The Linux MIDI−HOWTO

Table of Contents

Table of Contents

The Linux MIDI–HOWTO

By Phil Kerr, `phil@plus24.com`

v1.20, May 2002

This document describes the hardware, software and procedures needed to play and sequence using MIDI under Linux.

---

1. Introduction.

It covers:

- Configuring your MIDI interface
- Configuring and using softsynths
- Playing MIDI files
- Sequencing
- Controlling external MIDI equipment
- MIDI controlled software based sound synthesis
- Example MIDI code

---

2. Copyright of this document.

---

3. Where to get this document.

The official version of this document can be obtained from the Linux Documentation Project: http://www.tldp.org/.

The homepage for the most recent version of this HOWTO is: http://www.midi−howto.com

4. Acknowledgments.

This HOWTO is an expansion of the MIDI−SB mini−HOWTO written by Hideki Saito. Many thanks for his contribution to the Linux community.

This HOWTO now contains Frank Barknecht's 'HOWTO Use MIDI Sequencers With Softsynths'. Many thanks Frank.

Many of the code samples contained in this HOWTO originated from the Linux Audio Developers (LAD) mailinglist. Many thanks to them for allowing their inclusion.

5. Disclaimer.

Use the information in this document at your own risk.

I disavow any potential liability for the contents of this document.

Use of the concepts, examples, and/or other content of this document is entirely at your own risk.

All copyrights are owned by their owners, unless specifically noted otherwise.

Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark.

Naming of particular products or brands should not be seen as endorsements.

You are strongly recommended to take a backup of your system before major installation and backups at regular intervals.

6. Background to MIDI.

MIDI is a set of hardware and software protocols used by electronic musical instruments to communicate. It was first released in 1982 and has become the de−facto communication standard between electronic musical instruments. The protocol specifies the physical hardware parameters for the cable and interfaces, and provides a well defined set of communication protocols to exchange musical and timing data.

Prior to MIDI there were several analog, and later basic digital, connection protocols. The earliest, circa. 1974, would transmit note information as a voltage though a wire from one keyboard to another. Later, 1980 – 1981, Roland created a basic digital protocol, DCB. The MIDI Manufacturers Association (MMA), working with the equipment manufacturers, defined a standard protocol and physical connection which allowed equipment from all complying manufacturers to connect and communicate with each other.

>From 1985 MIDI interfaces started to appear on home computers and soon after sequencing programs appeared.

7. Configuring MIDI devices.

MIDI devices can be integrated into the soundcard or be a separate device. External MIDI interfaces may be attached to either the serial or USB port.

The first *and most important* thing you should do is check if your card is supported!

http://www.alsa−project.org/soundcards.php3

http://www.4front−tech.com/osshw.html

Configuring MIDI devices varies with Linux distributions. A well supported card may be configured when you install the OS.

The Linux kernel includes the OSS drivers and in the 2.5 kernel the ALSA drivers. Most distributions provide a configuration tool (mostly for soundcards), but if you are using the MIDI port of a sound card it should be configured. Under RedHat you would use sndconfig, under SuSE yast, and Mandrake, DrakConf.

If none of the above tools will configure your MIDI interface, or you are experiencing problems, the following steps should be taken:

Does lsmod show any MIDI related modules? Here's a typical output from an OSS based system.

```
[root@beatbox]# lsmod
Module                  Size  Used by
lockd                  32208   1  (autoclean)
sunrpc                 54640   1  (autoclean) [lockd]
autofs                  9456   2  (autoclean)
usb-ohci               12624   0  (unused)
usbcore                43632   1  [usb-ohci]
hisax                 470096   0  (autoclean) (unused)
isdn                  104208   0  (autoclean) [hisax]
slhc                    4544   0  (autoclean) [isdn]
eepro100               16144   1  (autoclean)

#---- Soundcard modules
    opl3                   11376   2
    mad16                   7968   1
    ad1848                 16848   1  [mad16]
    sb                     34752   1  [mad16]
    uart401                 6384   1  [mad16 sb]
    sound                  58368   0  [opl3 mad16 ad1848 sb uart401]

soundlow                 464   0  [sound]
soundcore               2800   6  [sb sound]
nls_cp437               3952   2  (autoclean)
vfat                    9408   1  (autoclean)
fat                    30432   1  (autoclean) [vfat]
ide-scsi                7664   0
```

Look for mpu401, olp3, uart401 and oss.

If you are using USB devices don't forget to check if the USB modules are there.

To check the config cat the sndstat file:

```
[root@beatbox]# cat /dev/sndstat
OSS/Free:3.8s2++-971130
Load type: Driver loaded as a module
Kernel: Linux mega 2.2.17-21mdk #1 Thu Oct 5 13:16:08 CEST 2000 i686
Config options: 0

Installed drivers:

Card config:

Audio devices:
0: MAD16 WSS (82C930) (DUPLEX)

Synth devices:
0: Yamaha OPL3

Midi devices:
0: Mad16/Mozart

Timers:
0: System clock

Mixers:
0: MAD16 WSS (82C930)
```

We see here that the MIDI device is a mad16 and this is listed in the lsmod output above.

