# Alsa 配置文件 – Asoundrc

来自<http://www.alsa-project.org/main/index.php/Asoundrc>

|  |  |
| --- | --- |
| |  | | --- | | 目录 | |
| |  | | --- | | * [1 什么时候需要asoundrc?](http://www.alsa-project.org/main/index.php/Asoundrc#Why_asoundrc.3F) * 2 ALSA配置概览 * [3 .asoundrc文件格式](http://www.alsa-project.org/main/index.php/Asoundrc#The_.asoundrc_file_format) * [4 asoundrc和周边的关系?](http://www.alsa-project.org/main/index.php/Asoundrc#Where_does_asoundrc_live.3F) * [5 默认的ALSA插件 - 那个叫做default的插件](http://www.alsa-project.org/main/index.php/Asoundrc#The_default_plugin) * [6 PCM设备的命名](http://www.alsa-project.org/main/index.php/Asoundrc#The_naming_of_PCM_devices)   + [6.1 控制设备](http://www.alsa-project.org/main/index.php/Asoundrc#The_Control_device) * [7 给你的设备一个别名](http://www.alsa-project.org/main/index.php/Asoundrc#Aliases) * [8 插件](http://www.alsa-project.org/main/index.php/Asoundrc#Plugins) * [9 软件混音](http://www.alsa-project.org/main/index.php/Asoundrc#Software_mixing)   + [9.1 输出混音插件dmix](http://www.alsa-project.org/main/index.php/Asoundrc#dmix)   + [9.2 输入混音插件dsnoop](http://www.alsa-project.org/main/index.php/Asoundrc#dsnoop)   + [9.3自定义插件例子JACK](http://www.alsa-project.org/main/index.php/Asoundrc#JACK_plugin)   + [9.4 虚拟多通道设备插件例子](http://www.alsa-project.org/main/index.php/Asoundrc#Virtual_multi_channel_devices)     - [9.4.1 Bindings explained](http://www.alsa-project.org/main/index.php/Asoundrc#Bindings_explained) * [10 参考例子](http://www.alsa-project.org/main/index.php/Asoundrc#Referenced_applications) * [11 扩展阅读](http://www.alsa-project.org/main/index.php/Asoundrc#Further_reading) | |

## 1.什么时候需要asoundrc?

用户侧的.asoundrc和asound.conf对ALSA(注: Advanced Linux Sound Architecture，高级Linux声音架构)来说都不是必须的.大多数应用都不依赖于他们.这两文件是用于在alsa-lib层面提供扩展功能,比如路由和码率转换(SRC, sample-rate conversion)(注:还可以有各种音效处理).当然,多数应用不需要用到这些文件的原因是,通常默认安装的alsa库已经能提供了足够多的功能,这些功能实际上包括了层层叠叠的配置文件(和asoundrc一样的语法)和相关的库,这个体系支持着不同类型的声卡.

## 2.ALSA配置概览

alsa-lib包(从Debian libasound2-data 1.0.27开始)使用了/usr/share/alsa/alsa.conf文件作为**主入口**.该文件负责包含系统上所有潜在的.asoundrc格式类型文件(.asoundrc-format-type files).她包含了指向ALSA 的"DATADIR"的引用(Debian和Ubuntu的这个目录在: /usr/share/alsa/, 注:也就是他自己所在的目录,建议到这个目录下转转).还引用了DATADIR目录下的cards/aliases.conf, 这里面定义了内核驱动中各声卡名和一个更详细名字的映射关系, 那个更详细的名字会被用于在DATADIR/cards/目录下查找对应的配置文件(注:其实那个更详细的名字也没有多详细,存在一对多的关系,几个相关内核设备被指向同一个配置,比如PMac Bugundy和PMac Tumbler被指向同一个配置PMac.conf).这些配置文件里面会根据特定卡的具体情况来做一些补偿,比如使用dmix来把单声道音频扩展到多个通道上,用downmix把立体声缩混到有单通道输出限制的设备上.最后,为了支持一些标准驱动无法支持的功能或者一些特需的定制配置,她还会载入一个全局的配置文件/etc/asound.conf和一个用户相关的配置文件~/.asoundrc.

最理想的情况是要给自己的声卡找到最合适的驱动和配置,避免自己却定义这些配置.

## .3.asoundrc文件格式

该文件提供了一些高级的控制接口,用于控制声卡和设备. .asoundrc文件用系统中可用的多个声卡的定义组成.也提供了访问alsa-lib中定义的PCM插件的接口.这使得你可以做很多拉风的事情,比如把多张卡合而为一或者访问一张多通道卡上的若干个接口.

