we register a
Listener to the Sequencer.
         *    It is called when there are
"meta" events. Meta event
         *    47 is end of track.
         *
         *    Thanks to Espen Riskedal for
finding this trick.
         */
        sm_sequencer.addMetaEventListener(new
MetaEventListener()
            {
                public void meta(MetaMessage
event)
                {
                    if (event.getType() == 47)
                    {
                        sm_sequencer.close();
                        if (sm_synthesizer !=
null)
                        {

sm_synthesizer.close();
                        }
                        System.exit(0);
                    }
                }
```

```
            });

        /*
         *    The Sequencer is still a dead
object.
         *    We have to open() it to become
live.
         *    This is necessary to allocate
some ressources in
         *    the native part.
         */
        /*
        try
        {
            sm_sequencer.open();
        }
        catch (MidiUnavailableException e)
        {
            e.printStackTrace();
            System.exit(1);
        }
        */
        /*
         *    Next step is to tell the
Sequencer which
         *    Sequence it has to play. In this
case, we
```

```
         *    set it as the Sequence object
created above.
         */
        try
        {
            sm_sequencer.setSequence(sequence);
        }
        catch (InvalidMidiDataException e)
        {
            e.printStackTrace();
            System.exit(1);
        }

        /*
         *    Now, we set up the destinations
the Sequence should be
         *    played on.
         */
        Receiver    synthReceiver = null;
        MidiDevice.Info[] devices;
        devices =
MidiSystem.getMidiDeviceInfo();

        for (MidiDevice.Info info: devices) {
            System.out.println("    Name: " +
info.toString() +
                        ", Decription: " +
```

```
                        info.getDescription() +
                        ", Vendor: " +
                        info.getVendor());
            if (info.toString().equals("SD20
[hw:2,0,0]")) {
            MidiDevice device =
MidiSystem.getMidiDevice(info);
            if (device.getMaxReceivers() != 0)
{
                try {
                device.open();
                System.out.println("  max
receivers: " + device.getMaxReceivers());
                receiver =
device.getReceiver();
                System.out.println("Found a
receiver");
                break;
                } catch(Exception e) {}
            }
            }
        }

        if (receiver == null) {
            System.out.println("Reeciver is
null");
            System.exit(1);
```

```
        }
        try
            {
            Transmitter    seqTransmitter =
sm_sequencer.getTransmitter();

seqTransmitter.setReceiver(receiver);
            }
        catch (MidiUnavailableException e)
            {
            e.printStackTrace();
            }

        /*
         *    Now, we can start over.
         */
        sm_sequencer.open();
        sm_sequencer.start();

        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }
```

```
    private static void printUsageAndExit()
    {
        out("SimpleMidiPlayer: usage:");
        out("\tjava SimpleMidiPlayer
<midifile>");
        System.exit(1);
    }



    private static void out(String strMessage)
    {
        System.out.println(strMessage);
    }
}



/*** SimpleMidiPlayer.java ***/
```

Changing the soundbank

The soundbank is a set of "sounds" encoded in some way that are used to generate the music played. The default sound synthesizer for Java is the Gervill synthesizer, and this looks for its default soundbank in `$HOME/.gervill/soundbank-emg.sf2`. This default soundbank is tiny, only 1.9MBytes in size, and sounds, well, poor quality.

DaWicked1 in Better Java-midi instrument sounds for Linux offers two methods to improve this: the simpler is to replace the soundfont with a better one such as the Fluidsynth font, using the default name.

The second method is programmatic and probably better as it allows more flexibility and choice at runtime.

Changing pitch and speed

Changing the speed of playback of a MIDI file means changing the rate that MIDI messages are sent from the sequencer. The Java sequencers have methods to control this such as `setTempoFactor`. The sequencer will respond to this method by sending the messages at a different rate.

Changing the pitch of the notes can be done by altering the pitch of the `NOTE_ON` and `NOTE_OFF` messages. The sequencer does not do this as it is unrelated to time. The default MIDI `Receiver` just gets MIDI messages and passes it onto its `Synthesizer`. We can create our own `Receiver` and interpose it between the default `Transmitter` and `Receiver`. This can examine MIDI messages in transit and adjust the pitch of note on/off messages.

We look for input from the user of ←, ↑, →, ↓ (ESC-[A, etc). These then call the appropriate method. The program illustrating this is an adaptation of the `SimpleMidiPlayer` given earlier in the chapter and is AdaptableMidiPlayer.java :

Using TiMidity instead of the default Gervill synthesizer

The softsynth TiMidity can be run as a backend synthesizer using the Alsa sequencer by

```
$timidity -iA -B2,8 -Os -EFreverb=0

Opening sequencer port: 128:0 128:1 128:2 128:3
```

(and similarly for Fluidsynth.). This is opened on ports 128:0 etc.

Unfortunately this is not directly visible to JavaSound which expects either the default Gervill synthesizer or a raw MIDI synthesizer such as a hardware synthesizer. As discussed in the MIDI Alsa chapter, we can fix this by using Alsa raw MIDI ports.

We add raw MIDI ports by

```
modprobe snd-seq snd-virmidi
```

This will bring virtual devices both into the ALSA raw MIDI and into the ALSA sequencer spaces:

```
$amidi -l
Dir Device    Name
IO  hw:3,0    Virtual Raw MIDI (16 subdevices)
IO  hw:3,1    Virtual Raw MIDI (16 subdevices)
IO  hw:3,2    Virtual Raw MIDI (16 subdevices)
IO  hw:3,3    Virtual Raw MIDI (16 subdevices)

$aplaymidi -l
 Port    Client name                      Port
name
 14:0    Midi Through                     Midi
Through Port-0
 28:0    Virtual Raw MIDI 3-0
VirMIDI 3-0
 29:0    Virtual Raw MIDI 3-1
VirMIDI 3-1
 30:0    Virtual Raw MIDI 3-2
VirMIDI 3-2
 31:0    Virtual Raw MIDI 3-3
VirMIDI 3-3
```

Virtual Raw MIDI port 3-0 can then be connected to TiMidity port 0 by

```
aconnect 28:0 128:0
```

The final step in playing to TiMidity is to change one line of AdaptableMidiPlayer.java from

```
if (info.toString().equals("SD20 [hw:2,0,0]"))
{
```

to

```
if (info.toString().equals("VirMIDI
[hw:3,0,0]")) {
```

Conclusion

This chapter has built a number of programs using the MIDI API and discussed how to use external hardware synthesizers and soft sythesizers such as TiMidity.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

ALSA has some support for MIDI devices by a sequencer API. Clients can send MIDI events to the sequencer and it will play them according to the timing of the events. Other clients can then receive these sequenced events and, for example, synthesize them.

Resources

- ALSA Sequencer - design document
- ALSA Programming HOWTO has Writing a sequencer client, a MIDI router, combining PCM and MIDI: miniFMsynth and scheduling MIDI events: miniArp
- MIDI Sequencer API
- Sequencer interface
- ALSA Sequencer System is an in-depth view of the sequencer system by Takashi Iwai

Introduction

ALSA suplies a sequencer that can receive MIDI events from one set of clients and play them according to the timing information in the events to other clients. The clients that can send such events are file readers such as `aplaymidi` or other sequencers. Clients can also read events as they should be played. Possible consuming clients include splitters, routers or soft synthesizers such as Timidity.

Timidity can be run as ALSA sequencer client which will consume MIDI events and synthesize them. From The TiMidity Howto - Using TiMidity as the ALSA sequencer client

```
timidity -iA -B2,8 -Os -EFreverb=0
```

On my computer this produced

```
Requested buffer size 2048, fragment size 1024
ALSA pcm 'default' set buffer size 2048, period
size 680 bytes
TiMidity starting in ALSA server mode
Opening sequencer port: 129:0 129:1 129:2 129:3
```

and then sat there waiting for a connection to be made.

FluidSynth can also be used as a server (Ted's Linux MIDI Guide):

```
fluidsynth --server --audio-driver=alsa -C0 -R1
-l /usr/share/soundfonts/FluidR3_GM.sf2
```

The ALSA sequencer sends MIDI "wire" events. This does not include MIDI file events such as Text or Lyric Meta-events. This makes it pretty useless for a Karaoke player. It is possible to modify the file reader `aplaymid` to send Meta Events to, say, a listener (like the Java MetaEventListener), but as these come from the file reader rather than the sequencer they generally arrive well before the time they will get sequenced to be played. Pity.

Programs such as `pykaraoke` make use of the ALSA sequencer. However, in order to get the timing of the lyrics right it includes a MIDI file parser and basically acts as a second sequencer just to extract and display the Text/Lyric events.

aconnect

the program `aconnect.c` can be used to test for sequencer servers and clients such as sequencers. I have set two clients running: Timidity and seqdemo (discussed later). The command

```
aconnect -o
```

shows

```
client 14: 'Midi Through' [type=kernel]
    0 'Midi Through Port-0'
client 128: 'TiMidity' [type=user]
    0 'TiMidity port 0 '
    1 'TiMidity port 1 '
    2 'TiMidity port 2 '
    3 'TiMidity port 3 '
client 129: 'ALSA Sequencer Demo' [type=user]
    0 'ALSA Sequencer Demo'
```

When run with the -i option it produces

```
$aconnect -i
client 0: 'System' [type=kernel]
    0 'Timer           '
    1 'Announce        '
client 14: 'Midi Through' [type=kernel]
    0 'Midi Through Port-0'
```

The program `aconnect` can establish a connection between input and output clients by

```
aconnect in out
```

The code for aconnect.c is from SourceArchive

```
/*
 ** connect / disconnect two subscriber ports
 **   ver.0.1.3
 **
 ** Copyright (C) 1999 Takashi Iwai
 **
 **  This program is free software; you can
redistribute it and/or modify
 **  it under the terms of the GNU General
Public License version 2 as
 **  published by the Free Software Foundation.
 **
 **  This program is distributed in the hope
that it will be useful,
 **  but WITHOUT ANY WARRANTY; without even the
implied warranty of
 **  MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.  See the
 **  GNU General Public License for more
details.
 **
 **/

#include <stdio.h>
#include <ctype.h>
#include <string.h>
```

```c
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <getopt.h>
#include <stdarg.h>
#include <sys/ioctl.h>
#include <alsa/asoundlib.h>

static void error_handler(const char *file, int
line, const char *function, int err, const char
*fmt, ...)
{
	va_list arg;

	if (err == ENOENT)	/* Ignore those
misleading "warnings" */
		return;
	va_start(arg, fmt);
	fprintf(stderr, "ALSA lib %s:%i:(%s) ",
file, line, function);
	vfprintf(stderr, fmt, arg);
	if (err)
		fprintf(stderr, ": %s",
snd_strerror(err));
	putc('\n', stderr);
	va_end(arg);
}
```

```
static void usage(void)
{
	fprintf(stderr, "aconnect - ALSA
sequencer connection manager\n");
	fprintf(stderr, "Copyright (C) 1999-2000
Takashi Iwai\n");
	fprintf(stderr, "Usage:\n");
	fprintf(stderr, " *
Connection/disconnection between two ports\n");
	fprintf(stderr, "   aconnect [-options]
sender receiver\n");
	fprintf(stderr, "     sender, receiver =
client:port pair\n");
	fprintf(stderr, "     -d,--disconnect
disconnect\n");
	fprintf(stderr, "     -e,--exclusive
exclusive connection\n");
	fprintf(stderr, "     -r,--real #
convert real-time-stamp on queue\n");
	fprintf(stderr, "     -t,--tick #
convert tick-time-stamp on queue\n");
	fprintf(stderr, " * List connected ports
(no subscription action)\n");
	fprintf(stderr, "   aconnect -i|-o [-
options]\n");
	fprintf(stderr, "     -i,--input
```

```
list input (readable) ports\n");
      fprintf(stderr, "     -o,--output
list output (writable) ports\n");
      fprintf(stderr, "     -l,--list
list current connections of each port\n");
      fprintf(stderr, " * Remove all exported
connections\n");
      fprintf(stderr, "     -x, --
removeall\n");
}

/*
 *  * check permission (capability) of
specified port
 *   */

#define LIST_INPUT      1
#define LIST_OUTPUT     2

#define perm_ok(pinfo,bits)
((snd_seq_port_info_get_capability(pinfo) &
(bits)) == (bits))

static int check_permission(snd_seq_port_info_t
*pinfo, int perm)
{
      if (perm) {
```

```
            if (perm & LIST_INPUT) {
                  if (perm_ok(pinfo,
SND_SEQ_PORT_CAP_READ|SND_SEQ_PORT_CAP_SUBS_READ)

                        goto __ok;
            }
            if (perm & LIST_OUTPUT) {
                  if (perm_ok(pinfo,
SND_SEQ_PORT_CAP_WRITE|SND_SEQ_PORT_CAP_SUBS_WRIT

                        goto __ok;
            }
            return 0;
      }
 __ok:
      if
(snd_seq_port_info_get_capability(pinfo) &
SND_SEQ_PORT_CAP_NO_EXPORT)
            return 0;
      return 1;
}

/*
 *  * list subscribers of specified type
 *   */
static void list_each_subs(snd_seq_t *seq,
snd_seq_query_subscribe_t *subs, int type,
```

```
const char *msg)
{
	int count = 0;
	snd_seq_query_subscribe_set_type(subs,
type);
	snd_seq_query_subscribe_set_index(subs,
0);
	while
(snd_seq_query_port_subscribers(seq, subs) >=
0) {
		const snd_seq_addr_t *addr;
		if (count++ == 0)
			printf("\t%s: ", msg);
		else
			printf(", ");
		addr =
snd_seq_query_subscribe_get_addr(subs);
		printf("%d:%d", addr->client, addr-
>port);
		if
(snd_seq_query_subscribe_get_exclusive(subs))
			printf("[ex]");
		if
(snd_seq_query_subscribe_get_time_update(subs))
			printf("[%s:%d]",

(snd_seq_query_subscribe_get_time_real(subs) ?
```

```
"real" : "tick"),

snd_seq_query_subscribe_get_queue(subs));

snd_seq_query_subscribe_set_index(subs,
snd_seq_query_subscribe_get_index(subs) + 1);
      }
      if (count > 0)
            printf("\n");
}

/*
 *  * list subscribers
 *   */
static void list_subscribers(snd_seq_t *seq,
const snd_seq_addr_t *addr)
{
      snd_seq_query_subscribe_t *subs;
      snd_seq_query_subscribe_alloca(&subs);
      snd_seq_query_subscribe_set_root(subs,
addr);
      list_each_subs(seq, subs,
SND_SEQ_QUERY_SUBS_READ, "Connecting To");
      list_each_subs(seq, subs,
SND_SEQ_QUERY_SUBS_WRITE, "Connected From");
}
```

```
/*
 *  * search all ports
 *   */
typedef void (*action_func_t)(snd_seq_t *seq,
snd_seq_client_info_t *cinfo,
snd_seq_port_info_t *pinfo, int count);

static void do_search_port(snd_seq_t *seq, int
perm, action_func_t do_action)
{
      snd_seq_client_info_t *cinfo;
      snd_seq_port_info_t *pinfo;
      int count;

      snd_seq_client_info_alloca(&cinfo);
      snd_seq_port_info_alloca(&pinfo);
      snd_seq_client_info_set_client(cinfo,
-1);
      while (snd_seq_query_next_client(seq,
cinfo) >= 0) {
            /* reset query info */
            snd_seq_port_info_set_client(pinfo,
snd_seq_client_info_get_client(cinfo));
            snd_seq_port_info_set_port(pinfo,
-1);
            count = 0;
            while (snd_seq_query_next_port(seq,
```

```
pinfo) >= 0) {
                    if (check_permission(pinfo,
perm)) {
                          do_action(seq, cinfo,
pinfo, count);
                          count++;
                    }
              }
        }
}


static void print_port(snd_seq_t *seq,
snd_seq_client_info_t *cinfo,
                     snd_seq_port_info_t *pinfo,
int count)
{
        if (! count) {
              printf("client %d: '%s'
[type=%s]\n",

snd_seq_client_info_get_client(cinfo),

snd_seq_client_info_get_name(cinfo),

(snd_seq_client_info_get_type(cinfo) ==
SND_SEQ_USER_CLIENT ? "user" : "kernel"));
```

```
        }
        printf("  %3d '%-16s'\n",
               snd_seq_port_info_get_port(pinfo),

snd_seq_port_info_get_name(pinfo));
}

static void print_port_and_subs(snd_seq_t *seq,
snd_seq_client_info_t *cinfo,
                                snd_seq_port_info_t
*pinfo, int count)
{
        print_port(seq, cinfo, pinfo, count);
        list_subscribers(seq,
snd_seq_port_info_get_addr(pinfo));
}



/*
 *  * remove all (exported) connections
 *   */
static void remove_connection(snd_seq_t *seq,
snd_seq_client_info_t *cinfo,
                                snd_seq_port_info_t
*pinfo, int count)
{
        snd_seq_query_subscribe_t *query;
```

```
	snd_seq_query_subscribe_alloca(&query);
	snd_seq_query_subscribe_set_root(query,
snd_seq_port_info_get_addr(pinfo));

	snd_seq_query_subscribe_set_type(query,
SND_SEQ_QUERY_SUBS_READ);
	snd_seq_query_subscribe_set_index(query,
0);
	for (;
snd_seq_query_port_subscribers(seq, query) >=
0;

snd_seq_query_subscribe_set_index(query,
snd_seq_query_subscribe_get_index(query) + 1))
{
		snd_seq_port_info_t *port;
		snd_seq_port_subscribe_t *subs;
		const snd_seq_addr_t *sender =
snd_seq_query_subscribe_get_root(query);
		const snd_seq_addr_t *dest =
snd_seq_query_subscribe_get_addr(query);
		snd_seq_port_info_alloca(&port);
		if (snd_seq_get_any_port_info(seq,
dest->client, dest->port, port) < 0)
			continue;
		if (!
```

```
(snd_seq_port_info_get_capability(port) &
SND_SEQ_PORT_CAP_SUBS_WRITE))
                  continue;
            if
(snd_seq_port_info_get_capability(port) &
SND_SEQ_PORT_CAP_NO_EXPORT)
                  continue;

snd_seq_port_subscribe_alloca(&subs);

snd_seq_port_subscribe_set_queue(subs,
snd_seq_query_subscribe_get_queue(query));

snd_seq_port_subscribe_set_sender(subs,
sender);

snd_seq_port_subscribe_set_dest(subs, dest);
            snd_seq_unsubscribe_port(seq,
subs);
      }

      snd_seq_query_subscribe_set_type(query,
SND_SEQ_QUERY_SUBS_WRITE);
      snd_seq_query_subscribe_set_index(query,
0);
      for (;
snd_seq_query_port_subscribers(seq, query) >=
```

```
0;

snd_seq_query_subscribe_set_index(query,
snd_seq_query_subscribe_get_index(query) + 1))
{
            snd_seq_port_info_t *port;
            snd_seq_port_subscribe_t *subs;
            const snd_seq_addr_t *dest =
snd_seq_query_subscribe_get_root(query);
            const snd_seq_addr_t *sender =
snd_seq_query_subscribe_get_addr(query);
            snd_seq_port_info_alloca(&port);
            if (snd_seq_get_any_port_info(seq,
sender->client, sender->port, port) < 0)
                  continue;
            if (!
(snd_seq_port_info_get_capability(port) &
SND_SEQ_PORT_CAP_SUBS_READ))
                  continue;
            if
(snd_seq_port_info_get_capability(port) &
SND_SEQ_PORT_CAP_NO_EXPORT)
                  continue;

snd_seq_port_subscribe_alloca(&subs);

snd_seq_port_subscribe_set_queue(subs,
```

```
snd_seq_query_subscribe_get_queue(query));

snd_seq_port_subscribe_set_sender(subs,
sender);

snd_seq_port_subscribe_set_dest(subs, dest);
            snd_seq_unsubscribe_port(seq,
subs);
      }
}

static void remove_all_connections(snd_seq_t
*seq)
{
      do_search_port(seq, 0,
remove_connection);
}



/*
 *   * main..
 *    */

enum {
      SUBSCRIBE, UNSUBSCRIBE, LIST, REMOVE_ALL
};
```

```
static struct option long_option[] = {
	{"disconnect", 0, NULL, 'd'},
	{"input", 0, NULL, 'i'},
	{"output", 0, NULL, 'o'},
	{"real", 1, NULL, 'r'},
	{"tick", 1, NULL, 't'},
	{"exclusive", 0, NULL, 'e'},
	{"list", 0, NULL, 'l'},
	{"removeall", 0, NULL, 'x'},
	{NULL, 0, NULL, 0},
};

int main(int argc, char **argv)
{
	int c;
	snd_seq_t *seq;
	int queue = 0, convert_time = 0,
convert_real = 0, exclusive = 0;
	int command = SUBSCRIBE;
	int list_perm = 0;
	int client;
	int list_subs = 0;
	snd_seq_port_subscribe_t *subs;
	snd_seq_addr_t sender, dest;

	while ((c = getopt_long(argc, argv,
"dior:t:elx", long_option, NULL)) != -1) {
```

```
            switch (c) {
            case 'd':
                  command = UNSUBSCRIBE;
                  break;
            case 'i':
                  command = LIST;
                  list_perm |= LIST_INPUT;
                  break;
            case 'o':
                  command = LIST;
                  list_perm |= LIST_OUTPUT;
                  break;
            case 'e':
                  exclusive = 1;
                  break;
            case 'r':
                  queue = atoi(optarg);
                  convert_time = 1;
                  convert_real = 1;
                  break;
            case 't':
                  queue = atoi(optarg);
                  convert_time = 1;
                  convert_real = 0;
                  break;
            case 'l':
                  list_subs = 1;
```

```
                        break;
                case 'x':
                        command = REMOVE_ALL;
                        break;
                default:
                        usage();
                        exit(1);
                }
        }

        if (snd_seq_open(&seq, "default",
SND_SEQ_OPEN_DUPLEX, 0) < 0) {
                fprintf(stderr, "can't open
sequencer\n");
                return 1;
        }

        snd_lib_error_set_handler(error_handler);

        switch (command) {
        case LIST:
                do_search_port(seq, list_perm,
                                list_subs ?
print_port_and_subs : print_port);
                snd_seq_close(seq);
                return 0;
        case REMOVE_ALL:
```

```
            remove_all_connections(seq);
            snd_seq_close(seq);
            return 0;
      }

      /* connection or disconnection */

      if (optind + 2 > argc) {
            snd_seq_close(seq);
            usage();
            exit(1);
      }

      if ((client = snd_seq_client_id(seq)) <
0) {
            snd_seq_close(seq);
            fprintf(stderr, "can't get client
id\n");
            return 1;
      }

      /* set client info */
      if (snd_seq_set_client_name(seq, "ALSA
Connector") < 0) {
            snd_seq_close(seq);
            fprintf(stderr, "can't set client
info\n");
```

```
            return 1;
      }

      /* set subscription */
      if (snd_seq_parse_address(seq, &sender,
argv[optind]) < 0) {
            snd_seq_close(seq);
            fprintf(stderr, "invalid sender
address %s\n", argv[optind]);
            return 1;
      }
      if (snd_seq_parse_address(seq, &dest,
argv[optind + 1]) < 0) {
            snd_seq_close(seq);
            fprintf(stderr, "invalid
destination address %s\n", argv[optind + 1]);
            return 1;
      }
      snd_seq_port_subscribe_alloca(&subs);
      snd_seq_port_subscribe_set_sender(subs,
&sender);
      snd_seq_port_subscribe_set_dest(subs,
&dest);
      snd_seq_port_subscribe_set_queue(subs,
queue);

snd_seq_port_subscribe_set_exclusive(subs,
```

```
exclusive);

snd_seq_port_subscribe_set_time_update(subs,
convert_time);

snd_seq_port_subscribe_set_time_real(subs,
convert_real);

      if (command == UNSUBSCRIBE) {
            if
(snd_seq_get_port_subscription(seq, subs) < 0)
{
                  snd_seq_close(seq);
                  fprintf(stderr, "No
subscription is found\n");
                  return 1;
            }
            if (snd_seq_unsubscribe_port(seq,
subs) < 0) {
                  snd_seq_close(seq);
                  fprintf(stderr,
"Disconnection failed (%s)\n",
snd_strerror(errno));
                  return 1;
            }
      } else {
            if
```

```
(snd_seq_get_port_subscription(seq, subs) == 0)
{
                    snd_seq_close(seq);
                    fprintf(stderr, "Connection
is already subscribed\n");
                    return 1;
            }
            if (snd_seq_subscribe_port(seq,
subs) < 0) {
                    snd_seq_close(seq);
                    fprintf(stderr, "Connection
failed (%s)\n", snd_strerror(errno));
                    return 1;
            }
      }

      snd_seq_close(seq);

      return 0;
}
```

seqdemo

The code for seqdemo.c is

```
/* seqdemo.c by Matthias Nagorni */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <alsa/asoundlib.h>

snd_seq_t *open_seq();
void midi_action(snd_seq_t *seq_handle);

snd_seq_t *open_seq() {

  snd_seq_t *seq_handle;
  int portid;

  if (snd_seq_open(&seq_handle, "default",
SND_SEQ_OPEN_INPUT, 0) < 0) {
    fprintf(stderr, "Error opening ALSA
sequencer.\n");
    exit(1);
  }
  snd_seq_set_client_name(seq_handle, "ALSA
Sequencer Demo");
  if ((portid =
snd_seq_create_simple_port(seq_handle, "ALSA
Sequencer Demo",
```

```
SND_SEQ_PORT_CAP_WRITE|SND_SEQ_PORT_CAP_SUBS_WRIT

            SND_SEQ_PORT_TYPE_APPLICATION)) <
0) {
    fprintf(stderr, "Error creating sequencer
port.\n");
    exit(1);
  }
  return(seq_handle);
}

void midi_action(snd_seq_t *seq_handle) {

  snd_seq_event_t *ev;

  do {
    snd_seq_event_input(seq_handle, &ev);
    switch (ev->type) {
      case SND_SEQ_EVENT_CONTROLLER:
        fprintf(stderr, "Control event on
Channel %2d: %5d       \r",
                ev->data.control.channel, ev-
>data.control.value);
        break;
      case SND_SEQ_EVENT_PITCHBEND:
        fprintf(stderr, "Pitchbender event on
```

```
Channel %2d: %5d   \r",
                ev->data.control.channel, ev-
>data.control.value);
        break;
      case SND_SEQ_EVENT_NOTEON:
        fprintf(stderr, "Note On event on
Channel %2d: %5d       \r",
                ev->data.control.channel, ev-
>data.note.note);
        break;
      case SND_SEQ_EVENT_NOTEOFF:
        fprintf(stderr, "Note Off event on
Channel %2d: %5d      \r",
                ev->data.control.channel, ev-
>data.note.note);
        break;
    }
    snd_seq_free_event(ev);
  } while
(snd_seq_event_input_pending(seq_handle, 0) >
0);
}

int main(int argc, char *argv[]) {

  snd_seq_t *seq_handle;
  int npfd;
```

```
  struct pollfd *pfd;

  seq_handle = open_seq();
  npfd =
snd_seq_poll_descriptors_count(seq_handle,
POLLIN);
  pfd = (struct pollfd *)alloca(npfd *
sizeof(struct pollfd));
  snd_seq_poll_descriptors(seq_handle, pfd,
npfd, POLLIN);
  while (1) {
    if (poll(pfd, npfd, 100000) > 0) {
      midi_action(seq_handle);
    }
  }
}
```

aplaymidi

The program `aplaymidi` will play to a backend MIDI synthesizer such as `TiMidity`. It requires a port name, which can be found by

```
aplaymidi -l
```

with output such as

```
Port    Client name                      Port
name
 14:0    Midi Through                     Midi
Through Port-0
128:0    TiMidity
TiMidity port 0
128:1    TiMidity
TiMidity port 1
128:2    TiMidity
TiMidity port 2
128:3    TiMidity
TiMidity port 3
131:0    aseqdump
aseqdump
```

The port numbers are the same as those used by `aconnect`. These are not the Alsa device names (hw:0, etc) but are special to the Alsa sequencer API.

It can then play a MIDI file to one of these ports as in

```
aplaymidi -p 128:0 54154.mid
```

The code can be found from SourceArchive.com

Raw MIDI ports

From RawMidi interface

> [The] RawMidi Interface is designed to write or read raw (unchanged) MIDI data over the MIDI line without any timestamps defined in interface.

Raw MIDI phyical devices

The raw MIDI interface is typically used to manage hardware MIDI devices. For example, if I plug in an Edirol SD-20 synthesizer to a USB port, it shows under `amidi` as

```
$amidi -l
Dir Device    Name
IO  hw:2,0,0  SD-20 Part A
IO  hw:2,0,1  SD-20 Part B
I   hw:2,0,2  SD-20 MIDI
```

These names use the same pattern as Alsa playback and record devices of `hw:...` .

Raw MIDI virtual devices

The Linux kernel module `snd_virmidi` can create virtual raw MIDI devices. First add the modules (Using TiMidity++ with ALSA raw MIDI and AlsaMidiOverview)

```
modprobe snd-seq snd-virmidi
```

This will bring virtual devices both into the ALSA raw MIDI and into the ALSA sequencer spaces:

```
$amidi -l
Dir Device    Name
IO  hw:3,0    Virtual Raw MIDI (16 subdevices)
IO  hw:3,1    Virtual Raw MIDI (16 subdevices)
IO  hw:3,2    Virtual Raw MIDI (16 subdevices)
IO  hw:3,3    Virtual Raw MIDI (16 subdevices)

$aplaymidi -l
 Port    Client name                      Port
name
 14:0    Midi Through                     Midi
Through Port-0
 28:0    Virtual Raw MIDI 3-0
VirMIDI 3-0
 29:0    Virtual Raw MIDI 3-1
VirMIDI 3-1
 30:0    Virtual Raw MIDI 3-2
VirMIDI 3-2
 31:0    Virtual Raw MIDI 3-3
VirMIDI 3-3
```

Mapping MIDI clients into MIDI raw space

Some programs/APIs use the Alsa sequencer space; others use the Alsa raw MIDI space. Virtual ports allow a client using one space to use a client from a different space.

For example, TiMidity can be run as a sequencer client by

```
timidity -iA -B2,8 -Os -EFreverb=0
```

This only shows in the sequencer space, not in the raw MIDI space, and shows to `aconnect -o` as

```
$aconnect -o
client 14: 'Midi Through' [type=kernel]
    0 'Midi Through Port-0'
client 28: 'Virtual Raw MIDI 3-0' [type=kernel]
    0 'VirMIDI 3-0     '
client 29: 'Virtual Raw MIDI 3-1' [type=kernel]
    0 'VirMIDI 3-1     '
client 30: 'Virtual Raw MIDI 3-2' [type=kernel]
    0 'VirMIDI 3-2     '
client 31: 'Virtual Raw MIDI 3-3' [type=kernel]
    0 'VirMIDI 3-3     '
client 128: 'TiMidity' [type=user]
    0 'TiMidity port 0 '
    1 'TiMidity port 1 '
    2 'TiMidity port 2 '
    3 'TiMidity port 3 '
```

while `aconnect -i` shows the virtual ports as

```
$aconnect -i
client 0: 'System' [type=kernel]
    0 'Timer           '
    1 'Announce        '
client 14: 'Midi Through' [type=kernel]
    0 'Midi Through Port-0'
client 28: 'Virtual Raw MIDI 3-0' [type=kernel]
    0 'VirMIDI 3-0     '
client 29: 'Virtual Raw MIDI 3-1' [type=kernel]
    0 'VirMIDI 3-1     '
client 30: 'Virtual Raw MIDI 3-2' [type=kernel]
    0 'VirMIDI 3-2     '
client 31: 'Virtual Raw MIDI 3-3' [type=kernel]
    0 'VirMIDI 3-3     '
```

Virtual Raw MIDI 3-0 can then be connected to TiMidity port 0 by

```
aconnect 28:0 128:0
```

Clients can then send MIDI messages to the raw MIDI device `hw:3,0` and TiMidity will synthesize them. We will use this in the next chapter by showing how to replace the default Java synthesizer by TiMidity.

Conclusion

This chapter has briefly discussed the MIDI models available under Alsa. While there is a substantial programming API behind this, we have mainly used the commands `amidi` , `aplaymidi` and `aconnect` .

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

A minimal chapter pointing to resources

Resources

- Introducing GStreamer by Marcelo Lira
- GStreamer by Gnome Project
- Multipurpose multimedia processing with GStreamer by Maciej Katafiasz
-

Example

```
gst-launch-0.10 filesrc location="rehab.mid" !
decodebin ! alsasink
```

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

FluidSynth is an application for playing MIDI files and a library for MIDI applications.

Resources

- FluidSynth home page
- FluidSynth download page
- FluidSynth 1.1 Developer Documentation
- Source Archive fluidsynth Documentation

Players

`fluidsynth` is a command line MIDI player. It runs under ALSA with a command line

```
fluidsynth -a alsa -l <soundfont> <files...>
```

`qsynth` is a GUI interface to `fluidsynth`.

Play MIDI files

The FluidSynth API consists of

- A sequencer created using `new_fluid_player`
- A synthesizer created using `new_fluid_synth`
- An audio player created using `new_fluid_audio_driver` which runs in a separate thread
- A "settings" object which can be used to control many features of the other components, created by `new_fluid_settings` and modified by calls such as `fluid_settings_setstr`

A typical program to play a sequence of MIDI files using ALSA follows. It creates the various objects, sets the audio player to use ALSA and then adds each soundfont and MIDI file to the player. The call to `fluid_player_play` then plays each MIDI file in turn:

```c
#include <fluidsynth.h>
#include <fluid_midi.h>


int main(int argc, char** argv)
{
    int i;
    fluid_settings_t* settings;
    fluid_synth_t* synth;
    fluid_player_t* player;
    fluid_audio_driver_t* adriver;

    settings = new_fluid_settings();
    fluid_settings_setstr(settings,
"audio.driver", "alsa");
    synth = new_fluid_synth(settings);
    player = new_fluid_player(synth);

    adriver = new_fluid_audio_driver(settings,
synth);
    /* process command line arguments */
    for (i = 1; i < argc; i++) {
        if (fluid_is_soundfont(argv[i])) {
        fluid_synth_sfload(synth, argv[1], 1);
        } else {
            fluid_player_add(player, argv[i]);
```

```
        }
    }
    /* play the midi files, if any */
    fluid_player_play(player);
    /* wait for playback termination */
    fluid_player_join(player);
    /* cleanup */
    delete_fluid_audio_driver(adriver);
    delete_fluid_player(player);
    delete_fluid_synth(synth);
    delete_fluid_settings(settings);
    return 0;
}
```

Python

`pyFluidSynth` is a Python binding to FluidSynth.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

Timidity is designed as a standalone application. To add to this, you should build a new "interface". It can also be subverted to act as though it were a library that can be called. This chapter explains both ways.

Files

Source files in this chapter are here .

Timidity design

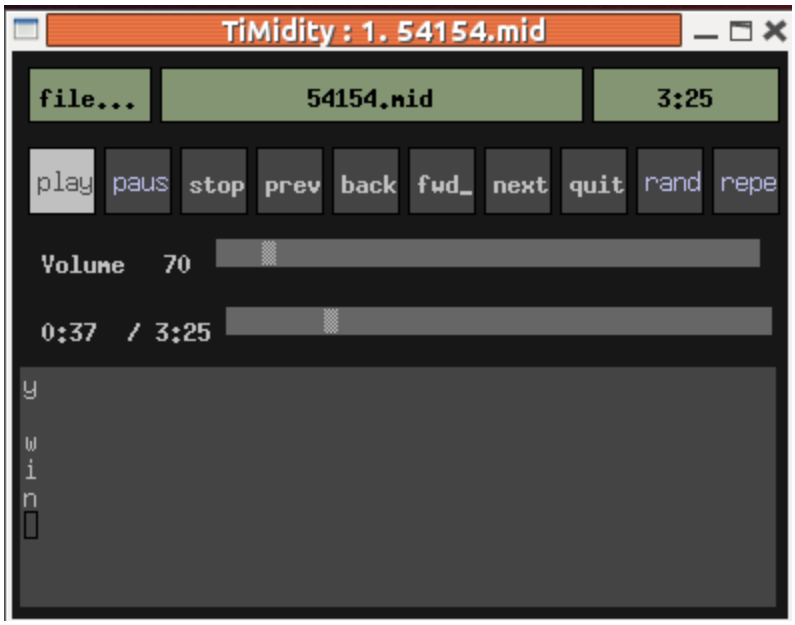
Timidty is designed as a standalone application. When built, you get a single executable but do not get a library of functions that can be called, unlike FluidSynth for example.

What you *can* do with Timidity is to add different interfaces. For example, there are ncurses, XaW and dumb interfaces which can be invoked at runtime by e.g.

```
timidity -in ...
timidity -ia ...
timidity -id ...
```

There are also others with more specialised uses such as WRD, emacs, ALSA and remote interfaces.

For example, the Xaw interface looks like



The idea seems to be that if you want something extra, perhaps you should build a custom interface and drive it from Timidity.

That doesn't always suit me, as I would prefer to be able to embed Timidity into my own applications in a simple way. The rest of this chapter looks at both ways:

- Turn TiMidity into a library and include it in your own code
- Build your own interface

Making TiMidity into a library

TiMidity is not designed as a library and you have to convince it otherwise. That isn't hard, by messing around with the build system.

Managed environments hook

A system whereby the application is in control doesn't work so well in a managed environment such as Windows (or probably the many more recent ones such as Android). In such environments you can't call Timidity's `main`, but rather the `main` function belonging to the framework. This in turn will call an appropriate function in the application.

To make use of such hooks, you need to download the source code of Timidity, either from a package manager or from the TiMidity++ web site.

For Timidity, the variations on the `main` function are in the file `timidity/timidity.c`. Controlled by various define's, you can have `main` or `win_main`. One of the more interesting

defines is `ANOTHER_MAIN`. If this is defined, none of the versions of the `main` function are compiled and you end with main-free object modules.

If you build Timidity from the toplevel source directory in the following way, it will produce an error that the `main` function is not defined:

```
CFLAGS="-DANOTHER_MAIN" ./configure
make
```

That's the hook you need to take Timidity from being a standalone application to being able to be called as a library from another application. Note that you *cannot* just remove `timidity/timidity.c` from the build - that file contains too many other critical functions!

Building the library

To build Timidity as a static library, remove the `main` function as above and attempt to build timidity. I found I needed to also specify which output system I wanted to use, such as ALSA:

```
CFLAGS="-DANOTHER_MAIN" ./configure --enable-
audio=alsa
nake clean
make
```

This builds several `.ar` files and lots of object `.o` modules, but fails to build the final `timidity` executable as (of course) there is no `main` function. It also leaves a bunch of unlinked files in the `timidity` subdirectory.

You can collect all of the object modules into an archive file by running this from the top of the TiMidity source directory:

```
ar cru  libtimidity.a */*.o
ranlib libtimidity.a
```

Since you will have to build Timidity from the source, check that it is working in normal mode before you try to build this alternative library version. That way you can find out that you need, say, the `libasound-dev` library in order to use ALSA, before you get mixed up in this other stuff!

Library entry points

Timidity built with `ANOTHER_MAIN` exposes these public entry points

```
void timidity_start_initialize(void);
int timidity_pre_load_configuration(void);
int timidity_post_load_configuration(void);
void timidity_init_player(void);
int timidity_play_main(int nfiles, char
**files);
int got_a_configuration;
```

They do not seem to be defined in any convenient header file.

A minimal application

The real Timidity application is coded to work on many different operating systems with many different versions of libraries. Most of those dependencies are taken care of in building the object files and library as above.

A minimal application just wraps our own `main` around the library entry points in `my_main.c` :

```c
#include <stdio.h>

extern void timidity_start_initialize(void);
extern int
timidity_pre_load_configuration(void);
extern int
timidity_post_load_configuration(void);
extern void timidity_init_player(void);
extern int timidity_play_main(int nfiles, char
**files);
extern int got_a_configuration;

int main(int argc, char **argv)
{
    int err, main_ret;

    timidity_start_initialize();




    if ((err =
timidity_pre_load_configuration()) != 0)
    return err;

    err += timidity_post_load_configuration();
```

```
    if (err) {
    printf("couldn't load configuration
file\n");
    exit(1);
    }

    timidity_init_player();

    main_ret = timidity_play_main(argc, argv);

    return main_ret;
}
```

The compile command needs to bring in the Timidity library and any other required library, and is for an ALSA application

```
my_timidity: my_main.o
        gcc -g -o my_timidity my_main.o
libtimidity.a  -lasound -lm
```

Playing a background video to a MIDI file

As a more complex example, we look at playing a video file while also playing a MIDI file. We assume that the video file has no audio component and there is no attempt to perform any synchronisation between the two streams - that is an extra order of complexity!

To play a video file we make use of the FFMpeg library to decode a video stream into video frames. We then need to display the frames in some kind of GUI object, and there are very, very many toolkits for doing this :-(. I've chosen the GTK toolkit as it underlies Gnome, is in C, supports many other things such as i18n and so on. I've based my code on An ffmpeg and SDL Tutorial by Stephen Dranger which uses the SDL toolkit for display.

This runs the video and the MIDI in separate threads using the `pthreads` package. I've cheated a bit by hard-coding the names of the files and fixing the size of the video frames. It was a real b*mmer getting it to work under GTK3.0 as that has removed pixmaps and it took too, too long to find out what was going on.

I've split the code into two files, one to play the video using Gtk and the other to play the TiMidity library and invoke the video. The video playing file is `video_code.c` :

```c
#include <gtk/gtk.h>

#include <libavcodec/avcodec.h>
#include <libavformat/avformat.h>
#include <libswscale/swscale.h>

#define MIDI_FILE "54154.mid"
#define VIDEO_FILE "short.mpg"

GtkWidget *image;

#if GTK_MAJOR_VERSION == 2
GdkPixmap *dbuf_pixmap;
#endif

/* FFMpeg vbls */
AVFormatContext *pFormatCtx = NULL;
AVCodecContext *pCodecCtx = NULL;
int videoStream;
struct SwsContext *sws_ctx = NULL;
AVCodec *pCodec = NULL;

static void pixmap_destroy_notify(guchar
*pixels,
                  gpointer data) {
    printf("Ddestroy pixmap\n");
```

```
}

static void *play_background(void *args) {
    /* based on code from

http://www.cs.dartmouth.edu/~xy/cs23/gtk.html

http://cdry.wordpress.com/2009/09/09/using-
custom-io-callbacks-with-ffmpeg/
    */

    int i;
    AVPacket packet;
    int frameFinished;
    AVFrame *pFrame = NULL;

    int oldSize;
    char *oldData;
    int bytesDecoded;
    GdkPixbuf *pixbuf;
    AVFrame *picture_RGB;
    char *buffer;

#if GTK_MAJOR_VERSION == 2
    GdkPixmap *pixmap;
#endif
```

```
    pFrame=avcodec_alloc_frame();

    i=0;
    picture_RGB = avcodec_alloc_frame();
    buffer = malloc
(avpicture_get_size(PIX_FMT_RGB24, 720, 576));
    avpicture_fill((AVPicture *)picture_RGB,
buffer, PIX_FMT_RGB24, 720, 576);

    while(av_read_frame(pFormatCtx,
&packet)>=0) {
    if(packet.stream_index==videoStream) {
        //printf("Frame %d\n", i++);
        usleep(33670);  // 29.7 frames per
second
        // Decode video frame
        avcodec_decode_video2(pCodecCtx,
pFrame, &frameFinished,   &packet);

        sws_ctx = sws_getContext(pCodecCtx-
>width, pCodecCtx->height,
                     pCodecCtx->pix_fmt,
pCodecCtx->width,
                     pCodecCtx->height,
PIX_FMT_RGB24,
                     SWS_BICUBIC, NULL, NULL,
NULL);
```

```
        if (frameFinished) {

        sws_scale(sws_ctx,  (uint8_t const *
const *) pFrame->data, pFrame->linesize, 0,
pCodecCtx->height, picture_RGB->data,
picture_RGB->linesize);

        pixbuf =
gdk_pixbuf_new_from_data(picture_RGB->data[0],
GDK_COLORSPACE_RGB, 0, 8, 720, 480,
picture_RGB->linesize[0],
pixmap_destroy_notify, NULL);


        /* start GTK thread lock for drawing.
           deprecated under GTK 3.0 but no
clear idea
           of what to do :-(
        */
        gdk_threads_enter();
#if GTK_MAJOR_VERSION == 3
        gtk_image_set_from_pixbuf((GtkImage*)
image, pixbuf);
#elif GTK_MAJOR_VERSION == 2
         gdk_draw_pixbuf((GdkDrawable *)
dbuf_pixmap, NULL,
```

```
                    pixbuf,
                    0, 0, 0, 0, 720, 480,
                    GDK_RGB_DITHER_NORMAL, 0, 0);
            gtk_image_set_from_pixmap((GtkImage*)
image, dbuf_pixmap, NULL);
#endif
            gtk_widget_queue_draw(image);

            /* release GTK thread lock */
            gdk_threads_leave();
            }
        }
        av_free_packet(&packet);
        }
        printf("Video over!\n");
        exit(0);
}

/* Called when the windows are realized
 */
static void realize_cb (GtkWidget *widget,
gpointer data) {
        /* start the video playing in its own
thread */
        pthread_t tid_video;
        pthread_create(&tid_video, NULL,
play_background, NULL);
```

```
}

static gboolean delete_event( GtkWidget
*widget,
                              GdkEvent  *event,
                              gpointer   data )
{
    /* If you return FALSE in the "delete-
event" signal handler,
     * GTK will emit the "destroy" signal.
Returning TRUE means
     * you don't want the window to be
destroyed.
     * This is useful for popping up 'are you
sure you want to quit?'
     * type dialogs. */

    g_print ("delete event occurred\n");

    /* Change TRUE to FALSE and the main window
will be destroyed with
     * a "delete-event". */
    return FALSE;
}

/* Another callback */
static void destroy( GtkWidget *widget,
```

```
                        gpointer   data )
{
    gtk_main_quit ();
}

void init_ffmpeg() {
    /* FFMpeg stuff */

    AVFrame *pFrame = NULL;
    AVPacket packet;

    AVDictionary *optionsDict = NULL;


    av_register_all();

    if(avformat_open_input(&pFormatCtx,
VIDEO_FILE, NULL, NULL)!=0)
    return; // Couldn't open file

    // Retrieve stream information
    if(avformat_find_stream_info(pFormatCtx,
NULL)<0)
    return; // Couldn't find stream information

    // Dump information about file onto
standard error
```

```
    av_dump_format(pFormatCtx, 0, VIDEO_FILE,
0);

    // Find the first video stream
    videoStream=-1;
    int i;
    for(i = 0; i < pFormatCtx->nb_streams; i++)
    if(pFormatCtx->streams[i]->codec-
>codec_type==AVMEDIA_TYPE_VIDEO) {
        videoStream=i;
        break;
    }
    if(videoStream==-1)
    return; // Didn't find a video stream

    for(i=0; i<pFormatCtx->nb_streams; i++)
    if(pFormatCtx->streams[i]->codec-
>codec_type==AVMEDIA_TYPE_AUDIO) {
        printf("Found an audio stream too\n");
        break;
    }


    // Get a pointer to the codec context for
the video stream
    pCodecCtx=pFormatCtx->streams[videoStream]-
>codec;
```

```
    // Find the decoder for the video stream
    pCodec=avcodec_find_decoder(pCodecCtx-
>codec_id);
    if(pCodec==NULL) {
    fprintf(stderr, "Unsupported codec!\n");
    return; // Codec not found
    }

    // Open codec
    if(avcodec_open2(pCodecCtx, pCodec,
&optionsDict)<0)
    return; // Could not open codec
}

void *play_gtk(void *args);

void init_gtk(int argc, char **argv) {
    /* GTK stuff now */

    printf(" GTK_MAJOR_VERSION is %d\n",
GTK_MAJOR_VERSION);

   /* GtkWidget is the storage type for widgets
*/
    GtkWidget *window;
    GtkWidget *box;
```

```
    /* This is called in all GTK applications.
Arguments are parsed
     * from the command line and are returned
to the application. */
    printf("Initing gtk\n");
    if ( !gtk_init_check (0, NULL)) {
    printf("Gtk init failed\n");
    } else {
    printf("Inited Gtk ok\n");
    }

    /* create a new window */
    window = gtk_window_new
(GTK_WINDOW_TOPLEVEL);

    /* When the window is given the "delete-
event" signal (this is given
     * by the window manager, usually by the
"close" option, or on the
     * titlebar), we ask it to call the
delete_event () function
     * as defined above. The data passed to the
callback
     * function is NULL and is ignored in the
callback function. */
    g_signal_connect (window, "delete-event",
```

```
                G_CALLBACK (delete_event), NULL);

    /* Here we connect the "destroy" event to a
signal handler.
     * This event occurs when we call
gtk_widget_destroy() on the window,
     * or if we return FALSE in the "delete-
event" callback. */
    g_signal_connect (window, "destroy",
                G_CALLBACK (destroy), NULL);

    g_signal_connect (window, "realize",
G_CALLBACK (realize_cb), NULL);

    /* Sets the border width of the window. */
    gtk_container_set_border_width
(GTK_CONTAINER (window), 10);

#if GTK_MAJOR_VERSION == 3
    box = gtk_box_new(TRUE, 1);
#else
    box = gtk_vbox_new(TRUE, 1);
#endif

    gtk_widget_show(box);

    image = gtk_image_new();
```

```
    gtk_widget_show (image);

    // gtk_box_pack_start (GTK_BOX (box),
lyric_labels[0], TRUE, TRUE, 0);

    gtk_box_pack_start (GTK_BOX (box), image,
TRUE, TRUE, 0);

    /* This packs the button into the window (a
gtk container). */
    gtk_container_add (GTK_CONTAINER (window),
box);

    /* and the window */
    gtk_widget_show (window);

    /* All GTK applications must have a
gtk_main(). Control ends here
     * and waits for an event to occur (like a
key press or
     * mouse event). */

    /* start Gtk in its own thread */
    pthread_t tid_gtk;
    pthread_create(&tid_gtk, NULL, play_gtk,
NULL);
```

```
    // play_gtk();
    //gtk_main ();
    printf("Finished initing gtk\n");
}


void *play_gtk(void *args) {
    printf("About to start gtk_main\n");
    gtk_main();
}
```

The file `video_player.c` is

```c
#include <string.h>



#define MIDI_FILE "54154.mid"

static void *play_midi(void *args) {
    char *argv[1];
    argv[0] = MIDI_FILE;
    int argc = 1;

    timidity_play_main(argc, argv);

    printf("Audio finished\n");
    exit(0);
}


int main(int argc, char** argv)
{

    int i;

    /* make X happy */
    XInitThreads();
```

```
    /* Timidity stuff */
    int err;

    timidity_start_initialize();
    if ((err =
timidity_pre_load_configuration()) == 0) {
    err = timidity_post_load_configuration();
    }
    if (err) {
        printf("couldn't load configuration
file\n");
        exit(1);
    }

    timidity_init_player();

    init_ffmpeg();
    init_gtk(argc, argv);

    play_midi(NULL);
    return 0;
}
```

Building a new interface

Shared objects

You can build your own interfaces and add them to TiMidity without changing or recompiling TiMidity. Such interfaces are built as dynamically loadable shared libraries, and are loaded when TiMidity starts.

You have to be a little careful with compile and link flags to build these libraries (see Building shared objects in Linux). To build the shared object `if_my_interface.so` from `my_interface.c` I use

```
gcc  -fPIC $(CFLAGS) -c -o my_interface.o
my_interface.c
gcc -shared -o if_my_interface.so
my_interface.o
```

TiMidity will only load files that begin with `if_` . They can reside in any directory, with the default being something like `/usr/lib/timidity` or `/usr/local/lib/timidity` (see

the "Supported dynamic load interfaces" directory from `timidity -h`).

The defaulty directory to load dynamic modules can be overridden by the option `-d` as in

```
timidity -d. -ik --trace 54154.mid
```

Entry point

Each interface must have a unique function that can be called by the dynamic loader. Recall that interfaces are selected by the command line option `-i` , such as `timidity -iT ...` to choose the VT100 interface. Your interface must have a single ASCII letter identifier which isn't used by any other interface, say `m` for "my interface". The loader will then look for a function

```
ControlMode *interface_m_loader(void)
```

where the "m" in the function name is the identifier. This function is simple: it just returns the address of a structure of type `ControlMode` which is defined elsewhere in the

interface's code:

```
ControlMode *interface_m_loader(void)
{
    return &ctl;
}
```

ControlMode

The `ControlMode` structure is

```
typedef struct {
  char *id_name, id_character;
  char *id_short_name;
  int verbosity, trace_playing, opened;

  int32 flags;

  int  (*open)(int using_stdin, int
using_stdout);
  void (*close)(void);
  int (*pass_playing_list)(int number_of_files,
char *list_of_files[]);
  int  (*read)(int32 *valp);
  int  (*write)(char *buf, int32 size);
  int  (*cmsg)(int type, int verbosity_level,
char *fmt, ...);
  void (*event)(CtlEvent *ev);  /* Control
events */
} ControlMode;
```

which defines information about the interface and a set of functions which are called by TiMidity in response to events and actions within TiMidity. For example, for "my interface" this structure is

```
ControlMode ctl=
    {
    "my interface", 'm',
    "my iface",
    1,          /* verbosity */
    0,          /* trace playing */
    0,          /* opened */
    0,          /* flags */
    ctl_open,
    ctl_close,
    pass_playing_list,
    ctl_read,
    NULL,       /* write */
    cmsg,
    ctl_event
    };
```

Some of these fields are obvious, some are less so:

- **`open`** : This is called to set which files are used for I/O...
- **`close`** : ... and to close them
- **`pass_playing_list`** : This function is passed a list of files to play. The most likely action is to walk through this list, calling `play_midi_file` on each
- **`read`** : Not sure yet

- `write` : Not sure yet
- `cmsg` : Called with information messages
- `event` : This is the major function for handling MIDI control events. Typically it will be a big switch for each type of control event

Include files

This is messy. A typical interface will need to know some of the constants and functions used by TiMidity. While these are organised logically for TiMidity, they are not organised conveniently for a new interface. So you have to keep pulling in extra includes, which point to other externals, which require more includes, etc. These may be in different directories such `timidity` and `utils` so you have to point to many different include directories.

Note that you will need the TiMidity source code to get these include files, downloaded from SourceForge TiMidity++ .

My simple interface

This basically does the same as the "dumb" interface built into TiMidity. It is loaded from the current directory and invoked by

```
timidity -im -d. 54154.mid
```

The code is in one file, `my_interface.c` :

```c
/*
  my_interface.c
*/

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif /* HAVE_CONFIG_H */
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#ifndef NO_STRING_H
#include <string.h>
#else
#include <strings.h>
#endif

#include "support.h"
#include "timidity.h"
#include "output.h"
#include "controls.h"
#include "instrum.h"
#include "playmidi.h"
#include "readmidi.h"

static int ctl_open(int using_stdin, int
using_stdout);
```

```
static void ctl_close(void);
static int ctl_read(int32 *valp);
static int cmsg(int type, int verbosity_level,
char *fmt, ...);
static void ctl_total_time(long tt);
static void ctl_file_name(char *name);
static void ctl_current_time(int ct);
static void ctl_lyric(int lyricid);
static void ctl_event(CtlEvent *e);
static int pass_playing_list(int
number_of_files, char *list_of_files[]);


/***********************************/
/* export the interface functions */

#define ctl karaoke_control_mode

ControlMode ctl=
    {
    "my interface", 'm',
    "my iface",
    1,          /* verbosity */
    0,          /* trace playing */
    0,          /* opened */
    0,          /* flags */
    ctl_open,
```

```
    ctl_close,
    pass_playing_list,
    ctl_read,
    NULL,       /* write */
    cmsg,
    ctl_event
    };

static FILE *outfp;
int karaoke_error_count;
static char *current_file;
struct midi_file_info *current_file_info;

static int pass_playing_list(int
number_of_files, char *list_of_files[]) {
    int n;

    for (n = 0; n < number_of_files; n++) {
    printf("Playing list %s\n",
list_of_files[n]);

    current_file = list_of_files[n];
    /*
      current_file_info =
get_midi_file_info(current_file, 1);
      if (current_file_info != NULL) {
      printf("file info not NULL\n");
```

```
      } else {
      printf("File info is NULL\n");
      }
    */
    play_midi_file( list_of_files[n]);
    }
    return 0;
}

/*ARGSUSED*/
static int ctl_open(int using_stdin, int
using_stdout)
{
    if(using_stdout)
    outfp=stderr;
    else
    outfp=stdout;
    ctl.opened=1;
    return 0;

    /*
    // dont know what this function does
    if (current_file != NULL) {
    current_file_info =
get_midi_file_info(current_file, 1);
    printf("Opening info for %s\n",
current_file);
```

```
    } else {
    printf("Current is NULL\n");
    }
    return 0;
    */
}

static void ctl_close(void)
{
    fflush(outfp);
    ctl.opened=0;
}

/*ARGSUSED*/
static int ctl_read(int32 *valp)
{
    return RC_NONE;
}

static int cmsg(int type, int verbosity_level,
char *fmt, ...)
{
    va_list ap;

    if ((type==CMSG_TEXT || type==CMSG_INFO ||
type==CMSG_WARNING) &&
    ctl.verbosity<verbosity_level)
```

```
    return 0;
    va_start(ap, fmt);
    if(type == CMSG_WARNING || type ==
CMSG_ERROR || type == CMSG_FATAL)
    karaoke_error_count++;
    if (!ctl.opened)
    {
        vfprintf(stderr, fmt, ap);
        fputs(NLS, stderr);
    }
    else
    {
        vfprintf(outfp, fmt, ap);
        fputs(NLS, outfp);
        fflush(outfp);
    }
    va_end(ap);
    return 0;
}

static void ctl_total_time(long tt)
{
    int mins, secs;
    if (ctl.trace_playing)
    {
        secs=(int)(tt/play_mode->rate);
        mins=secs/60;
```

```
        secs-=mins*60;
        cmsg(CMSG_INFO, VERB_NORMAL,
         "Total playing time: %3d min %02d s",
mins, secs);
    }
}

static void ctl_file_name(char *name)
{
    current_file = name;

    if (ctl.verbosity>=0 || ctl.trace_playing)
    cmsg(CMSG_INFO, VERB_NORMAL, "Playing %s",
name);
}

static void ctl_current_time(int secs)
{
    int mins;
    static int prev_secs = -1;

#ifdef __W32__
    if(wrdt->id == 'w')
    return;
#endif /* __W32__ */
    if (ctl.trace_playing && secs != prev_secs)
    {
```

```
        prev_secs = secs;
        mins=secs/60;
        secs-=mins*60;
        //fprintf(outfp, "\r%3d:%02d", mins,
secs);
        //fflush(outfp);
    }
}

static void ctl_lyric(int lyricid)
{
    char *lyric;

    current_file_info =
get_midi_file_info(current_file, 1);

    lyric = event2string(lyricid);
    if(lyric != NULL)
    {
        if(lyric[0] == ME_KARAOKE_LYRIC)
        {
            if(lyric[1] == '/' || lyric[1] ==
'\\')
            {
                fprintf(outfp, "\n%s", lyric +
2);
                fflush(outfp);
```

```
            }
            else if(lyric[1] == '@')
            {
                if(lyric[2] == 'L')
                fprintf(outfp, "\nLanguage:
%s\n", lyric + 3);
                else if(lyric[2] == 'T')
                fprintf(outfp, "Title: %s\n",
lyric + 3);
                else
                fprintf(outfp, "%s\n", lyric +
1);
            }
            else
            {
                fputs(lyric + 1, outfp);
                fflush(outfp);
            }
        }
        else
        {
            if(lyric[0] == ME_CHORUS_TEXT ||
lyric[0] == ME_INSERT_TEXT)
            fprintf(outfp, "\r");
            fputs(lyric + 1, outfp);
            fflush(outfp);
        }
```

```
    }
}


static void ctl_event(CtlEvent *e)
{
    switch(e->type)
    {
    case CTLE_NOW_LOADING:
        ctl_file_name((char *)e->v1);
        break;
    case CTLE_LOADING_DONE:
        // MIDI file is loaded, about to play
        current_file_info =
get_midi_file_info(current_file, 1);
        if (current_file_info != NULL) {
        printf("file info not NULL\n");
        } else {
        printf("File info is NULL\n");
        }
        break;
    case CTLE_PLAY_START:

        ctl_total_time(e->v1);
        break;
    case CTLE_CURRENT_TIME:
        ctl_current_time((int)e->v1);
```

```
        break;
#ifndef CFG_FOR_SF
    case CTLE_LYRIC:
        ctl_lyric((int)e->v1);
        break;
#endif
    }
}

/*
 * interface_<id>_loader();
 */
ControlMode *interface_k_loader(void)
{
    return &ctl;
}
```

Running my simple interface

When I tried to run the interface using the standard package TiMidity v2.13.2-40.1 it crashed in a memory free call. The code is stripped, so tracking down why is not easy and I haven't bothered to do so yet - I'm not sure what libraries, versions of code, etc, the package distro was compiled against.

I had built my own copy of TiMidity from source. This worked fine. Note that when you build TiMidity from source, you will need to specify that it can load dynamic modules, for example by

```
congfigure --enable-audio=alsa --enable-vt100 -
-enable-debug --enable-dynamic
```

Playing a background video to a MIDI file

We can take the code from playing a video given earlier and put it as the "back end" of a TiMidity systems as a "video" interface. Essentially all that needs to be done is to change `ctl_open` from the simple interface to call the Gtk code to play the video, and change the identity of the interface.

The new "video" interface is `video_player_interface.c`

```c
/*
  video_player_interface.c
*/

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif /* HAVE_CONFIG_H */
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#ifndef NO_STRING_H
#include <string.h>
#else
#include <strings.h>
#endif

#include "support.h"
#include "timidity.h"
#include "output.h"
#include "controls.h"
#include "instrum.h"
#include "playmidi.h"
#include "readmidi.h"

static int ctl_open(int using_stdin, int
using_stdout);
```

```
static void ctl_close(void);
static int ctl_read(int32 *valp);
static int cmsg(int type, int verbosity_level,
char *fmt, ...);
static void ctl_total_time(long tt);
static void ctl_file_name(char *name);
static void ctl_current_time(int ct);
static void ctl_lyric(int lyricid);
static void ctl_event(CtlEvent *e);
static int pass_playing_list(int
number_of_files, char *list_of_files[]);


/***********************************/
/* export the interface functions */

#define ctl karaoke_control_mode

ControlMode ctl=
    {
    "video player interface", 'v',
    "video player",
    1,          /* verbosity */
    0,          /* trace playing */
    0,          /* opened */
    0,          /* flags */
    ctl_open,
```

```
    ctl_close,
    pass_playing_list,
    ctl_read,
    NULL,        /* write */
    cmsg,
    ctl_event
    };

static FILE *outfp;
int video_player_error_count;
static char *current_file;
struct midi_file_info *current_file_info;

static int pass_playing_list(int
number_of_files, char *list_of_files[]) {
    int n;

    for (n = 0; n < number_of_files; n++) {
    printf("Playing list %s\n",
list_of_files[n]);

    current_file = list_of_files[n];
    /*
      current_file_info =
get_midi_file_info(current_file, 1);
      if (current_file_info != NULL) {
      printf("file info not NULL\n");
```

```
      } else {
      printf("File info is NULL\n");
      }
    */
    play_midi_file( list_of_files[n]);
    }
    return 0;
}

extern void *play_gtk(void *args);

/*ARGSUSED*/
static int ctl_open(int using_stdin, int
using_stdout)
{
    if(using_stdout)
    outfp=stderr;
    else
    outfp=stdout;
    ctl.opened=1;

    init_ffmpeg();

    /* start Gtk in its own thread */
    pthread_t tid_gtk;
    init_gtk(0, NULL);
    //pthread_create(&tid_gtk, NULL, play_gtk,
```

```
NULL);

    return 0;
}

static void ctl_close(void)
{
    fflush(outfp);
    ctl.opened=0;
}

/*ARGSUSED*/
static int ctl_read(int32 *valp)
{
    return RC_NONE;
}

static int cmsg(int type, int verbosity_level,
char *fmt, ...)
{
    va_list ap;

    if ((type==CMSG_TEXT || type==CMSG_INFO ||
type==CMSG_WARNING) &&
    ctl.verbosity<verbosity_level)
    return 0;
    va_start(ap, fmt);
```

```
    if(type == CMSG_WARNING || type ==
CMSG_ERROR || type == CMSG_FATAL)
    video_player_error_count++;
    if (!ctl.opened)
    {
        vfprintf(stderr, fmt, ap);
        fputs(NLS, stderr);
    }
    else
    {
        vfprintf(outfp, fmt, ap);
        fputs(NLS, outfp);
        fflush(outfp);
    }
    va_end(ap);
    return 0;
}

static void ctl_total_time(long tt)
{
    int mins, secs;
    if (ctl.trace_playing)
    {
        secs=(int)(tt/play_mode->rate);
        mins=secs/60;
        secs-=mins*60;
        cmsg(CMSG_INFO, VERB_NORMAL,
```

```
          "Total playing time: %3d min %02d s",
mins, secs);
    }
}

static void ctl_file_name(char *name)
{
    current_file = name;

    if (ctl.verbosity>=0 || ctl.trace_playing)
    cmsg(CMSG_INFO, VERB_NORMAL, "Playing %s",
name);
}

static void ctl_current_time(int secs)
{
    int mins;
    static int prev_secs = -1;

#ifdef __W32__
    if(wrdt->id == 'w')
    return;
#endif /* __W32__ */
    if (ctl.trace_playing && secs != prev_secs)
    {
        prev_secs = secs;
        mins=secs/60;
```

```
        secs-=mins*60;
        //fprintf(outfp, "\r%3d:%02d", mins,
secs);
        //fflush(outfp);
    }
}

static void ctl_lyric(int lyricid)
{
    char *lyric;

    current_file_info =
get_midi_file_info(current_file, 1);

    lyric = event2string(lyricid);
    if(lyric != NULL)
    {
        if(lyric[0] == ME_KARAOKE_LYRIC)
        {
            if(lyric[1] == '/' || lyric[1] ==
'\\')
            {
                fprintf(outfp, "\n%s", lyric +
2);
                fflush(outfp);
            }
            else if(lyric[1] == '@')
```

```
            {
                if(lyric[2] == 'L')
                fprintf(outfp, "\nLanguage:
%s\n", lyric + 3);
                else if(lyric[2] == 'T')
                fprintf(outfp, "Title: %s\n",
lyric + 3);
                else
                fprintf(outfp, "%s\n", lyric +
1);
            }
            else
            {
                fputs(lyric + 1, outfp);
                fflush(outfp);
            }
        }
        else
        {
            if(lyric[0] == ME_CHORUS_TEXT ||
lyric[0] == ME_INSERT_TEXT)
            fprintf(outfp, "\r");
            fputs(lyric + 1, outfp);
            fflush(outfp);
        }
    }
}
```

```
static void ctl_event(CtlEvent *e)
{
    switch(e->type)
    {
    case CTLE_NOW_LOADING:
        ctl_file_name((char *)e->v1);
        break;
    case CTLE_LOADING_DONE:
        // MIDI file is loaded, about to play
        current_file_info =
get_midi_file_info(current_file, 1);
        if (current_file_info != NULL) {
        printf("file info not NULL\n");
        } else {
        printf("File info is NULL\n");
        }
        break;
    case CTLE_PLAY_START:

        ctl_total_time(e->v1);
        break;
    case CTLE_CURRENT_TIME:
        ctl_current_time((int)e->v1);
        break;
#ifndef CFG_FOR_SF
```

```
    case CTLE_LYRIC:
        ctl_lyric((int)e->v1);
        break;
#endif
    }
}

/*
 * interface_<id>_loader();
 */
ControlMode *interface_v_loader(void)
{
    return &ctl;
}
```

The build command is

```
video_code.o: video_code.c
        gcc  -fPIC $(CFLAGS) -c -o video_code.o
video_code.c $(LIBS3)

if_video_player.so: video_player_interface.c
video_code.o
        gcc  -fPIC $(CFLAGS) -c -o
video_player_interface.o
video_player_interface.c
        gcc -shared -o if_video_player.so
video_player_interface.o video_code.o $(LIBS3)
```

You may hit a hiccup with running this Gtk-based interface: Gtk version mismatch :-(. The current builds of TiMidity either do not have the Gtk interface compiled in, or have Gtk version 2. If Gtk is not compiled in, you should have no problem. Otherwise, if you have compiled this interface with Gtk version 3, you will get runtime errors about type mismatches, inability to load widgets and no visual display.

Check for Gtk in the executable by

```
nm timidity | grep gtk
```

Summary

Timidity is not designed for use by other applications. Either you add a new interface or work around the TiMidity design to produce a library. This chapter has shown both mechanisms and illustrated them with simple and more complex examples.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

PART 5: KARAOKE

This chapter gives a quick summary of the successive chapters.

Overview

The whole purpose of this book - from my point of view - is to document what is going on in Linux Sound in my path to building a Linux Karaoke system. I'm still on the way. This chapter looks at various explorations I have made using the material of previous chapters.

First, what are my goals?

- Be able to play KAR files (one of the possible Karaoke file formats)
- Show the lyrics at least one line at a time, highlighting the characters when they should be sung
- For Chinese songs, also show the Pinyin (English) form of the lyrics as well as the Chinese characters
- Play a movie in the background
- Display the melody in some form
- Show the notes actually sung against the melody
- Score the results in some way

Nothing I have yet done gets anywhere near these goals. Let me pick out the highlights of my explorations so far:

- The simplest "off the shelf" system is PyKaraoke , with kmid a close follower. These play KAR files and highlight the lyrics but no more.
- The simplest way to add microphone input to such a system is to use an external mixer . These can also do reverb and other effects
- Jack and PulseAudio can trivially be used to add in microphone input as play-through, but effects take more work
- Java is really cool for nearly everything - except latency ruins it in the end
- FluidSynth can be hacked to give hooks to hang Karaoke from. But it is CPU intensive and doesn't leave room for much other processing
- TiMidity is a standalone system with configurable backends. It can be configured to give a crude Karaoke system. But it can be hacked to make it into a library which gives it more potential. It is not as CPU intensive as FluidSynth.
- Playing a background movie can be done using FFMpeg and a GUI such as Gtk . Gtk also has the mechanisms for overlaying highlighted lyrics on top of the video, but Gtk 2 and Gtk 3 differ in the mechanism
- TiMidity can be combined with FFMpeg and GtK to display hightlighted lyrics against a movie background

- Scoring is still out of sight right now, although the Java library TarsosDSP can give lots of information

The following chapters cover:

- **User level tools** : Karaoke is an "audience participation" sound system, in which the soundtrack and usually the melody are played along with a moving display of the lyrics. This chapter considers the features, formats and user-level tools for playing Karaoke.
- **Decoding the DKD files on the Songken Karaoke DVD** : This chapter is about getting the information off my Songken Karaoke DVD so that I can start writing programs to play the songs. It is not directly involved in playing sound under Linux and can be skipped.
- **JavaSound** : JavaSound has no direct support for Karaoke. This chapter looks at how to combine the JavaSound libraries with other libraries such as Swing to give a Karaoke player for MIDI files.
- **Subtitles** : Many Karaoke systems use subtitles imposed over a movie of some kind. This chapter looks at how to do this with Linux systems. There are limited choices, but it is possible.
- **FluidSynth** : FluidSynth is an application for playing MIDI files and a library for MIDI applications. It does not have the

hooks for playing Karaoke files. This chapter discusses an extension to FluidSynth which adds appropriate hooks, and then uses these to build a variety of Karaoke systems.

- **Timidity** : TiMidity is designed as a standalone application with a particular kind of extensibility. Out of the box it can sort-of play Karaoke but not well. This chapter looks at how to work with TiMidity to build a Karaoke system.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

Karaoke is an "audience participation" sound system, in which the soundtrack and usually the melody are played along with a moving display of the lyrics. This chapter considers the features, formats and user-level tools for playing Karaoke.

Karaoke

Karaoke is an "audience participation" sound system, in which the soundtrack and usually the melody are played along with a moving display of the lyrics. Within this, there can be variations or different features:

- The lyrics can be shown all at once, while the music plays in sequence
- The lyrics can be highlighted in synchronisation with the melody line
- The melody line maybe will always play, or can be switched off
- Some players will also include a vocalist singing the song
- Some players with vocals will turn off the vocals when someone is singing
- Some players will give a graphical display of the notes of the melody
- Some players will give a graphical display of the melody and also show the notes the singers are singing
- Some players will produce scores based on some evaluation of the singer's accuracy. The basis of such scoring is usually not known

- Some players allow you to change the playback speed and the playback pitch
- Most players will accept two microphones, and can have reverb effects added to the singer's voices
- Many players will allow you to select songs in advance to build a dynamic playlist

Karaoke is very popular in Asia, and has a following in European countries. Karaoke systems are believed to have originated in Asia although the history according to Wikipedia is a little muddy.

There are various file formats for Karaoke described by Jean-Pierre.Cocatrix KaraWin

Video CD systems

Video CDs are an older form of video storage on an optical disk. The resolution is fairly low, typically 352x240 pixels, with a frame rate of 25 framse per second. Although they were used by a few movies they have been supplanted for movies by DVDs. However, they were used extensively at one stage for Karaoke disks.

The cheaper CD/DVD players from Asia often have microphone inputs, and can be used as Karaoke players with VCD disks. Typically files are simple movies in AVI or MPEG format and so you can just sing along. While the lyrics are usually displayed, highlighted in time to the melody, there are no advanced features such as scoring, or display of the melody.

If you have VCD disks, they can be mounted as IC9660 files on your computer, but on a Linux system you cannot directly extract the files. Players such as VLC, mplayer and Totem can play files from them.

You need to use something like `vcdimager` to extract files from a VCD disk. This may be in your package system, or you can download it from the GNU developer site and build it from

source. The video files can then be extracted as MPEG or AVI files by

```
vcdxrip --cdrom-device=/dev/cdrom --rip
```

(On my system I had to replace /dev/cdrom with /dev/sr1 as I could not extract from the default DVD player. I found out what device it was by running `mount`, then unmounted it by `umount`.)

CD+G disks

From Wikipedia "CD+G":

> CD+G (also known as CD+Graphics) is an extension of the compact disc standard that can present low-resolution graphics alongside the audio data on the disc when played on a compatible device. CD+G discs are often used for karaoke machines, which utilize this functionality to present on-screen lyrics for the song contained on the disc.

Each song is composed of two files: an audio file and a video file containing the lyrics (and maybe some background scenes).

There are many disks that you can buy using this format. You can't play them on directly your computer. rythmbox will play the audio, but not the video. vlc and totem don't like them.

Ripping the files onto your computer for storage on your hard disk is not so straightforward. The audio disks do not have a file system in the normal sense. For example, you cannot mount them using the Unix `mount` command; they are not even in ISO format. Instead you need to use a program like `cdrdao` to rip the files to a binary file and then work on that,

```
$ cdrdao read-cd --driver generic-mmc-raw --
device /dev/cdroms/cdrom0 --read-subchan rw_raw
mycd.toc
```

creates a data file and a table of contents file.

The format of the CDG files has not apparently been publically released, but is decribed by Jim Bumgardner (back in 1995!) at CD+G Revealed: Playing back Karaoke tracks in Software

Programs such as SoundJuicer will extract the audio tracks, but leave the video behind.

MP3+G files

MP3+G files are CD+G files adapted for use on a normal PC. They consist of an MP3 file containing the audio and a CDG file containing the lyrics. Frequently they are zipp'ed together.

Many sites selling CD+G files also sell MP3+G files. Various web sites give instructions on how to create your own MP3+G files. There are not many free sites.

The program `cgdrip.py` from cdgtools-0.3.2 can rip CD+G files from an audio disk and convert them to a pair of MP3+G files. The instructions from the (Python) source code are

> To start using cdgrip immediately, try the following from the command-line (replacing the --device option by the path to your CD device):

```
$ cdrdao read-cd --driver generic-mmc-raw --device /dev/cdroms/cdrom0 --read-subchan rw_raw mycd.toc
 $ python cdgrip.py --with-cddb --delete-bin-toc mycd.toc
```

> The MP3 and CDG files usually share the same root e.g. Track1.mp3 and Track1.cdg. When unzipped, `vlc` can play them by giving it the MP3 file *only* to play as in

```
vlc Track1.mp3
```

> Other players such as mplayer do not recognise this combination and only play the audio. Buying CD+G or MP3+G files
>
> There are many sites selling CD+G or MP3+G songs. Just do a Google search. However, the average price per song is about US$3, and if you want to build up a large collection that can become very expensive. Some sites will give discounts for larger volume purchases, but even at US$30 for 100 songs, the expense can be high.
>
> The cheapest site for *really* large collections that I have found so far is Unify mall There you can purchase (for example) 29,000 English CDG songs for US$399. There is a little hiccup in this: if you buy their "UNIFY PCKP1000A 500-2000GB HDD MP4/DVD/CDG PC Karaoke Player" for US$559 you should be able to get these English songs for free! If you want to also get, say, 30,000 Mandarin songs thrown in for free, the economics of building your own Karaoke player become more shaky... We will ignore that issue here - it's your choice! Converting MP3+G to video files
>
> The tool `ffmpeg` can merge the audio and video to a single video file by e.g.