If you see nothing related to MIDI check the contents of your /etc/modules.conf file.

```
[root@beatbox]# cat /etc/modules.conf
alias net-pf-4 ipx
pre-install pcmcia_core /etc/rc.d/init.d/pcmcia start
alias usb-interface usb-ohci
alias parport_lowlevel parport_pc
alias block-major-11 scsi_hostadapter
pre-install plip modprobe parport_pc ; echo 7 > /proc/parport/0/irq
alias scsi_hostadapter ide-scsi
alias eth0 eepro100
alias eth1 hisax

#---- Soundcard
    alias sound-slot-0 mad16
    options sound dmabuf=1
    alias midi opl3
    options opl3 io=0x388
    options sb support=1
    options mad16 io=0x530 irq=5 dma=0 dma16=1 mpu_io=0x300 mpu_irq=7 joystick=1
```

Here's the output of /proc/modules to check to see if the MIDI modules are loaded into the Kernel.

```
[root@mega /proc]# cat modules
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0070-007f : rtc
0080-008f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
01f0-01f7 : ide0
02f8-02ff : serial(auto)

#---- MIDI device
    0300-0303 : MPU-401 UART

0376-0376 : ide1
0388-038b : Yamaha OPL3
03c0-03df : vga+
03f6-03f6 : ide0
03f8-03ff : serial(auto)
0530-0533 : MAD16 WSS config
0534-0537 : MAD16 WSS
de00-de1f : Intel Speedo3 Ethernet
f000-f007 : ide0
f008-f00f : ide1
```

You should see something similar to the above. If not you'll need to install MIDI drivers.

If you are going to be using ALSA 0.5x divers, which you shouldn't do, I suggest a good read of Valentijn Sessink's Alsa−sound−mini−HOWTO which can be found at the link below:

http://www.linuxdoc.org/HOWTO/mini/Alsa−sound.html

You are strongly recommended to use ALSA greater than version 0.9. For ALSA drivers later than 0.9x you should have a good read of the ALSA−HOWTO by Madhu Maddy.

http://www.alsa−project.org/alsa−doc/alsa−howto/

7.1 ALSA 0.9 quick install

Below is a very quick install run−though for installing the ALSA 0.9 drivers and libs which is a required configuration for most MIDI apps.

```
[root@beatbox] # tar jxvf alsa-driver....tar.bz2
[root@beatbox] # cd alsa-driver.....
[root@beatbox] # ./configure
```

```
messages - no errors

[root@beatbox] # make

messages - no errors

[root@beatbox] # make install

messages - no errors

[root@beatbox] # ./snddevices
```

Now you will need to edit /etc/modules.conf, or the ALSA file in your modules directory on some distributions. There may be entries for other, non−MIDI, devices, so be careful when you are editing the file.

A typical system may have old ALSA or OSS configurations in the file, you will need to remove, or better still comment them out.

Below is a typical modules.conf file showing the ALSA config with OSS.

```
alias char-major-116 snd
alias char-major-14 soundcore

alias snd-card-0 (MIDI/Sound card)
alias sound-slot-0 snd-card-0

alias sound-service-0-0 snd-mixer-oss
alias sound-service-0-1 snd-seq-oss
alias sound-service-0-3 snd-pcm-oss
alias sound-service-0-12 snd-pcm-oss
```

Change the (MIDI/Sound card) entry to that of your card. This information can normally be found on the ALSA website.

With the ALSA drivers installed, now you will need to install the header library files needed by ALSA based programs. This is what is contained in the alsa−libs package.

Make sure you have a matching pair of alsa−drivers and alsa−libs!

```
[root@beatbox] # tar jxvf alsa-libs....tar.bz2
[root@beatbox] # cd alsa-libs.....
[root@beatbox] # ./configure

messages - no errors

[root@beatbox] # make

messages - no errors

[root@beatbox] # make install
```

Your system should now be configured :)

You can check this with a simple C program, if it compiles and can be executed then your system should be ok.

```
// Compile this test program like so: gcc alsatest.c -o alsatest -lasound

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

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

  if (snd_seq_open(&seq_handle, "hw", SND_SEQ_OPEN_DUPLEX, 0) < 0) {
    fprintf(stderr, "Error opening ALSA sequencer.\n");
    exit(1);
  }

printf("The ALSA libraries are installed.\n");
return 0;
}
```

7.2 Latency

MIDI is a real−time protocol and latency issues are a serious problem.

There are now several developers working on improving the latency times and improvements in the kernel are making Linux a fine platform for MIDI.

Although stock Linux distributions may run fine, pro set−ups should apply low−latency patches. More information can be found here:

http://www.gardena.net/benno/linux/audio/

http://www.linuxdj.com/audio/lad/resourceslatency.php3

Low Latency Mini Howto

http://www.boosthardware.com/LAU/guide/Low_latency−Mini−HOWTO.html

8. Software.

Interest in Linux based MIDI is growing and this list will probably not reflect the true amount of MIDI software available, but should provide a reasonable selection of applications.

If you are developing, or know of, any other MIDI apps not listed here please drop me a note.