(注:语法细节要再后面几节才讲)

## 4.asoundrc和周边的关系?

该文件通常在用户的home目录下($HOME/.asoundrc), 在/usr/share/alsa/alsa.conf中引用.

也可以放在/etc/asound.conf作为一个全局配置文件.

当一个alsa应用启动时,这两个文件都会被读取.

## 5.默认的ALSA插件 – 那个叫做default的插件

在你的home目录或者root目录先创建一个.asoundrc文件:

vi /home/xxx/.asoundrc

填入以下内容并保存.

pcm.!default {

type hw

card 0

}

ctl.!default {

type hw

card 0

}

关键词default在ALSA lib API中被定义为始终访问hw:0,0(默认声卡上的默认设备).指定!default会取代ALSA lib API中的设置.

现在可以试一下:

aplay -D default test.wav

(注:aplay是一个播放器,-D default是指定在default这个设备上播放)

## 6.PCM设备的命名

典型的asoundrc以'PCM hw type'开头. 这让ALSA应用可以通过指定名称来启动一个虚拟的声卡设备(插件或者从设备). 没有这个命名,就只能使用hw:0,0这样的名字或者default来访问这些设备了.使用的例子如下:

aplay -D hw:0,0 test.wav

(注:这是用aplay在hw:0,0上面播放音乐test.wav)

或者使用ecasound

ecasound -i test.wav -o alsa,hw:0,0

hw:后面的数字分别表示声卡号(soundcard number)和设备号(device number).这可能有点混淆,有的声卡可能被叫做设备更合适,比如一个USB声卡设备. 不管怎么说,在某种意义上他们还是卡(cards),一个特定的驱动程序控制一个特定的硬件。它们也对应于文件/proc/asound/cards

中的索引。

多数情况下第一个设备从0开始,而不是1.这也是pcm设备(物理IO通道)在ALSA中的表现方式.从pcm0c (录音设备)和pcm0p (播放设备)开始.

对于能混合几个音频流的硬件,我们使用了子设备的概念. 子设备可以用特定的地址来打开,通常第一个空闲的子设备被打开.我们会临时使用子设备来访问有多个流(I/O连接器)的硬件设备比如MIDI. 子设备有一些数量限制(每张卡上最多只能有8个PCM设备,8个MIDI设备,等等).

举个例子,要访问第一个声卡上的第一个设备,应该使用hw:0,0.

如果要访问第二个声卡上的第一个设备,应该使用hw:1,0.

访问第三个声卡上的第二个设备,应该使用hw:2,1.

### 6.1控制设备

卡上的控制设备被用于操作不同的控制接口.对于大多数卡,这就包括mixer(混音器),但是有的卡比如rme9652就没有mixer.不管怎么说,他们都会有不少的其他控制接口,以便类似JACK这样的程序访问他们.例子还包括数字I/O同步指示器(digital I/O sync indicators)和采样时钟源开关(sample clock source switch)之类的.

## 7.给你的设备一个别名

可以使用'PCM hw type'来给设备定义一个别名(alias), 相关语法如下:

pcm.NAME {

type hw # Kernel PCM

card INT/STR # 卡名或数字,这里的INT表示整数类型,STR表示字符串

[device] INT # 设备编号(默认为0)

[subdevice] INT # 子设备编号, -1 表示第一个可用的子设备 (默认的-1)

mmap\_emulation BOOL # enable mmap emulation for ro/wo devices

}

举个例子,以下代码给第一个声卡设定了一个别名:

pcm.primary {

type hw

card 0

device 0

}

现在你可以用 'primary'这个别名来使用该设备了.

aplay -D primary test.wav

## 8.插件

**Q:** 什么是插件?

**A:** 在ALSA中,PCM插件用于扩展PCM设备的功能和特性.插件可以执行设备命名,码率转换(sample rate conversions), 通道间的采样数据复制,记录到文件,联合多个声卡/设备实现多通道输入输出(无需采样对齐),使用多通道声卡/设备, 当然还有更多可能等待您去发现. 使用这些功能,需要创建一张虚拟从设备(slave device).

要查看完整的插件列表和选项,请查看[alsa-lib 文档库](http://www.alsa-project.org/alsa-doc/alsa-lib/pcm_plugins.html). 以下是一个简要介绍.