```
ffmpeg -i Track1.cdg -i Track1.mp3 -y
Track1.avi
```

> or

```
avconv -i Track1.cdg -i Track1.mp3 test.avi
avconv -i test.avi -c:v libx264 -c:a copy
outputfile.mp4
```

to create an AVI file containing both video and audio. This can be played by vlc, mplayer, rhythmbox, etc.

There is a program cdg2video . While it is last dated February, 2011, changes in the FFMpeg internals means that it no longer compiles. Even if you fix the obvious changes, there are a huge number of complaints from the C compiler about the use of deprecated FFMpeg functions. MPEG 4 files

It is becoming common to have Karaoke systems using MPEG 4 video players. These embed all of the information into a video. There is no scoring system with players of these files.

Some rate them as much higher sound quality: "MIDI files sound so cheesy it's sick" . I would suggest it is more an issue with the synthesizer used than the format. Certainly high-end synthesizer manufacturers such as Yamaha would not agree!

MPEG 4 files are certainly larger than the corrsponding MIDI files, and you will need a substantial disk to hold many of them.

There are many sites selling MP4 songs. Just do a Google search. However, the average price per song is about US$3, and if you want to build up a large collection that can become

very expensive. Some sites will give discounts for larger volume purchases, but even at US$30 for 100 songs, the expense can be high.

The cheapest site for *really* large collections that I have found so far is Unify mall There you can purchase (for example) 6,200 English MP4 songs for US$199. There is a little hiccup in this: if you buy their "UNIFY PCKP1000A 500-2000GB HDD MP4/DVD/CDG PC Karaoke Player" for US$559 you should be able to get these English songs for free! If you want to also get, say, 25,000 Mandarin songs thrown in for free, the economics of building your own Karaoke player become more shaky. We will ignore that issue here - it's your choice! Karaoke machines

There are many Karaoke machines which come with a DVD, such as the Malata or Songken systems. In most cases the songs are stored as MIDI files, with the songtrack in one MIDI file and the lyrics in another. Some more recent systems will use WMA files for the soundtrack, and this allows one track to have a vocal supplied and the other without the vocal. Such systems will usually include a scoring mechanism, although the basis for the scoring is not

made explicit. The most recent ones are hard-disk based, usually with MP4 files. They do not seem to have a scoring system.

The two systems I own show different characteristics. The Songken MD-388 plays songs from multiple languages, such as Chinese, Korean, English, etc. My wife is Chinese, but I cannot read Chinese characters. There is an Anglicised script called *PinYin* and the Songken shows both the Chinese characters and the PinYin, so I can sing along too. It looks like

> The Malata MDVD-6619 does not show the PinYin when playing Chinese songs. But it does show the notes you are supposed to be singing and the notes you are actually singing. This screen image shows that I am way off key:

MIDI players

Karaoke files in MIDI format can be found from several sites, usually ending in .kar. Any MIDI player such as Timidity can play such files. However, they do not always show the lyrics synchronised to the melody. Finding MIDI files

There are several sites on the Web offering files in MIDI format.

- MIDIZone
- midiworld
- CoolMIDI
- ElectroFresh
- Freemidi
- Karaoke Version e.g. "House of the Rising Sun" melody on track 4
- MIDaoke e.g. Pink Floyd "Wish you were here" melody on track 2
- Home Musician e.g. the Eages "Hotel California" has melody labelled "Melody" on Clarinet on track 4. kmid plays through lyrics. e.g. Jimi Hendrix "Purple Haze" with melody labelled "Vocals - Choir Ahhs" on track 4. e.g. The Polics "Don't stand so close to me" melody on track 4

- 1001 Midis e.g. 10cc "I'm not in love" has melody on track 4, no lyrics (MIDI only)KAR file format

There is no formal standard for Karaoke MIDI files. There is a widely accepted industry format called the MIDI Karaoke Type 1 file format.

From MIDI Karaoke Frequently Asked Questions

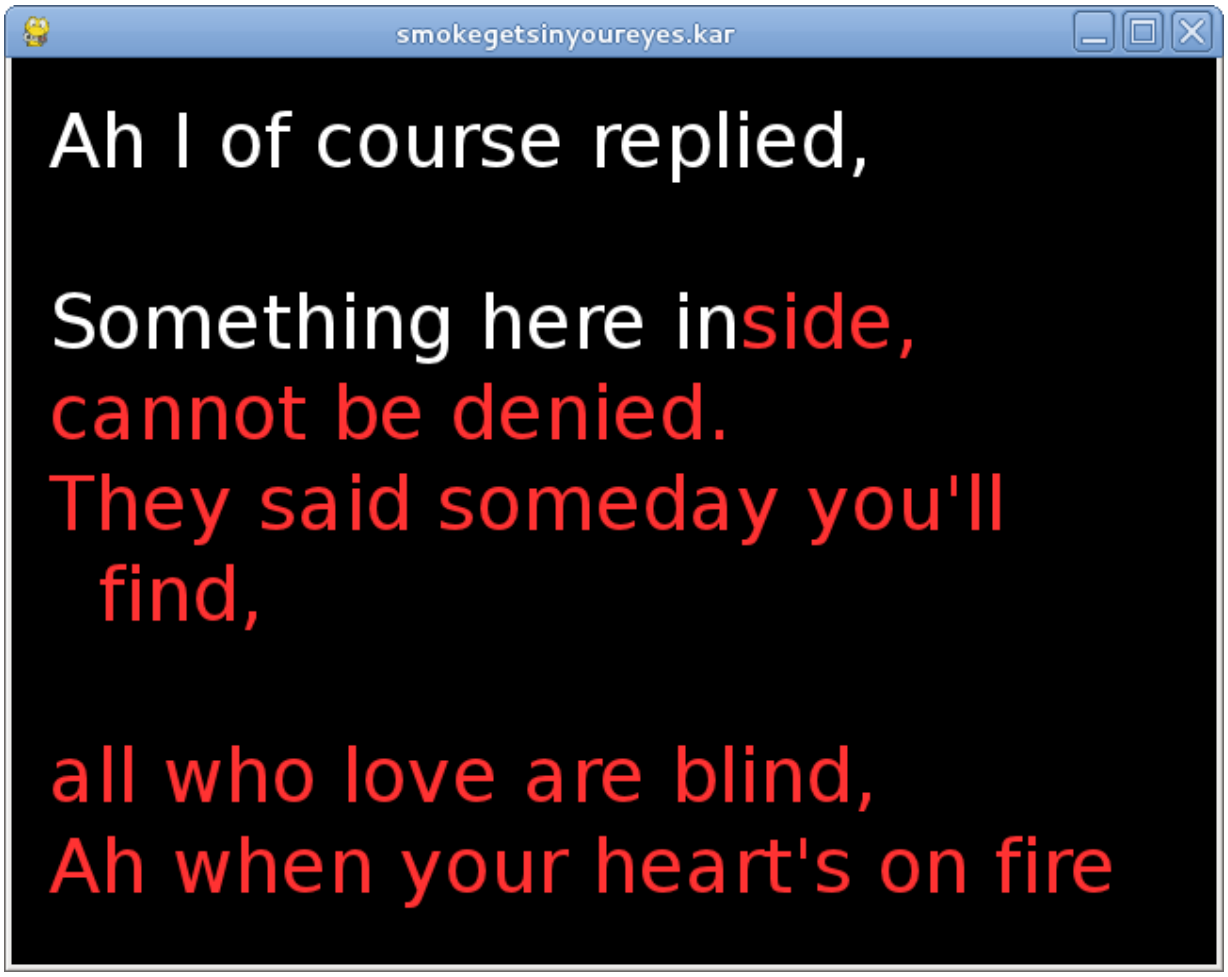
> What is the MIDI Karaoke Type 1 (.KAR) file format? A: A MIDI Karaoke file is a Standard MIDI File type 1 that contains a separate track with lyrics of the song entered as text events. Load one of the MIDI karaoke files into a sequencer to examine the contents of the tracks of the file. The first track contains text events that are used to make the file recognizable as the MIDI Karaoke file. @KMIDI KARAOKE FILE text event is used to for that purpose. The optional text event @V0100 denotes the format version number. Anything starting with @I is any information you want to include in the file.
>
> The second track contains the text meta events for the lyrics of the song. The first event is @LENGL. It identifies the language of the song, in this case English. The next couple of events start with @T which identifies the title of the songs. You can have up to three events like these. The first event should contain the title of the song. Some programs (ex. Soft Karaoke) read this event to get the name of the song to be displayed in the File Open dialog box. The second event usually contains the performer or author of the song. The third event can contain any copyright information or anything else.

> The rest of the second track contains the words of the song. Each event is the syllable that is supposed to be sung at the time of the event. If the text starts with \, it means to clear the screen and show the words at the top of the screen. If the text starts with /, it means to go to the next line.
>
> Important note: There can be only 3 lines per screen in a .kar file for Soft Karaoke being able to play the file correctly. In other words, there can be only two / (forward slashes) beginning each line in a line of lyrics. The next line has to start with \ (back slash).

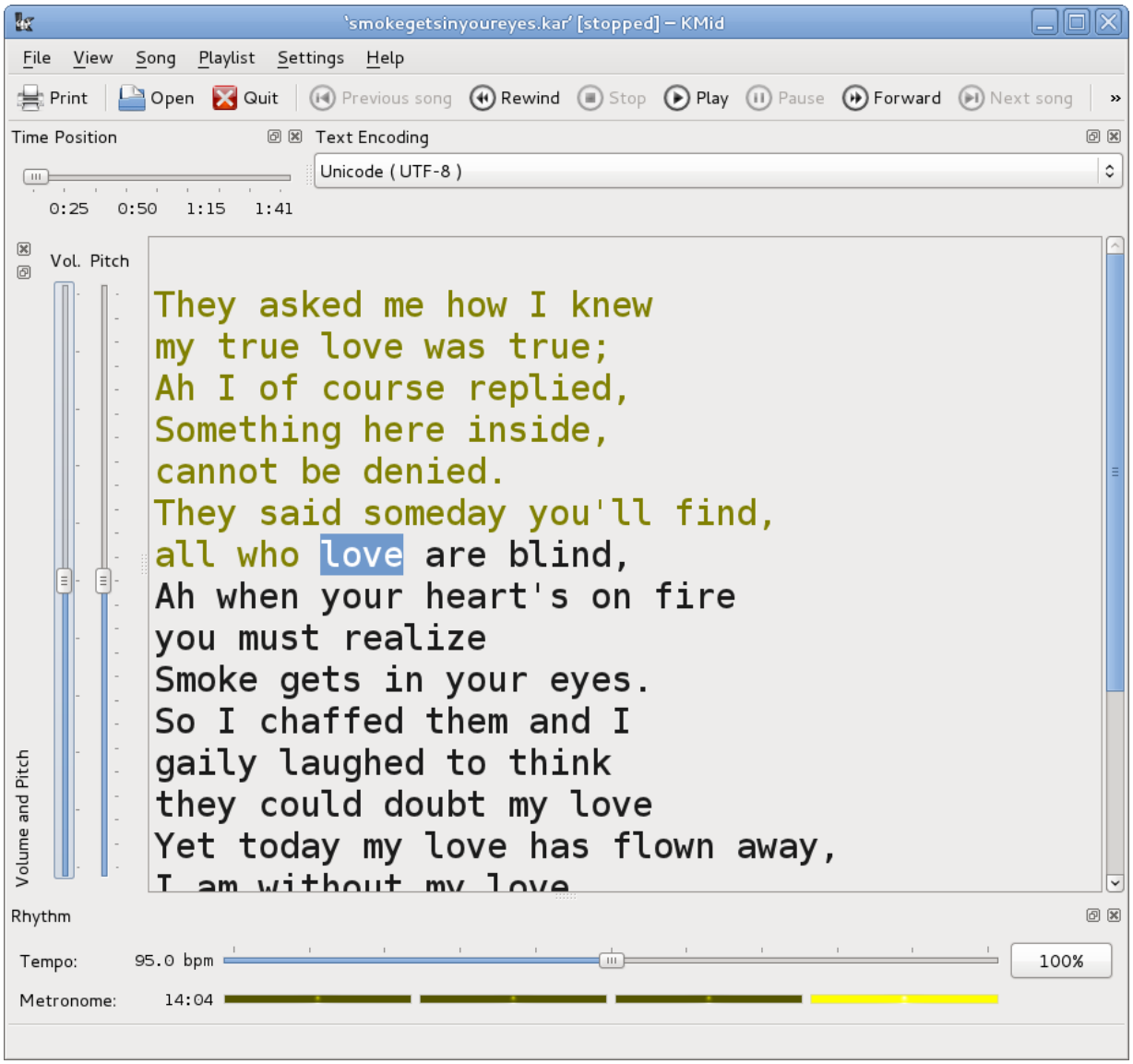
There are several weaknesses in this format:

- The list of possible languages is not specified, only English
- The encoding of text is not specified (e.g. Unicode UTF-8?))
- There is no means of identifying the channel carrying the melodypykaraoke

This is a dedicated Karaoke player written in Python, using a variety of libraries such as Pygame and WxPython. It plays the song and shows where in the lyrics you are. A screen dump of "Smoke gets in your eyes" looks like


smokegetsinyoureyes.kar
Ah I of course replied,
Something here inside,
cannot be denied.
They said someday you'll
find,
all who love are blind,
Ah when your heart's on fire

> PyKaraoke plays the soundtrack and displays the lyrics. It does not act as a proper Karaoke system by also playing the singer's input. But PyKaraoke uses the PulseAudio system, so you can simultaneously play other programs. In particular, you can have PyKaraoke running in one window, while `pa-mic-2-speaker` is running in another. PulseAudio will mix the two output streams and play both sources together. Of course, there will no scoring possible in such a system without extra work. kmid
>
> `kmid` is a KDE-based Karaoke player. It plays the song and shows where in the lyrics you are. A screen dump of "Smoke gets in your eyes" looks like



kmid uses either Timidity or Fluidsynth as MIDI backend.

> `kmid` plays the soundtrack and displays the lyrics. It does not act as a proper Karaoke system by also playing the singer's input. But `kmid` can use the PulseAudio system, so you can simultaneously play other programs. In particular, you can have `kmid` running in one window, while `pa-mic-2-speaker` is running in another. PulseAudio will mix the two output streams and play both sources together. Of course, there will no scoring possible in such a system without extra work. Microphone inputs and reverb effects
>
> Nearly all PCs and laptops have a soundcard to play audio. While nearly all of these also have a microphone input, some do not. For example, my Dell laptop does not; the Raspberry Pi does not; and many Android TV media boxes do not.
>
> Those computers without microphone inputs often have USB ports. They will usually accept USB soundcards, and if the USB has a microphone input then that is recognised.
>
> If you want to support two or more microphones, then you will need the corresponding number of soundcards, or a mixer device. I have seen the Behringer MX-400 MicroMix, a "Four Channel Compact Low Noise Mixer" for US$20, or you can find circuit diagrams on electronics web sites (Google for "circuit diagram for audio mixer").

Reverb is an effect which gives a fuller "body" to the voice by adding (artificial) echoes with different delays. Behringer also make the MIX800 MiniMix which can mix two microphones with reverb effects and also has a passthrough for line input (so you can play the music and control the microphones). (I have no links to Behringer.). A similar unit is the UNIFY K9 Reverb Computer Karaoke Mixer.

DVD players from China often have dual microphone inputs with mix and reverb capabilities. Given that they can cost as little as US$13 (admittedly for 1,000 units :-), it shows that mixing and reverb should not be too costly. My guess is that they use something like the Mitsubishi M65845AFP , "DIGITAL ECHO WITH MICROPHONE MIXING CIRCUIT". The data sheet shows a number of possible configurations, for those who like to build their own.
Conclusion

There are a variety of Karaoke systems, using VCD disks or dedicated systems. MIDI format Karaoke files files can be played using ordinary MIDI software, and there are a couple of Linux Karaoke players.

> Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

> If you like this book, please contribute using PayPal

Or Flattr me:

Buying CD+G or MP3+G files

> There are many sites selling CD+G or MP3+G songs. Just do a Google search. However, the average price per song is about US$3, and if you want to build up a large collection that can become very expensive. Some sites will give discounts for larger volume purchases, but even at US$30 for 100 songs, the expense can be high.
>
> The cheapest site for *really* large collections that I have found so far is Unify mall There you can purchase (for example) 29,000 English CDG songs for US$399. There is a little hiccup in this: if you buy their "UNIFY PCKP1000A 500-2000GB HDD MP4/DVD/CDG PC Karaoke Player" for US$559 you should be able to get these English songs for free! If you want to also get, say, 30,000 Mandarin songs thrown in for free, the economics of building your own Karaoke player become more shaky... We will ignore that issue here - it's your choice!

Converting MP3+G to video files

> The tool `ffmpeg` can merge the audio and video to a single video file by e.g.

```
ffmpeg -i Track1.cdg -i Track1.mp3 -y
Track1.avi
```

> or

```
avconv -i Track1.cdg -i Track1.mp3 test.avi
avconv -i test.avi -c:v libx264 -c:a copy
outputfile.mp4
```

> to create an AVI file containing both video and audio. This can be played by vlc, mplayer, rhythmbox, etc.
>
> There is a program cdg2video . While it is last dated February, 2011, changes in the FFMpeg internals means that it no longer compiles. Even if you fix the obvious changes, there are a huge number of complaints from the C compiler about the use of deprecated FFMpeg functions.

MPEG 4 files

It is becoming common to have Karaoke systems using MPEG 4 video players. These embed all of the information into a video. There is no scoring system with players of these files.

Some rate them as much higher sound quality: "MIDI files sound so cheesy it's sick" . I would suggest it is more an issue with the synthesizer used than the format. Certainly high-end synthesizer manufacturers such as Yamaha would not agree!

MPEG 4 files are certainly larger than the corrsponding MIDI files, and you will need a substantial disk to hold many of them.

There are many sites selling MP4 songs. Just do a Google search. However, the average price per song is about US$3, and if you want to build up a large collection that can become very expensive. Some sites will give discounts for larger volume purchases, but even at US$30 for 100 songs, the expense can be high.

The cheapest site for *really* large collections that I have found so far is Unify mall There you can purchase (for example) 6,200 English MP4 songs for US$199. There is a little hiccup in this: if you buy their "UNIFY PCKP1000A 500-2000GB HDD MP4/DVD/CDG PC Karaoke Player" for

> US$559 you should be able to get these English songs for free! If you want to also get, say, 25,000 Mandarin songs thrown in for free, the economics of building your own Karaoke player become more shaky. We will ignore that issue here - it's your choice!

Karaoke machines

> There are many Karaoke machines which come with a DVD, such as the Malata or Songken systems. In most cases the songs are stored as MIDI files, with the songtrack in one MIDI file and the lyrics in another. Some more recent systems will use WMA files for the soundtrack, and this allows one track to have a vocal supplied and the other without the vocal. Such systems will usually include a scoring mechanism, although the basis for the scoring is not made explicit. The most recent ones are hard-disk based, usually with MP4 files. They do not seem to have a scoring system.
>
> The two systems I own show different characteristics. The Songken MD-388 plays songs from multiple languages, such as Chinese, Korean, English, etc. My wife is Chinese, but I cannot read Chinese characters. There is an Anglicised script called *PinYin* and the Songken shows both the Chinese characters and the PinYin, so I can sing along too. It looks like

> The Malata MDVD-6619 does not show the PinYin when playing Chinese songs. But it does show the notes you are supposed to be singing and the notes you are actually singing. This screen image shows that I am way off key:

000
00344
wants you back again.
That boy isn't

MIDI players

> Karaoke files in MIDI format can be found from several sites, usually ending in .kar. Any MIDI player such as Timidity can play such files. However, they do not always show the lyrics synchronised to the melody.

Finding MIDI files

> There are several sites on the Web offering files in MIDI format.
>
> - MIDIZone
> - midiworld
> - CoolMIDI
> - ElectroFresh
> - Freemidi
> - Karaoke Version e.g. "House of the Rising Sun" melody on track 4
> - MIDaoke e.g. Pink Floyd "Wish you were here" melody on track 2
> - Home Musician e.g. the Eages "Hotel California" has melody labelled "Melody" on Clarinet on track 4. kmid plays through lyrics. e.g. Jimi Hendrix "Purple Haze" with melody labelled "Vocals - Choir Ahhs" on track 4. e.g. The Polics "Don't stand so close to me" melody on track 4
> - 1001 Midis e.g. 10cc "I'm not in love" has melody on track 4, no lyrics (MIDI only)

KAR file format

> There is no formal standard for Karaoke MIDI files. There is a widely accepted industry format called the MIDI Karaoke Type 1 file format.
>
> From MIDI Karaoke Frequently Asked Questions

> What is the MIDI Karaoke Type 1 (.KAR) file format? A: A MIDI Karaoke file is a Standard MIDI File type 1 that contains a separate track with lyrics of the song entered as text events. Load one of the MIDI karaoke files into a sequencer to examine the contents of the tracks of the file. The first track contains text events that are used to make the file recognizable as the MIDI Karaoke file. @KMIDI KARAOKE FILE text event is used to for that purpose. The optional text event @V0100 denotes the format version number. Anything starting with @I is any information you want to include in the file.
>
> The second track contains the text meta events for the lyrics of the song. The first event is @LENGL. It identifies the language of the song, in this case English. The next couple of events start with @T which identifies the title of the songs. You can have up to three events like these. The first event should contain the title of the song. Some programs (ex. Soft Karaoke) read this event to get the name of the song to be displayed in the File Open dialog box. The second event usually contains the performer or author of the song. The third event can contain any copyright information or anything else.

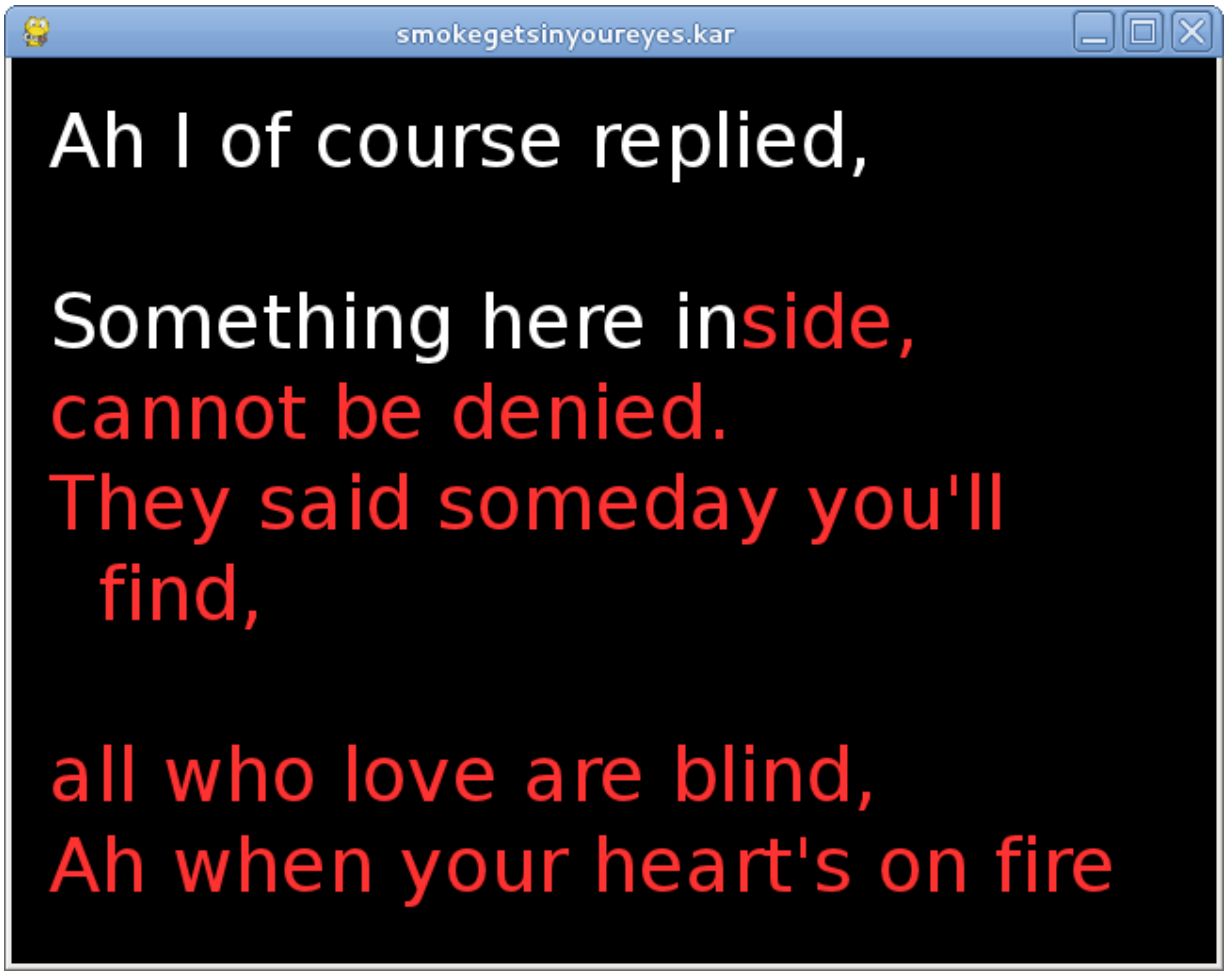
> The rest of the second track contains the words of the song. Each event is the syllable that is supposed to be sung at the time of the event. If the text starts with \, it means to clear the screen and show the words at the top of the screen. If the text starts with /, it means to go to the next line.
>
> Important note: There can be only 3 lines per screen in a .kar file for Soft Karaoke being able to play the file correctly. In other words, there can be only two / (forward slashes) beginning each line in a line of lyrics. The next line has to start with \ (back slash).

There are several weaknesses in this format:

- The list of possible languages is not specified, only English
- The encoding of text is not specified (e.g. Unicode UTF-8?))
- There is no means of identifying the channel carrying the melody

pykaraoke

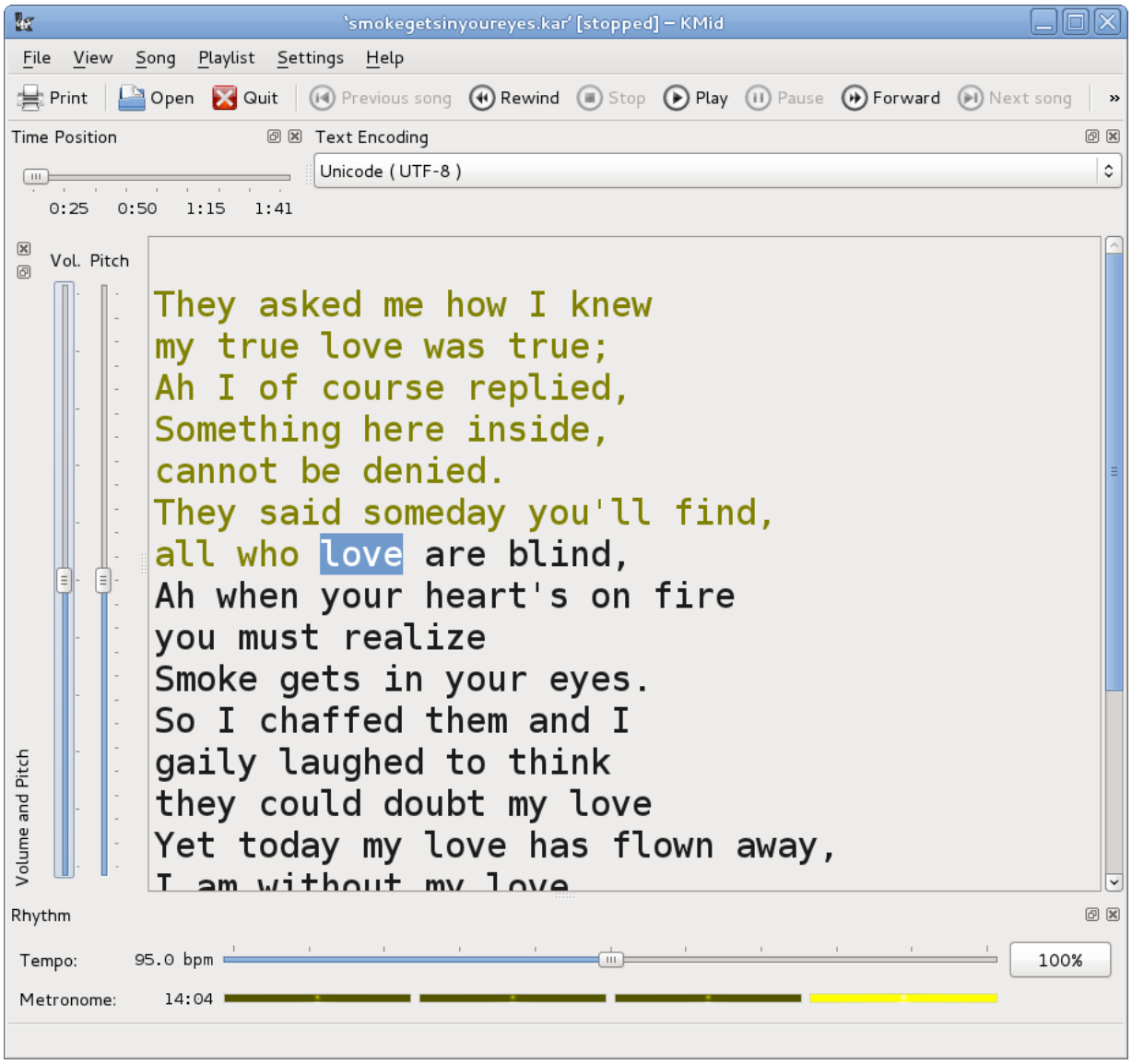
> This is a dedicated Karaoke player written in Python, using a variety of libraries such as Pygame and WxPython. It plays the song and shows where in the lyrics you are. A screen dump of "Smoke gets in your eyes" looks like



> PyKaraoke plays the soundtrack and displays the lyrics. It does not act as a proper Karaoke system by also playing the singer's input. But PyKaraoke uses the PulseAudio system, so you can simultaneously play other programs. In particular, you can have PyKaraoke running in one window, while `pa-mic-2-speaker` is running in another. PulseAudio will mix the two output streams and play both sources together. Of course, there will no scoring possible in such a system without extra work.

kmid

> `kmid` is a KDE-based Karaoke player. It plays the song and shows where in the lyrics you are. A screen dump of "Smoke gets in your eyes" looks like



kmid uses either Timidity or Fluidsynth as MIDI backend.

> `kmid` plays the soundtrack and displays the lyrics. It does not act as a proper Karaoke system by also playing the singer's input. But `kmid` can use the PulseAudio system, so you can simultaneously play other programs. In particular, you can have `kmid` running in one window, while `pa-mic-2-speaker` is running in another. PulseAudio will mix the two output streams and play both sources together. Of course, there will no scoring possible in such a system without extra work.

Microphone inputs and reverb effects

Nearly all PCs and laptops have a soundcard to play audio. While nearly all of these also have a microphone input, some do not. For example, my Dell laptop does not; the Raspberry Pi does not; and many Android TV media boxes do not.

Those computers without microphone inputs often have USB ports. They will usually accept USB soundcards, and if the USB has a microphone input then that is recognised.

If you want to support two or more microphones, then you will need the corresponding number of soundcards, or a mixer device. I have seen the Behringer MX-400 MicroMix, a "Four Channel Compact Low Noise Mixer" for US$20, or you can find circuit diagrams on electronics web sites (Google for "circuit diagram for audio mixer").

Reverb is an effect which gives a fuller "body" to the voice by adding (artificial) echoes with different delays. Behringer also make the MIX800 MiniMix which can mix two microphones with reverb effects and also has a passthrough for line input (so you can play the music and control the microphones). (I have no links to Behringer.). A similar unit is the UNIFY K9 Reverb Computer Karaoke Mixer.

> DVD players from China often have dual microphone inputs with mix and reverb capabilities. Given that they can cost as little as US$13 (admittedly for 1,000 units :-), it shows that mixing and reverb should not be too costly. My guess is that they use something like the Mitsubishi M65845AFP , "DIGITAL ECHO WITH MICROPHONE MIXING CIRCUIT". The data sheet shows a number of possible configurations, for those who like to build their own.

Conclusion

> There are a variety of Karaoke systems, using VCD disks or dedicated systems. MIDI format Karaoke files files can be played using ordinary MIDI software, and there are a couple of Linux Karaoke players.

> Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

> If you like this book, please contribute using PayPal

Or Flattr me:

This chapter explores using Karaoke files in MP3+G format. Files are pulled off a server to a (small) computer attached to a display device (my TV). Files are chosen using a Java Swing application running on Linux or Windows, or by an Android application.

Files

Source files in this chapter are here .

Introduction

In the chapter on User level Karaoke tools we discussed the MP3+G format for Karaoke. Each "song" consists of two files: an MP3 file for the audio and a low-quality CDG file for the video (mainly the lyrics). Often this pair of files are zipped together.

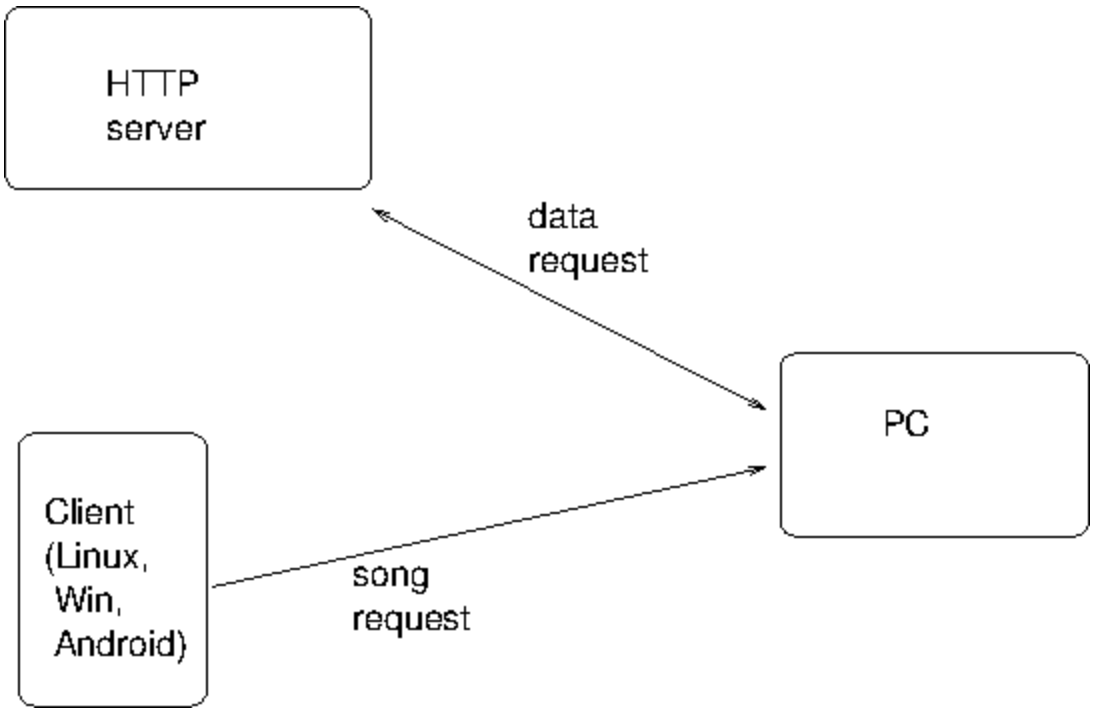
Files can be extracted from CDG Karaoke disks by using `cdrdao` and `cdgrip.py` . They can be played by `vlc` when given the MP3 file as argument - it wil pick up the CDG file from the same directory.

Many people will have built up a sizeable collection of MP3+G songs. In this chapter we consider how to list and play them, along with keeping lists of favourite songs. The chapter just looks at a Java application to perform this, and is really just standard Swing and Android programming. There are no special audio or Karaoke features considered in this chapter.

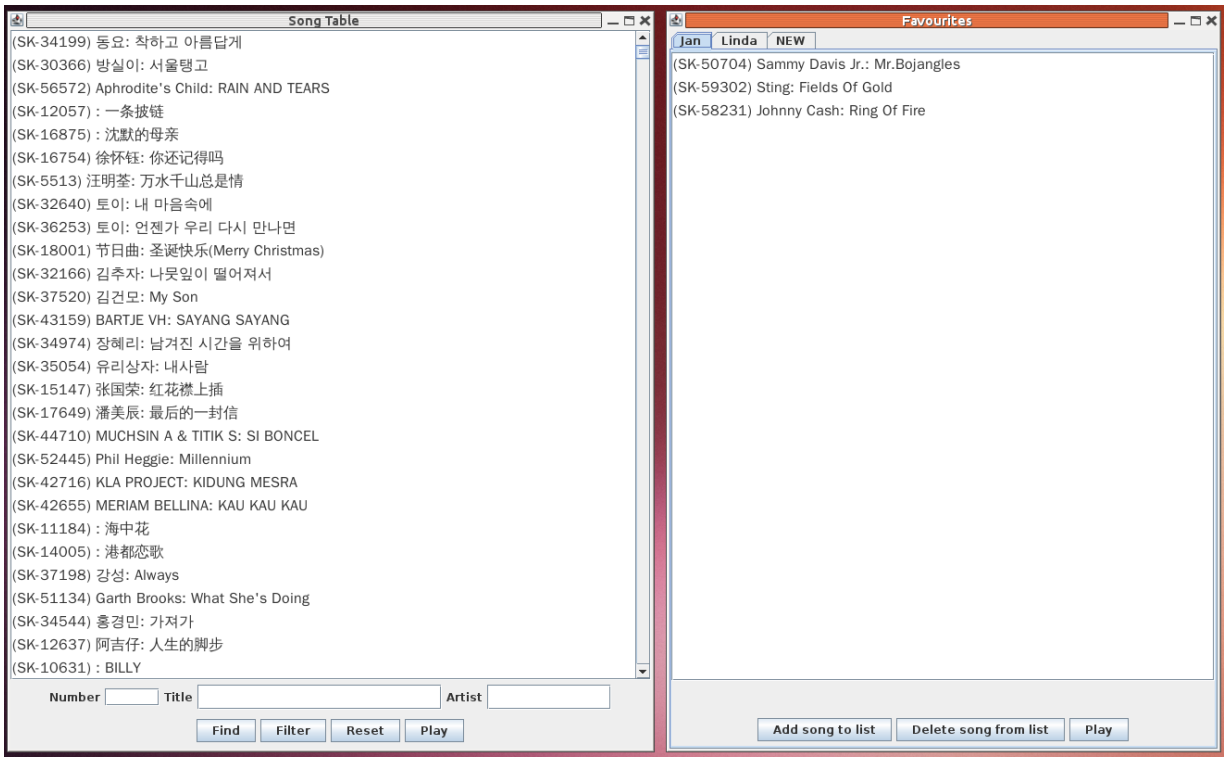
I keep my files on a server. I can access them in many ways on the other computers in the house: SAMBA shares, HTTP downloads, SSH file system (sshfs), etc. Some mechanisms are less portable than others; for example sshfs is not a standard

Windows application and SMB/SAMBA is not a standard Android client. So after getting everything working using sshfs (a no-brainer under standard Linux), I then converted the applications to HTTP access. This has its own wrinkles :-).

The environment looks like

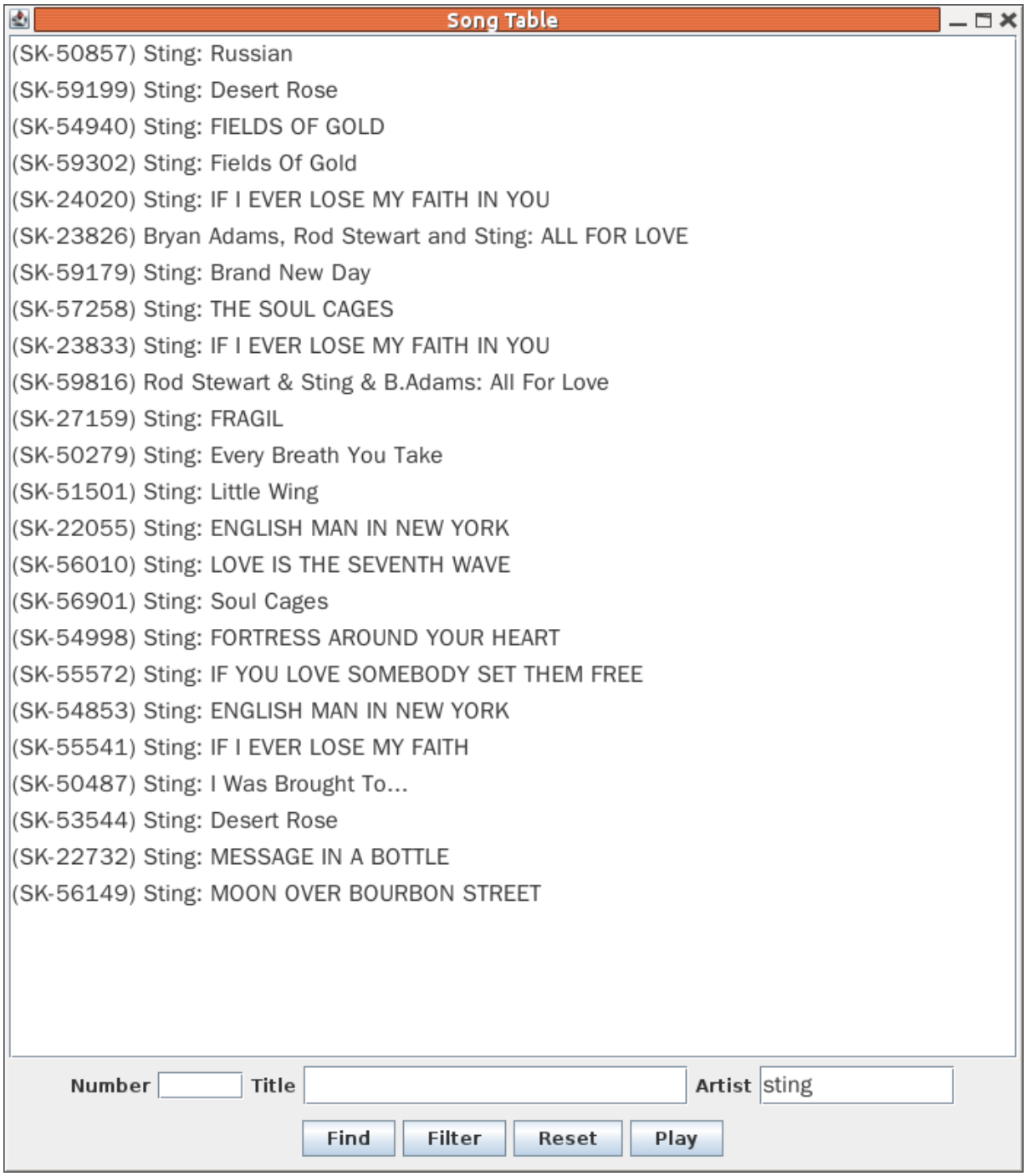


The Java client application for Linux and Windows looks like MP3+G player



This shows the main window of songs and on its right the favourites window for two people, Jan and Linda. The application handles multiple languages - english, korean and chinese are shown.

Filters can be applied to the main song list. For example, filtering on the singer Sting gives Songs by Sting



The Android application looks like XXX

When `Play` is clicked, information about the selection is sent to the media player - currently a CubieBoard2 connected to my hifi/TV. The media computer fetches the files from the HTTP server. Files are played on the media computer using `vlc` as it can handle MP3+G files.

File organisation

If MP3+G songs are ripped from CDG Karaoke disks, then a natural organisation would be to store the files in directories, each directory corresponding to one disk. More structure may be given by grouping the directories by common artist, or by style of music, etc. We just assume a directory structure with music files as leaf nodes. These files are kept on the HTTP server.

I currently have a large number of these files on my server. Information about these needs to be supplied to the clients. After a bit of experimentation a `Vector` of `SongInformation` is created and serialised using Java's object serialisation methods. The serialised file is also kept on the HTTP server. When a client starts up, it gets this file from the HTTP server and deserialises it.

Building this vector means walking the directory tree on the HTTP server and recording information as it goes. Java code to walk directory trees is fairly straightforward. It is a little tedious if you want it to be O/S independent. Java 1.7 introduced mechanisms to make this easier. These belong to the New I/O (NIO) system. The first class of importance is the java.nio.file.Path which "[is] an object that may be used to locate

a file in a file system. It will typically represent a system dependent file path." A string representing a file location in, say, a Linux or a Windows file system can be turned into a `Path` object by

```
Path startingDir =
FileSystems.getDefault().getPath(dirString);
```

Traversal of a file system from a given `path` is done by walking a file tree, calling a node "visitor" at each point. The visitor is a subclass of `SimpleFileVisitor<Path>` and for leaf nodes only you would override the method

```
public FileVisitResult visitFile(Path file,
BasicFileAttributes attr)
```

```
Visitor pf = new Visitor();
Files.walkFileTree(startingDir, pf);
```

A full explanation of this is given in the Java Tutorials on Walking the File Tree . We use this to load all song information from disk into a vector of song paths in `SongTable.java` .

Song information

The information about each song should include its path in the file system, the name of the artist(s), the title of the song and any other useful information. This information has to be pulled out of the the file path of the song. In my current setup, files look like

```
/server/KARAOKE/Sonken/SK-50154 - Crosby,
Stills - Carry On.mp3
```

Each song has a reasonably unique identifier ("SK-50154"), a unique path and an artist and title. Reasonably straight-forward pattern matching code can extract these parts:

```
Path file = ...
String fname = file.getFileName().toString();
if (fname.endsWith(".zip") ||
    fname.endsWith(".mp3")) {
    String root = fname.substring(0,
fname.length()-4);
    String parts[] = root.split(" - ", 3);
    if (parts.length != 3)
        return;

    String index = parts[0];
    String artist = parts[1];
    String title = parts[2];

        SongInformation info = new
SongInformation(file,
                            index,
                            title,
                            artist);
```

(The patterns produced by `cdrip.py` are not quite the same, but the code is easily changed.)

The `SongInformation` class captures this information and also includes methods for pattern matching of a string against the various fields. For example, to check if a title matches,

```
public boolean titleMatch(String pattern) {
    return title.matches("(?i).*" + pattern +
".*");
}
```

This gives a case-independent match using Java regular expression support. See Java Regex Tutorial by Lars Vogel for more details.

The complete `SongInformation` file is

```java
import java.nio.file.Path;
import java.io.Serializable;

public class SongInformation implements
Serializable {


    // Public fields of each song record

    public String path;

    public String index;

    /**
     * song title in Unicode
     */
    public String title;

    /**
     * artist in Unicode
     */
    public String artist;



    public SongInformation(Path path,
```

```
                String index,
                String title,
                String artist) {
    this.path = path.toString();
    this.index = index;
    this.title = title;
    this.artist = artist;
    }

    public String toString() {
    return "(" + index + ") " + artist + ": " +
title;
    }

    public boolean titleMatch(String pattern) {
    return title.matches("(?i).*" + pattern +
".*");
    }

    public boolean artistMatch(String pattern)
{
    return artist.matches("(?i).*" + pattern +
".*");
    }

    public boolean numberMatch(String pattern)
{
```

```
    return index.equals(pattern);
    }
}
```

Song table

The `SongTable` builds up a vector of `SongInformation` objects by traversing the file tree.

If there are many songs (say, in the thousands) this can lead to a slow startup time. To reduce this, once a table is loaded, it is saved to disk as a persistent object by writing it to an `ObjectOutputStream`. The next time the program is started, an attempt is made to read it back from this using an `ObjectInputStream`. Note that we do *not* use the Java Persistence API - designed for J2EE, it is too heavyweight for our purpose here.

The `SongTable` also includes code to build smaller song tables based on matches between patterns and the title (or artist or number). It can search for matches between a pattern and a song and build a new table based on the matches. It contains a pointer to the original table for restoration later. This allows searches for patterns to use the same data structure.

The code for `SongTable` is

```
import java.util.Vector;
import java.io.FileInputStream;
import java.io.*;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.SimpleFileVisitor;
import java.nio.file.FileVisitResult;
import java.nio.file.FileSystems;
import java.nio.file.attribute.*;


class Visitor
    extends SimpleFileVisitor<Path> {

    private Vector<SongInformation> songs;

    public Visitor(Vector<SongInformation>
songs) {
    this.songs = songs;
    }

    @Override
    public FileVisitResult visitFile(Path file,
```

```
BasicFileAttributes attr) {
    if (attr.isRegularFile()) {
        String fname =
file.getFileName().toString();
        //System.out.println("Regular file " +
fname);
        if (fname.endsWith(".zip") ||
        fname.endsWith(".mp3") ||
        fname.endsWith(".kar")) {
        String root = fname.substring(0,
fname.length()-4);
        //System.err.println(" root " + root);
        String parts[] = root.split(" - ", 3);
        if (parts.length != 3)
            return
java.nio.file.FileVisitResult.CONTINUE;

        String index = parts[0];
        String artist = parts[1];
        String title = parts[2];

        SongInformation info = new
SongInformation(file,
                                index,
                                title,
                                artist);
        songs.add(info);
```

```
        }
    }

        return
java.nio.file.FileVisitResult.CONTINUE;
    }
}

public class SongTable {

    private static final String SONG_INFO_ROOT
= "/server/KARAOKE/KARAOKE/";

    private static Vector<SongInformation>
allSongs;

    public Vector<SongInformation> songs =
    new Vector<SongInformation>  ();

    public static long[] langCount = new
long[0x23];

    public SongTable(Vector<SongInformation>
songs) {
    this.songs = songs;
    }
```

```
    public SongTable(String[] args) throws
java.io.IOException,

java.io.FileNotFoundException {
    if (args.length >= 1) {
        System.err.println("Loading from " +
args[0]);
        loadTableFromSource(args[0]);
        saveTableToStore();
    } else {
        loadTableFromStore();
    }
    }

    private boolean loadTableFromStore() {
    try {
        /*
        String userHome =
System.getProperty("user.home");
        Path storePath =
FileSystems.getDefault().getPath(userHome,
                                    ".karaoke",
                                    "SongStore");

        File storeFile = storePath.toFile();
        */
        File storeFile = new
```

```
File("/server/KARAOKE/SongStore");

        FileInputStream in = new
FileInputStream(storeFile);
        ObjectInputStream is = new
ObjectInputStream(in);
        songs = (Vector<SongInformation>)
is.readObject();
        in.close();
    } catch(Exception e) {
        System.err.println("Can't load store
file " + e.toString());
        return false;
    }
    return true;
    }

    private void saveTableToStore() {
    try {
        /*
        String userHome =
System.getProperty("user.home");
        Path storePath =
FileSystems.getDefault().getPath(userHome,
                                    ".karaoke",
                                    "SongStore");
        File storeFile = storePath.toFile();
```

```
        */
        File storeFile = new
File("/server/KARAOKE/SongStore");
        FileOutputStream out = new
FileOutputStream(storeFile);
        ObjectOutputStream os = new
ObjectOutputStream(out);
        os.writeObject(songs);
        os.flush();
        out.close();
    } catch(Exception e) {
        System.err.println("Can't save store
file " + e.toString());
    }
    }

    private void loadTableFromSource(String
dir) throws java.io.IOException,
                  java.io.FileNotFoundException
{

    Path startingDir =
FileSystems.getDefault().getPath(dir);
    Visitor pf = new Visitor(songs);
    Files.walkFileTree(startingDir, pf);
    }
```

```
    public java.util.Iterator<SongInformation>
iterator() {
    return songs.iterator();
    }

    public SongTable titleMatches( String
pattern) {
    Vector<SongInformation> matchSongs =
        new Vector<SongInformation>  ();

    for (SongInformation song: songs) {
        if (song.titleMatch(pattern)) {
        matchSongs.add(song);
        }
    }
    return new SongTable(matchSongs);
    }

     public SongTable artistMatches( String
pattern) {
    Vector<SongInformation> matchSongs =
        new Vector<SongInformation>  ();

    for (SongInformation song: songs) {
        if (song.artistMatch(pattern)) {
        matchSongs.add(song);
        }
```

```
    }
    return new SongTable(matchSongs);
    }

    public SongTable numberMatches( String
pattern) {
    Vector<SongInformation> matchSongs =
        new Vector<SongInformation>  ();

    for (SongInformation song: songs) {
        if (song.numberMatch(pattern)) {
        matchSongs.add(song);
        }
    }
    return new SongTable(matchSongs);
    }

    public String toString() {
    StringBuffer buf = new StringBuffer();
    for (SongInformation song: songs) {
        buf.append(song.toString() + "\n");
    }
    return buf.toString();
    }

    public static void main(String[] args) {
    // for testing
```

```
    SongTable songs = null;
    try {
        songs = new SongTable(new String[]
{SONG_INFO_ROOT});
    } catch(Exception e) {
        System.err.println(e.toString());
        System.exit(1);
    }

    System.out.println(songs.artistMatches("Tom
Jones").toString());

    System.exit(0);
    }
}
```

Favourites

I've built this system for my home environment where I have a regular group of friends visiting. We each have our favourite songs to sing and so we have made up lists on scraps of paper which get lost, have wine spilt on them, etc. So this system includes favourite lists of songs.

Each favourites list is essentially just another `SongTable`. But I have put a `JList` around the table to display it. The `JList` uses a `DefaultListModel`, and the constructor loads a song table into this list by iterating through the table and adding elements

```
int n = 0;
    java.util.Iterator<SongInformation> iter =
favouriteSongs.iterator();
    while(iter.hasNext()) {
        model.add(n++, iter.next());
    }
```

Other Swing code adds three buttons along the bottom:

- Add song to list

- Delete song from list
- Play song

Adding a song to the list means taking the selected item from the main song table and adding it to this table. The main table is passed into the constructor and just kept for the purpose of getting its selection. The selected object is added to both the Swing `JList` and to the favourites `SongTable`.

"Playing a song" is done in a simple way: the full path to the song is written to standard output, newline terminated. Another program in a pipeline can then pick this up - see later.

Favourites aren't much good if they don't persist from one day to the next! So the same object storage method as before is used as with the full song table. Each favourites file is saved on each change. There are some Linux/Unix dependencies here, in that application information is stored in a subdirectory beginning with a "." in the user's home directory.

The code for `Favourites` is

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
import javax.swing.*;
import javax.swing.SwingUtilities;
import java.util.regex.*;
import java.io.*;
import java.nio.file.FileSystems;
import java.nio.file.*;

public class Favourites extends JPanel {
    private DefaultListModel model = new
DefaultListModel();
    private JList list;

    // whose favoutites these are
    private String user;

    // songs in this favourites list
    private final SongTable favouriteSongs;

    // pointer back to main song table list
    private final SongTableSwing songTable;

    // This font displays Asian and European
characters.
```

```
    // It should be in your distro.
    // Fonts displaying all Unicode are
zysong.ttf and Cyberbit.ttf
    // See
http://unicode.org/resources/fonts.html
    private Font font = new Font("WenQuanYi Zen
Hei", Font.PLAIN, 16);

    private int findIndex = -1;

    public Favourites(final SongTableSwing
songTable,
                final SongTable favouriteSongs,
                String user) {
    this.songTable = songTable;
    this.favouriteSongs = favouriteSongs;
    this.user = user;

    if (font == null) {
        System.err.println("Can't find font");
    }

    int n = 0;
    java.util.Iterator<SongInformation> iter =
favouriteSongs.iterator();
    while(iter.hasNext()) {
        model.add(n++, iter.next());
```

```
    }

    BorderLayout mgr = new BorderLayout();

    list = new JList(model);
    list.setFont(font);
    JScrollPane scrollPane = new
JScrollPane(list);

    setLayout(mgr);
    add(scrollPane, BorderLayout.CENTER);

    JPanel bottomPanel = new JPanel();
    bottomPanel.setLayout(new GridLayout(2,
1));
    add(bottomPanel, BorderLayout.SOUTH);

    JPanel searchPanel = new JPanel();
    bottomPanel.add(searchPanel);
    searchPanel.setLayout(new FlowLayout());

    JPanel buttonPanel = new JPanel();
    bottomPanel.add(buttonPanel);
    buttonPanel.setLayout(new FlowLayout());

    JButton addSong = new JButton("Add song to
list");
```

```
    JButton deleteSong = new JButton("Delete
song from list");
    JButton play = new JButton("Play");

    buttonPanel.add(addSong);
    buttonPanel.add(deleteSong);
    buttonPanel.add(play);

    play.addActionListener(new ActionListener()
{
        public void actionPerformed(ActionEvent
e) {
            playSong();
        }
        });

    deleteSong.addActionListener(new
ActionListener() {
        public void actionPerformed(ActionEvent
e) {
            SongInformation song =
(SongInformation) list.getSelectedValue();
            model.removeElement(song);
            favouriteSongs.songs.remove(song);
            saveToStore();
        }
        });
```

```
    addSong.addActionListener(new
ActionListener() {
        public void actionPerformed(ActionEvent
e) {
            SongInformation song =
songTable.getSelection();
            model.addElement(song);
            favouriteSongs.songs.add(song);
            saveToStore();
        }
        });
     }

    private void saveToStore() {
    try {
        /*
        String userHome =
System.getProperty("user.home");
        Path storePath =
FileSystems.getDefault().getPath(userHome,
                                 ".karaoke",
                                 "favourites",
                                 user);
        File storeFile = storePath.toFile();
        */
        File storeFile = new
```

```
File("/server/KARAOKE/favourites/" + user);
        FileOutputStream out = new
FileOutputStream(storeFile);
        ObjectOutputStream os = new
ObjectOutputStream(out);
        os.writeObject(favouriteSongs.songs);
        os.flush();
        out.close();
    } catch(Exception e) {
        System.err.println("Can't save
favourites file " + e.toString());
    }
    }



    /**
     * "play" a song by printing its file path
to standard out.
     * Can be used in a pipeline this way
     */
    public void playSong() {
    SongInformation song = (SongInformation)
list.getSelectedValue();
    if (song == null) {
        return;
    }
    System.out.println(song.path.toString());
```

```
    }


    class SongInformationRenderer extends
JLabel implements ListCellRenderer {

    public Component
getListCellRendererComponent(
                                JList list,
                                Object value,
                                int index,
                                boolean
isSelected,
                                boolean
cellHasFocus) {
        setText(value.toString());
        return this;
    }
    }
}
```

All favourites

There's nothing special here. It just loads the tables for each person and builds a `Favourites` object which it places in a `JTabbedPane`. It also adds in a "NEW" tab to add additional users.

The code for `AllFavourites` is

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.Vector;
import java.nio.file.*;
import java.io.*;

public class AllFavourites extends JTabbedPane
{
    private SongTableSwing songTable;

    public AllFavourites(SongTableSwing
songTable) {
    this.songTable = songTable;

    loadFavourites();

    NewPanel newP = new NewPanel(this);
    addTab("NEW", null, newP);
    }

    private void loadFavourites() {
    String userHome =
System.getProperty("user.home");
    /*
    Path favouritesPath =
```

```
FileSystems.getDefault().getPath(userHome,
                                 ".karaoke",
                                 "favourites");
    */
    Path favouritesPath =
FileSystems.getDefault().getPath("/server/KARAOKE

    try {
        DirectoryStream<Path> stream =

Files.newDirectoryStream(favouritesPath);
        for (Path entry: stream) {
        int nelmts = entry.getNameCount();
        Path last = entry.subpath(nelmts-1,
nelmts);
        System.err.println("Favourite: " +
last.toString());
        File storeFile = entry.toFile();

        FileInputStream in = new
FileInputStream(storeFile);
        ObjectInputStream is = new
ObjectInputStream(in);
        Vector<SongInformation> favouriteSongs
=
            (Vector<SongInformation>)
is.readObject();
```

```
        in.close();
        for (SongInformation s: favouriteSongs)
{
            System.err.println("Fav: " +
s.toString());
        }

        SongTable favouriteSongsTable = new
SongTable(favouriteSongs);
        Favourites f = new
Favourites(songTable,
                            favouriteSongsTable,
                            last.toString());
        addTab(last.toString(), null, f,
last.toString());
        System.err.println("Loaded favs " +
last.toString());
        }
    } catch(Exception e) {
        System.err.println(e.toString());
    }
    }

    class NewPanel extends JPanel {
    private JTabbedPane pane;

    public NewPanel(final JTabbedPane pane) {
```

```
        this.pane = pane;

        setLayout(new FlowLayout());
        JLabel nameLabel = new JLabel("Name of
new person");
        final JTextField nameField = new
JTextField(10);
        add(nameLabel);
        add(nameField);

        nameField.addActionListener(new
ActionListener(){
            public void
actionPerformed(ActionEvent e){
            String name = nameField.getText();

            SongTable songs = new SongTable(new
Vector<SongInformation>());
            Favourites favs = new
Favourites(songTable, songs, name);

            pane.addTab(name, null, favs);
            }});

    }
    }
}
```

Swing song table

This is mainly code to get the different song tables loaded and to buld the Swing interface. It also filters the showing table based on patterns matched. The originally loaded table is kept for restoration and patching matching. The code for `SongTableSwing` is

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
import javax.swing.*;
import javax.swing.SwingUtilities;
import java.util.regex.*;
import java.io.*;

public class SongTableSwing extends JPanel {
   private DefaultListModel model = new
DefaultListModel();
    private JList list;
    private static SongTable allSongs;

    private JTextField numberField;
    private JTextField langField;
    private JTextField titleField;
    private JTextField artistField;

    // This font displays Asian and European
characters.
    // It should be in your distro.
    // Fonts displaying all Unicode are
zysong.ttf and Cyberbit.ttf
    // See
http://unicode.org/resources/fonts.html
```

```
    private Font font = new Font("WenQuanYi Zen
Hei", Font.PLAIN, 16);
    // font = new Font("Bitstream Cyberbit",
Font.PLAIN, 16);

    private int findIndex = -1;

    /**
     * Describe <code>main</code> method here.
     *
     * @param args a <code>String</code> value
     */
    public static final void main(final
String[] args) {
    if (args.length >= 1 &&
        args[0].startsWith("-h")) {
        System.err.println("Usage: java
SongTableSwing [song directory]");
        System.exit(0);
    }

    allSongs = null;
    try {
        allSongs = new SongTable(args);
    } catch(Exception e) {
        System.err.println(e.toString());
        System.exit(1);
```

```
    }

    JFrame frame = new JFrame();
    frame.setTitle("Song Table");
    frame.setSize(700, 800);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLO


    SongTableSwing panel = new
SongTableSwing(allSongs);
    frame.getContentPane().add(panel);

    frame.setVisible(true);

    JFrame favourites = new JFrame();
    favourites.setTitle("Favourites");
    favourites.setSize(600, 800);

favourites.setDefaultCloseOperation(JFrame.EXIT_O


    AllFavourites lists = new
AllFavourites(panel);
    favourites.getContentPane().add(lists);

    favourites.setVisible(true);
```

```
    }

    public SongTableSwing(SongTable songs) {

    if (font == null) {
        System.err.println("Can't fnd font");
    }

    int n = 0;
    java.util.Iterator<SongInformation> iter =
songs.iterator();
    while(iter.hasNext()) {
        model.add(n++, iter.next());
        // model.add(n++,
iter.next().toString());
    }

    BorderLayout mgr = new BorderLayout();

    list = new JList(model);
    // list = new JList(songs);
    list.setFont(font);
    JScrollPane scrollPane = new
JScrollPane(list);

    // Support DnD
```

```
    list.setDragEnabled(true);

    setLayout(mgr);
    add(scrollPane, BorderLayout.CENTER);

    JPanel bottomPanel = new JPanel();
    bottomPanel.setLayout(new GridLayout(2,
1));
    add(bottomPanel, BorderLayout.SOUTH);

    JPanel searchPanel = new JPanel();
    bottomPanel.add(searchPanel);
    searchPanel.setLayout(new FlowLayout());

    JLabel numberLabel = new JLabel("Number");
    numberField = new JTextField(5);

    JLabel langLabel = new JLabel("Language");
    langField = new JTextField(8);

    JLabel titleLabel = new JLabel("Title");
    titleField = new JTextField(20);
    titleField.setFont(font);

    JLabel artistLabel = new JLabel("Artist");
    artistField = new JTextField(10);
    artistField.setFont(font);
```

```
    searchPanel.add(numberLabel);
    searchPanel.add(numberField);
    searchPanel.add(titleLabel);
    searchPanel.add(titleField);
    searchPanel.add(artistLabel);
    searchPanel.add(artistField);


titleField.getDocument().addDocumentListener(new
 DocumentListener() {
        public void changedUpdate(DocumentEvent
e) {
            // rest find to -1 to restart any
find searches
            findIndex = -1;
            // System.out.println("reset find
index");
        }
        public void insertUpdate(DocumentEvent
e) {
            findIndex = -1;
            // System.out.println("reset insert
find index");
        }
        public void removeUpdate(DocumentEvent
e) {
```

```
            findIndex = -1;
            // System.out.println("reset remove
find index");
        }
        }
        );

artistField.getDocument().addDocumentListener(new
 DocumentListener() {
        public void changedUpdate(DocumentEvent
e) {
            // rest find to -1 to restart any
find searches
            findIndex = -1;
            // System.out.println("reset insert
find index");
        }
        public void insertUpdate(DocumentEvent
e) {
            findIndex = -1;
            // System.out.println("reset insert
find index");
        }
        public void removeUpdate(DocumentEvent
e) {
            findIndex = -1;
            // System.out.println("reset insert
```

```
find index");
        }
        }
        );

    titleField.addActionListener(new
ActionListener(){
                public void
actionPerformed(ActionEvent e){
            filterSongs();
                }});
    artistField.addActionListener(new
ActionListener(){
                public void
actionPerformed(ActionEvent e){
            filterSongs();
                }});

    JPanel buttonPanel = new JPanel();
    bottomPanel.add(buttonPanel);
    buttonPanel.setLayout(new FlowLayout());

    JButton find = new JButton("Find");
    JButton filter = new JButton("Filter");
    JButton reset = new JButton("Reset");
    JButton play = new JButton("Play");
    buttonPanel.add(find);
```

```
    buttonPanel.add(filter);
    buttonPanel.add(reset);
    buttonPanel.add(play);

    find.addActionListener(new ActionListener()
{
        public void actionPerformed(ActionEvent
e) {
            findSong();
        }
        });

    filter.addActionListener(new
ActionListener() {
        public void actionPerformed(ActionEvent
e) {
            filterSongs();
        }
        });

    reset.addActionListener(new
ActionListener() {
        public void actionPerformed(ActionEvent
e) {
            resetSongs();
        }
        });
```

```
    play.addActionListener(new ActionListener()
{
        public void actionPerformed(ActionEvent
e) {
            playSong();
        }
        });

     }

    public void findSong() {
    String number = numberField.getText();
    String language = langField.getText();
    String title = titleField.getText();
    String artist = artistField.getText();

    if (number.length() != 0) {
        return;
    }

    for (int n = findIndex + 1; n <
model.getSize(); n++) {
        SongInformation info =
(SongInformation) model.getElementAt(n);

        if ((title.length() != 0) &&
```

```
(artist.length() != 0)) {
        if (info.titleMatch(title) &&
info.artistMatch(artist)) {
            findIndex = n;
            list.setSelectedIndex(n);
            list.ensureIndexIsVisible(n);
            break;
        }
        } else {
        if ((title.length() != 0) &&
info.titleMatch(title)) {
            findIndex = n;
            list.setSelectedIndex(n);
            list.ensureIndexIsVisible(n);
            break;
        } else if ((artist.length() != 0) &&
info.artistMatch(artist)) {
            findIndex = n;
            list.setSelectedIndex(n);
            list.ensureIndexIsVisible(n);
            break;

        }
        }

    }
    }
```

```
    public void filterSongs() {
    String title = titleField.getText();
    String artist = artistField.getText();
    String number = numberField.getText();
    SongTable filteredSongs = allSongs;

    if (allSongs == null) {
        return;
    }

    if (title.length() != 0) {
        filteredSongs =
filteredSongs.titleMatches(title);
    }
    if (artist.length() != 0) {
        filteredSongs =
filteredSongs.artistMatches(artist);
    }
    if (number.length() != 0) {
        filteredSongs =
filteredSongs.numberMatches(number);
    }

    model.clear();
    int n = 0;
    java.util.Iterator<SongInformation> iter =
```

```
filteredSongs.iterator();
    while(iter.hasNext()) {
        model.add(n++, iter.next());
    }
    }

    public void resetSongs() {
    artistField.setText("");
    titleField.setText("");
    model.clear();
    int n = 0;
    java.util.Iterator<SongInformation> iter =
allSongs.iterator();
    while(iter.hasNext()) {
        model.add(n++, iter.next());
    }
    }
    /**
     * "play" a song by printing its file path
to standard out.
     * Can be used in a pipeline this way
     */
    public void playSong() {
    SongInformation song = (SongInformation)
list.getSelectedValue();
    if (song == null) {
        return;
```

```
    }
    System.out.println(song.path);
    }

    public SongInformation getSelection() {
    return (SongInformation)
(list.getSelectedValue());
    }

    class SongInformationRenderer extends
JLabel implements ListCellRenderer {

    public Component
getListCellRendererComponent(
                                JList list,
                                Object value,
                                int index,
                                boolean
isSelected,
                                boolean
cellHasFocus) {
        setText(value.toString());
        return this;
    }
    }
}
```

Playing songs

Whenever a song is "played" its file path is written to standard output. This makes it suitable for use in a pipeline such as

```
VLC_OPTS="--play-and-exit --fullscreen"

java  SongTableSwing |
while read line
do
        if expr match "$line" ".*mp3"
        then
                vlc $VLC_OPTS "$line"
        elif expr match "$line" ".*zip"
        then
                rm -f /tmp/karaoke/*
                unzip -d /tmp/karaoke "$line"
                vlc $VLC_OPTS
/tmp/karaoke/*.mp3
        fi
done
```

VLC

VLC is an immensely flexible media player. It relies on a large set of plugins to enhance its basic core functionality. We saw in an earlier chapter that if a directory contains both an MP3 and a CDG file with the same basename then by asking it to play the MP3 file it will also show the CDG video.

Common expectations of Karaoke players are that you can adjust the speed and pitch. Currently VLC cannot adjust pitch, but it does have a plugin to adjust speed (while keeping the pitch unchanged). This plugin can be accessed by the Lua interface to VLC. Once set up, you can send commands such as

```
rate 1.04
```

across standard input from the process that started VLC (such as a command line shell). This will change the speed and leave the pitch unchanged.

Setting up VLC to accept Lua commands from stdin can be done by the command options

```
vlc -I luaintf --lua-intf cli ...
```

Note that this takes away the standard GUI controls (menus, etc) and controls VLC from stdin only.

At present, it is not simple to add pitch control to VLC. Take a deep breath:

- Turn off PulseAudio and start Jack
- Run `jack-rack` and install the `TAP_pitch` filter
- Run VLC with Jack output
- Using `qjackctl` hook VLC to output through `jack-rack`, which outputs to system
- Control pitch through the `jack-rack` GUI

Playing songs across the network

I actually want to play songs from my server disk to a Raspberry Pi or CubieBoard connected to my TV, and control the play from a netbook sitting on my lap. (Later I will try to get Android code running to do the same.). This is a distributed system.

Mounting server files on a computer is simple: you can use NFS, Samba, ... I am currently using `sshfs` as in

```
sshfs -o idmap=user -o rw -o allow_other
newmarch@192.168.1.101:/home/httpd/html /server
```

For remote access/control I replace the `run` command of the last section by a TCP client/server. On the client, controlling the player, I have

```
java SongTableSwing | client 192.168.1.7
```

while on the (Raspberry Pi/CubieBoard) server I run

```bash
#!/bin/bash
set -x
VLC_OPTS="--play-and-exit -f"

server |
while read line
do
    if expr match "$line" ".*mp3"
    then
        vlc $VLC_OPTS "$line"
    elif expr match "$line" ".*zip"
    then
        rm -f /tmp/karaoke/*
        unzip -d /tmp/karaoke "$line"
        vlc $VLC_OPTS /tmp/karaoke/*.mp3
    fi
done
```

The client/server files are just standard TCP files. The client reads a newline-terminated string from standard input and writes it to the server, and the server prints the same line to standard output. Here is `client.c`:

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>

#define SIZE 1024
char buf[SIZE];
#define PORT 13000
int main(int argc, char *argv[]) {
    int sockfd;
    int nread;
    struct sockaddr_in serv_addr;
    if (argc != 2) {
    fprintf(stderr, "usage: %s IPaddr\n",
argv[0]);
    exit(1);
    }


    while (fgets(buf, SIZE , stdin) != NULL) {
    /* create endpoint */
    if ((sockfd = socket(AF_INET, SOCK_STREAM,
0)) < 0) {
        perror(NULL); exit(2);
```

```
    }
    /* connect to server */
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr =
inet_addr(argv[1]);
    serv_addr.sin_port = htons(PORT);

    while (connect(sockfd,
                   (struct sockaddr *) &serv_addr,
                   sizeof(serv_addr)) < 0) {
        /* allow for timesouts etc */
        perror(NULL);
        sleep(1);
    }

    printf("%s", buf);
    nread = strlen(buf);
    /* transfer data and quit */
    write(sockfd, buf, nread);
    close(sockfd);
    }
}
```

and here is `server.c`

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <signal.h>

#define SIZE 1024
char buf[SIZE];
#define TIME_PORT 13000

int sockfd, client_sockfd;

void intHandler(int dummy) {
    close(client_sockfd);
    close(sockfd);
    exit(1);
}

int main(int argc, char *argv[]) {
    int sockfd, client_sockfd;
    int nread, len;
    struct sockaddr_in serv_addr, client_addr;
    time_t t;

    signal(SIGINT, intHandler);
```

```
    /* create endpoint */
    if ((sockfd = socket(AF_INET, SOCK_STREAM,
0)) < 0) {
    perror(NULL); exit(2);
    }
    /* bind address */
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr =
htonl(INADDR_ANY);
    serv_addr.sin_port = htons(TIME_PORT);
    if (bind(sockfd,
         (struct sockaddr *) &serv_addr,
         sizeof(serv_addr)) < 0) {
    perror(NULL); exit(3);
    }
    /* specify queue */
    listen(sockfd, 5);
    for (;;) {
    len = sizeof(client_addr);
    client_sockfd = accept(sockfd,
                    (struct sockaddr *)
&client_addr,
                    &len);
    if (client_sockfd == -1) {
        perror(NULL); continue;
    }
```

```
    while ((nread = read(client_sockfd, buf,
SIZE-1)) > 0) {
        buf[nread] = '\0';
        fputs(buf, stdout);
        fflush(stdout);
    }
    close(client_sockfd);
    }
}
```

Conclusion

This chapter has built a player for MP3+G files.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

This chapter is about getting the information off my Sonken Karaoke DVD so that I can start writing programs to play the songs. It is not directly involved in playing sound under Linux and can be skipped.

Introduction

I have two Karaoke players, a Sonken MD-388 and a Malata MDVD-6619. Between the two of them they have all the features I think I need from Karaoke players. These are

- Selecting and playing tunes (of course!)
- Huge range of both Chinese and English songs - my wife is Chinese and I am English
- Both Mandarin and Pinyin shown for the Chinese songs so that I can sing along too
- The notes of the melody displayed along with the notes that the singer is actually singing
- Scoring system showing different features

The Malata is really good in that it shows the notes of the melody and also shows the notes that you are singing. But it has a pathetic range of English songs and doesn't show the Pinyin for the Chinese songs. The Songen has a good selection of both and shows the Pinyin, but doesn't show the notes and has a simplistic scoring system.

So I want to take the songs off my Sonken DVD and either play them on the Malata or on my PC. Playing them on my PC is preferred because then I am only limited by the programs that I can write and am not so dependent on the vendor's machines. So my immediate goal is to get the songs off the Sonken DVD and start playing them in the ways that I want.