8.1 Drivers

Alsa Driver

The Advanced Linux Sound Architecture is composed of several parts. The first is a fully modularized sound driver which supports module autoloading, devfs, isapnp autoconfiguration, and gives complete access to analog audio, digital audio, control, mixer, synthesizer, DSP, MIDI, and timer components of audio hardware. It also includes a fully−featured kernel−level sequencer, a full compatibility layer for OSS/Free applications, an object−oriented C library which covers and enhances the ALSA kernel driver functionality for applications (client/server, plugins, PCM sharing/multiplexing, PCM metering, etc.), an interactive configuration program for the driver, and some simple utilities for basic management.

http://www.alsa−project.org/

OSS

OSS provides sound card drivers for most popular sound cards under Linux and FreeBSD. These drivers support digital audio, MIDI, synthesizers, and mixers found on sound cards. These sound drivers comply with the Open Sound System API specification. OSS provides a user−friendly GUI which makes the installation of sound drivers and configuration of sound cards very simple. It supports over 200 brand name sound cards, and provides automatic sound card detection, Plug−n−Play support, support for PCI audio soundcards, and support for full duplex audio.

http://www.opensound.com/

Notemidi

Notemidi is a device driver for MIDI output via the RS−232 serial port on notebook/laptop computers. Notemidi can be used with the MIDIator MS−124W interface, Roland Sound Canvas sound modules, or Yamaha MU−x series sound modules.

http://www.michaelminn.com/linux/notemidi

8.2 MIDI file players

KMid

KMid is an X11 / KDE based midi player for Linux and FreeBSD. It displays the text of karaoke files and changes its colour as it is being played so that the tune can be easily followed. KMid uses /dev/sequencer as output device supporting external synths, AWE , FM and GUS cards.

http://perso.wanadoo.es/antlarr/kmid.html

Pmidi

Pmidi is a straightforward command line program to play Midi files through the ALSA sequencer.

http://www.parabola.demon.co.uk/alsa/pmidi.html

TiMidity++

TiMidity is a MIDI to WAVE converter that uses Gravis Ultrasound(*)−compatible patch files to generate digital audio data from General MIDI files. The audio data can be played through any sound device or stored on disk. On a fast machine, music can be played in real time.

http://www.goice.co.jp/member/mo/timidity/

8.3 Sequencers

Brahms

Brahms is a sequencer and music notation program with several editing methods so far including Score−, Pianoroll−, Drum−, and Mastertrack Editors. For C++ programmers, it is easy to implement further editors by deriving from a general editor−class. MIDI Import and Export is also implemented. In combination with aRts−0.3.4, one can play wave−files and make use of the midibus to send midi−events to the software synthesizer.

Formerly known as KooBase

http://brahms.sourceforge.net/

Anthem

Anthem is an advanced open source MIDI sequencer. Anthem allows you to record, edit and playback music using a sophisticated and acclaimed object oriented song technology.

http://anthem.sourceforge.net/

Jazz++

JAZZ++ is a full featured, audio capable midi sequencer for Linux and Windows.

http://www.jazzware.com/cgi−bin/Zope.cgi/jazzware/

Linux Ultimate Music Editor

UltiMusE−LX (the Ultimate Music Editor) is a composing program. No, it doesn't compose for you; it's a "word processor" for music. You draw sheet music on the screen using the mouse and/or computer keyboard. Up to 16 parts or voices fit on up to seven staves (staffs). Most standard musical notations are supported, as are MIDI instrument patch changes, events, and real−time clocks.

http://hometown.aol.com/knudsenmj/myhomepage/umuselx.htm

Melys

Melys is a Midi sequencer application for the Advanced Linux Sound Architecture (ALSA). Melys uses the sequencer support of ALSA, along with the GNOME libraries to produce a powerful and easy to use sequencer.

http://www.parabola.demon.co.uk/melys/

MidiMountain Sequencer

MidiMountain is a sequencer to edit standard MIDI files. Its easy−to−use interface should help beginners to edit and create MIDI songs (sequences), and it is designed to edit every definition known to standard MIDI files and the MIDI transfer protocol, from easy piano roll editing to changing binary system exclusive messages.

http://www.midimountain.com/

MusE

MusE is a Qt 2.1−based MIDI sequencer for Linux with editing and recording capabilities. While the sequencer is playing you can edit events in realtime with the pianoroll editor or the score editor. Recorded MIDI events can be grouped as parts and arranged in the arrange editor.

http://muse.seh.de/

Rosegarden

Rosegarden is an integrated MIDI sequencer and musical notation editor.

http://www.all−day−breakfast.com/rosegarden/

8.4 MIDI Trackers

tektracker

ttrk (tektracker) is a console MIDI sequencer with a tracker−style step editor. It is built for live playing, with convenient track mute buttons and loop control. ttrk supports both sending and syncing to MIDI clock pulses.

http://div8.net/ttrk/

ShakeTracker

ShakeTracker aims to be a fully−featured MIDI sequencer with a tracker interface. It currently works well and supports most tracker effects. Anyone who has used Impulse Tracker before will feel at home, and for the new users, a simple but extensive help system is provided. Most commands and shortcuts resemble their tracker counterparts.

http://reduz.com.ar/shaketracker/

8.5 Drum editors

DrumPatterns

DrumPatterns is a free, open source, web oriented drum patterns generator, whose purpose is to help teach drum patterns. It can teach the rudiments as well as advanced rhythms. It can output characters, score, or Midi, and includes hours of samples.

http://www.linux−france.org/prj/drumpatterns/index−en.html

8.6 Patch editors

JSynthLib

JSynthLib is an Open Source Universal Synthesizer Patch Editor / Librarian written in the Java Language. The project aims to eventually provide support for all existing Synthesizers by providing methods and documentation which allow users to develop drivers and editors for unsupported synths and contribute them to the project.