一个非常简单的从设备可以定义如下:

pcm\_slave.sltest {

pcm "hw:1,0"

}

这个虚拟的从设备没有任何参数,他只是提供了一个别名.稍微要注意的就是'pcm types'的参数必须定义在从设备定义部分(slave-definition-block). 我们再来配置一个码率转换设备.

pcm\_slave.sl2 {

pcm "hw:1,0"

rate 48000

}

pcm.rate\_convert {

type rate

slave sl2

}

这样你只需要使用命令aplay -D rate\_convert test.wav 就可以使用此设备来自动把test.wav转换成44.1 kHz采样频率的数据并播放. 这个转换器用处也不是很大,因为多数播放器和alsa会把采样频率转换成你的声卡所支持的频率,不过如果你想完成一个转成较低的特定采样频率转换时就会用的上.

我们也可以用嵌套写法重写如下:

pcm.rate\_convert {

type rate

slave {

pcm

rate 48000

}

}

一种更加复杂的写法叫做pcm type plug. 语法如下:

type plug # 格式修改,注:这里可以理解为 type = plug

slave STR # 从设备名称

# 或者

slave { # 从设备定义

pcm STR # 从设备名

# 或者

pcm { } # 从设备定义

[format STR] # 格式 (默认为最接近的) 或 "unchanged"

[channels INT] # 通道数 (默认为最接近的) 或 "unchanged"

[rate INT] # 采样频率 (默认为最接近的) 或 "unchanged"

}

route\_policy STR # 路由策略,用于下面ttbal的自动生成

# 取值可以是'默认', '平均', '复制(copy)', '重复(duplicate)'

# 平均: 结果是各输入通道的平均值

# 复制: 只有第一个通道的输入拷贝的输出

# 重复: 重复第一组输入(注:这里没理解透,应该是在输入少于输出的情况时)

# 默认: 拷贝策略,除了单声道录音的情况

ttable { # 转移表,二维转移矩阵(输入通道数\*输出通道数) (bidimensional compound of

# cchannels \* schannels numbers)

CCHANNEL {

SCHANNEL REAL # 路由取值 (0.0 ... 1.0)

}

}

比如以下配置可以把采用格式转换成16kHZ的单通道S16\_LE:

pcm\_slave.sl3 {

pcm "hw:1,0"

format S16\_LE

channels 1

rate 16000

}

pcm.complex\_convert {

type plug

slave sl3

}

调用方法为:

aplay -vD complex\_convert test.wav

aplay的-v选项可以用来查看原文件的采样格式,命令为:

aplay -v test.wav

## 9.软件混音

Software mixing is the ability to play multiple sound files or applications at the same time through the same device. There are many ways to have software mixing in the Linux environment. Usually it requires a server application such as ARTSD, ESD, JACK... The list is large and the apps can often be confusing to use.

### dmix

These days we have a native plugin for ALSA called the dmix (direct mixing) plugin. It allows software mixing in an easy to use syntax and without the hassle of installing/understanding a new application first.

A very interesting and potentially extremely useful aspect of this plugin is using it combined with the default plugin name. In theory this means all applications that have native ALSA support will share the sound device. In practice not many applications are able to take advantage of this functionality yet. However if you have time to test and report your findings to the application developers it is worth a try:

pcm.!default {

type plug

slave.pcm "dmixer"

}

pcm.dmixer {

type dmix

ipc\_key 1024

slave {

pcm "hw:1,0"

period\_time 0

period\_size 1024

buffer\_size 4096

rate 44100

}

bindings {

0 0

1 1

}

}

ctl.dmixer {

type hw

card 0

}

(To use it with your own name just change the word !default above)

Now try:

aplay -f cd -D default test.wav

on multiple consoles.

Some notes:

The dmix PCM name is already defined in the global configuration file /usr/share/alsa/alsa.conf.

- The default sample rate for this device is 48000Hz. If you would like to change it use:

aplay -D"plug:'dmix:RATE=44100'" test.wav

- An example command for dmix plugin to use 44100Hz sample-rate and hw:1,0 output device:

aplay -Dplug:\'dmix:SLAVE=\"hw:1,0\",RATE=44100\' test.wav

- The dmix PCM name is already defined in the global configuration file /usr/share/alsa/alsa.conf.

Defaults are:

SLAVE="hw:0,0",RATE=48000

NB the dmix plugin is not based on client/server architecture, but it writes directly to the soundcard's DMA buffer. There is no limit to the amount of instances that can be run at one time.