The files on the Sonken DVD are in DKD format. This is an undocumented format probably standing for Digital Karaoke Disk. Many people have worked on this format, and there has been much discussion in forums such as the Karaoke Engineering. These include Understanding the HOTDOG files on DVD of California electronics , Decoding JBK 6628 DVD Karaoke Disc and Karaoke Huyndai 99 .

When I started looking at my disk, I went about it in a different direction to many of the posters in these forums. Also, the results in the forums were presented in an adhoc and often confusing manner - as could be expected. So I ended up re-inventing a lot of what had already been discovered, as well as coming up with some new stuff.

In hindsight, I could have saved myself weeks of work if I had paid proper attention to what was said in the forums. So this chapter is my attempt to lay out the results in a simple and logical

enough way so that people trying to do similar things with their own disks can easily work out what is applicable to their situation and what is different.

What this chapter will cover is

- What files are on my DVD
- What each file contains (overview)
- Matching song titles to song numbers
- Finding the song data on the disk
- Extracting the song data
- Decoding the song data

This section is not complete, as there is still more to be discovered.

Format shifting

Isn't it illegal to copy your DVDs? Not in Australia, under the right conditions (Copyright Amendment Act 2006 - FAQs):

> Will I be able to copy my music collection onto my iPod?

Yes. You can format-shift music that you own to devices such as an MP3 player, X-Box 360 or your computer.

I am just copying the music I legally bought with the Sonken DVD to my computer for personal use. That is within the revised Australian copyright act. You should check if your country allows the same rights.

Don't ask for any copies of the files off my DVD. That would be illegal, and I'm not going to do it.

Files on the DVD

My Sonken DVD disk contains these files:

```
BACK01.MPG
      DTSMUS00.DKD
      DTSMUS01.DKD
      DTSMUS02.DKD
      DTSMUS03.DKD
      DTSMUS04.DKD
      DTSMUS05.DKD
      DTSMUS06.DKD
      DTSMUS07.DKD
      DTSMUS10.DKD
      DTSMUS20.DKD
```

BACK01.MPG

This is the MP3 file that plays in the background

DTSMUS00.DKD - DTSMUS07.DKD

These are the song files. The number of these depends on how many songs are on the DVD.

DTSMUS10.DKD

No-one has worked out what this file is for yet.

DTSMUS20.DKD

This file contains the list of song number/song title/artist as given in the song book. The song number in this file is one less than the song number in the book.

Decoding DTSMUS20.DKD

I'm on a Linux system and I use Linux/Unix utilities and applications. Equivalents exist under other O/S's such as Windows and Apple.

Song information

The Unix command `strings` lists all the ASCII 8-bit encoded strings in a file that are at least 4 characters long. Running this command on all the DVD files shows that DTSMUS20.DKD is the only one with lots of english-language strings, and these strings are the song titles on the DVD.

A brief selection is

```
Come To Me
      Come To Me Boy
      Condition Of My Heart
      Fly To The Sky
      Cool Love
      Count Down
      Cowboy
      Crazy
```

The actual strings that would show on your disk depends of course on the songs on it. You would need some english language titles on it for this to work, of course!

To make further progress you need a binary editor. I use `bvi`. `emacs` has a binary editor mode as well. Search in there for a song title you know is on the disk. For example, searching for the Beatles "Here Comes The Sun" shows the block

```
000AA920  12 D3 88 48 65 72 65 20 43 6F 6D 65
73 20 54 68 ...Here Comes Th
      000AA930  65 20 52 61 69 6E 20 41 67 61
69 6E 00 45 75 72 e Rain Again.Eur
      000AA940  79 74 68 6D 69 63 73 00 1F 12
D3 89 48 65 72 65 ythmics.....Here
      000AA950  20 43 6F 6D 65 73 20 54 68 65
20 53 75 6E 00 42  Comes The Sun.B
      000AA960  65 61 74 6C 65 73 00 1B 12 D3
8A 48 65 72 65 20 eatles.....Here
      000AA970  46 6F 72 20 59 6F 75 00 46 69
72 65 68 6F 75 73 For You.Firehous
```

The string "Here Comes The Sun" starts at 0xAA94C followed by a null byte. This is followed at 0xAA95F by the null-terminated "Beatles". Immediately before this is 4 bytes. The length of these two strings (including the null bytes) and the 4 bytes is 0x1F and this is the first of the 4 preceding bytes. So the block consists of a 4-byte header followed by a null-terminated song title followed by a null-terminated artist. Byte 1 is the length of the song information block including the 4 byte header.

Byte 2 of the header block is 0x12. jim75 at Decoding JBK 6628 DVD Karaoke Disc discovered the document JBK_Manual%5B1%5D.doc . In there is a list of country codes:

```
00 : KOREAN
      01 : CHINESE( reserved )
      02 : CHINESE
      03 : TAIWANESE
      04 : JAPANESE
      05 : RUSSIAN
      06 : THAI
      07 : TAIWANESE( reserved )
      08 : CHINESE( reserved )
      09 : CANTONESE
      12 : ENGLISH
      13 : VIETNAMESE
      14 : PHILIPPINE
      15 : TURKEY
      16 : SPANISH
      17 : INDONESIAN
      18 : MALAYSIAN
      19 : PORTUGUESE
      20 : FRENCH
      21 : INDIAN
      22 : BRASIL
```

The Beatle's song has 0x12 in byte 2 of the header and this matches the country codes in the table. This is confirmed by looking at other language files (later).

I've discovered later that the WMA files have their own codes. So far I have seen

```
83 : CHINESE WMA
      92 : ENGLISH WMA
      94 : PHILIPPINE WMA
```

I guess you can see the pattern with the earlier ones!

Bytes 3 and 4 of the header are 0xD389 which is 54153 in decimal. This is one less than the song number in the book (54154). So bytes 3 and 4 are a 16-bit short integer, one less than the song index in the book.

This pattern is repeated throughout the file, so that each record is of this format.

Beginning/end of data

There is a long sequence of bytes near the beginning of the file "01 01 01 01 01 ...". This finishes on my file at 0x9F23. By comparing the index number with those in my song book, I confirm this is the start of the Korean songs, and probably the start of all songs. I haven't found any table giving me this start value.

Checking a number of songs gives me this table:

- English songs start at 60x9562D, song 24452 type 0x12
- Cantonese at 0x8F5D2, song 13701 type 3
- Korean at 0x9F23, song 37847 type 0
- Indonesian at 0x11F942, song 42002 type 0x17
- Hindi at 0x134227, song 45058 type 0x21
- Phillipine at 0xD5D20, song 62775 type 0x14
- Russian at 0x110428, song 41012 type 5
- Spanish at 0xF5145, song 26487 type 0x16
- Mandarin (1 char) at 0x413BE, song 1388 type 3

I can't find the Vietnamese songs, though. There don't seem to any on my disk. My song book is lying! I guess there is some table somewhere giving these start points, but I haven't found it - these were all found by looking at my song book and then in the file.

The end of the block is signalled by a sequence of "FF FF FF FF ..." at 0x136C92.

But there is lots of stuff both before and after the song information block. I don't know what it means.

Chinese songs

The first English song in my book is "Gump by Al Wierd", song number 24452. In the table of contents file DTSMUS20.DK this is at 0x9562D (611885). The entry before this is "20 03 3A 04 CE D2 B4 F2 C1 CB D2 BB CD A8 B2 BB CB B5 BB B0 B5 C4 B5 E7 BB B0 B8 F8 C4 E3 00 00". The song code is "3A 04" i.e. 14852 which is song number 14853 (one offset, remember!). When I play that song on my karaoke machine I'm in luck: the first character of the song is "我", which I recognise as the word "I" (in Pinyin: wo3). It's encoding in the file is "CE D2". I've got Chinese input installed on my computer so I can search for this Chinese character.

A Google search for "unicode value of 我" shows me

```
[RESOLVED] Converting Unicode Character Literal
to Uint16 variable ...
      www.codeguru.com › ... › C++ (Non Visual
C++ Issues)
      5 posts - 2 authors - 1 Jul 2011

          I've determined that the unicode
character '我' has a hex value of
          0x6211 by looking it up on the "GNOME
Character Map 2.32.1"
          and if I do this....
```

and then looking up 0x6211 on Unicode Search gives gold:

```
Unicode    6211 (25105)
      GB Code    CED2 (4650)
      Big 5 Code    A7DA
      CNS Code    1-4A3C
```

There's the CED2 in the second line as GB Code. So there you go: the character set is GB (probably GB2312 with EUC-CN encoding) with code for 我 as CED2.

Just to make sure: using the table by Mary Ansell at GB Code Table the bytes "CE D2 B4 F2 C1 CB D2 BB CD A8 B2 BB CB B5 BB B0 B5 C4 B5 E7 BB B0 B8 F8 C4 E3" translate into "我 打 了 一 通 ..." which is indeed the song.

Other languages

I'm not familiar with other language encodings so haven't investigated the Thai, Vietnamese, etc. The Korean seems to be EUC-KR.

Programs

The earlier investigations by others have created programs in C or C++. These are generally standalone programs. I would like to build a collection of reusable modules, so I have chosen Java as implementation language.

Java goodies

Java is a good O/O language which supports good design. It includes a Midi player and Midi classes. It supports multiple language encodings so it is easy to switch from, say GB-2312 to Unicode. It has good cross-platform GUI support.

Java baddies

Java doesn't support unsigned integer types. This sucks *really* badly here since so many data types are unsigned for these programs. Even bytes in Java are signed :-(. Here are some of the tricks :-(.

- Make all types the next size up: byte to int, int to long, long to long... Just hope that unsigned longs aren't really needed
- If you need an unsigned byte and you've got an int, and you need it to fit into 8 bits, cast to a byte and hope it's not too big :-(

- Typecast all over the place to keep the compiler happy e.g. when a byte is required from an int, `(byte) n`
- Watch signs all over the place. If you want to right shift a number, the operator >> preserves sign extensions so eg in binary 1XYZ... shifts to 1111XYZ.. You need to use >>> which results in 0001XYZ.
- If you want to assign an unsigned byte to an int, watch signs again. You may need

```
n = b ≥ 0 ? b : 256 - b
```

- To build an unsigned int from 2 unsigned bytes, signs will stuff you again: n = (b1 << 8) + b2 will get it wrong if either b1 or b2 is -ve. Instead use

```
n = ((b1 ≥ 0 ? b1 : 256 - b1) << 8) + (b2 ≥
0 ? b2 : 256 - b2)
```

(no joke!)

Classes

The song class contains information about a single song and is given here: SongInformation.java

```java
public class SongInformation {


    // Public fields of each song record
    /**
     *  Song number in the file, one less than
in songbook
     */
    public long number;

    /**
     * song title in Unicode
     */
    public String title;

    /**
     * artist in Unicode
     */
    public String artist;

    /**
     * integer value of language code
     */
    public int language;

    public static final int  KOREAN = 0;
```

```
    public static final int  CHINESE1 = 1;
    public static final int  CHINESE2 = 2;
    public static final int  TAIWANESE3 = 3 ;
    public static final int  JAPANESE = 4;
    public static final int  RUSSIAN = 5;
    public static final int  THAI = 6;
    public static final int  TAIWANESE7 = 7;
    public static final int  CHINESE8 = 8;
    public static final int  CANTONESE = 9;
    public static final int  ENGLISH = 0x12;
    public static final int  VIETNAMESE = 0x13;
    public static final int  PHILIPPINE = 0x14;
    public static final int  TURKEY = 0x15;
    public static final int  SPANISH = 0x16;
    public static final int  INDONESIAN = 0x17;
    public static final int  MALAYSIAN = 0x18;
    public static final int  PORTUGUESE = 0x19;
    public static final int  FRENCH = 0x20;
    public static final int  INDIAN = 0x21;
    public static final int  BRASIL = 0x22;
    public static final int  CHINESE131 = 131;
    public static final int  ENGLISH146 = 146;
    public static final int  PHILIPPINE148 =
148;

    public SongInformation(long number,
                String title,
```

```
                String artist,
                int language) {
    this.number = number;
    this.title = title;
    this.artist = artist;
    this.language = language;
    }

    public String toString() {
    return "" + (number+1) + " (" + language +
") \"" + title + "\" " + artist;
    }

    public boolean titleMatch(String pattern) {
    // System.out.println("Pattern: " +
pattern);
    return title.matches("(?i).*" + pattern +
".*");
    }

    public boolean artistMatch(String pattern)
{
    return artist.matches("(?i).*" + pattern +
".*");
    }

    public boolean numberMatch(String pattern)
```

```
{
    Long n;
    try {
        n = Long.parseLong(pattern) - 1;
        //System.out.println("Long is " + n);
    } catch(Exception e) {
        //System.out.println(e.toString());
        return false;
    }
    return number == n;
    }


    public boolean languageMatch(int lang) {
    return language == lang;
    }
}
```

The song table class holds a list of song information objects and is given by SongTable.java

```
import java.util.Vector;
import java.io.FileInputStream;
import java.io.*;
import java.nio.charset.Charset;

// public class SongTable implements
java.util.Iterator {
// public class SongTable extends
Vector<SongInformation> {
public class SongTable {

    private static final String SONG_INFO_FILE
=
"/home/newmarch/Music/karaoke/sonken/DTSMUS20.DKD

    private static final long INFO_START =
0x9F23;

    public static final int ENGLISH = 0x12;

    private static Vector<SongInformation>
allSongs;

    private Vector<SongInformation> songs =
    new Vector<SongInformation>  ();
```

```
    public static long[] langCount = new
long[0x23];

    public SongTable(Vector<SongInformation>
songs) {
    this.songs = songs;
    }

    public SongTable() throws
java.io.IOException,
                  java.io.FileNotFoundException
{
    FileInputStream fstream = new
FileInputStream(SONG_INFO_FILE);
    fstream.skip(INFO_START);
    while (true) {
        int len;
        int lang;
        long number;

        len = fstream.read();
        lang = fstream.read();
        number = readShort(fstream);
        if (len == 0xFF && lang == 0xFF &&
number == 0xFFFFL) {
        break;
        }
```

```
        byte[] bytes = new byte[len - 4];
        fstream.read(bytes);
        int endTitle;
        // find null at end of title
        for (endTitle = 0; bytes[endTitle] !=
0; endTitle++)
        ;
        byte[] titleBytes = new byte[endTitle];
        byte[] artistBytes = new byte[len -
endTitle - 6];

        System.arraycopy(bytes, 0, titleBytes,
0, titleBytes.length);
        System.arraycopy(bytes, endTitle + 1,
                 artistBytes, 0,
artistBytes.length);
        String title = toUnicode(lang,
titleBytes);
        String artist = toUnicode(lang,
artistBytes);
        // System.out.printf("artist: %s,
title: %s, lang: %d, number %d\n", artist,
title, lang, number);
        SongInformation info = new
SongInformation(number,
                                title,
                                artist,
```

```
                                lang);
        songs.add(info);

        if (lang > 0x22) {
        //System.out.println("Illegal lang
value " + lang + " at song " + number);
        } else {
        langCount[lang]++;
        }
    }
    allSongs = songs;
    }

    public void dumpTable() {
    for (SongInformation song: songs) {
        System.out.println("" + (song.number+1)
+ " - " +
                    song.artist + " - " +
                    song.title);
    }
    }

    public java.util.Iterator<SongInformation>
iterator() {
    return songs.iterator();
    }
```

```
    private int readShort(FileInputStream f)
throws java.io.IOException {
    int n1 = f.read();
    int n2 = f.read();
    return (n1 << 8) + n2;
    }

    private String toUnicode(int lang, byte[]
bytes) {
    switch (lang) {
    case SongInformation.ENGLISH:
    case SongInformation.ENGLISH146:
    case SongInformation.PHILIPPINE:
    case SongInformation.PHILIPPINE148:
        // case SongInformation.HINDI:
    case SongInformation.INDONESIAN:
    case SongInformation.SPANISH:
        return new String(bytes);

    case SongInformation.CHINESE1:
    case SongInformation.CHINESE2:
    case SongInformation.CHINESE8:
    case SongInformation.CHINESE131:
    case SongInformation.TAIWANESE3:
    case SongInformation.TAIWANESE7:
    case SongInformation.CANTONESE:
            Charset charset =
```

```
Charset.forName("gb2312");
            return new String(bytes, charset);

    case SongInformation.KOREAN:
        charset = Charset.forName("euckr");
            return new String(bytes, charset);

    default:
        return "";
    }
    }

    public SongInformation getNumber(long
number) {
    for (SongInformation info: songs) {
        if (info.number == number) {
        return info;
        }
    }
    return null;
    }

    public SongTable titleMatches( String
pattern) {
    Vector<SongInformation> matchSongs =
        new Vector<SongInformation>  ();
```

```
    for (SongInformation song: songs) {
        if (song.titleMatch(pattern)) {
        matchSongs.add(song);
        }
    }
    return new SongTable(matchSongs);
    }

     public SongTable artistMatches( String
pattern) {
    Vector<SongInformation> matchSongs =
        new Vector<SongInformation>  ();

    for (SongInformation song: songs) {
        if (song.artistMatch(pattern)) {
        matchSongs.add(song);
        }
    }
    return new SongTable(matchSongs);
    }

      public SongTable numberMatches( String
pattern) {
    Vector<SongInformation> matchSongs =
        new Vector<SongInformation>  ();

    for (SongInformation song: songs) {
```

```
        if (song.numberMatch(pattern)) {
        matchSongs.add(song);
        }
    }
    return new SongTable(matchSongs);
    }

    public String toString() {
    StringBuffer buf = new StringBuffer();
    for (SongInformation song: songs) {
        buf.append(song.toString() + "\n");
    }
    return buf.toString();
    }

    public static void main(String[] args) {
    // for testing
    SongTable songs = null;
    try {
        songs = new SongTable();
    } catch(Exception e) {
        System.err.println(e.toString());
        System.exit(1);
    }
    songs.dumpTable();
    System.exit(0);
```

```
    // Should print "54151 Help Yourself Tom
Jones"

System.out.println(songs.getNumber(54150).toStrin



    // Should print "18062 伦巴(恋歌) 伦巴"

System.out.println(songs.getNumber(18061).toStrin



    System.out.println(songs.artistMatches("Tom
Jones").toString());
    /* Prints
54151 Help Yourself Tom Jones
50213 Daughter Of Darkness Tom Jones
23914 DELILAH Tom Jones
52834 Funny Familiar Forgotten Feelings Tom
Jones
54114 Green green grass of home Tom Jones
54151 Help Yourself Tom Jones
55365 I (WHO HAVE NOTHING) TOM JONES
52768 I Believe Tom Jones
55509 I WHO HAVE NOTHING TOM JONES
55594 I'll Never Fall Inlove Again Tom Jones
55609 I'm Coming Home Tom Jones
51435 It's Not Unusual Tom Jones
```

```
55817 KISS Tom Jones
52842 Little Green Apples Tom Jones
51439 Love Me Tonight Tom Jones
56212 My Elusive Dream TOM JONES
56386 ONE DAY SOON Tom Jones
22862 THAT WONDERFUL SOUND Tom Jones
57170 THE GREEN GREEN GRASS OF HOME TOM JONES
57294 The Wonderful Sound Tom Jones
23819 TILL Tom Jones
51759 What's New Pussycat Tom Jones
52862 With These Hands Tom Jones
57715 Without Love Tom Jones
57836 You're My World Tom Jones
    */

    for (int n = 1; n < langCount.length; n++)
{
        if (langCount[n] != 0) {
        System.out.println("Count: " +
langCount[n] + " of lang " + n);
        }
    }

    // Check Russian, etc
    System.out.println("Russian " + '\u0411');
    System.out.println("Korean " + '\u0411');
    System.exit(0);
```

```
    }
}
```

You may need to adjust the constant values in the file-based constructor for this to work properly for you.

A Java program using Swing to allow display and searching of the song titles is SongTableSwing.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
import javax.swing.*;
import javax.swing.SwingUtilities;
import java.util.regex.*;
import java.io.*;

public class SongTableSwing extends JPanel {
   private DefaultListModel model = new
DefaultListModel();
    private JList list;
    private static SongTable allSongs;

    private JTextField numberField;
    private JTextField langField;
    private JTextField titleField;
    private JTextField artistField;

    // This font displays Asian and European
characters.
    // It should be in your distro.
    // Fonts displaying all Unicode are
zysong.ttf and Cyberbit.ttf
    // See
http://unicode.org/resources/fonts.html
```

```
    private Font font = new Font("WenQuanYi Zen
Hei", Font.PLAIN, 16);
    // font = new Font("Bitstream Cyberbit",
Font.PLAIN, 16);

    private int findIndex = -1;

    /**
     * Describe <code>main</code> method here.
     *
     * @param args a <code>String</code> value
     */
    public static final void main(final
String[] args) {
    allSongs = null;
    try {
        allSongs = new SongTable();
    } catch(Exception e) {
        System.err.println(e.toString());
        System.exit(1);
    }

    JFrame frame = new JFrame();
    frame.setTitle("Song Table");
    frame.setSize(1000, 800);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLO
```

```
    SongTableSwing panel = new
SongTableSwing(allSongs);
    frame.getContentPane().add(panel);

    frame.setVisible(true);
    }

    public SongTableSwing(SongTable songs) {

    if (font == null) {
        System.err.println("Can't fnd font");
    }

    int n = 0;
    java.util.Iterator<SongInformation> iter =
songs.iterator();
    while(iter.hasNext()) {
        model.add(n++, iter.next());
        // model.add(n++,
iter.next().toString());
    }

    BorderLayout mgr = new BorderLayout();

    list = new JList(model);
```

```
    // list = new JList(songs);
    list.setFont(font);
    JScrollPane scrollPane = new
JScrollPane(list);

    setLayout(mgr);
    add(scrollPane, BorderLayout.CENTER);

    JPanel bottomPanel = new JPanel();
    bottomPanel.setLayout(new GridLayout(2,
1));
    add(bottomPanel, BorderLayout.SOUTH);

    JPanel searchPanel = new JPanel();
    bottomPanel.add(searchPanel);
    searchPanel.setLayout(new FlowLayout());

    JLabel numberLabel = new JLabel("Number");
    numberField = new JTextField(5);

    JLabel langLabel = new JLabel("Language");
    langField = new JTextField(8);

    JLabel titleLabel = new JLabel("Title");
    titleField = new JTextField(20);
    titleField.setFont(font);
```

```
    JLabel artistLabel = new JLabel("Artist");
    artistField = new JTextField(10);
    artistField.setFont(font);

    searchPanel.add(numberLabel);
    searchPanel.add(numberField);
    // searchPanel.add(langLabel);
    // searchPanel.add(langField);
    searchPanel.add(titleLabel);
    searchPanel.add(titleField);
    searchPanel.add(artistLabel);
    searchPanel.add(artistField);


titleField.getDocument().addDocumentListener(new
 DocumentListener() {
        public void changedUpdate(DocumentEvent
e) {
            // rest find to -1 to restart any
find searches
            findIndex = -1;
            // System.out.println("reset find
index");
        }
        public void insertUpdate(DocumentEvent
e) {
            findIndex = -1;
```

```
            // System.out.println("reset insert
find index");
        }
        public void removeUpdate(DocumentEvent
e) {
            findIndex = -1;
            // System.out.println("reset remove
find index");
        }
        }
        );

artistField.getDocument().addDocumentListener(new
 DocumentListener() {
        public void changedUpdate(DocumentEvent
e) {
            // rest find to -1 to restart any
find searches
            findIndex = -1;
            // System.out.println("reset insert
find index");
        }
        public void insertUpdate(DocumentEvent
e) {
            findIndex = -1;
            // System.out.println("reset insert
find index");
```

```
        }
        public void removeUpdate(DocumentEvent
e) {
            findIndex = -1;
            // System.out.println("reset insert
find index");
        }
        }
        );

    titleField.addActionListener(new
ActionListener(){
                public void
actionPerformed(ActionEvent e){
            filterSongs();
                }});
    artistField.addActionListener(new
ActionListener(){
                public void
actionPerformed(ActionEvent e){
            filterSongs();
                }});
    numberField.addActionListener(new
ActionListener(){
                public void
actionPerformed(ActionEvent e){
            filterSongs();
```

```
                }});

    JPanel buttonPanel = new JPanel();
    bottomPanel.add(buttonPanel);
    buttonPanel.setLayout(new FlowLayout());

    JButton find = new JButton("Find");
    JButton filter = new JButton("Filter");
    JButton reset = new JButton("Reset");
    JButton play = new JButton("Play");
    buttonPanel.add(find);
    buttonPanel.add(filter);
    buttonPanel.add(reset);
    buttonPanel.add(play);

    find.addActionListener(new ActionListener()
{
        public void actionPerformed(ActionEvent
e) {
            findSong();
        }
        });

    filter.addActionListener(new
ActionListener() {
        public void actionPerformed(ActionEvent
e) {
```

```
            filterSongs();
        }
        });

    reset.addActionListener(new
ActionListener() {
        public void actionPerformed(ActionEvent
e) {
            resetSongs();
        }
        });

    play.addActionListener(new ActionListener()
{
        public void actionPerformed(ActionEvent
e) {
            playSong();
        }
        });

     }

    public void findSong() {
    String number = numberField.getText();
    String language = langField.getText();
    String title = titleField.getText();
    String artist = artistField.getText();
```

```
    if (number.length() != 0) {
        try {

        long num = Integer.parseInt(number) -
1;
        for (int n = 0; n < model.getSize();
n++) {
            SongInformation info =
(SongInformation) model.getElementAt(n);
            if (info.number == num) {
            list.setSelectedIndex(n);
            list.ensureIndexIsVisible(n);
            return;
            }
        }
        } catch(Exception e) {
        System.err.println("Not a number");
        numberField.setText("");
        }

        return;
    }

    /*
    System.out.println("Title " + title +
title.length() +
```

```
                "artist " + artist +
artist.length() +
                " find start " + findIndex +
                " model size " +
model.getSize());
    if (title.length() == 0 && artist.length()
== 0) {
        System.err.println("no search terms");
        return;
    }
    */

    //System.out.println("Search " + searchStr
+ " from index " + findIndex);
    for (int n = findIndex + 1; n <
model.getSize(); n++) {
        SongInformation info =
(SongInformation) model.getElementAt(n);
        //System.out.println(info.toString());

        if ((title.length() != 0) &&
(artist.length() != 0)) {
        if (info.titleMatch(title) &&
info.artistMatch(artist)) {
            // System.out.println("Found " +
info.toString());
            findIndex = n;
```

```
            list.setSelectedIndex(n);
            list.ensureIndexIsVisible(n);
            break;
        }
        } else {
        if ((title.length() != 0) &&
info.titleMatch(title)) {
            // System.out.println("Found " +
info.toString());
            findIndex = n;
            list.setSelectedIndex(n);
            list.ensureIndexIsVisible(n);
            break;
        } else if ((artist.length() != 0) &&
info.artistMatch(artist)) {
            // System.out.println("Found " +
info.toString());
            findIndex = n;
            list.setSelectedIndex(n);
            list.ensureIndexIsVisible(n);
            break;

        }
        }

    }
    }
```

```
    public void filterSongs() {
    String title = titleField.getText();
    String artist = artistField.getText();
    String number = numberField.getText();
    SongTable filteredSongs = allSongs;

    if (allSongs == null) {
        // System.err.println("Songs is null");
        return;
    }

    if (title.length() != 0) {
        filteredSongs =
filteredSongs.titleMatches(title);
    }
    if (artist.length() != 0) {
        filteredSongs =
filteredSongs.artistMatches(artist);
    }
    if (number.length() != 0) {
        filteredSongs =
filteredSongs.numberMatches(number);
    }

    model.clear();
    int n = 0;
```

```
    java.util.Iterator<SongInformation> iter =
filteredSongs.iterator();
    while(iter.hasNext()) {
        model.add(n++, iter.next());
    }
    }

    public void resetSongs() {
    artistField.setText("");
    titleField.setText("");
    numberField.setText("");
    model.clear();
    int n = 0;
    java.util.Iterator<SongInformation> iter =
allSongs.iterator();
    while(iter.hasNext()) {
        model.add(n++, iter.next());
    }
    }
    /**
     * "play" a song by printing its id to
standard out.
     * Can be used in a pipeline this way
     */
    public void playSong() {
    SongInformation song = (SongInformation)
list.getSelectedValue();
```

```
    if (song == null) {
        return;
    }
    long number = song.number + 1;
    System.out.println("" + number);
    }


    class SongInformationRenderer extends
JLabel implements ListCellRenderer {

    public Component
getListCellRendererComponent(
                                JList list,
                                Object value,
                                int index,
                                boolean
isSelected,
                                boolean
cellHasFocus) {
        setText(value.toString());
        return this;
    }
    }
}
```

When "play" is selected it will print the song id to standard output for use in a pipeline.

The data files

General

The files DTSMUS00.DKD - DTSMUS07.DKD contain the music files. There are two formats for the music: Microsoft WMA files and MIDI files. In my song books some songs are marked as having a singer. These turn out to be the WMA files. Those without a singer are MIDI files.

The WMA files are just that. The MIDI files are slightly compressed and have to be decoded before they can be played.

Each song block has at the beginning a section containing the lyrics. These are compressed and have to be decoded.

The data for one song forms a record of contiguous bytes. These records are collected into blocks, also contiguous. The blocks are separate. There is a "super block" of pointers to these blocks. Part of the song number is an index into the super block, selecting the block. The rest of the song number is an index of the record in the block.

My route into this

I came backwards into this and only arrived at understanding what others had accomplished after some time. So in case it helps any others, here is my route.

I used the Unix command `strings` to discover the songs information in DTSMUS10.DKD. On the other files it didn't seem to produce much. But there were ASCII strings in these files and some were repeated. So I wrote a shell pipeline to sort these strings and count them. The pipeline for one file was

```
strings DTSMUS05.DKD | sort |uniq -c | sort -n
-r |less
```

This produced results

```
1229 :^y|
      1018 j?wK
      843 ]/<
      756  Seh
      747  Ser
      747 _\D+P
      674 :^yt
      234 IRI$
```

The results weren't inspiring. But when I looked inside the files to see where "Ser" was occurring, I also saw:

```
q03C3E230  F6 01 00 00 00 02 00 16 00 57 00 69
00 6E 00 64 .........W.i.n.d
      03C3E240  00 6F 00 77 00 73 00 20 00 4D
00 65 00 64 00 69 .o.w.s. .M.e.d.i
      03C3E250  00 61 00 20 00 41 00 75 00 64
00 69 00 6F 00 20 .a. .A.u.d.i.o.
      03C3E260  00 39 00 00 00 24 00 20 00 34
00 38 00 20 00 6B .9...$. .4.8. .k
      03C3E270  00 62 00 70 00 73 00 2C 00 20
00 34 00 34 00 20 .b.p.s.,. .4.4.
      03C3E280  00 6B 00 48 00 7A 00 2C 00 20
00 73 00 74 00 65 .k.H.z.,. .s.t.e
      03C3E290  00 72 00 65 00 6F 00 20 00 31
00 2D 00 70 00 61 .r.e.o. .1.-.p.a
      03C3E2A0  00 73 00 73 00 20 00 43 00 42
00 52 00 00 00 02 .s.s. .C.B.R....
      03C3E2B0  00 61 01 91 07 DC B7 B7 A9 CF
11 8E E6 00 C0 0C .a..............
      03C3E2C0  20 53 65 72 00 00 00 00 00 00
00 40 9E 69 F8 4D  Ser.......@.i.M
```

Wow! *two byte* characters!

The `strings` command has options to look at e.g. 2-byte big-endian character strings. The command

```
strings -e b DTSMUS05.DKD
```

turned up

```
IsVBR
      DeviceConformanceTemplate
      WM/WMADRCPeakReference
      WM/WMADRCAverageReference
      WMFSDKVersion
      9.00.00.2980
      WMFSDKNeeded
      0.0.0.0000
```

These are all part of the WMA format.

According to Gary Kessler's FILE SIGNATURES TABLE , the signature of a WMA file is given by the header

```
30 26 B2 75 8E 66 CF 11
      A6 D9 00 AA 00 62 CE 6C
```

and that pattern does occur, with the above strings appearing some time later.

The spec for the ASF/WMA file format is at Advanced Systems Format (ASF) Specification although you are advised not to read it in case you want to do anything open source with such files.

So on that basis I could indentify the start of WMA files. The 4 bytes preceding each WMA file are the length of the file. From that I could find the *end* of the file, which turned out to be the start of a record for the *next* record containing some stuff and then the next WMA file.

In these records I could see patterns I couldn't understand, but also from byte 36 on I could see strings like

```
AIN'T IT FUNNY HOW TIME SLIPS AWAY, Str length:
34



      00000000  10 50 41 10 50 49 10 50 4E 10
50 27 10 50 54 10 .PA.PI.PN.P'.PT.
      00000010  50 20 11 F1 25 12 71 05 04 61
05 05 51 21 13 01 P ..%.q..a..Q!..
      00000020  02 05 91 2B 10 20 48 10 50 4F
10 50 57 13 40 00 ...+. H.PO.PW.@.
      00000030  12 61 02 12 01 02 04 D1 05 04
51 3B 05 31 05 04 .a........Q;.1..
      00000040  C1 29 10 20 50 10 51 45 10 21
28 10 21 1E 10 21 .). P.QE.!(.!..!
      00000050  3A 14 F1 05 13 31 02 10 C1 0E
11 A1 58 15 A0 00 :....1......X...
      00000060  15 70 00 13 A0 A9
.p....
```

Can you see "A.I.N.'.T" (as ".PA.PI.PN.P'.PT")?

But I couldn't figure out what the encoding was or how to find the table of song starts. That's when I was ready to look at the earlier stuff and understand how it applied to me. (

Understanding the HOTDOG files on DVD of California electronics , Decoding JBK 6628 DVD Karaoke Disc and Karaoke Huyndai 99).

The super block

The file DTSMUS00.DKD starts with a bunch of nulls. At 0x200 it starts to kick in with data. This was identified as the start of a "table of tables" i.e. a superblock. Each entry in this superblock is a 4-byte integer, which turns out to be an index to tables in the data files. The superblock is terminated by a sequence of nulls (for me at 0x5F4) and there are less than 256 indexes in the table.

The value of these superblock entries seems to have changed in different versions. In the JBK disk and also on mine, the values have to be multiplied by 0x800 to give a "virtual offset" in the data files.

To give meaning to this: on my disk at 0x200 is

```
00000200  00 00 00 01 00 00 08 6C 00 00 0F C1
00 00 17 7A
      00000210  00 00 1E 81 00 00 25 21 00 00
2B 8D 00 00 32 B7
```

So the table values are 0x1, 0x86C, 0xFC1, 0x177A, ... The "virtual addresses" are 0x800, 0x436000 (0x86C * 0x800) and so on. If you go to these addresses, then before the address is a bunch of nulls, and at that address is data.

Why I call them virtual addresses is because there are 8 data files on my DVD and most addresses are larger than any of the files. The files in my case are all 1065353216L (except the last) bytes. The "obvious" solution works: the file number is address / file size, and the offset into the file is address % file size. You can check this by looking for the nulls before the address of each block.

Song start tables

Each of the tables indexed from the super block is a table of song indexes. Each table contains 4-byte indexes. Each table has at most 0x100 entries, or is terminated by a zero index. Each index is the offset from the table start of the beginning of a song entry.

Locating song entry from song number

Given a song number such as 54154 "Here Comes The Sun" we can now find the song entry. Reduce the song number by one to 54153. It is a 16-bit number. The top 8 bits are the index of the song index table in the superblock. The bottom 8 bits are the index of the song entry in the song index table.

Pseudocode:

```
songNumber = get number for song from
DTSMUS20.DKD
      superBlockIdx = songNumber >>
      indexTableIdx = songNumber & 0xFF

      seek(DTSMUS00.DKD, superBlockIdx)
      superBlockValue = read 4-byte int from
DTSMUS00.DKD

      locationIndexTable = superBlockValue *
0x800
      fileNumber = locationIndexTable /
fileSize
      indexTableStart = locationIndexTable %
fileSize
      entryLocation = indexTableStart +
indexTableIdx

      seek(fileNumber, entryLocation)
      read song entry
```

Song entries

Each song entry has a header and is followed by two blocks that I call the information block and the song data block. Each header block has a 2-byte type code and a 2-byte integer length. The type code is either 0x0800 or 0x0000. The code signals the encoding of the song data: 0x0800 is a WMA file while 0x0000 is a MIDI file.

If the type code is 0x0 such as the Beatles "Help!" (song number 51765) then the information block has the length in the header block and starts 12 bytes further in. The song data block immediately follows this.

If the type code is 0x8000 then the information block starts 4 bytes in for the length given in the header. The song block starts on the next 16-byte boundary from the end of the information block.

The song block starts with a 4-byte header which is the length of the song data for all types.

Song data

If the song type is 0x8000 then the song data is a WMA file. All songs looked at have a singer included in this file.

If the song type is 0x0 then (from the book) there is no singer in the songs looked at. The file is encoded, and decodes to a MIDI file.

Decoding MIDI files

All files have a lyric block followed by a music block. The lyric block is compressed and it has been discovered that this is LZW compression. This decompresses to a set of 4-byte chunks. The first two bytes are characters of the lyric. For 1-byte encodings such as English or Vietnamese, the first byte is one character and the second is either zero or another character (two byts such as "\r\n"). For two byte encodings such as GB-2312, the two bytes form one character.

The next two bytes are the length of time the character string plays for.

Lyric block

Each lyric block starts with strings such as "#0001 @@00@12 @Help Yourself @ @@Tom Jones " The language code is in there as NN in "@00@NN". The song title, writer, singer are clear. (Note: these characters are all 4 bytes apart!). For English it is "12" and so on.

Bytes 0 and 1 of each block are a character in the lyric. Bytes 2 and 3 are the duration of each character. To turn them into MIDI data, the durations have to be turned into start/stop of each character.

My Java program to do this is SongExtracter.java

```
import java.io.*;
import javax.sound.midi.*;
import java.nio.charset.Charset;

public class SongExtracter {
    private static final boolean DEBUG = false;

    private String[] dataFiles = new String[] {
    "DTSMUS00.DKD", "DTSMUS01.DKD",
"DTSMUS02.DKD",
    "DTSMUS03.DKD", "DTSMUS04.DKD",
"DTSMUS05.DKD",
    "DTSMUS06.DKD", "DTSMUS07.DKD"};
    private String superBlockFileName =
dataFiles[0];
    private static final String DATADIR =
"/home/newmarch/Music/karaoke/sonken/";
    private static final String SONGDIR
="/home/newmarch/Music/karaoke/sonken/songs/";
    //private static final String SONGDIR
="/server/KARAOKE/KARAOKE/Sonken/";
    private static final long SUPERBLOCK_OFFSET
= 0x200;
    private static final long BLOCK_MULTIPLIER
= 0x800;
    private static final long FILE_SIZE =
```

```
0x3F800000L;

    private static final int SIZE_UINT = 4;
    private static final int SIZE_USHORT = 2;

    private static final int ENGLISH = 12;

    public RawSong getRawSong(int songNumber)
    throws java.io.IOException,
           java.io.FileNotFoundException {
    if (songNumber < 1) {
        throw new FileNotFoundException();
    }

    // song number in files is one less than
song number in books, so
    songNumber--;

    long locationIndexTable =
getTableIndexFromSuperblock(songNumber);
    debug("Index table at %X\n",
locationIndexTable);

    long locationSongDataBlock =
getSongIndex(songNumber, locationIndexTable);

    // Now we are at the start of the data
```

```
block
    return
readRawSongData(locationSongDataBlock);

    //debug("Data block at %X\n", songStart);
    }

    private long
getTableIndexFromSuperblock(int songNumber)
    throws java.io.IOException,
           java.io.FileNotFoundException {
    // index into superblock of table of song
offsets
    int superBlockIdx = songNumber >> 8;

    debug("Superblock index %X\n",
superBlockIdx);

    File superBlockFile = new File(DATADIR +
superBlockFileName);

        FileInputStream fstream = new
FileInputStream(superBlockFile);

    fstream.skip(SUPERBLOCK_OFFSET +
superBlockIdx * SIZE_UINT);
    debug("Skipping to %X\n", SUPERBLOCK_OFFSET
```

```
+ superBlockIdx*4);
    long superBlockValue = readUInt(fstream);

    // virtual address of the index table for
this song
    long locationIndexTable = superBlockValue *
BLOCK_MULTIPLIER;

    return locationIndexTable;
    }

    /*
     * Virtual address of song data block
     */
    private long getSongIndex(int songNumber,
long locationIndexTable)
    throws java.io.IOException,
           java.io.FileNotFoundException {
    // index of song into table of song ofsets
    int indexTableIdx = songNumber & 0xFF;
    debug("Index into index table %X\n",
indexTableIdx);

    // translate virtual address to physical
address
    int whichFile = (int) (locationIndexTable /
FILE_SIZE);
```

```
    long indexTableStart =  locationIndexTable
% FILE_SIZE;
    debug("Which file %d index into file %X\n",
whichFile, indexTableStart);

    File songDataFile = new File(DATADIR +
dataFiles[whichFile]);
        FileInputStream dataStream = new
FileInputStream(songDataFile);
    dataStream.skip(indexTableStart +
indexTableIdx * SIZE_UINT);
    debug("Song data index is at %X\n",
indexTableStart + indexTableIdx*SIZE_UINT);

    long songStart = readUInt(dataStream) +
indexTableStart;

    return songStart + whichFile * FILE_SIZE;
    }

    private RawSong readRawSongData(long
locationSongDataBlock)
    throws java.io.IOException {
    int whichFile = (int)
(locationSongDataBlock / FILE_SIZE);
    long dataStart =  locationSongDataBlock %
FILE_SIZE;
```

```
    debug("Which song file %d  into file %X\n",
whichFile, dataStart);

    File songDataFile = new File(DATADIR +
dataFiles[whichFile]);
        FileInputStream dataStream = new
FileInputStream(songDataFile);
    dataStream.skip(dataStart);

    RawSong rs = new RawSong();
    rs.type = readUShort(dataStream);
    rs.compressedLyricLength =
readUShort(dataStream);
    // discard next short
    readUShort(dataStream);
    rs.uncompressedLyricLength =
readUShort(dataStream);
    debug("Type %X, cLength %X uLength %X\n",
rs.type, rs.compressedLyricLength,
rs.uncompressedLyricLength);

    // don't know what the next word is for,
skip it
    //dataStream.skip(4);
    readUInt(dataStream);

    // get the compressed lyric
```

```
    rs.lyric = new
byte[rs.compressedLyricLength];
    dataStream.read(rs.lyric);

    long toBoundary = 0;
    long songLength = 0;
    long uncompressedSongLength = 0;

    // get the song data
    if (rs.type == 0) {
        // Midi file starts in 4 bytes time
        songLength = readUInt(dataStream);
        uncompressedSongLength =
readUInt(dataStream);
        System.out.printf("Song data length %d,
uncompressed %d\n",
                  songLength,
uncompressedSongLength);
        rs.uncompressedSongLength =
uncompressedSongLength;

        // next word is language again?
        //toBoundary = 4;
        //dataStream.skip(toBoundary);
        readUInt(dataStream);
    } else {
        // WMA starts on next 16-byte boundary
```

```
        if( (dataStart +
rs.compressedLyricLength + 12) % 16 != 0) {
        // dataStart already on 16-byte
boundary, so just need extra since then
        toBoundary = 16 -
((rs.compressedLyricLength + 12) % 16);
        debug("Read lyric data to %X\n",
dataStart + rs.compressedLyricLength + 12);
        debug("Length %X to boundary %X\n",
rs.compressedLyricLength, toBoundary);
        dataStream.skip(toBoundary);
        }
        songLength = readUInt(dataStream);
    }

    rs.music = new byte[(int) songLength];
    dataStream.read(rs.music);

    return rs;
    }

    private long readUInt(InputStream is)
throws IOException {
    long val = 0;
    for (int n = 0; n < SIZE_UINT; n++) {
        int c = is.read();
        val = (val << 8) + c;
```

```
    }
    debug("ReadUInt %X\n", val);
    return val;
    }

    private int readUShort(InputStream is)
throws IOException {
    int val = 0;
    for (int n = 0; n < SIZE_USHORT; n++) {
        int c = is.read();
        val = (val << 8) + c;
    }
    debug("ReadUShort %X\n", val);
    return val;
    }

    void debug(String f, Object ...args) {
    if (DEBUG) {
        System.out.printf("Debug: " + f, args);
    }
    }

    public Song getSong(RawSong rs) {
    Song song;
    if (rs.type == 0x8000) {
        song = new WMASong(rs);
    } else {
```

```
        song = new MidiSong(rs);
    }
    return song;
    }

    public static void main(String[] args) {
    if (args.length != 1) {
        System.err.println("Usage: java
SongExtractor <song numnber>");
        System.exit(1);
    }

    SongExtracter se = new SongExtracter();
    try {
        RawSong rs =
se.getRawSong(Integer.parseInt(args[0]));
        rs.dumpToFile(args[0]);

        Song song = se.getSong(rs);
        song.dumpToFile(args[0]);
        song.dumpLyric();
    } catch(Exception e) {
        e.printStackTrace();
    }
    }

    private class RawSong {
```

```
    /**
     * type == 0x0 is Midi
     * type == 0x8000 is WMA
     */
    public int type;
    public int compressedLyricLength;
    public int uncompressedLyricLength;
    public long uncompressedSongLength; // only
needed for compressed Midi
    public byte[] lyric;
    public byte[] music;

    public void dumpToFile(String fileName)
throws IOException {
        FileOutputStream fout = new
FileOutputStream(SONGDIR + fileName +
".lyric");
        fout.write(lyric);
        fout.close();

        fout = new FileOutputStream(SONGDIR +
fileName + ".music");
        fout.write(music);
        fout.close();
    }
    }
```

```
    private class Song {
    public int type;
    public byte[] lyric;
    public byte[] music;
    protected Sequence sequence;
    protected int language = -1;

    public Song(RawSong rs) {
        type = rs.type;
        lyric = decodeLyric(rs.lyric,
                rs.uncompressedLyricLength);
    }

    /**
     * Raw lyric is LZW compressed. Decompress
it
     */
    public byte[] decodeLyric(byte[]
compressedLyric, long uncompressedLength) {
        // uclen is short by at least 2 - other
code adds 10 so we do too
        // TODO: change LZW to use a Vector to
build result so we don't have to guess at
length
        byte[] result = new byte[(int)
uncompressedLength + 10];
        LZW lzw = new LZW();
```

```
        int len = lzw.expand(compressedLyric,
compressedLyric.length, result);
        System.out.printf("uncompressedLength
%d, actual %d\n", uncompressedLength, len);
        lyric = new byte[len];
        System.arraycopy(result, 0, lyric, 0,
(int) uncompressedLength);
        return lyric;
    }

    public void dumpToFile(String fileName)
throws IOException {
        FileOutputStream fout = new
FileOutputStream(SONGDIR + fileName +
".decodedlyric");
        fout.write(lyric);
        fout.close();

        fout = new FileOutputStream(SONGDIR +
fileName + ".decodedmusic");
        fout.write(music);
        fout.close();

        fout = new FileOutputStream(SONGDIR +
fileName + ".mid");
        if (sequence == null)  {
        System.out.println("Seq is null");
```

```
        } else {
        // type is MIDI type 0
        MidiSystem.write(sequence, 0, fout);
        }
    }

    public void dumpLyric() {
        for (int n = 0; n < lyric.length; n +=
4) {
        if (lyric[n] == '\r') {
            System.out.println();
        } else {
            System.out.printf("%c", lyric[n] &
0xFF);
        }
        }
        System.out.println();
        System.out.printf("Language is %X\n",
getLanguageCode());
    }

    /**
     * Lyric contains the language code as
string @00@NN in header section
     */
    public int getLanguageCode() {
        int lang = 0;
```

```
        // Look for @00@NN and return NN
        for (int n = 0; n < lyric.length-20; n
+= 4) {
        if (lyric[n] == (byte) '@' &&
            lyric[n+4] == (byte) '0' &&
            lyric[n+8] == (byte) '0' &&
            lyric[n+12] == (byte) '@') {
            lang = ((lyric[n+16]-'0') << 4) +
lyric[n+20]-'0';
            break;
        }
        }
        return lang;
    }

    /**
     * Lyric is in a language specific
encoding. Translate to Unicode UTF-8.
     * Not all languages are handled because I
don't have a full set of examples
     */
    public byte[] lyricToUnicode(byte[] bytes)
{
        if (language == -1) {
        language = getLanguageCode();
        }
```

```
        switch (language) {
        case SongInformation.ENGLISH:
        return bytes;

        case SongInformation.KOREAN: {
         Charset charset =
Charset.forName("gb2312");
        String str = new String(bytes,
charset);
        bytes = str.getBytes();
        System.out.println(str);
        return bytes;
        }

        case SongInformation.CHINESE1:
        case SongInformation.CHINESE2:
        case SongInformation.CHINESE8:
        case SongInformation.CHINESE131:
        case SongInformation.TAIWANESE3:
        case SongInformation.TAIWANESE7:
        case SongInformation.CANTONESE:
        Charset charset =
Charset.forName("gb2312");
        String str = new String(bytes,
charset);
        bytes = str.getBytes();
        System.out.println(str);
```

```
        return bytes;
        }
        // language not handled
        return bytes;
    }

    public void durationToOnOff() {

    }

    public Track createSequence() {
        Track track;

        try {
        sequence = new Sequence(Sequence.PPQ,
30);
        } catch(InvalidMidiDataException e) {
        // help!!!
        }
        track = sequence.createTrack();
        addLyricToTrack(track);
        return track;
    }

    public void addMsgToTrack(MidiMessage msg,
Track track, long tick) {
        MidiEvent midiEvent = new
```

```
MidiEvent(msg, tick);



        // No need to sort or delay insertion.
From the Java API
        // "The list of events is kept in time
order, meaning that this
        // event inserted at the appropriate
place in the list"
        track.add(midiEvent);
    }

    /**
     * return byte as int, converting to
unsigned if needed
     */
    protected int ub2i(byte b) {
        return  b >= 0 ? b : 256 + b;
    }

    public void addLyricToTrack(Track track) {
        long lastDelay = 0;
        int offset = 0;
        int data0;
        int data1;
        final int LYRIC = 0x05;
        MetaMessage msg;
```

```
        while (offset < lyric.length-4) {
        int data3 = ub2i(lyric[offset+3]);
        int data2 = ub2i(lyric[offset+2]);
        data0 = ub2i(lyric[offset]);
        data1 = ub2i(lyric[offset+1]);

        long delay = (data3 << 8) + data2;

        offset += 4;
        byte[] data;
        int len;
        long tick;

        //     System.out.printf("Lyric offset
%X char %X after %d with delay %d made of %d
%d\n", offset, data0, lastDelay, delay,
lyric[offset-1], lyric[offset-2]);

        if (data1 == 0) {
            data = new byte[] {(byte) data0};
//, (byte) MetaMessage.META};
        } else {
            data = new byte[] {(byte) data0,
(byte) data1}; // , (byte) MetaMessage.META};
        }
        data = lyricToUnicode(data);
```

```
        msg = new MetaMessage();

        if (delay > 0) {
            tick = delay;
            lastDelay = delay;
        } else {
            tick = lastDelay;
        }

        try {
            msg.setMessage(LYRIC, data,
data.length);
        } catch(InvalidMidiDataException e) {
            e.printStackTrace();
            continue;
        }
        addMsgToTrack(msg, track, tick);
        }
    }

    }

    private class WMASong extends Song {

    public WMASong(RawSong rs) {
        // We want to decode the lyric, but
```

```
just copy the music data
        super(rs);
        music = rs.music;
        createSequence();
    }

    public void dumpToFile(String fileName)
throws IOException {
        System.out.println("Dumping WMA to " +
fileName + ".wma");
        super.dumpToFile(fileName);
        FileOutputStream fout = new
FileOutputStream(fileName + ".wma");
        fout.write(music);
        fout.close();
    }

    }

    private class MidiSong extends Song {

        private String[] keyNames = {"C", "C#",
"D", "D#", "E", "F", "F#", "G", "G#", "A",
"A#", "B"};

    public MidiSong(RawSong rs) {
        // We want the decoded lyric plus also
```

```
need to decode the music
        // and then turn it into a Midi
sequence
        super(rs);
        decodeMusic(rs);
        createSequence();
    }

    public void dumpToFile(String fileName)
throws IOException {
        System.out.println("Dumping Midi to " +
fileName);
        super.dumpToFile(fileName);
    }

        public String getKeyName(int
nKeyNumber)
        {
        if (nKeyNumber > 127)
                {
            return "illegal value";
                }
        else
                {
            int     nNote = nKeyNumber % 12;
            int     nOctave = nKeyNumber / 12;
            return keyNames[nNote] + (nOctave -
```

```
1);
                }
        }

    public byte[] decodeMusic(RawSong rs) {
        byte[]  compressedMusic = rs.music;
        long uncompressedSongLength =
rs.uncompressedSongLength;

        // TODO: change LZW to use a Vector to
build result so we don't have to guess at
length
        byte[] expanded = new byte[(int)
uncompressedSongLength + 20];
        LZW lzw = new LZW();
        int len = lzw.expand(compressedMusic,
compressedMusic.length, expanded);
        System.out.printf("Uncompressed %d,
Actual %d\n", compressedMusic.length, len);
        music = new byte[len];
        System.arraycopy(expanded, 0, music, 0,
(int) len);



        return music;
    }
```

```
    public Track createSequence() {
        Track track = super.createSequence();
        addMusicToTrack(track);
        return track;
    }



    public void addMusicToTrack(Track track) {
        int timeLine = 0;
        int offset = 0;
        int midiChannelNumber = 1;

        /* From
http://board.midibuddy.net/showpost.php?
p=533722&postcount=31
           Block of 5 bytes :
           xx xx xx xx xx
           1st byte = Delay Time
           2nd byte = Delay Time when the
velocity will be 0,
           this one will generate another midi
event
           with velocity 0 (see above).
           3nd byte = Event, for example 9x :
Note On for channel x+1,
           cx for PrCh, bx for Par, ex for
```

```
Pitch Bend....
           4th byte = Note
           5th byte = Velocity
        */
        System.out.println("Adding music to
track");
        while (offset < music.length - 5) {

        int startDelayTime =
ub2i(music[offset++]);
        int endDelayTime =
ub2i(music[offset++]);
        int event = ub2i(music[offset++]);
        int data1 = ub2i(music[offset++]);
        int data2 = ub2i(music[offset++]);



        int tick = timeLine + startDelayTime;
        System.out.printf("Offset %X event %X
timeline %d\n", offset, event & 0xFF, tick);

        ShortMessage msg = new ShortMessage();
        ShortMessage msg2 = null;

        try {
            // For Midi event types see
http://www.midi.org/techspecs/midimessages.php
```

```
            switch (event & 0xF0) {
            case ShortMessage.CONTROL_CHANGE:
// Control Change 0xB0
            case ShortMessage.PITCH_BEND:  //
Pitch Wheel Change 0xE0
            msg.setMessage(event, data1,
data2);
            /*
              writeChannel(midiChannelNumber,
chunk[2], false);
              writeChannel(midiChannelNumber,
chunk[3], false);
              writeChannel(midiChannelNumber,
chunk[4], false);
            */
            break;

            case ShortMessage.PROGRAM_CHANGE:
// Program Change 0xC0
            case ShortMessage.CHANNEL_PRESSURE:
// Channel Pressure (After-touch) 0xD0
            msg.setMessage(event, data1, 0);
            break;

            case 0x00:
            // case 0x90:
            // Note on
```

```
            int note = data1;
            int velocity = data2;

            /* We have to generate a pair of
note on/note off.
               The C code manages getting the
order of events
               done correctly by keeping a list
of note off events
               and sticking them into the Midi
sequence when appropriate.
               The Java add() looks after
timing for us, so we'll
               generate a note off first and
add it, and then do the note on
            */
            System.out.printf("Note on %s at
%d, off at %d at offset %X channel %d\n",
                      getKeyName(note),
                      tick, tick +
endDelayTime, offset, (event &0xF)+1);
            // ON
            msg.setMessage(ShortMessage.NOTE_ON
| (event & 0xF),
                       note, velocity);

            // OFF
```

```
            msg2 = new ShortMessage();

msg2.setMessage(ShortMessage.NOTE_OFF  | (event
& 0xF),
                    note, velocity);

            break;

            case 0xF0: // System Exclusive
            // We'll write the data as is to
the buffer
            offset -= 3;
            // msg = SysexMessage();
            while (music[offset] != (byte)
0xF7) // bytes only go upto 127 GRRRR!!!
                {
                //
writeChannel(midiChannelNumber,
midiData[midiOffset], false);
                System.out.printf("sysex:
%x\n", music[offset]);
                offset++;
                if (offset >= music.length) {
                    System.err.println("Run off
end of array while processing Sysex");
                    break;
                }
```

```
                }
            //
writeChannel(midiChannelNumber,
midiData[midiOffset], false);
            offset++;
            System.out.printf("Ignoring sysex
%02X\n", event);

            // ignore the message for now
            continue;
            // break;

            default:
            System.out.printf("Unrecognized
code %02X\n", event);
            continue;
            }
        } catch(InvalidMidiDataException e) {
            e.printStackTrace();
        }

        addMsgToTrack(msg, track, tick);
        if (msg2 != null ) {
            if (endDelayTime <= 0)
System.out.println("Start and end at same
time");
            addMsgToTrack(msg2, track, tick +
```

```
endDelayTime);
            msg2 = null;
        }

        timeLine = tick;
        }
    }
    }
}
```

with support classes LZW.java

```
/**
 * Based on code by Mark Nelson
 * http://marknelson.us/1989/10/01/lzw-data-
compression/
 */

public class LZW {


    private final int BITS = 12;
/* Setting the number of bits to 12, 13*/
    private final int HASHING_SHIFT = (BITS-8);
/* or 14 affects several constants.    */
    private final int MAX_VALUE = (1 << BITS) -
1; /* Note that MS-DOS machines need to   */
    private final int MAX_CODE = MAX_VALUE - 1;
/* compile their code in large model if*/
    /* 14 bits are selected.               */

    private final int TABLE_SIZE = 5021;
/* The string table size needs to be a */
    /* prime number that is somewhat larger*/
    /* than 2**BITS.                       */
    private final int NEXT_CODE = 257;

    private long[] prefix_code = new
```

```
long[TABLE_SIZE];;        /* This array holds
the prefix codes   */
    private int[] append_character = new
int[TABLE_SIZE];  /* This array holds the
appended chars */
    private int[] decode_stack; /* This array
holds the decoded string */

    private int input_bit_count=0;
    private long input_bit_buffer=0; // must be
32 bits
    private int offset = 0;

    /*
    ** This routine simply decodes a string
from the string table, storing
    ** it in a buffer.  The buffer can then be
output in reverse order by
    ** the expansion program.
    */
    /* JN: returns size of buffer used
     */
    private int decode_string(int idx, long
code)
    {
    int i;
```

```
    i=0;
    while (code > (NEXT_CODE - 1))
        {
        decode_stack[idx++] =
append_character[(int) code];
        code=prefix_code[(int) code];
        if (i++>=MAX_CODE)
            {
            System.err.printf("Fatal error
during code expansion.\n");
            return 0;
            }
        }

    decode_stack[idx]= (int) code;

    return idx;
    }

    /*
    ** The following two routines are used to
output variable length
    ** codes.  They are written strictly for
clarity, and are not
    ** particularyl efficient.
    */
```

```
    long input_code(byte[] inputBuffer, int
inputLength, int dummy_offset, boolean
firstTime)
    {
    long return_value;

    //int pOffsetIdx = 0;
    if (firstTime)
        {
        input_bit_count = 0;
        input_bit_buffer = 0;
        }

    while (input_bit_count <= 24 && offset <
inputLength)
        {
        /*
        input_bit_buffer |= (long)
inputBuffer[offset++] << (24 -
input_bit_count);
        input_bit_buffer &= 0xFFFFFFFFL;
        System.out.printf("input buffer %d\n",
(long) inputBuffer[offset]);
        */
        // Java doesn't have unsigned types.
Have to play stupid games when mixing
        // shifts and type coercions
```

```
        long val = inputBuffer[offset++];
        if (val < 0) {
            val = 256 + val;
        }
        // System.out.printf("input buffer:
%d\n", val);
        //        if ( ((long) inpu) < 0)
System.out.println("Byte is -ve???");
        input_bit_buffer |= (((long) val) <<
(24 - input_bit_count)) & 0xFFFFFFFFL;
        //input_bit_buffer &= 0xFFFFFFFFL;
        // System.out.printf("input bit buffer
%d\n", input_bit_buffer);

        /*
        if (input_bit_buffer < 0) {
            System.err.println("Negative!!!");
        }
        */

        input_bit_count  += 8;
        }

    if (offset >= inputLength &&
input_bit_count < 12)
        return MAX_VALUE;
```

```
    return_value       = input_bit_buffer >>>
(32 - BITS);
    input_bit_buffer <<= BITS;
    input_bit_buffer &= 0xFFFFFFFFL;
    input_bit_count   -= BITS;

    return return_value;
    }

    void dumpLyric(int data)
    {
    System.out.printf("LZW: %d\n", data);
    if (data == 0xd)
        System.out.printf("\n");
    }

    /*
    **  This is the expansion routine.  It
takes an LZW format file, and expands
    **  it to an output file.  The code here
should be a fairly close match to
    **  the algorithm in the accompanying
article.
    */

    public int expand(byte[] intputBuffer, int
inputBufferSize, byte[] outBuffer)
```

```
    {
    long next_code = NEXT_CODE;/* This is the
next available code to define */
    long new_code;
    long old_code;
    int character;
    int string_idx;

    int offsetOut = 0;


    prefix_code      = new long[TABLE_SIZE];
    append_character = new int[TABLE_SIZE];
    decode_stack     = new int[4000];

    old_code= input_code(intputBuffer,
inputBufferSize, offset, true);  /* Read in the
first code, initialize the */
    character = (int) old_code;          /*
character variable, and send the first */
    outBuffer[offsetOut++] = (byte) old_code;
/* code to the output file                */
    //outTest(output, old_code);
    // dumpLyric((int) old_code);

    /*
    **  This is the main expansion loop.  It
```

```
reads in characters from the LZW file
    **  until it sees the special code used to
inidicate the end of the data.
    */
    while ((new_code=input_code(intputBuffer,
inputBufferSize, offset, false)) !=
(MAX_VALUE))
        {
        // dumpLyric((int)new_code);
        /*
        ** This code checks for the special
STRING+CHARACTER+STRING+CHARACTER+STRING
        ** case which generates an undefined
code.  It handles it by decoding
        ** the last code, and adding a single
character to the end of the decode string.
        */

        if (new_code>=next_code)
            {
            if (new_code > next_code)
                {
                System.err.printf("Invalid
code: offset:%X new:%X next:%X\n", offset,
new_code, next_code);
                break;
                }
```

```
            decode_stack[0]= (int) character;
            string_idx=decode_string(1,
old_code);
            }
        else
            {
            /*
            ** Otherwise we do a straight
decode of the new code.
            */

string_idx=decode_string(0,new_code);
            }

        /*
        ** Now we output the decoded string in
reverse order.
        */
        character=decode_stack[string_idx];
        while (string_idx >= 0)
            {
            int data = decode_stack[string_idx-
-];
            outBuffer[offsetOut] = (byte) data;
            //outTest(output, *string--);
```

```
            if (offsetOut % 4 == 0) {
                //dumpLyric(data);
            }

            offsetOut++;
            }

        /*
        ** Finally, if possible, add a new code
to the string table.
        */
        if (next_code > 0xfff)
            {
            next_code = NEXT_CODE;
            System.err.printf("*");
            }

        // test code
        if (next_code > 0xff0 || next_code <
0x10f)
            {
            Debug.printf("%02X ", new_code);
            }

        prefix_code[(int) next_code]=old_code;
        append_character[(int) next_code] =
(int) character;
```

```
        next_code++;

        old_code=new_code;
        }
    Debug.printf("offset out is %d\n",
offsetOut);
    return offsetOut;
    }
}
```

SongInformation.java

```java
public class SongInformation {


    // Public fields of each song record
    /**
     *  Song number in the file, one less than
in songbook
     */
    public long number;

    /**
     * song title in Unicode
     */
    public String title;

    /**
     * artist in Unicode
     */
    public String artist;

    /**
     * integer value of language code
     */
    public int language;

    public static final int  KOREAN = 0;
```

```
    public static final int  CHINESE1 = 1;
    public static final int  CHINESE2 = 2;
    public static final int  TAIWANESE3 = 3 ;
    public static final int  JAPANESE = 4;
    public static final int  RUSSIAN = 5;
    public static final int  THAI = 6;
    public static final int  TAIWANESE7 = 7;
    public static final int  CHINESE8 = 8;
    public static final int  CANTONESE = 9;
    public static final int  ENGLISH = 0x12;
    public static final int  VIETNAMESE = 0x13;
    public static final int  PHILIPPINE = 0x14;
    public static final int  TURKEY = 0x15;
    public static final int  SPANISH = 0x16;
    public static final int  INDONESIAN = 0x17;
    public static final int  MALAYSIAN = 0x18;
    public static final int  PORTUGUESE = 0x19;
    public static final int  FRENCH = 0x20;
    public static final int  INDIAN = 0x21;
    public static final int  BRASIL = 0x22;
    public static final int  CHINESE131 = 131;
    public static final int  ENGLISH146 = 146;
    public static final int  PHILIPPINE148 =
148;

    public SongInformation(long number,
                String title,
```

```
                String artist,
                int language) {
    this.number = number;
    this.title = title;
    this.artist = artist;
    this.language = language;
    }

    public String toString() {
    return "" + (number+1) + " (" + language +
") \"" + title + "\" " + artist;
    }

    public boolean titleMatch(String pattern) {
    // System.out.println("Pattern: " +
pattern);
    return title.matches("(?i).*" + pattern +
".*");
    }

    public boolean artistMatch(String pattern)
{
    return artist.matches("(?i).*" + pattern +
".*");
    }

    public boolean numberMatch(String pattern)
```

```
{
    Long n;
    try {
        n = Long.parseLong(pattern) - 1;
        //System.out.println("Long is " + n);
    } catch(Exception e) {
        //System.out.println(e.toString());
        return false;
    }
    return number == n;
    }


    public boolean languageMatch(int lang) {
    return language == lang;
    }
}
```

and Debug.java

```java
public class Debug {

    public static final boolean DEBUG = false;

    public static void println(String str) {
    if (DEBUG) {
        System.out.println(str);
    }
    }

    public static void printf(String format,
Object... args) {
    if (DEBUG) {
        System.out.printf(format, args);
    }
    }
}
```

To compile these, run

```
javac SongExtracter.java LZW.java Debug.java
SongInformation.java
```

and run by

```
java SongExtracter <song number >
```

The program to convert these MIDI files to Karaoke KAR files is KARConverter.java

```
/*
 * KARConverter.java
 *
 * The output from decodnig the Sonken data is
not in
 * the format required by the KAR "standard".
 * e.g. we need @T for the title,
 * and LYRIC events need to be changed to TEXT
events
 * Tempo has to be changed too
 *
 */

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

import javax.sound.midi.MidiSystem;
import
javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.Sequence;
import javax.sound.midi.Track;
import javax.sound.midi.MidiEvent;
import javax.sound.midi.MidiMessage;
import javax.sound.midi.ShortMessage;
import javax.sound.midi.MetaMessage;
```

```
import javax.sound.midi.SysexMessage;
import javax.sound.midi.Receiver;




public class KARConverter {
    private static int LYRIC = 5;
    private static int TEXT = 1;

    private static boolean firstLyricEvent =
true;

    public static void main(String[] args) {
    if (args.length != 1) {
        out("KARConverter: usage:");
        out("\tjava KARConverter <file>");
        System.exit(1);
    }
    /*
     *    args[0] is the common prefix of the
two files
     */
    File    inFile = new File(args[0] +
".mid");
    File    outFile = new File(args[0] +
".kar");
```

```
    /*
     *    We try to get a Sequence object,
which the content
     *    of the MIDI file.
     */
    Sequence    inSequence = null;
    Sequence    outSequence = null;
    try {
        inSequence =
MidiSystem.getSequence(inFile);
    } catch (InvalidMidiDataException e) {
        e.printStackTrace();
        System.exit(1);
    } catch (IOException e) {
        e.printStackTrace();
        System.exit(1);
    }

    if (inSequence == null) {
        out("Cannot retrieve Sequence.");
    } else {
        try {
        outSequence = new
Sequence(inSequence.getDivisionType(),

inSequence.getResolution());
```

```
        } catch(InvalidMidiDataException e) {
        e.printStackTrace();
        System.exit(1);
        }

        createFirstTrack(outSequence);
        Track[]    tracks =
inSequence.getTracks();
        fixTrack(tracks[0], outSequence);
    }
    FileOutputStream outStream = null;
    try {
        outStream = new
FileOutputStream(outFile);
        MidiSystem.write(outSequence, 1,
outStream);
    } catch(Exception e) {
        e.printStackTrace();
        System.exit(1);
    }
    }


    public static void fixTrack(Track oldTrack,
Sequence seq) {
    Track lyricTrack = seq.createTrack();
    Track dataTrack = seq.createTrack();
```

```
    int nEvent = fixHeader(oldTrack,
lyricTrack);
    System.out.println("nEvent " + nEvent);
    for ( ; nEvent < oldTrack.size(); nEvent++)
{
        MidiEvent event = oldTrack.get(nEvent);
        if (isLyricEvent(event)) {
        event = convertLyricToText(event);
        lyricTrack.add(event);
        } else {
        dataTrack.add(event);
        }
    }
    }

    public static int fixHeader(Track oldTrack,
Track lyricTrack) {
    int nEvent;

    // events at 0-10 are meaningless
    // events at 11, 12 should be the language
code,
    // but maybe at 12, 13
    nEvent = 11;
    MetaMessage lang1 = (MetaMessage)
(oldTrack.get(nEvent).getMessage());
```

```
    String val = new String(lang1.getData());
    if (val.equals("@")) {
        // try 12
        lang1 = (MetaMessage)
(oldTrack.get(++nEvent).getMessage());
    }
    MetaMessage lang2 = (MetaMessage)
(oldTrack.get(++nEvent).getMessage());
    String lang = new String(lang1.getData()) +
        new String(lang2.getData());
    System.out.println("Lang " + lang);
    byte[] karLang = getKARLang(lang);

    MetaMessage msg = new MetaMessage();
    try {
        msg.setMessage(TEXT, karLang,
karLang.length);
        MidiEvent evt = new MidiEvent(msg, 0L);
        lyricTrack.add(evt);
    } catch(InvalidMidiDataException e) {
    }

    // song title is next
    StringBuffer titleBuff = new
StringBuffer();
    for (nEvent = 15; nEvent < oldTrack.size();
nEvent++) {
```

```
        MidiEvent event = oldTrack.get(nEvent);
        msg = (MetaMessage)
(event.getMessage());
        String contents = new
String(msg.getData());
        if (contents.equals("@")) {
        break;
        }
        if (contents.equals("\r\n")) {
        continue;
        }
        titleBuff.append(contents);
    }
    String title = "@T" + titleBuff.toString();
    System.out.println("Title '" + title +"'");
    byte[] titleBytes = title.getBytes();

    msg = new MetaMessage();
    try {
        msg.setMessage(TEXT, titleBytes,
titleBytes.length);
        MidiEvent evt = new MidiEvent(msg, 0L);
        lyricTrack.add(evt);
    } catch(InvalidMidiDataException e) {
    }
```

```
    // skip the next 2 @'s
    for (int skip = 0; skip < 2; skip++) {
        for (++nEvent; nEvent <
oldTrack.size(); nEvent++) {
        MidiEvent event = oldTrack.get(nEvent);
        msg = (MetaMessage)
(event.getMessage());
        String contents = new
String(msg.getData());
        if (contents.equals("@")) {
            break;
        }
        }
    }

    // then the singer
    StringBuffer singerBuff = new
StringBuffer();
    for (++nEvent; nEvent < oldTrack.size();
nEvent++) {
        MidiEvent event = oldTrack.get(nEvent);
        if (event.getTick() != 0) {
        break;
        }
        if (! isLyricEvent(event)) {
        break;
        }
```

```
        msg = (MetaMessage)
(event.getMessage());
        String contents = new
String(msg.getData());
        if (contents.equals("\r\n")) {
        continue;
        }
        singerBuff.append(contents);
    }
    String singer = "@T" +
singerBuff.toString();
    System.out.println("Singer '" + singer
+"'");

    byte[] singerBytes = singer.getBytes();

    msg = new MetaMessage();
    try {
        msg.setMessage(1, singerBytes,
singerBytes.length);
        MidiEvent evt = new MidiEvent(msg, 0L);
        lyricTrack.add(evt);
    } catch(InvalidMidiDataException e) {
    }

    return nEvent;
```

```
    }

    public static boolean
isLyricEvent(MidiEvent event) {
    if (event.getMessage() instanceof
MetaMessage) {
        MetaMessage msg = (MetaMessage)
(event.getMessage());
        if (msg.getType() == LYRIC) {
        return true;
        }
    }
    return false;
    }

    public static MidiEvent
convertLyricToText(MidiEvent event) {
    if (event.getMessage() instanceof
MetaMessage) {
        MetaMessage msg = (MetaMessage)
(event.getMessage());
        if (msg.getType() == LYRIC) {
        byte[] newMsgData = null;
        if (firstLyricEvent) {
            // need to stick a \ at the front
            newMsgData = new
byte[msg.getData().length + 1];
```

```
            System.arraycopy(msg.getData(), 0,
newMsgData, 1, msg.getData().length);
            newMsgData[0] = '\\';
            firstLyricEvent = false;
        } else {
            newMsgData = msg.getData();
            if ((new
String(newMsgData)).equals("\r\n")) {
            newMsgData = "\\".getBytes();
            }
        }
        try {
            /*
            msg.setMessage(TEXT,
                   msg.getData(),
                   msg.getData().length);
            */
            msg.setMessage(TEXT,
                   newMsgData,
                   newMsgData.length);
        } catch(InvalidMidiDataException e) {
            e.printStackTrace();
        }
        }
    }
    return event;
    }
```

```
    public static byte[] getKARLang(String
lang) {
    System.out.println("lang is " + lang);
    if (lang.equals("12")) {
        return "@LENG".getBytes();
    }

    // don't know any other language specs, so
guess
    if (lang.equals("01")) {
        return "@LCHI".getBytes();
    }
    if (lang.equals("02")) {
        return "@LCHI".getBytes();
    }
    if (lang.equals("08")) {
        return "@LCHI".getBytes();
    }
    if (lang.equals("09")) {
        return "@LCHI".getBytes();
    }
    if (lang.equals("07")) {
        return "@LCHI".getBytes();
    }
    if (lang.equals("")) {
        return "@L".getBytes();
```

```
    }
    if (lang.equals("")) {
        return "@LENG".getBytes();
    }
    if (lang.equals("")) {
        return "@LENG".getBytes();
    }
    if (lang.equals("")) {
        return "@LENG".getBytes();
    }
    if (lang.equals("")) {
        return "@LENG".getBytes();
    }
    if (lang.equals("")) {
        return "@LENG".getBytes();
    }


    return ("@L" + lang).getBytes();
    }


    public static void copyNotesTrack(Track
oldTrack, Sequence seq) {
    Track newTrack = seq.createTrack();

    for (int nEvent = 0; nEvent <
```

```
oldTrack.size(); nEvent++)
        {
        MidiEvent event = oldTrack.get(nEvent);

        newTrack.add(event);
        }
    }

    public static void
createFirstTrack(Sequence sequence) {
    Track track = sequence.createTrack();
    MetaMessage msg1 = new MetaMessage();
    MetaMessage msg2 = new MetaMessage();

    byte data[] = "Soft Karaoke".getBytes();
    try {
        msg1.setMessage(3, data, data.length);
    } catch(InvalidMidiDataException e) {
        e.printStackTrace();
        return;
    }
    MidiEvent event = new MidiEvent(msg1, 0L);
    track.add(event);

    data = "@KMIDI KARAOKE FILE".getBytes();
    try {
        msg2.setMessage(1, data, data.length);
```

```
    } catch(InvalidMidiDataException e) {
        e.printStackTrace();
        return;
    }
    MidiEvent event2 = new MidiEvent(msg2, 0L);
    track.add(event2);
    }

    public static void output(MidiEvent event)
    {
    MidiMessage    message =
event.getMessage();
    long        lTicks = event.getTick();
    }



    private static void out(String strMessage)
    {
    System.out.println(strMessage);
    }
}