8.7 Software synths

Spiral Synth

Spiral Synth is a physically modeled polyphonic analogue synthesizer. It is capable of creating the kind of sounds made by hardware analogue synths, the noises used in electronic music. You can also use it to make stranger sounds too. MIDI is supported, and it uses the standard OSS/Free sound output (/dev/dsp).

http://www.pawfal.org/SpiralSynth/

UltraMaster Juno−6

UltraMaster Juno−6 is a faithful virtual reproduction of a Roland Juno−6 Polyphonic Synthesizer. It features realtime 64−bit internal ULTRANALOG wave synthesis, an early 80's style arpeggiator and chorus, and 100s of patches to save your own custom settings. All parameters can be controlled in realtime, via on−screen GUI or external MIDI controllers.

http://www.ultramaster.com/juno6/index.html

Pure−Data (PD)

"Pd" stands for "pure data". Pd is a real−time software system for live musical and multimedia performances. It is in active development by Miller Puckette , and perhaps others. The system is unfinished, but quite useable for sophisticated projects. It has been ported to Linux, IRIX, and many flavors of Windows.

http://www.pure−data.org/

Csound

Csound is a software synthesis program. But more than that, Csound doesn't suffer the same kinds of limitations that other software & hardware synthesizers have. There are no limits to the amount of oscillators or filters one can use. Csound is also completely modular, so that any function in Csound can be used in an array of ways.

http://www.csound.org/

Bristol synthesiser emulator

Bristol is a synthesizer emulation package. It includes a Moog Mini, Moog Voyager, Hammond B3, Prophet 5, Juno 6, DX 7, and others.

http://www.slabexchange.org/index.cgi?DOWNLOAD

8.8 Plugins

xmms−midi

Adds midi file support for x11amp (via timidity). A crude mixer interface is provided via the configuration dialog.

http://ban.joh.cam.ac.uk/~cr212/xmms−midi/

8.9 Notation

Mup

Mup takes a text file as input and produces PostScript output for printed music. It can handle both regular notation and tablature notation. (It can also produce MIDI output.)

http://www.arkkra.com/

Lilypond

LilyPond is a music typesetter. It produces beautiful sheet music using a high level description file as input. LilyPond is part of the GNU Project.

http://www.lilypond.org/

8.10 Development

sfront

Sfront compiles MPEG 4 Structured Audio (MP4−SA) bitstreams into efficient C programs that generate audio when executed. MP4−SA is a standard for normative algorithmic sound, that combines an audio signal processing language (SAOL) with score languages (SASL, and the legacy MIDI File Format). Under Linux, sfront supports real−time, low−latency audio input/output, local MIDI input from soundcards, and networked MIDI input using RTP and SIP. A SIP server hosted on the Berkeley campus manages sessions. The website includes an online book about MP4−SA.

http://www.cs.berkeley.edu/~lazzaro/sa/index.html

jMax

jMax allows one to interactively design dataflow circuits. The basic data types that can go through are integers, symbols, lists, etc. It is an event−driven system and has been used for MIDI processing. A second part of the system (DSP) allows a continuous signal to flow through a circuit, which is most useful for PCM sound (ie. microphone, sound files, etc). The system is extensible by using shared libraries, you may add data processor types, data types, GUI elements, device types, and more. Data processors may also be designed as circuits and reused.

http://www.ircam.fr/equipes/temps−reel/jmax/

TSE3

TSE3 is a powerful open source sequencer engine written in C++. It is a 'sequencer engine' because it provides the actual driving force elements of a sequencer but provides no form of user interface. Sequencer applications or multimedia presentation packages will incorporate the TSE3 libraries to provide a user with MIDI sequencing facilities.

http://TSE3.sourceforge.net/

KeyKit

KeyKit is a multi−tasking interpreted programming language (inspired by awk) designed exclusively for realtime and algorithmic MIDI manipulation. KeyKit's GUI provides several dozen tools for algorithmic music experimentation, including a multi−track sequencer and drum pattern editor. The GUI and all tools are completely written in the KeyKit language itself. This allows users to add new tools and operations to the existing tools, even while the system is running.

http://nosuch.com/keykit/

---

9. MIDI Development.

For those looking to develop MIDI applications, good examples are often needed to get you started.

The following examples were posted to the LAD mailing list in a thread about examples and documentation/tutorials.

9.1 Example 1

Below is a little sequencer routine from Dr. Matthias Nagorni. More examples can be obtained from Matthias's site which is listed in the Links section.