### dsnoop

While the dmix plugin is for mixing multiple output(playback) streams together, if you want to use multiple input(capture) clients you need to use the dsnoop plugin:

pcm.mixin {

type dsnoop

ipc\_key 5978293 # must be unique for all dmix plugins!!!!

ipc\_key\_add\_uid yes

slave {

pcm "hw:0,0"

channels 2

period\_size 1024

buffer\_size 4096

rate 44100

periods 0

period\_time 0

}

bindings {

0 0

0 1

}

}

### JACK plugin

This plugin allows the user to connect applications that support alsa natively to the JACK daemon.

To compile and install jack, you need to move to src/pcm/ext directory, and run "make jack" and "make install-jack". this is intentional.

To use the JACK plugin you will need to add the following to your .asoundrc.

pcm.jackplug {

type plug

slave { pcm "jack" }

}

pcm.jack {

type jack

playback\_ports {

0 alsa\_pcm:playback\_1

1 alsa\_pcm:playback\_2

}

capture\_ports {

0 alsa\_pcm:capture\_1

1 alsa\_pcm:capture\_2

}

}

Now, you can use:

aplay -Djackplug somefile

arecord -Djackplug somefile

### Virtual multi channel devices

If you would like to link two or more ALSA devices together so that you have a virtual multi channel device it is possible. However this will not create the mythical "multi-channel soundcard out of el-cheapo consumer cards". The real devices will drift out of sync over time. It is sometimes helpful to make applications that can for example see one 4 channel card to allow for flexible routing if they can't easily be made to talk to multiple cards (making use of JACK being one example).

Copy and paste the following into your asoundrc.

# create a virtual four-channel device with two sound devices:

# This is in fact two interleaved stereo streams in

# different memory locations, so JACK will complain that it

# cannot get mmap-based access. see below.

pcm.multi {

type multi;

slaves.a.pcm "hw:0,0";

slaves.a.channels 2;

slaves.b.pcm "hw:1,0";

slaves.b.channels 2;

bindings.0.slave a;

bindings.0.channel 0;

bindings.1.slave a;

bindings.1.channel 1;

bindings.2.slave b;

bindings.2.channel 0;

bindings.3.slave b;

bindings.3.channel 1;

}

# JACK will be unhappy if there is no mixer to talk to, so we set

# this to card 0. This could be any device but 0 is easy.

ctl.multi {

type hw;

card 0;

}

# This creates a 4 channel interleaved pcm stream based on

# the multi device. JACK will work with this one.

pcm.ttable {

type route;

slave.pcm "multi";

slave.channels 4;

ttable.0.0 1;

ttable.1.1 1;

ttable.2.2 1;

ttable.3.3 1;

}

# see above.

ctl.ttable {

type hw;

card 0;

}

This will give you xruns, but it's suitable for testing. To test the above setting feed an audio signal to the real sound devices play it back and listen to it with an external mixer:

arecord -f S16\_LE -r 44100 -c 4 -D multi | aplay -f S16\_LE -r 44100 -c 4 -D multi

To start JACK with the new device, use

jackd [-a] -R [-v] -d alsa -d ttable [-p 1024]

#### Bindings explained

The above example for a virtual multi channel device uses bindings to make the connections work. The following is a more advanced asoundrc for 2 RME Hammerfalls which is a professional multichannel sound device. Below is a full explanation of how bindings work.

# This is for two RME Hammerfall cards. They have been conceptually split into rows

# and will be mapped to ALSA channels 0-27: with channels 0-7 and 16-23 on rme9652\_0 and channels 8-15 on

# rme9652\_1, and finally channels 24-25 of both cards, used as two stereo channels, mapped to ALSA channels 24-27 (while the others are mono).

# eg. ALSA channels on card1 = rme9652\_0

# | 0 1 2 3 4 5 6 7 |

# | 8 9 10 11 12 13 14 15 |

# | 24 25|

#

# ALSA channels on card2 = rme9652\_1

# | 16 17 18 19 20 21 22 23 |

# | 26 27|

pcm\_slave.rme9652\_s {

pcm rme9652\_0

}

pcm.rme9652\_1 {

type hw

card 1

}

ctl.rme9652\_1 {

type hw

card 1

}

pcm.rme9652\_0 {

type hw

card 0

}

ctl.rme9652\_0 {

type hw

card 0

}

ctl.rme9652\_48 {

type hw

card 0

}

pcm.rme9652\_48 {

type multi;

slaves.a.pcm rme9652\_0;

slaves.a.channels 26;

slaves.b.pcm rme9652\_1;

slaves.b.channels 26;