/*** KARConverter.java ***/
```

Playing MIDI files

The MIDI files extracted from the disk can be played using standard MIDI players such as Timidity. The lyrics are included and the melody line is in MIDI channel one. I've written a batch of Java programs using Swing and also the Java Sound framework which can play and do things to MIDI files. At the same time as playing MIDI files I can also do cool karaoke things like show the lyrics, show the notes that should be played and show progress through the lyrics. I'm still working on those, they will get posted later.

Playing WMA files

WMA files are "evil." They are based on two Microsoft proprietary formats. The first is the Advanced Systems Format (ASF) file format which describes the "container" for the music data. The second is the codec, Windows Media Audio 9.

The ASF is the primary problem. Microsoft have a published specification . This specification is strongly antagonistic to anything open source. The license states that if you build an implementation based on that specification then you:

- cannot distribute the source code
- can only distribute the object code
- cannot distribute the object code except as part of a "Solution" i.e. libraries seem to be banned
- cannot distribute your object code for no charge
- cannot set your license to allow derivative works

And what's more, you are not allowed to begin any new implementation after January 1, 2012 - and it is already May, 2012!

Just to make it a little worse, Microsoft have Patent 6041345 "Active stream format for holding multiple media streams" filed in Mar 7, 1997. The patent appears to cover the same ground as many other such formats which were in existence at the time, so the standing of this patent (were it to be challenged) is not clear. However, it has been used to block the GPL-licensed project VirtualDub from supporting ASF. The status of patenting a file format is a little suspect anyway, but may become a little clearer after Oracle wins or loses its claim to patent the Java API.

The FFmpeg project has nevertheless done a clean-room implementation of ASF, reverse-engineering the file format and not using the ASF specification at all. It has also reverse-engineered the WMA codec. This allows players such as mplayer and VLC to play ASF/WMA files. FFmpeg itself can also convert from ASF/WMA to better formats such as Ogg Vorbis.

There is no Java handler for WMA files, and given the license there is unlikely to be one unless it is based on FFmpeg.

The WMA files that I have extracted from the DVD have the following characteristics:

- Each file has two channels
- Each channel carries a mono signal

- The right channel carries all of the instruments, backing vocals and also the lead singer
- The left channel carries all of the instruments and backing vocals but not the lead singer

The Sonken player plays the right channel if no-one is singing into the microphones, but switches to the left channel (effectively muting the lead singer) as soon as someone sings into a microphone. Simple and effective.

The lyrics are still there in the track data as MIDI and can be extracted as before. They can be played by a MIDI player. I have no idea (yet) how to synchronise playing the MIDI and the WMA files.

KAR format

The resultant MIDI files are not in KAR format. This means that Karaoke players such as pykaraoke may have problems playing them. It is not too hard to convert the files to this format: loop through the sequence, writing or modifying MIDI events as appropriate. The program is not very exciting but is downloadable as KARConverter .

Playing songs with pykar

One of the simplest ways to play Karaoke Midi files is by using pykar . Regrettably, the songs ripped fromt he Sonken disk do not play properly - this is a mixture of bugs in pykar and features required which are not supplied. The problems and their solutions follow.

Tempo

Many Midi files will set the tempo explicitly using the Meta Event Set Tempo, 0x51. These files often do not. Pykar expects a Midi file to include this event, and otherwise defaults to a tempo of zero beats per minute. As might be expected, this trhows out all timing calculations performed by PyKar.

As The Sonic Spot explains, "If no set tempo event is present, 120 beats per minute is assumed" and gives a formula for calculating the appropriate tempo value, which is 60000000/120.

This requires one change to one PyKaraoke file: change line 190 of `pykar.py` from

```
sele.Tempo = [(0, 0)]
```

to

```
self.Tempo = [(0, 500000)]
```

Language encoding

The file `pykdb.py` clains that `cp1252` is the default character encoding for Karaoke files, ans uses a font `DejaVuSans.tt` which is appropriate for displaying such characters. This encoding adds in various European symbols such as 'á' in the top 128 bits of a byte, in addition to standard ASCII.

I'm not sure where PyKaraoke got that information from, but it certainly doesn't apply to Chinese Karaoke. I don't know what encodings Chinese, Japanese, Korean, etc use, but my code dumps them out as Unicode UTF-8. A suitable font for Unicode is Cyberbit.ttf. (See the Fonts chapter in my lecture notes on Global Software).

The file `pykdb.py` needs the lines

```
self.KarEncoding = 'cp1252'  # Default text
encoding in karaoke files
        self.KarFont =
FontData("DejaVuSans.ttf")
```

changed to

```
self.KarEncoding = 'utf-8'  # Default text
encoding in karaoke files
        self.KarFont = FontData("Cyberbit.ttf")
```

and a copy of `Cyberbit.tt` copied to the directory `/usr/share/pykaraoke/fonts/` .

Songs with no notes

Some songs on the disk have no MIDI notes, as this is all in a WMA file. The MIDI file only has the lyrics. PyKaraoke only plays upto the last note, which is at zero!. So no lyrics are played.

Conclusion

This chapter has discussed basically a forensics issue: how to get information off a DVD when the format of the files is not known. It doesn't have anything directly to do with playing sound, although it does give me a big source of files that I have already paid for.

I have a Malata 6619 Karaoke player. It has about 15,000 songs, but only about 500 English songs. My wife insists that the Chinese songs on it are of far higher quality than on my Sonken. I'm not so sure I agree, but anyway it's time to try to decode its file format.This is only partially successful: while I have got some information, I can't as yet play a MIDI file from the disk.

Resources

- Arirang MIDI Karaoke DVD storage file struct and MP3 Extraction by thanhth
- Decoding a Midi-DVD used by a Malata DVD player by Zhuowei
- DGTEFF - The definitive word in archive exploration

Files

The files on the disk are

```
AUDIO_TS      MALATAF3.BIN  MALATAS3.IDX
MULTAK.DA4
F6838.IDX     MALATAJ1.BIN  MALATAS4.IDX
MULTAK.DAT
FONT.BIN      MALATAM1.INF
FONTFT.BIN    MALATAM2.INF  MULTAK.DA1
VIDEO_TS
Makefile      MALATAM3.INF  MULTAK.DA2
SEARCH.IDX
MALATAF1.BIN  MALATARM.DAT  MULTAK.DA3
```

Song titles

The command `strings` shows that song titles are in several files, including `MALATAS4.IDX`. This seems to contain the most song titles, so I looked at that.

The block of song titles starts at 0x5F000. Before that is a bunch of nulls. To confirm this, the first song in my Malata songbook is "1001次吵架" and I can see the string "1001" as the first entry in that table.

This first song contains Chinese characters in the title, and the site GB Code Table by Mary Ansell confirms that they are encoded using GB2312.

The song titles are just concatenated, for example as "AlrightAmourAndyAngelAre You ReadyAsk For MoreBABY I'M YOUR MANBACK HOME". So there must be a table somewhere showing start and end of each title. I looked for any table giving offsets of the start of songs from 0x5F000. There is such a table, at 0x800! After playing with that for a while, it turns out that this table consists of records of 25 bytes. I'm not sure of the start of the records from 0x800: if I take a starting

offset of one, then bytes 19 and 20 hold the offset into the song title table while byte 25 is the length of the song title. But the offset could be higher.

The language appears to be specified in byte 11:

- 00 is Chinese (GB2132)
- 07 is English
- 01 is ???

The song number is a bit sneaky: in bytes 15-17 are 3 hexadecimal numbers. If they are concatenated then they are the song number. For example, for the song Medley One, the record is

```
00 00 00 6D 6F 00 00 00 00 04 07 20 00 02 02 01
35 01 7C A8 9E 02 02 00 0A
```

Bytes 15, 16, 17 are "02 01 35" (in hex) and this is the song number 20135 for the Beatles "Medley One".

The earliest English song is 20001 "7Days" and the last one is 20501 "Take Me To Your Heart"

We currently have

```
11    language
15-17 song number
19,20 offset into song title table
21,22 offset into artist name table?
25    length of title
```

e.g. for Medley One

```
00 00 00 6D 6F 00 00 00 00 04 07 20 00 02 02 01
35 01 7C A8 9E 02 02 00 0A
                              07          02 01
35 01 7C    9E 02
                              La
SongNumb SongT    Artis
```

I don't at present know what else is in these records.

A program to list the titles is `SongTitles.java`:

```
import java.io.FileInputStream;
import java.io.*;
import java.nio.charset.Charset;



class SongTitles {
    //private static int BASE = 0x5F23A;
    private static long OFFSET = 1; // must be
>= 1 or blows array bound
    private static long INDEX_BASE = 0x800 +
OFFSET;
    private static long TITLE_BASE = 0x5F000;
    private static int NUM = 12000;
    private static int INDEX_SIZE = 25;
    private static int TITLE_OFFSET = (int) (19
- OFFSET);
    private static int LENGTH_TITLE = (int) (25
- OFFSET);
    private static int ARTIST_OFFSET = (int)
(21 - OFFSET);
    private static int SONG_INDEX = (int) (15 -
OFFSET);

    public static void main(String[] args)
throws Exception {
```

```
    long nread = INDEX_BASE;
    byte[] titleBytes = new byte[512];
     FileInputStream fstream = new
FileInputStream("MALATAS4.IDX");
    fstream.skip(INDEX_BASE);

    byte[][] indexes = new byte[NUM][];
    long titleStart[] = new long[NUM];
    long titleEnd[] = new long[NUM];
    for (int n = 0; n < NUM; n++) {
        indexes[n] = new byte[INDEX_SIZE];
        fstream.read(indexes[n]);
        if (isNull(indexes[n]))
        break;
        // printIndex(indexes[n]);
        nread += INDEX_SIZE;

        if (isNull(indexes[n]))
        break;

        byte b1 = indexes[n][TITLE_OFFSET];
        byte b2 = indexes[n][TITLE_OFFSET+1];
        // corect for negative
        int first, second;
        //System.out.printf("%X %X\n", firstB,
secondB);
        //first = firstB >= 0 ? firstB : 256 -
```

```
firstB;
        //second = secondB >= 0 ? secondB : 256
- secondB;

        titleStart[n] = ((b1 >= 0 ? b1 : 256 +
b1) << 8) + (b2 >= 0 ? b2 : 256 + b2); //first
* 256 + second;
        if (titleStart[n] > 0xfff) {
        // System.out.println("too big");
        titleStart[n] &= 0xfff;
        } else if (titleStart[n] < 0) {
        System.out.println("too small");
        }

        long end = indexes[n][LENGTH_TITLE];
        if (end >= 0x80) {
        end -= 0x80;
        // System.out.println("End too big");
        } else if (end < 0) {
        // System.out.println("End negative");
        end += 128;
        }
        titleEnd[n] = end;

        /*
        System.out.printf("Numbers %X %X %X
%X\n", b1, b2,
```

```
                    titleStart[n],
                    titleStart[n]+titleEnd[n]);
        */
    }

    System.out.printf("Skip to %X\n",
(TITLE_BASE - nread));
    fstream.skip(TITLE_BASE - nread);
    nread = TITLE_BASE;

    for (int n = 0; n < NUM-1; n++) {
        // int len = (int) (titleStart[n+1]-
titleStart[n]);
        int len = (int) titleEnd[n];
        //System.out.println("Reading " + len);
        fstream.read(titleBytes, 0, len);

        Charset charset =
Charset.forName("gb2312");
        String translated = new
String(titleBytes, 0, len, charset);

        printFullIndex(indexes[n]);
        System.out.print(" SongIndex ");
        printSongIndex(indexes[n]);
        System.out.print(" ArtistIndex ");
        printArtistIndex(indexes[n]);
```

```
        System.out.println("" + n + ": "
+translated);
        //printIndex(indexes[n]);
    }

    /*
    fstream.skip(TITLE_BASE);

    byte[] bytes = new byte[NUM];
    fstream.read(bytes);

    Charset charset =
Charset.forName("gb2312");
    String translated = new String(bytes,
charset);

    System.out.println(translated);
    */
    }

    private static void printFullIndex(byte[]
bytes) {
    for (int n = 0; n < bytes.length; n++) {
        System.out.printf("%02X ", bytes[n]);
    }
    }
```

```
    private static void printArtistIndex(byte[]
bytes) {
    System.out.printf("%02X%02X ",
              bytes[ARTIST_OFFSET],
              bytes[ARTIST_OFFSET+1]);
    }

    private static void printSongIndex(byte[]
bytes) {
    System.out.printf("%X%02X%02X ",
              bytes[SONG_INDEX],
              bytes[SONG_INDEX+1],
              bytes[SONG_INDEX+2]);
    }

    private static void print1byte(byte[]
bytes, int index) {
    System.out.printf("%02X ", bytes[index]);
    }

    private static boolean isNull(byte[] bytes)
{
    for (int n = 0; n < bytes.length; n++) {
        if (bytes[n] != 0)
        return false;
    }
    return true;
```

```
    }
}
```

One table finishes at 0xF5260, maybe starting another at 0xF5800. Another table starts at 0x88000 and finishes about 0x9ADF0. Another starts at 0x9B000 and finishes at 0x9B2B0 I don't know what is in these tables.

Song data

Most of this section was discovered by thanth . However, he only deals with a single data file, and as the Malata has more, it becomes more complex.

There are four data files: MULTAK.DAT, MULTAK.DA1, MULTAK.DA2 and MULTAK.DA3. The primary data file is MULTAK.DAT, and this contains tables of pointers to song data. The other files seem to just contain the song data.

The number of songs (minus one) is in byte-swapped order at 0x14E in MULTAK.DAT. In my files, this is "FB 3D" which when swapped to "3D FB" is one less than the number of songs, 0x3DFC (15868). This was identified by thanth.

Starting at 0xD20 is a table of 4 byte numbers (prefixed by "FF 00 FF FF" which are indexes into the table of song data. If the bytes are "b0 b1 b2 b3" then thanth discovered that the song data starts at

```
(((b0 * 0x3C) + b1) * 0x48 +b2) * 0x800 +
0x10000
```

Actually, it is more complex than that: for the songs with data on the first disk MULTAK.DAT this is the case. The table also contains pointers to data in the other files, and for these the formula is

```
(((b0 * 0x3C) + b1) * 0x48 +b2) * 0x800
```

That is, the data in these later files starts immediately with no offset.

The file for each song is given in the top half of the fourth byte of the song index: (b3 >> 4), where zero is MULTAK.DAT, one is MULTAK.DA1, etc.

At the locations of the song data pointers is either the phrase "OK" which means a "simple song" or "FF FF" which means "complex song", according to thanth. Simple songs just contain lyrics and MIDI data, while complex songs also have MP3 data. I haven't yet found any information about the size of the data for each song.

The program `SongData.java` splits the MULTAK files into individual song data files. It only saves a part of the data for each song, since I don't know where the data finishes.

```
import java.io.FileInputStream;
import java.io.*;
import java.nio.charset.Charset;



class SongData {
    private static int MAX_SONGS = 20000;

    private static long OFFSET = 0;
    private static long INDEX_BASE = 0xD20 +
OFFSET;

    private static long MULTAK_SIZES[] =
{0x2441D800,
                        943441920,
                        943405056,
                        943812608,
                        58099712};
    //private long offset = 0x10000;

    private enum SongType {
    SIMPLE, COMPLEX;
    }

    private static long TITLE_BASE = 0x5F000;
```

```
    private static int NUM = 12000;
    private static int INDEX_SIZE = 25;
    private static int TITLE_OFFSET = (int) (19
- OFFSET);
    private static int LENGTH_TITLE = (int) (25
- OFFSET);
    private static int ARTIST_OFFSET = (int)
(21 - OFFSET);
    private static int SONG_INDEX = (int) (15 -
OFFSET);

    private class SongStart {
    public long start;
    public int fileNumber;
    public int songNumber;
    public byte[] indexBytes;
    public byte[] data;
    public SongType type;
    }


    public static void main(String[] args)
throws Exception {
    new SongData();
    }

    public SongData() throws Exception {
```

```
    long nread = INDEX_BASE;
    long index;
    int[] bytes;
    int[] ibytes = new int[4];
    int numSongs = 0;

    SongStart songStarts[] = new
SongStart[MAX_SONGS];

     FileInputStream fstream = new
FileInputStream("MULTAK.DAT");
    fstream.skip(INDEX_BASE);
    bytes = read4(fstream);
    long lval = 0;
    int currentFileNumber = 0;

    while (numSongs < MAX_SONGS) {
        // System.out.printf("%X\n", index);
        bytes = read4(fstream);
        if (isNull(bytes)) {
        System.out.printf("Read %d songs\n",
numSongs);
        break;
        }

        if (isFF00FFFF(bytes)) {
        // these seem to occur sometimes e.g.
```

```
at A8B8
        continue;
        }

        songStarts[numSongs] = new SongStart();
        songStarts[numSongs].songNumber =
numSongs;
        songStarts[numSongs].fileNumber =
bytes[3] >> 4;
        if ((bytes[3] & 0xF) == 0xA) {
        songStarts[numSongs].type =
SongType.COMPLEX;
        } else {
        songStarts[numSongs].type =
SongType.SIMPLE;
        }
        songStarts[numSongs].start =
songStart(bytes);

        /*
        // update fileNumber number?
        if (numSongs > 0 &&
        songStarts[numSongs-1].start >
songStarts[numSongs].start) {

        offset = 0;
        // may need to reset this since offset
```

```
may have changed
        songStarts[numSongs].start =
songStart(bytes);
        currentFileNumber++;
        }
        */

        // songStarts[numSongs].fileNumber =
currentFileNumber;
        System.out.printf("Song %d starts %X
fileNumber %X (",
                  numSongs,
                  songStarts[numSongs].start,

songStarts[numSongs].fileNumber);
        printBytes(bytes);
        System.out.println(')');

        numSongs++;
    }
    fstream.close();

    for (int n = 0; n < numSongs; n++) {
        System.out.printf("Number %d start %X
bytes ",
                  n, songStarts[n].start);
        getSongFromStart(songStarts[n]);
```

```
        System.out.println();

        saveSong(songStarts[n]);
    }

    /*
    fstream =  new
FileInputStream("MULTAK.DAT");
    long totalRead = 0;
    for (int n = 0; n < MAX_SONGS; n++) {
        fstream.skip(songStarts[n] -
totalRead);
        totalRead = songStarts[n];
        System.out.printf("Skipped to %X %X\n",
songStarts[n], totalRead);
        // check next song
        bytes = read4(fstream);
        totalRead += 4;

        for (n = 0; n < 4; n++) {
        System.out.printf("   %X\n", bytes[n]);
        ibytes[n] = (bytes[n] >= 0 ? bytes[n] :
256 + bytes[n]);
        }

        lval = 0;
        for (n = 0; n < 4; n++) {
```

```
        lval = (lval << 8) + bytes[n];
        }
        System.out.printf("  next bytes %X\n",
lval);
    }
    */

    }

    private int[] read4(FileInputStream f)
throws Exception {
    byte[] bytes = new byte[4];
    int[] ibytes = new int[4];
    long ret = 0;

    f.read(bytes);

    // ensure they are unsigned bytes
    for (int n = 0; n < 4; n++) {
        ibytes[n] = (bytes[n] >= 0 ? bytes[n] :
256 + bytes[n]);
    }
    return ibytes;
    }

    private long songStart(int[] indexBytes) {
    long offset;
```

```
    int fileNumber = indexBytes[3] >> 4;
    if (fileNumber == 0)
        offset = 0x10000;
    else
        offset = 0;

    long idx =  ((indexBytes[0] * 0x3C) +
             indexBytes[1]) * 0x4B +
indexBytes[2];
    return idx * 0x800 + offset;
    }

    private void getSongFromStart(SongStart
songInfo) throws Exception {
    String fname;

    if (songInfo.fileNumber == 0) {
        fname = "MULTAK.DAT";
    } else {
        fname = "MULTAK.DA" +
songInfo.fileNumber;
    }

     FileInputStream fstream = new
FileInputStream(fname);
    fstream.skip(songInfo.start);
```

```
    int[] bytes = read4(fstream);
    if (bytes[0] == 0xFF && bytes[1] == 0xFF) {
        songInfo.type = SongType.COMPLEX;
    } else {
        songInfo.type = SongType.SIMPLE;
    }
    songInfo.data = new byte[0x4B00];
    fstream.read(songInfo.data);
    fstream.close();
    }

    private void saveSong(SongStart songInfo)
throws Exception {
    String fname = "songs/" +
songInfo.songNumber;
     FileOutputStream fstream = new
FileOutputStream(fname);
    if (songInfo.type == SongType.SIMPLE)
        fstream.write(new byte[] {0, 0, 'O',
'K'});
    else
        fstream.write(new byte[] {(byte)0xFF,
(byte)0xFF, 0, 0});

    fstream.write(songInfo.data);
    fstream.close();
    }
```

```
    private boolean greaterThanEqual(int[] x,
int[] y) {
    for (int n = 0; n < x.length; n++) {
        if (x[n] < y[n]) {
        return false;
        }
    }
    return true;
    }

    private void printFullIndex(byte[] bytes) {
    for (int n = 0; n < bytes.length; n++) {
        System.out.printf("%02X ", bytes[n]);
    }
    }

    private void printArtistIndex(byte[] bytes)
{
    System.out.printf("%02X%02X ",
              bytes[ARTIST_OFFSET],
              bytes[ARTIST_OFFSET+1]);
    }

    private void printSongIndex(byte[] bytes) {
    System.out.printf("%X%02X%02X ",
              bytes[SONG_INDEX],
```

```
                bytes[SONG_INDEX+1],
                bytes[SONG_INDEX+2]);
    }

    private void print1byte(byte[] bytes, int
index) {
    System.out.printf("%02X ", bytes[index]);
    }

    private void printBytes(int[] bytes) {
    for (int n = 0; n < bytes.length; n++) {
        System.out.printf("%02X ", bytes[n]);
    }
    }

    private boolean isNull(int[] bytes) {
    for (int n = 0; n < bytes.length; n++) {
        if (bytes[n] != 0)
        return false;
    }
    return true;
    }

    private boolean isFF00FFFF(int[] bytes) {
    if (bytes[0] == 0xFF &&
        bytes[2] == 0xFF &&
        bytes[3] == 0xFF)
```

```
        return true;
    else
        return false;
    }
}
```

Linking songs in the song book to song data

The file MULATAS4.IDX, we have determined, holds a table of 25 byte blocks starting at 0x800. The current knowledge of these blocks is 11language (00 is GB2132, 07 is English)15-17song number in song book (read the hex number as decimal)19-20offset of song title into table at 0x5F00025length of title

The song data is spread across files MULTAK.DAT - MULTAK.DA4. At 0xd20 in MULTAK.DAT is a table of 4 bytes indexes. Each index translates into a starting location in the various data files. These are of type "simple" (probably MIDI data only) and "complex" (MIDI plus MP3).

The problem now is to link the song information to its data.

Using the starting points of each data file, we can extract, say, a few thousand bytes from each file. The correct length is at present unknown. I saved them them as files 1, 2, 3, ..., 15460.

With them all in separate files, I tried to see if any of them were recognisable. Completely by fluke by running `strings` looking for Fool on the Hill (which I knew was on

the disk), I hit upon song 10397 by searching for "hill", with `bvi` showing

```
00000000  00 00 4F 4B 00 00 00 00 00 00 00 00
00 00 00 00 ..OK............
00000010  00 00 00 00 00 20 00 00 00 07 00 00
00 00 00 00 ..... ..........
00000020  00 00 91 06 00 00 20 11 00 00 77 48
41 54 00 64 ...... ...wHAT.d
00000030  52 45 41 4D 53 00 61 52 45 00 6D 41
44 45 00 6F REAMS.aRE.mADE.o
00000040  46 0F 6F 72 69 67 69 6E 61 6C 1A 0F
68 49 4C 4C F.original..hILL
00000050  41 52 59 00 64 55 46 46 20 20 20 52
20 25 C8 2E ARY.dUFF   R %..
```

This song isn't in my song book, but it is in the list I pulled out of MALATAS4.IDX:

```
00 00 00 77 64 61 6D 6F 00 04 07 00 00 05 02 04
42 01 87 CE F1 02 05 80 17  SongIndex 20442
ArtistIndex F102 10415: What Dreams Are Made Of
```

So song title 20442 has its song data in file 10397.

With this clue, doing something like `strings -f -n 30 *` quickly shows up other files with english text. Enough to draw up a table IDSong # in bookIndex to dataSong titleS12024710202NEXT 100 YEARSS22042810383LoversS32044210397What dreams are...S42015410109Only sleeping

and this is just linear:

```
data index = song number - 10045
```

NB: this only works for some songs - I guess the English ones! If this pattern holds for all English songs, the earliest data file is (20001-10045) = 9956 and the last one is (20501-10045) = 10456.

Decoding song data

It would be easy if all songs contained English text in the data. But I only found the above four. So the rest must be encoded in some way.

A couple that I looked at seemed a bit messy. So I settled (for no particular reason) on Don Gibson's "Oh Lonesome Me". By playing the song on the Malata, the headers were

```
Oh Lonesome Me
ORIGINAL:
Don Gibson
```

while the lyrics are

```
Every body's goin'
out and havin' fun
I'm just a fool
for stayin' home
and havinnone
```

Running `bvi` on the data file gives

```
00000000  00 00 4F 4B 00 00 00 00 00 00 00 00
00 00 00 00 ..OK............
00000010  00 00 00 00 00 20 00 00 00 07 00 00
00 00 00 00 ..... ..........
00000020  00 00 6D 05 00 00 33 0F 00 00 7C 5B
13 7F 5C 5D ..m...3...|[..\]
00000030  56 40 5C 5E 56 13 7E 56 1C 7C 61 7A
74 7A 7D 72 V@\^V.~V.|aztz}r
00000040  7F 09 1C 77 5C 5D 13 74 5A 51 40 5C
5D 1C 33 33 ...w\].tZQ@\].33
00000050  33 1F 33 36 27 37 33 33 32 33 33 33
05 30 30 32 3.36'7332333.002
00000060  33 33 3D 0B 35 3A 32 33 33 7F 17 37
3C 32 33 32 33=.5:233..7<232
```

From the songs with English, I know the song title starts at 0x2A, and we have to do this match:

```
7C 5B 13 7F 5C 5D 56 40 5C 5E 56 13 7E 56 1C 7C
61 7A 74 7A 7D 72
O  h     L  o  n  e  s  o  m  e     M  e  /  O
R  I  G  I  N  A

7F 09 1C 77 5C 5D 13 74 5A 51 40 5C 5D 1C 33 33
L  :  /  D  o  n     G  i  b  s  o  n  /
```

The obvious thing to try is a substitution cipher: for example, the 'o's are encoded as 0x5c while the 'O's are 0x7c. It's a game of pattern matching, and the answer is the following piece of C code

```c
#include <stdio.h>

int main(int argc, char **argv) {
    FILE *fp;
    FILE *ofp;

    if (argc == 1) {
    fp = stdin;
    ofp = stdout;
    } else if (argc == 2) {
    fp = fopen(argv[1], "r");
    ofp = stdout;
    } else {
    fp = fopen(argv[1], "r");
    ofp = fopen(argv[2], "w");
    }

    char ch;
    int n = 0;

    while (n++ < 1500) {
    ch = getc(fp);

    switch (ch) {
    case 0x13: ch = ' '; break;
```

```
    case 0x52: ch = 'a'; break;
    case 0x51: ch = 'b'; break;
    case 0x50: ch = 'c'; break;

    case 0x57: ch = 'd'; break;
    case 0x56: ch = 'e'; break;
    case 0x55: ch = 'f'; break;
    case 0x54: ch = 'g'; break;

    case 0x5B: ch = 'h'; break;
    case 0x5A: ch = 'i'; break;
    case 0x59: ch = 'j'; break;
    case 0x58: ch = 'k'; break;

    case 0x5F: ch = 'l'; break;
    case 0x5E: ch = 'm'; break;
    case 0x5D: ch = 'n'; break;
    case 0x5C: ch = 'o'; break;

    case 0x43: ch = 'p'; break;
    case 0x42: ch = 'q'; break;
    case 0x41: ch = 'r'; break;
    case 0x40: ch = 's'; break;

    case 0x47: ch = 't'; break;
    case 0x46: ch = 'u'; break;
    case 0x45: ch = 'v'; break;
```

```
    case 0x44: ch = 'w'; break;

    case 0x4A: ch = 'y'; break;

        // These aren't organised yet
        //case 0x: ch = ''; break;

    case 0x72: ch = 'A'; break;
    case 0x71: ch = 'B'; break;
    case 0x70: ch = 'C'; break;

    case 0x77: ch = 'D'; break;
    case 0x76: ch = 'E'; break;
    case 0x75: ch = 'F'; break;
    case 0x74: ch = 'G'; break;

    case 0x7B: ch = 'H'; break;
    case 0x7A: ch = 'I'; break;
    case 0x79: ch = 'J'; break;
    case 0x78: ch = 'K'; break;

    case 0x7F: ch = 'L'; break;
    case 0x7E: ch = 'M'; break;
    case 0x7D: ch = 'N'; break;
    case 0x7C: ch = 'O'; break;

    case 0x63: ch = 'P'; break;
```

```
    case 0x62: ch = 'Q'; break;
    case 0x61: ch = 'R'; break;
    case 0x60: ch = 'S'; break;

    case 0x67: ch = 'T'; break;
    case 0x66: ch = 'U'; break;
    case 0x65: ch = 'V'; break;
    case 0x64: ch = 'W'; break;

    case 0x6B: ch = 'X'; break;
    case 0x6A: ch = 'Y'; break;
    case 0x69: ch = 'Z'; break;

    case 0x09: ch = ':'; break;

    case 0x14: ch = '\''; break;

    default: ch = '.'; break;
    }
    putc(ch, ofp);
    }
    fclose(ofp);
    exit(0);
}
```

The substitutions here are grouped in fours, but of course there is no reason why they should be. (This is repeated in some other decodings, but not all.).

Following application of that substitution, the file looks like

```
00000000  2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E
2E 2E 2E 2E ................
00000010  2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E
2E 2E 2E 2E ................
00000020  2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 4F 68
20 4C 6F 6E ..........Oh Lon
00000030  65 73 6F 6D 65 20 4D 65 2E 4F 52 49
47 49 4E 41 esome Me.ORIGINA
00000040  4C 3A 2E 2E 6F 6E 20 47 69 62 73 6F
6E 2E 2E 2E L:..on Gibson...
00000050  2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E
2E 2E 2E 2E ................
00000060  2E 2E 2E 2E 2E 2E 2E 2E 2E 4C 2E 2E
2E 2E 2E 2E .........L......
00000070  2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E
2E 2E 2E 2E ................
00000080  2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E
2E 2E 2E 2E ................
00000090  45 2E 76 2E 65 2E 72 2E 79 2E 20 2E
62 2E 6F 2E E.v.e.r.y. .b.o.
000000A0  64 2E 79 2E 27 2E 73 2E 20 2E 67 2E
6F 2E 69 2E d.y.'.s. .g.o.i.
000000B0  6E 2E 27 2E 2E 2E 2E 2E 2E 2E 2E 2E
2E 49 2E 27 n.'..........I.'
000000C0  2E 6D 2E 20 2E 6A 2E 75 2E 73 2E 74
2E 20 2E 61 .m. .j.u.s.t. .a
```

```
000000D0  2E 20 2E 66 2E 6F 2E 6F 2E 6C 2E 2E
2E 2E 2E 2E . .f.o.o.l......
000000E0  2E 2E 2E 61 2E 6E 2E 64 2E 20 2E 68
2E 61 2E 76 ...a.n.d. .h.a.v
000000F0  2E 2E 2E 69 2E 6E 2E 6E 2E 6F 2E 6E
2E 65 2E 2E ...i.n.n.o.n.e..
00000100  2E 2E 2E 2E 2E 2E 2E 2E 68 2E 6F 2E
77 2E 20 2E ........h.o.w. .
00000110  2E 2E 73 2E 68 2E 65 2E 20 2E 73 2E
65 2E 74 2E ..s.h.e. .s.e.t.
00000120  20 2E 2E 2E 6D 2E 65 2E 20 2E 66 2E
72 2E 65 2E  ...m.e. .f.r.e.
00000130  65 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 54
2E 68 2E 61 e..........T.h.a
00000140  2E 74 2E 20 2E 6D 2E 69 2E 73 2E 74
2E 61 2E 6B .t. .m.i.s.t.a.k
```

Which is much more readable! it doesn't quite follow the lyrics though - an issue for later.

The next occurrence of this substitution cipher is at file 10281, song number 20326 "Give me all night" so there are other substitutions used, of course! Then at 10326, song number 20371 "Stand By Your Man"

So then I tried another song, California dreaming, song 20088, file 10043. That was pretty straightforward. Other songs using this substitution are 20033 Heartbreaker, file 9988, 20082 Another Girl, file 10037, 20213 Don't talk, file 10168, 20382 Things we Said Today, file 10336. I also tried some others: no pattern as to song numbers.

However, the label for the substitution pattern appears to be byte 0x26. Here we another coincidence: the byte which is mapped to the space character ' ' is 0x20 less than byte 0x26. e.g. the substitution for songs like file 10337 is

```
switch (ch) {
        case 0x95: ch = ' '; break;

        case 0xD4: ch = 'a'; break;
        case 0xD7: ch = 'b'; break;
        case 0xD6: ch = 'c'; break;

        case 0xD1: ch = 'd'; break;
        case 0xD0: ch = 'e'; break;
        case 0xD3: ch = 'f'; break;
        case 0xD2: ch = 'g'; break;

        case 0xDD: ch = 'h'; break;
        case 0xDC: ch = 'i'; break;
        case 0xDF: ch = 'j'; break;
        case 0xDE: ch = 'k'; break;

        case 0xD9: ch = 'l'; break;
        case 0xD8: ch = 'm'; break;
        case 0xDB: ch = 'n'; break;
        ...
```

and byte 0x26 for that file is 0xB5, and 0x20 = 0xB5 - 0x95. Maybe the pattern is just based on these bytes (e.g. what is byte 0x27 - an index into pattern types?).

To be followed up...

There is one critival issue in this: it isn't only the English alphabetic characters that are encoded: others are too. But I have no clues as to what the other 65000+ Unicode characters should be!

MIDI from data file

Here is where I am currently stuck. The data files are *not* MIDI files. For example, song 10383 Lovers by Abba is

```
00000000  00 00 4F 4B 00 00 00 00 00 00 00 00
00 00 00 00 ..OK............
00000010  00 00 00 00 00 24 00 00 00 07 00 00
00 00 00 00 .....$..........
00000020  00 00 77 07 00 00 00 12 00 00 4C 6F
76 65 72 73 ..w.......Lovers
00000030  2F 4F 52 49 47 49 4E 41 4C 3A 2F 41
62 62 61 2F /ORIGINAL:/Abba/
00000040  28 4C 69 76 65 20 41 20 4C 69 74 74
6C 65 20 4C (Live A Little L
00000050  6F 6E 67 65 72 29 2F 00 00 00 56 00
06 EB 0A 00 onger)/...V.....
00000060  00 01 00 00 00 36 05 01 01 00 00 17
2A 06 02 01 .....6......*...
00000070  00 00 38 2A 04 03 01 00 00 46 36 05
04 01 00 00 ..8*.....F6.....
00000080  88 39 05 05 01 00 00 A8 36 05 06 01
00 00 F1 2A .9......6......*
00000090  04 07 01 00 01 16 53 02 08 01 00 01
24 42 04 09 ......S.....$B..
000000A0  01 00 01 32 23 06 11 CA 00 0D 5B 00
0D 62 81 80 ...2#.....[..b..
000000B0  05 82 9D 09 30 07 18 07 18 07 18 07
18 07 00 26 ....0..........&
000000C0  01 5C 00 53 04 69 04 74 06 20 00 64
07 6F 05 77 .\.S.i.t. .d.o.w
```

```
000000D0  05 6E 07 20 00 61 09 6E 07 64 09 20
00 6C 0B 69 .n. .a.n.d. .l.i
000000E0  09 73 08 74 08 65 09 6E 08 5E 01 00
1D 01 1E 26 .s.t.e.n.^.....&
```

This isn't a MIDI file. From my Sonken, I have a MIDI file for this song (obviously not the same recording) which looks like

```
00000000  4D 54 68 64 00 00 00 06 00 01 00 03
00 1E 4D 54 MThd..........MT
00000010  72 6B 00 00 00 2B 00 FF 03 0C 53 6F
66 74 20 4B rk...+....Soft K
00000020  61 72 61 6F 6B 65 00 FF 01 13 40 4B
4D 49 44 49 araoke....@KMIDI
00000030  20 4B 41 52 41 4F 4B 45 20 46 49 4C
45 00 FF 2F  KARAOKE FILE../
00000040  00 4D 54 72 6B 00 00 1A 5E 00 FF 01
05 40 4C 45 .MTrk...^....@LE
00000050  4E 47 00 FF 01 1E 40 54 4C 6F 76 65
72 73 28 4C NG....@TLovers(L
00000060  69 76 65 20 61 20 4C 69 74 74 6C 65
20 4C 6F 6E ive a Little Lon
00000070  67 65 72 29 00 FF 01 02 40 54 8A 5B
FF 01 02 5C ger)....@T.[...\
00000080  53 06 FF 01 01 69 06 FF 01 01 74 09
FF 01 01 20 S....i....t....
00000090  06 FF 01 01 64 06 FF 01 01 6F 06 FF
01 01 77 06 ....d....o....w.
000000A0  FF 01 01 6E 07 FF 01 01 20 07 FF 01
01 61 07 FF ...n.... ....a..
000000B0  01 01 6E 07 FF 01 01 64 0A FF 01 01
20 07 FF 01 ..n....d.... ...
000000C0  01 6C 07 FF 01 01 69 07 FF 01 01 73
0A FF 01 01 .l....i....s....
```

```
000000D0  74 0A FF 01 01 65 0A FF 01 01 6E 00
FF 01 01 5C t....e....n....\
```

and this *is* a conforming MIDI file.

Just looking at the lyric part, MIDI files require

```
<delta> FF 01 <string length>
```

It is likely that the Malata just has

```
<delta>  single-char
```

That's easy to adjust - if we know where they lyrics stop!

The lyrics start after a sequence "18 07 18 07 18 07 18 07 00 26".

A much trickier problem is that the lyrics are not contiguous! They should be

```
Sit down and listen
'cause I've got
good news for you
It was in the
papers today
```

But if we look at the file, lines 2 and 4 are missing:

```
000000C0  01 5C 00 53 04 69 04 74 06 20 00 64
07 6F 05 77 .\.S.i.t. .d.o.w
000000D0  05 6E 07 20 00 61 09 6E 07 64 09 20
00 6C 0B 69 .n. .a.n.d. .l.i
000000E0  09 73 08 74 08 65 09 6E 08 5E 01 00
1D 01 1E 26 .s.t.e.n.^.....&
000000F0  01 67 05 6F 05 6F 05 64 07 20 00 6E
08 65 05 77 .g.o.o.d. .n.e.w
00000100  04 73 07 20 00 66 09 6F 07 72 09 20
00 79 09 6F .s. .f.o.r. .y.o
00000110  05 75 06 5E 01 00 10 01 33 26 01 70
0D 61 0C 70 .u.^....3&.p.a.p
```

For the missing line 2, there is some sort of pointer "09 6E 08 5E 01 00 1D 01 1E 26", and it turns out that the missing lines are elsewhere:

```
00000790  68 19 5E 01 00 81 C9 1A FF 85 21 26
02 27 02 63 h.^.......!&.'.c
000007A0  03 61 02 75 03 73 02 65 04 20 00 49
04 27 04 76 .a.u.s.e. .I.'.v
000007B0  03 65 05 20 00 67 0A 6F 08 74 08 5E
02 00 10 02 .e. .g.o.t.^....
000007C0  4C 26 02 49 06 74 08 20 00 77 09 61
07 73 09 20 L&.I.t. .w.a.s.
000007D0  00 69 07 6E 06 20 00 74 07 68 04 65
04 5E 02 00 .i.n. .t.h.e.^..
```

The Malata displays two lines at a time. All of the first lines form a chunk. The second lines form another chunk, later on. I haven't found an offset or length to say where this second chunk occurs. Each line appears to consist of a delta value (I guess) for the delay of the lyric, followed by a lyric character.

Between lines is a section that starts with the character '^' and finishes with the character '&'. I haven't any idea what is in these sections apart from two observations:

- The character 0xFF appears twice: the first time it signals a shift from first line lyrics to second line lyrics. The next time it occurs it signals the end of the lyrics.

- Several times the sequence "09 6E 08 5E 01 00 1D 01 1E 26" occurs. This seems to signal a long break (e.g. for a solo) in the lyrics. In the "Lovers" song this occurs several times: it occurs in the first line, but sometimes the second line is played immediately, sometimes it is delayed until after the next first line is played. I haven't resolved this yet :-(

The file `printLyrics.c` gives a dump fo the lyrics for songs like Lovers (non-coded lyrics) plus the deltas (assumed) and the stuff between lines

```
#include <stdio.h>

#define NUM_LINES 100
#define LINE_LEN 25

typedef enum {BEFORE_SONG, IN_SONG_AND_LINE,
IN_SONG_BETWEEN} state_t;

state_t state = BEFORE_SONG;
int half = 1;

FILE *fp;
FILE *ofp;

unsigned char prev_ch, curr_ch;
unsigned char lines[NUM_LINES][LINE_LEN];
unsigned char separator[NUM_LINES][LINE_LEN];
unsigned char deltas[NUM_LINES][LINE_LEN];

unsigned char *first_lines,
    *second_lines,
    *first_separators,
    *second_separators,
    *first_deltas,
    *second_deltas;
```

```
int max_lines;

void read1() {
    prev_ch = curr_ch;
    curr_ch = getc(fp);
    // fprintf(ofp, "%X %X\n", prev_ch,
curr_ch);
}

void read2() {
  prev_ch = getc(fp);
  curr_ch = getc(fp);
    // fprintf(ofp, "%X %X\n", prev_ch,
curr_ch);
}



void printLine(unsigned char *line) {
    fprintf(ofp, "%2d: ", (line -
lines[0])/LINE_LEN);
    int m = 0;
    for (m = 0; m < LINE_LEN; m++) {
    if (line[m] != 0) {
        putc(line[m], ofp);
    } else {
        putc(' ', ofp);
    }
```

```
    }
}

void printDeltas(unsigned char *deltas) {
    int m, max;

    for (max = LINE_LEN-1; max >= 0; max--) {
    if (deltas[max] != 0)
        break;
    }

    for (m = 0; m <= max; m++) {
    fprintf(ofp, "%2X ", deltas[m]);
    }

    for (m = max+1; m < LINE_LEN; m++) {
    fprintf(ofp, "   ");
    }
}

void printSeparator(unsigned char *sep) {
    int m, max;

    for (max = LINE_LEN-1; max >= 0; max--) {
    if (sep[max] != 0)
        break;
    }
```

```
    for (m = 0; m <= max; m++) {
    fprintf(ofp, "%2X ", sep[m]);
    }

}

int endSection(unsigned char *sep) {
    // does this sep end with 00 26, a break in
the music?
    int m, max;

    for (max = LINE_LEN-1; max >= 0; max--) {
    if (sep[max] != 0)
        break;
    }

    if ((max >= 1) && (sep[max] == 0x26) &&
(sep[max-1] == 0))
    return 1;
    return 0;
}

int main(int argc, char **argv) {
    int inSong = 0;
    // int lineNo = 0;
    int inLine = 0;
```

```
    bzero(lines, NUM_LINES*LINE_LEN*
sizeof(unsigned char));
    bzero(separator, NUM_LINES*LINE_LEN*
sizeof(unsigned char));
    bzero(deltas, NUM_LINES*LINE_LEN*
sizeof(unsigned char));

    int lineNo = 0;
    int charNo = 0;

    if (argc == 1) {
    fp = stdin;
    ofp = stdout;
    } else if (argc == 2) {
    fp = fopen(argv[1], "r");
    ofp = stdout;
    } else {
    fp = fopen(argv[1], "r");
    ofp = fopen(argv[2], "w");
    }

    int n = 0;

    while ((n++ < 6000) && (half <= 2)) {

    switch (state) {
```

```
    case BEFORE_SONG:
        read2();
        if ((prev_ch == 0) && (curr_ch == '&'))
{
        state = IN_SONG_AND_LINE;
        }
        break;
    case IN_SONG_AND_LINE:
        read2();
        if (curr_ch == '^') {
        state = IN_SONG_BETWEEN;
        putc('\n', ofp);

        fprintf(ofp, "%d: ", lineNo);

        charNo = 0;
        separator[lineNo][charNo++] = prev_ch;
        separator[lineNo][charNo++] = curr_ch;



        } else {
        if (curr_ch != 0) {
            putc(curr_ch, ofp);
            lines[lineNo][charNo] = curr_ch;
            deltas[lineNo][charNo] = prev_ch;
            charNo++;
        }
```

```
        }
        break;
    case IN_SONG_BETWEEN:
        read1();
        if (curr_ch == '&') {
        state = IN_SONG_AND_LINE;

        separator[lineNo][charNo++] = curr_ch;

        charNo = 0;
        lineNo += 1;
        }  else if (curr_ch == 0xFF) {
        if (half == 1) {
            // the 2nd half
            fprintf(ofp, "\n\nStarting 2nd
half\n\n");
            // discard extra
            //getc(fp);
            //lineNo = -1;
            max_lines = lineNo;
            half = 2;
        } else {
            half = 3;
        }
        } else {
        separator[lineNo][charNo++] = curr_ch;
        }
```

```
        break;
    }
    }

    fprintf(ofp, "\n\nDumping lines\n\n");

    first_lines = lines[0];
    second_lines = lines[0] + (max_lines+1) *
LINE_LEN;

    first_deltas = deltas[0];
    second_deltas = deltas[0] + (max_lines+1) *
LINE_LEN;

    first_separators = separator[0];
    second_separators = separator[0] +
(max_lines+1) * LINE_LEN;

    for (n = 0; n < max_lines; n++) {
    fprintf(ofp, "%2d: ", n);
    /*
    if (lines[n][0] == 0) {
        break;
    }
    */
    //printLine(lines[n]);
    printLine(first_lines);
```

```
    first_lines += LINE_LEN;

    fprintf(ofp, "\n   ");

    //printDeltas(deltas[n]);
    printDeltas(first_deltas);
    first_deltas += LINE_LEN;

    //printSeparator(separator[n]);
    printSeparator(first_separators);
    first_separators += LINE_LEN;
    putc('\n', ofp);

    if (endSection(first_separators -
LINE_LEN)) {
        fprintf(ofp, "Break occurring in first
line\n");
        //continue;
    }

    fprintf(ofp, "%2d: ", n + max_lines + 1);
    //printLine(lines[n + max_lines + 1]);
    printLine(second_lines);
    second_lines += LINE_LEN;


    fprintf(ofp, "\n   ");
```

```
    //printDeltas(deltas[n + max_lines + 1]);
    printDeltas(second_deltas);
    second_deltas += LINE_LEN;

    //printSeparator(separator[n + max_lines +
1]);
    printSeparator(second_separators);
    second_separators += LINE_LEN;

    if (endSection(second_separators -
LINE_LEN)) {
        fprintf(ofp, "Break occurring in second
line\n");
        //continue;
    }

    /*
    for (m = 0; m < LINE_LEN/2; m++) {
        fprintf(ofp, "%2X ", separator[n][m]);
    }
    */
    putc('\n', ofp);
    }


    fclose(ofp);
```

```
    exit(0);
}
```

it prints out stuff like

```
0:  0: \Sit down and listen
    1  0  4  4  6  0  7  5  5  7  0  9  7  9  0
B  9  8  8  9                 8 5E  1  0 1D  1
1E 26
47: 47: 'cause I've got
    2  2  3  2  3  2  4  0  4  4  3  5  0  A  8
8 5E  2  0 10  2 4C 26
 1:  1: good news for you
    1  5  5  5  7  0  8  5  4  7  0  9  7  9  0
9  5                          6 5E  1  0 10  1
33 26
48: 48: It was in the
    2  6  8  0  9  7  9  0  7  6  0  7  4
4 5E  2  0 10  2 80 C8 26
 2:  2: papers today
    1  D  C  5  5  5  7  0 14 10 11 10
11 5E  1  0 10  1 80 A7 26
49: 49: Some physician
    2  5  5  5  7  0  B  9  8  D  C  6  7  6
6 5E  2  0 10  2 80 96 26
 3:  3: had made a discovery
    1  4  5  6  0  5  3  4  4  0 1B  0  E  6  5
8  9  8 11  8                 4 5E  1  0 10  1
3F 26
```

That's the data - not sure what information it is conveying.

Conclusion

The Malata uses a more complex encoding than the Sonken. This is only partially solved at present.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

JavaSound has no direct support for Karaoke. This chapter looks at how to combine the JavaSound libraries with other libraries such as Swing to give a Karaoke player for MIDI files.

Resources

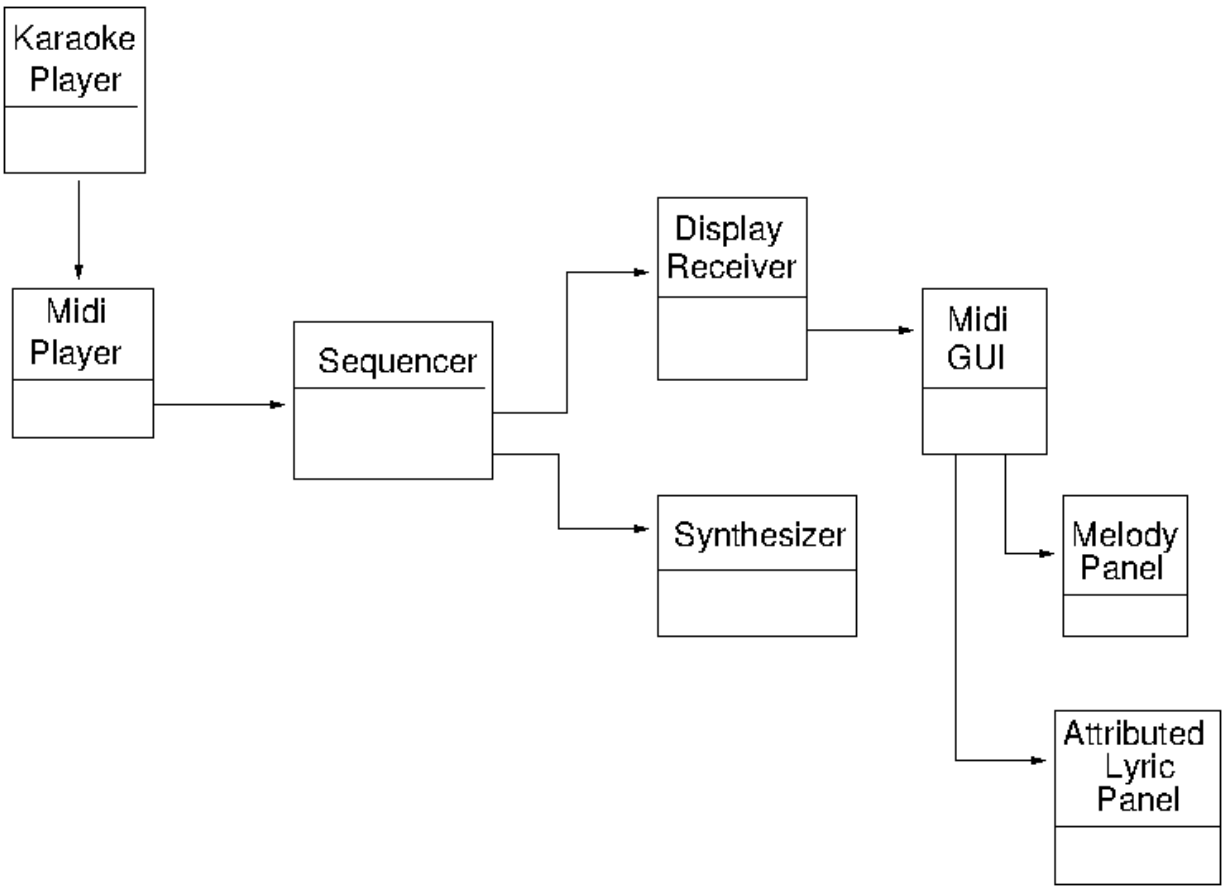
- Check out Karaoke Time downloaded as KaraokeTime.zip
- Chapter 8: Overview of the MIDI Package

Introduction

Java has no library support for Karaoke. That is too application specific. In this chapter we give code for a Karaoke player that can play KAR files. The player will show two lines of the lyrics to be played, with words already played beigin highlighted in red. Along the top it shows using a simple piano keyboard the notes that are played in channel one of the MIDI file. In the middle it shows the melody line, with a vertical line in the middle to show the currently playing note.

The player looks like

The UML diagram is basically


Karaoke Player
Midi Player
Sequencer
Display Receiver
Synthesizer
Midi GUI
Melody Panel
Attributed Lyric Panel

KaraokePlayer

The KaraokePlayer class extracts the filename of the Karaoke file and creates a MidiPlayer to handle the file:

```java
/*
 * KaraokePlayer.java
 *
 */

import javax.swing.*;

public class KaraokePlayer {


    public static void main(String[] args)
throws Exception {
    if (args.length != 1) {
        System.err.println("KaraokePlayer:
usage: " +
                 "KaraokePlayer <midifile>");
        System.exit(1);
    }
    String    strFilename = args[0];

    MidiPlayer midiPlayer = new MidiPlayer();
    midiPlayer.playMidiFile(strFilename);
    }
}
```

MidiPlayer

The MidiPlayer creates a Sequence from the file. Sequence information is required at many places, and so rather than pass the sequence in parameters, it is stored in a singleton (static) object, a SequenceInformation. This makes the sequence effectively a global object to the system.

The player then gets the default sequencer and transmits MIDI events to two Receiver objects: the default synthesizer to play the events and a DisplayReceiver to manage all the GUI handling. The Sequencer method getTransmitter() is mis-named: each time it is called it returns a *new* transmitter, each playing the same events to their respective receivers. From Java SE Documentation: Chapter 10: Transmitting and Receiving MIDI Messages

> This code [in their example] introduces a dual invocation of the MidiDevice.getTransmitter method, assigning the results to inPortTrans1 and inPortTrans2. As mentioned earlier, a device can own multiple transmitters and receivers. Each time MidiDevice.getTransmitter() is invoked for a given device, another transmitter is returned, until no more are available, at which time an exception will be thrown.

That way, the sequencer can send to two receivers.

Receivers do not get MetaMessages. These contain information such as Text or Lyric events. The DisplayReceiver is registered as an MetaEventListener so that it can manage these events as well as other events.

The MidiPlayer is:

```
import javax.sound.midi.MidiSystem;
import
javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.Sequence;
import javax.sound.midi.Receiver;
import javax.sound.midi.Sequencer;
import javax.sound.midi.Transmitter;
import javax.sound.midi.MidiChannel;
import javax.sound.midi.MidiDevice;
import javax.sound.midi.Synthesizer;
import javax.sound.midi.ShortMessage;
import javax.sound.midi.SysexMessage;

import java.io.File;
import java.io.IOException;

public class MidiPlayer {

    private DisplayReceiver receiver;

    public  void playMidiFile(String
strFilename) throws Exception {
    File    midiFile = new File(strFilename);

    /*
     *    We try to get a Sequence object,
```

```
loaded with the content
     *    of the MIDI file.
     */
    Sequence    sequence = null;
    try {
        sequence =
MidiSystem.getSequence(midiFile);
    }
    catch (InvalidMidiDataException e) {
        e.printStackTrace();
        System.exit(1);
    }
    catch (IOException e) {
        e.printStackTrace();
        System.exit(1);
    }

    if (sequence == null) {
        out("Cannot retrieve Sequence.");
    } else {

SequenceInformation.setSequence(sequence);
        playMidi(sequence);
    }
    }

    public  void playMidi(Sequence sequence)
```

```
throws Exception {

        Sequencer sequencer =
MidiSystem.getSequencer(true);
        sequencer.open();
        sequencer.setSequence(sequence);

    receiver = new DisplayReceiver(sequencer);

sequencer.getTransmitter().setReceiver(receiver);

    sequencer.addMetaEventListener(receiver);

    if (sequencer instanceof Synthesizer) {
        Debug.println("Sequencer is also a
synthesizer");
    } else {
        Debug.println("Sequencer is not a
synthesizer");
    }
        //sequencer.start();

    /*
    Synthesizer synthesizer =
MidiSystem.getSynthesizer();
    synthesizer.open();
```

```
    if (synthesizer.getDefaultSoundbank() ==
null) {
        // then you know that java sound is
using the hardware soundbank
        Debug.println("Synthesizer using h/w
soundbank");
    } else Debug.println("Synthesizer using s/w
soundbank");


    Receiver synthReceiver =
synthesizer.getReceiver();
    Transmitter seqTransmitter =
sequencer.getTransmitter();
    seqTransmitter.setReceiver(synthReceiver);
    MidiChannel[] channels =
synthesizer.getChannels();
    Debug.println("Num channels is " +
channels.length);
    */
        sequencer.start();

    /* default synth doesn't support pitch
bending
    Synthesizer synthesizer =
MidiSystem.getSynthesizer();
    MidiChannel[] channels =
```

```
synthesizer.getChannels();
    for (int i = 0; i < channels.length; i++) {
        System.out.printf("Channel %d has bend
%d\n", i, channels[i].getPitchBend());
        channels[i].setPitchBend(16000);
        System.out.printf("Channel %d now has
bend %d\n", i, channels[i].getPitchBend());
    }
    */

    /* set volume - doesn't work */
    /*
    for (int i = 0; i < channels.length; i++) {
        channels[i].controlChange(7, 0);
    }
    */
    /*
    System.out.println("Turning notes off");
    for (int i = 0; i < channels.length; i++) {
        channels[i].allNotesOff();
        channels[i].allSoundOff();
    }
    */

    /* set volume - doesn't work either */
    /*
    try {
```

```
        Thread.sleep(5000);
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    if (synthReceiver ==
MidiSystem.getReceiver())
        System.out.println("Reciver is
default");
    else
        System.out.println("Reciver is not
default");
    System.out.println("Receiver is " +
synthReceiver.toString());
    //synthReceiver = MidiSystem.getReceiver();
    System.out.println("Receiver is now " +
synthReceiver.toString());
    ShortMessage volMessage = new
ShortMessage();
    int midiVolume = 1;
    for (Receiver rec:
synthesizer.getReceivers()) {
        System.out.println("Setting vol on
recveiver " + rec.toString());
    for (int i = 0; i < channels.length; i++) {
        try {
```

```
        //
volMessage.setMessage(ShortMessage.CONTROL_CHANGE
 i, 123, midiVolume);

volMessage.setMessage(ShortMessage.CONTROL_CHANGE
 i, 7, midiVolume);
        } catch (InvalidMidiDataException e) {
        e.printStackTrace();
}
        synthReceiver.send(volMessage, -1);
        rec.send(volMessage, -1);
    }
    }
    System.out.println("Changed midi volume");
    */
    /* master volume control using sysex */
    /*
http://www.blitter.com/~russtopia/MIDI/~jglatt/te
 */
    /*
    SysexMessage sysexMessage = new
SysexMessage();
    /* volume values from
http://www.bandtrax.com.au/sysex.htm */
    /* default volume 0x7F * 128 + 0x7F from */
    /*
    byte[] data = {(byte) 0xF0, (byte) 0x7F,
```

```
(byte) 0x7F, (byte) 0x04,
               (byte) 0x01, (byte) 0x0, (byte)
0x7F, (byte) 0xF7};
    sysexMessage.setMessage(data, data.length);
    synthReceiver.send(sysexMessage, -1);
    for (Receiver rec:
synthesizer.getReceivers()) {
        System.out.println("Setting vol on
recveiver " + rec.toString());
        rec.send(sysexMessage, -1);
    }
    */
     }


    public DisplayReceiver getReceiver() {
    return receiver;
    }

    private static void out(String strMessage)
    {
    System.out.println(strMessage);
    }
}
```

DisplayReceiver

The DisplayReceiver collects both ShortMessages as a Receiver and MetaMessages as a MetaEventListener. These are needed to see both the notes and the lyrics.

The DisplayReceiver decodes the notes and text sent to it. In turn, it passes these to a MidiGUI to show them. This class is

```
/**
 * DisplayReceiver
 *
 * Acts as a Midi receiver to the default Java
Midi sequencer.
 * It collects Midi events and Midi meta
messages from the sequencer.
 * these are handed to a UI object for display.
 *
 * The current UI object is a MidiGUI but could
be replaced.
 */

import javax.sound.midi.*;
import javax.swing.SwingUtilities;

public class DisplayReceiver implements
Receiver,
                    MetaEventListener {
    private MidiGUI gui;
    private Sequencer sequencer;
    private int melodyChannel =
SequenceInformation.getMelodyChannel();

    public DisplayReceiver(Sequencer sequencer)
{
```

```
    this.sequencer = sequencer;
    gui = new MidiGUI(sequencer);
    }

    public void close() {
    }

    /**
     * Called by a Transmitter to receive
events
     * as a Receiver
     */
    public void send(MidiMessage msg, long
timeStamp) {
    // Note on/off messages come from the midi
player
    // but not meta messages

    if (msg instanceof ShortMessage) {
        ShortMessage smsg = (ShortMessage) msg;



        String strMessage = "Channel " +
smsg.getChannel() + " ";

        switch (smsg.getCommand())
        {
```

```
        case Constants.MIDI_NOTE_OFF:
            strMessage += "note Off " +
            getKeyName(smsg.getData1()) + " " +
timeStamp;
            break;

        case Constants.MIDI_NOTE_ON:
            strMessage += "note On " +
            getKeyName(smsg.getData1()) + " " +
timeStamp;
            break;
        }
        Debug.println(strMessage);
        if (smsg.getChannel() == melodyChannel)
{
        gui.setNote(timeStamp,
smsg.getCommand(), smsg.getData1());
        }

    }
    }


    public void meta(MetaMessage msg) {
    Debug.println("Reciever got a meta
message");
    if (((MetaMessage) msg).getType() ==
```

```
Constants.MIDI_TEXT_TYPE) {
        setLyric((MetaMessage) msg);
    } else if (((MetaMessage) msg).getType() ==
Constants.MIDI_END_OF_TRACK)  {
        System.exit(0);
    }
    }

    public void setLyric(MetaMessage message) {
    byte[] data = message.getData();
    String str = new String(data);
    Debug.println("Lyric +\"" + str + "\" at "
+ sequencer.getTickPosition());
    gui.setLyric(str);

    }

    private static String[] keyNames = {"C",
"C#", "D", "D#", "E", "F", "F#", "G", "G#",
"A", "A#", "B"};

    public static String getKeyName(int
keyNumber) {
        if (keyNumber > 127) {
        return "illegal value";
    } else {
        int note = keyNumber % 12;
```

```
        int octave = keyNumber / 12;
        return keyNames[note] + (octave - 1);
    }
    }

}
```

MidiGUI

The MidiGUI is called by two methods: setLyric() and setNote(). The GUI consists of three main areas: an area to give a "piano" view of the melody as it is played (pianoPanel), an area to show the complete set of melody notes (melodyPanel) and a set of Panels to show the lyrics. setNote() is fairly straightforward in that it just calls drawNote() in the pianoPanel. setLyric() is considerably more complex.

Most Karaoke players show a couple of lines of text for the lyrics. As lyrics are played, typically the text will change colour to match. When the end of a line is reached, focus will switch to the next line, and the previous line will be replaced with another line of lyrics.

Each line must hold a line of lyrics. The line must be able to react to lyrics as they are played. This is handled by an AttributedTextPanel, shown later. The main task is to feed changes in lyrics through to the selected panel so that it can display them in the correct colours.

The other principal task for the MidiGUI here is to switch focus between AttributedTextPanel's when end of line is detected and to update the next line of text. The new lines of text can't come from the lyrics as they are played, but must instead be constructed from the sequence containing all of the notes and lyrics. The convenience class SequenceInformation (shown later) takes a Sequence object and has a method to extract an array of LyricLine objects. Each panel displaying a line is given a line from this array.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.sound.midi.*;
import java.util.Vector;
import java.util.Map;
import java.io.*;


public class MidiGUI extends JFrame {
    //private GridLayout mgr = new
GridLayout(3,1);
    private BorderLayout mgr = new
BorderLayout();

    private PianoPanel pianoPanel;
    private MelodyPanel melodyPanel;

    private AttributedLyricPanel lyric1;
    private AttributedLyricPanel lyric2;
    private AttributedLyricPanel[]
lyricLinePanels;
    private int whichLyricPanel = 0;

    private JPanel lyricsPanel = new JPanel();
```

```
    private Sequencer sequencer;
    private Sequence sequence;
    private Vector<LyricLine> lyricLines;

    private int lyricLine = -1;

    private boolean inLyricHeader = true;
    private Vector<DurationNote> melodyNotes;

    private Map<Character, String> pinyinMap;

    private int language;

    public MidiGUI(final Sequencer sequencer) {
    this.sequencer = sequencer;
    sequence = sequencer.getSequence();

    // get lyrics and notes from Sequence Info
    lyricLines =
SequenceInformation.getLyrics();
    melodyNotes =
SequenceInformation.getMelodyNotes();
    language =
SequenceInformation.getLanguage();

    pianoPanel = new PianoPanel(sequencer);
    melodyPanel = new MelodyPanel(sequencer);
```

```
    pinyinMap =
CharsetEncoding.loadPinyinMap();
    lyric1 = new
AttributedLyricPanel(pinyinMap);
    lyric2 = new
AttributedLyricPanel(pinyinMap);
    lyricLinePanels = new
AttributedLyricPanel[] {
        lyric1, lyric2};

    Debug.println("Lyrics ");

    for (LyricLine line: lyricLines) {
        Debug.println(line.line + " " +
line.startTick + " " + line.endTick +
              " num notes " +
line.notes.size());
    }

    getContentPane().setLayout(mgr);
    /*
    getContentPane().add(pianoPanel);
    getContentPane().add(melodyPanel);

    getContentPane().add(lyricsPanel);
    */
```

```
    getContentPane().add(pianoPanel,
BorderLayout.PAGE_START);
    getContentPane().add(melodyPanel,
BorderLayout.CENTER);

    getContentPane().add(lyricsPanel,
BorderLayout.PAGE_END);



    lyricsPanel.setLayout(new GridLayout(2,
1));
    lyricsPanel.add(lyric1);
    lyricsPanel.add(lyric2);
    setLanguage(language);

    setText(lyricLinePanels[whichLyricPanel],
lyricLines.elementAt(0).line);

    Debug.println("First lyric line: " +
lyricLines.elementAt(0).line);
    if (lyricLine < lyricLines.size() - 1) {

setText(lyricLinePanels[(whichLyricPanel+1) %
2], lyricLines.elementAt(1).line);
        Debug.println("Second lyric line: " +
lyricLines.elementAt(1).line);
    }
```

```
    // handle window closing

setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLO

    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent
e) {
            sequencer.stop();
            System.exit(0);
                }
            });

    // handle resize events
    addComponentListener(new ComponentAdapter()
{
        public void
componentResized(ComponentEvent e) {
            Debug.printf("Component has resized
to width %d, height %d\n",
                    getWidth(), getHeight());
            // force resize of children -
especially the middle MelodyPanel
            e.getComponent().validate();
        }
        public void
componentShown(ComponentEvent e) {
```

```
            Debug.printf("Component is visible
with width %d, height %d\n",
                        getWidth(), getHeight());
        }
        });

    setSize(1600, 900);
    setVisible(true);
    }

    public void setLanguage(int lang) {
    lyric1.setLanguage(lang);
    lyric2.setLanguage(lang);
    }



    /**
     * A lyric starts with a header section
     * We have to skip over that, but can pick
useful
     * data out of it
     */

    /**
     * header format is
     *   \@Llanguage code
     *   \@Ttitle
```

```
     *   \@Tsinger
     */

    public void setLyric(String txt) {
    Debug.println("Setting lyric to " + txt);
    if (inLyricHeader) {
        if (txt.startsWith("@")) {
        Debug.println("Header: " + txt);
        return;
        } else {
        inLyricHeader = false;
        }
    }

    if ((lyricLine == -1) && (txt.charAt(0) ==
'\\')) {
        lyricLine = 0;

colourLyric(lyricLinePanels[whichLyricPanel],
txt.substring(1));
        //
lyricLinePanels[whichLyricPanel].colourLyric(txt.

        return;
    }

    if (txt.equals("\r\n") || (txt.charAt(0) ==
```

```
'/') || (txt.charAt(0) == '\\')) {
        if (lyricLine < lyricLines.size() -1)
        Debug.println("Setting next lyric line
to \"" +

lyricLines.elementAt(lyricLine + 1).line +
"\"");

        final int thisPanel = whichLyricPanel;
        whichLyricPanel = (whichLyricPanel + 1)
% 2;

        Debug.println("Setting new lyric line
at tick " +
              sequencer.getTickPosition());

        lyricLine++;

        // if it's a \ r /, the rest of the txt
should be the next  word to
        // be coloured

        if ((txt.charAt(0) == '/') ||
(txt.charAt(0) == '\\')) {
        Debug.println("Colouring newline of " +
txt);
```

```
colourLyric(lyricLinePanels[whichLyricPanel],
txt.substring(1));
        }

        // Update the current line of text to
show the one after next
        // But delay the update until 0.25
seconds after the next line
        // starts playing, to preserve visual
continuity
        if (lyricLine + 1 < lyricLines.size())
{
        /*
          long startNextLineTick =
lyricLines.elementAt(lyricLine).startTick;
          long delayForTicks =
startNextLineTick -
sequencer.getTickPosition();
          Debug.println("Next  current "  +
startNextLineTick + " " +
sequencer.getTickPosition());
          float microSecsPerQNote =
sequencer.getTempoInMPQ();
          float delayInMicroSecs =
microSecsPerQNote * delayForTicks / 24 +
250000L;
        */
```

```
        final Vector<DurationNote> notes =
lyricLines.elementAt(lyricLine).notes;

        final int nextLineForPanel = lyricLine
+ 1;

        if (lyricLines.size() >=
nextLineForPanel) {
            Timer timer = new Timer((int) 1000,
                        new ActionListener() {
                        public void
actionPerformed(ActionEvent e) {
                            if
(nextLineForPanel >= lyricLines.size()) {
                            return;
                            }

setText(lyricLinePanels[thisPanel],
lyricLines.elementAt(nextLineForPanel).line);

//lyricLinePanels[thisPanel].setText(lyricLines.e



                        }
                        });
            timer.setRepeats(false);
```

```
            timer.start();
        } else {
            // no more lines
        }
        }
    } else {
        Debug.println("Playing lyric " + txt);

colourLyric(lyricLinePanels[whichLyricPanel],
txt);

//lyricLinePanels[whichLyricPanel].colourLyric(tx

    }
    }

    /**
     * colour the lyric of a panel.
     * called by one thread, makes changes in
GUI thread
     */
    private void colourLyric(final
AttributedLyricPanel p, final String txt) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            Debug.print("Colouring lyric \"" +
txt + "\"");
```

```
            if (p == lyric1) Debug.println(" on
panel 1");
            else Debug.println(" on panel 2");
            p.colourLyric(txt);
        }
        }
        );
    }

    /**
     * set the lyric of a panel.
     * called by one thread, makes changes in
GUI thread
     */
    private void setText(final
AttributedLyricPanel p, final String txt) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            Debug.println("Setting text \"" +
txt + "\"");
            if (p == lyric1) Debug.println(" on
panel 1");
            else Debug.println(" on panel 2");
            p.setText(txt);
        }
        }
        );
```

```
    }

    public void setNote(long timeStamp, int
onOff, int note) {
    Debug.printf("Setting note in gui to %d\n",
note);

    if (onOff == Constants.MIDI_NOTE_OFF) {
        pianoPanel.drawNoteOff(note);
    } else if (onOff == Constants.MIDI_NOTE_ON)
{
        pianoPanel.drawNoteOn(note);
    }
    }
}
```

AttributedLyricPanel

The panel to display a line of lyrics must be able to show text in two colours: the lyrics already played and the lyrics yet to be played. The Java AttributedText class is useful for this, as the text can be marked with different attributes such as colours. This is wrapped in an AttributedTextPanel, shown later.

One minor wrinkle concerns language. Chinese has both the character form and a Romanised form called PinYin. Chinese speakers can read the character form. People like me can only understand the PinYin form. So if the language is Chinese then the AttributedTextPanel shows the PinYin alongside the Chinese characters. The language identity should be passed to the AttributedLyricPanel as well.

The AttributedLyricPanel is

```java
import javax.swing.*;
import java.awt.*;
import java.awt.font.*;
import java.text.*;
import java.util.Map;


public class AttributedLyricPanel extends
JPanel {

    private final int PINYIN_Y = 40;
    private final int TEXT_Y = 90;

    private String text;
    private AttributedString attrText;
    private int coloured = 0;
    private Font font = new
Font(Constants.CHINESE_FONT, Font.PLAIN, 36);
    private Font smallFont = new
Font(Constants.CHINESE_FONT, Font.PLAIN, 24);
    private Color red = Color.RED;
    private int language;
    private String pinyinTxt = null;

    private Map<Character, String> pinyinMap =
null;
```

```
    public AttributedLyricPanel(Map<Character,
String> pinyinMap) {
    this.pinyinMap = pinyinMap;
    }

    public Dimension getPreferredSize() {
    return new Dimension(1000, TEXT_Y + 20);
    }

    public void setLanguage(int lang) {
    language = lang;
    Debug.printf("Lang in panel is %X\n",
lang);
    }

    public boolean isChinese() {
    switch (language) {
    case SongInformation.CHINESE1:
    case SongInformation.CHINESE2:
    case SongInformation.CHINESE8:
    case SongInformation.CHINESE131:
    case SongInformation.TAIWANESE3:
    case SongInformation.TAIWANESE7:
    case SongInformation.CANTONESE:
        return true;
    }
```

```
    return false;
    }

    public void setText(String txt) {
    coloured = 0;
    text = txt;
    Debug.println("set text " + text);
    attrText = new AttributedString(text);
    if (text.length() == 0) {
        return;
    }
    attrText.addAttribute(TextAttribute.FONT,
font, 0, text.length());

    if (isChinese()) {
        pinyinTxt = "";
        for (int n = 0; n < txt.length(); n++)
{
        char ch = txt.charAt(n);
        String pinyin = pinyinMap.get(ch);
        if (pinyin != null) {
            pinyinTxt += pinyin + " ";
        } else {
            Debug.printf("No pinyin map for
character \"%c\"\n", ch);
        }
        }
```

```
    }

    repaint();
    }

    public void colourLyric(String txt) {
    coloured += txt.length();
    if (coloured != 0) {
        repaint();
    }
    }

    /**
     * Draw the string with the first part in
red, rest in green.
     * String is centred
     */

    @Override
    public void paintComponent(Graphics g) {
    if ((text.length() == 0) || (coloured >
text.length())) {
        return;
    }
    g.setFont(font);
    FontMetrics metrics = g.getFontMetrics();
```

```
    int strWidth = metrics.stringWidth(text);
    int panelWidth = getWidth();
    int offset = (panelWidth - strWidth) / 2;

    if (coloured != 0) {
        try {

attrText.addAttribute(TextAttribute.FOREGROUND,
red, 0, coloured);
        } catch(Exception e) {
        System.out.println(attrText.toString()
+ " " + e.toString());
        }
    }
    g.clearRect(0, 0, getWidth(), getHeight());
    try {
        g.drawString(attrText.getIterator(),
offset, TEXT_Y);
    } catch (Exception e) {
        System.err.println("Attr Str exception
on " + text);
    }
    // Draw the Pinyin if it's not zero
    if (pinyinTxt != null && pinyinTxt.length()
!= 0) {
        g.setFont(smallFont);
        metrics = g.getFontMetrics();
```

```
        strWidth =
metrics.stringWidth(pinyinTxt);
        offset = (panelWidth - strWidth) / 2;

        g.drawString(pinyinTxt, offset,
PINYIN_Y);
        g.setFont(font);
    }
    }
}
```

PianoPanel

The pianoPanel shows a piano-like keyboard. As a note is turned on, it colours the note in blue and returns to normal any previously playing note. When a note is turned off, the note reverts to its normal colour (black or white).

Colouring notes is called by setNote as nonte on/note off messsages come from the sequencer.