You compile it like so:

```
[phil@beatbox] $ gcc seqdemo.c −o seqdemo −lasound
```

```
#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, "hw", SND_SEQ_OPEN_DUPLEX, 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_WRITE,
            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);
    }
  }
}
```

9.2 Example 2

Below is a ALSA 0.9 MIDI redirector by Nick Dowell.

```
/* ALSA Sequencer MIDI redirector.
   Redirects the input to outputs determined by the MIDI channel
   (as requested by Nathaniel Virgo on Linux-Audio-Dev ;-)
   based on Dr. Matthias Nagorni's ALSA seq example

   Nick Dowell <nixx@nixx.org.uk>
   */

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

int
main()
{
  snd_seq_t *seq_handle;
  snd_seq_event_t *ev;
  int i;
  int portid;              /* input port */
  int oportid[16];         /* output ports */
  int npfd;
  struct pollfd *pfd;
  char txt[20];

  if (snd_seq_open(&seq_handle, "hw", SND_SEQ_OPEN_DUPLEX, 0) < 0) {
    fprintf(stderr, "Error opening ALSA sequencer.\n");
    exit(1);
  }

  snd_seq_set_client_name(seq_handle, "MIDI Redirect");

  /* open one input port */
  if ((portid = snd_seq_create_simple_port
       (seq_handle, "Input",
        SND_SEQ_PORT_CAP_WRITE | SND_SEQ_PORT_CAP_SUBS_WRITE,
        SND_SEQ_PORT_TYPE_APPLICATION)) < 0) {
    fprintf(stderr, "fatal error: could not open input port.\n");
    exit(1);
  }
  /* open 16 output ports for the MIDI channels */
  for (i=0; i<16; i++){
```

```
    sprintf( txt, "MIDI Channel %d", i );
    if ((oportid[i] = snd_seq_create_simple_port
         (seq_handle, txt,
          SND_SEQ_PORT_CAP_READ | SND_SEQ_PORT_CAP_SUBS_READ,
          SND_SEQ_PORT_TYPE_APPLICATION)) < 0) {
      fprintf(stderr, "fatal error: could not open output port.\n");
      exit(1);
    }
  }

  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)  /* main loop */
    if (poll(pfd, npfd, 1000000) > 0){
      do {
        snd_seq_event_input(seq_handle, &ev);
        snd_seq_ev_set_source( ev, oportid[ev->data.control.channel] );
        snd_seq_ev_set_subs( ev );
        snd_seq_ev_set_direct( ev );
        snd_seq_event_output_direct( seq_handle, ev );
        snd_seq_free_event(ev);
      } while (snd_seq_event_input_pending(seq_handle, 0) > 0);
    }
  return 0;
}
```

9.3 Example 3

Below is an example of writing data to the OSS /dev/midi interface by Craig Stuart Sapp.

More examples can be found at Craig's site listed in the Links section.

```
//
// Programmer:    Craig Stuart Sapp [craig@ccrma.stanford.edu]
// Creation Date: Mon Dec 21 18:00:42 PST 1998
// Last Modified: Mon Dec 21 18:00:42 PST 1998
// Filename:      ...linuxmidi/output/method1.c
// Syntax:        C
// $Smake:        gcc -O -o devmidiout devmidiout.c && strip devmidiout
//

#include <linux/soundcard.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>

int main(void) {
   char* device =  "/dev/midi" ;
   unsigned char data[3] = {0x90, 60, 127};

   // step 1: open the OSS device for writing
   int fd = open(device, O_WRONLY, 0);
   if (fd < 0) {
      printf("Error: cannot open %s\n", device);
      exit(1);
```

```
    }

    // step 2: write the MIDI information to the OSS device
    write(fd, data, sizeof(data));

    // step 3: (optional) close the OSS device
    close(fd);

    return 0;
}
```

10. HOWTO Use MIDI Sequencers With Softsynths.

Frank Barknecht <barknech@ph−cip.uni−koeln.de>

http://linux−sound.org/quick−toots/4−sequencers_and_softsynths/quick−toot−midisynth_howto.html

This HOWTO describes the needed setup to control a MIDI capable software synthesizer from a MIDI sequencer through a virtual MIDI connection under ALSA 0.9. This document can be freely translated and distributed. It's released under the GNU Free Documentation License.

10.1 Introduction

Software synthesizers like Csound, PD, jMax or Spiral Synth Modular offer nearly endless freedom to create known or unknown, common or unusual sound experiences. They can also replace pricy hardware synths or the inexpensive, often bad−sounding MIDI synths on some soundcards, if these are supported under Linux at all. On the other hand, composing inside those softsynths can be a tedious task: in Csound for example one has to write endless rows of numbers in a spreadsheet−like manner − not a comfortable way to make music.

MIDI sequencer applications are more suitable for this task. They provide an interface to insert notes and control data in convenient ways: as notes in a real score, as marks in a piano roll or as a list of MIDI events, if you prefer this view. Another kind of MIDI sequencers offer a tracker−like way of entering notes, like many people are used to from the good old days when that was the state of art in the Scene. Last but not least, some MIDI sequencers allow you to record your own playing on a keyboard or on another physical device, which is for many users the most natural way of writing music.

But MIDI sequencers like to output their notes to MIDI devices that normally route their events to the outside world, i.e., to hardware synths and samplers. With virtual MIDI devices one can keep the MIDI data inside the computer and let it control other software running on the same machine. This HOWTO describes all that is necessary to achieve this goal.

10.2 Device Setup

In our setup we use the ALSA library and driver modules, as this is (or should be) the standard way of doing serious audio and MIDI on Linux. The tutorial assumes that we are running the 0.9.0 branch of ALSA, but the virtual MIDI module was also present in ALSA 0.5.x so most of the following should apply for this as well. In the OSS/Free (the sound modules found in kernels previous to the 2.5.x track) and OSS/Linux sound architectures the v_midi module can be used, but this is beyond the scope of this document.