# Use rme9652\_0 ch 0-7 mapped to ALSA ch 0-7

bindings.0.slave a;

bindings.0.channel 0;

bindings.1.slave a;

bindings.1.channel 1;

bindings.2.slave a;

bindings.2.channel 2;

bindings.3.slave a;

bindings.3.channel 3;

bindings.4.slave a;

bindings.4.channel 4;

bindings.5.slave a;

bindings.5.channel 5;

bindings.6.slave a;

bindings.6.channel 6;

bindings.7.slave a;

bindings.7.channel 7;

# Use rme9652\_0 ch 16-23 mapped to ALSA ch 8-15

bindings.8.slave a;

bindings.8.channel 16;

bindings.9.slave a;

bindings.9.channel 17;

bindings.10.slave a;

bindings.10.channel 18;

bindings.11.slave a;

bindings.11.channel 19;

bindings.12.slave a;

bindings.12.channel 20;

bindings.13.slave a;

bindings.13.channel 21;

bindings.14.slave a;

bindings.14.channel 22;

bindings.15.slave a;

bindings.15.channel 23;

# Use rme9652\_1 ch 8-15 mapped to ALSA ch 16-23

bindings.16.slave b;

bindings.16.channel 8;

bindings.17.slave b;

bindings.17.channel 9;

bindings.18.slave b;

bindings.18.channel 10;

bindings.19.slave b;

bindings.19.channel 11;

bindings.20.slave b;

bindings.20.channel 12;

bindings.21.slave b;

bindings.21.channel 13;

bindings.22.slave b;

bindings.22.channel 14;

bindings.23.slave b;

bindings.23.channel 15;

# stereo channels: slave a+b ch 24-25 mapped to ALSA ch 24-27

bindings.24.slave a;

bindings.24.channel 24;

bindings.25.slave a;

bindings.25.channel 25;

bindings.26.slave b;

bindings.26.channel 24;

bindings.27.slave b;

bindings.27.channel 25;

}

What is happening?

There are two sound cards which are linked with a wordclock pipe. That allows them to keep sample sync with each other which is very important for multichannel work. If the sample rates are not in sync then your sounds become out of time with each other.

In this example each sound card has a number of physical channels being used: 18 (a) + 10 (b). They are represented in /proc/asound/cardx as pcmXc (capture) and pcmXp (playback). Where X equals the number of the physical input/output (i/o) channels starting at 0.

If you look at the lines:

type multi;

slaves.a.pcm rme9652\_0;

slaves.a.channels 26;

You can see that the card has been nicknamed "a" and given a range of 26 channels. You can assign the card any number of channels you want but you can only use as many channels as the card has physically available. The bindings start at the first available pcm device for the card ie. pcm0c pcm0p - and move upwards sequentially from there.

The first card for this file has 18 physical pcm devices. Nine of them are capture devices ie. pcm0c pcm1c pcm2c ... pcm8c, each with corresponding playback channels ie. pcm0p pcm1p pcm2p ... pcm8p. The second card has 10 physical pcm devices ie. pcm0c pcm1c pcm2c ... pcm9c.

If you look at the lines:

bindings.0.slave a;

bindings.0.channel 0;

bindings.1.slave a;

bindings.1.channel 1;

The first binding points to the first available pcm device on the card represented as "a". The second binding points to the second available pcm device on "a" and so on up to the last one available. We then assign a channel number to the binding so that the channels on the new virtual "soundcard" we have created are easy for us to access.

Another way of saying it is:

address of.the first channel on my new soundcard.using my real soundcard called "a";

make this address of.the first channel on my new soundcard.be the first pcm device on my new soundcard;

address of.the second channel on my new soundcard.using my real soundcard called "a";

make this address of.the second channel on my real soundcard.be the second pcm device on my new soundcard;

## Referenced applications

* **aRTsd** - the aRTs sound server was the basis of desktop sound for KDE 3.
* **ESD** - the Enlightened Sound Daemon mixes several audio streams for playback by a single audio device.
* **Ecasound** - a commandline multitrack recorder and editor with various GUI apps.
* **JACK** - Jack Audio Connection Kit. If you don't know this app you don't know JACK.

## Further reading

* [A close look at ALSA](http://www.volkerschatz.com/noise/alsa.html) explaining ALSA and the asoundrc in particular from A to Z. By Volker Schatz
* [.asoundrc](http://alsa.opensrc.org/.asoundrc) at the [Unofficial wiki](http://alsa.opensrc.org/)

注:建议参考<http://www.volkerschatz.com/noise/alsa.html>

<http://www.alsa-project.org/alsa-doc/alsa-lib/pcm_plugins.html>