The PianoPanel is

```java
import java.util.Vector;
import javax.swing.*;
import java.awt.*;
import javax.sound.midi.*;


public class PianoPanel extends JPanel {

    private final int HEIGHT = 100;
    private final int HEIGHT_OFFSET = 10;

    long timeStamp;
    private Vector<DurationNote> notes;
    private Vector<DurationNote> sungNotes;
    private int lastNoteDrawn = -1;
    private Sequencer sequencer;
    private Sequence sequence;
    private int maxNote;
    private int minNote;

    private Vector<DurationNote>
unresolvedNotes = new Vector<DurationNote> ();

    private int playingNote = -1;

    public PianoPanel(Sequencer sequencer) {
```

```
    maxNote =
SequenceInformation.getMaxMelodyNote();
    minNote =
SequenceInformation.getMinMelodyNote();
    Debug.println("Max: " + maxNote + " Min " +
minNote);
    }

    public Dimension getPreferredSize() {
    return new Dimension(1000, 120);
    }

    public void drawNoteOff(int note) {
    if (note < minNote || note > maxNote) {
        return;
    }

    Debug.println("Note off played is " +
note);
    if (note != playingNote) {
        // Sometimes "note off" followed
immediately by "note on"
        // gets mixed up to "note on" followed
by "note off".
        // Ignore the "note off" since the next
note has already
```

```
        // been processed
        Debug.println("Ignoring note off");
        return;
    }
    playingNote = -1;
    repaint();
    }

    public void drawNoteOn(int note) {
    if (note < minNote || note > maxNote) {
        return;
    }

    Debug.println("Note on played is " + note);
    playingNote = note;
    repaint();

    }



    private void drawPiano(Graphics g, int
width, int height) {
    int noteWidth = width /
(Constants.MIDI_NOTE_C8 -
Constants.MIDI_NOTE_A0);
    for (int noteNum =  Constants.MIDI_NOTE_A0;
```

```
// A0
         noteNum <=  Constants.MIDI_NOTE_C8; //
C8
         noteNum++) {

        drawNote(g, noteNum, noteWidth);
    }
    }

    private void drawNote(Graphics g, int
noteNum, int width) {
    if (isWhite(noteNum)) {
        noteNum -= Constants.MIDI_NOTE_A0;
        g.setColor(Color.WHITE);
        g.fillRect(noteNum*width,
HEIGHT_OFFSET, width, HEIGHT);
        g.setColor(Color.BLACK);
        g.drawRect(noteNum*width,
HEIGHT_OFFSET, width, HEIGHT);
    } else {
        noteNum -= Constants.MIDI_NOTE_A0;
        g.setColor(Color.BLACK);
        g.fillRect(noteNum*width,
HEIGHT_OFFSET, width, HEIGHT);
    }
    if (playingNote != -1) {
        g.setColor(Color.BLUE);
```

```
        g.fillRect((playingNote -
Constants.MIDI_NOTE_A0) * width, HEIGHT_OFFSET,
width, HEIGHT);
    }
    }

    private boolean isWhite(int noteNum) {
    noteNum = noteNum % 12;
    switch (noteNum) {
    case 1:
    case 3:
    case 6:
    case 8:
    case 10:
    case 13:
        return false;
    default:
        return true;
    }
    }

    @Override
    public void paintComponent(Graphics g) {

    int ht = getHeight();
    int width = getWidth();
```

```
    drawPiano(g, width, ht);


    }
}
```

MelodyPanel

The MelodyPanel is a scrolling panel showing all the notes of the melody. The currently playing note is centred in the display. This is done by drawing all of the notes into a BufferedImage and then copying across the relevant part every 50 milliseconds.

The MelodyPanel is

```java
import java.util.Vector;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.sound.midi.*;
import java.awt.image.BufferedImage;
import java.io.*;
import javax.imageio.*;

public class MelodyPanel extends JPanel {

    private static int DBL_BUF_SCALE = 2;
    private static final int NOTE_HEIGHT = 10;
    private static final int SLEEP_MSECS = 5;

    private long timeStamp;
    private Vector<DurationNote> notes;
    private Sequencer sequencer;
    private Sequence sequence;
    private int maxNote;
    private int minNote;
    private long tickLength = -1;
    private long currentTick = -1;
    private Image image = null;

    /**
```

```
     * The panel where the melody notes are
shown in a
     * scrolling panel
     */
    public MelodyPanel(Sequencer sequencer) {

    maxNote =
SequenceInformation.getMaxMelodyNote();
    minNote =
SequenceInformation.getMinMelodyNote();
    Debug.println("Max: " + maxNote + " Min " +
minNote);
    notes =
SequenceInformation.getMelodyNotes();
    this.sequencer = sequencer;
    tickLength = sequencer.getTickLength() +
1000; // hack to make white space at end, plus
fix bug

    //new TickPointer().start();
    // handle resize events
    addComponentListener(new ComponentAdapter()
{
        public void
componentResized(ComponentEvent e) {
            Debug.printf("Component melody
panel has resized to width %d, height %d\n",
```

```
                        getWidth(), getHeight());
        }
        public void
componentShown(ComponentEvent e) {
            Debug.printf("Component malody
panel is visible with width %d, height %d\n",
                        getWidth(), getHeight());
        }
        });

    }

    /**
     * Redraw the melody image after each tick
     * to give a scrolling effect
     */
    private class TickPointer extends Thread {
    public void run() {
        while (true) {
        currentTick =
sequencer.getTickPosition();
        MelodyPanel.this.repaint();
        /*
        SwingUtilities.invokeLater(
                        new Runnable() {
                        public void run() {
```

```
synchronized(MelodyPanel.this) {

MelodyPanel.this.repaint();
                            }
                        }
                        });
        */
        try {
            sleep(SLEEP_MSECS);
        } catch (Exception e) {
            // keep going
            e.printStackTrace();
        }
        }
    }

    }

    /**
     * Draw the melody into a buffer so we can
just copy bits to the screen
     */
    private void drawMelody(Graphics g, int
front, int width, int height) {
    try {
    g.setColor(Color.WHITE);
    g.fillRect(0, 0, width, height);
```

```
    g.setColor(Color.BLACK);

    String title =
SequenceInformation.getTitle();
    if (title != null) {
        //Font f = new Font("SanSerif",
Font.ITALIC, 40);
        Font f = new
Font(Constants.CHINESE_FONT, Font.ITALIC, 40);
        g.setFont(f);
        int strWidth =
g.getFontMetrics().stringWidth(title);
        g.drawString(title, (front -
strWidth/2), height/2);
        Debug.println("Drawn title " + title);
    }

    for (DurationNote note: notes) {
        long startNote = note.startTick;
        long endNote = note.endTick;
        int value = note.note;

        int ht = (value - minNote) * (height -
NOTE_HEIGHT) / (maxNote - minNote) +
NOTE_HEIGHT/2;
        // it's upside down
        ht = height - ht;
```

```
        long start = front + (int) (startNote *
DBL_BUF_SCALE);
        long end = front + (int) (endNote *
DBL_BUF_SCALE);

        drawNote(g, ht, start, end);
        //g.drawString(title, (int)start,
(int)height/2);
    }
    } catch(Exception e) {
        System.err.println("Drawing melody
error " + e.toString());
    }
    }

    /**
     * Draw a horizontal bar to represent a
nore
     */
    private void drawNote(Graphics g, int
height, long start, long end) {
    Debug.printf("Drawing melody at start %d
end %d height %d\n", start, end,  height -
NOTE_HEIGHT/2);

    g.fillRect((int) start, height -
```

```
NOTE_HEIGHT/2, (int) (end-start), NOTE_HEIGHT);
    }

    /**
     * Draw a vertical line in the middle of
the screen to
     * represent where we are in the playing
notes
     */
    private void paintTick(Graphics g, long
width, long height) {
    long x = (currentTick * width) /
tickLength;
    g.drawLine((int) width/2, 0, (int) width/2,
(int) height);
    //System.err.println("Painted tcik");
    }

    // leave space at the front of the image to
draw title, etc
    int front = 1000;

    /**
     * First time, draw the melody notes into
an off-screen buffer
     * After that, copy a segment of the buffer
into the image,
```

```
     * with the centre of the image the current
note
     */
    @Override
    public void paintComponent(Graphics g) {
    int ht = getHeight();
    int width = getWidth();
    //int front = width / 2;

    synchronized(this) {
    if (image == null) {
        /*
         * We want to stretch out the notes so
that they appear nice and wide on the screen.
         * A DBL_BUF_SCALE of 2 does this okay.
But then tickLength * DBL_BUF_SCALE may end
         * up larger than an int, and we can't
make a BufferedImage wider than MAXINT.
         * So we may have to adjust
DBL_BUF_SCALE.
         *
         * Yes, I know we ask Java to rescale
images on the fly, but that costs in runtime.
         */

        Debug.println("tick*DBLBUFSCALE " +
tickLength * DBL_BUF_SCALE);
```

```
        if ((long) (tickLength * DBL_BUF_SCALE)
> (long) Short.MAX_VALUE) {
        // DBL_BUF_SCALE = ((float)
Integer.MAX_VALUE) / ((float) tickLength);
        DBL_BUF_SCALE = 1;
        Debug.println("Adjusted DBL_BUF_SCALE
to "+ DBL_BUF_SCALE);
        }

        Debug.println("DBL_BUF_SCALE is "+
DBL_BUF_SCALE);

        // draw melody into a buffered image
        Debug.printf("New buffered img width %d
ht %d\n", tickLength, ht);
        image = new BufferedImage(front + (int)
(tickLength * DBL_BUF_SCALE), ht,
BufferedImage.TYPE_INT_RGB);
        Graphics ig = image.getGraphics();
        drawMelody(ig, front, (int) (tickLength
* DBL_BUF_SCALE), ht);
        new TickPointer().start();


        try {
        File outputfile = new
```

```
File("saved.png");
        ImageIO.write((BufferedImage) image,
"png", outputfile);
        } catch (Exception e) {
        System.err.println("Error in image
write " + e.toString());
        }

    }
    //System.err.printf("Drawing img from %d ht
%d width %d\n",
    //          front + (int) (currentTick *
DBL_BUF_SCALE - width/2), ht, width);

    boolean b = g.drawImage(image, 0, 0, width,
ht,
                front + (int) (currentTick *
DBL_BUF_SCALE - width/2), 0,
                front + (int) (currentTick *
DBL_BUF_SCALE + width/2), ht,
            null);
    /*System.out.printf("Ht of BI %d, width
%d\n", ((BufferedImage)image).getHeight(),
              ((BufferedImage)
image).getWidth());
    */
```

```
    //if (b) System.err.println("Drawn ok");
else System.err.println("NOt drawn ok");
    paintTick(g, width, ht);
    }
    }

}
```

SequenceInformation

SequenceInformation is given by

```
import javax.sound.midi.*;
import java.util.Vector;

public class SequenceInformation {

    private static Sequence sequence = null;
    private static Vector<LyricLine> lyricLines
= null;
    private static Vector<DurationNote>
melodyNotes = null;
    private static int lang = -1;
    private static String title = null;
    private static String performer = null;
    private static int maxNote;
    private static int minNote;

    private static int melodyChannel = -1;// no
such value // default for Songken

    public static void setSequence(Sequence
seq) {
    if (sequence != seq) {
        sequence = seq;
        lyricLines = null;
        melodyNotes = null;
        lang = -1;
```

```
        title = null;
        performer = null;
        maxNote = -1;
        minNote = Integer.MAX_VALUE;
    }
    }

    public static long getTickLength() {
    return sequence.getTickLength();
    }

    public static int getMelodyChannel() {
    boolean firstNoteSeen[] = {false, false,
false, false, false, false, false, false,
                    false, false, false, false,
false, false, false, false};
    boolean possibleChannel[] = {false, false,
false, false, false, false, false, false,
                    false, false, false, false,
false, false, false, false};
    if (melodyChannel != -1) {
        return melodyChannel;
    }

    if (lyricLines == null) {
        lyricLines = getLyrics();
    }
```

```
    long startLyricTick = ((LyricLine)
lyricLines.get(0)).startTick;
    Debug.printf("Lyrics start at %d\n",
startLyricTick);

    Track[]    tracks = sequence.getTracks();
    for (int nTrack = 0; nTrack <
tracks.length; nTrack++) {
        // out("Track " + nTrack + ":");
        // out("-----------------------");
        Track track = tracks[nTrack];
        for (int nEvent = 0; nEvent <
track.size(); nEvent++) {
        MidiEvent evt = track.get(nEvent);
        MidiMessage msg = evt.getMessage();
        if (msg instanceof ShortMessage) {
            ShortMessage smsg= (ShortMessage)
msg;
            int channel = smsg.getChannel();
            if (firstNoteSeen[channel]) {
            continue;
            }
            if (smsg.getCommand() ==
Constants.MIDI_NOTE_ON) {
            long tick = evt.getTick();
            Debug.printf("First note on for
```

```
channel %d at tick %d\n",
                        channel, tick);
            firstNoteSeen[channel] = true;
            if (Math.abs(startLyricTick - tick)
< 10) {
                // close enough - we hope!
                melodyChannel = channel;
                possibleChannel[channel] =
true;
                Debug.printf("Possible melody
channel is %d\n", channel);
            }
            if (tick > startLyricTick + 11) {
                break;
            }
            }
        }



        //output(event);
        }
        // out("-------------------------------
----------------------------------------------");
    }

    return melodyChannel;
```

```
    }

    public static int getLanguage() {
    return lang;
    }

    public static String getTitle() {
    return title;
    }

    public static String getPerformer() {
    return performer;
    }

    public static void dumpLyrics() {
    for (LyricLine lyric: lyricLines) {
        System.out.println(lyric.line);
    }
    }


    /**
     * Build a vector of lyric lines
     * Each line has a start and an end tick
     * and a string for the lyrics in that line
     */
    public static Vector<LyricLine> getLyrics()
```

```
{
    if (lyricLines != null) {
        return lyricLines;
    }

    lyricLines = new Vector<LyricLine> ();
    LyricLine nextLyricLine = new LyricLine();
    StringBuffer buff = new StringBuffer();
    long ticks = 0L;

    Track[] tracks = sequence.getTracks();
    for (int nTrack = 0; nTrack <
tracks.length; nTrack++) {
        for (int n = 0; n <
tracks[nTrack].size(); n++) {
        MidiEvent evt = tracks[nTrack].get(n);
        MidiMessage msg = evt.getMessage();
        ticks = evt.getTick();

        if (msg instanceof MetaMessage) {
            Debug.println("Got a meta mesg in
seq");
            if (((MetaMessage) msg).getType()
== Constants.MIDI_TEXT_TYPE) {
            MetaMessage message = (MetaMessage)
msg;
```

```
            byte[] data = message.getData();
            String str = new String(data);
            Debug.println("Got a text mesg in
seq \"" + str + "\" " + ticks);

            if (ticks == 0) {
                if (str.startsWith("@L")) {
                lang =
decodeLang(str.substring(2));
                } else if
(str.startsWith("@T")) {
                if (title == null) {
                    title = str.substring(2);
                } else {
                    performer =
str.substring(2);
                }
                }

            }
            if (ticks > 0) {
                //if (str.equals("\r") ||
str.equals("\n")) {
                if ((data[0] == '/') ||
(data[0] == '\\')) {
                if (buff.length() == 0) {
                    // blank line -  maybe at
```

```
start of song
                    // fix start time from
NO_TICK
                    nextLyricLine.startTick =
ticks;
                } else {
                    nextLyricLine.line =
buff.toString();
                    nextLyricLine.endTick =
ticks;

lyricLines.add(nextLyricLine);
                    buff.delete(0,
buff.length());

                    nextLyricLine = new
LyricLine();
                }
                buff.append(str.substring(1));
                } else {
                if (nextLyricLine.startTick ==
Constants.NO_TICK) {
                    nextLyricLine.startTick =
ticks;
                }
                buff.append(str);
                }
```

```
            }
            }
        }
        }
        // save last line (but only once)
        if (buff.length() != 0) {
        nextLyricLine.line = buff.toString();
        nextLyricLine.endTick = ticks;
        lyricLines.add(nextLyricLine);
        buff.delete(0, buff.length());
        }
    }
    if (Debug.DEBUG) {
        dumpLyrics();
    }
    return lyricLines;
    }

    /**
     * gets a vector of lyric notes
     * side-effect: sets last tick
     */
    public static Vector<DurationNote>
getMelodyNotes() {
    if (melodyChannel == -1) {
        getMelodyChannel();
    }
```

```
    if (melodyNotes != null) {
        return melodyNotes;
    }

    melodyNotes = new Vector<DurationNote> ();
    Vector<DurationNote> unresolvedNotes = new
Vector<DurationNote> ();

    Track[] tracks = sequence.getTracks();
    for (int nTrack = 0; nTrack <
tracks.length; nTrack++) {
        for (int n = 0; n <
tracks[nTrack].size(); n++) {
        MidiEvent evt = tracks[nTrack].get(n);
        MidiMessage msg = evt.getMessage();
        long ticks = evt.getTick();

        if (msg instanceof ShortMessage) {
            ShortMessage smsg= (ShortMessage)
msg;
            if (smsg.getChannel() ==
melodyChannel) {
            int note = smsg.getData1();
            if (note < Constants.MIDI_NOTE_A0
|| note > Constants.MIDI_NOTE_C8) {
                continue;
```

```
            }

            if (smsg.getCommand() ==
Constants.MIDI_NOTE_ON) {
                // note on
                DurationNote dnote = new
DurationNote(ticks, note);
                melodyNotes.add(dnote);
                unresolvedNotes.add(dnote);

            } else if (smsg.getCommand() ==
Constants.MIDI_NOTE_OFF) {
                // note off
                for (int m = 0; m <
unresolvedNotes.size(); m++) {
                DurationNote dnote =
unresolvedNotes.elementAt(m);
                if (dnote.note == note) {
                    dnote.duration = ticks -
dnote.startTick;
                    dnote.endTick = ticks;
                    unresolvedNotes.remove(m);
                }
                }

            }
```

```
            }
        }
        }
    }
    return melodyNotes;
    }

    public static int getMaxMelodyNote() {
    if (maxNote == -1) {
        getMaxMin();
    }
    return maxNote;
    }

    public static int getMinMelodyNote() {
    if (minNote == Integer.MAX_VALUE) {
        getMaxMin();
    }
    return minNote;
    }

    private static void getMaxMin() {
    StringBuffer buff = new StringBuffer();

    Track[] tracks = sequence.getTracks();
    for (int nTrack = 0; nTrack <
tracks.length; nTrack++) {
```

```
        for (int n = 0; n <
tracks[nTrack].size(); n++) {
        MidiEvent evt = tracks[nTrack].get(n);
        MidiMessage msg = evt.getMessage();
        long ticks = evt.getTick();

        if (msg instanceof ShortMessage) {
            ShortMessage smsg= (ShortMessage)
msg;
            if (smsg.getChannel() ==
melodyChannel) {
            if (smsg.getCommand() !=
Constants.MIDI_NOTE_OFF) {
                // not note on
                continue;
            }
            int note = smsg.getData1();
            if (note < 21 || note > 107) {
                continue;
            }
            //Debug.println("Note is " + note);
            if (note > maxNote) {
                Debug.println("Setting max to
"+ note);
                maxNote = note;
            } else if (note < minNote) {
                minNote = note;
```

```
                }
                }
            }
            }
        }
        }

        private static int decodeLang(String str) {
        if (str.equals("ENG")) {
            return SongInformation.ENGLISH;
        }
        if (str.equals("CHI")) {
            return SongInformation.CHINESE1;
        }
        return -1;
        }
    }
```

PinYin

For Chinese language files, one of my aims was to display the PinYin (Romanised form) of the Chinese hierographic characters. For this, I need to be able rewrite any sequence of Chinese characters into their PinYin form. I couldn't find a list of characters and their corresponding characters. The closest is the Chinese-English Dictionary from which you can download the dictionary as a text file. Typical lines in this file are

```
不賴 不赖 [bu4 lai4] /not bad/good/fine/
```

Each line has the Traditional characters followed by the Simplified characters, the PinYin in [...] and then English meanings.

I used the following shell script to make a list of character/PinYin pairs:

```
#!/bin/bash

# get pairs of character + pinyin by throwing
away other stuff in the dictionary

awk '{print $2, $3}' cedict_ts.u8 | grep -v
'[A-Z]' |
  grep -v '^.[^ ]' | sed -e 's/\[//' -e
's/\]//' -e 's/[0-9]$//' |
    sort | uniq -w 1 > pinyinmap.txt
```

to give lines such as

```
好 hao
妁 shuo
如 ru
妃 fei
```

This can then be read into a Java Map, and then quick lookups can be done to translate Chinese to PinYin.

Karaoke player with sampling

The Karaoke player described so far is functionally equivalent to kmidi and pykar. It plays KAR files, shows the notes and scrolls through the lyrics. To sing along with it, you need to use an ALSA or PulseAudio player.

But Java can also play sampled sounds, as discussed in an earlier chapter. So that code can be brought into the Karaoke player to give a more complete solution. For MIDI, Java normally gives only a Gervill synthesizer which is a software synthesizer that plays out through The PulseAudio default device. The actual output device is not accessible through Java, and is controlled by the underlying PulseAudio output device. But for sampled media, the input devices can be controlled. So in the following code a selection box allows choice of sampled input device, and leaves the output device to the default.

```java
/*
 * KaraokePlayer.java
 *
 */

import javax.swing.*;
import javax.sound.sampled.*;

public class KaraokePlayerSampled {


    public static void main(String[] args)
throws Exception {
    if (args.length != 1) {
        System.err.println("KaraokePlayer:
usage: " +
                    "KaraokePlayer <midifile>");
        System.exit(1);
    }
    String    strFilename = args[0];

    Mixer.Info[] mixerInfo =
AudioSystem.getMixerInfo();

    String[] possibleValues = new
String[mixerInfo.length];
```

```
    for(int cnt = 0; cnt < mixerInfo.length;
cnt++){
        possibleValues[cnt] =
mixerInfo[cnt].getName();
    }
    Object selectedValue =
JOptionPane.showInputDialog(null, "Choose
mixer", "Input",

JOptionPane.INFORMATION_MESSAGE, null,
                                possibleValues,
possibleValues[0]);

    System.out.println("Mixer string selected "
+ ((String)selectedValue));



    Mixer mixer = null;
    for(int cnt = 0; cnt < mixerInfo.length;
cnt++){
        if
(mixerInfo[cnt].getName().equals((String)selected
 {
        mixer =
AudioSystem.getMixer(mixerInfo[cnt]);
        System.out.println("Got a mixer");
        break;
```

```
        }
    }//end for loop


    MidiPlayer midiPlayer = new MidiPlayer();
    midiPlayer.playMidiFile(strFilename);

        SampledPlayer sampledPlayer = new
SampledPlayer(/* midiPlayer.getReceiver(), */
mixer);
        sampledPlayer.playAudio();
    }
}
```

The code to play the sampled media is pretty much the same as we have seen before:

```
import java.io.IOException;

import javax.sound.sampled.Line;
import javax.sound.sampled.Mixer;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.TargetDataLine;
import javax.sound.sampled.FloatControl;
import
javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.SourceDataLine;
import javax.sound.sampled.Control;

import javax.swing.*;

public class SampledPlayer {

    private DisplayReceiver receiver;
    private Mixer mixer;

    public SampledPlayer(/* DisplayReceiver
receiver, */ Mixer mixer) {
    this.receiver = receiver;
    this.mixer = mixer;
```

```
    }

    //This method creates and returns an
    // AudioFormat object for a given set of
format
    // parameters.  If these parameters don't
work
    // well for you, try some of the other
    // allowable parameter values, which are
shown
    // in comments following the declarations.
    private static AudioFormat getAudioFormat()
{
    float sampleRate = 44100.0F;
    //8000,11025,16000,22050,44100
    int sampleSizeInBits = 16;
    //8,16
    int channels = 1;
    //1,2
    boolean signed = true;
    //true,false
    boolean bigEndian = false;
    //true,false
    return new AudioFormat(sampleRate,
                           sampleSizeInBits,
                           channels,
                           signed,
```

```
                    bigEndian);
    }//end getAudioFormat



    public  void playAudio() throws Exception {
    AudioFormat audioFormat;
    TargetDataLine targetDataLine;

    audioFormat = getAudioFormat();
    DataLine.Info dataLineInfo =
        new DataLine.Info(
                  TargetDataLine.class,
                  audioFormat);
    targetDataLine = (TargetDataLine)
        AudioSystem.getLine(dataLineInfo);

    targetDataLine.open(audioFormat,
                 audioFormat.getFrameSize() *
Constants.FRAMES_PER_BUFFER);
    targetDataLine.start();

    /*
    for (Control control:
targetDataLine.getControls()) {
        System.out.println("Target control: " +
control.getType());
```

```
    }
    */

    playAudioStream(new
AudioInputStream(targetDataLine), mixer);
    } // playAudioFile

    /** Plays audio from the given audio input
stream. */
    public  void
playAudioStream(AudioInputStream
audioInputStream, Mixer mixer) {

    new AudioPlayer(audioInputStream,
mixer).start();
    } // playAudioStream

    class AudioPlayer extends Thread {
    AudioInputStream audioInputStream;
    SourceDataLine dataLine;
    AudioFormat audioFormat;

    // YIN stuff
    // PitchProcessorWrapper ppw;

    AudioPlayer( AudioInputStream
audioInputStream, Mixer mixer) {
```

```
        this.audioInputStream =
audioInputStream;

        // Set to nearly max, like Midi
sequencer does
        Thread curr = Thread.currentThread();
        Debug.println("Priority on sampled: " +
curr.getPriority());
        int priority = Thread.NORM_PRIORITY
        + ((Thread.MAX_PRIORITY -
Thread.NORM_PRIORITY) * 3) / 4;
        curr.setPriority(priority);
        Debug.println("Priority now on sampled:
" + curr.getPriority());

        // Audio format provides information
like sample rate, size, channels.
        audioFormat =
audioInputStream.getFormat();
        Debug.println( "Play input audio
format=" + audioFormat );

        // Open a data line to play our type of
sampled audio.
        // Use SourceDataLine for play and
TargetDataLine for record.
```

```
        if (mixer == null) {
        System.out.println("can't find a
mixer");
        } else {
        Line.Info[] lines =
mixer.getSourceLineInfo();
        if (lines.length >= 1) {
            try {
            dataLine = (SourceDataLine)
AudioSystem.getLine(lines[0]);
            System.out.println("Got a source
line for " + mixer.toString());
            } catch(Exception e) {
            }
        } else {
            System.out.println("no source lines
for this mixer " + mixer.toString());
        }
        }

        for (Control control:
mixer.getControls()) {
            System.out.println("Mixer control:
" + control.getType());
        }
```

```
        DataLine.Info info = null;
        if (dataLine == null) {
        info = new DataLine.Info(
SourceDataLine.class, audioFormat );
        if ( !AudioSystem.isLineSupported( info
) ) {
            System.out.println(
"Play.playAudioStream does not handle this type
of audio on this system." );
            return;
        }
        }

        try {
        // Create a SourceDataLine for play
back (throws LineUnavailableException).
        if (dataLine == null) {
            dataLine = (SourceDataLine)
AudioSystem.getLine( info );
        }
        Debug.println( "SourceDataLine class="
+ dataLine.getClass() );

        // The line acquires system resources
(throws LineAvailableException).
        dataLine.open( audioFormat,
```

```
                    audioFormat.getFrameSize() *
Constants.FRAMES_PER_BUFFER);

        for (Control control:
dataLine.getControls()) {
            System.out.println("Source control:
" + control.getType());
        }
        // Adjust the volume on the output
line.
        if( dataLine.isControlSupported(
FloatControl.Type.VOLUME) ) {
            // if( dataLine.isControlSupported(
FloatControl.Type.MASTER_GAIN ) ) {
            //FloatControl volume =
(FloatControl) dataLine.getControl(
FloatControl.Type.MASTER_GAIN );
            FloatControl volume =
(FloatControl) dataLine.getControl(
FloatControl.Type.VOLUME);
            System.out.println("Max vol " +
volume.getMaximum());
            System.out.println("Min vol " +
volume.getMinimum());
            System.out.println("Current vol " +
volume.getValue());
            volume.setValue( 60000.0F );
```

```
            System.out.println("New vol " +
volume.getValue());
        } else {
            System.out.println("Volume control
not supported");
        }
        if (dataLine.isControlSupported(
FloatControl.Type.REVERB_RETURN)) {
            System.out.println("reverb return
supported");
        } else {
            System.out.println("reverb return
not supported");
        }
        if (dataLine.isControlSupported(
FloatControl.Type.REVERB_SEND)) {
            System.out.println("reverb send
supported");
        } else {
            System.out.println("reverb send not
supported");
        }



        } catch ( LineUnavailableException e )
{
        e.printStackTrace();
```

```
        }

        // ppw = new
PitchProcessorWrapper(audioInputStream,
receiver);
    }


    public void run() {

        // Allows the line to move data in and
out to a port.
        dataLine.start();

        // Create a buffer for moving data from
the audio stream to the line.
        int bufferSize = (int)
audioFormat.getSampleRate() *
audioFormat.getFrameSize();
        bufferSize =
audioFormat.getFrameSize() *
Constants.FRAMES_PER_BUFFER;
        Debug.println("Buffer size: " +
bufferSize);
        byte [] buffer = new byte[bufferSize];

        try {
```

```
        int bytesRead = 0;
        while ( bytesRead >= 0 ) {
            bytesRead = audioInputStream.read(
buffer, 0, buffer.length );
            if ( bytesRead >= 0 ) {
            int framesWritten = dataLine.write(
buffer, 0, bytesRead );
            // ppw.write(buffer, bytesRead);
            }
        } // while
        } catch ( IOException e ) {
        e.printStackTrace();
        }

        // Continues data line I/O until its
buffer is drained.
        dataLine.drain();

        Debug.println( "Sampled player closing
line." );
        // Closes the data line, freeing any
resources such as the audio device.
        dataLine.close();
    }
    }

    // Turn into a GUI version or pick up from
```

```
prefs
    public void listMixers() {
    try{
        Mixer.Info[] mixerInfo =
        AudioSystem.getMixerInfo();
        System.out.println("Available
mixers:");
        for(int cnt = 0; cnt <
mixerInfo.length;
        cnt++){
        System.out.println(mixerInfo[cnt].
                   getName());

        Mixer mixer =
AudioSystem.getMixer(mixerInfo[cnt]);
        Line.Info[] sourceLines =
mixer.getSourceLineInfo();
        for (Line.Info s: sourceLines) {
            System.out.println("  Source line:
" + s.toString());
        }
        Line.Info[] targetLines =
mixer.getTargetLineInfo();
        for (Line.Info t: targetLines) {
            System.out.println("  Target line:
" + t.toString());
        }
```

```
        }//end for loop
    } catch(Exception e) {
    }
    }
}
```

Comments on device choices

if the default devices are chosen, the input and output devices are the PulseAudio default devices. Normally these would both be the computer's sound card. However, the default devices can be changed using for example, the PulseAudio volume control. These can set either the input device, the output device or both. The dialogue can also be used to set the input device for sampled media.

This raises a number of possible scenarios:

- The default PulseAudio device selects the same device for input and output
- The default PulseAudio device selects different devices for input and output
- The default PulseAudio device is used for output while the ALSA device is used for input, but the physical device is the same
- The default PulseAudio device is used for output while the ALSA device is used for input, and the physical devices are different

Using different devices raises the problem of "clock drift", where the devices have different clocks which are not synchronised. The worst case seems to be the second one, where over a three minute song on my system I could hear a noticeable lag in playing the sampled audio, while the KAR file played happily. It also introduced a noticeable latency in playing the sampled audio.

Performance

The program `top` can give a good idea of how much CPU is used by various processes. My current computer is a high-end Dell laptop with a quad-core Intel i7-2760QM CPU running at 2.4GHz. According to CPU Benchmarks the processor is in the "High End CPU Chart." On this computer, tested with various KAR files, PulseAudio takes about 30% of the CPU while Java takes about 60%. On occassions these figures exceeded. There is not much left for additional functionality!

In addition, while playing a MIDI file, sometimes the Java process hangs, resuming with upto 600% CPU usage (I don't know how `top` manages to record that)!. This makes it effectively unusable, and I am not sure where the problem lies.

Downloads

The Java source files are here and the Chinese lookup table is here

Conclusion

JavaSound has no direct support for Karaoke. This chapter has looked at how to combine the JavaSound libraries with other libraries such as Swing to give a Karaoke player for MIDI files. It requires a high-end computer to run these programs.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

Many Karaoke systems use subtitles imposed over a movie of some kind. This chapter looks at how to do this with Linux systems. There are limited choices, but it is possible.

Resources

- Subtitling with Linux Tutorial

Introduction

Programs like `kmid` and my Java programs play lyrics on some sort of canvas object. This gives a pretty boring background. Video CDs or MPEG 4 files have a nicer background, but have the lyrics hard-coded onto the background video so there is little chance for manipulation of them. CD+G files keep the lyrics separate from the video but there doesn't seem to be any way of playing them directly from Linux. They can be converted to MP3+G and these can be played by `vlc`: load the MP3 file and it will pick up the corresponding `.cdg` file.

This chapter considers subtitles that can be created independently, combined with video and audio in some way and then played. The current situation is not completely satisfactory.

Subtitle formats

We are concerned here with what are called "soft subtitles" where the subtitles are stored in a separate file to the video or audio and are combined during rendering. The Wikipedia page Subtitle (captioning) is a long article going into many issues about subtitling. It also contains a list of subtitle formats, but the one that seems to be of most use in in this context is SubStation Alpha.

MPlayer

According to the MPlayer page Subtitles and OSD The formats recognised by `mplayer` are

- VOBsub
- OGM
- CC (closed caption)
- MicroDVD
- SubRip
- SubViewer
- Sami
- VPlayer
- RT
- SSA
- PJS (Phoenix Japanimation Society)
- MPsub
- AQTitle
- JACOsub

VLC

According to the VLC playback Features - Subtitle/Tag VLC support under Linux includes the following subtitle formats:

- DVD
- Text files (MicroDVD, SubRIP, SubViewer, SSA1-5, SAMI, VPlayer)
- Closed captions
- Vobsub
- Universal Subtitle Format (USF)
- SVCD / CVD
- DVB
- OGM
- CMML
- Kate

If you play some sort of video file, say XYZ.mpg, and there is also a file with the same root name and appropriate extension such as XYZ.ass (extension for SubStation Alpha), then VLC will automatically load the subtitles file and play it. If the

subtitles file has a different name then it can be loaded from the VLC menu Video->Subtitles Track. However, this does not appear to be as reliable as sharing the name.

GStreamer

See Playback tutorial 2: Subtitle management

See subtitleoverlay (Pipeline given doesn't work).

Has plugin assrender.

Gnome subtitles

See Gnome Subtitles 1.3 is out! supports Adobe Encore DVD Advanced Sub Station Alpha AQ Title DKS Subtitle Format FAB Subtitler Karaoke Lyrics LRC Karaoke Lyrics VKT MacSUB MicroDVD MPlayer MPlayer 2 MPSub Panimator Phoenix Japanimation Society Power DivX Sofni SubCreator 1.x SubRip Sub Station Alpha SubViewer 1.0 SubViewer 2.0 ViPlay Subtitle File

SubStation Alpha

The SSA/ASS specification is at MooDub.free It is brief and appears to contain some minor errors with respect to later specifications and implementations: for example, the time format is different. Or are the later ones all wrong?

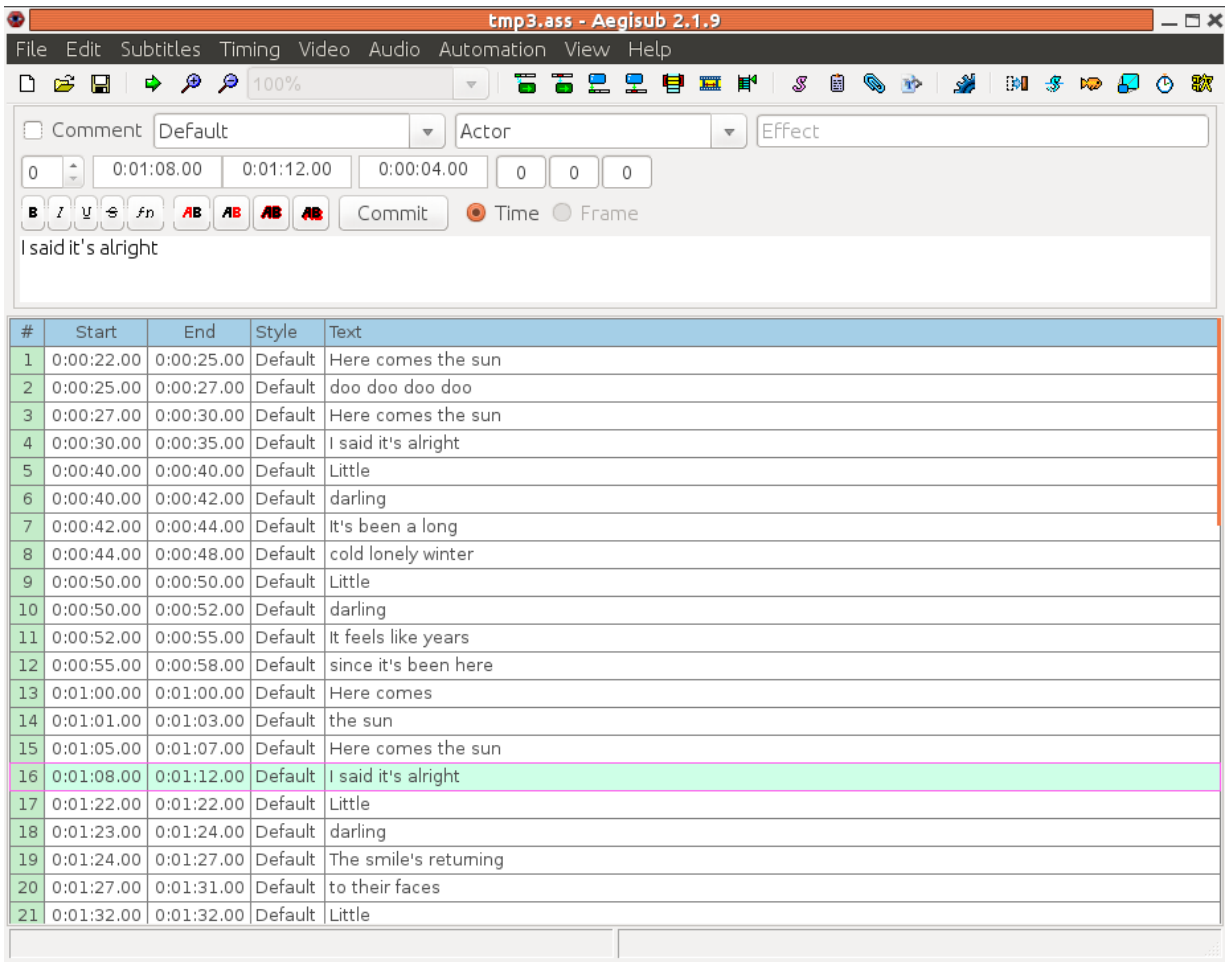
SSA/ASS files can be used standalone. They can also be included in container formats such as Matroska files, discussed briefly in the Codecs chapter. When they are embedded into MKV files, some restrictions are made, such as the text being converted into UTF-8 Unicode.

ASS files are divided into several sections:

- General information about the environment the subtitle file expects, such as the X and Y resolutions
- Style information such as colours and fonts
- Event information, which is where the subtitle text is given along with timing information and any special effects to be applied

Under normal circumstances you would not directly create such files using a text editor. Instead, the program `Aegisub` gives you a GUI environment in which to create the files. Essentially, you just enter the text by lines, plus the start and end times for the line to be displayed.

A screen dump is



Many special effects are possible. The video on Bill Cresswell's blog is an excellent example. Here is the direct YouTube link.

For completeness, here is part of an ASS file I created:

```
[Script Info]
; Script generated by Aegisub 2.1.9
; http://www.aegisub.org/
Title: Default Aegisub file
ScriptType: v4.00+
WrapStyle: 0
PlayResX: 640
PlayResY: 480
ScaledBorderAndShadow: yes
Video Aspect Ratio: 0
Video Zoom: 6
Video Position: 0

[V4+ Styles]
Format: Name, Fontname, Fontsize,
PrimaryColour, SecondaryColour, OutlineColour,
BackColour, Bold, Italic, Underline, StrikeOut,
ScaleX, ScaleY, Spacing, Angle, BorderStyle,
Outline, Shadow, Alignment, MarginL, MarginR,
MarginV, Encoding
Style:
Default,Arial,20,&H00FFFFFF,&H00B4FCFC,&H00000008



[Events]
Format: Layer, Start, End, Style, Name,
```

```
MarginL, MarginR, MarginV, Effect, Text
Dialogue:
0,0:00:18.22,0:00:19.94,Default,,0000,0000,0000,,
 comes the sun
Dialogue:
0,0:00:20.19,0:00:21.75,Default,,0000,0000,0000,,
 doo doo doo
Dialogue:
0,0:00:22.16,0:00:24.20,Default,,0000,0000,0000,,
 comes the sun
Dialogue:
0,0:00:24.61,0:00:28.24,Default,,0000,0000,0000,,
 said it's alright
...
```

Karaoke effects in ASS files

A line in an ASS file essentially consists of a time to start display, a time to finish the display and the text itself. However, Karaoke users are accustomed to the text being highlighted as it is played.

ASS supports two major highlight styles:

- Words are highlighted one at a time
- The text is highlighted by filling from the left

These effects are done by embedding "Karaoke overrides" into the text. These are in {} brackets with a duration time in hundredths of a second.

The details are

- **Word highlighting**: An override of the form {\k} will highlight the following word for *time* hundredths of a second. An example would be

  ```
  {\k100}Here {\k150}comes {\k50}the
  {\k150}sun
  ```

- **Fill highlighting**: An override of the form {\kf} will progressively fill up the following word for *time* hundredths of a second. An example would be

```
{\kf100}Here {\kf150}comes {\kf50}the
{\kf150}sun
```

The three styles appear as

- Lines with no highlighting

- Word highlighting

- Fill highlighting

Here comes the sun

Multi-line Karaoke

Ideally, a Karaoke system should have a "look ahead" mechanism whereby you can see the next line before having to sing it. This can be done by showing two lines of text with overlapping times at different heights. The algorithm is:

```
When line N with markup is shown,
    show line N+1 without markup
After line N is finished, continue showing line
N+1
When line N+1 is due to show,
     finish showing unmarked line N+1
     show line N+1 with markup
```

For the song Here Comes the Sun with lyrics

```
Here comes the sun
doo doo doo doo
Here comes the sun
I said it's alrigh
```

the resultant ASS file should look like

```
Dialogue:
0,0:00:18.22,0:00:19.94,Default,,0000,0000,0100,,
{\kf16}Here {\kf46}comes {\kf43}the {\kf67}sun
Dialogue:
0,0:00:18.22,0:00:20.19,Default,,0000,0000,0000,,
 doo doo doo
Dialogue:
0,0:00:20.19,0:00:21.75,Default,,0000,0000,0000,,
{\kf17}doo {\kf25}doo {\kf21}doo {\kf92}doo
Dialogue:
0,0:00:20.19,0:00:22.16,Default,,0000,0000,0100,,
 comes the sun
Dialogue:
0,0:00:22.16,0:00:24.20,Default,,0000,0000,0100,,
{\kf17}Here {\kf46}comes {\kf43}the {\kf97}sun
Dialogue:
0,0:00:22.16,0:00:24.61,Default,,0000,0000,0000,,
 said it's alright
```

with appearance

Here comes the sun
doo doo doo doo

libass

Sub Station Alpha and its renderers appear to have been through a complex history. According to The old and present: VSFilter the ASS format was finalised about 2004, and the renderer VSFilter was made open source at this time. However, around 2007 development of VSFilter ceased and several forks were made. These introduced several extensions to the format, such as the *blur* tag by Aegisub. Some of these forks since merged, some were abandoned, and for some of these forks there is still code in the wild.

libass is the main rendering library for Linux. An alternative, xy-vsfilter, claims to be faster, more reliable, etc but does not seem to have a Linux implementation. libass supports some of the later extensions - these seem to be the Aegisub 2008 extensions: VSFilter hacks .

Converting KAR files to MKV files with ASS subtitles

Well, it's not hard...

- In order to pull out the lyrics form a KAR or MIDI file, use the Java `DumpSequence` given in the MIDI - JavaSound chapter, as in

  ```
  java DumpSequence  song.kar  > song.dump
  ```

to get a dump of all events

- For line-only display, use the following Python script to extract the lyrics and save them in ASS format ```

!/usr/bin/python

import fileinput import string import math

TEXT_STR = "Dialogue: 0,%s,%s,Default,,0000,0000,0000,,"

textStr = TEXT_STR startTime = -1 endTime = -1

def printPreface(): print '[Script Info]\r\n\ ; Script generated by Aegisub 2.1.9\r\n\ ; http://www.aegisub.org/\r\n\ Title: Default Aegisub file\r\n\ ScriptType: v4.00+\r\n\ WrapStyle: 0\r\n\ PlayResX: 640\r\n\ PlayResY: 480\r\n\ ScaledBorderAndShadow: yes\r\n\ Video Aspect Ratio: 0\r\n\ Video Zoom: 6\r\n\ Video Position: 0\r\n\ \r\n\ [V4+ Styles]\r\n\ Format: Name, Fontname, Fontsize, PrimaryColour, SecondaryColour, OutlineColour, BackColour, Bold, Italic, Underline, StrikeOut, ScaleX, ScaleY, Spacing, Angle, BorderStyle, Outline, Shadow, Alignment, MarginL, MarginR, MarginV, Encoding\r\n\ Style: Default,Arial,36,&H00FFFFFF,&H000000FF,&H00000000,&H00000000,0,0,0,0,100,100,0,0,1,2,2,2,10,10,10,1\r\n\ \r\n\ [Events]\r\n\ Format: Layer, Start, End, Style, Name, MarginL, MarginR, MarginV, Effect, Text'

def timeFormat(s): global microSecondsPerTick

```
tf = float(s)
tf /= 62.6  #ticks per sec

# This should be right , but is too slow
#tf = (tf * microSecondsPerTick) / 1000000

t = int(math.floor(tf))
hundredths = round((tf-t)*100)
secs = t % 60
t /= 60
mins = t % 60
t /= 60
hrs = t
return "%01d:%02d:%02d.%02d" % (hrs, mins,
secs, hundredths)
```

def doLyric(words): global textStr global startTime global endTime global TEXT_STR

```
if words[1] == "0:":
    #print "skipping"
    return

time = string.rstrip(words[1], ':')
if startTime == -1:
    startTime = time
#print words[1],
if len(words) == 5:
    if words[4][0] == '\\' or words[4][0] ==
'/':
        #print "My name is %s and weight is %d
kg!" % ('Zara', 21)
        #print startTime, endTime
        if len(words[4]) == 1:
            print textStr %
(timeFormat(startTime), timeFormat(endTime)) +
"\r\n",
        textStr = TEXT_STR + words[4][1:]
        startTime = -1
    else:
        textStr += words[4]
else:
    textStr += ' '

endTime = time
```

printPreface()

for line in fileinput.input(): words = line.split()

```
if len(words)  >= 2:
    if words[0] == "Resolution:":
        ticksPerBeat = words[1]
    elif words[0] == "Length:":
        numTicks = int(words[1])
    elif words[0] == "Duration:":
        duration = int(words[1])
        microSecondsPerTick = duration/numTicks
        # print "Duration %d numTicks %d
microSecondsPerTick %d" % (duration, numTicks,
microSecondsPerTick)

if len(words) >= 3 and words[2] == "Text":
    doLyric(words)
```

```
as in
```

python lyric2ass4kar.py song.dump > song.ass

```
+ For fill lyrics display, use the following
Python script to extract the lyrics and
save them in ASS format
```

!/usr/bin/python

import fileinput import string import math

TEXT_STR = "Dialogue: 0,%s,%s,Default,,0000,0000,0000,,"

textStr = "{\kf%d}" plainTextStr = "" startTime = -1 startWordTime = -1 endTime = -1

def printPreface(): print '[Script Info]\r\n\ ; Script generated by Aegisub 2.1.9\r\n\ ; http://www.aegisub.org/\r\n\ Title: Default Aegisub file\r\n\ ScriptType: v4.00+\r\n\ WrapStyle: 0\r\n\ PlayResX: 640\r\n\ PlayResY: 480\r\n\ ScaledBorderAndShadow: yes\r\n\ Video Aspect Ratio: 0\r\n\ Video Zoom: 6\r\n\ Video Position: 0\r\n\ \r\n\ [V4+ Styles]\r\n\ Format: Name, Fontname, Fontsize, PrimaryColour, SecondaryColour, OutlineColour, BackColour, Bold, Italic, Underline, StrikeOut, ScaleX, ScaleY, Spacing, Angle, BorderStyle, Outline, Shadow, Alignment, MarginL, MarginR, MarginV, Encoding\r\n\ Style: Default,Arial,36,&H00FFFFFF,&H000000FF,&H00000000,&H

00000000,0,0,0,0,100,100,0,0,1,2,2,2,10,10,10,1\r\n\ \r\n\ [Events]\r\n\ Format: Layer, Start, End, Style, Name, MarginL, MarginR, MarginV, Effect, Text'

def timeFormat(s): global microSecondsPerTick

```
tf = float(s)

# frames per sec should be 60: 120 beats/min,
30 ticks per beat
# but it is too slow on 54154
tf /= 62.6  #ticks per sec

# This should be right , but is too slow
# tf = (tf * microSecondsPerTick) / 1000000

t = int(math.floor(tf))
hundredths = round((tf-t)*100)
secs = t % 60
t /= 60
mins = t % 60
t /= 60
hrs = t
return "%01d:%02d:%02d.%02d" % (hrs, mins,
secs, hundredths)
```

def durat(end, start): fend = float(end) fstart = float(start) d = (fend - fstart) / 62.9

```
#print end, start, d
return round(d*100)
```

def doLyric(words): global textStr global plainTextStr global startTime global endTime global TEXT_STR global startWordTime global lineNum

```
if words[1] == "0:":
    #print "skipping"
    return

time = string.rstrip(words[1], ':')
if startTime == -1:
    startTime = time
    startWordTime = time
    previousEndTime = time
#print words[1],
if len(words) == 5:
    if words[4][0] == '\\' or words[4][0] ==
'/':
        #print "My name is %s and weight is %d
kg!" % ('Zara', 21)
        #print startTime, endTime
        dur = durat(time, startWordTime)
        textStr = textStr % (dur)
        if len(words[4]) == 1:
            print TEXT_STR %
(timeFormat(startTime),

timeFormat(endTime)) + \
                                textStr + "\r\n",

        # next word
```

```
        textStr = "{\kf%d}" + words[4][1:]
        startTime = -1
    else:
        textStr += words[4]
else:
    # it's a space, gets lost by the split
    dur = durat(time, startWordTime)
    textStr = textStr % (dur) + " {\kf%d}"
    startWordTime = time

endTime = time
```

printPreface()

print "Dialogue: 0,0:00:18.22,0:00:19.94,Default,,0000,0000,0000,,{\k16}Here {\k46}comes {\k43}the {\k67}sun"

for line in fileinput.input(): words = line.split()

```
if len(words)  >= 2:
    if words[0] == "Resolution:":
        ticksPerBeat = words[1]
    elif words[0] == "Length:":
        numTicks = int(words[1])
    elif words[0] == "Duration:":
        duration = int(words[1])
        microSecondsPerTick = duration/numTicks
        # print "Duration %d numTicks %d
microSecondsPerTick %d" % (duration, numTicks,
microSecondsPerTick)

if len(words) >= 3 and words[2] == "Text":
    doLyric(words)
```

```
as in
```

python lyric2karaokeass4kar.py song.dump > song.ass

```
+ For multiline lyrics display, use the
following Python script to extract the lyrics
and
save them in ASS format
```

!/usr/bin/python

import fileinput import string import math

START_EVENTS = ["Dialogue: 0,%s,%s,Default,,0000,0000,0000,,", "Dialogue: 0,%s,%s,Default,,0000,0000,0100,,"]

TEXT_STR = "Dialogue: 0,%s,%s,Default,,0000,0000,0000,,"
TEXT_STR2 = "Dialogue: 0,%s,%s,Default,,0000,0000,0100,,"

textStr = "{\kf%d}" plainTextStr = "" startTime = -1 previousStartTime = -1 startWordTime = -1 endTime = -1 previousEndTime = -1 lineNum = 0

def printPreface(): print '[Script Info]\r\n\ ; Script generated by Aegisub 2.1.9\r\n\ ; http://www.aegisub.org/\r\n\ Title: Default Aegisub file\r\n\ ScriptType: v4.00+\r\n\ WrapStyle: 0\r\n\ PlayResX: 640\r\n\ PlayResY: 480\r\n\ ScaledBorderAndShadow: yes\r\n\ Video Aspect Ratio: 0\r\n\ Video Zoom: 6\r\n\ Video Position: 0\r\n\ \r\n\ [V4+ Styles]\r\n\ Format: Name, Fontname, Fontsize, PrimaryColour, SecondaryColour, OutlineColour, BackColour, Bold, Italic, Underline, StrikeOut, ScaleX, ScaleY, Spacing, Angle,

BorderStyle, Outline, Shadow, Alignment, MarginL, MarginR, MarginV, Encoding\r\n\ Style: Default,Arial,36,&H00FFFFFF,&H000000FF,&H00000000,&H00000000,0,0,0,0,100,100,0,0,1,2,2,2,10,10,10,1\r\n\ \r\n\ [Events]\r\n\ Format: Layer, Start, End, Style, Name, MarginL, MarginR, MarginV, Effect, Text'

def timeFormat(s): global microSecondsPerTick

```
tf = float(s)
# print "factori is %f instead of %f" %
((1.0*microSecondsPerTick / 1000000),
(1.0/62.9))
# frames per sec should be 60: 120 beats/min,
30 ticks per beat
# but it is too slow on 54154
tf /= 62.6  #ticks per sec

# This should be right , but is too slow
# tf = (tf * microSecondsPerTick) / 1000000

t = int(math.floor(tf))
hundredths = round((tf-t)*100)
secs = t % 60
t /= 60
mins = t % 60
t /= 60
hrs = t
return "%01d:%02d:%02d.%02d" % (hrs, mins,
secs, hundredths)
```

def durat(end, start): fend = float(end) fstart = float(start) d = (fend - fstart) / 62.9

```
#print end, start, d
return round(d*100)
```

def doLyric(words): global textStr global plainTextStr global startTime global endTime global previousStartTime global previousEndTime global TEXT_STR global startWordTime global lineNum

```
if words[1] == "0:":
    #print "skipping"
    return

time = string.rstrip(words[1], ':')
if startTime == -1:
    startTime = time
    startWordTime = time
    previousEndTime = time
#print words[1],
if len(words) == 5:
    if words[4][0] == '\\' or words[4][0] ==
'/':
        #print "My name is %s and weight is %d
kg!" % ('Zara', 21)
        #print startTime, endTime
        dur = durat(time, startWordTime)
        textStr = textStr % (dur)

        if len(words[4]) == 1:

            if previousStartTime != -1:
                print START_EVENTS[lineNum % 2]
% (timeFormat(previousStartTime),

timeFormat(previousEndTime)) + \
```

```
plainTextStr + "\r\n",
            print START_EVENTS[lineNum % 2] %
(timeFormat(startTime),

timeFormat(endTime)) + \

textStr + "\r\n",

        # next word
        lineNum += 1
        #previousEndTime = time
        textStr = "{\kf%d}" + words[4][1:]
        plainTextStr = words[4][1:]
        previousStartTime = startTime
        startTime = -1
    else:
        textStr += words[4]
        plainTextStr += words[4]
else:
    #print textStr
    #dur = duration(time, startWordTime)
    dur = durat(time, startWordTime)
    textStr = textStr % (dur) + " {\kf%d}"
    plainTextStr += ' '
    startWordTime = time
```

```
endTime = time
```

printPreface()

print "Dialogue: 0,0:00:18.22,0:00:19.94,Default,,0000,0000,0000,,{\k16}Here {\k46}comes {\k43}the {\k67}sun"

for line in fileinput.input(): words = line.split()

```
if len(words)  >= 2:
    if words[0] == "Resolution:":
        ticksPerBeat = words[1]
    elif words[0] == "Length:":
        numTicks = int(words[1])
    elif words[0] == "Duration:":
        duration = int(words[1])
        microSecondsPerTick = duration/numTicks
        # print "Duration %d numTicks %d
microSecondsPerTick %d" % (duration, numTicks,
microSecondsPerTick)

if len(words) >= 3 and words[2] == "Text":
    doLyric(words)
```

```
as in
```

python lyric2karaokeass4kar.py song.dump > song.ass

```
+ Convert the MIDI sound file to a WAV file
using `fluidsynth`:
```

fluidsynth -F song.wav /usr/share/sounds/sf2/FluidR3_GM.sf2 song.kar

```
+ Convert the WAV file to MP3:
```

lame song.wav song.mp3

```
+ Find a suitable video-only file for your
background -
I used one off my Karaoke disks - and then
merge
then into an MKV file:
```

mkvmerge -o 54154.mkv 54154.mp3 54154.ass BACK01.MPG

```
The resultant MKV file can then by played as a
standalone
file by MPlayer:
```

mplayer song.mkv

```
It can also be played by VLC, but only with the
ASS file
present
```

vlc song.mkv ```

Screen captures were shown before, depending on the Karaoke effect chosen.

Timing is however an issue: the default MIDI tempo is 120 beats per minute and a common tick rate is 30 ticks per beat. This leads to a rate of 60 MIDI ticks per second. However, we are now playing MP3 files and ASS files neither of which are MIDI files any more and which are not necessarily synchronised, and with a rate of 60 ticks per second in converting from MIDI to ASS, the lyrics run too slow. Experimentally I have found 62.9 to be a reasonable rate for at least some files...

HTML5 subtitles

HTML 5 has support for video types, although exactly what video format is supported by which brower is variable. Standard audio and video was discussed in the chapter on Streaming HTML5 .

A more recent development has been to include subtitles in HTML5 video. Under Linux it is currently (June 2013) only supported by the Chromium browser. A tutorial is at Video Subtitling and WebVTT: HTML5 Doctor and specified at HTML5: 4.8.9 The track element .

You need to prepare a file of timing and text instructions. The format shown in examples is as a .vtt file and can be

```
WEBVTT

1
00:00:01.000 --> 00:00:30.000  D:vertical
A:start
This is the first line of text, displaying from
1-30 seconds

2
00:00:35.000 --> 00:00:50.000
And the second line of text
separated over two lines from 35 to 50 seconds
```

where the first line is "WEBVTT" and blocks of text are separated by blank lines. The format of VTT files is specified at WebVTT: The Web Video Text Tracks Format .

The HTML then references the A/V files and the subtitle file as in

```
<video  controls>
      <source src="output.webm" controls>
      <track src="54154.vtt" kind="subtitles"
srclang="en" label="English" default />
      <!-- fallback for rubbish browsers -->
    </video>
```

A screen capture is

There does no seem to be any mechanism for highlighting words progressively in a line. Possibly JavaScript may be able to do so, but at a cursory look it doesn't seem likely. This makes it not yet suitable for Karaoke.

Conclusion

This chapter has discussed methods for overlaying subtitle text onto a changing video image. It is feasible, but there are only a few viable mechanisms.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

FluidSynth is an application for playing MIDI files and a library for MIDI applications. It does not have the hooks for playing Karaoke files. This chapter discusses an extension to FluidSynth which adds appropriate hooks, and then uses these to build a variety of Karaoke systems.

Resources

- FluidSynth home page
- FluidSynth download page
- FluidSynth 1.1 Developer Documentation
- Source Archive fluidsynth Documentation

Files

Source files in this chapter are here .

Players

`fluidsynth` is a command line MIDI player. It runs under ALSA with a command line

```
fluidsynth -a alsa -l <soundfont> <files...>
```

Play MIDI files

The FluidSynth API consists of

- A sequencer created using `new_fluid_player`
- A synthesizer created using `new_fluid_synth`
- An audio player created using `new_fluid_audio_driver` which runs in a separate thread
- A "settings" object which can be used to control many features of the other components, created by `new_fluid_settings` and modified by calls such as `fluid_settings_setstr`

A typical program to play a sequence of MIDI files using ALSA follows. It creates the various objects, sets the audio player to use ALSA and then adds each soundfont and MIDI file to the player. The call to `fluid_player_play` then plays each MIDI file in turn. This program is just a repeat of the program seen in the chapter on FluidSynth MIDI .

```c
#include <fluidsynth.h>
#include <fluid_midi.h>


int main(int argc, char** argv)
{
    int i;
    fluid_settings_t* settings;
    fluid_synth_t* synth;
    fluid_player_t* player;
    fluid_audio_driver_t* adriver;

    settings = new_fluid_settings();
    fluid_settings_setstr(settings,
"audio.driver", "alsa");
    synth = new_fluid_synth(settings);
    player = new_fluid_player(synth);

    adriver = new_fluid_audio_driver(settings,
synth);
    /* process command line arguments */
    for (i = 1; i < argc; i++) {
        if (fluid_is_soundfont(argv[i])) {
        fluid_synth_sfload(synth, argv[1], 1);
        } else {
            fluid_player_add(player, argv[i]);
```

```
        }
    }
    /* play the midi files, if any */
    fluid_player_play(player);
    /* wait for playback termination */
    fluid_player_join(player);
    /* cleanup */
    delete_fluid_audio_driver(adriver);
    delete_fluid_player(player);
    delete_fluid_synth(synth);
    delete_fluid_settings(settings);
    return 0;
}
```

Extending FluidSynth with callbacks

Callbacks are functions registered with an application which are called when certain events occur. In order to build a Karaoke player we need to know

- When a file is loaded so that we can extract all of the lyrics from it for display at the right times.
- When each Meta Lyric or Text event occurs as output from a sequencer, so that we can see what lyric is about to be sung.

The first of these is fairly straightforward: FluidSynth has a function `fluid_player_load` which will load a file. We can change the code to add a suitable callback into that function which will give us access to the loaded MIDI file.

Getting Lyric or Text events out of a sequencer is not so easy, since they are never meant to appear! The MIDI specification allows these event types within a MIDI file, but they are not wire-types so should never be sent from a sequencer to a synthesizer. The Java MIDI API makes them available by an out-of-band call to a Meta event handler. FluidSynth just throws them away.

On the other hand, FluidSynth already has a callback to handle MIDI events sent from the sequencer to the synthesizer. It is the function `fluid_synth_handle_midi_event` and is set by the call `fluid_player_set_playback_callback`. What we need to do is to firstly alter the existing FluidSynth code so that Lyric and Text events are passed through, and then insert a new playback callback that will intercept those events and do something with them while passing on all other events to the default handler. The default handler will ignore any such events anyway, so it does not need to be changed.

I have added one new function to FluidSynth, `fluid_player_set_onload_callback` and added appropriate code to pass on some Meta events. Then it is a matter of writing an onload callback to walk through the MIDI data from the parsed input file, and writing a suitable MIDI event callback to handle the intercepted Meta events while passing the rest through to the default handler.

These changes have been made to give a new source package fluidsynth-1.1.6-karaoke.tar.bz2 . If you just want to work from a patch file, that is fluid.patch . The patch has been submitted to the FluidSynth maintainers.

To build from this package, do the same as you normally would:

```
tar jxf fluidsynth-1.1.6-karaoke.tar.bz2
cd fluidsynth-1.1.6
./configure
make clean
make
```

To get ALSA support, you will need to have installed the `libasound2-dev` package, and similarly for Jack or other packages. You probably won't have many of them installed, so don't run `make install` or you will overwrite the normal `fluidsynth` package which will probably have more features.

The previous program modified to just print out the lyric lines and the lyric events as they occur is `karaoke_player.c`:

```c
#include <fluidsynth.h>
#include <fluid_midi.h>

/**
 * This MIDI event callback filters out the
TEXT and LYRIC events
 * and passes the rest to the default event
handler.
 * Here we just print the text of the event,
more
 * complex handling can be done
 */
int event_callback(void *data,
fluid_midi_event_t *event) {
    fluid_synth_t* synth = (fluid_synth_t*)
data;
    int type =
fluid_midi_event_get_type(event);
    int chan =
fluid_midi_event_get_channel(event);
    if (synth == NULL) printf("Synth is
null\n");
    switch(type) {
    case MIDI_TEXT:
    printf("Callback: Playing text event %s
(length %d)\n",
```

```
            (char *) event->paramptr, event-
>param1);
    return  FLUID_OK;

    case MIDI_LYRIC:
    printf("Callback: Playing lyric event %d
%s\n",
            event->param1, (char *) event-
>paramptr);
    return  FLUID_OK;
    }
    return fluid_synth_handle_midi_event( data,
event);
}

/**
 * This is called whenever new data is loaded,
such as a new file.
 * Here we extract the TEXT and LYRIC events
and just print them
 * to stdout. They could e.g. be saved and
displayed in a GUI
 * as the events are received by the event
callback.
 */
int onload_callback(void *data, fluid_player_t
*player) {
```

```
    printf("Load callback, tracks %d \n",
player->ntracks);
    int n;
    for (n = 0; n < player->ntracks; n++) {
    fluid_track_t *track = player->track[n];
    printf("Track %d\n", n);
    fluid_midi_event_t *event =
fluid_track_first_event(track);
    while (event != NULL) {
        switch (event->type) {
        case MIDI_TEXT:
        case MIDI_LYRIC:
        printf("Loaded event %s\n", (char *)
event->paramptr);
        }
        event = fluid_track_next_event(track);
    }
    }
    return FLUID_OK;
}

int main(int argc, char** argv)
{
    int i;
    fluid_settings_t* settings;
    fluid_synth_t* synth;
    fluid_player_t* player;
```

```
    fluid_audio_driver_t* adriver;
    settings = new_fluid_settings();
    fluid_settings_setstr(settings,
"audio.driver", "alsa");
    fluid_settings_setint(settings,
"synth.polyphony", 64);
    synth = new_fluid_synth(settings);
    player = new_fluid_player(synth);

    /* Set the MIDI event callback to our own
functions rather than the system default */
    fluid_player_set_playback_callback(player,
event_callback, synth);

    /* Add an onload callback so we can get
information from new data before it plays */
    fluid_player_set_onload_callback(player,
onload_callback, NULL);

    adriver = new_fluid_audio_driver(settings,
synth);
    /* process command line arguments */
    for (i = 1; i < argc; i++) {
        if (fluid_is_soundfont(argv[i])) {
        fluid_synth_sfload(synth, argv[1], 1);
        } else {
            fluid_player_add(player, argv[i]);
```

```
        }
    }
    /* play the midi files, if any */
    fluid_player_play(player);
    /* wait for playback termination */
    fluid_player_join(player);
    /* cleanup */
    delete_fluid_audio_driver(adriver);
    delete_fluid_player(player);
    delete_fluid_synth(synth);
    delete_fluid_settings(settings);
    return 0;
}
```

Assuming the new fluidsynth package is in an immediate subdirectory, to compile the program you will need to pick up the local includes and libraries

```
gcc -g -I fluidsynth-1.1.6/include/ -I
fluidsynth-1.1.6/src/midi/ -I fluidsynth-
1.1.6/src/utils/ -c -o karaoke_player.o
karaoke_player.c

gcc karaoke_player.o -Lfluidsynth-
1.1.6/src/.libs -l fluidsynth -o karaoke_player
```

To run the program, you will also need to pick up the local library and the soundfont file:

```
export LD_LIBRARY_PATH=./fluidsynth-
1.1.6/src/.libs/
./karaoke_player
/usr/share/soundfonts/FluidR3_GM.sf2 54154.mid
```

The output for a typical `KAR` file is

```
Load callback, tracks 1
Track 0
Loaded event #
Loaded event 0
Loaded event 0
Loaded event 0
Loaded event 1
Loaded event

...

Callback: Playing lyric event 2 #
Callback: Playing lyric event 2 0
Callback: Playing lyric event 2 0
Callback: Playing lyric event 2 0
Callback: Playing lyric event 2 1
Callback: Playing lyric event 3
```

Displaying and colouring text with Gtk

While there are many ways in which Karaoke text can be displayed, a common pattern is to display two lines of text, the currently playing line and the next one. The current line is progressively highlighted and on completion is replaced by the next line.

In the Java chapter we did that. But the Java libraries have not been polished and are distinctly slow and heavyweight. They also seem to be low on Oracle's development schedule for Java. So here we look at an alternative GUI and make use of the FluidSynth library. The Gtk library is chosen for reasons outlined in an earlier chapter on Gtk.

The first task is to build up an array of lyric lines as the file is loaded. We are asssuming KAR format files with upfront information as to title, etc, prefixed by '@', and newlines prefixed by '\'.

```
struct _lyric_t {
    gchar *lyric;
    long tick;
};
typedef struct _lyric_t lyric_t;

struct _lyric_lines_t {
    char *language;
    char *title;
    char *performer;
    GArray *lines; // array of GString *
};
typedef struct _lyric_lines_t lyric_lines_t;

GArray *lyrics;
lyric_lines_t lyric_lines;

void build_lyric_lines() {
    int n;
    lyric_t *plyric;
    GString *line = g_string_new("");
    GArray *lines =  g_array_sized_new(FALSE,
FALSE, sizeof(GString *), 64);

    lyric_lines.title = NULL;
```

```
    for (n = 0; n < lyrics->len; n++) {
    plyric = g_array_index(lyrics, lyric_t *,
n);
    gchar *lyric = plyric->lyric;
    int tick = plyric->tick;

    if ((strlen(lyric) >= 2) && (lyric[0] ==
'@') && (lyric[1] == 'L')) {
        lyric_lines.language =  lyric + 2;
        continue;
    }

    if ((strlen(lyric) >= 2) && (lyric[0] ==
'@') && (lyric[1] == 'T')) {
        if (lyric_lines.title == NULL) {
        lyric_lines.title = lyric + 2;
        } else {
        lyric_lines.performer = lyric + 2;
        }
        continue;
    }

    if (lyric[0] == '@') {
        // some other stuff like @KMIDI KARAOKE
FILE
        continue;
    }
```

```
    if ((lyric[0] == '/') || (lyric[0] ==
'\\')) {
        // start of a new line
        // add to lines
        g_array_append_val(lines, line);
        line = g_string_new(lyric + 1);
    }  else {
        line = g_string_append(line, lyric);
    }
    }
    lyric_lines.lines = lines;

    printf("Title is %s, performer is %s,
language is %s\n",
       lyric_lines.title,
lyric_lines.performer, lyric_lines.language);
    for (n = 0; n < lines->len; n++) {
    printf("Line is %s\n", g_array_index(lines,
GString *, n)->str);
    }
}
```

This is called from the onload callback

```
int onload_callback(void *data, fluid_player_t
*player) {
    long ticks = 0L;
    lyric_t *plyric;

    printf("Load callback, tracks %d \n",
player->ntracks);
    int n;
    for (n = 0; n < player->ntracks; n++) {
    fluid_track_t *track = player->track[n];
    printf("Track %d\n", n);
    fluid_midi_event_t *event =
fluid_track_first_event(track);
    while (event != NULL) {
        switch (fluid_midi_event_get_type
(event)) {
        case MIDI_TEXT:
        case MIDI_LYRIC:
        /* there's no
fluid_midi_event_get_sysex()
           or fluid_midi_event_get_time() so we
           have to look inside the opaque
struct
        */
        ticks += event->dtime;
        printf("Loaded event %s for time %d\n",
```

```
                event->paramptr,
                ticks);
        plyric = g_new(lyric_t, 1);
        plyric->lyric = g_strdup(event-
>paramptr);
        plyric->tick = ticks;
        g_array_append_val(lyrics, plyric);
        }
        event = fluid_track_next_event(track);
    }
    }

    printf("Saved %d lyric events\n", lyrics-
>len);
    for (n = 0; n < lyrics->len; n++) {
    plyric = g_array_index(lyrics, lyric_t *,
n);
    printf("Saved lyric %s at %d\n", plyric-
>lyric, plyric->tick);
    }

    build_lyric_lines();
}
```

The standard GUI part is to build an interface consisting of two labels, one above the other to hold lines of lyrics. This is just ordinary Gtk.

The final part is to handle lyric or text events from the sequencer. If the event is a '\', then the current text in a label must be replaced with new text, afer a small pause. Otherwise, the text in the label has to be progressively coloured to indicate what is next to be played.

In the earlier chapter on Gtk we discussed using Cairo to draw in pixbufs, and Pango to structure the text. The Gtk Label understands Pango directly, so we just use Pango to format the text and display it in the label. This involves constructing an HTML string with the first part coloured red and the rest in black. This can be set in the label, and there is no need to use Cairo.