To use ALSA's virtual MIDI card the snd−card−virmidi module must be present. In the most recent versions of ALSA (and in the 2.5.x development kernel) that module lost its '−card' middle−fix and was renamed to snd−virmidi. Make sure that you did build this module, it might be missing if you configured ALSA to build only the modules for your actual card(s).

The virmidi module has to be loaded to make the virtual MIDI ports available. You can test−load it by hand with

```
$ modprobe snd-virmidi snd_index=1
```

where snd_index is set appropriately to the first free card index (=1 if you have only one card that already has index 0), but it is more convenient to adapt your modules configuration to have it sitting around already when you need it. For that we need to extend the ALSA section in /etc/modules.conf (or in another location, depending on your distribution) with the following:

```
# Configure support for OSS /dev/sequencer and
# /dev/music (aka /dev/sequencer2)
# (Takashi Iwai advises that it is unnecessary
# to alias these services beyond the first card, i.e., card 0)
alias sound-service-0-1 snd-seq-oss
alias sound-service-0-8 snd-seq-oss

# Configure card 1 (second card) as a virtual MIDI card
alias sound-slot-1 snd-card-1
alias snd-card-1 snd-virmidi
```

Now you have configured a virtual MIDI card as the second card with index 1, assuming you have one real soundcard (which would be very useful). If you have a second real card like I do, change the configuration above to read:

```
# Configure card 2 (third card) as a virtual MIDI card
alias sound-slot-2 snd-card-2
alias snd-card-2 snd-virmidi
```

If you have even more cards, well, you should know the deal by now...

It might be necessary to restart the ALSA sound system, after which the virtual MIDI card should be seen in /proc/asound/cards:

```
$ cat /proc/asound/cards
 0 [card0          ]: ICE1712 - M Audio Audiophile 24/96
                      M Audio Audiophile 24/96 at 0xb800, irq 5
 1 [card1          ]: EMU10K1 - Sound Blaster Live!
                      Sound Blaster Live! at 0xc800, irq 11
```

```
2 [card2          ]: VirMIDI − VirMIDI
                     Virtual MIDI Card 1
```

In this example of my own machine I have the VirMIDI card as third card, index 2. This setup results in the following raw MIDI devices, found in /proc/asound/devices [showing only the MIDI devices]:

```
$ cat /proc/asound/devices
  8: [0− 0]: raw MIDI
 41: [1− 1]: raw MIDI
 42: [1− 2]: raw MIDI
 75: [2− 3]: raw MIDI
 74: [2− 2]: raw MIDI
 73: [2− 1]: raw MIDI
 72: [2− 0]: raw MIDI
```

The devices starting with '2−' are my virtual MIDI devices. Yours would start with '1−' if you only had one physical card in your system.

You can get a nicer listing with ALSA's own aconnect utility, which we will need anyway. Called with option −o (or −lo) it will show the MIDI devices capable of MIDI output, while a call with −i shows those with MIDI input capabilities:

```
$ aconnect −o
[...]
client 80: 'Virtual Raw MIDI 2−0' [type=kernel]
    0 'VirMIDI 2−0     '
client 81: 'Virtual Raw MIDI 2−1' [type=kernel]
    0 'VirMIDI 2−1     '
client 82: 'Virtual Raw MIDI 2−2' [type=kernel]
    0 'VirMIDI 2−2     '
client 83: 'Virtual Raw MIDI 2−3' [type=kernel]
    0 'VirMIDI 2−3     '
$ aconnect −i
[...]
client 80: 'Virtual Raw MIDI 2−0' [type=kernel]
    0 'VirMIDI 2−0     '
client 81: 'Virtual Raw MIDI 2−1' [type=kernel]
    0 'VirMIDI 2−1     '
client 82: 'Virtual Raw MIDI 2−2' [type=kernel]
    0 'VirMIDI 2−2     '
client 83: 'Virtual Raw MIDI 2−3' [type=kernel]
    0 'VirMIDI 2−3     '
```

The devices shown correspond to ALSA's own OSS−compatible raw MIDI devices in the /proc/asound/dev directory tree. For example /proc/asound/dev/midiC2D0 is the first MIDI device of our virtual MIDI card at index 2, called Virtual Raw MIDI 2−0 by aconnect. In Debian those devices are available in /dev/snd/ as well, and they also are internally linked with the old OSS device locations: /dev/midiXX. To make sure that I can get the ALSA raw MIDI ports from /dev/midiXX I symlinked them with

```
$ ln −s /dev/snd/midiC2D0 /dev/midi20
$ ln −s /dev/snd/midiC2D1 /dev/midi21
[...]
```

but this should not be necessary, so don't do this at home, kids!

Now that we have created and configured a VirtualMIDI card we can use it in our applications just like any other MIDI devices. Just insert the needed device, be it an OSS−compatible /dev/midi20 or an ALSA MIDI port like 80:0, at the correct configuration point of your favourite sequencer and synthesizer application.

10.3 Routing MIDI Events

aconnect

Without further arrangements we will not be able to send the MIDI events from our sequencer to a softsynth. For that, we first need to connect two ports with (you guessed it) the aconnect utility. This tool connects two or more ports. Its −i and −o output above has already shown us the available ports, and with a simple syntax these can now be connected in a one−way fashion:

```
$ aconnect [sender port] [receiver port]
$ aconnect 80:0 81:0
```

This routes all MIDI data sent to port 80:0 over to port 80:1. In our setup this means that every event coming in at /dev/midi20 gets sent to /dev/midi21, where it can be read ('received') by another application.

If you had configured the VirMIDI card as your second card (with card index 1) you should have these ports:

```
$ aconnect −lo
client 72: 'Virtual Raw MIDI 1−0' [type=kernel]
    0 'VirMIDI 1−0     '
client 73: 'Virtual Raw MIDI 1−1' [type=kernel]
    0 'VirMIDI 1−1     '
client 74: 'Virtual Raw MIDI 1−2' [type=kernel]
    0 'VirMIDI 1−2     '
client 75: 'Virtual Raw MIDI 1−3' [type=kernel]
    0 'VirMIDI 1−3
```

Here you can connect for example port 72:0 (/dev/midi10) to port 73:0 (/dev/midi11) with:

```
$ aconnect 72:0 73:0
```

aconnect can show us what was created with its −lo and −li options:

```
$ aconnect -lo
client 72: 'Virtual Raw MIDI 1-0' [type=kernel]
    0 'VirMIDI 1-0     '
        Connecting To: 73:0
client 73: 'Virtual Raw MIDI 1-1' [type=kernel]
    0 'VirMIDI 1-1     '
        Connected From: 72:0
client 74: 'Virtual Raw MIDI 1-2' [type=kernel]
    0 'VirMIDI 1-2     '
client 75: 'Virtual Raw MIDI 1-3' [type=kernel]
    0 'VirMIDI 1-3
```

You see that 'Virtual Raw MIDI 1−0' now is connected to 'Virtual Raw MIDI 1−1'. Now, depending on your applications, you can read MIDI data that was sent to 'Virtual Raw MIDI 1−0' from 'Virtual Raw MIDI 1−1', or in OSS−device speak: What was sent to /dev/midi10 gets routed to /dev/midi11 and can be read from there.

You can also connect more than one port to a port. If you call aconnect twice like this:

```
$ aconnect 72:0 73:0
$ aconnect 72:0 74:0
```

you can receive the same data send to /dev/midi10 at /dev/midi11 and at /dev/midi12 as well. And of course you can really hammer your machine if you create more VirMIDI cards and wildly connect these. There's no stopping us now...

To disconnect all ports use

```
$ aconnect -x
```

or disconnect only one connection with:

```
$ aconnect -d 72:0 74:0
```

10.4 Graphical MIDI Patch Bays

Bob Ham's ALSA MIDI Patch Bay is a very useful graphic frontend to ALSA's MIDI connection setup. Usage is very simple and intuitive: On the left are the MIDI ports that are capable of sending MIDI events, while on the right you see the ports with receiving capability. If you click on a left−side port it gets selected for a new connection to the port on the right that you click next. Clicking on a right−side port will disconnect the port if it was connected. A clean and easy tool that has the potential to redundantize this HOWTO. ;)

aseqview

Another useful tool for routing MIDI events is aseqview by ALSA developer Takashi Iwai. You can download it at Iwai−sama's homepage http//members.tripod.de/iwai/alsa.html, but it is also included in many distributions. This graphic utility was designed to view and change MIDI events as they pass through your computer, but it also has the capability to route events to different MIDI ports like aconnect does. This comes in handy when you have to deal with applications that use the OSS sequencer device, which aconnect is sometimes unable to reach. If you start aseqview without any options you get a nice GUI and a new MIDI port. The default is port 128:0, and it looks like this:

```
client 128: 'MIDI Viewer' [type=user]
   0 'Viewer Port 0 '
```

With this port all the aconnect tricks that we have seen by now are possible. But if you just need to connect the aseqview port to another port, aseqview can do this by itself with the −d option :

```
$ aseqview −d 73:0 &
```

This connects port 128:0 (if that was available) to port 73:0 right from the start of aseqview.

There are some more graphical aconnect tools with very similar functionality. Maarten de Boer used most af the original aconnect source code to write a graphical frontend called "aconnectgui" with the FLTK toolkit. It is available at http://www.iua.upf.es/~mdeboer/. His software has the best looks, in my opinion.

Personally I use kaconnect, maybe because it has the shortest name, that is the fastest to type. kaconnect was developed by Suse's own Dr. Matthias Nagorni as a part of his series of tools and softsynths for ALSA, the kalsatools. Don't let the "k" in the name fool you, the software does not require KDE, it uses the pure QT GUI libraries. kaconnect and more is available at http://www.suse.de/~mana/kalsatools.html.