The program is `gtkkaraoke_player.c` *Warning: the following program crashes regularly when trying to copy a Pango attribute list in the Gtk code for sizing a label. Debugging shows that the Pango copy function is set to NULL somewhere in Gtk, and shouldn't be. I have no fix as yet and haven't replicated the bug in a simple enough way to log a bug report.*

```
#include <fluidsynth.h>
#include <fluid_midi.h>
#include <string.h>

#include <gtk/gtk.h>

/* GString stuff from
https://developer.gnome.org/glib/2.31/glib-
Strings.html
   Memory alloc from
https://developer.gnome.org/glib/2.30/glib-
Memory-Allocation.html
   Packing demo from
https://developer.gnome.org/gtk-
tutorial/2.90/x386.html
   Thread stuff from
https://developer.gnome.org/gtk-
faq/stable/x481.html
   GArrays from
http://www.gtk.org/api/2.6/glib/glib-
Arrays.html
   Pango attributes from
http://www.ibm.com/developerworks/library/l-u-
pango2/
   Timeouts at
```

```
http://www.gtk.org/tutorial1.2/gtk_tut-17.html
 */

struct _lyric_t {
    gchar *lyric;
    long tick;

};
typedef struct _lyric_t lyric_t;

struct _lyric_lines_t {
    char *language;
    char *title;
    char *performer;
    GArray *lines; // array of GString *
};
typedef struct _lyric_lines_t lyric_lines_t;

GArray *lyrics;

lyric_lines_t lyric_lines;

fluid_synth_t* synth;

GtkWidget *lyric_labels[2];

fluid_player_t* player;
```

```
int current_panel = -1;  // panel showing
current lyric line
int current_line = 0;  // which line is the
current lyric
gchar *current_lyric;   // currently playing
lyric line
GString *front_of_lyric;  // part of lyric to
be coloured red
GString *end_of_lyric;    // part of lyric to
not be coloured

gchar *markup[] = {"<span foreground=\"red\">",
           "</span><span
foreground=\"black\">",
           "</span>"};
gchar *markup_newline[] = {"<span
foreground=\"black\">",
           "</span>"};
GString *marked_up_label;

struct _reset_label_data {
    GtkLabel *label;
    gchar *text;
    PangoAttrList *attrs;
};
```

```
typedef struct _reset_label_data
reset_label_data;

/**
 * redraw a label some time later
 */
gint reset_label_cb(gpointer data) {
    reset_label_data *rdata = (
reset_label_data *) data;

    if (rdata->label == NULL) {
    printf("Label is null, cant set its text
\n");
    return FALSE;
    }

    printf("Resetting label callback to
\"%s\"\n", rdata->text);

    gdk_threads_enter();

    gchar *str;
    str = g_strconcat(markup_newline[0], rdata-
>text, markup_newline[1], NULL);

    PangoAttrList *attrs;
    gchar *text;
```

```
    pango_parse_markup (str, -1,0, &attrs,
&text, NULL, NULL);

    gtk_label_set_text(rdata->label, text);
    gtk_label_set_attributes(rdata->label,
attrs);

    gdk_threads_leave();

    GtkAllocation* alloc = g_new(GtkAllocation,
1);
    gtk_widget_get_allocation((GtkWidget *)
(rdata->label), alloc);
    printf("Set label text to \"%s\"\n",
gtk_label_get_text(rdata->label));
    printf("Label has height %d width %d\n",
alloc->height, alloc->width);
    printf("Set other label text to \"%s\"\n",
       gtk_label_get_text(rdata->label ==
lyric_labels[0] ?
                  lyric_labels[1] :
lyric_labels[0]));
    gtk_widget_get_allocation((GtkWidget *)
(rdata->label  == lyric_labels[0] ?
                  lyric_labels[1] :
lyric_labels[0]), alloc);
    printf("Label has height %d width %d\n",
```

```
alloc->height, alloc->width);

    return FALSE;
}



/**
 * This MIDI event callback filters out the
TEXT and LYRIC events
 * and passes the rest to the default event
handler.
 * Here we colour the text in a Gtk label
 */
int event_callback(void *data,
fluid_midi_event_t *event) {
    fluid_synth_t* synth = (fluid_synth_t*)
data;
    int type =
fluid_midi_event_get_type(event);
    int chan =
fluid_midi_event_get_channel(event);
    if (synth == NULL) printf("Synth is
null\n");
    switch(type) {
    case MIDI_TEXT:
    printf("Callback: Playing text event %s
(length %d)\n",
```

```
            (char *) event->paramptr, event-
>param1);

    if (((char *) event->paramptr)[0] == '\\')
{
        // we've got a new line, change the
label text on the NEXT panel
        int next_panel = current_panel; //
really (current_panel+2)%2
        int next_line = current_line + 2;
        gchar *next_lyric;

        if (current_line + 2 >=
lyric_lines.lines->len) {
        return FLUID_OK;
        }
        current_line += 1;
        current_panel = (current_panel + 1) %
2;

        // set up new line as current line
        char *lyric =  event->paramptr;

        // find the next line from lyric_lines
array
        current_lyric =
g_array_index(lyric_lines.lines, GString *,
```

```
current_line)->str;

        // lyric is in 2 parts: front coloured,
end uncoloured
        front_of_lyric = g_string_new(lyric+1);
// lose \
        end_of_lyric =
g_string_new(current_lyric);
        printf("New line. Setting front to %s
end to \"%s\"\n", lyric+1, current_lyric);

        // update label for next line after
this one
        char *str =
g_array_index(lyric_lines.lines, GString *,
next_line)->str;
        printf("Setting text in label %d to
\"%s\"\n", next_panel, str);

        next_lyric =
g_array_index(lyric_lines.lines, GString *,
next_line)->str;

        gdk_threads_enter();

        // change the label after one second to
avoid visual "jar"
```

```
        reset_label_data *label_data;
        label_data = g_new(reset_label_data,
1);
        label_data->label = (GtkLabel *)
lyric_labels[next_panel];
        label_data->text = next_lyric;
        g_timeout_add(1000, reset_label_cb,
label_data);

        // Dies if you try to flush at this
point!
        // gdk_flush();

        gdk_threads_leave();
    } else {
        // change text colour as chars are
played, using Pango attributes
        char *lyric =  event->paramptr;
        if ((front_of_lyric != NULL) && (lyric
!= NULL)) {
        // add the new lyric to the front of
the existing coloured
        g_string_append(front_of_lyric, lyric);
        char *s = front_of_lyric->str;
        printf("Displaying \"%s\"\n",
current_lyric);
        printf("  Colouring \"%s\"\n", s);
```

```
        printf("  Not colouring \"%s\"\n",
current_lyric + strlen(s));

        // todo: avoid memory leak
        marked_up_label =
g_string_new(markup[0]);
        g_string_append(marked_up_label, s);
        g_string_append(marked_up_label,
markup[1]);
        g_string_append(marked_up_label,
current_lyric + strlen(s));
        g_string_append(marked_up_label,
markup[2]);
        printf("Marked up label \"%s\"\n",
marked_up_label->str);

        /* Example from
http://www.ibm.com/developerworks/library/l-u-
pango2/
         */
        PangoAttrList *attrs;
        gchar *text;
        gdk_threads_enter();
        pango_parse_markup (marked_up_label-
>str, -1,0, &attrs, &text, NULL, NULL);
        printf("Marked up label parsed ok\n");
        gtk_label_set_text((GtkLabel *)
```

```
lyric_labels[current_panel],
                    text);

gtk_label_set_attributes(GTK_LABEL(lyric_labels[c
 attrs);
        // Dies if you try to flush at this
point!
        //gdk_flush();

        gdk_threads_leave();
        }
    }
    return  FLUID_OK;

    case MIDI_LYRIC:
    printf("Callback: Playing lyric event %d
%s\n",
           event->param1, (char *) event-
>paramptr);
    return  FLUID_OK;

    case MIDI_EOT:
    printf("End of track\n");
    exit(0);
    }
    // default handler for all other events
    return fluid_synth_handle_midi_event( data,
```

```
event);
}

/*
 * Build array of lyric lines from the MIDI
file data
 */
void build_lyric_lines() {
    int n;
    lyric_t *plyric;
    GString *line = g_string_new("");
    GArray *lines =  g_array_sized_new(FALSE,
FALSE, sizeof(GString *), 64);

    lyric_lines.title = NULL;

    for (n = 0; n < lyrics->len; n++) {
    plyric = g_array_index(lyrics, lyric_t *,
n);
    gchar *lyric = plyric->lyric;
    int tick = plyric->tick;

    if ((strlen(lyric) >= 2) && (lyric[0] ==
'@') && (lyric[1] == 'L')) {
        lyric_lines.language =  lyric + 2;
        continue;
    }
```

```
    if ((strlen(lyric) >= 2) && (lyric[0] ==
'@') && (lyric[1] == 'T')) {
        if (lyric_lines.title == NULL) {
        lyric_lines.title = lyric + 2;
        } else {
        lyric_lines.performer = lyric + 2;
        }
        continue;
    }

    if (lyric[0] == '@') {
        // some other stuff like @KMIDI KARAOKE
FILE
        continue;
    }

    if ((lyric[0] == '/') || (lyric[0] ==
'\\')) {
        // start of a new line
        // add to lines
        g_array_append_val(lines, line);
        line = g_string_new(lyric + 1);
    }  else {
        line = g_string_append(line, lyric);
    }
    }
```

```
    lyric_lines.lines = lines;

    printf("Title is %s, performer is %s,
language is %s\n",
       lyric_lines.title,
lyric_lines.performer, lyric_lines.language);
    for (n = 0; n < lines->len; n++) {
    printf("Line is %s\n", g_array_index(lines,
GString *, n)->str);
    }

}

/**
 * This is called whenever new data is loaded,
such as a new file.
 * Here we extract the TEXT and LYRIC events
and save them
 * into an array
 */
int onload_callback(void *data, fluid_player_t
*player) {
    long ticks = 0L;
    lyric_t *plyric;

    printf("Load callback, tracks %d \n",
player->ntracks);
```

```
    int n;
    for (n = 0; n < player->ntracks; n++) {
    fluid_track_t *track = player->track[n];
    printf("Track %d\n", n);
    fluid_midi_event_t *event =
fluid_track_first_event(track);
    while (event != NULL) {
        switch (fluid_midi_event_get_type
(event)) {
        case MIDI_TEXT:
        case MIDI_LYRIC:
        /* there's no
fluid_midi_event_get_sysex()
           or fluid_midi_event_get_time() so we
           have to look inside the opaque
struct
        */
        ticks += event->dtime;
        printf("Loaded event %s for time
%ld\n",
               (char *) event->paramptr,
               ticks);
        plyric = g_new(lyric_t, 1);
        plyric->lyric = g_strdup(event-
>paramptr);
        plyric->tick = ticks;
        g_array_append_val(lyrics, plyric);
```

```
        }
        event = fluid_track_next_event(track);
    }
    }



    printf("Saved %d lyric events\n", lyrics-
>len);
    for (n = 0; n < lyrics->len; n++) {
    plyric = g_array_index(lyrics, lyric_t *,
n);
    printf("Saved lyric %s at %ld\n", plyric-
>lyric, plyric->tick);
    }

    build_lyric_lines();

    // stick the first two lines into the
labels so we can see
    // what is coming
    gdk_threads_enter();
    char *str =
g_array_index(lyric_lines.lines, GString *, 1)-
>str;
    gtk_label_set_text((GtkLabel *)
lyric_labels[0], str);
```

```
    str = g_array_index(lyric_lines.lines,
GString *, 2)->str;
    gtk_label_set_text((GtkLabel *)
lyric_labels[1], str);
    // gdk_flush ();

    /* release GTK thread lock */
    gdk_threads_leave();

    return FLUID_OK;
}

/* Called when the windows are realized
 */
static void realize_cb (GtkWidget *widget,
gpointer data) {
    /* now we can play the midi files, if any
*/
    fluid_player_play(player);
}

static gboolean delete_event( GtkWidget
*widget,
                              GdkEvent  *event,
                              gpointer   data )
{
    /* If you return FALSE in the "delete-
```

```
event" signal handler,
     * GTK will emit the "destroy" signal.
Returning TRUE means
     * you don't want the window to be
destroyed.
     * This is useful for popping up 'are you
sure you want to quit?'
     * type dialogs. */

    g_print ("delete event occurred\n");

    /* Change TRUE to FALSE and the main window
will be destroyed with
     * a "delete-event". */

    return TRUE;
}

/* Another callback */
static void destroy( GtkWidget *widget,
                     gpointer   data )
{
    gtk_main_quit ();
}

int main(int argc, char** argv)
{
```

```
    /* set up the fluidsynth stuff */
    int i;
    fluid_settings_t* settings;

    fluid_audio_driver_t* adriver;
    settings = new_fluid_settings();
    fluid_settings_setstr(settings,
"audio.driver", "alsa");
    fluid_settings_setint(settings,
"synth.polyphony", 64);
    fluid_settings_setint(settings,
"synth.reverb.active", FALSE);
    fluid_settings_setint(settings,
"synth.sample-rate", 22050);
    synth = new_fluid_synth(settings);
    player = new_fluid_player(synth);

    lyrics = g_array_sized_new(FALSE, FALSE,
sizeof(lyric_t *), 1024);

    /* Set the MIDI event callback to our own
functions rather than the system default */
    fluid_player_set_playback_callback(player,
event_callback, synth);

    /* Add an onload callback so we can get
```

```
information from new data before it plays */
    fluid_player_set_onload_callback(player,
onload_callback, NULL);

    adriver = new_fluid_audio_driver(settings,
synth);
    /* process command line arguments */
    for (i = 1; i < argc; i++) {
        if (fluid_is_soundfont(argv[i])) {
        fluid_synth_sfload(synth, argv[1], 1);
        } else {
            fluid_player_add(player, argv[i]);
        }
    }

    // Gtk stuff now

   /* GtkWidget is the storage type for widgets
*/
    GtkWidget *window;
    GtkWidget *button;
    GtkWidget *lyrics_box;



    /* This is called in all GTK applications.
Arguments are parsed
     * from the command line and are returned
```

```
to the application. */
    gtk_init (&argc, &argv);

    /* create a new window */
    window = gtk_window_new
(GTK_WINDOW_TOPLEVEL);

    /* When the window is given the "delete-
event" signal (this is given
     * by the window manager, usually by the
"close" option, or on the
     * titlebar), we ask it to call the
delete_event () function
     * as defined above. The data passed to the
callback
     * function is NULL and is ignored in the
callback function. */
    g_signal_connect (window, "delete-event",
              G_CALLBACK (delete_event), NULL);

    /* Here we connect the "destroy" event to a
signal handler.
     * This event occurs when we call
gtk_widget_destroy() on the window,
     * or if we return FALSE in the "delete-
event" callback. */
    g_signal_connect (window, "destroy",
```

```
                G_CALLBACK (destroy), NULL);

    g_signal_connect (window, "realize",
G_CALLBACK (realize_cb), NULL);

    /* Sets the border width of the window. */
    gtk_container_set_border_width
(GTK_CONTAINER (window), 10);

    // Gtk 3.0 deprecates gtk_vbox_new in
favour of gtk_grid
    // but that isn't in Gtk 2.0, so we ignore
warnings for now
    lyrics_box = gtk_vbox_new(TRUE, 1);
    gtk_widget_show(lyrics_box);

    char *str = "  ";
    lyric_labels[0] = gtk_label_new(str);
    lyric_labels[1] = gtk_label_new(str);

    gtk_widget_show (lyric_labels[0]);
    gtk_widget_show (lyric_labels[1]);


    gtk_box_pack_start (GTK_BOX (lyrics_box),
lyric_labels[0], TRUE, TRUE, 0);
    gtk_box_pack_start (GTK_BOX (lyrics_box),
```

```
lyric_labels[1], TRUE, TRUE, 0);

    /* This packs the button into the window (a
gtk container). */
    gtk_container_add (GTK_CONTAINER (window),
lyrics_box);

    /* and the window */
    gtk_widget_show (window);

    /* All GTK applications must have a
gtk_main(). Control ends here
     * and waits for an event to occur (like a
key press or
     * mouse event). */
    gtk_main ();

    /* wait for playback termination */
    fluid_player_join(player);
    /* cleanup */
    delete_fluid_audio_driver(adriver);
    delete_fluid_player(player);
    delete_fluid_synth(synth);
    delete_fluid_settings(settings);
    return 0;
}
```

When run it looks like

Playing a background video with Gtk

The chapter on Gtk showed how to play a background video with images (using Pixbufs), text (using Cairo) and coloured text (using Pango). We can extend that by adding in the dynamic text display for playing Karaoke.

We capture each lyric line in a structure which keeps the whole of the line, the part that has been sung already, the Pango markup for the line and the Pango attributes:

```
typedef struct _coloured_line_t {
    gchar *line;
    gchar *front_of_line;
    gchar *marked_up_line;
    PangoAttrList *attrs;
} coloured_line_t;
```

This is updated each time a MIDI Lyric event occurs, in a thread listening to the FluidSynth sequencer.

A separate thread plays the video, and on each frame overlays the frame image with the current and next lyric. This is set into a `GdkImage` for display by Gtk.

The program is `gtkkaraoke_player_video_pango.c`

```
#include <fluidsynth.h>
#include <fluid_midi.h>
#include <string.h>

#include <gtk/gtk.h>

#include <libavcodec/avcodec.h>
#include <libavformat/avformat.h>
#include <libswscale/swscale.h>

// saving as pixbufs leaks memory
//#define USE_PIXBUF

/* run by
   gtkkaraoke_player_video_pango
/usr/share/sounds/sf2/FluidR3_GM.sf2
/home/newmarch/Music/karaoke/sonken/songs/54154.k

*/

/*
 * APIs:
 * GString:
https://developer.gnome.org/glib/2.28/glib-
Strings.html
 * Pango text attributes:
```

```
https://developer.gnome.org/pango/stable/pango-Text-Attributes.html#pango-parse-markup
 * Pango layout: http://www.gtk.org/api/2.6/pango/pango-Layout-Objects.html
 * Cairo rendering: https://developer.gnome.org/pango/stable/pango-Cairo-Rendering.html#pango-cairo-create-layout
 * Cairo surface_t: http://cairographics.org/manual/cairo-cairo-surface-t.html
 * GTK+ 3 Reference Manual: https://developer.gnome.org/gtk3/3.0/
 * Gdk Pixbufs: https://developer.gnome.org/gdk/stable/gdk-Pixbufs.html
 */

struct _lyric_t {
    gchar *lyric;
    long tick;

};
typedef struct _lyric_t lyric_t;

struct _lyric_lines_t {
    char *language;
```

```
    char *title;
    char *performer;
    GArray *lines; // array of GString *
};
typedef struct _lyric_lines_t lyric_lines_t;

GArray *lyrics;

lyric_lines_t lyric_lines;

typedef struct _coloured_line_t {
    gchar *line;
    gchar *front_of_line;
    gchar *marked_up_line;
    PangoAttrList *attrs;
#ifdef USE_PIXBUF
    GdkPixbuf *pixbuf;
#endif
} coloured_line_t;

coloured_line_t coloured_lines[2];

fluid_synth_t* synth;

GtkWidget *image;
#if GTK_MAJOR_VERSION == 2
GdkPixmap *dbuf_pixmap;
```

```
#endif

int height_lyric_pixbufs[] = {300, 400}; //
vertical offset of lyric in video

fluid_player_t* player;

int current_panel = 1;  // panel showing
current lyric line
int current_line = 0;  // which line is the
current lyric
gchar *current_lyric;   // currently playing
lyric line
GString *front_of_lyric;  // part of lyric to
be coloured red
//GString *end_of_lyric;    // part of lyrci to
not be coloured



// Colours seem to get mixed up when putting a
pixbuf onto a pixbuf
#ifdef USE_PIXBUF
#define RED blue
#else
#define RED red
#endif
```

```
gchar *markup[] = {"<span font=\"28\"
foreground=\"RED\">",
           "</span><span font=\"28\"
foreground=\"white\">",
           "</span>"};
gchar *markup_newline[] = {"<span
foreground=\"black\">",
               "</span>"};
GString *marked_up_label;

/* FFMpeg vbls */
AVFormatContext *pFormatCtx = NULL;
AVCodecContext *pCodecCtx = NULL;
int videoStream;
struct SwsContext *sws_ctx = NULL;
AVCodec *pCodec = NULL;

void markup_line(coloured_line_t *line) {
    GString *str =  g_string_new(markup[0]);
    g_string_append(str, line->front_of_line);
    g_string_append(str, markup[1]);
    g_string_append(str, line->line +
strlen(line->front_of_line));
    g_string_append(str, markup[2]);
    printf("Marked up label \"%s\"\n", str-
>str);
```

```
    line->marked_up_line = str->str;
    // we have to free line->marked_up_line

    pango_parse_markup(str->str, -1,0, &(line-
>attrs), NULL, NULL, NULL);
    g_string_free(str, FALSE);
}

#ifdef USE_PIXBUF
void update_line_pixbuf(coloured_line_t *line)
{
    //return;
    cairo_surface_t *surface;
    cairo_t *cr;

    int lyric_width = 480;
    int lyric_height = 60;
    surface = cairo_image_surface_create
(CAIRO_FORMAT_ARGB32,

lyric_width, lyric_height);
    cr = cairo_create (surface);

    PangoLayout *layout;
    PangoFontDescription *desc;
```

```
    // draw the attributed text
    layout = pango_cairo_create_layout (cr);
    pango_layout_set_text (layout, line->line,
-1);
    pango_layout_set_attributes(layout, line-
>attrs);

    // centre the image in the surface
    int width, height;
    pango_layout_get_pixel_size(layout,
                &width,
                &height);
    cairo_move_to(cr, (lyric_width-width)/2,
0);

    pango_cairo_update_layout (cr, layout);
    pango_cairo_show_layout (cr, layout);

    // pull the pixbuf out of the surface
    unsigned char *data =
cairo_image_surface_get_data(surface);
    width =
cairo_image_surface_get_width(surface);
    height =
cairo_image_surface_get_height(surface);
    int stride =
cairo_image_surface_get_stride(surface);
```

```
    printf("Text surface width %d height %d
stride %d\n", width, height, stride);

    GdkPixbuf *old_pixbuf = line->pixbuf;
    line->pixbuf =
gdk_pixbuf_new_from_data(data,
GDK_COLORSPACE_RGB, 1, 8, width, height,
stride, NULL, NULL);
    cairo_surface_destroy(surface);
    g_object_unref(old_pixbuf);
}
#endif

/**
 * This MIDI event callback filters out the
TEXT and LYRIC events
 * and passes the rest to the default event
handler.
  */
int event_callback(void *data,
fluid_midi_event_t *event) {
    fluid_synth_t* synth = (fluid_synth_t*)
data;
    int type =
fluid_midi_event_get_type(event);
    int chan =
fluid_midi_event_get_channel(event);
```

```
    if (synth == NULL) printf("Synth is
null\n");

    //return 0;

    switch(type) {
    case MIDI_TEXT:
    printf("Callback: Playing text event %s
(length %d)\n",
           (char *) event->paramptr, (int)
event->param1);

    if (((char *) event->paramptr)[0] == '\\')
{
        int next_panel = current_panel; //
really (current_panel+2)%2
        int next_line = current_line + 2;
        gchar *next_lyric;

        if (current_line + 2 >=
lyric_lines.lines->len) {
        return FLUID_OK;
        }
        current_line += 1;
        current_panel = (current_panel + 1) %
2;
```

```
        // set up new line as current line
        char *lyric =  event->paramptr;
        current_lyric =
g_array_index(lyric_lines.lines, GString *,
current_line)->str;
        front_of_lyric = g_string_new(lyric+1);
// lose \
        printf("New line. Setting front to %s
end to \"%s\"\n", lyric+1, current_lyric);

        coloured_lines[current_panel].line =
current_lyric;

coloured_lines[current_panel].front_of_line =
lyric+1;

markup_line(coloured_lines+current_panel);
#ifdef USE_PIXBUF

update_line_pixbuf(coloured_lines+current_panel);

#endif
        // update label for next line after
this one
        next_lyric =
g_array_index(lyric_lines.lines, GString *,
next_line)->str;
```

```
        marked_up_label =
g_string_new(markup_newline[0]);

        g_string_append(marked_up_label,
next_lyric);
        g_string_append(marked_up_label,
markup_newline[1]);
        PangoAttrList *attrs;
        gchar *text;
        pango_parse_markup (marked_up_label-
>str, -1,0, &attrs, &text, NULL, NULL);

        coloured_lines[next_panel].line =
next_lyric;

coloured_lines[next_panel].front_of_line = "";
        markup_line(coloured_lines+next_panel);
#ifdef USE_PIXBUF

update_line_pixbuf(coloured_lines+next_panel);
#endif
    } else {
        // change text colour as chars are
played
        char *lyric =  event->paramptr;
        if ((front_of_lyric != NULL) && (lyric
```

```
!= NULL)) {
        g_string_append(front_of_lyric, lyric);
        char *s = front_of_lyric->str;

coloured_lines[current_panel].front_of_line =
s;

markup_line(coloured_lines+current_panel);
#ifdef USE_PIXBUF

update_line_pixbuf(coloured_lines+current_panel);

#endif
        }
    }
    return  FLUID_OK;

    case MIDI_LYRIC:
    printf("Callback: Playing lyric event %d
%s\n", (int) event->param1, (char *) event-
>paramptr);
    return  FLUID_OK;

    case MIDI_EOT:
    printf("End of track\n");
    exit(0);
    }
```

```
    return fluid_synth_handle_midi_event( data,
event);
}

void build_lyric_lines() {
    int n;
    lyric_t *plyric;
    GString *line = g_string_new("");
    GArray *lines =  g_array_sized_new(FALSE,
FALSE, sizeof(GString *), 64);

    lyric_lines.title = NULL;

    for (n = 0; n < lyrics->len; n++) {
    plyric = g_array_index(lyrics, lyric_t *,
n);
    gchar *lyric = plyric->lyric;
    int tick = plyric->tick;

    if ((strlen(lyric) >= 2) && (lyric[0] ==
'@') && (lyric[1] == 'L')) {
        lyric_lines.language =  lyric + 2;
        continue;
    }

    if ((strlen(lyric) >= 2) && (lyric[0] ==
'@') && (lyric[1] == 'T')) {
```

```
        if (lyric_lines.title == NULL) {
        lyric_lines.title = lyric + 2;
        } else {
        lyric_lines.performer = lyric + 2;
        }
        continue;
    }

    if (lyric[0] == '@') {
        // some other stuff like @KMIDI KARAOKE
FILE
        continue;
    }

    if ((lyric[0] == '/') || (lyric[0] ==
'\\')) {
        // start of a new line
        // add to lines
        g_array_append_val(lines, line);
        line = g_string_new(lyric + 1);
    }  else {
        line = g_string_append(line, lyric);
    }
    }
    lyric_lines.lines = lines;

    printf("Title is %s, performer is %s,
```

```
language is %s\n",
       lyric_lines.title,
lyric_lines.performer, lyric_lines.language);
    for (n = 0; n < lines->len; n++) {
    printf("Line is %s\n", g_array_index(lines,
GString *, n)->str);
    }

}



/**
 * This is called whenever new data is loaded,
such as a new file.
 * Here we extract the TEXT and LYRIC events
and just print them
 * to stdout. They could e.g. be saved and
displayed in a GUI
 * as the events are received by the event
callback.
 */
int onload_callback(void *data, fluid_player_t
*player) {
    long ticks = 0L;
    lyric_t *plyric;

    printf("Load callback, tracks %d \n",
```

```
player->ntracks);
    int n;
    for (n = 0; n < player->ntracks; n++) {
    fluid_track_t *track = player->track[n];
    printf("Track %d\n", n);
    fluid_midi_event_t *event =
fluid_track_first_event(track);
    while (event != NULL) {
        switch (fluid_midi_event_get_type
(event)) {
        case MIDI_TEXT:
        case MIDI_LYRIC:
        /* there's no
fluid_midi_event_get_sysex()
           or fluid_midi_event_get_time() so we
           have to look inside the opaque
struct
        */
        ticks += event->dtime;
        printf("Loaded event %s for time
%ld\n",
               (char *) event->paramptr,
               ticks);
        plyric = g_new(lyric_t, 1);
        plyric->lyric = g_strdup(event-
>paramptr);
        plyric->tick = ticks;
```

```
        g_array_append_val(lyrics, plyric);
        }
        event = fluid_track_next_event(track);
    }
    }

    printf("Saved %d lyric events\n", lyrics-
>len);
    for (n = 0; n < lyrics->len; n++) {
    plyric = g_array_index(lyrics, lyric_t *,
n);
    printf("Saved lyric %s at %ld\n", plyric-
>lyric, plyric->tick);
    }

    build_lyric_lines();

    return FLUID_OK;
}

static void overlay_lyric(cairo_t *cr,
              coloured_line_t *line,
              int ht) {
    PangoLayout *layout;
    int height, width;

    if (line->line == NULL) {
```

```
    return;
    }

    layout = pango_cairo_create_layout (cr);
    pango_layout_set_text (layout, line->line,
-1);
    pango_layout_set_attributes(layout, line-
>attrs);
    pango_layout_get_pixel_size(layout,
                &width,
                &height);
    cairo_move_to(cr, (720-width)/2, ht);

    pango_cairo_update_layout (cr, layout);
    pango_cairo_show_layout (cr, layout);

    g_object_unref(layout);
}

static void pixmap_destroy_notify(guchar
*pixels,
                  gpointer data) {
    printf("Ddestroy pixmap\n");
}

static void *play_background(void *args) {
    /* based on code from
```

```
http://www.cs.dartmouth.edu/~xy/cs23/gtk.html

http://cdry.wordpress.com/2009/09/09/using-
custom-io-callbacks-with-ffmpeg/
    */

    int i;
    AVPacket packet;
    int frameFinished;
    AVFrame *pFrame = NULL;

    int oldSize;
    char *oldData;
    int bytesDecoded;
    GdkPixbuf *pixbuf;
    AVFrame *picture_RGB;
    char *buffer;

#if GTK_MAJOR_VERSION == 2
    GdkPixmap *pixmap;
    GdkBitmap *mask;
#endif

    pFrame=avcodec_alloc_frame();

    i=0;
```

```
    picture_RGB = avcodec_alloc_frame();
    buffer = malloc
(avpicture_get_size(PIX_FMT_RGB24, 720, 576));
    avpicture_fill((AVPicture *)picture_RGB,
buffer, PIX_FMT_RGB24, 720, 576);

    while(av_read_frame(pFormatCtx,
&packet)>=0) {
    if(packet.stream_index==videoStream) {
        //printf("Frame %d\n", i++);
        usleep(33670);  // 29.7 frames per
second
        // Decode video frame
        avcodec_decode_video2(pCodecCtx,
pFrame, &frameFinished,
                  &packet);
        int width = pCodecCtx->width;
        int height = pCodecCtx->height;

        sws_ctx = sws_getContext(pCodecCtx-
>width, pCodecCtx->height, pCodecCtx->pix_fmt,
pCodecCtx->width, pCodecCtx->height,
PIX_FMT_RGB24, SWS_BICUBIC, NULL, NULL, NULL);

        if (frameFinished) {
        printf("Frame %d\n", i++);
```

```
        sws_scale(sws_ctx,  (uint8_t const *
const *) pFrame->data, pFrame->linesize, 0,
pCodecCtx->height, picture_RGB->data,
picture_RGB->linesize);



        pixbuf =
gdk_pixbuf_new_from_data(picture_RGB->data[0],
GDK_COLORSPACE_RGB, 0, 8, 720, 480,
picture_RGB->linesize[0],
pixmap_destroy_notify, NULL);



        /* start GTK thread lock for drawing */
        gdk_threads_enter();

#define SHOW_LYRIC
#ifdef SHOW_LYRIC
                // Create the destination
surface
                cairo_surface_t *surface =
cairo_image_surface_create
(CAIRO_FORMAT_ARGB32,

width, height);
                cairo_t *cr =
cairo_create(surface);
```

```
                // draw the background image
                gdk_cairo_set_source_pixbuf(cr,
pixbuf, 0, 0);
                cairo_paint (cr);

#ifdef USE_PIXBUF
        // draw the lyric
        GdkPixbuf *lyric_pixbuf =
coloured_lines[current_panel].pixbuf;
        if (lyric_pixbuf != NULL) {
            int width =
gdk_pixbuf_get_width(lyric_pixbuf);
            gdk_cairo_set_source_pixbuf(cr,
                        lyric_pixbuf,
                        (720-width)/2,

height_lyric_pixbufs[current_panel]);
            cairo_paint (cr);
        }

        int next_panel = (current_panel+1) % 2;
        lyric_pixbuf =
coloured_lines[next_panel].pixbuf;
        if (lyric_pixbuf != NULL) {
            int width =
gdk_pixbuf_get_width(lyric_pixbuf);
```

```
            gdk_cairo_set_source_pixbuf(cr,
                        lyric_pixbuf,
                        (720-width)/2,

height_lyric_pixbufs[next_panel]);
            cairo_paint (cr);
        }
#else

        overlay_lyric(cr,
                  coloured_lines+current_panel,

height_lyric_pixbufs[current_panel]);

        int next_panel = (current_panel+1) % 2;
        overlay_lyric(cr,
                  coloured_lines+next_panel,

height_lyric_pixbufs[next_panel]);
#endif
        pixbuf =
gdk_pixbuf_get_from_surface(surface,
                            0,
                            0,
                            width,
                            height);
```

```
        gtk_image_set_from_pixbuf((GtkImage*)
image, pixbuf);

        g_object_unref(pixbuf);        /*
reclaim memory */
        //g_object_unref(layout);
        cairo_surface_destroy(surface);
        cairo_destroy(cr);
#else
    gtk_image_set_from_pixbuf((GtkImage*)
image, pixbuf);
#endif /* SHOW_LYRIC */

        /* release GTK thread lock */
        gdk_threads_leave();
        }
    }
    av_free_packet(&packet);
    }
    sws_freeContext(sws_ctx);

    printf("Video over!\n");
    exit(0);
}

static void *play_midi(void *args) {
    fluid_player_play(player);
```

```
    printf("Audio finished\n");
    //exit(0);
}


/* Called when the windows are realized
 */
static void realize_cb (GtkWidget *widget,
gpointer data) {
    /* start the video playing in its own
thread */
    pthread_t tid;
    pthread_create(&tid, NULL, play_background,
NULL);

    /* start the MIDI file playing in its own
thread */
    pthread_t tid_midi;
    pthread_create(&tid_midi, NULL, play_midi,
NULL);
}

static gboolean delete_event( GtkWidget
*widget,
                              GdkEvent  *event,
                              gpointer   data )
```

```
{
    /* If you return FALSE in the "delete-
event" signal handler,
     * GTK will emit the "destroy" signal.
Returning TRUE means
     * you don't want the window to be
destroyed.
     * This is useful for popping up 'are you
sure you want to quit?'
     * type dialogs. */

    g_print ("delete event occurred\n");

    /* Change TRUE to FALSE and the main window
will be destroyed with
     * a "delete-event". */

    return TRUE;
}

/* Another callback */
static void destroy( GtkWidget *widget,
                     gpointer   data )
{
    gtk_main_quit ();
}
```

```
int main(int argc, char** argv)
{
    XInitThreads();

    int i;

    fluid_settings_t* settings;

    fluid_audio_driver_t* adriver;
    settings = new_fluid_settings();
    fluid_settings_setstr(settings,
"audio.driver", "alsa");
    //fluid_settings_setint(settings,
"lash.enable", 0);
    fluid_settings_setint(settings,
"synth.polyphony", 64);
    fluid_settings_setint(settings,
"synth.reverb.active", FALSE);
    fluid_settings_setint(settings,
"synth.sample-rate", 22050);
    synth = new_fluid_synth(settings);
    player = new_fluid_player(synth);

    lyrics = g_array_sized_new(FALSE, FALSE,
sizeof(lyric_t *), 1024);

    /* Set the MIDI event callback to our own
```

```
functions rather than the system default */
    fluid_player_set_playback_callback(player,
event_callback, synth);

    /* Add an onload callback so we can get
information from new data before it plays */
    fluid_player_set_onload_callback(player,
onload_callback, NULL);

    adriver = new_fluid_audio_driver(settings,
synth);

    /* process command line arguments */
    for (i = 1; i < argc; i++) {
        if (fluid_is_soundfont(argv[i])) {
            fluid_synth_sfload(synth, argv[1],
1);
        } else {
            fluid_player_add(player, argv[i]);
        }
    }

    /* FFMpeg stuff */

    AVFrame *pFrame = NULL;
    AVPacket packet;
```

```
    AVDictionary *optionsDict = NULL;

    av_register_all();

    if(avformat_open_input(&pFormatCtx,
"short.mpg", NULL, NULL)!=0) {
    printf("Couldn't open video file\n");
    return -1; // Couldn't open file
    }

    // Retrieve stream information
    if(avformat_find_stream_info(pFormatCtx,
NULL)<0) {
    printf("Couldn't find stream
information\n");
    return -1; // Couldn't find stream
information
    }

    // Dump information about file onto
standard error
    av_dump_format(pFormatCtx, 0, argv[1], 0);

    // Find the first video stream
    videoStream=-1;
    for(i=0; i<pFormatCtx->nb_streams; i++)
    if(pFormatCtx->streams[i]->codec-
```

```
>codec_type==AVMEDIA_TYPE_VIDEO) {
        videoStream=i;
        break;
    }
    if(videoStream==-1)
    return -1; // Didn't find a video stream

    for(i=0; i<pFormatCtx->nb_streams; i++)
    if(pFormatCtx->streams[i]->codec-
>codec_type==AVMEDIA_TYPE_AUDIO) {
        printf("Found an audio stream too\n");
        break;
    }

    // Get a pointer to the codec context for
the video stream
    pCodecCtx=pFormatCtx->streams[videoStream]-
>codec;

    // Find the decoder for the video stream
    pCodec=avcodec_find_decoder(pCodecCtx-
>codec_id);
    if(pCodec==NULL) {
    fprintf(stderr, "Unsupported codec!\n");
    return -1; // Codec not found
    }
```

```
    // Open codec
    if(avcodec_open2(pCodecCtx, pCodec,
&optionsDict)<0) {
    printf("Could not open codec\n");
    return -1; // Could not open codec
    }

    sws_ctx =
    sws_getContext
    (
     pCodecCtx->width,
     pCodecCtx->height,
     pCodecCtx->pix_fmt,
     pCodecCtx->width,
     pCodecCtx->height,
     PIX_FMT_YUV420P,
     SWS_BILINEAR,
     NULL,
     NULL,
     NULL
     );

    /* GTK stuff now */

    /* GtkWidget is the storage type for
widgets */
    GtkWidget *window;
```

```
    GtkWidget *button;
    GtkWidget *lyrics_box;


    /* This is called in all GTK applications.
Arguments are parsed
     * from the command line and are returned
to the application. */
    gtk_init (&argc, &argv);

    /* create a new window */
    window = gtk_window_new
(GTK_WINDOW_TOPLEVEL);

    /* When the window is given the "delete-
event" signal (this is given
     * by the window manager, usually by the
"close" option, or on the
     * titlebar), we ask it to call the
delete_event () function
     * as defined above. The data passed to the
callback
     * function is NULL and is ignored in the
callback function. */
    g_signal_connect (window, "delete-event",
              G_CALLBACK (delete_event), NULL);
```

```
    /* Here we connect the "destroy" event to a
signal handler.
     * This event occurs when we call
gtk_widget_destroy() on the window,
     * or if we return FALSE in the "delete-
event" callback. */
    g_signal_connect (window, "destroy",
              G_CALLBACK (destroy), NULL);

    g_signal_connect (window, "realize",
G_CALLBACK (realize_cb), NULL);

    /* Sets the border width of the window. */
    gtk_container_set_border_width
(GTK_CONTAINER (window), 10);

    lyrics_box = gtk_vbox_new(TRUE, 1);
    gtk_widget_show(lyrics_box);

    /*
    char *str = "     ";
    lyric_labels[0] = gtk_label_new(str);
    str =  "World";
    lyric_labels[1] = gtk_label_new(str);
    */

    image = gtk_image_new();
```

```
    //image_drawable = gtk_drawing_area_new();
    //gtk_widget_set_size_request (canvas, 720,
480);
    //gtk_drawing_area_size((GtkDrawingArea *)
image_drawable, 720, 480);

    //gtk_widget_show (lyric_labels[0]);
    //gtk_widget_show (lyric_labels[1]);

    gtk_widget_show (image);

    //gtk_box_pack_start (GTK_BOX (lyrics_box),
lyric_labels[0], TRUE, TRUE, 0);
    //gtk_box_pack_start (GTK_BOX (lyrics_box),
lyric_labels[1], TRUE, TRUE, 0);
    gtk_box_pack_start (GTK_BOX (lyrics_box),
image, TRUE, TRUE, 0);
    //gtk_box_pack_start (GTK_BOX (lyrics_box),
canvas, TRUE, TRUE, 0);
    //gtk_box_pack_start (GTK_BOX (lyrics_box),
image_drawable, TRUE, TRUE, 0);

    /* This packs the button into the window (a
gtk container). */
    gtk_container_add (GTK_CONTAINER (window),
lyrics_box);
```

```
    /* and the window */
    gtk_widget_show (window);

    /* All GTK applications must have a
gtk_main(). Control ends here
     * and waits for an event to occur (like a
key press or
     * mouse event). */
    gtk_main ();

    return 0;

    /* wait for playback termination */
    fluid_player_join(player);
    /* cleanup */
    delete_fluid_audio_driver(adriver);
    delete_fluid_player(player);
    delete_fluid_synth(synth);
    delete_fluid_settings(settings);

    return 0;
}
```

The application looks like

It's been a long
cold lonely winter

Conclusion

By extending FluidSynth it an be made into a Karaoke player in various ways. It is quite heavy in CPU usage though. On my laptop, the final version runs at about 100% CPU.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Donate

Or Flattr me:

TiMidity is designed as a standalone application with a particular kind of extensibility. Out of the box it can sort-of play Karaoke but not well. This chapter looks at how to work with TiMidity to build a Karaoke system.

Files

Source files in this chapter are here .

Introduction

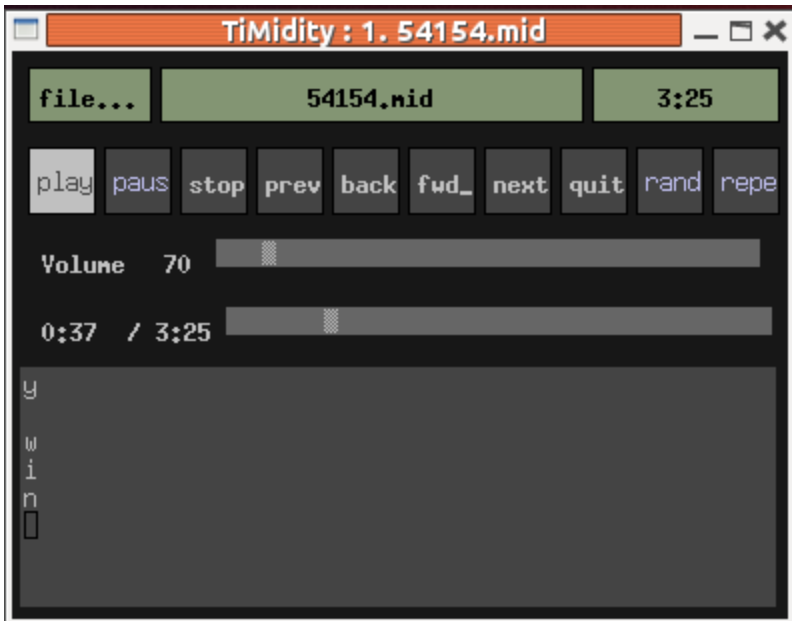
TiMidity is a MIDI player, not a Karaoke player. By default it just plays the MIDI music, with the lyrics printed out:

```
$timidity ../54154.mid
Requested buffer size 32768, fragment size 8192
ALSA pcm 'default' set buffer size 32768,
period size 8192 bytes
Playing ../54154.mid
MIDI file: ../54154.mid
Format: 1  Tracks: 1  Divisions: 30
No instrument mapped to tone bank 0, program 92
- this instrument will not be heard
#0001
@@00@12
@Here Comes The Sun
@
@@Beatles
Here comes the sun
doo doo doo doo
Here comes the sun
I said it's alright
Little
darling
```

But it has a number of alternative interfaces which give different displays. If you run `timidity` with the `-h` (help) option, it will show a screen including something like

```
Available interfaces (-i, --interface option):
  -in           ncurses interface
  -ie           Emacs interface (invoked from
`M-x timidity')
  -ia           XAW interface
  -id           dumb interface
  -ir           remote interface
  -iA           ALSA sequencer interface
```

The default interface is "dumb", but if you run with, say, the XAW interface you get a display like



There is, however, one unfortunate effect: the lyrics are displayed *before* they are due to be played! To get the lyrics played just as they should be sung, you need to turn on the `--`

`trace` option. From the man page, "Toggles trace mode. In trace mode, TiMidity++ attempts to display its current state in real time." You may find the link between documentation and behaviour a lttle less than obvious...

```
timidity --trace ../54154.mid
```

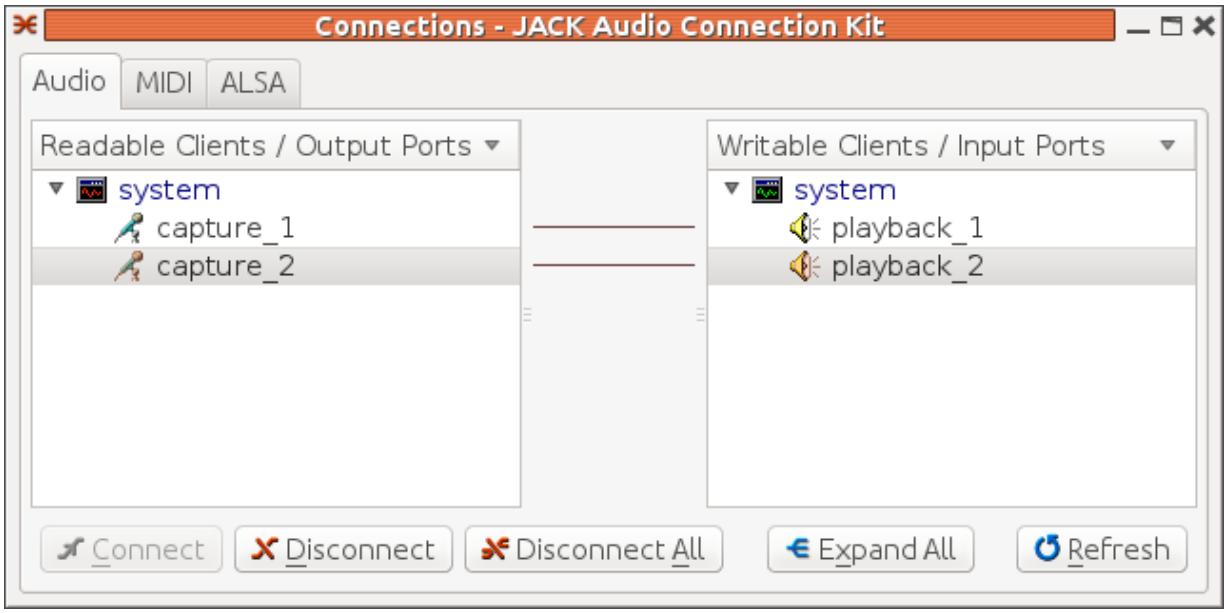
This now works fine for MIDI files, the lyrics are displayed when they should be sung. But it doesn't display the lyrics for `.KAR` files. For that you need the `--trace-text-meta` option:

```
timidity --trace --trace-text-meta ../54154.kar
```

So by this stage, TiMidity will display the lyrics on the screen in realtime for Karaoke files (and MIDI files with LYRIC events). To have our own control over this display, we need to build our own TiMidity interface.

TiMidity and Jack

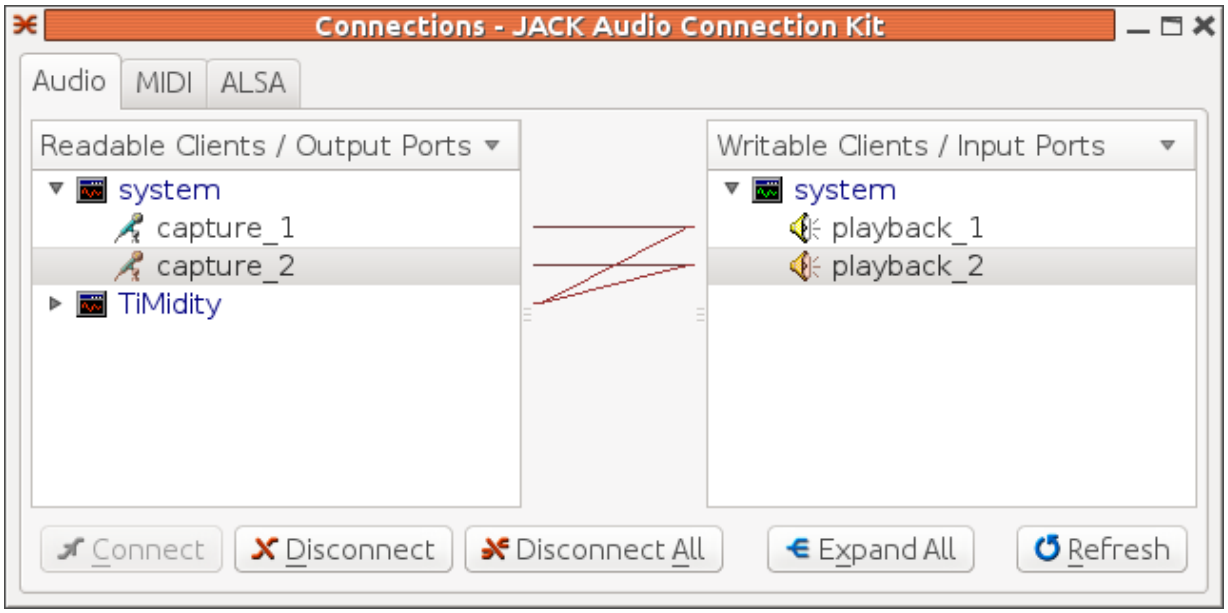
In the MIDI chapter on User space tools we discussed playing MIDI files using Jack. Jack is designed to link audio sources and sinks in arbitrary configurations. By running `qjackctl` you can link, for example, microphone outputs to speaker inputs. This is done by dragging "capture_1" to "playback_1" etc and appears like



If TiMidity is then run with Jack output, you get instant Karaoke. You can also see the lyrics played in real time using the `--trace` option:

```
timidity -Oj --trace 54154.mid
```

The connections are shown in `qjackctl` as



The lyric display is klunky, and will be improved later.

TiMidity interface

You will need to have the TiMidity source downloaded from SourceForge TiMidity++ .

In the earlier chapter on MIDI and TiMidity we discussed two alternative ways of building applications using TiMidity:

- Build a front-end with TiMidity as a library back-end
- Use standard TiMidity with a custom-built interface as back-end to TiMidity

Both options are possible here, with one wrinkle: if we want to capture MIDI events then we have to do so as a back-end to TiMidity, which requires that we build a TiMidity interface.

To recap on this, the different interface files for TiMidity are stored in the directory `interface` and include files such as `dumb_c.c` for the dumb interface. They all revolve around a data structure `ControlMode` defined in `timidity/controls.h` :

```
typedef struct {
  char *id_name, id_character;
  char *id_short_name;
  int verbosity, trace_playing, opened;

  int32 flags;

  int  (*open)(int using_stdin, int
using_stdout);
  void (*close)(void);
  int (*pass_playing_list)(int number_of_files,
char *list_of_files[]);
  int  (*read)(int32 *valp);
  int  (*write)(char *buf, int32 size);
  int  (*cmsg)(int type, int verbosity_level,
char *fmt, ...);
  void (*event)(CtlEvent *ev);  /* Control
events */
} ControlMode;
```

.

For the simplest values of the functions in this structure, see the code for the dumb interface in `interface/dumb_c.c` .

For dealing with lyrics, the main field to set is the function `event()`. This will be passed a pointer to a `CtlEvent` which is defined in `timidity/controls.h`:

```
typedef struct _CtlEvent {
    int type;           /* See above */
    ptr_size_t v1, v2, v3, v4;/* Event value */
} CtlEvent;
```

The type field distinguishes a large number of event types such as `CTLE_NOW_LOADING` and `CTLE_PITCH_BEND`. The type of interest to us is `CTLE_LYRIC`. `interface/dumb_c.c`

```
static void ctl_event(CtlEvent *e)
{
    switch(e->type) {
      case CTLE_LYRIC:
        ctl_lyric((int)e->v1);
        break;
   }
}

static void ctl_lyric(int lyricid)
{
    char *lyric;

    lyric = event2string(lyricid);
    if(lyric != NULL)
    {
        if(lyric[0] == ME_KARAOKE_LYRIC)
        {
            if(lyric[1] == '/' || lyric[1] ==
'\\')
            {
                fprintf(outfp, "\n%s", lyric +
2);
                fflush(outfp);
            }
            else if(lyric[1] == '@')
```

```
            {
                if(lyric[2] == 'L')
                    fprintf(outfp, "\nLanguage:
%s\n", lyric + 3);
                else if(lyric[2] == 'T')
                   fprintf(outfp, "Title:
%s\n", lyric + 3);
                else
                    fprintf(outfp, "%s\n",
lyric + 1);
            }
            else
            {
                fputs(lyric + 1, outfp);
                fflush(outfp);
            }
        }
        else
        {
            if(lyric[0] == ME_CHORUS_TEXT ||
lyric[0] == ME_INSERT_TEXT)
                fprintf(outfp, "\r");
            fputs(lyric + 1, outfp);
            fflush(outfp);
        }
    }
}
```

Getting the list of lyrics

The failing of the current interfaces in TiMidity with regard to Karaoke is that while they can show the lyrics as they are played, they don't show the lyric lines and progressively highlight them as they are played. For that, you need the set of lyrics.

TiMidity in fact builds a list of lyrics, and makes them accessible. It has a function `event2string()` which takes an integer parameter from one upwards. For each value it returns the string of a lyric or text event, finally returning `NULL` on the end of the list. The first character returned is a type parameter, the rest is the string. Using GLib functions, we can build up an array of lines for a `KAR` file by

```
struct _lyric_t {
    gchar *lyric;
    long tick; // not used here
};
typedef struct _lyric_t lyric_t;

struct _lyric_lines_t {
    char *language;
    char *title;
    char *performer;
    GArray *lines; // array of GString *
};
typedef struct _lyric_lines_t lyric_lines_t;

GArray *lyrics;
lyric_lines_t lyric_lines;

static void build_lyric_lines() {
    int n;
    lyric_t *plyric;
    GString *line = g_string_new("");
    GArray *lines =  g_array_sized_new(FALSE,
FALSE, sizeof(GString *), 64);

    lyric_lines.title = NULL;
```

```
    n = 1;
    char *evt_str;
    while ((evt_str = event2string(n++)) !=
NULL) {
        gchar *lyric = evt_str+1;

        if ((strlen(lyric) >= 2) && (lyric[0]
== '@') && (lyric[1] == 'L')) {
            lyric_lines.language =  lyric + 2;
            continue;
        }

        if ((strlen(lyric) >= 2) && (lyric[0]
== '@') && (lyric[1] == 'T')) {
            if (lyric_lines.title == NULL) {
                lyric_lines.title = lyric + 2;
            } else {
                lyric_lines.performer = lyric +
2;
            }
            continue;
        }

        if (lyric[0] == '@') {
            // some other stuff like @KMIDI
KARAOKE FILE
            continue;
```

```
        }

        if ((lyric[0] == '/') || (lyric[0] ==
'\\')) {
            // start of a new line
            // add to lines
            g_array_append_val(lines, line);
            line = g_string_new(lyric + 1);
        }  else {
            line = g_string_append(line,
lyric);
        }
    }
    lyric_lines.lines = lines;

    printf("Title is %s, performer is %s,
language is %s\n",
           lyric_lines.title,
lyric_lines.performer, lyric_lines.language);
    for (n = 0; n < lines->len; n++) {
        printf("Line is %s\n",
g_array_index(lines, GString *, n)->str);
    }
}
```

The function `build_lyric_lines()` should be called from the `CTLE_LOADING_DONE` branch of `ctl_event()`.

TiMidity options

If we choose to use TiMidity as front-end then we need to run it

```
timidity -d. -iv --trace  --trace-text-meta ...
```

for a "v" interface in the current directory.

The alternative is building a main program that calls TiMidity as a library. The command line parameters to TiMidity then have to be included as hard-coded parameters in the application. One is easy: the `CtlMode` has a field `trace_playing` and setting that to one turns tracing on. Including Text events as Lyric events requires digging a bit deeper into TiMidity, but just requires (shortly after initialising the library)

```
extern int opt_trace_text_meta_event;
opt_trace_text_meta_event = 1;
```

Playing lyrics using Pnago + Cairo + Xlib

I want to be able to play my Karaoke files on the Raspberry Pi and similar Systems on a Chip (SOCs). Unfortuanately the rapsberry Pi has a grossly underpowered CPU, so I have ended up using a CubieBoard 2 .

Even then I don't know how to program the GPU, so anything involving heavy graphics is not possible on this CPU. Any of the MIDI players hit close to (or over) 100% CPU usage just be themselves. So the system discussed in the next section, showing background video, isn't feasible.

In this section we use TiMidity as MIDI player with a minimal backend to display the lyrics as they are played. The lowest level of GUI support is used, namely Xlib. This can be used to draw text using low-level Xlib calls such as `XDrawImageString` . This wrks fine with ASCII languages, and with appropriate font choices, with other languages in the ISO-8859 family.

Asian languages are harder to deal with in standard C. They involve one or two byte characters when using an encoding such as UTF-8. To manage these. it is easiest to switch to a library

designed to handle them such, such as Cairo.

Cairo is good for drawing simple text. For e.g. Chinese characters you have to find a font that will allow you to draw them. Alternatively, you can jump up one further level to Pango. Pango looks after all the font issues and produces glyphs which are sent to the X server.

that approach is adopted in the following interface, `x_code.c`

```c
#include <X11/Xlib.h>
#include <X11/Xutil.h>

#include <gtk/gtk.h>

#include <libavcodec/avcodec.h>
#include <libavformat/avformat.h>
#include <libswscale/swscale.h>

#include "mytimidity.h"

#define WIDTH 720
#define HEIGHT 480

#define NUM_LINES 4

Display *display;
Window window;
GC gc, color_gc;

struct _lyric_t {
    gchar *lyric;
    long tick;

};
typedef struct _lyric_t lyric_t;
```

```
struct _lyric_lines_t {
    char *language;
    char *title;
    char *performer;
    GArray *lines; // array of GString *
};
typedef struct _lyric_lines_t lyric_lines_t;

GArray *lyrics;
GString *lyrics_array[NUM_LINES];

lyric_lines_t lyric_lines;

typedef struct _coloured_line_t {
    gchar *line;
    gchar *front_of_line;
    gchar *marked_up_line;
    PangoAttrList *attrs;
} coloured_line_t;

int height_lyric_pixbufs[] = {100, 200, 300,
400}; // vertical offset of lyric in video
int coloured_text_offset;

// fluid_player_t* player;
```

```
// int current_panel = 1;  // panel showing
current lyric line
int current_line = 0;  // which line is the
current lyric
gchar *current_lyric;   // currently playing
lyric line
GString *front_of_lyric;  // part of lyric to
be coloured red
//GString *end_of_lyric;    // part of lyrci to
not be coloured


gchar *markup[] = {"<span font=\"28\"
foreground=\"RED\">",
           "</span><span font=\"28\"
foreground=\"white\">",
           "</span>"};
gchar *markup_newline[] = {"<span
foreground=\"black\">",
               "</span>"};
GString *marked_up_label;

PangoFontDescription *font_description;

cairo_surface_t *surface;
cairo_t *cr;
```

```
extern ControlMode  *ctl;

ControlMode video_ctl=
    {
    "x interface", 'N',
    "x",
    1,          /* verbosity */
    1,          /* trace playing */
    0,          /* opened */
    0,          /* flags */
    ctl_open,
    ctl_close,
    pass_playing_list,
    ctl_read,
    NULL,       /* write */
    cmsg,
    ctl_event
    };

static FILE *outfp;
int video_error_count;
static char *current_file;
struct midi_file_info *current_file_info;

static int pass_playing_list(int
number_of_files, char *list_of_files[]) {
    int n;
```

```
    for (n = 0; n < number_of_files; n++) {
    printf("Playing list %s\n",
list_of_files[n]);

    current_file = list_of_files[n];
    play_midi_file( list_of_files[n]);
    }
    XCloseDisplay(display);
    exit(0);
    return 0;
}

static void paint_background() {
    cr = cairo_create(surface);
    cairo_set_source_rgb(cr, 0.0, 0.8, 0.0);
    cairo_paint(cr);
    cairo_destroy(cr);
}

static void set_font() {
    font_description =
pango_font_description_new ();
    pango_font_description_set_family
(font_description, "serif");
    pango_font_description_set_weight
(font_description, PANGO_WEIGHT_BOLD);
```

```
    pango_font_description_set_absolute_size
(font_description, 32 * PANGO_SCALE);
}

static int draw_text(char *text, float red,
float green, float blue, int height, int
offset) {
  // See http://cairographics.org/FAQ/
  PangoLayout *layout;
  int width, ht;
  cairo_text_extents_t extents;

  layout = pango_cairo_create_layout (cr);
  pango_layout_set_font_description (layout,
font_description);
  pango_layout_set_text (layout, text, -1);

  if (offset == 0) {
      pango_layout_get_size(layout, &width,
&ht);
      offset = (WIDTH - (width/PANGO_SCALE)) /
2;
  }

  cairo_set_source_rgb (cr, red, green, blue);
  cairo_move_to (cr, offset, height);
  pango_cairo_show_layout (cr, layout);
```

```
  g_object_unref (layout);
  return offset;
}

static void init_X() {
    int screenNumber;
    unsigned long foreground, background;
    int screen_width, screen_height;
    Screen *screen;
    XSizeHints hints;
    char **argv = NULL;
    XGCValues gcValues;
    Colormap colormap;
    XColor rgb_color, hw_color;
    Font font;
    //char *FNAME = "hanzigb24st";
    char *FNAME = "-misc-fixed-medium-r-normal-
-0-0-100-100-c-0-iso10646-1";

    display = XOpenDisplay(NULL);
    if (display == NULL) {
    fprintf(stderr, "Can't open dsplay\n");
    exit(1);
    }
    screenNumber = DefaultScreen(display);
    foreground = BlackPixel(display,
```

```
screenNumber);
    background = WhitePixel(display,
screenNumber);

    screen = DefaultScreenOfDisplay(display);
    screen_width = WidthOfScreen(screen);
    screen_height = HeightOfScreen(screen);

    hints.x = (screen_width - WIDTH) / 2;
    hints.y = (screen_height - HEIGHT) / 2;
    hints.width = WIDTH;
    hints.height = HEIGHT;
    hints.flags = PPosition | PSize;

    window = XCreateSimpleWindow(display,
                 DefaultRootWindow(display),
                 hints.x, hints.y, WIDTH,
HEIGHT, 10,
                 foreground, background);

    XSetStandardProperties(display, window,
               "TiMidity", "TiMidity",
               None,
               argv, 0,
               &hints);

    XMapWindow(display, window);
```

```
    set_font();
    surface =
cairo_xlib_surface_create(display, window,
                      DefaultVisual(display, 0),
WIDTH, HEIGHT);
    cairo_xlib_surface_set_size(surface, WIDTH,
HEIGHT);

    paint_background();

    /*
    cr = cairo_create(surface);
    draw_text(g_array_index(lyric_lines.lines,
GString *, 0)->str,
          0.0, 0.0, 1.0,
height_lyric_pixbufs[0]);
    draw_text(g_array_index(lyric_lines.lines,
GString*, 1)->str,
          0.0, 0.0, 1.0,
height_lyric_pixbufs[0]);
    cairo_destroy(cr);
    */
    XFlush(display);
}
```

```
static int inited_video = 0;
/*ARGSUSED*/
static int ctl_open(int using_stdin, int
using_stdout)
{
  init_X();

    // dont know what this function does
    /*
      if (current_file != NULL) {
      current_file_info =
get_midi_file_info(current_file, 1);
      printf("Opening info for %s\n",
current_file);
      } else {
      printf("Current is NULL\n");
      }
    */
    ctl->opened = 1;
    return 0;
}

static void ctl_close(void)
{
    fflush(outfp);
    video_ctl.opened=0;
```

```
    exit(0);
}

/*ARGSUSED*/
static int ctl_read(int32 *valp)
{
    return RC_NONE;
}

static int cmsg(int type, int verbosity_level,
char *fmt, ...)
{
    /*
      va_list ap;

      if ((type==CMSG_TEXT || type==CMSG_INFO
|| type==CMSG_WARNING) &&
      video_ctl.verbosity<verbosity_level)
      return 0;
      va_start(ap, fmt);
      if(type == CMSG_WARNING || type ==
CMSG_ERROR || type == CMSG_FATAL)
      video_error_count++;
      if (!video_ctl.opened)
      {
      vfprintf(stderr, fmt, ap);
      fputs(NLS, stderr);
```

```
      }
      else
      {
      vfprintf(outfp, fmt, ap);
      fputs(NLS, outfp);
      fflush(outfp);
      }
      va_end(ap);
    */
    return 0;
}

static void ctl_total_time(long tt)
{
    /*
      int mins, secs;
      if (video_ctl.trace_playing)
      {
      secs=(int)(tt/play_mode->rate);
      mins=secs/60;
      secs-=mins*60;
      cmsg(CMSG_INFO, VERB_NORMAL,
      "Total playing time: %3d min %02d s",
mins, secs);
      }
    */
}
```

```
static void ctl_file_name(char *name)
{
    current_file = name;

    if (video_ctl.verbosity>=0 ||
video_ctl.trace_playing)
    cmsg(CMSG_INFO, VERB_NORMAL, "Playing %s",
name);
}

static void ctl_current_time(int secs)
{
    int mins;
    static int prev_secs = -1;

#ifdef __W32__
    if(wrdt->id == 'w')
    return;
#endif /* __W32__ */
    if (ctl->trace_playing && secs !=
prev_secs)
    {
        prev_secs = secs;
        mins=secs/60;
        secs-=mins*60;
        fprintf(stdout, "\r%3d:%02d", mins,
```

```
secs);
    }
}

void build_lyric_lines() {
    int n;
    lyric_t *plyric;
    GString *line = g_string_new("");
    GArray *lines =  g_array_sized_new(FALSE,
FALSE, sizeof(GString *), 64);

    lyric_lines.title = NULL;

    n = 1;
    char *evt_str;
    while ((evt_str = event2string(n++)) !=
NULL) {

        gchar *lyric = evt_str+1;
    printf("Building line %s\n", lyric);

    if ((strlen(lyric) >= 2) && (lyric[0] ==
'@') && (lyric[1] == 'L')) {
        lyric_lines.language =  lyric + 2;
        continue;
    }
```

```
    if ((strlen(lyric) >= 2) && (lyric[0] ==
'@') && (lyric[1] == 'T')) {
        if (lyric_lines.title == NULL) {
        lyric_lines.title = lyric + 2;
        } else {
        lyric_lines.performer = lyric + 2;
        }
        continue;
    }

    if (lyric[0] == '@') {
        // some other stuff like @KMIDI KARAOKE
FILE
        continue;
    }

    if ((lyric[0] == '/') || (lyric[0] ==
'\\')) {
        // start of a new line
        // add to lines
        printf("New line\n");
        g_array_append_val(lines, line);
        line = g_string_new(lyric + 1);
    }  else {
        line = g_string_append(line, lyric);
    }
    }
```

```
    lyric_lines.lines = lines;

    printf("Title is %s, performer is %s,
language is %s\n",
       lyric_lines.title,
lyric_lines.performer, lyric_lines.language);
    for (n = 0; n < lines->len; n++) {
    printf("Line is %s\n", g_array_index(lines,
GString *, n)->str);
    }

}

static void ctl_lyric(int lyricid)
{
    char *lyric;

    current_file_info =
get_midi_file_info(current_file, 1);

    lyric = event2string(lyricid);
    if(lyric != NULL)
    lyric++;
    printf("Got a lyric %s\n", lyric);


    if ((*lyric == '\\') || (*lyric == '/')) {
```

```
    // int next_panel = (current_panel+2) %
NUM_LINES; // really (current_panel+2)%2
    int next_line = current_line + NUM_LINES;
    gchar *next_lyric;

    if (current_line + NUM_LINES <
lyric_lines.lines->len) {
        current_line += 1;

        //lyrics_array[(next_line-1) %
NUM_LINES] =
        //    g_array_index(lyric_lines.lines,
GString *, next_line);

        // update label for next line after
this one
        next_lyric =
g_array_index(lyric_lines.lines, GString *,
next_line)->str;

    } else {
        current_line += 1;
        lyrics_array[(next_line-1) % NUM_LINES]
= NULL;
        next_lyric = "";
    }
```

```
    // set up new line as current line
    if (current_line < lyric_lines.lines->len)
{
        GString *gstr =
g_array_index(lyric_lines.lines, GString *,
current_line);
        current_lyric = gstr->str;
        front_of_lyric = g_string_new(lyric+1);
// lose    slosh
    }
    printf("New line. Setting front to %s end
to \"%s\"\n", lyric+1, current_lyric);


    // Now draw stuff
    paint_background();

    cr = cairo_create(surface);

    int n;
    for (n = 0; n < NUM_LINES; n++) {
        //lyrics_array[n] =
g_array_index(lyric_lines.lines, GString *,
n+1);
        if (lyrics_array[n] != NULL) {
        draw_text(lyrics_array[n]->str,
              0.0, 0.0, 0.5,
```

```
height_lyric_pixbufs[n], 0);
        }
    }
    // redraw current and next lines
    if (current_line < lyric_lines.lines->len)
{
        if (current_line >= 2) {
        // redraw last line still in red
        GString *gstr =
lyrics_array[(current_line-2) % NUM_LINES];
        if (gstr != NULL) {
            draw_text(gstr->str,
                  1.0, 0.0, 00,

height_lyric_pixbufs[(current_line-2) %
NUM_LINES],
                  0);
        }
        }
        // draw next line in brighter blue
        coloured_text_offset =
draw_text(lyrics_array[(current_line-1) %
NUM_LINES]->str,
              0.0, 0.0, 1.0,
height_lyric_pixbufs[(current_line-1) %
NUM_LINES], 0);
        printf("coloured text offset %d\n",
```

```
coloured_text_offset);
    }

    //try
    if (next_line < lyric_lines.lines->len) {
        lyrics_array[(next_line-1) % NUM_LINES]
=
        g_array_index(lyric_lines.lines,
GString *, next_line);
    }


        /*
    draw_text(current_lyric, 0.0, 0.0, 1.0,
          height_lyric_pixbufs[(current_line-1)
% NUM_LINES]);


    draw_text(next_lyric, 0.0, 0.0, 1.0,
          height_lyric_pixbufs[(next_line-1) %
NUM_LINES]);
        */
    cairo_destroy(cr);
    XFlush(display);


    } else {
```

```
    // change text colour as chars are played
    if ((front_of_lyric != NULL) && (lyric !=
NULL)) {
        g_string_append(front_of_lyric, lyric);
        char *s = front_of_lyric->str;

//coloured_lines[current_panel].front_of_line =
s;

        cairo_t *cr = cairo_create(surface);

        // See http://cairographics.org/FAQ/
        draw_text(s, 1.0, 0.0, 0.0,

height_lyric_pixbufs[(current_line-1) %
NUM_LINES],
              coloured_text_offset);

        cairo_destroy(cr);
        XFlush(display);

    }
    }
}


static void ctl_event(CtlEvent *e)
```

```
{
    //printf("Got ctl event %d\n", e->type);
    switch(e->type)
    {
    case CTLE_NOW_LOADING:
        ctl_file_name((char *)e->v1);
        break;
    case CTLE_LOADING_DONE:
        // MIDI file is loaded, about to play
        current_file_info =
get_midi_file_info(current_file, 1);
        if (current_file_info != NULL) {
        printf("file info not NULL\n");
        } else {
        printf("File info is NULL\n");
        }

        int n = 1;
        char *evt_str;
        while ((evt_str = event2string(n++)) !=
NULL) {
        printf("Event in tabel: %s\n",
evt_str);
        }

        build_lyric_lines();
        cr = cairo_create(surface);
```

```
        // draw line to be sung slightly
brighter
        // than the rest
        for (n = 0; n < NUM_LINES; n++) {
        lyrics_array[n] =
g_array_index(lyric_lines.lines, GString *,
n+1);
        draw_text(lyrics_array[n]->str,
              0.0, 0.0, 0.5,
height_lyric_pixbufs[n], 0);
        }
        draw_text(lyrics_array[0]->str,
              0.0, 0.0, 1.0,
height_lyric_pixbufs[0], 0);

        /*

draw_text(g_array_index(lyric_lines.lines,
GString *, 1)->str,
              0.0, 0.0, 1.0,
height_lyric_pixbufs[0]);

draw_text(g_array_index(lyric_lines.lines,
GString*, 2)->str,
              0.0, 0.0, 1.0,
height_lyric_pixbufs[1]);
```

```
draw_text(g_array_index(lyric_lines.lines,
GString*, 3)->str,
              0.0, 0.0, 1.0,
height_lyric_pixbufs[2]);
        */
        cairo_destroy(cr);
        XFlush(display);

        break;
    case CTLE_PLAY_START:

        ctl_total_time(e->v1);
        break;
        case CTLE_PLAY_END:
        printf("Exiting, play ended\n");
        exit(0);
        break;
    case CTLE_CURRENT_TIME:
        ctl_current_time((int)e->v1);
        break;
    case CTLE_LYRIC:
        ctl_lyric((int)e->v1);
        break;
        /*
          case CTLE_REFRESH:
          printf("Refresh\n");
```

```
            break;
        */
    default:
        0;
        //printf("Other event\n");
    }
}

/*
 * interface_<id>_loader();
 */
ControlMode *interface_x_loader(void)
{
    return &video_ctl;
}
```

it is compiled by

```
gcc  -fPIC $(CFLAGS) -c -o x_code.o x_code.c
$(LIBS)
        gcc -shared -o if_x.so x_code.o $(LIBS)
```

and run by

```
timidity -d. -ix --trace --trace-text-meta ...
```

Playing a background video with Gtk

In the chapter on FluidSynth case: we discussed a program to show lyrics overlaid onto a movie. Apart from the above considerations, the rest of the application follows similarly to the FluidSynth case: build a set of lyric lines, display them using Pango over Gtk Pixbufs, and when a new lyric event occurs update the corresponding colours in the lyric line.