10.5 Applications

In this last section I will show some examples, how to use the virtual MIDI connections in various applications. I will assume a VirMIDI card as a third card in the system, using ALSA MIDI ports 80:0 to 83:0 that correspond to the raw OSS MIDI devices /dev/midi20 to /dev/midi23 and to the ALSA raw MIDI devices /dev/snd/midiC2D0 to /dev/snd/midiC2D3. Of these, the first two have been 'aconnected' with

```
$ aconnect 80:0 81:0
```

As shown, this means, that all MIDI data sent to /dev/midi20 (or port 80:0 or /dev/snd/midiC2D0) can now be read at /dev/midi21 (or port 80:1 or /dev/snd/midiC2D1)

10.6 Sequencers

MusE

MusE is a full−featured MIDI sequencer written by Werner Schweer, available at http://muse.seh.de. We will need to configure the the virtual MIDI port as an output port in the section 'Config−>MIDI Ports'. In MusE, the ports are named by their ALSA names 'VirMIDI X−X':

Now make sure that the right port is selected as the output port for the channel on which you want the software synthesizer to listen to and play the MIDI events:

For some reason I could not use 'VirMIDI 2−0' as output device in MusE 0.4.9. That is the expected device when I wanted to receive on 'VirMIDI 2−1' but I had to use it the other way around. I don't know why, I'm probably stupid, and you might have to experiment a bit. One could also use the midi02 or midi2 devices.

ttrk

Billy Biggs's ttrk is a simple, quick and tight MIDI sequencer with a tracker interface. It can output its MIDI data to any MIDI port configured in the file $HOME/.ttrkrc.

Put this line there to have ttrk write to /dev/snd/midiC2D0: and you're good to go...

Shaketracker

Juan Linietsky's Shaketracker revives the MIDI tracker interface like ttrk, but it has a more complete translation of the classic tracker effects to MIDI data. Unfortunatly it uses the OSS sequencer device (/dev/sequencer) for its MIDI output, not the raw MIDI devices, and I could not get it to work with aconnect. But there is a workaround that involves aseqview. If we start aseqview before Shaketracker, the tracker will recognize and use the aseqview port. One just has to select it in the 'User Devices' section of Shaketracker, where it shows up by its ALSA name 'Viewer Port 0':

It is convenient to give this User Device a good name instead of 'Null Output'.

If we start aseqview without options we would need to aconnect the aseqview port with the softsynth port, but as shown previously we could also start aseqview directly with a destination port. Don't forget to use the new User Device in every track that should go to the softsynth. I always run Shaketracker with a little shell script that starts aseqview, waits for the creation of ports, and then starts Shaketracker:

```
#!/bin/sh
aseqview −d 81:0 &
```

```
# sleep 2 seconds to let aseqview do its work:
sleep 2
shaketracker
```

10.7 Software Synthesizer

Pure Data

Miller Puckette is the genius behind the open−source software synthesis and multimedia development environment Pure Data (PD), which evolved out of MAX and in turn was the basis for the MAX−extension MSP. PD can use raw MIDI devices to read MIDI events that are specified with the option '−midiindev <devnumber>' but it has an irritating way of specifying which device to use. The formula is as follows: To use /dev/midi0, start PD with 'midiindev 1', to use /dev/midi1 start it with '−midiindev 2' and so on. Got it? You must specify the real device number plus 1 here. Another example: For /dev/midi21 start PD with '−midiindev 22'

PD has a help patch 'Test audio and MIDI', that is invaluable in locating the right MIDI device:

Csound

Csound is the grandmother of most of todays software synthesizers, and it has learned MIDI as well. Running 'csound −−help' shows where one has to configure the MIDI input device:

−M dnam or −−midieventdev=dnam .... Read MIDI realtime events from device

So in our example we need to start Csound as

```
$ csound −M /dev/midi22 −o devaudio midi.csd
```

Conclusion

By now you should know how to use a software synthesizer to orchestrate music composed in and played by a MIDI sequencer. Of course, tools like aconnect and aseqview don't need to be used with a MIDI software sequencer. You could also redirect events that come into your machine from an external sequencer or from an external MIDI keyboard directly to the software synth without the MIDI sequencer step. Just 'aconnect' the external MIDI device to your softsynth or to the on−board synth of your soundcard. Or go the other way around: PD, Csound or environments like KeyKit allow you to create MIDI events in algorithmic ways that are nearly impossible with classic Cubase−like MIDI sequencers. With aconnect you can route these events to any MIDI capable sound module you have.

10.8 Acknowledgments

The author would like to thank Takashi Iwai for his technical assistance in preparing this article and for his invaluable advice regarding the Japanese language. Thanks, Takashi !

11. Useful Links.

Below is a list of links to sites and resources covering MIDI and Linux.

http://www.linux−sound.org/ Sound & MIDI Software For Linux. A huge resource of MIDI and Audio related information.

http://www.linuxdj.com/audio/lad/ Linux Audio Developers mailing list. For developer related discussion.

http://www.linuxdj.com/audio/quality The Linux Audio Quality HOWTO. A good section on MIDI cards, essential reading for building a Linux studio.

http://www.alsa−project.org/ Alsa Project − Audio/MIDI Driver

http://www.4front−tech.com/ OSS − Audio/MIDI Driver

http://www.gardena.net/benno/linux/audio/ Low Latancy patches

http://ccrma−www.stanford.edu/~craig/articles/linuxmidi/ Introduction to MIDI programming in Linux. Craig Stuart Sapp covers writing basic MIDI utilities in C/C++ for the OSS driver.

http://www.suse.de/~mana Excellent collection of ALSA 0.9 MIDI and PCM example C programs by Dr. Matthias Nagorni.

---

12. Feedback.

As ever, a HOWTO is a work in progress. You are encouraged to give feedback on new applications and other interesting developments with MIDI under Linux.

MIDI cards are, by their wide variety, sometimes difficult to configure as they have various levels of driver support and configuration options. Your best source of reference for problems are ALSA's and OSS's websites, newsgroups, and if you get stuck please try searching the Linux Audio mailing lists.