All of the dynamic action needs to occur out of the back-end of TiMidity. particularly in the function `ctl_event`. Other parts such as initialising FFMpeg and Gtk must also occur in the back-end when using standard TiMidity. If TiMidity is used as a library, this initialisation could occur in the front or the back. For simplicity, we just place it all in the back in the file `video_code.c`:

```
#include <gtk/gtk.h>

#include <libavcodec/avcodec.h>
#include <libavformat/avformat.h>
#include <libswscale/swscale.h>

#include "mytimidity.h"

#define USE_PIXBUF

// saving as pixbufs leaks memory
//#define USE_PIXBUF

/* run by
   gtkkaraoke_player_video_pango
/usr/share/sounds/sf2/FluidR3_GM.sf2
/home/newmarch/Music/karaoke/sonken/songs/54154.k

*/

/*
 * APIs:
 * GString:
https://developer.gnome.org/glib/2.28/glib-
Strings.html
 * Pango text attributes:
```

```
https://developer.gnome.org/pango/stable/pango-Text-Attributes.html#pango-parse-markup
 * Pango layout: http://www.gtk.org/api/2.6/pango/pango-Layout-Objects.html
 * Cairo rendering: https://developer.gnome.org/pango/stable/pango-Cairo-Rendering.html#pango-cairo-create-layout
 * Cairo surface_t: http://cairographics.org/manual/cairo-cairo-surface-t.html
 * GTK+ 3 Reference Manual: https://developer.gnome.org/gtk3/3.0/
 * Gdk Pixbufs: https://developer.gnome.org/gdk/stable/gdk-Pixbufs.html
 */

struct _lyric_t {
    gchar *lyric;
    long tick;

};
typedef struct _lyric_t lyric_t;

struct _lyric_lines_t {
    char *language;
```

```
    char *title;
    char *performer;
    GArray *lines; // array of GString *
};
typedef struct _lyric_lines_t lyric_lines_t;

GArray *lyrics;

lyric_lines_t lyric_lines;

typedef struct _coloured_line_t {
    gchar *line;
    gchar *front_of_line;
    gchar *marked_up_line;
    PangoAttrList *attrs;
#ifdef USE_PIXBUF
    GdkPixbuf *pixbuf;
#endif
} coloured_line_t;

coloured_line_t coloured_lines[2];

// fluid_synth_t* synth;

GtkWidget *image;
#if GTK_MAJOR_VERSION == 2
GdkPixmap *dbuf_pixmap;
```

```
#endif

int height_lyric_pixbufs[] = {300, 400}; //
vertical offset of lyric in video

// fluid_player_t* player;

int current_panel = 1;  // panel showing
current lyric line
int current_line = 0;  // which line is the
current lyric
gchar *current_lyric;   // currently playing
lyric line
GString *front_of_lyric;  // part of lyric to
be coloured red
//GString *end_of_lyric;    // part of lyrci to
not be coloured



// Colours seem to get mixed up when putting a
pixbuf onto a pixbuf
#ifdef USE_PIXBUF
#define RED blue
#else
#define RED red
#endif
```

```
gchar *markup[] = {"<span font=\"28\"
foreground=\"RED\">",
           "</span><span font=\"28\"
foreground=\"white\">",
           "</span>"};
gchar *markup_newline[] = {"<span
foreground=\"black\">",
               "</span>"};
GString *marked_up_label;

/* FFMpeg vbls */
AVFormatContext *pFormatCtx = NULL;
AVCodecContext *pCodecCtx = NULL;
int videoStream;
struct SwsContext *sws_ctx = NULL;
AVCodec *pCodec = NULL;


void markup_line(coloured_line_t *line) {
    GString *str =  g_string_new(markup[0]);
    g_string_append(str, line->front_of_line);
    g_string_append(str, markup[1]);
    g_string_append(str, line->line +
strlen(line->front_of_line));
    g_string_append(str, markup[2]);
    printf("Marked up label \"%s\"\n", str-
```

```
>str);

    line->marked_up_line = str->str;
    // we have to free line->marked_up_line

    pango_parse_markup(str->str, -1,0, &(line-
>attrs), NULL, NULL, NULL);
    g_string_free(str, FALSE);
}

#ifdef USE_PIXBUF
void update_line_pixbuf(coloured_line_t *line)
{
    //return;
    cairo_surface_t *surface;
    cairo_t *cr;

    int lyric_width = 480;
    int lyric_height = 60;
    surface = cairo_image_surface_create
(CAIRO_FORMAT_ARGB32,

lyric_width, lyric_height);
    cr = cairo_create (surface);

    PangoLayout *layout;
    PangoFontDescription *desc;
```

```
    // draw the attributed text
    layout = pango_cairo_create_layout (cr);
    pango_layout_set_text (layout, line->line,
-1);
    pango_layout_set_attributes(layout, line-
>attrs);

    // centre the image in the surface
    int width, height;
    pango_layout_get_pixel_size(layout,
                &width,
                &height);
    cairo_move_to(cr, (lyric_width-width)/2,
0);

    pango_cairo_update_layout (cr, layout);
    pango_cairo_show_layout (cr, layout);

    // pull the pixbuf out of the surface
    unsigned char *data =
cairo_image_surface_get_data(surface);
    width =
cairo_image_surface_get_width(surface);
    height =
cairo_image_surface_get_height(surface);
    int stride =
```

```
cairo_image_surface_get_stride(surface);
    printf("Text surface width %d height %d
stride %d\n", width, height, stride);

    GdkPixbuf *old_pixbuf = line->pixbuf;
    line->pixbuf =
gdk_pixbuf_new_from_data(data,
GDK_COLORSPACE_RGB, 1, 8, width, height,
stride, NULL, NULL);
    cairo_surface_destroy(surface);
    g_object_unref(old_pixbuf);
}
#endif

extern ControlMode  *ctl;

ControlMode video_ctl=
    {
    "video interface", 'v',
    "video",
    1,          /* verbosity */
    1,          /* trace playing */
    0,          /* opened */
    0,          /* flags */
    ctl_open,
    ctl_close,
    pass_playing_list,
```

```
    ctl_read,
    NULL,        /* write */
    cmsg,
    ctl_event
    };

static void init_timidity() {
    int err;

    timidity_start_initialize();

    if ((err =
timidity_pre_load_configuration()) != 0) {
    printf("couldn't pre-load configuration
file\n");
    exit(1);
    }

    err += timidity_post_load_configuration();

    if (err) {
    printf("couldn't post-load configuration
file\n");
    exit(1);
    }

    timidity_init_player();
```

```
    extern int opt_trace_text_meta_event;
    opt_trace_text_meta_event = 1;

    //ctl = &video_ctl;
    //ctl->trace_playing = 1;
    //opt_trace_text_meta_event = 1;

}

static FILE *outfp;
int video_error_count;
static char *current_file;
struct midi_file_info *current_file_info;

static int pass_playing_list(int
number_of_files, char *list_of_files[]) {
    int n;

    for (n = 0; n < number_of_files; n++) {
    printf("Playing list %s\n",
list_of_files[n]);

    current_file = list_of_files[n];
    /*
      current_file_info =
```

```
get_midi_file_info(current_file, 1);
      if (current_file_info != NULL) {
      printf("file info not NULL\n");
      } else {
      printf("File info is NULL\n");
      }
    */
    play_midi_file( list_of_files[n]);
    }
    return 0;
}

static int inited_video = 0;
/*ARGSUSED*/
static int ctl_open(int using_stdin, int
using_stdout)
{
    if (! inited_video) {
    init_video();
    inited_video = 1;
    }

    // dont know what this function does
    /*
      if (current_file != NULL) {
      current_file_info =
get_midi_file_info(current_file, 1);
```

```
      printf("Opening info for %s\n",
current_file);
      } else {
      printf("Current is NULL\n");
      }
    */
    ctl->opened = 1;
    return 0;
}

static void ctl_close(void)
{
    fflush(outfp);
    video_ctl.opened=0;
}

/*ARGSUSED*/
static int ctl_read(int32 *valp)
{
    return RC_NONE;
}

static int cmsg(int type, int verbosity_level,
char *fmt, ...)
{
    /*
      va_list ap;
```

```
      if ((type==CMSG_TEXT || type==CMSG_INFO
|| type==CMSG_WARNING) &&
      video_ctl.verbosity<verbosity_level)
      return 0;
      va_start(ap, fmt);
      if(type == CMSG_WARNING || type ==
CMSG_ERROR || type == CMSG_FATAL)
      video_error_count++;
      if (!video_ctl.opened)
      {
      vfprintf(stderr, fmt, ap);
      fputs(NLS, stderr);
      }
      else
      {
      vfprintf(outfp, fmt, ap);
      fputs(NLS, outfp);
      fflush(outfp);
      }
      va_end(ap);
    */
    return 0;
}

static void ctl_total_time(long tt)
{
```

```
    /*
      int mins, secs;
      if (video_ctl.trace_playing)
      {
      secs=(int)(tt/play_mode->rate);
      mins=secs/60;
      secs-=mins*60;
      cmsg(CMSG_INFO, VERB_NORMAL,
      "Total playing time: %3d min %02d s",
mins, secs);
      }
    */
}

static void ctl_file_name(char *name)
{
    current_file = name;

    if (video_ctl.verbosity>=0 ||
video_ctl.trace_playing)
    cmsg(CMSG_INFO, VERB_NORMAL, "Playing %s",
name);
}

static void ctl_current_time(int secs)
{
    int mins;
```

```
    static int prev_secs = -1;

#ifdef __W32__
    if(wrdt->id == 'w')
    return;
#endif /* __W32__ */
    if (ctl->trace_playing && secs !=
prev_secs)
    {
        prev_secs = secs;
        mins=secs/60;
        secs-=mins*60;
        fprintf(stdout, "\r%3d:%02d", mins,
secs);
    }
}

void build_lyric_lines() {
    int n;
    lyric_t *plyric;
    GString *line = g_string_new("");
    GArray *lines =  g_array_sized_new(FALSE,
FALSE, sizeof(GString *), 64);

    lyric_lines.title = NULL;

    n = 1;
```

```
    char *evt_str;
    while ((evt_str = event2string(n++)) !=
NULL) {

        gchar *lyric = evt_str+1;
    printf("Building line %s\n", lyric);

    if ((strlen(lyric) >= 2) && (lyric[0] ==
'@') && (lyric[1] == 'L')) {
        lyric_lines.language =  lyric + 2;
        continue;
    }

    if ((strlen(lyric) >= 2) && (lyric[0] ==
'@') && (lyric[1] == 'T')) {
        if (lyric_lines.title == NULL) {
        lyric_lines.title = lyric + 2;
        } else {
        lyric_lines.performer = lyric + 2;
        }
        continue;
    }

    if (lyric[0] == '@') {
        // some other stuff like @KMIDI KARAOKE
FILE
        continue;
```

```
    }

    if ((lyric[0] == '/') || (lyric[0] ==
'\\')) {
        // start of a new line
        // add to lines
        g_array_append_val(lines, line);
        line = g_string_new(lyric + 1);
    }  else {
        line = g_string_append(line, lyric);
    }
    }
    lyric_lines.lines = lines;

    printf("Title is %s, performer is %s,
language is %s\n",
       lyric_lines.title,
lyric_lines.performer, lyric_lines.language);
    for (n = 0; n < lines->len; n++) {
    printf("Line is %s\n", g_array_index(lines,
GString *, n)->str);
    }

}

static void overlay_lyric(cairo_t *cr,
              coloured_line_t *line,
```

```
                int ht) {
    PangoLayout *layout;
    int height, width;

    if (line->line == NULL) {
    return;
    }

    layout = pango_cairo_create_layout (cr);
    pango_layout_set_text (layout, line->line,
-1);
    pango_layout_set_attributes(layout, line-
>attrs);
    pango_layout_get_pixel_size(layout,
                &width,
                &height);
    cairo_move_to(cr, (720-width)/2, ht);

    pango_cairo_update_layout (cr, layout);
    pango_cairo_show_layout (cr, layout);

    g_object_unref(layout);
}

static void pixmap_destroy_notify(guchar
*pixels,
                  gpointer data) {
```

```
    printf("Ddestroy pixmap\n");
}

static void *play_background(void *args) {
    /* based on code from

http://www.cs.dartmouth.edu/~xy/cs23/gtk.html

http://cdry.wordpress.com/2009/09/09/using-
custom-io-callbacks-with-ffmpeg/
    */

    int i;
    AVPacket packet;
    int frameFinished;
    AVFrame *pFrame = NULL;

    int oldSize;
    char *oldData;
    int bytesDecoded;
    GdkPixbuf *pixbuf;
    AVFrame *picture_RGB;
    char *buffer;

#if GTK_MAJOR_VERSION == 2
    GdkPixmap *pixmap;
    GdkBitmap *mask;
```

```
#endif

    pFrame=avcodec_alloc_frame();

    i=0;
    picture_RGB = avcodec_alloc_frame();
    buffer = malloc
(avpicture_get_size(PIX_FMT_RGB24, 720, 576));
    avpicture_fill((AVPicture *)picture_RGB,
buffer, PIX_FMT_RGB24, 720, 576);

    int width = pCodecCtx->width;
    int height = pCodecCtx->height;

    sws_ctx = sws_getContext(pCodecCtx->width,
pCodecCtx->height, pCodecCtx->pix_fmt,
pCodecCtx->width, pCodecCtx->height,
PIX_FMT_RGB24, SWS_BICUBIC, NULL, NULL, NULL);

    while(av_read_frame(pFormatCtx,
&packet)>=0) {
    if(packet.stream_index==videoStream) {
        //printf("Frame %d\n", i++);
        usleep(33670);  // 29.7 frames per
second
        // Decode video frame
        avcodec_decode_video2(pCodecCtx,
```

```
pFrame, &frameFinished,
                  &packet);


        if (frameFinished) {
        //printf("Frame %d\n", i++);

        sws_scale(sws_ctx,  (uint8_t const *
const *) pFrame->data, pFrame->linesize, 0,
pCodecCtx->height, picture_RGB->data,
picture_RGB->linesize);


        pixbuf =
gdk_pixbuf_new_from_data(picture_RGB->data[0],
GDK_COLORSPACE_RGB, 0, 8, 720, 480,
picture_RGB->linesize[0],
pixmap_destroy_notify, NULL);


        /* start GTK thread lock for drawing */
        gdk_threads_enter();

#define SHOW_LYRIC
#ifdef SHOW_LYRIC
                // Create the destination
surface
```

```
                cairo_surface_t *surface =
cairo_image_surface_create
(CAIRO_FORMAT_ARGB32,

width, height);
                cairo_t *cr =
cairo_create(surface);

                // draw the background image
                gdk_cairo_set_source_pixbuf(cr,
pixbuf, 0, 0);
                cairo_paint (cr);

#ifdef USE_PIXBUF
        // draw the lyric
        GdkPixbuf *lyric_pixbuf =
coloured_lines[current_panel].pixbuf;
        if (lyric_pixbuf != NULL) {
            int width =
gdk_pixbuf_get_width(lyric_pixbuf);
            gdk_cairo_set_source_pixbuf(cr,
                        lyric_pixbuf,
                        (720-width)/2,

height_lyric_pixbufs[current_panel]);
            cairo_paint (cr);
        }
```

```
        int next_panel = (current_panel+1) % 2;
        lyric_pixbuf =
coloured_lines[next_panel].pixbuf;
        if (lyric_pixbuf != NULL) {
            int width =
gdk_pixbuf_get_width(lyric_pixbuf);
            gdk_cairo_set_source_pixbuf(cr,
                        lyric_pixbuf,
                        (720-width)/2,

height_lyric_pixbufs[next_panel]);
            cairo_paint (cr);
        }
#else

        overlay_lyric(cr,
                  coloured_lines+current_panel,

height_lyric_pixbufs[current_panel]);

        int next_panel = (current_panel+1) % 2;
        overlay_lyric(cr,
                  coloured_lines+next_panel,

height_lyric_pixbufs[next_panel]);
#endif
```

```
        pixbuf =
gdk_pixbuf_get_from_surface(surface,
                                0,
                                0,
                                width,
                                height);

        gtk_image_set_from_pixbuf((GtkImage*)
image, pixbuf);

        g_object_unref(pixbuf);        /*
reclaim memory */
        //sws_freeContext(sws_ctx);
        //g_object_unref(layout);
        cairo_surface_destroy(surface);
        cairo_destroy(cr);
#else
        gtk_image_set_from_pixbuf((GtkImage*)
image, pixbuf);
#endif /* SHOW_LYRIC */

        /* release GTK thread lock */
        gdk_threads_leave();
        }
    }
    av_free_packet(&packet);
    }
```

```
    sws_freeContext(sws_ctx);

    printf("Video over!\n");
    exit(0);
}

static void ctl_lyric(int lyricid)
{
    char *lyric;

    current_file_info =
get_midi_file_info(current_file, 1);

    lyric = event2string(lyricid);
    if(lyric != NULL)
    lyric++;
    printf("Got a lyric %s\n", lyric);
    if (*lyric == '\\') {
    int next_panel = current_panel; // really
(current_panel+2)%2
    int next_line = current_line + 2;
    gchar *next_lyric;

    if (current_line + 2 >= lyric_lines.lines-
>len) {
        return;
    }
```

```
    current_line += 1;
    current_panel = (current_panel + 1) % 2;

    // set up new line as current line
    current_lyric =
g_array_index(lyric_lines.lines, GString *,
current_line)->str;
    front_of_lyric = g_string_new(lyric+1); //
lose \
    printf("New line. Setting front to %s end
to \"%s\"\n", lyric+1, current_lyric);

    coloured_lines[current_panel].line =
current_lyric;
    coloured_lines[current_panel].front_of_line
= lyric+1;
    markup_line(coloured_lines+current_panel);
#ifdef USE_PIXBUF

update_line_pixbuf(coloured_lines+current_panel);

#endif
    // update label for next line after this
one
    next_lyric =
g_array_index(lyric_lines.lines, GString *,
next_line)->str;
```

```
    marked_up_label =
g_string_new(markup_newline[0]);

    g_string_append(marked_up_label,
next_lyric);
    g_string_append(marked_up_label,
markup_newline[1]);
    PangoAttrList *attrs;
    gchar *text;
    pango_parse_markup (marked_up_label->str,
-1,0, &attrs, &text, NULL, NULL);

    coloured_lines[next_panel].line =
next_lyric;
    coloured_lines[next_panel].front_of_line =
"";
    markup_line(coloured_lines+next_panel);
#ifdef USE_PIXBUF

update_line_pixbuf(coloured_lines+next_panel);
#endif
    } else {
    // change text colour as chars are played
    if ((front_of_lyric != NULL) && (lyric !=
NULL)) {
        g_string_append(front_of_lyric, lyric);
```

```
        char *s = front_of_lyric->str;

coloured_lines[current_panel].front_of_line =
s;

markup_line(coloured_lines+current_panel);
#ifdef USE_PIXBUF

update_line_pixbuf(coloured_lines+current_panel);

#endif
    }
    }
}


static void ctl_event(CtlEvent *e)
{
    //printf("Got ctl event %d\n", e->type);
    switch(e->type)
    {
    case CTLE_NOW_LOADING:
        ctl_file_name((char *)e->v1);
        break;
    case CTLE_LOADING_DONE:
        // MIDI file is loaded, about to play
        current_file_info =
```

```
get_midi_file_info(current_file, 1);
        if (current_file_info != NULL) {
        printf("file info not NULL\n");
        } else {
        printf("File info is NULL\n");
        }

        int n = 1;
        char *evt_str;
        while ((evt_str = event2string(n++)) !=
NULL) {
        printf("Event in tabel: %s\n",
evt_str);
        }

        build_lyric_lines();

        break;
    case CTLE_PLAY_START:

        ctl_total_time(e->v1);
        break;
    case CTLE_CURRENT_TIME:
        ctl_current_time((int)e->v1);
        break;
    case CTLE_LYRIC:
        ctl_lyric((int)e->v1);
```

```
        break;
        /*
          case CTLE_REFRESH:
          printf("Refresh\n");
          break;
        */
    default:
        0;
        //printf("Other event\n");
    }
}



static void *play_timidity(void *args) {
    char *argv[] = {"54154.kar"};

    timidity_play_main(1, argv);
    printf("Play timidity finished\n");
}

/* Called when the windows are realized
 */
static void realize_cb (GtkWidget *widget,
gpointer data) {
    /* start the video playing in its own
thread */
```

```
    pthread_t back_id;
    pthread_create(&back_id, NULL,
play_background, NULL);

#if 0
    pthread_t timidity_id;
    pthread_create(&timidity_id, NULL,
play_timidity, NULL);
#endif
}

static gboolean delete_event( GtkWidget
*widget,
                              GdkEvent  *event,
                              gpointer   data )
{
    /* If you return FALSE in the "delete-
event" signal handler,
     * GTK will emit the "destroy" signal.
Returning TRUE means
     * you don't want the window to be
destroyed.
     * This is useful for popping up 'are you
sure you want to quit?'
     * type dialogs. */

    g_print ("delete event occurred\n");
```

```
    /* Change TRUE to FALSE and the main window
will be destroyed with
     * a "delete-event". */

    return TRUE;
}

/* Another callback */
static void destroy( GtkWidget *widget,
                     gpointer   data )
{
    gtk_main_quit ();
}

void *play_gtk(void *args) {
    printf("About to start gtk_main\n");
    gtk_main();
}

int init_video() {
    #define FNAME "short.mpg"

    lyrics = g_array_sized_new(FALSE, FALSE,
sizeof(lyric_t *), 1024);

    /* FFMpeg stuff */
```

```
    AVFrame *pFrame = NULL;
    AVPacket packet;

    AVDictionary *optionsDict = NULL;

    av_register_all();

    if(avformat_open_input(&pFormatCtx, FNAME,
NULL, NULL)!=0) {
    printf("Couldn't open video file called
short.mpg\n");
    return -1; // Couldn't open file
    } else {
    printf("Opened short.mpg\n");
    }

    // Retrieve stream information
    if(avformat_find_stream_info(pFormatCtx,
NULL)<0) {
    printf("Couldn't find stream
information\n");
    return -1; // Couldn't find stream
information
    }

    // Dump information about file onto
```

```
standard error
    av_dump_format(pFormatCtx, 0, FNAME, 0);

    // Find the first video stream
    videoStream=-1;

    int i;
    for(i=0; i<pFormatCtx->nb_streams; i++)
    if(pFormatCtx->streams[i]->codec-
>codec_type==AVMEDIA_TYPE_VIDEO) {
        videoStream=i;
        break;
    }
    if(videoStream==-1)
    return -1; // Didn't find a video stream

    for(i=0; i<pFormatCtx->nb_streams; i++)
    if(pFormatCtx->streams[i]->codec-
>codec_type==AVMEDIA_TYPE_AUDIO) {
        printf("Found an audio stream too\n");
        break;
    }


    // Get a pointer to the codec context for
the video stream
    pCodecCtx=pFormatCtx->streams[videoStream]-
```

```
>codec;

    // Find the decoder for the video stream
    pCodec=avcodec_find_decoder(pCodecCtx-
>codec_id);
    if(pCodec==NULL) {
    fprintf(stderr, "Unsupported codec!\n");
    return -1; // Codec not found
    }

    // Open codec
    if(avcodec_open2(pCodecCtx, pCodec,
&optionsDict)<0) {
    printf("Could not open codec\n");
    return -1; // Could not open codec
    }

    /* GTK stuff now */

    /* GtkWidget is the storage type for
widgets */
    GtkWidget *window;
    GtkWidget *button;
    GtkWidget *lyrics_box;


    /* This is called in all GTK applications.
```

```
Arguments are parsed
     * from the command line and are returned
to the application. */
    gtk_init (0, NULL);

    /* create a new window */
    window = gtk_window_new
(GTK_WINDOW_TOPLEVEL);

    /* When the window is given the "delete-
event" signal (this is given
     * by the window manager, usually by the
"close" option, or on the
     * titlebar), we ask it to call the
delete_event () function
     * as defined above. The data passed to the
callback
     * function is NULL and is ignored in the
callback function. */
    g_signal_connect (window, "delete-event",
              G_CALLBACK (delete_event), NULL);

    /* Here we connect the "destroy" event to a
signal handler.
     * This event occurs when we call
gtk_widget_destroy() on the window,
     * or if we return FALSE in the "delete-
```

```
event" callback. */
    g_signal_connect (window, "destroy",
              G_CALLBACK (destroy), NULL);

    g_signal_connect (window, "realize",
G_CALLBACK (realize_cb), NULL);

    /* Sets the border width of the window. */
    gtk_container_set_border_width
(GTK_CONTAINER (window), 10);

    lyrics_box = gtk_vbox_new(TRUE, 1);
    gtk_widget_show(lyrics_box);

    /*
      char *str = "     ";
      lyric_labels[0] = gtk_label_new(str);
      str =  "World";
      lyric_labels[1] = gtk_label_new(str);
    */

    image = gtk_image_new();

    //image_drawable = gtk_drawing_area_new();
    //gtk_widget_set_size_request (canvas, 720,
480);
    //gtk_drawing_area_size((GtkDrawingArea *)
```

```
image_drawable, 720, 480);

    //gtk_widget_show (lyric_labels[0]);
    //gtk_widget_show (lyric_labels[1]);

    gtk_widget_show (image);

    //gtk_box_pack_start (GTK_BOX (lyrics_box),
lyric_labels[0], TRUE, TRUE, 0);
    //gtk_box_pack_start (GTK_BOX (lyrics_box),
lyric_labels[1], TRUE, TRUE, 0);
    gtk_box_pack_start (GTK_BOX (lyrics_box),
image, TRUE, TRUE, 0);
    //gtk_box_pack_start (GTK_BOX (lyrics_box),
canvas, TRUE, TRUE, 0);
    //gtk_box_pack_start (GTK_BOX (lyrics_box),
image_drawable, TRUE, TRUE, 0);

    /* This packs the button into the window (a
gtk container). */
    gtk_container_add (GTK_CONTAINER (window),
lyrics_box);

    /* and the window */
    gtk_widget_show (window);

    /* All GTK applications must have a
```

```
gtk_main(). Control ends here
     * and waits for an event to occur (like a
key press or
     * mouse event). */
    pthread_t tid_gtk;
    pthread_create(&tid_gtk, NULL, play_gtk,
NULL);
    //gtk_main ();
}

/*
 * interface_<id>_loader();
 */
ControlMode *interface_v_loader(void)
{
    return &video_ctl;
}
```

Background video with TiMidity as library

The code for this follows the same structure as the code in the MIDI TiMidity chapter. It is is in the file `gtkkaraoke_player_video_pango.` :

```
#include <stdio.h>
#include <stdlib.h>

#include "sysdep.h"
#include "controls.h"


extern ControlMode  *video_ctl;
extern ControlMode  *ctl;

static void init_timidity() {
    int err;

    timidity_start_initialize();

    if ((err =
timidity_pre_load_configuration()) != 0) {
    printf("couldn't pre-load configuration
file\n");
    exit(1);
    }

    err += timidity_post_load_configuration();

    if (err) {
    printf("couldn't post-load configuration
```

```
file\n");
    exit(1);
    }

    timidity_init_player();


    extern int opt_trace_text_meta_event;
    opt_trace_text_meta_event = 1;

    ctl = &video_ctl;
    //ctl->trace_playing = 1;
    //opt_trace_text_meta_event = 1;

}


#define MIDI_FILE "54154.kar"

static void *play_midi(void *args) {
    char *argv[1];
    argv[0] = MIDI_FILE;
    int argc = 1;

    timidity_play_main(argc, argv);

    printf("Audio finished\n");
```

```
    exit(0);
}


int main(int argc, char** argv)
{
    XInitThreads();

    int i;

    /* TiMidity */
    init_timidity();
    play_midi(NULL);


    return 0;
}
```

Background video with TiMidity as front-end

The interface needs to be built as a shared library by

```
if_video.so: video_code.c
        gcc  -fPIC $(CFLAGS) -c -o video_code.o
video_code.c $(LIBS)
        gcc -shared -o if_video.so video_code.o
$(LIBS)
```

TiMidity is then run with options

```
timidity -d. -iv --trace  --trace-text-meta
```

As before, it crashes TiMidity from the Ubuntu distro but works fine with TiMidity built from source in the current Linux environment.

Adding microphone input

At this stage we have a single application which can play a MIDI file, play a background movie and display highlighted lyrics on top of the video. There is no microphone input to sing along.

Singing along can be handled either within this application or by an external process. if we want to include it in the current application then we will have to build a mixer for two audio streams. Java does this in the JavaSound package, but in C we would need to do that ourselves. Now I think that can be done in ALSA, but at present their mixer code is gobbledygook to me.

Jack makes it easy to mix audio - from different *processes* . The earlier section showed how to do that.

My long term goal is to include scoring, etc. I need to split out the GUI code into a process that can also deal with microphone input. The Gtk interface in TiMidity is a "separate process" model, so I'll build on that - when I have Jack under control.

This section is a place-holder for now.

Conclusion

This chapter has shown how to use TiMidity as MIDI player for a Karaoke system. On my laptop it uses less than 60% of CPU with Gtk 3.0, better than the 100% of FluidSynth.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

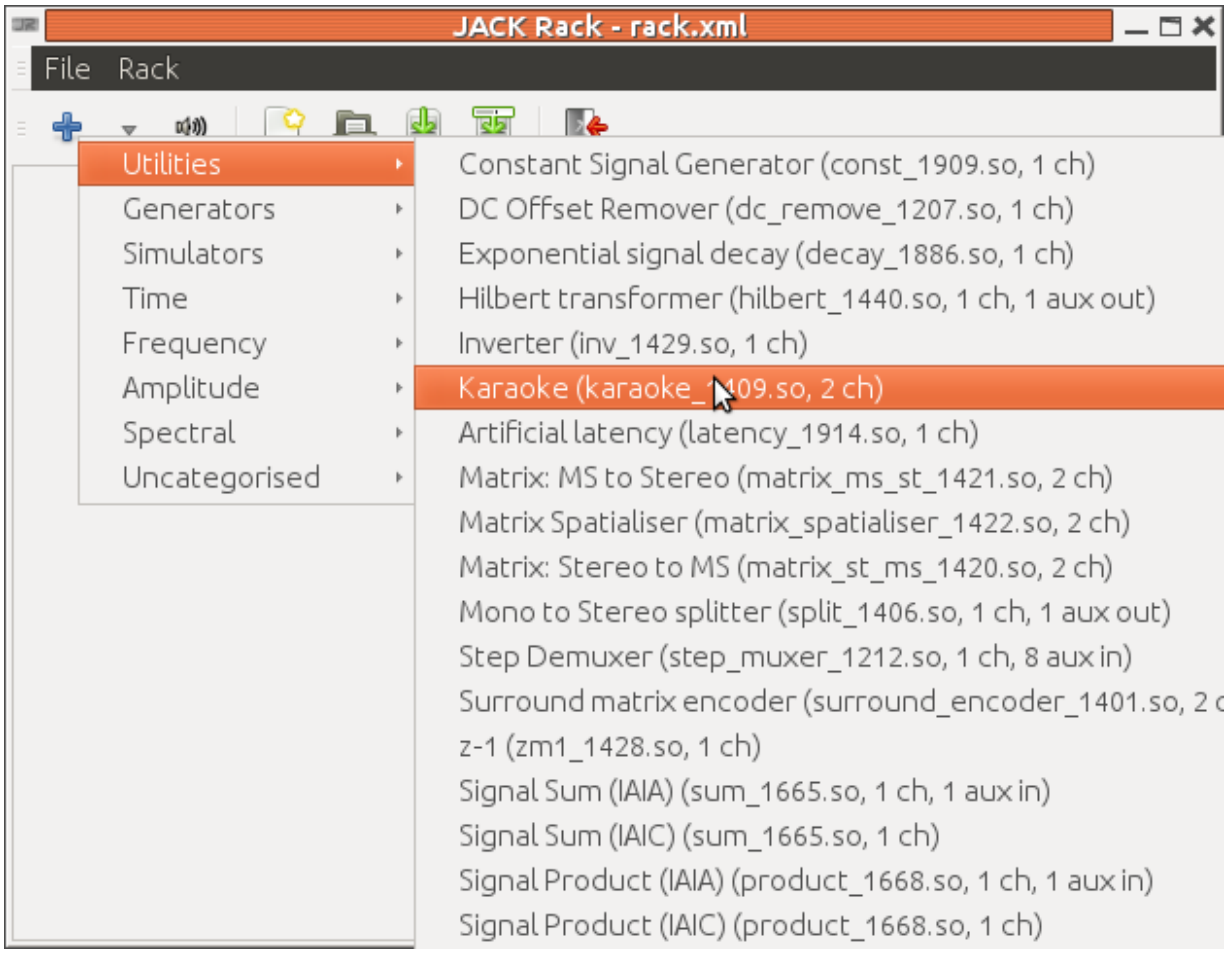
Jack is designed for professional audio. In this chapter we apply the techniques from earlier chapters to building a Karaoke system.

Using jack rack for effects

Karaoke takes inputs from MIDI sources and from microphones. These are mixed together. Generally there is an overall volume control but there is usually a volume control for the microphones as well. While the MIDI source should be passed straight through, it is common to apply a reverb effect to the microphones.

These are all effects that can be supplied by LADSPA modules (see the LADSPA chapter). The Jack application `jack-rack` makes these plugins accessible to Jack applications so that LADSPA effects can be applied to Jack pipelines.

It is fairly straighforward to add a module to a session. Click on the "+" button and choose from the enormous menu of effects. For example, choosing the Karaoke plugin from the Utilities menu looks like



Some of the modules that might be relevant are

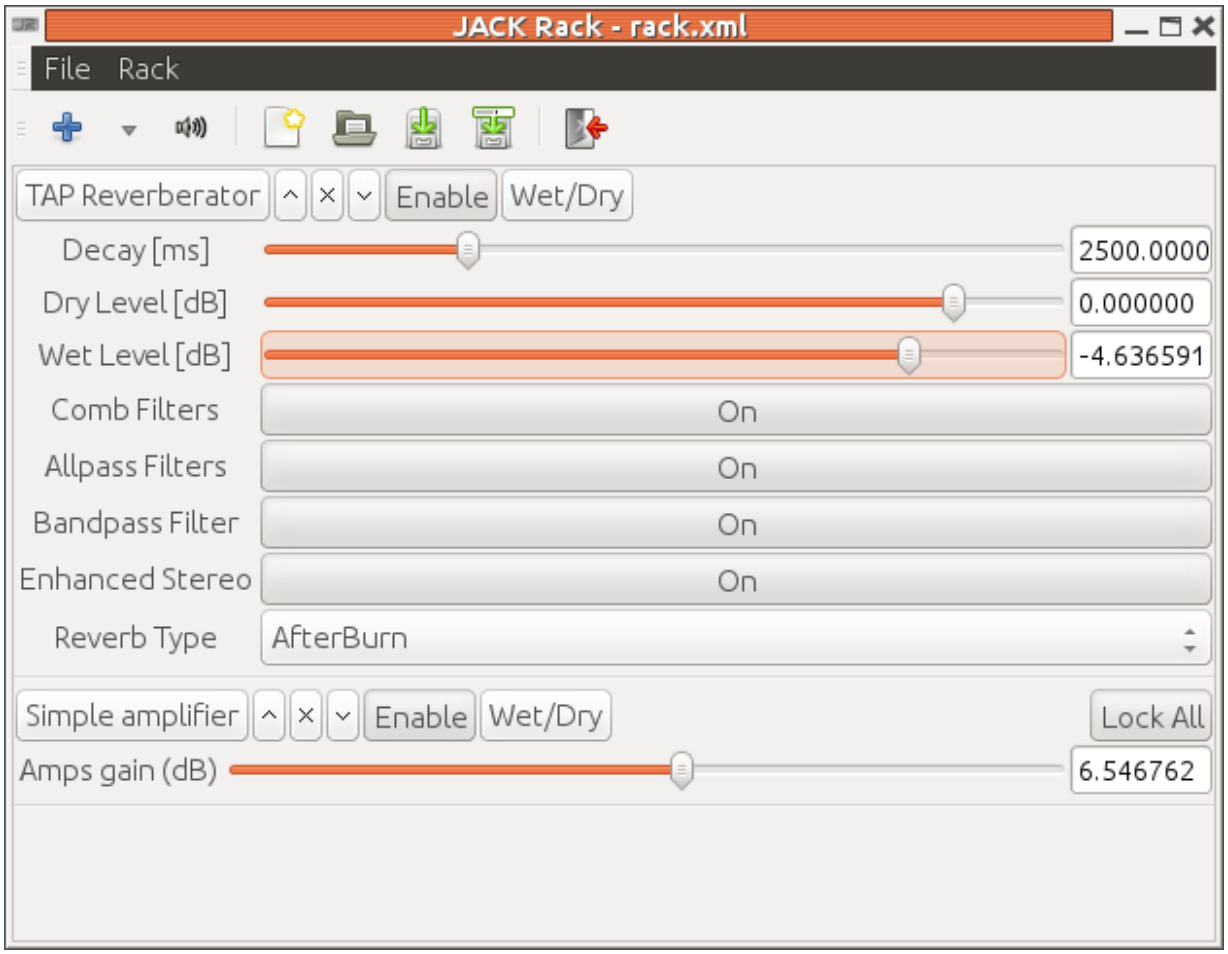
- Karaoke (number 1409) which shows under the Utilities menu. This will attempt to remove centre vocals from a music track.
- There are a number of reverb modules which show under the Simulators -> Reverb menus
- GVerb
- Plate reverb
- TAP reverberator (from the TAP plugins)

The TAP reverberator seems to be the most full featured (but is not guaranteed to be realtime).

- There are a number of amplifiers under Amplitude -> Amplifiers

Multiple modules may be applied in a single `jack-rack` application, and multiple applications can be run. For example, to apply a volume control to a microphone and then apply reverb before sending to the speakers can be done by

adding the TAP reverberator and then one of the amplifiers. This looks like Jack Rack with reverb and amplifier plugins



I'm running this on a USB Sound Blaster TruStudioPro. This is only 16-bits and I can't seem to find a suitable Jack hardware configuration. So I'm running Jack by hand using a plug device, which Jack complains about but works anyway:

```
jackd -dalsa -dplughw:2 -r 48000
```

Although `gladish` can see it under its Jack configuration menu, I haven't managed to get `gladish` to accept the SoundBlaster as a setting - so far I can only manage to get Jack running under as a plug device and gladish keeps swapping it back to a hardware device.

`qjackctl` does an okay job of saving and restoring sessions, starting `jack-rack` with its correct plugins and their settings, and linking it to the correct capture and playback ports.

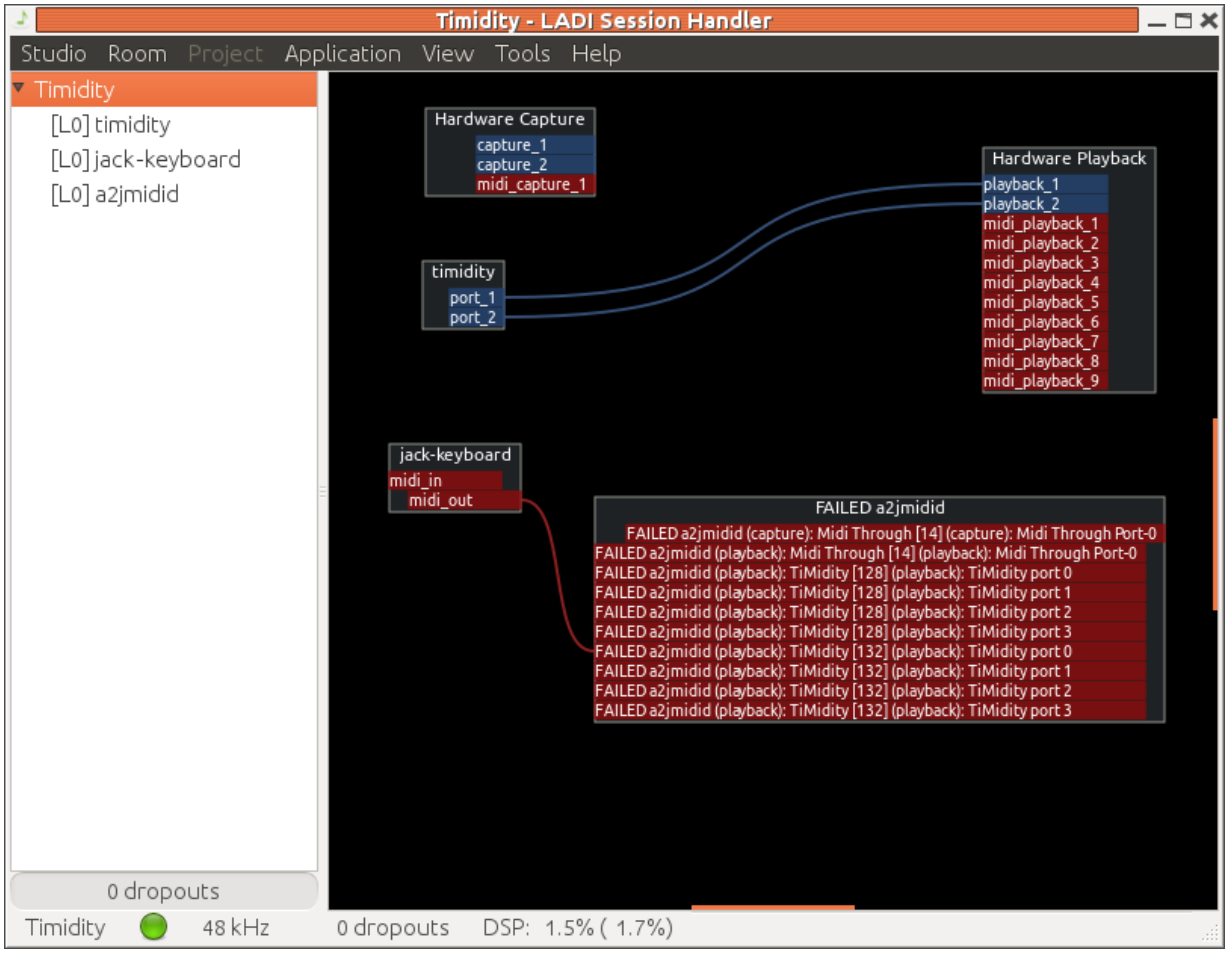
Playing MIDI

The major synthesizer engines TiMidity and FluidSynth will output to ALSA devices. To bring them into the Jack world, Jack needs to be started with the `-Xseq` option or for `a2jmidid` to be run.

We can try to manage the connections using the Jack session manager (e.g. `qjackctl`). But this hits a snag using the MIDI synthesizers such as TiMidity or FluidSynth since they assume a PulseAudio output rather than Jack output. Restoring a session fails to restore the synthesizer with Jack output.

We can try to manage the connections using LADSPA. Unfortunately I have so far been unable to manage the Jack server settings using `gladish`. So it starts Jack using the default ALSA settings and doesn't use the `-Xseq` setting to map the ALSA ports to Jack. We need to start `a2jmidid` and then it can succesfully manage a session of e.g `timidity`, `jack_keyboard` and `a2jmidid`.

Even then, the connection diagram looks a mess: LADISH playing MIDI



TiMidity plus Jack Rack

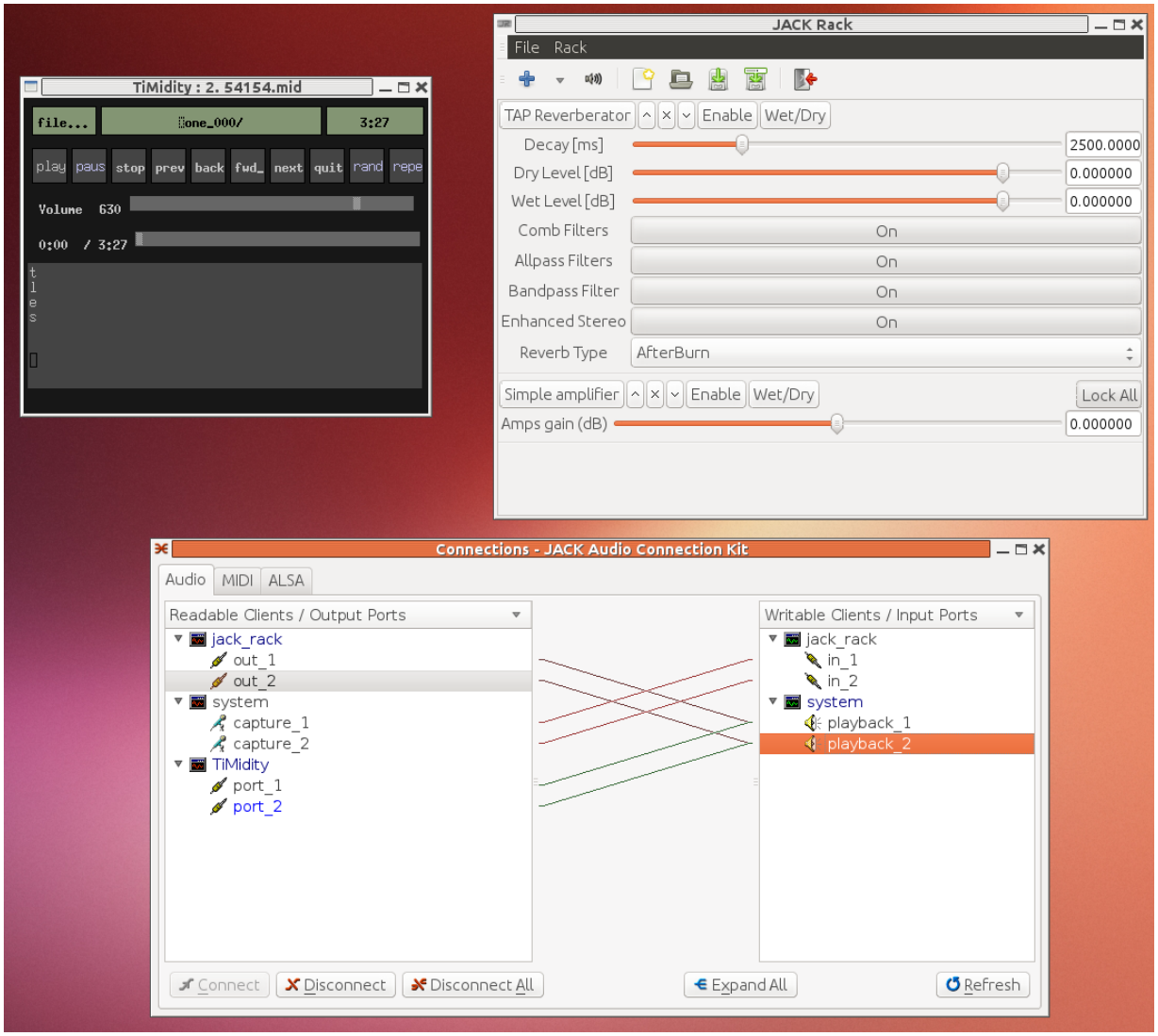
In the chapter on Karaoke TiMidity we used TiMidity with a Jack backend and an Xaw interface to give a basic Karaoke system. We can now improve on that by using Jack Rack effects:

- Run TiMidity with Jack output, an Xaw interface and synchronising the lyrics to sound by

  ```
  timidity -ia -B2,8 -Oj -EFverb=0 --trace --
  trace-text-meta
  ```

- Run Jack rack with TAP reverberator and a volume control installed
- Connect ports using `qjackctl`

The resulting system looks like TiMidity with Jack Rack



Customising TiMidity build

The version of TiMidity from the Ubuntu distro crashes if I try to dynamically load another interface. As the code is stripped, it is not possible to find out why. So to add a new interface we need to build TiMidity from source.

The commands I now use are

```
./configure --enable-audio=alsa,jack \
            --enable-interface=xaw,gtk \
            --enable-server \
            --enable-dynamic
make clean
make
```

An interface with key, say 'k', can then be run with Jack output by

```
timidity -d. -ik -Oj --trace  --trace-text-meta
54154.mid
```

Playing MP3+G with Jack Rack pitch shifting

The player `vlc` will play MP3+G files. Often the MP3+G is a zipped files containing both an MP3 file and CDG file with the same root. This must be unzipped, and then can be played by giving `vlc` the MP3 filename:

```
vlc file.mp3
```

This will pick up the CDG file and display the lyrics.

`vlc` can be used with Jack by the `--aout jack` option

```
vlc --aout jack file.mp3
```

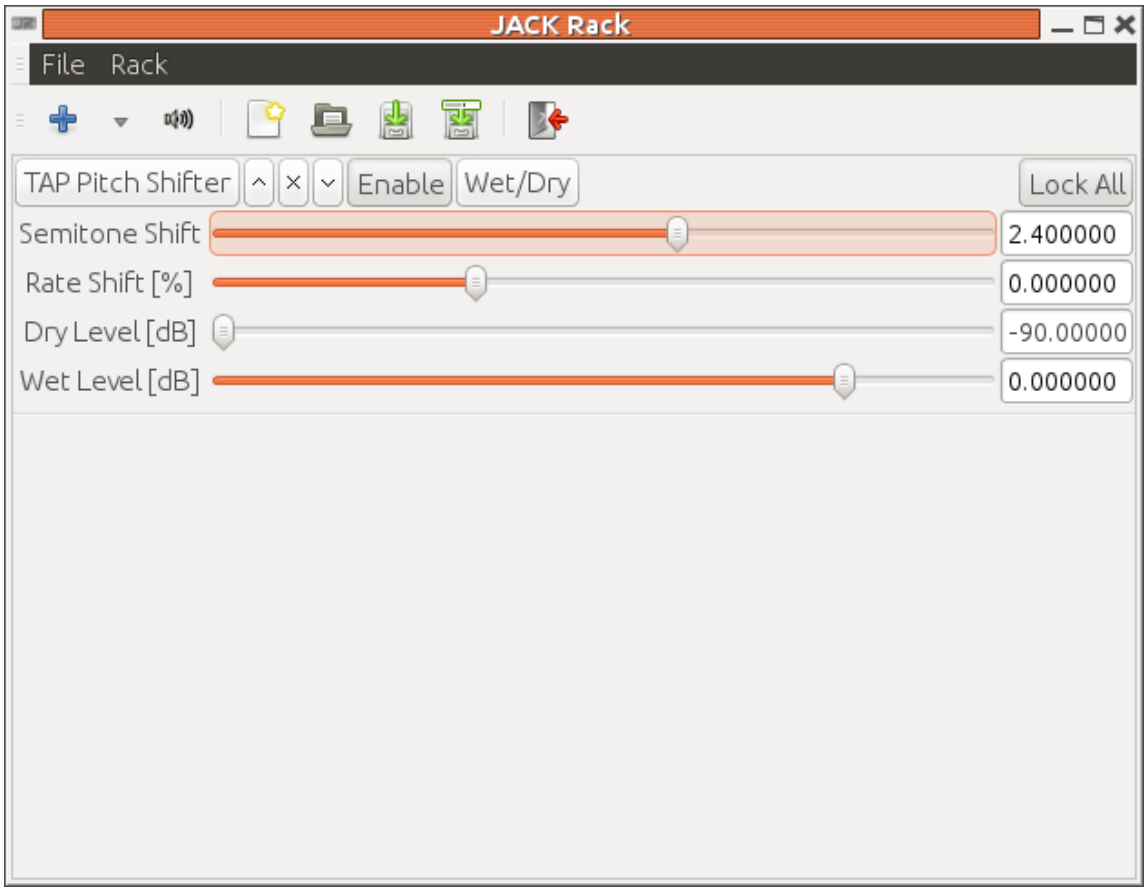
One of the common requests for `vlc` is to have a "pitch control" mechanism. While it should be possible to add LADPSA pitch controls to `vlc`, no-one has got around to it yet. But we can still add LADSPA effects through `jack-rack`.

The steps are

- You may need to stop `pulseaudio` temporarily by e.g `pasuspender cat`
- Start the Jack daemon running as usual by

```
jackd -d alsa
```

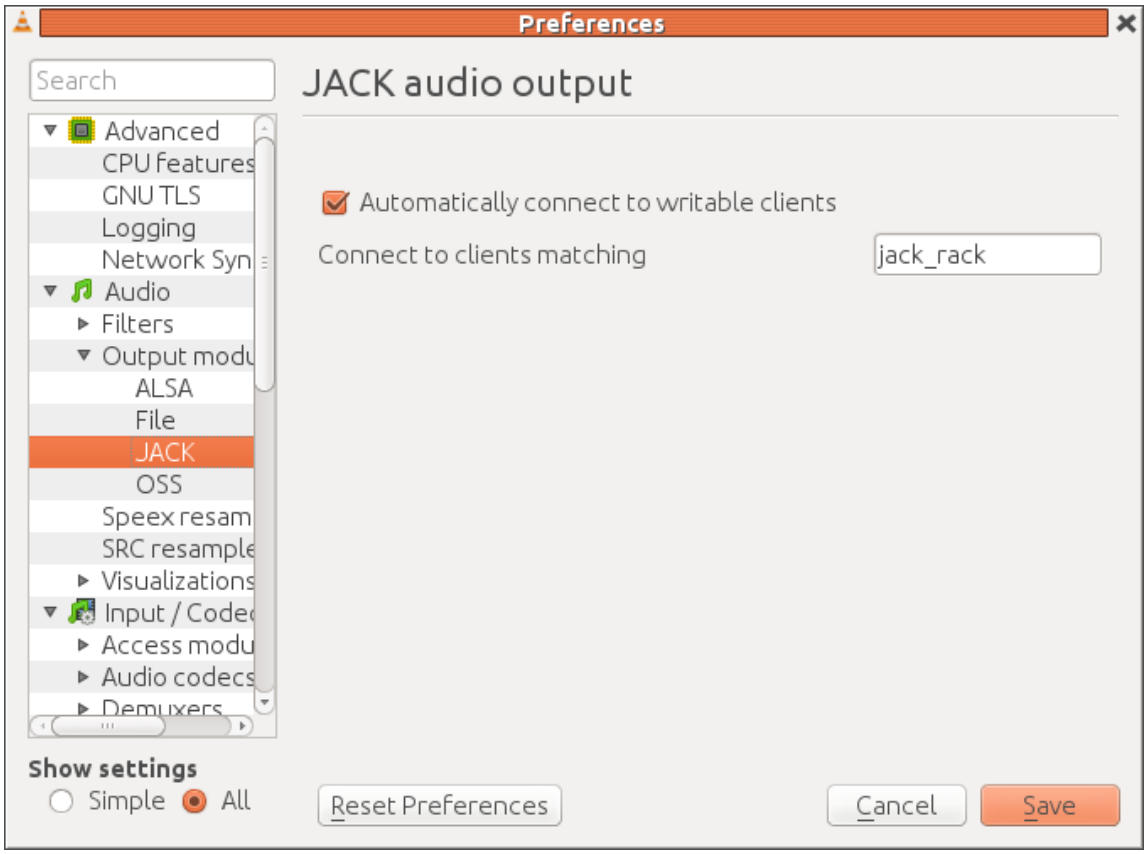
- Start `qjackctl` so you have some control over Jack connections
- Start `jack-rack`. Using the '+' button, select Frequency -> Pitch shifters -> TAP Pitch Shifter. Don't forget to Enable it, so it should look like



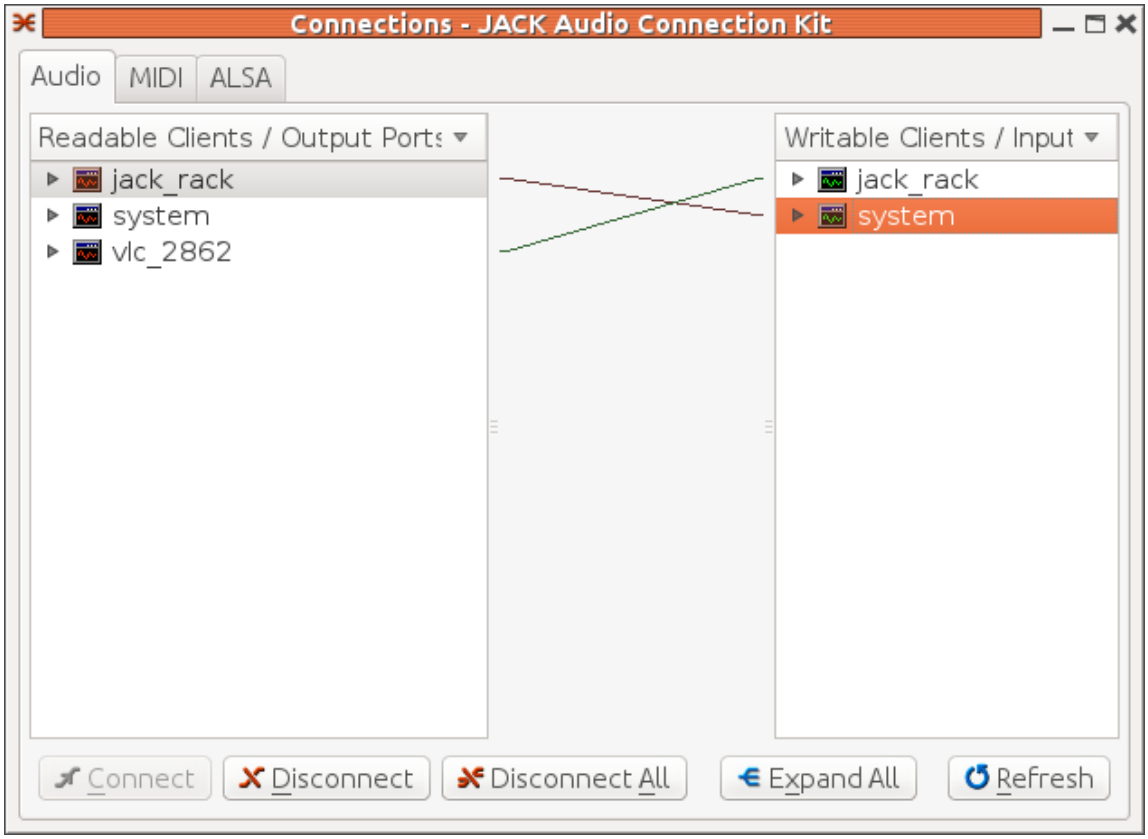
Note that in `qjackctl`, `jack-rack` shows as `jack_rack` - the minus has been replaced by an underscore, and is the proper Jack name of `jack-rack`. Connect the output of `jack-rack` to `system`.

- Now start `vlc --aout jack` so we can set up the correct configuration. Choose Tools -> Preferences, and in "Show settings" set the radio button to "All". Then under Audio -> Output modules -> Jack check "Automatically connect to writable clients" and connect to clients matching "jack_rack"

(note the underscore). This should look like

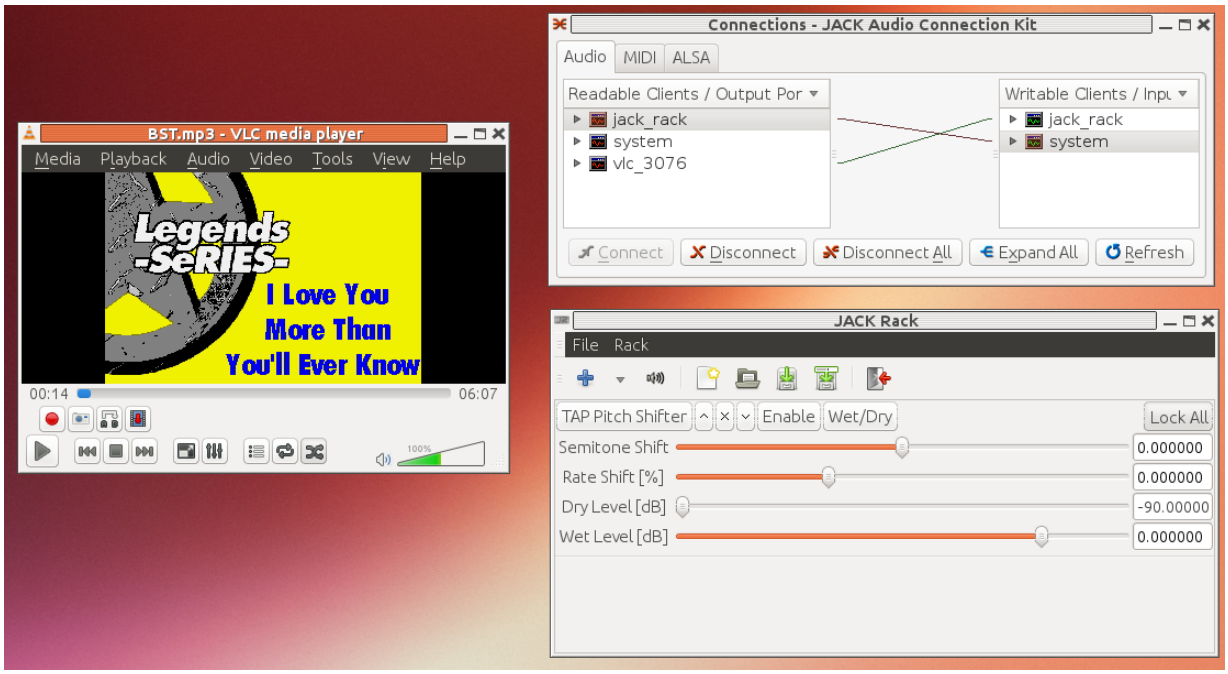


- Next time you start `vlc` by e.g. `vlc --aout jack BST.mp3`, `qjackctl` should show



and the music should play through `jack-rack` where you can adjust the pitch.

The result of this with `vlc` playing MP3 audio through the pitch filter and also showing the CDG video should look like



Conclusion

This chapter has discussed a number of ways of building Jack pipelines to add effects to MIDI and MP3+G files.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

PART 6: STREAMING AUDIO

HTTP is the protocol underlying the Web. The protocol is agnostic to the content it carries - while it was originally designed to carry HTML documents it is now used to transport image files, Postscript documents, PowerPoint files and almost anything. This includes media files, the subject of this book.

HTTP servers

Content is delivered from a Web site by means of HTTP servers. The most well-known of these is Apache, but in the Linux world Nginx and Lighttpd are also common. There are a number of proprietary servers as well.

An HTTP server can deliver static files stored on the server, or construct content dynamically from e.g. a database connection.

HTTP clients

HTTP Browsers

Point the browser to the URL of an audio file and it will pass the content to a helper which will attempt to play the file. The browser will choose the helper based on the file extension of the URL, or based on the Content-Type of the file as delivered in the HTTP header from the HTTP server.

mplayer

MPlayer is HTTP-aware: just give the URL of the file

```
mplayer
http://localhost/audio/enigma/audio_01.ogg
```

VLC

VLC is also HTTP-aware: just give the URL of the file

```
vlc http://localhost/audio/enigma/audio_01.ogg
```

Streaming vs downloading

If you download a file from the Web, then you can play it once it has finished downloading. This means that play is delayed until the entire file has been saved into the local file file system. On the other hand, since it is now local it can play without worrying about network delays. A simple shell script to illustrate this is

```
wget -O tmp
http://localhost/audio/enigma/audio_01.ogg
mplayer tmp
rm tmp
```

The alternative is to read the resource from the Web and hand it as it is received to a player, using some sort of pipeline. This is fine as long as the pipeline is large enough to buffer enough of the resource that it can cope with network delays. It is illustrated by

```
wget -O -
http://localhost/audio/enigma/audio_01.ogg |
mplayer -
```

(Yes, I know, mplayer can stream URLs directly - I'm using this way it for the point I'm making.).

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

Shoutcast is a serious piece of server software for internet streaming of audio, which has set the standard for streaming. Icecast is the serious open source competitor which is just as good in quality and superior as open source. To the receiver of a stream, icecast is just an HTTP server. The backend is the interesting part, as Icecast uses the shoutcast protocol to receive audio from a variety of sources such as online radio, microphones or playlists.

Icecast

See Icecast 2 Documentation Table of Contents

IceS

IceS is one of the ways that Icecast can get its audio streams. See IceS v2.0 Documentation

---

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

Resources

- DLNA Home page

Introduction

DLNA (Digital Living Network Alliance) is designed for sharing digital media such as photos, audio and video in a home network. It is built on top of the UPnP (Universal Plug 'n Play) protocol suite. This in turn is built on some of the uglier of the internet standards such as SOAP. UPnP itself compounds this poor choice of base technologies by using what can only be described as an appallingly badly engineered hack in order to handle media information: with its most complex data type being a string, UPnP buries complete XML documents inside these strings, so that one XML document contains another XML document as embedded string. Oh dear, better quality engineers could surely have come up with better solutions than this!

UPnP is open in that it can describe many different home network devices and formats of data. DLNA restricts this to only a few "approved" types, and then makes the specification private, only available after paying a fee.

Despite all this, an increasing number of devices are "DLNA enabled" such as TVs, BluRay players, etc. It seems like DLNA is here to stay :-(.

Matthew Panton in DLNA for media streamers--what does it all mean? points out some further issues with DLNA. mainly relating to supported file formats. The truth of his comments are illustrated by my latest purchase of a Sony BDP-S390 BluRay player. It supports LPCM (.wav) as required, but out of the optional MP3, WMA9, AC-3, AAC, ATRAC3plus it supports MP3, AAC/HE-AAC (.m4a) and WMA9 Standard (.wma). And of course, Ogg is nowhere in any of the DLNA lists.

DLNA open source projects

See DLNA Open Source Projects I have used the Cybergarage Java client and server.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

From theFlumotion site"Flumotion Streaming Software allows broadcasters and companies to stream content live and on demand in all the leading formats from a single server. Flumotion also offers a Streaming Platform and WebTV which reduce workflow and costs by covering the entire streaming value chain. This end-to-end yet modular solution includes signal acquisition, encoding, multi-format transcoding, streaming of contents and state-of-the art interface design. The media back office allows for advanced content management and optimal monetisation through rich media advertising."

PART 7: MISCELLANEOUS

Android is a Linux-based system from Google now found on phones, tablets and various miscellaneous devices.

Resources

- MediaPlayer API
- Media Playback guide
- Android Supported Media Formats
- Android audio streaming with OpenSL ES and the NDK
- Google I/O 2012 - New Low-Level Media APIs in Android

Identifying devices

Android doesn't have a means of listing the input and output devices. Instead there are methods to query the state of various assumed or possible devices. The page Dealing with Audio Output Hardware gives the code using `AudioManager` methods

```
if (isBluetoothA2dpOn()) {
    // Adjust output for Bluetooth.
} else if (isSpeakerphoneOn()) {
    // Adjust output for Speakerphone.
} else if (isWiredHeadsetOn()) {
    // Adjust output for headsets
} else {
    // If audio plays and noone can hear it, is
it still playing?
}
```

My Android experience

I have an Android device designed to be hooked up to a TV (or similar) and used as a media player, browser, etc. It is branded in various ways: in Australia it was sold by Kogan as the Agora Adroid TV, and is available in other countries under different names while the OEM appears to be jointech.com.hk selling it as the JTV220 Android TV.

This device is stuck on Android 2.2 and the OEM has not released any upgrades. Android 2.2 was not the greatest for media support and will not - for example - stream Ogg Vorbis audio files from HTTP servers.

I also have an Asus TF101 Transformer running 4.0.3. Media support is much more advanced.

The list of supported types is Android Supported Media Formats .

Playing files

An app can get media from various sources

- Packaged with the app
- From the MediaStore
- From an external source such as an SD card
- Streamed from the network

I'm not so much interested in the first two. They are discussed in books such as "Pro Android Media Developing Graphics, Music, Video, and Rich Media Apps for Smartphones and Tablets" by Shawn Van Every from APress.

An SD card on my ASUS Transformer is mounted into `/Removable/SD`. I put audio files into a subdirectory `Music`. Files from here can be played using an Android `MediaPlayer`. There is a state machine model for setting up the media player to play files: here it consists of setting a data source and then calling `prepare()` and `start()`. The only wrinkle in this is actually finding the path to the data file: in the Android, the `File` class doesn't like a path with embedded '/'. So you have to walk the file path one directory at a time till you get to the file itself. Note that the ASUS Transformer has an

internal SD card, and the "recommended" way of getExternalStorageDirectory returns the internal card and *not* the external one!

I'm not going to go through any of how to set up an environment such as Eclipse or how to build projects. Once done, an app can be "exported" to an SD card and installed from the file browser. A player for a single file on the SD disk looks like

```
package jan.newmarch.DiskPlayer;

import java.io.File;

import android.media.AudioManager;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.app.Activity;
import android.app.AlertDialog;
import android.view.Menu;

public class DiskPlayerActivity extends
Activity {

    private String filePath =
"/Removable/SD/Music/audio_01.ogg";

    @Override
    protected void onCreate(Bundle
savedInstanceState) {
    super.onCreate(savedInstanceState);

setContentView(R.layout.activity_disk_player);
    try {
        String[] array=filePath.split("/");
        File f = null;
```

```
        for(int t = 0; t < array.length - 1;
t++)
        {
            f = new File(f, array[t]);
        }
        File songFile=new File(f,
array[array.length-1]);

        MediaPlayer player = new MediaPlayer();

player.setAudioStreamType(AudioManager.STREAM_MUS



player.setDataSource(songFile.getAbsolutePath());

        player.prepare()    ;
        player.start();
    } catch (Exception e) {
        AlertDialog.Builder builder = new
AlertDialog.Builder(this);
        builder.setMessage(e.toString());
        builder.create();
        builder.show();
    }
    }

    @Override
```

```
    public boolean onCreateOptionsMenu(Menu
menu) {
    // Inflate the menu; this adds items to the
action bar if it is present.

getMenuInflater().inflate(R.menu.activity_disk_pl
 menu);
    return true;
    }

}
```

This app requires access to read the SD card. In the AndroidManifest file this is given by

```
<uses-permission
android:name="android.permission.READ_EXTERNAL_ST
```

Streaming audio

The `MediaPlayer` will take the URL of an audio file - in some versions of Android and for some file types. This is sad, particularly for me since Android 2.2 won't stream Ogg files and my media player can't be upgraded from 2.2.

When an app can't stream a file from the web, we have to do silly tricks like downloading it locally and then playing it. That code looks like

```
package jan.newmarch.HttpPlayer;

import jan.newmarch.HttpPlayer.R;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.net.URL;

import android.media.AudioManager;
import android.media.MediaPlayer;
import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.Context;
import android.view.Menu;

public class HttpPlayerActivity extends
Activity {

    private String uriStr =
"http://192.168.1.9/music/test.ogg";
    private HttpPlayerActivity activity;
    private MediaPlayer player;
```

```
    @Override
    protected void onCreate(Bundle
savedInstanceState) {
        super.onCreate(savedInstanceState);

setContentView(R.layout.activity_http_player);

        activity = this;
        try {
            // Try the URL directly (ok for
Android 3.0 upwards)
            tryMediaPlayer(uriStr);
        } catch(Exception e) {
            // Try downloading the file and
then playing it - needed for Android 2.2
            try {
                downloadToLocalFile(uriStr,
"audiofile.ogg");
                File localFile =
activity.getFileStreamPath("audiofile.ogg");

tryMediaPlayer(localFile.getAbsolutePath());
            } catch(Exception e2) {
                System.out.println("File error
" + e2.toString());
                AlertDialog.Builder builder =
```

```
new AlertDialog.Builder(this);

builder.setMessage(e.toString());
                builder.create();
                builder.show();
            }
        }
    }

    private void downloadToLocalFile(String
uriStr, String filename) throws Exception {
        URL url = new URL(Uri.encode(uriStr,
":/"));
        BufferedInputStream reader =
                new
BufferedInputStream(url.openStream());
        FileOutputStream fOut =
activity.openFileOutput(filename,
                Context.MODE_WORLD_READABLE);
        BufferedOutputStream writer = new
BufferedOutputStream(fOut);

        byte[] buff = new byte[1024];
        int nread;
        while ((nread = reader.read(buff, 0,
1024)) != -1) {
            writer.write(buff, 0, nread);
```

```
        }
        writer.close();
    }

    private void tryMediaPlayer(String uriStr)
throws Exception {
        player = new MediaPlayer();

player.setAudioStreamType(AudioManager.STREAM_MUS

        player.setDataSource(uriStr);
        player.prepare();
        player.start();
    }



    @Override
    public boolean onCreateOptionsMenu(Menu
menu) {
        // Inflate the menu; this adds items to
the action bar if it is present.

getMenuInflater().inflate(R.menu.activity_http_pl
 menu);
        return true;
    }
```

```
}
```

This app requires access to the internet to get the resource. In the AndroidManifest file this is given by

```
<uses-permission
android:name="android.permission.INTERNET"/>
```

Recording audio

There are two ways: using `MediaRecorder` or `AudioRecorder`. The second gives you more control. An example using this is given as Android audio recording, part 2 by Krishnaraj Varma.

This app requires RECORD_AUDIO permission. In the AndroidManifest file this is given by

```
<uses-permission
android:name="android.permission.INTERNET"/>
```

Playing audio from the microphone

Best so far: http://stackoverflow.com/questions/2413426/playing-an-arbitrary-tone-with-android

```
void playSound(){
        final AudioTrack audioTrack = new
AudioTrack(AudioManager.STREAM_MUSIC,
                sampleRate,
AudioFormat.CHANNEL_CONFIGURATION_MONO,
                AudioFormat.ENCODING_PCM_16BIT,
generatedSnd.length,
                AudioTrack.MODE_STATIC);
        audioTrack.write(generatedSnd, 0,
generatedSnd.length);
        audioTrack.play();
    }
```

Similar stuff at http://www.speakingcode.com/2012/01/01/playing-with-android-audiotrack-to-generate-sounds/ Playing with Android AudioTrack to Generate Sounds, but no code

Code at http://masterex.github.com/archive/2012/05/28/android-audio-synthesis.html Android: Crafting a Metronome with Audio Synthesis

Replaying the audio recorded from the microphone to the output device makes use of two objects: an `AudioRecord` to capture data from the microphone and an `AudioTrack` to send it to the output device. The previous program for recording audio just needs to write the data to an `AudioTrack` object instead of to a file.

The code is PlayMicrophone.java:

```
package jan.newmarch.playmicrophone;

import java.io.FileOutputStream;

import android.app.Activity;
import android.media.AudioFormat;
import android.media.AudioManager;
import android.media.AudioRecord;
import android.media.AudioTrack;
import android.media.MediaRecorder;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class PlayMicrophoneActivity extends
Activity {
    private static final int RECORDER_BPP = 16;
    private static final int RECORD_CHANNELS =
AudioFormat.CHANNEL_IN_MONO;
    private static final int PLAY_CHANNELS =
AudioFormat.CHANNEL_OUT_MONO;
    private static final int AUDIO_ENCODING =
AudioFormat.ENCODING_PCM_16BIT;
```

```
    private AudioRecord recorder = null;
    private AudioTrack player = null;
    private int bufferSize = 0;
    private Thread recordingThread = null;
    private boolean isRecording = false;
    private int sampleRate;

    @Override
    protected void onCreate(Bundle
savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        setButtonHandlers();
        enableButtons(false);

        sampleRate =
AudioTrack.getNativeOutputSampleRate(AudioManager

        int bSize1 =
AudioRecord.getMinBufferSize(sampleRate,
                RECORD_CHANNELS,
AUDIO_ENCODING);
        int bSize2 =
AudioTrack.getMinBufferSize(sampleRate,
                PLAY_CHANNELS, AUDIO_ENCODING);
        bufferSize =  (bSize1 > bSize2) ?
```

```
bSize1 : bSize2;
        Toast toast =
Toast.makeText(getApplicationContext(),
                    "Sample rate: " +
sampleRate +
                        " min record buf size:
" + bSize1 +
                            " min play buf
size: " + bSize2,
                    Toast.LENGTH_SHORT);
        toast.show();
        if (bSize1 < 0 || bSize2 < 0) {
            fatal("No microphone or no
speaker");
        }
    }

    private void fatal(String msg) {
        /*
        Toast toast =
Toast.makeText(getApplicationContext(),
                msg,
                Toast.LENGTH_SHORT);
        toast.show();
        */
        System.out.println(msg);
        finish();
```

```
    }

    private void setButtonHandlers() {
        ((Button)
findViewById(R.id.btnStart)).setOnClickListener(b

        ((Button)
findViewById(R.id.btnStop)).setOnClickListener(bt

    }

    private void enableButton(int id, boolean
isEnable) {
        ((Button)
findViewById(id)).setEnabled(isEnable);
    }

    private void enableButtons(boolean
isRecording) {
        enableButton(R.id.btnStart,
!isRecording);
        enableButton(R.id.btnStop,
isRecording);
    }

    private void startRecording() {
        try {
```

```
        recorder = new
AudioRecord(MediaRecorder.AudioSource.MIC,
                sampleRate, RECORD_CHANNELS,
                AUDIO_ENCODING, bufferSize);

        player = new
AudioTrack(AudioManager.STREAM_MUSIC,
                sampleRate, PLAY_CHANNELS,
                AUDIO_ENCODING, bufferSize,
                AudioTrack.MODE_STREAM);
        recorder.startRecording();
        player.play();
        } catch(Exception e) {
            fatal("Can't create recorder or
player");
        }

        isRecording = true;

        recordingThread = new Thread(new
Runnable() {

            // @Override
            public void run() {
                writeAudioDataToSpeaker();
            }
```

```
        }, "AudioRecorder Thread");

        recordingThread.start();
    }

    private void writeAudioDataToSpeaker() {
        byte data[] = new byte[bufferSize];

        int read = 0;

        while (isRecording) {
            read = recorder.read(data, 0,
bufferSize);

            if
(AudioRecord.ERROR_INVALID_OPERATION != read) {
                player.write(data, 0, read);
            }
        }
    }

    private void stopRecording() {
        if (null != recorder) {
            isRecording = false;

            recorder.stop();
            recorder.release();
```

```
            recorder = null;
            recordingThread = null;

            player.stop();
            player.release();
        }
    }

    private View.OnClickListener btnClick = new
View.OnClickListener() {
        @Override
        public void onClick(View v) {
            switch (v.getId()) {
            case R.id.btnStart: {
                enableButtons(true);
                startRecording();
                break;
            }
            case R.id.btnStop: {
                enableButtons(false);
                stopRecording();
                break;
            }
            }
            Toast toast =
Toast.makeText(getApplicationContext(),
```

```
                        "Btn clicked handler
called",
                        Toast.LENGTH_SHORT);
                toast.show();
            }
        };

        @Override
        public boolean onCreateOptionsMenu(Menu
menu) {
            // Inflate the menu; this adds items to
the action bar if it is present.

getMenuInflater().inflate(R.menu.activity_main,
menu);
            return true;
        }

    }
```

This app requires RECORD_AUDIO permission. In the AndroidManifest file this is given by

```
<uses-permission
android:name="android.permission.RECORD_AUDIO"/>
```

On my ASUS Transformer with 4.0, the input and output devices both require a minimum buffer of 8192 bytes with a sample rate of 44100hz. The latency is poor: 330 milliseconds and is similar to other reports. It is not usable for realtime record and playback. The new APIs in 4.1 may improve the situation.

MIDI playback

The Android `MediaPlayer` supports the playback of MIDI files. See The state of MIDI support on Android . It uses the Sonivox EAS (Embedded Audio Synthesis).

OpenMAX

The audio/video components of Android systems are Java layered on top of OpenMAX - a C-based system for low-end devices. This is partly covered in the chapter OpenMAX and and OpenSL . The links using NDK will be explored inlater editions of this book.

Conclusion

Android offers a Java API for dealing with audio. It suffers from version dependencies and the failure to have an open model for the ROMs means that you can be stuck with an old version of Android which you can't upgrade.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

The Raspberry Pi (RPi) is a low cost Linux computer developed with the intention of improving the background of students entering university computer science courses by giving them a good, cheap environment in which to play. And its true! I've got a bunch of colleagues at work, well into middle-age who have leapt upon it to play with. So far their kids haven't had a look-in though...

Resources

- Why is my Audio (Sound) Output not working?
- Raspberry Pi github sources
- RPi VideoCore APIs
- Linux Gizmos reports on many System on a Chip (SoCs) including the RPi
- Tutorial: VLC with hardware acceleration on Raspberry Pi by Helder Araujo Carneiro
- Using direct textures on Android how to get from EGLImage to OpenGL texture
- Raspberry Pi and realtime, low-latency audio
-
-

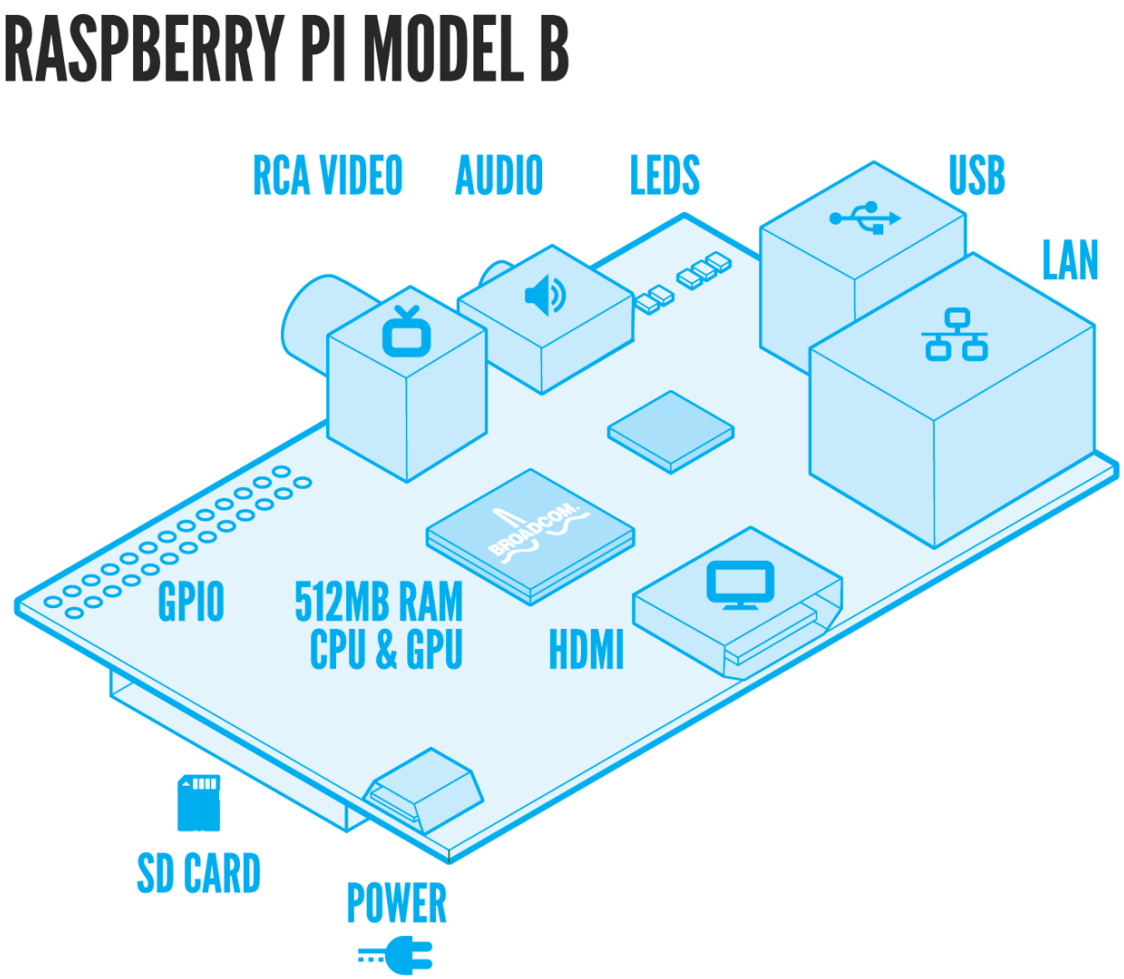
Introduction

Hardware

The Raspberry Pi (RPi) Model B has 512Mb RAM, 2 USB ports and an Ethernet port. It has HDMI, and analogue audio and video outputs. From the FAQ

> The SoC is a Broadcom BCM2835. This contains an ARM1176JZFS, with floating point, running at 700Mhz, and a Videocore 4 GPU. The GPU is capable of BluRay quality playback, using H.264 at 40MBits/s. It has a fast 3D core accessed using the supplied OpenGL ES2.0 and OpenVG libraries.

Graphically it looks like



and physically it looks like

.

The RPi has audio out through the HDMI port and also through an analogue 3.5 mm audio out. There is no audio in. However, there are USB ports, and a USB sound card can be plugged which is recognised by the Linux distros.

The CPU is an ARM CPU. A simple overview of the differences between ARM and Intel instruction sets is given by ARM vs x86 Processors: What's the Difference?

Alternative Single Board Computers

There are many single board computers. Wikipedia has a List of single-board computers . These are all potential alternatives to the RPi. Here is just a quick selection

- **Gumstix** : This is a single board computer that has been around for many years (I got one in 2004). It isn't high powered, but isn't meant to be.
- **Arduino** : The Arduino is designed as a micro-controller for electronic projects. It uses an ARM Cortec-M3 CPU which has even lower specs than the RPi
- **UDOO** : UDOO attempts to marry the best of the RPi and Arduino with two CPUs into a single board computer
- **ODroid** : The ODroid U2 is a higher powered system than the RPI, evaluated by Gigaom . It is about double the price.
- **BeagleBone** : The BeagleBone Black has a slightly better CPU (ARM Cortex-A8) than the RPi, and is a bit more expensive
- **Wandboard**:
- **PandaBoard**:
- **CubieBoard**:

Distros

There are a number of Linux images available from the Raspberry Pi site, and others being developed elsewhere. I'm using the Debian-based image which essentially comes in two forms: with soft float using Debian and with hard float using the FPU, called Raspbian. The hard float image is required for decent sound processing which is heavily floating point dependent. There is a good article benchmarking these against each other: Raspbian Benchmarking – armel vs armhf . Another set of benchmarks is at RPi Performance . Basically, these show that you should use the hard float version if you want good floating point performance, and this is required for audio processing.

ELinux.org maintains a list of RPi Distributions There are many standard Linux distros included here, such as Fedora, Debian, Arch, SUSE, Gentoo and others. The RPi has gained traction as a media centre based on the XBMC media centre, and this is represented by distros such as Raspbmc and OpenElec.

So how does it go with the various audio tools we have been discussing so far? It's a mixed bag.

No sound

I plugged mine into a 29 inch ViewSonic monitor using the HDMI connectors. Initially I got no sound from either the 3.5mm analogue output or the HDMI monitor. This is explained at Why is my Audio (Sound) Output not working? I edited the file `/boot/config.txt` and uncommented the line "hdmi_drive=2". I also used the command

```
sudo amixer cset numid=3 <n>
```

where n is 0=auto, 1=headphones, 2=hdmi to toggle between outputs.

After that sound is fine.

ALSA

The Raspberry Pi uses the ALSA driver snd_bcm2835, and this can manage HDMI output. The command `alsa-info` is not present, but as this is a shell script it can be copied from elsewhere and will run on the RPi. Some of the usual configuration files and commands on a larger distro are missing, but it shows on the soft float Debian distro

```
upload=true&script=true&cardinfo=
!!################################
!!ALSA Information Script v 0.4.61
!!################################

!!Script ran on: Sun Nov  4 02:33:41 UTC 2012



!!Linux Distribution
!!------------------

Debian GNU/Linux wheezy/sid \n \l
PRETTY_NAME="Debian GNU/Linux wheezy/sid"
NAME="Debian GNU/Linux" ID=debian



!!DMI Information
!!---------------

Manufacturer:
Product Name:
Product Version:
Firmware Version:



!!Kernel Information
```

```
!!-------------------

Kernel release:    3.1.9+
Operating System:  GNU/Linux
Architecture:      armv6l
Processor:         unknown
SMP Enabled:       No


!!ALSA Version
!!------------

Driver version:     1.0.24
Library version:    1.0.25
Utilities version:  1.0.25


!!Loaded ALSA modules
!!--------------------

snd_bcm2835


!!Sound Servers on this system
!!----------------------------

No sound servers found.
```

```
!!Soundcards recognised by ALSA
!!-----------------------------

 0 [ALSA           ]: BRCM bcm2835 ALSbcm2835
ALSA - bcm2835 ALSA
                      bcm2835 ALSA



!!PCI Soundcards installed in the system
!!--------------------------------------




!!Advanced information - PCI
Vendor/Device/Subsystem ID's
!!----------------------------------------------
----------




!!Modprobe options (Sound related)
!!---------------------------------

snd_atiixp_modem: index=-2
snd_intel8x0m: index=-2
```

```
snd_via82xx_modem: index=-2
snd_pcsp: index=-2
snd_usb_audio: index=-2



!!Loaded sound module options
!!----------------------------

!!Module: snd_bcm2835
    * :



!!ALSA Device nodes
!!-----------------

crw-rw---T+ 1 root audio 116,  0 Nov  3 22:17
/dev/snd/controlC0
crw-rw---T+ 1 root audio 116, 16 Nov  3 22:17
/dev/snd/pcmC0D0p
crw-rw---T+ 1 root audio 116,  1 Nov  3 22:17
/dev/snd/seq
crw-rw---T+ 1 root audio 116, 33 Nov  3 22:17
/dev/snd/timer

/dev/snd/by-path:
total 0
drwxr-xr-x 2 root root  60 Nov  3 22:17 .
```

```
drwxr-xr-x 3 root root 140 Nov  3 22:17 ..
lrwxrwxrwx 1 root root  12 Nov  3 22:17
platform-bcm2835_AUD0.0 -gt; ../controlC0


!!Aplay/Arecord output
!!--------------------

APLAY

**** List of PLAYBACK Hardware Devices ****
card 0: ALSA [bcm2835 ALSA], device 0: bcm2835
ALSA [bcm2835 ALSA]
  Subdevices: 8/8
  Subdevice #0: subdevice #0
  Subdevice #1: subdevice #1
  Subdevice #2: subdevice #2
  Subdevice #3: subdevice #3
  Subdevice #4: subdevice #4
  Subdevice #5: subdevice #5
  Subdevice #6: subdevice #6
  Subdevice #7: subdevice #7

ARECORD

**** List of CAPTURE Hardware Devices ****
```

```
!!Amixer output
!!--------------

!!-------Mixer controls for card 0 [ALSA]

Card hw:0 'ALSA'/'bcm2835 ALSA'
  Mixer name    : 'Broadcom Mixer'
  Components    : ''
  Controls      : 3
  Simple ctrls  : 1
Simple mixer control 'PCM',0
  Capabilities: pvolume pvolume-joined pswitch
pswitch-joined penum
  Playback channels: Mono
  Limits: Playback -10239 - 400
  Mono: Playback -1725 [80%] [-17.25dB] [on]


!!Alsactl output
!!--------------

--startcollapse--
state.ALSA {
    control.1 {
        iface MIXER
        name 'PCM Playback Volume'
        value -1725
```

```
        comment {
            access 'read write'
            type INTEGER
            count 1
            range '-10239 - 400'
            dbmin -9999999
            dbmax 400
            dbvalue.0 -1725
        }
    }
    control.2 {
        iface MIXER
        name 'PCM Playback Switch'
        value true
        comment {
            access 'read write'
            type BOOLEAN
            count 1
        }
    }
    control.3 {
        iface MIXER
        name 'PCM Playback Route'
        value 2
        comment {
            access 'read write'
            type INTEGER
```

```
            count 1
            range '0 - 2'
        }
    }
}
--endcollapse--


!!All Loaded Modules
!!------------------

Module
snd_bcm2835
snd_pcm
snd_seq
snd_timer
snd_seq_device
snd
snd_page_alloc
evdev


!!ALSA/HDA dmesg
!!--------------

[   16.369584] EXT4-fs (mmcblk0p2): re-mounted.
Opts: (null)
```

```
[   16.750460] ### snd_bcm2835_alsa_probe
c05c88e0 ############### PROBING FOR bcm2835
ALSA device (0):(1) ###############
[   16.776071] Creating card...
--
[   16.834082] Registering card ....
[   16.853728] bcm2835 ALSA CARD CREATED!
[   16.883259] ### BCM2835 ALSA driver init OK
###
[   23.608360] smsc95xx 1-1.1:1.0: eth0: link
up, 100Mbps, full-duplex, lpa 0x45E1
```

Sampled audio players

MPlayer

MPlayer plays fine on the default ALSA modules, for mp3, ogg and wav.

VLC

VLC attempts to play wav files, but it is very broken up on the soft float distro. CPU usage is up around 90% and it is quite unplayable. The hard float distro can play wav, mp3 and ogg.

Alsaplayer

The program `alsaplayer` can attempt to play files in formats such as Ogg-Vorbis and MP3. However, CPU usage hits as high as 80% on the soft float at which point the sound falls to pieces. The hard float distro is okay.

OmxPlayer

The RPi has GPU, and this can be used by omxplayer. It can play Ogg-Vorbis files with only 12% CPU usage and looks to be a good candidate for audio as well as video.

Is it X using the CPU?

Apparently not just X: gnome-player works fine.

Sampled audio capture

The RPi does not have an audio-in or line-in port. I connected my SoundBlaster USB card through a powered USB hub. It shows up with `arecord -l` as

```
**** List of CAPTURE Hardware Devices ****
card 1: Pro [SB X-Fi Surround 5.1 Pro], device
0: USB Audio [USB Audio]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

so that to ALSA it is device `hw:1,0` .

ALSA

The standard program `arecord` works if you get the options correct:

```
arecord -D hw:1,0 -f S16_LE -c 2 -r 48000
tmp.s16
Recording WAVE 'tmp.s16' : Signed 16 bit Little
Endian, Rate 48000 Hz, Stereo
```

The resulting file can be played back using

```
aplay -D hw:1,1 -c 2 -r 48000 -f S16_LE tmp.s16
```

In the chapter on ALSA I gave the source for a program `alsa_capture.c` . When run by

```
alsa_capture hw:1,0 tmp.s16
```

it records PCM data in stereo at 48,000hz.

MIDI players

Timidity

Timidity hits upto 90% CPU. It can't play MIDI files on the RPi properly using the soft float dsitro, but is just about okay on the hard float distro.

To make the RPi usable, in the `timidity.cfg` file uncomment the lines

```
## If you have a slow CPU, uncomment these:
#opt EFresamp=d          #disable resampling
#opt EFvlpf=d            #disable VLPF
#opt EFreverb=d          #disable reverb
#opt EFchorus=d          #disable chorus
#opt EFdelay=d           #disable delay
#opt anti-alias=d        #disable sample anti-
aliasing
#opt EWPVSETOZ           #disable all Midi
Controls
#opt p32a                #default to 32 voices
with auto reduction
#opt s32kHz              #default sample
frequency to 32kHz
#opt fast-decay          #fast decay notes
```

This brings the CPU usage down to about 30%. (Thanks to Chivalry Timber).

PyKaraoke

This only uses 40% of the CPU and plays okay, even giving a GUI on the soft float distro.

Fluidsynth/Qsynth

This just sat there doing not very at all using soft float Debian wheezy out of the box. I spent some time with the FluidSynth team investigating possibilities.

Images

The soft float image is unusable. You need to use the hard float image.

Scheduling

Sometimes fluidsynth complains about not being able to reset the scheduler. Aere Greenway suggests making the following security changes:

> You need to create a file (whose name starts with your user-ID) in the following folder: /etc/security/limits.d For example, my user-ID is aere, so the filename I use is: aere.conf The file needs to contain the following lines:
>
> ```
> aere - rtprio 85
> aere - memlock unlimited
> ```
>
> Except, substitute your user-ID in place of "aere".

This is necessary, but not sufficient It did help with the Raspbian hard-float image, but only a little: some MIDI files played fine while others broke up badly.

Analysis tools

The simplest tool to analyse performance is `perf`. This will give a breakdown of the percentage of CPU usage for the function calls within a program.

`perf` averages the results over an execution period. The MIDI songs I tried only misplayed in parts, not over the whole song. `perf` can however be run as a separate process to sample every second by the command

```
perf top -d 1
```

You can then observe function calls over one second periods.

This didn't reveal much in this instance, but may be helpful in other cases.

The command `pidstat` can be run by e.g.

```
pidstat -C fluidsynth -r -u 5
```

to give CPU and memory usage every 5 seconds. it is similar to `top` but just writes figures for the command to stdout. The output can be massaged using shell scripts and shown as a histogram using GNU octave. This shows that the distortion occurs when CPU usage hits over 100% (don't ask me how!). Memory usage is fine, around 40%.

Non-causes

The following were suggested as causes of the problems, but ultimately discarded

- Fluidsynth can be configured to use either doubles or floats. The default is doubles, and these are slow on Arm chips.

Switching to floats didn't remove the problem peaks in CPU use

- Fluidsynth uses soundfont files and these are quite large: about 40M is typical. Switching to smaller fonts didn't help - memory use was not the problem
- Buffering is small in fluidsynth. The "-z" parameter can be used to make it larger. Buffering was not the problem and changing it didn't help
- A number of operations are known to be expensive in CPU. Fluidsynth supports a number of interpolation algorithms and these can be set using its command interpreter by e.g. "interp 0" to turn off interpolation. Other expensive operations include reverb, polyphony and chorus. Mucking around with any of these in isolation proved fruitless.

Solutions

The two solutions that I have found are

- polyphony=64 and reverb=false; or
- rate=22050

JavaSound

I installed OpenJDK version 6, the default Java install at present. The program `DeviceInfo` was given in the JavaSound Sampled chapter The output from this on the RPi is

```
Mixers:
   PulseAudio Mixer, version 0.02
    Mixer:
org.classpath.icedtea.pulseaudio.PulseAudioMixer@

      Source lines
        interface SourceDataLine supporting 42
audio formats, and buffers of 0 to 1000000
bytes
        interface Clip supporting 42 audio
formats, and buffers of 0 to 1000000 bytes
      Target lines
        interface TargetDataLine supporting 42
audio formats, and buffers of 0 to 1000000
bytes
   ALSA [default], version 1.0.24
    Mixer:
com.sun.media.sound.DirectAudioDevice@12558d6
      Source lines
        interface SourceDataLine supporting 84
audio formats, and buffers of at least 32 bytes
        interface Clip supporting 84 audio
formats, and buffers of at least 32 bytes
      Target lines
   ALSA [plughw:0,0], version 1.0.24
    Mixer:
```

```
com.sun.media.sound.DirectAudioDevice@eb7859
      Source lines
        interface SourceDataLine supporting 8
audio formats, and buffers of at least 32 bytes
        interface Clip supporting 8 audio
formats, and buffers of at least 32 bytes
      Target lines
   Port ALSA [hw:0], version 1.0.24
    Mixer: com.sun.media.sound.PortMixer@fd54d6
      Source lines
      Target lines
        PCM target port
```

Althjough this is using the PulseAudio mixer, pulse audio isn't actually running (at this stage)! So it can only use the ALSA interface.

The program `PlayAudioFile` was given in the JavaSound Sampled chapter. This can play .wav files okay. But it can't play Ogg-Vorbis or MP3 files and throws anUnsupportedAudioFileException.

PulseAudio

PulseAudio installs okay from the repositories and runs with no problems. The output from pulsedevlist is

```
=======[ Output Device #1 ]=======
Description: bcm2835 ALSA Analog Stereo
Name: alsa_output.platform-
bcm2835_AUD0.0.analog-stereo
Index: 0

=======[ Input Device #1 ]=======
Description: Monitor of bcm2835 ALSA Analog
Stereo
Name: alsa_output.platform-
bcm2835_AUD0.0.analog-stereo.monitor
Index: 0
```

Java MIDI

openJDK supports the Java MIDI devices. The program `DeviceInfo` reports

```
MIDI devices:
    Name: Gervill, Decription: Software MIDI
Synthesizer, Vendor: OpenJDK
        Device is a synthesizer
        Open receivers:

        Default receiver:
com.sun.media.sound.SoftReceiver@10655dd

        Open receivers now:

com.sun.media.sound.SoftReceiver@10655dd

        Open transmitters:
        No default transmitter
    Name: Real Time Sequencer, Decription:
Software sequencer, Vendor: Sun Microsystems
        Device is a sequencer
        Open receivers:

        Default receiver:
com.sun.media.sound.RealTimeSequencer$SequencerRe



        Open receivers now:
```

```
com.sun.media.sound.RealTimeSequencer$SequencerRe



        Open transmitters:

        Default transmitter:
com.sun.media.sound.RealTimeSequencer$SequencerTr



        Open transmitters now:

com.sun.media.sound.RealTimeSequencer$SequencerTr

Default system sequencer is Real Time Sequencer
Default system synthesizer is Gervill
```

Programs like DumpSequence work okay. But the SimpleMidiPlayer hits 80% CPU usage and is unusable. So any idea of a Karaoke player using Java on the RPi is simply not on :-(. There is a thread on the Raspberry Pi site discussing problems with sound .

OpenMAX

Audio and video can be played on the Raspberry Pi using the OpenMAX IL toolkit. This has been implemented by Broadcom for their GPU used by the RPi. This is partly covered in the chapter OpenMAX and and OpenSL . More examples will be explored in later editions of this book.

Displaying images

As a step in using the RPi, I decided to try drawing JPEG images into the GPU using OpenMAX. As with all things related to OpenMAX this is far too hard - OpenMAX is one of the worst APIs I have ever had to deal with, and the version on the RPi brings its own set of problems.

The directory `/opt/vc/src/hello_pi/hello_jpeg` contains an example to read in a JPEG file and resize the image. This program works. But it uses the Broadcom convenience functions such as `ilclient_create_component` and I don't want to use those just in case they aen't available elsewhere.

I adapted their code, and it didn't work. Digging deep with the debugger showed a bizarre set of values:

- The JPEG decoder has the OpenMAX label "image_decode"
- This has an input port number 320 on which to receive the original iamge and an output port number 321 to place the decoded output
- When I created a buffer for the input, the buffer showed that port 320 was it's output port!
- The Broadcom example showed that (correctly) it was the input port

Duh!

The problem was the type `OMX_TICKS`. In a 64-bit environment it is a 64-bit integer, otherwise it is a struct of two 32-bit values. I used one version, the Broadcom code on the RPi is using the other.

By looking at the compile environment for the Broadcom example, it uses the defines

```
-DSTANDALONE -D__STDC_CONSTANT_MACROS -
D__STDC_LIMIT_MACROS -DTARGET_POSIX\
-D_LINUX -fPIC -DPIC -D_REENTRANT -
D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64\
-U_FORTIFY_SOURCE -Wall -g -DHAVE_LIBOPENMAX=2
-DOMX -DOMX_SKIP64BIT\
-ftree-vectorize -pipe -DUSE_EXTERNAL_OMX -
DHAVE_LIBBCM_HOST\
-DUSE_EXTERNAL_LIBBCM_HOST -DUSE_VCHIQ_ARM
```

The `OMX_SKIP64BIT` flag is needed to choose the right value for the RPi type. What else is needed for what, I don't know yet :-(

Conclusion

The Raspberry Pi is an exciting new toy to play with. There are many competitors on the block but it has still sold over one million devices. This chapter has covered some of the audio aspects of the device.

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me:

PART 8: CONCLUSION

The dying words...

Where did we start?

The starting point for all this was

- I have two Karaoke machines, each with different capabilities
- I wanted to build a 'best of both worlds' machine from my computer(s)
- My initial attempt using the Java Sound API worked, but suffered from latency, enough to make it unusable
- Atttempts to move any part of this Java solution to low-powered devices like the Raspberry Pi just failed miserably

Where did I get to?

Well, I got part way there. The system ended up consisting of

- I have three types of Karaoke files
- MP3+G files
- KAR format Karaoke files
- KAR files containing the lyrics with the music in .WMA files
- All of these files are stored on my (Linux) server
- For playing the files, I have a CubieBoard2 connected to my TV/HiFi system
- I have a variety of devices to control the selection and playing of files: Linux netbook, Linux laptop, several Android tablets of different dimensions and even one Windows PC
- The Linux and Windows boxes use a common Java program. The Android tablets use...Android

I haven't ended up with a better Karaoke machine. Nevertheless, the system can play all of the file types, showing the lyrics and playing the music. It doesn't score, doesn't show a video in the

background and doesn't show the notes that should be sung against the notes actually sung. Still to come :-(.

The Karaoke files

The Karaoke files are all stored on a server. They could be accessible from Samba, NFS, SSHFS, etc, etc. The only thing I can guarantee across Linux, Windows and Android is HTTP, so I just make the files available through an Apache server.

The files are stored in subdirectories according to their source. For example, the files I ripped from the Sonken disk are in subdirectory `KARAOKE/Sonken`. These files can be accessed through a simple HTTP GET request, given the URL for the file.

Communicating choices

When a controller chooses a song, the URL needs to be communicated to the playing device. For commonality across Linux/Windows/Android clients, a simple TCP connection is used. The bandwidth is not high, so to avoid possible blocks caused by the server crashing and clients holding TCP connections open, each client opens a TCP connection, just sends a single URL across a connection and then closes it.

The Java controller

Information about each song is stored in a file `SongInformation.java` :

```java
package newmarch.songtable;

import java.nio.file.Path;
import java.io.Serializable;
import java.util.Vector;

public class SongInformation implements
Serializable {

    private static final long serialVersionUID
= -7465256749074820597L;

    // Public fields of each song record

    public String path;

    public String index;

    /**
     * song title in Unicode
     */
    public String title;

    /**
     * artist in Unicode
     */
```

```
    public String artist;



    public SongInformation(String path,
                String index,
                String title,
                String artist) {
    this.path = path;
    this.index = index;
    this.title = title;
    this.artist = artist;
    }

    public String toString() {
    return "(" + index + ") " + artist + ": " +
title;
    }

    public Vector<String> toVector() {
    Vector<String> v = new Vector<String>();
    v.add(index);
    v.add(artist);
    v.add(title);
    // will be hidden field
    v.add(path);
```

```
    return v;
    }

    public boolean titleMatch(String pattern) {
    return title.matches("(?i).*" + pattern +
".*");
    }

    public boolean artistMatch(String pattern)
{
    return artist.matches("(?i).*" + pattern +
".*");
    }

    public boolean numberMatch(String pattern)
{
    if (pattern.length() > 0 &&
Character.isDigit(pattern.charAt(0))) {
        // user typed a number only, assume it
is
        // from the Karaoke songs
        pattern = "SK-" + pattern;
    }
    return index.equalsIgnoreCase(pattern);
    }
}
```

This includes methods to perform Boolean pattern matches on the object.

Multiple songs are stored in a `SongTable` which is basically a `Vector` of `SongInformation` objects. This table is serialisable so that it can be stored and then loaded quickly from the HTTP server. The class contains some messy code to initialise the table if it is serialised from a directory or from a list of files. Apart from that, it contains code to build a (smaller) song table from a set of pattern matches. This is useful from returning tables that meet a pattern such as the artist being the Beatles.

The table is given by `SongTable.java`:

```
package newmarch.songtable;

import java.util.Vector;
import java.io.FileInputStream;
import java.io.*;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.SimpleFileVisitor;
import java.nio.file.FileVisitResult;
import java.nio.file.FileSystems;
import java.nio.file.attribute.*;
import java.net.URL;
import java.net.URLConnection;
import java.net.HttpURLConnection;

class Visitor
    extends SimpleFileVisitor<Path> {

    private Vector<SongInformation> songs;

    public Visitor(Vector<SongInformation>
songs) {
    this.songs = songs;
    }
```

```
    @Override
    public FileVisitResult visitFile(Path file,

BasicFileAttributes attr) {
    if (attr.isRegularFile()) {
        String fname =
file.getFileName().toString();
        //System.out.println("Regular file " +
fname);
        if (fname.endsWith(".zip") ||
        fname.endsWith(".mp3") ||
        fname.endsWith(".kar")) {
        String root = fname.substring(0,
fname.length()-4);
        //System.err.println(" root " + root);
        String parts[] = root.split(" - ", 3);
        if (parts.length != 3)
            return
java.nio.file.FileVisitResult.CONTINUE;

        String index = parts[0];
        String artist = parts[1];
        String title = parts[2];

        SongInformation info = new
SongInformation(file.toString(),
```

```
                                    index,
                                    title,
                                    artist);
        songs.add(info);
        }
    }

        return
java.nio.file.FileVisitResult.CONTINUE;
    }
}

public class SongTable {

    private static final String SONG_INFO_ROOT
= "/server/KARAOKE/KARAOKE/";
    private static final String
SONG_STORE_DEFAULT =
"http://192.168.1.101/KARAOKE/SongStore";
    private static final String SONG_STORE_ROOT
= "http://192.168.1.101/KARAOKE/";

    private static Vector<SongInformation>
allSongs;

    public Vector<SongInformation> songs =
    new Vector<SongInformation>  ();
```

```
    public static long[] langCount = new
long[0x23];

    public SongTable(Vector<SongInformation>
songs) {
    this.songs = songs;
    }

    public SongTable(String[] args) throws
java.io.IOException,

java.io.FileNotFoundException {
    if (args.length >= 1) {
        if (args[0].startsWith("-s")) {
        loadTableFromStore(SONG_STORE_ROOT +
args[0].substring(2));
        } else {
        System.err.println("Loading from " +
args[0]);
        loadTableFromSource(args[0]);
        saveTableToStore(args[1]);
        }
    } else {
        loadTableFromStore(SONG_STORE_DEFAULT);
    }
    }
```

```
    public boolean loadTableFromStore(String
urlStr) {
    try {
        /*
        String userHome =
System.getProperty("user.home");
        Path storePath =
FileSystems.getDefault().getPath(userHome,
                                    ".karaoke",
                                    "SongStore");

        File storeFile = storePath.toFile();
        */

        URL url = new URL(urlStr);

        BufferedInputStream reader = new
BufferedInputStream(url

.openConnection().getInputStream());
        ObjectInputStream is = new
ObjectInputStream(reader);
        songs = (Vector<SongInformation>)
is.readObject();
        reader.close();
    } catch(Exception e) {
```

```
        System.err.println("Can't load store
file " + e.toString());
        return false;
    }
    return true;
    }

    public void saveTableToStore(String urlStr)
{
    try {
        URL url = new URL(urlStr);
        URLConnection connection =
url.openConnection();
        HttpURLConnection httpConnection =
(HttpURLConnection) connection;
        httpConnection.setDoOutput(true);
        httpConnection.setRequestMethod("PUT");

        BufferedOutputStream writer = new
BufferedOutputStream(connection.getOutputStream()

        ObjectOutputStream os = new
ObjectOutputStream(writer);
        os.writeObject(songs);
        os.close();
        writer.close();
        int responseCode =
```

```
httpConnection.getResponseCode();
        String responseMsg =
httpConnection.getResponseMessage();
        System.out.println("Saved table to " +
urlStr + " with code " + responseCode +
                   responseMsg + " size " +
songs.size());

    } catch(Exception e) {
        System.err.println("Can't save store
file " + e.toString());
    }
    }

    private void loadTableFromSource(String
source) throws java.io.IOException,
                   java.io.FileNotFoundException
{

    Path path =
FileSystems.getDefault().getPath(source);
    File f = path.toFile();
    if (f.isDirectory()) {
        Visitor pf = new Visitor(songs);
        Files.walkFileTree(path, pf);
    } else if (f.isFile()) {
        loadTableFromFile(path);
```

```
    }
    }

    private void loadTableFromFile(Path path)
throws java.io.IOException {
    BufferedReader in = new BufferedReader(new
FileReader(path.toFile()));
    String fname;

    while ((fname = in.readLine()) != null) {
        // System.out.println(fname);
        if (fname.endsWith(".zip") ||
        fname.endsWith(".mp3") ||
        fname.endsWith(".kar")) {
        // lose extension
        String root = fname.substring(0,
fname.length()-4);
        // lose /.../
        root =
root.substring(root.lastIndexOf('/')+1);
        //System.err.println(" root " + root);
        String parts[] = root.split(" - ", 3);
        if (parts.length != 3)
            continue;

        String index = parts[0];
        String artist = parts[1];
```

```
        String title = parts[2];

        SongInformation info = new
SongInformation(fname,
                                index,
                                title,
                                artist);
        songs.add(info);
        }
    }
    }

    public java.util.Iterator<SongInformation>
iterator() {
    return songs.iterator();
    }

    public SongInformation getSongAt(int index)
{
    if (index < 0 || index >= songs .size()) {
        return null;
    }
    return songs.elementAt(index);
    }

    public SongTable titleMatches( String
pattern) {
```

```
    Vector<SongInformation> matchSongs =
        new Vector<SongInformation>  ();

    for (SongInformation song: songs) {
        if (song.titleMatch(pattern)) {
        matchSongs.add(song);
        }
    }
    return new SongTable(matchSongs);
    }

     public SongTable artistMatches( String
pattern) {
    Vector<SongInformation> matchSongs =
        new Vector<SongInformation>  ();

    for (SongInformation song: songs) {
        if (song.artistMatch(pattern)) {
        matchSongs.add(song);
        }
    }
    return new SongTable(matchSongs);
    }

    public SongTable numberMatches( String
pattern) {
    Vector<SongInformation> matchSongs =
```

```
        new Vector<SongInformation>  ();

    for (SongInformation song: songs) {
        if (song.numberMatch(pattern)) {
        matchSongs.add(song);
        }
    }
    return new SongTable(matchSongs);
    }

    public String toString() {
    StringBuffer buf = new StringBuffer();
    for (SongInformation song: songs) {
        buf.append(song.toString() + "\n");
    }
    return buf.toString();
    }

    public static void main(String[] args) {
    // for testing
    SongTable songs = null;
    try {
        songs = new SongTable(new String[]
{SONG_INFO_ROOT});
    } catch(Exception e) {
        System.err.println(e.toString());
        System.exit(1);
```

```
    }

    System.out.println(songs.artistMatches("Tom
Jones").toString());

    System.exit(0);
    }
}
```

Swing song table UI

There is a principal song table containing all the songs. This may have filters applied to it to show, say, only the Beatles songs. In addition, the visitors to my house have built up a collection of "favourite" songs. Each of these is its own song table. The UI for this is the Swing application `SongTableSwing.java` :

```
package newmarch.songtable;

import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
import javax.swing.*;
import javax.swing.SwingUtilities;
import java.util.regex.*;
import java.io.*;
import java.net.Socket;
import java.net.InetAddress;

public class SongTableSwing extends JPanel {
    private DefaultListModel model = new
DefaultListModel();
    private JList list;
    private static SongTable allSongs;

    private JTextField numberField;
    private JTextField langField;
    private JTextField titleField;
    private JTextField artistField;

    // This font displays Asian and European
characters.
    // It should be in your distro.
```

```
    // Fonts displaying all Unicode are
zysong.ttf and Cyberbit.ttf
    // See
http://unicode.org/resources/fonts.html
    private Font font = new Font("WenQuanYi Zen
Hei", Font.PLAIN, 16);
    // font = new Font("Bitstream Cyberbit",
Font.PLAIN, 16);

    private int findIndex = -1;

    /**
     * Describe <code>main</code> method here.
     *
     * @param args a <code>String</code> value
     */
    public static final void main(final
String[] args) {
    if (args.length >= 1 &&
        args[0].startsWith("-h")) {
        System.err.println("Usage: java
SongTableSwing [song songdir|songfile]");
        System.err.println("  -sSongStore");
        System.exit(0);
    }

    allSongs = null;
```

```
    try {
        allSongs = new SongTable(args);
    } catch(Exception e) {
        System.err.println(e.toString());
        System.exit(1);
    }

    JFrame frame = new JFrame();
    frame.setTitle("Song Table");

    Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
    frame.setSize(800,
((int)screenSize.getHeight())*5/6);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLO



    SongTableSwing panel = new
SongTableSwing(allSongs);
    frame.getContentPane().add(panel);

    frame.setVisible(true);

    JFrame favourites = new JFrame();
    favourites.setTitle("Favourites");
    favourites.setSize(600,
```

```
((int)screenSize.getHeight())*3/4);

favourites.setDefaultCloseOperation(JFrame.EXIT_O



    AllFavourites lists = new
AllFavourites(panel);
    favourites.getContentPane().add(lists);

    favourites.setVisible(true);

    }

    public SongTableSwing(SongTable songs) {

    if (font == null) {
        System.err.println("Can't fnd font");
    }

    int n = 0;
    java.util.Iterator<SongInformation> iter =
songs.iterator();
    while(iter.hasNext()) {
        model.add(n++, iter.next());
        // model.add(n++,
iter.next().toString());
    }
```

```
    BorderLayout mgr = new BorderLayout();

    list = new JList(model);
    // list = new JList(songs);
    list.setFont(font);
    JScrollPane scrollPane = new
JScrollPane(list);

    // Support DnD
    list.setDragEnabled(true);

    setLayout(mgr);
    add(scrollPane, BorderLayout.CENTER);

    JPanel bottomPanel = new JPanel();
    bottomPanel.setLayout(new GridLayout(2,
1));
    add(bottomPanel, BorderLayout.SOUTH);

    JPanel searchPanel = new JPanel();
    bottomPanel.add(searchPanel);
    searchPanel.setLayout(new FlowLayout());

    JLabel numberLabel = new JLabel("Number");
    numberField = new JTextField(5);
```

```
    JLabel langLabel = new JLabel("Language");
    langField = new JTextField(8);

    JLabel titleLabel = new JLabel("Title");
    titleField = new JTextField(20);
    titleField.setFont(font);

    JLabel artistLabel = new JLabel("Artist");
    artistField = new JTextField(10);
    artistField.setFont(font);

    searchPanel.add(numberLabel);
    searchPanel.add(numberField);
    searchPanel.add(titleLabel);
    searchPanel.add(titleField);
    searchPanel.add(artistLabel);
    searchPanel.add(artistField);


titleField.getDocument().addDocumentListener(new
 DocumentListener() {
        public void changedUpdate(DocumentEvent
e) {
            // rest find to -1 to restart any
find searches
            findIndex = -1;
            // System.out.println("reset find
```

```
index");
        }
        public void insertUpdate(DocumentEvent
e) {
            findIndex = -1;
            // System.out.println("reset insert
find index");
        }
        public void removeUpdate(DocumentEvent
e) {
            findIndex = -1;
            // System.out.println("reset remove
find index");
        }
        }
        );

artistField.getDocument().addDocumentListener(new
 DocumentListener() {
        public void changedUpdate(DocumentEvent
e) {
            // rest find to -1 to restart any
find searches
            findIndex = -1;
            // System.out.println("reset insert
find index");
        }
```

```
        public void insertUpdate(DocumentEvent
e) {
            findIndex = -1;
            // System.out.println("reset insert
find index");
        }
        public void removeUpdate(DocumentEvent
e) {
            findIndex = -1;
            // System.out.println("reset insert
find index");
        }
        }
        );

    titleField.addActionListener(new
ActionListener(){
                public void
actionPerformed(ActionEvent e){
            filterSongs();
                }});
    artistField.addActionListener(new
ActionListener(){
                public void
actionPerformed(ActionEvent e){
            filterSongs();
                }});
```

```
    numberField.addActionListener(new
ActionListener(){
                public void
actionPerformed(ActionEvent e){
            filterSongs();
                }});

    JPanel buttonPanel = new JPanel();
    bottomPanel.add(buttonPanel);
    buttonPanel.setLayout(new FlowLayout());

    JButton find = new JButton("Find");
    JButton filter = new JButton("Filter");
    JButton reset = new JButton("Reset");
    JButton play = new JButton("Play");
    buttonPanel.add(find);
    buttonPanel.add(filter);
    buttonPanel.add(reset);
    buttonPanel.add(play);

    find.addActionListener(new ActionListener()
{
        public void actionPerformed(ActionEvent
e) {
            findSong();
        }
        });
```

```
    filter.addActionListener(new
ActionListener() {
        public void actionPerformed(ActionEvent
e) {
            filterSongs();
        }
        });

    reset.addActionListener(new
ActionListener() {
        public void actionPerformed(ActionEvent
e) {
            resetSongs();
        }
        });

    play.addActionListener(new ActionListener()
{
        public void actionPerformed(ActionEvent
e) {
            playSong();
        }
        });

    }
```

```
    public void findSong() {
    String number = numberField.getText();
    String language = langField.getText();
    String title = titleField.getText();
    String artist = artistField.getText();

    if (number.length() != 0) {
        return;
    }

    for (int n = findIndex + 1; n <
model.getSize(); n++) {
        SongInformation info =
(SongInformation) model.getElementAt(n);

        if ((title.length() != 0) &&
(artist.length() != 0)) {
        if (info.titleMatch(title) &&
info.artistMatch(artist)) {
            findIndex = n;
            list.setSelectedIndex(n);
            list.ensureIndexIsVisible(n);
            break;
        }
        } else {
        if ((title.length() != 0) &&
info.titleMatch(title)) {
```

```
            findIndex = n;
            list.setSelectedIndex(n);
            list.ensureIndexIsVisible(n);
            break;
        } else if ((artist.length() != 0) &&
info.artistMatch(artist)) {
            findIndex = n;
            list.setSelectedIndex(n);
            list.ensureIndexIsVisible(n);
            break;

        }
        }

    }
    }

    public void filterSongs() {
    String title = titleField.getText();
    String artist = artistField.getText();
    String number = numberField.getText();
    SongTable filteredSongs = allSongs;

    if (allSongs == null) {
        return;
    }
```

```
    if (title.length() != 0) {
        filteredSongs =
filteredSongs.titleMatches(title);
    }
    if (artist.length() != 0) {
        filteredSongs =
filteredSongs.artistMatches(artist);
    }
    if (number.length() != 0) {
        filteredSongs =
filteredSongs.numberMatches(number);
    }

    model.clear();
    int n = 0;
    java.util.Iterator<SongInformation> iter =
filteredSongs.iterator();
    while(iter.hasNext()) {
        model.add(n++, iter.next());
    }
    }

    public void resetSongs() {
    artistField.setText("");
    titleField.setText("");
    model.clear();
    int n = 0;
```

```
    java.util.Iterator<SongInformation> iter =
allSongs.iterator();
    while(iter.hasNext()) {
        model.add(n++, iter.next());
    }
    }
    /**
     * "play" a song by printing its file path
to standard out.
     * Can be used in a pipeline this way
     */
    public void playSong() {
    String SERVERIP = "192.168.1.110";
    int SERVERPORT = 13000;
    PrintWriter out;

    SongInformation song = (SongInformation)
list.getSelectedValue();
    if (song == null) {
        return;
    }
    System.out.println(song.path);

    try {
        InetAddress serverAddr =
InetAddress.getByName(SERVERIP);
        Socket socket = new Socket(serverAddr,
```

```
SERVERPORT);

        //send the message to the server
        out = new PrintWriter(
                  new BufferedWriter(
                              new
OutputStreamWriter(socket.getOutputStream())),
                  true);
        // Avoid println to socket for Windows
        out.print(song.path + "\n");
        out.flush();
        socket.close();

    } catch (Exception e) {
        System.err.println(e.toString());
    }
    }

    public SongInformation getSelection() {
    return (SongInformation)
(list.getSelectedValue());
    }

    class SongInformationRenderer extends
JLabel implements ListCellRenderer {

    public Component
```

```
getListCellRendererComponent(
                                JList list,
                                Object value,
                                int index,
                                boolean
isSelected,
                                boolean
cellHasFocus) {
        setText(value.toString());
        return this;
    }
    }
}
```

The favourites classes are `AllFavourites.java`:

```
package newmarch.songtable;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.Vector;
import java.nio.file.*;
import java.io.*;
import java.net.URL;

public class AllFavourites extends JTabbedPane
{
    public static String FAVOURITES_DIR =
"http://192.168.1.101:8000/KARAOKE/favourites/";


    private SongTableSwing songTable;
    public Vector<FavouritesInfo> favourites =
new Vector<FavouritesInfo>();

    public AllFavourites(SongTableSwing
songTable) {
    this.songTable = songTable;

    loadFavourites();
```

```
    NewPanel newP = new NewPanel(this);
    addTab("NEW", null, newP);
    }

    private void loadFavourites() {
    //String userHome =
System.getProperty("user.home");
    /*
    Path favouritesPath =
FileSystems.getDefault().getPath(userHome,
                                 ".karaoke",
                                 "favourites");
    */

    try {
        URL url = new URL(FAVOURITES_DIR);
        InputStreamReader reader = new
InputStreamReader(url

.openConnection().getInputStream());
        BufferedReader in = new
BufferedReader(reader);
        String line = in.readLine();
        while (line != null) {
        if (line.startsWith(".")) {
            // ignore .htacess etc
            continue;
```

```
        }
        favourites.add(new FavouritesInfo(null,
line, null));
        line = in.readLine();
        }
        in.close();
        reader.close();

        for (FavouritesInfo f: favourites) {
        // TODO checkout the Songtable
constructors - messy
        f.songTable = new SongTable(new
Vector<SongInformation>());

f.songTable.loadTableFromStore(FAVOURITES_DIR +
                        f.owner);

        Favourites fav = new
Favourites(songTable,
                         f.songTable,
                         f.owner);
        addTab(f.owner, null, fav, f.owner);
        }
    } catch(Exception e) {
        System.out.println(e.toString());
    }
```

```
    /*
    Path favouritesPath =
FileSystems.getDefault().getPath(FAVOURITES_DIR);

    try {
        DirectoryStream<Path> stream =

Files.newDirectoryStream(favouritesPath);
        for (Path entry: stream) {
        int nelmts = entry.getNameCount();
        Path last = entry.subpath(nelmts-1,
nelmts);
        if (last.toString().startsWith(".")) {
            // ignore .htaccess etc
            continue;
        }
        System.err.println("Favourite: " +
last.toString());
        File storeFile = entry.toFile();

        FileInputStream in = new
FileInputStream(storeFile);
        ObjectInputStream is = new
ObjectInputStream(in);
        Vector<SongInformation> favouriteSongs
=
            (Vector<SongInformation>)
```

```
is.readObject();
        in.close();
        for (SongInformation s: favouriteSongs)
{
            System.err.println("Fav: " +
s.toString());
        }

        SongTable favouriteSongsTable = new
SongTable(favouriteSongs);
        Favourites f = new
Favourites(songTable,
                            favouriteSongsTable,
                            last.toString());
        addTab(last.toString(), null, f,
last.toString());
        System.err.println("Loaded favs " +
last.toString());
        }
    } catch(Exception e) {
        System.err.println(e.toString());
    }
    */
    }

    class NewPanel extends JPanel {
    private JTabbedPane pane;
```

```
    public NewPanel(final JTabbedPane pane) {
        this.pane = pane;

        setLayout(new FlowLayout());
        JLabel nameLabel = new JLabel("Name of
new person");
        final JTextField nameField = new
JTextField(10);
        add(nameLabel);
        add(nameField);

        nameField.addActionListener(new
ActionListener(){
            public void
actionPerformed(ActionEvent e){
            String name = nameField.getText();

            SongTable songs = new SongTable(new
Vector<SongInformation>());
            Favourites favs = new
Favourites(songTable, songs, name);

            pane.addTab(name, null, favs);
            }});

    }
```

```
    }

    class FavouritesInfo {
    public SongTable songTable;
    public String owner;
    public Image image;

    public FavouritesInfo(SongTable songTable,
                          String owner,
                          Image image) {
        this.songTable = songTable;
        this.owner = owner;
        this.image = image;
    }
    }

}
```

and

```
package newmarch.songtable;

import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
import javax.swing.*;
import javax.swing.table.*;
import javax.swing.SwingUtilities;
import java.util.regex.*;
import java.io.*;
import java.nio.file.FileSystems;
import java.nio.file.*;
import java.net.Socket;
import java.net.InetAddress;
import java.util.Vector;

public class Favourites extends JPanel {
    private DefaultListModel model = new
DefaultListModel();
    private JList list;

    private JTable table;
    private DefaultTableModel tmodel = new
DefaultTableModel();

    // whose favorites these are
```

```
    private String user;

    // songs in this favourites list
    private final SongTable favouriteSongs;

    // pointer back to main song table list
    private final SongTableSwing songTable;

    // This font displays Asian and European
characters.
    // It should be in your distro.
    // Fonts displaying all Unicode are
zysong.ttf and Cyberbit.ttf
    // See
http://unicode.org/resources/fonts.html
    private Font font = new Font("WenQuanYi Zen
Hei", Font.PLAIN, 16);

    private int findIndex = -1;

    public Favourites(final SongTableSwing
songTable,
                final SongTable favouriteSongs,
                String user) {
    this.songTable = songTable;
    this.favouriteSongs = favouriteSongs;
    this.user = user;
```

```
    if (font == null) {
        System.err.println("Can't find font");
    }


    System.out.println("Favourites for user: "
+ user);

    tmodel.addColumn("ID");
    tmodel.addColumn("Artist");
    tmodel.addColumn("Title");
    tmodel.addColumn("Path");

    int n = 0;
    java.util.Iterator<SongInformation> iter =
favouriteSongs.iterator();
    while(iter.hasNext()) {
        SongInformation info = iter.next();
        if (info == null)
        continue;
        /*
        System.out.println("UID for
SongInformation " +

ObjectStreamClass.lookup(info.getClass()).getSeri
```

```
        */
        model.add(n++, info);

        tmodel.addRow(info.toVector());
    }



    BorderLayout mgr = new BorderLayout();

    list = new JList(model);
    list.setFont(font);

    table = new JTable(tmodel);
    table.setRowSorter(new
TableRowSorter(tmodel));
    // hide column 3

table.removeColumn(table.getColumnModel().getColu

    table.setFillsViewportHeight(true);
    table.setFont(font);

table.getColumnModel().getColumn(0).setPreferredW



table.getColumnModel().getColumn(0).setMaxWidth(1
```

```
table.getColumnModel().getColumn(1).setPreferredW


table.getColumnModel().getColumn(1).setMaxWidth(2

    table.setRowHeight(24);

    //JScrollPane scrollPane = new
JScrollPane(list);
    JScrollPane scrollPane = new
JScrollPane(table);

    setLayout(mgr);
    add(scrollPane, BorderLayout.CENTER);

    JPanel bottomPanel = new JPanel();
    bottomPanel.setLayout(new GridLayout(2,
1));
    add(bottomPanel, BorderLayout.SOUTH);

    JPanel searchPanel = new JPanel();
    bottomPanel.add(searchPanel);
    searchPanel.setLayout(new FlowLayout());

    JPanel buttonPanel = new JPanel();
```

```
    bottomPanel.add(buttonPanel);
    buttonPanel.setLayout(new FlowLayout());

    JButton addSong = new JButton("Add song to
list");
    JButton deleteSong = new JButton("Delete
song from list");
    JButton play = new JButton("Play");

    buttonPanel.add(addSong);
    buttonPanel.add(deleteSong);
    buttonPanel.add(play);

    play.addActionListener(new ActionListener()
{
        public void actionPerformed(ActionEvent
e) {
            playSong();
        }
        });

    deleteSong.addActionListener(new
ActionListener() {
        public void actionPerformed(ActionEvent
e) {
            /*
            SongInformation song =
```

```
(SongInformation) list.getSelectedValue();
            model.removeElement(song);
            favouriteSongs.songs.remove(song);
            */

            int realIndex =
table.convertRowIndexToModel(table.getSelectedRow

            tmodel.removeRow(realIndex);

favouriteSongs.songs.removeElementAt(realIndex);

            saveToStore();
        }
        });

    addSong.addActionListener(new
ActionListener() {
        public void actionPerformed(ActionEvent
e) {
            SongInformation song =
songTable.getSelection();
            //model.addElement(song);
            tmodel.addRow(song.toVector());
            favouriteSongs.songs.add(song);
            saveToStore();
        }
```

```
        });
     }

    private void saveToStore() {
    try {
        /*
        String userHome =
System.getProperty("user.home");
        Path storePath =
FileSystems.getDefault().getPath(userHome,
                                 ".karaoke",
                                 "favourites",
                                 user);
        File storeFile = storePath.toFile();
        */


//favouriteSongs.saveTableToStore("/server/KARAOK
 + user);

favouriteSongs.saveTableToStore(AllFavourites.FAV
 + user);

        /*
        File storeFile = new
File("/server/KARAOKE/favourites/" + user);
        FileOutputStream out = new
```

```
FileOutputStream(storeFile);
        ObjectOutputStream os = new
ObjectOutputStream(out);
        os.writeObject(favouriteSongs.songs);
        os.flush();
        out.close();
        */
    } catch(Exception e) {
        System.err.println("Can't save
favourites file " + e.toString());
    }
    }



    /**
     * "play" a song by printing its file path
to standard out.
     * Can be used in a pipeline this way
     */
    public void playSong() {
    /*
    SongInformation song = (SongInformation)
list.getSelectedValue();
    if (song == null) {
        return;
    }
    System.out.println(song.path.toString());
```

```
    */
    String path = table.getModel().getValueAt(
                    table.convertRowIndexToModel(

table.getSelectedRow()), 3).toString();
    System.out.println(path);

    String SERVERIP = "192.168.1.110";
    int SERVERPORT = 13000;
    PrintWriter out;

    try {
        InetAddress serverAddr =
InetAddress.getByName(SERVERIP);
        Socket socket = new Socket(serverAddr,
SERVERPORT);

        //send the message to the server
        out = new PrintWriter(
                    new BufferedWriter(
                                new
OutputStreamWriter(socket.getOutputStream())),
                    true);

        // Avoid println - on Windows it is
\r\n
        //out.print(song.path + "\n");
```

```
        out.print(path + "\n");
        out.flush();
        socket.close();

    } catch (Exception e) {
        System.err.println(e.toString());
    }

    }



    class SongInformationRenderer extends
JLabel implements ListCellRenderer {

    public Component
getListCellRendererComponent(
                                JList list,
                                Object value,
                                int index,
                                boolean
isSelected,
                                boolean
cellHasFocus) {
        setText(value.toString());
        return this;
    }
```

```
        }
  }
```

The audio player side

On the audio player, a service has to listen for URLs to be played. Each song will be pulled off the file server by an HTTP request (using a tool such as `wget`). This should maintain a queue of requests, playing the next song as the previous one completes. This should show the queue, so is implemented using a Java program showing a Swing list.

The program is `Player.java`:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
import javax.swing.*;
import java.io.*;
import java.net.*;


public class Player extends JFrame {

    private DefaultListModel model = new
DefaultListModel();
    private JList queue;
    private JLabel playingLabel;

    public boolean isPlaying = false;
    private Process songPlayingProcess = null;

    // we have to keep track of the tempo rate,
vlc won't tell us
    private double rate = 1.0;


    public static void main(String[] args) {
    new Player();
    }
```

```
    public Player() {
    Container contentPane = getContentPane();

    setTitle("Song Queue");
        Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
        setSize(400, 400);

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    contentPane.setLayout(new BorderLayout());

    queue = new JList(model);
    contentPane.add(queue,
BorderLayout.CENTER);

        JPanel bottomPanel = new JPanel();
        bottomPanel.setLayout(new GridLayout(2,
1));
        add(bottomPanel, BorderLayout.SOUTH);

    playingLabel = new JLabel(" ");
    bottomPanel.add(playingLabel);

        JPanel controlPanel = new JPanel();
        bottomPanel.add(controlPanel);
```

```
        controlPanel.setLayout(new
FlowLayout());

    JButton stopBtn = new JButton("Stop");
    JButton fasterBtn = new JButton("Faster");
    JButton slowerBtn = new JButton("Slower");

    controlPanel.add(stopBtn);
    controlPanel.add(fasterBtn);
    controlPanel.add(slowerBtn);

    stopBtn.addActionListener(new
ActionListener(){
                public void
actionPerformed(ActionEvent evt){
            if (songPlayingProcess != null) {
            PrintStream writer = new
PrintStream(songPlayingProcess.getOutputStream())

            writer.println("quit");
            writer.flush();
            }
        }
        });


    fasterBtn.addActionListener(new
```

```
ActionListener(){
                public void
actionPerformed(ActionEvent evt){
            if (songPlayingProcess != null) {
            PrintStream writer = new
PrintStream(songPlayingProcess.getOutputStream())

            rate += 0.03;
            writer.println("rate " + rate);
            writer.flush();
            }
        }
        });

    slowerBtn.addActionListener(new
ActionListener(){
                public void
actionPerformed(ActionEvent evt){
            if (songPlayingProcess != null) {
            PrintStream writer = new
PrintStream(songPlayingProcess.getOutputStream())

            rate -= 0.03;
            writer.println("rate " + rate);
            writer.flush();
            }
        }
```

```
        });

    /*
    model.addListDataListener(new
ListDataListener() {



        });
    */
    setVisible(true);

    new Thread(new TCPListener()).start();
    }

    public void playSong(SongInformation info)
{
    isPlaying = true;
    try {
        if (info.path.endsWith(".zip")) {
        songPlayingProcess =
Runtime.getRuntime().exec(new String[] {"bash",
"playZip", info.path});
        }  else if (info.path.endsWith(".kar"))
{
        songPlayingProcess =
Runtime.getRuntime().exec(new String[] {"bash",
"playKar", info.path});
```

```
        } else {
        isPlaying = false;
        return;
        }
        playingLabel.setText("Now playing: " +
info.toString());
        new Thread(new
WaitForSongToEnd()).start();
    } catch(IOException e) {
        System.err.println(e.toString());
        isPlaying = false;
    }
    }

    public void playNextSong() {
    if (model.isEmpty()) {
        return;
    }
    SongInformation info = (SongInformation)
model.remove(0);
    playSong(info);
    }

    class SongInformationRenderer extends
JLabel implements ListCellRenderer {

    public Component
```

```
getListCellRendererComponent(
                                JList list,
                                Object value,
                                int index,
                                boolean
isSelected,
                                boolean
cellHasFocus) {
        setText(value.toString());
        return this;
    }
    }

    class WaitForSongToEnd implements Runnable
{

    public void run() {
        try {
        songPlayingProcess.waitFor();
        } catch(Exception e) {
        System.err.println(e.toString());
        }
        Player.this.isPlaying = false;
        rate = 1.0;
        songPlayingProcess = null;
        System.out.println("Song finished!");
        Player.this.playingLabel.setText(" ");
```

```
        Player.this.playNextSong();
    }
    }

    class TCPListener implements Runnable {
    public int PORT = 13000;

    public void run() {
        try {
        System.out.println("Listening...");
        ServerSocket s = new
ServerSocket(PORT);
        while (true) {
            Socket incoming = s.accept();
            handleSocket(incoming);
            incoming.close();
        }
        } catch(IOException e) {
        System.err.println(e.toString());
        }
    }

    public void handleSocket(Socket incoming) {
        try {
        BufferedReader reader =
            new BufferedReader(new
InputStreamReader(
```

```
incoming.getInputStream()));

        String fname = reader.readLine();
        if (fname == null) {
            return;
        }
        System.out.println("Echo: " + fname);

        if (fname.endsWith(".zip") ||
            fname.endsWith(".mp3") ||
            fname.endsWith(".kar")) {
            // lose extension
            String root = fname.substring(0,
fname.length()-4);
            // lose /.../
            root =
root.substring(root.lastIndexOf('/')+1);
            //System.err.println(" root " +
root);
            String parts[] = root.split(" - ",
3);
            if (parts.length != 3)
            return;

            String index = parts[0];
            String artist = parts[1];
```

```
            String title = parts[2];

            SongInformation info = new
SongInformation(fname,
                                    index,
                                    title,
                                    artist);
            if (isPlaying) {
            model.addElement(info);
            } else {
            Player.this.playSong(info);
            }
        //songs.add(info);
        }


        } catch(IOException e) {
        System.err.println(e.toString());
        }
    }
    }
}
```

It contains buttons to control the speed of play and to stop play. More importantly, it hands control to different shell scripts based on the type of file downloaded.

Jack or PulseAudio?

One of the common requests is to raise or lower the pitch of a song. Jack can easily build a network of components, and if one of these is `jack-rack` then it can use a LADPSA plugin, such as the TAP pitch plugin.

So I use Jack.

Playing an MP3+G file

An Mp3+G pair is pulled off the server as a ZIP file. After unzipping, it can be played by `vlc` . `vlc` can use a Lua interface to which you can send commands such as "faster", "slower".

The script is `playZip`

```
#!/bin/bash

VLC_OPTS=" -I luaintf --lua-intf cli   --play-
and-exit --aout jack"

HTTP=http:/

[ ! -d /tmp/karaoke ] && mkdir /tmp/karaoke
rm -f /tmp/karaoke/*
wget "${HTTP}$*" -O /tmp/karaoke/tmp.zip
unzip -d /tmp/karaoke /tmp/karaoke/tmp.zip
rm /tmp/karaoke/tmp.zip
vlc $VLC_OPTS /tmp/karaoke/*.??3
```

Playing a KAR file

KAR files ripped of the Sonken player may have audio plus lyrics, or lyrics only with the audio in a linked WMA files. The MIDI player `timidity` can be used to play the KAR part, but for WMA files another player such as `mplayer` is needed.

An annoying part of `timidity` is that you cannot set the Jack output device to connect to. So you need to wait till it has registered with Jack, and then call `jack-connect` to link it to `jack-rack`.

The shell script is `playKar`:

```
#!/bin/bash
set -x

HTTP=http:/

echo Fetching $*

[ ! -d /tmp/karaoke ] && mkdir /tmp/karaoke
rm -f /tmp/karaoke/*
wget "${HTTP}$*" -O /tmp/karaoke/tmp.kar

WMAFILE="${*/.kar/.wma}"
echo wget "${HTTP}${WMAFILE}" -O
/tmp/karaoke/tmp.wma > /tmp/out0
wget "${HTTP}${WMAFILE}" -O
/tmp/karaoke/tmp.wma

if [ -s "/tmp/karaoke/tmp.wma" ]
then
    bash ./playMplayer /tmp/karaoke/tmp.wma &
    cd ../timidity
    ./TiMidity++-2.14.0/timidity/timidity -d. -
ix --trace --trace-text-meta
/tmp/karaoke/tmp.kar &
    cd ../Karaoke
    sleep 3
```

```
    jack_disconnect TiMidity:port_1
system:playback_1
    jack_disconnect TiMidity:port_2
system:playback_2

    wait %1
    killall mplayer
    # WMA file
#    java -jar WMAPlayer.jar
/tmp/karaoke/tmp.kar  > /dev/null 2> /dev/null
else
    # pykar /tmp/karaoke/tmp.kar > /dev/null 2>
/dev/null
    # give pykar time to give up audio card
    #sleep 4

    cd ../timidity
    ./TiMidity++-2.14.0/timidity/timidity -d. -
ix --trace --trace-text-meta
/tmp/karaoke/tmp.kar &
    sleep 3
    jack_disconnect TiMidity:port_1
system:playback_1
    jack_disconnect TiMidity:port_2
system:playback_2
    jack_connect TiMidity:port_1 jack_rack:in_1
    jack_connect TiMidity:port_2 jack_rack:in_2
```

```
fi
    wait %1
    cd ../Karaoke
fi
```

TiMidity plugin

The last bit of the puzzle is how TiMidity will show the lyrics. There is no supplied plugin that will display lyrics in the right format. The flags `--trace --trace-meta-text` will make the lyrics available in real-time to a plugin. I wrote one based on Xlib, with drawing done by Cairo and Pango. It is `x_code.c`

```c
#include <X11/Xlib.h>
#include <X11/Xutil.h>

#include <gtk/gtk.h>

#include <libavcodec/avcodec.h>
#include <libavformat/avformat.h>
#include <libswscale/swscale.h>

#include "mytimidity.h"

#define WIDTH 720
#define HEIGHT 480

#define NUM_LINES 4

Display *display;
Window window;
GC gc, color_gc;

struct _lyric_t {
    gchar *lyric;
    long tick;

};
typedef struct _lyric_t lyric_t;
```

```
struct _lyric_lines_t {
    char *language;
    char *title;
    char *performer;
    GArray *lines; // array of GString *
};
typedef struct _lyric_lines_t lyric_lines_t;

GArray *lyrics;
GString *lyrics_array[NUM_LINES];

lyric_lines_t lyric_lines;

typedef struct _coloured_line_t {
    gchar *line;
    gchar *front_of_line;
    gchar *marked_up_line;
    PangoAttrList *attrs;
} coloured_line_t;

int height_lyric_pixbufs[] = {100, 200, 300,
400}; // vertical offset of lyric in video
int coloured_text_offset;

// fluid_player_t* player;
```

```
// int current_panel = 1;  // panel showing
current lyric line
int current_line = 0;  // which line is the
current lyric
gchar *current_lyric;   // currently playing
lyric line
GString *front_of_lyric;  // part of lyric to
be coloured red
//GString *end_of_lyric;    // part of lyrci to
not be coloured


gchar *markup[] = {"<span font=\"28\"
foreground=\"RED\">",
           "</span><span font=\"28\"
foreground=\"white\">",
           "</span>"};
gchar *markup_newline[] = {"<span
foreground=\"black\">",
               "</span>"};
GString *marked_up_label;

PangoFontDescription *font_description;

cairo_surface_t *surface;
cairo_t *cr;
```

```
extern ControlMode  *ctl;

ControlMode video_ctl=
    {
    "x interface", 'x',
    "x",
    1,          /* verbosity */
    1,          /* trace playing */
    0,          /* opened */
    0,          /* flags */
    ctl_open,
    ctl_close,
    pass_playing_list,
    ctl_read,
    NULL,       /* write */
    cmsg,
    ctl_event
    };

static FILE *outfp;
int video_error_count;
static char *current_file;
struct midi_file_info *current_file_info;

static int pass_playing_list(int
number_of_files, char *list_of_files[]) {
    int n;
```

```
    for (n = 0; n < number_of_files; n++) {
    printf("Playing list %s\n",
list_of_files[n]);

    current_file = list_of_files[n];
    play_midi_file( list_of_files[n]);
    }
    XCloseDisplay(display);
    exit(0);
    return 0;
}

static void paint_background() {
    cr = cairo_create(surface);
    cairo_set_source_rgb(cr, 0.0, 0.8, 0.0);
    cairo_paint(cr);
    cairo_destroy(cr);
}

static void set_font() {
    font_description =
pango_font_description_new ();
    pango_font_description_set_family
(font_description, "serif");
    pango_font_description_set_weight
(font_description, PANGO_WEIGHT_BOLD);
```

```
    pango_font_description_set_absolute_size
(font_description, 32 * PANGO_SCALE);
}

static int draw_text(char *text, float red,
float green, float blue, int height, int
offset) {
  // See http://cairographics.org/FAQ/
  PangoLayout *layout;
  int width, ht;
  cairo_text_extents_t extents;

  layout = pango_cairo_create_layout (cr);
  pango_layout_set_font_description (layout,
font_description);
  pango_layout_set_text (layout, text, -1);

  if (offset == 0) {
      pango_layout_get_size(layout, &width,
&ht);
      offset = (WIDTH - (width/PANGO_SCALE)) /
2;
  }

  cairo_set_source_rgb (cr, red, green, blue);
  cairo_move_to (cr, offset, height);
  pango_cairo_show_layout (cr, layout);
```

```
  g_object_unref (layout);
  return offset;
}

static void init_X() {
    int screen;
    unsigned long foreground, background;
    XSizeHints hints;
    char **argv = NULL;
    XGCValues gcValues;
    Colormap colormap;
    XColor rgb_color, hw_color;
    Font font;
    //char *FNAME = "hanzigb24st";
    char *FNAME = "-misc-fixed-medium-r-normal-
-0-0-100-100-c-0-iso10646-1";

    display = XOpenDisplay(NULL);
    if (display == NULL) {
    fprintf(stderr, "Can't open dsplay\n");
    exit(1);
    }
    screen = DefaultScreen(display);
    foreground = BlackPixel(display, screen);
    background = WhitePixel(display, screen);
```

```
    window = XCreateSimpleWindow(display,
                 DefaultRootWindow(display),
                 0, 0, WIDTH, HEIGHT, 10,
                 foreground, background);
    hints.x = 0;
    hints.y = 0;
    hints.width = WIDTH;
    hints.height = HEIGHT;
    hints.flags = PPosition | PSize;

    XSetStandardProperties(display, window,
               "X", "X",
               None,
               argv, 0,
               &hints);

    XMapWindow(display, window);



    set_font();
    surface =
cairo_xlib_surface_create(display, window,
                   DefaultVisual(display, 0),
WIDTH, HEIGHT);
    cairo_xlib_surface_set_size(surface, WIDTH,
HEIGHT);
```

```
    paint_background();

    /*
    cr = cairo_create(surface);
    draw_text(g_array_index(lyric_lines.lines,
GString *, 0)->str,
          0.0, 0.0, 1.0,
height_lyric_pixbufs[0]);
    draw_text(g_array_index(lyric_lines.lines,
GString*, 1)->str,
          0.0, 0.0, 1.0,
height_lyric_pixbufs[0]);
    cairo_destroy(cr);
    */
    XFlush(display);
}


static int inited_video = 0;
/*ARGSUSED*/
static int ctl_open(int using_stdin, int
using_stdout)
{
    init_X();

    // dont know what this function does
    /*
```

```
      if (current_file != NULL) {
      current_file_info =
get_midi_file_info(current_file, 1);
      printf("Opening info for %s\n",
current_file);
      } else {
      printf("Current is NULL\n");
      }
    */
    ctl->opened = 1;
    return 0;
}

static void ctl_close(void)
{
    fflush(outfp);
    video_ctl.opened=0;
    exit(0);
}

/*ARGSUSED*/
static int ctl_read(int32 *valp)
{
    return RC_NONE;
}

static int cmsg(int type, int verbosity_level,
```

```
char *fmt, ...)
{
    /*
      va_list ap;

      if ((type==CMSG_TEXT || type==CMSG_INFO
|| type==CMSG_WARNING) &&
      video_ctl.verbosity<verbosity_level)
      return 0;
      va_start(ap, fmt);
      if(type == CMSG_WARNING || type ==
CMSG_ERROR || type == CMSG_FATAL)
      video_error_count++;
      if (!video_ctl.opened)
      {
      vfprintf(stderr, fmt, ap);
      fputs(NLS, stderr);
      }
      else
      {
      vfprintf(outfp, fmt, ap);
      fputs(NLS, outfp);
      fflush(outfp);
      }
      va_end(ap);
    */
    return 0;
```

```
}

static void ctl_total_time(long tt)
{
    /*
      int mins, secs;
      if (video_ctl.trace_playing)
      {
      secs=(int)(tt/play_mode->rate);
      mins=secs/60;
      secs-=mins*60;
      cmsg(CMSG_INFO, VERB_NORMAL,
      "Total playing time: %3d min %02d s",
mins, secs);
      }
    */
}

static void ctl_file_name(char *name)
{
    current_file = name;

    if (video_ctl.verbosity>=0 ||
video_ctl.trace_playing)
    cmsg(CMSG_INFO, VERB_NORMAL, "Playing %s",
name);
}
```

```
static void ctl_current_time(int secs)
{
    int mins;
    static int prev_secs = -1;

#ifdef __W32__
    if(wrdt->id == 'w')
    return;
#endif /* __W32__ */
    if (ctl->trace_playing && secs !=
prev_secs)
    {
        prev_secs = secs;
        mins=secs/60;
        secs-=mins*60;
        fprintf(stdout, "\r%3d:%02d", mins,
secs);
    }
}

void build_lyric_lines() {
    int n;
    lyric_t *plyric;
    GString *line = g_string_new("");
    GArray *lines =  g_array_sized_new(FALSE,
FALSE, sizeof(GString *), 64);
```

```
    lyric_lines.title = NULL;

    n = 1;
    char *evt_str;
    while ((evt_str = event2string(n++)) !=
NULL) {

        gchar *lyric = evt_str+1;
    printf("Building line %s\n", lyric);

    if ((strlen(lyric) >= 2) && (lyric[0] ==
'@') && (lyric[1] == 'L')) {
        lyric_lines.language =  lyric + 2;
        continue;
    }

    if ((strlen(lyric) >= 2) && (lyric[0] ==
'@') && (lyric[1] == 'T')) {
        if (lyric_lines.title == NULL) {
        lyric_lines.title = lyric + 2;
        } else {
        lyric_lines.performer = lyric + 2;
        }
        continue;
    }
```

```
    if (lyric[0] == '@') {
        // some other stuff like @KMIDI KARAOKE
FILE
        continue;
    }

    if ((lyric[0] == '/') || (lyric[0] ==
'\\')) {
        // start of a new line
        // add to lines
        g_array_append_val(lines, line);
        line = g_string_new(lyric + 1);
    }  else {
        line = g_string_append(line, lyric);
    }
    }
    lyric_lines.lines = lines;

    printf("Title is %s, performer is %s,
language is %s\n",
       lyric_lines.title,
lyric_lines.performer, lyric_lines.language);
    for (n = 0; n < lines->len; n++) {
    printf("Line is %s\n", g_array_index(lines,
GString *, n)->str);
    }
```

```
}

static void ctl_lyric(int lyricid)
{
    char *lyric;

    current_file_info =
get_midi_file_info(current_file, 1);

    lyric = event2string(lyricid);
    if(lyric != NULL)
    lyric++;
    printf("Got a lyric %s\n", lyric);


    if (*lyric == '\\') {
    // int next_panel = (current_panel+2) %
NUM_LINES; // really (current_panel+2)%2
    int next_line = current_line + NUM_LINES;
    gchar *next_lyric;

    if (current_line + NUM_LINES <
lyric_lines.lines->len) {
        current_line += 1;

        //lyrics_array[(next_line-1) %
NUM_LINES] =
```

```
        //    g_array_index(lyric_lines.lines,
GString *, next_line);

        // update label for next line after
this one
        next_lyric =
g_array_index(lyric_lines.lines, GString *,
next_line)->str;

    } else {
        current_line += 1;
        lyrics_array[(next_line-1) % NUM_LINES]
= NULL;
        next_lyric = "";
    }

    // set up new line as current line
    if (current_line < lyric_lines.lines->len)
{
        GString *gstr =
g_array_index(lyric_lines.lines, GString *,
current_line);
        current_lyric = gstr->str;
        front_of_lyric = g_string_new(lyric+1);
// lose    slosh
    }
    printf("New line. Setting front to %s end
```

```
to \"%s\"\n", lyric+1, current_lyric);


    // Now draw stuff
    paint_background();

    cr = cairo_create(surface);

    int n;
    for (n = 0; n < NUM_LINES; n++) {
        //lyrics_array[n] =
g_array_index(lyric_lines.lines, GString *,
n+1);
        if (lyrics_array[n] != NULL) {
        draw_text(lyrics_array[n]->str,
              0.0, 0.0, 0.5,
height_lyric_pixbufs[n], 0);
        }
    }
    // redraw current and next lines
    if (current_line < lyric_lines.lines->len)
{
        if (current_line >= 2) {
        // redraw last line still in red
        GString *gstr =
lyrics_array[(current_line-2) % NUM_LINES];
        if (gstr != NULL) {
```

```
            draw_text(gstr->str,
                  1.0, 0.0, 00,

height_lyric_pixbufs[(current_line-2) %
NUM_LINES],
                  0);
        }
        }
        // draw next line in brighter blue
        coloured_text_offset =
draw_text(lyrics_array[(current_line-1) %
NUM_LINES]->str,
              0.0, 0.0, 1.0,
height_lyric_pixbufs[(current_line-1) %
NUM_LINES], 0);
        printf("coloured text offset %d\n",
coloured_text_offset);
    }

    //try
    if (next_line < lyric_lines.lines->len) {
        lyrics_array[(next_line-1) % NUM_LINES]
=
        g_array_index(lyric_lines.lines,
GString *, next_line);
    }
```

```
        /*
    draw_text(current_lyric, 0.0, 0.0, 1.0,
          height_lyric_pixbufs[(current_line-1)
% NUM_LINES]);


    draw_text(next_lyric, 0.0, 0.0, 1.0,
          height_lyric_pixbufs[(next_line-1) %
NUM_LINES]);
        */
    cairo_destroy(cr);
    XFlush(display);


    } else {
    // change text colour as chars are played
    if ((front_of_lyric != NULL) && (lyric !=
NULL)) {
        g_string_append(front_of_lyric, lyric);
        char *s = front_of_lyric->str;

//coloured_lines[current_panel].front_of_line =
s;

        cairo_t *cr = cairo_create(surface);
```

```
        // See http://cairographics.org/FAQ/
        draw_text(s, 1.0, 0.0, 0.0,

height_lyric_pixbufs[(current_line-1) %
NUM_LINES],
              coloured_text_offset);

        cairo_destroy(cr);
        XFlush(display);

    }
    }
}


static void ctl_event(CtlEvent *e)
{
    //printf("Got ctl event %d\n", e->type);
    switch(e->type)
    {
    case CTLE_NOW_LOADING:
        ctl_file_name((char *)e->v1);
        break;
    case CTLE_LOADING_DONE:
        // MIDI file is loaded, about to play
        current_file_info =
get_midi_file_info(current_file, 1);
```

```
        if (current_file_info != NULL) {
        printf("file info not NULL\n");
        } else {
        printf("File info is NULL\n");
        }

        int n = 1;
        char *evt_str;
        while ((evt_str = event2string(n++)) !=
NULL) {
        printf("Event in tabel: %s\n",
evt_str);
        }

        build_lyric_lines();
        cr = cairo_create(surface);

        // draw line to be sung slightly
brighter
        // than the rest
        for (n = 0; n < NUM_LINES; n++) {
        lyrics_array[n] =
g_array_index(lyric_lines.lines, GString *,
n+1);
        draw_text(lyrics_array[n]->str,
              0.0, 0.0, 0.5,
height_lyric_pixbufs[n], 0);
```

```
        }
        draw_text(lyrics_array[0]->str,
              0.0, 0.0, 1.0,
height_lyric_pixbufs[0], 0);

        /*

draw_text(g_array_index(lyric_lines.lines,
GString *, 1)->str,
              0.0, 0.0, 1.0,
height_lyric_pixbufs[0]);

draw_text(g_array_index(lyric_lines.lines,
GString*, 2)->str,
              0.0, 0.0, 1.0,
height_lyric_pixbufs[1]);

draw_text(g_array_index(lyric_lines.lines,
GString*, 3)->str,
              0.0, 0.0, 1.0,
height_lyric_pixbufs[2]);
        */
        cairo_destroy(cr);
        XFlush(display);

        break;
    case CTLE_PLAY_START:
```

```
        ctl_total_time(e->v1);
        break;
        case CTLE_PLAY_END:
        printf("Exiting, play ended\n");
        exit(0);
        break;
    case CTLE_CURRENT_TIME:
        ctl_current_time((int)e->v1);
        break;
    case CTLE_LYRIC:
        ctl_lyric((int)e->v1);
        break;
        /*
          case CTLE_REFRESH:
          printf("Refresh\n");
          break;
        */
    default:
        0;
        //printf("Other event\n");
    }
}

/*
 * interface_<id>_loader();
 */
```

```
ControlMode *interface_x_loader(void)
{
    return &video_ctl;
}
```

Conclusion

All done! Well, as with anything, more can be done. But more will, I think, require use of the GPU in these small computers. Plus X is giving way to Wayland (or Mir on Ubuntu) and that should have hooks into the GPUs. So it's off to there I will probably go next.

Regards, and good luck with your own projects!

Copyright © Jan Newmarch, jan@newmarch.name

"Programming and Using Linux Sound - in depth"by Jan Newmarch is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License .

Based on a work at https://jan.newmarch.name/LinuxSound/ .

If you like this book, please contribute using PayPal

Or Flattr me: