NAME

mixer_open, **mixer_close**, **mixer_get_dev**, **mixer_get_dev_byname**, **mixer_add_ctl**, **mixer_add_ctl_s**, **mixer_remove_ctl**, **mixer_get_ctl**, **mixer_get_ctl_byname**, **mixer_set_vol**, **mixer_set_mute**, **mixer_mod_recsrc**, **mixer_get_dunit**, **mixer_set_dunit**, **mixer_get_mode**, **mixer_get_nmixers**, **mixer_get_path**, **MIX_ISDEV**, **MIX_ISMUTE**, **MIX_ISREC**, **MIX_ISRECSRC**, **MIX_VOLNORM**, **MIX_VOLDENORM** - interface to OSS mixers

LIBRARY

Mixer library (libmixer, -lmixer)

SYNOPSIS

#include <mixer.h>

*struct mixer **
mixer_open(*const char *name*);

int
mixer_close(*struct mixer *m*);

*struct mix_dev **
mixer_get_dev(*struct mixer *m, int devno*);

*struct mix_dev **
mixer_get_dev_byname(*struct mixer *m, name*);

int
mixer_add_ctl(*struct mix_dev *parent, int id, const char *name, int (*mod)(struct mix_dev *d, void *p), int (*print)(struct mix_dev *d, void *p)*);

int
mixer_add_ctl_s(*mix_ctl_t *ctl*);

int
mixer_remove_ctl(*mix_ctl_t *ctl*);

*mix_ctl_t **
mixer_get_ctl(*struct mix_dev *d, int id*);

*mix_ctl_t **
mixer_get_ctl_byname(*struct mix_dev *d, const char *name*);

int
mixer_set_vol(*struct mixer *m, mix_volume_t vol*);

int
mixer_set_mute(*struct mixer *m, int opt*);

int
mixer_mod_recsrc(*struct mixer *m, int opt*);

int
mixer_get_dunit(*void*);

int
mixer_set_dunit(*struct mixer *m, int unit*);

int
mixer_get_mode(*int unit*);

int
mixer_get_nmixers(*void*);

int
mixer_get_path(*char * buf, size_t size, int unit*);

int
MIX_ISDEV(*struct mixer *m, int devno*);

int
MIX_ISMUTE(*struct mixer *m, int devno*);

int
MIX_ISREC(*struct mixer *m, int devno*);

int
MIX_ISRECSRC(*struct mixer *m, int devno*);

float
MIX_VOLNORM(*int v*);

int

MIX_VOLDENORM(*float v*);

DESCRIPTION

The **mixer** library allows userspace programs to access and manipulate OSS sound mixers in a simple way.

Mixer

A mixer is described by the following structure:

```
struct mixer {
        TAILQ_HEAD(mix_devhead, mix_dev) devs;    /* device list */
        struct mix_dev *dev;                      /* selected device */
        oss_mixerinfo mi;                 /* mixer info */
        oss_card_info ci;                 /* audio card info */
        char name[NAME_MAX];                      /* mixer name (e.g /dev/mixer0) */
        int fd;                                   /* file descriptor */
        int unit;                         /* audio card unit */
        int ndev;                         /* number of devices */
        int devmask;                              /* supported devices */
#define MIX_MUTE                0x01
#define MIX_UNMUTE              0x02
#define MIX_TOGGLEMUTE                  0x04
        int mutemask;                             /* muted devices */
        int recmask;                              /* recording devices */
#define MIX_ADDRECSRC           0x01
#define MIX_REMOVERECSRC        0x02
#define MIX_SETRECSRC           0x04
#define MIX_TOGGLERECSRC        0x08
        int recsrc;                       /* recording sources */
#define MIX_MODE_MIXER                  0x01
#define MIX_MODE_PLAY           0x02
#define MIX_MODE_REC            0x04
        int mode;                         /* dev.pcm.X.mode sysctl */
        int f_default;                            /* default mixer flag */
};
```

The fields are follows:

devs A tail queue structure containing all supported mixer devices.

dev A pointer to the currently selected device. The device is one of the elements in *devs*.

mi OSS information about the mixer. Look at the definition of the *oss_mixerinfo* structure in <*sys/soundcard.h*> to see its fields.

ci OSS audio card information. This structure is also defined in <*sys/soundcard.h*>.

name Path to the mixer (e.g /dev/mixer0).

fd File descriptor returned when the mixer is opened in **mixer_open**().

unit Audio card unit. Since each mixer device maps to a pcmX device, *unit* is always equal to the number of that pcmX device. For example, if the audio device's number is 0 (i.e pcm0), then *unit* is 0 as well. This number is useful when checking if the mixer's audio card is the default one.

ndev Number of devices in *devs*.

devmask Bit mask containing all supported devices for the mixer. For example, if device 10 is supported, then the 10th bit in the mask will be set. By default, **mixer_open**() stores only the supported devices in devs, so it is very unlikely this mask will be needed.

mutemask
Bit mask containing all muted devices. The logic is the same as with *devmask*.

recmask Bit mask containing all recording devices. Again, same logic as with the other masks.

recsrc Bit mask containing all recording sources. Yes, same logic again.

mode Bit mask containing the supported modes for this audio device. It holds the value of the *dev.pcm.X.mode* sysctl.

f_default Flag which tells whether the mixer's audio card is the default one.

Mixer device

Each mixer device stored in a mixer is described as follows:

```
struct mix_dev {
        struct mixer *parent_mixer;             /* parent mixer */
        char name[NAME_MAX];                    /* device name (e.g "vol") */
```

```
        int devno;                                  /* device number */
        struct mix_volume {
#define MIX_VOLMIN              0.0f
#define MIX_VOLMAX              1.0f
#define MIX_VOLNORM(v)          ((v) / 100.0f)
#define MIX_VOLDENORM(v)        ((int)((v) * 100.0f + 0.5f))
                float left;             /* left volume */
                float right;                    /* right volume */
        } vol;
        int nctl;                               /* number of controls */
        TAILQ_HEAD(mix_ctlhead, mix_ctl) ctls;  /* control list */
        TAILQ_ENTRY(mix_dev) devs;
};
```

The fields are follows:

parent_mixer Pointer to the mixer the device is attached to.

name Device name given by the OSS API. Devices can have one of the following names:

vol, bass, treble, synth, pcm, speaker, line, mic, cd, mix, pcm2, rec, igain, ogain, line1, line2, line3, dig1, dig2, dig3, phin, phout, video, radio, and monitor.

devno Device's index in the SOUND_MIXER_NRDEVICES macro defined in <*sys/soundcard.h*>. This number is used to check against the masks defined in the *mixer* structure.

left right Left and right-ear volumes. Although the OSS API stores volumes in integers from 0-100, we normalize them to 32-bit floating point numbers. However, the volumes can be denormalized using the *MIX_VOLDENORM* macro if needed.

nctl Number of user-defined mixer controls associated with the device.

ctls A tail queue containing user-defined mixer controls.

User-defined mixer controls

Each mixer device can have user-defined controls. The control structure is defined as follows:

```
struct mix_ctl {
        struct mix_dev *parent_dev;             /* parent device */
```

```
        int id;                                 /* control id */
        char name[NAME_MAX];                    /* control name */
        int (*mod)(struct mix_dev *, void *);   /* modify control values */
        int (*print)(struct mix_dev *, void *); /* print control */
        TAILQ_ENTRY(mix_ctl) ctls;
};
```

The fields are follows:

parent_dev Pointer to the device the control is attached to.

id Control ID assigned by the caller. Even though the library will report it, care has to be taken to not give a control the same ID in case the caller has to choose controls using their ID.

name Control name. As with *id*, the caller has to make sure the same name is not used more than once.

mod Function pointer to a control modification function. As in mixer(8), each mixer control's values can be modified. For example, if we have a volume control, the *mod* function will be responsible for handling volume changes.

print Function pointer to a control print function.

Opening and closing the mixer

The application must first call the **mixer_open**() function to obtain a handle to the device, which is used as an argument in most other functions and macros. The parameter *name* specifies the path to the mixer. OSS mixers are stored under */dev/mixerN* where *N* is the number of the mixer device. Each device maps to an actual *pcm* audio card, so */dev/mixer0* is the mixer for *pcm0*, and so on. If *name* is *NULL* or */dev/mixer*, **mixer_open**() opens the default mixer (hw.snd.default_unit).

The **mixer_close**() function frees resources and closes the mixer device. It is a good practice to always call it when the application is done using the mixer.

Manipulating the mixer

The **mixer_get_dev**() and **mixer_get_dev_byname**() functions select a mixer device, either by its number or by its name respectively. The mixer structure keeps a list of all the devices, but only one can be manipulated at a time. Each time a new device is to be manipulated, one of the two functions has to be called.

The **mixer_set_vol**() function changes the volume of the selected mixer device. The *vol* parameter is a structure that stores the left and right volumes of a given device. The allowed volume values are between MIX_VOLMIN (0.0) and MIX_VOLMAX (1.0).

The **mixer_set_mute**() function modifies the mute of a selected device. The *opt* parameter has to be one of the following options:

MIX_MUTE Mute the device.

MIX_UNMUTE Unmute the device.

MIX_TOGGLEMUTE Toggle the device’s mute (e.g mute if unmuted and unmute if muted).

The **mixer_mod_recsrc**() function modifies a recording device. The selected device has to be a recording device, otherwise the function will fail. The *opt* parameter has to be one of the following options:

MIX_ADDRECSRC Add device to the recording sources.

MIX_REMOVERECSRC Remove device from the recording sources.

MIX_SETRECSRC Set device as the only recording source.

MIX_TOGGLERECSRC Toggle device from the recording sources.

The **mixer_get_dunit**() and **mixer_set_dunit**() functions get and set the default audio card in the system. Although this is not really a mixer feature, it is useful to have instead of having to use the sysctl(3) controls.

The **mixer_get_mode**() function returns the operating mode of the audio device the mixer belongs to. The following values can be OR’ed in case more than one mode is supported:

MIX_MODE_MIXER The audio device has a mixer.

MIX_MODE_PLAY The audio device supports playback.

MIX_MODE_REC The audio device supports recording.

The **mixer_get_nmixers**() function returns the maximum mixer unit number. Although this might sound as incorrect behavior, given that one would expect "nmixers" to refer to the total number of active

mixers, it is more intuitive for applications that want to loop through all mixer devices (see the *EXAMPLES* section).

The **mixer_get_path**() function writes the path of the mixer device specified in the *unit* argument to the buffer specified in *buf*. *unit* can be either -1, in which case **mixer_get_path**() will fetch the path of the default mixer, or between 0 and the maximum mixer unit.

The **MIX_ISDEV**() macro checks if a device is actually a valid device for a given mixer. It is very unlikely that this macro will ever be needed since the library stores only valid devices by default.

The **MIX_ISMUTE**() macro checks if a device is muted.

The **MIX_ISREC**() macro checks if a device is a recording device.

The **MIX_ISRECSRC**() macro checks if a device is a recording source.

The **MIX_VOLNORM**() macro normalizes a value to 32-bit floating point number. It is used to normalize the volumes read from the OSS API.

The **MIX_VOLDENORM**() macro denormalizes the left and right volumes stores in the *mix_dev* structure.

Defining and using mixer controls

The **mix_add_ctl**() function creates a control and attaches it to the device specified in the *parent* argument.

The **mix_add_ctl_s**() function does the same thing as with **mix_add_ctl**() but the caller passes a *mix_ctl_t* * structure instead of each field as a separate argument.

The **mixer_remove_ctl**() functions removes a control from the device its attached to.

The **mixer_get_ctl**() function searches for a control in the device specified in the *d* argument and returns a pointer to it. The search is done using the control’s ID.

The **mixer_get_ctl_byname**() function is the same as with **mixer_get_ctl**() but the search is done using the control’s name.

RETURN VALUES

The **mixer_open**() function returns the newly created handle on success and NULL on failure.

The **mixer_close**(), **mixer_set_vol**(), **mixer_set_mute**(), **mixer_mod_recsrc**(), **mixer_get_dunut**(), **mixer_set_dunit**(), **mixer_get_nmixers**(), and **mixer_get_path**() functions return 0 or positive values on success and -1 on failure.

The **mixer_get_dev**() and **mixer_get_dev_byname**() functions return the selected device on success and NULL on failure.

All functions set the value of *errno* on failure.

EXAMPLES

Change the volume of a device

```
struct mixer *m;
mix_volume_t vol;
char *mix_name, *dev_name;

mix_name = ...;
if ((m = mixer_open(mix_name)) == NULL)
        err(1, "mixer_open: %s", mix_name);

dev_name = ...;
if ((m->dev = mixer_get_dev_byname(m, dev_name)) < 0)
        err(1, "unknown device: %s", dev_name);

vol.left = ...;
vol.right = ....;
if (mixer_set_vol(m, vol) < 0)
        warn("cannot change volume");

(void)mixer_close(m);
```

Mute all unmuted devices

```
struct mixer *m;
struct mix_dev *dp;

if ((m = mixer_open(NULL)) == NULL)          /* Open the default mixer. */
        err(1, "mixer_open");
TAILQ_FOREACH(dp, &m->devs, devs) {
        m->dev = dp;                         /* Select device. */
        if (M_ISMUTE(m, dp->devno))
                continue;
```

```
        if (mixer_set_mute(m, MIX_MUTE) < 0)
                warn("cannot mute device: %s", dp->name);
}

(void)mixer_close(m);
```

Print all recording sources' names and volumes

```
struct mixer *m;
struct mix_dev *dp;

char *mix_name, *dev_name;

mix_name = ...;
if ((m = mixer_open(mix_name)) == NULL)
        err(1, "mixer_open: %s", mix_name);

TAILQ_FOREACH(dp, &m->devs, devs) {
        if (M_ISRECSRC(m, dp->devno))
                printf("%s\t%.2f:%.2f\n",
                   dp->name, dp->vol.left, dp->vol.right);
}

(void)mixer_close(m);
```

Loop through all mixer devices in the system

```
struct mixer *m;
char buf[NAME_MAX];
int n;

if ((n = mixer_get_nmixers()) < 0)
        errx(1, "no mixers present in the system");
for (i = 0; i < n; i++) {
        (void)mixer_get_path(buf, sizeof(buf), i);
        if ((m = mixer_open(buf)) == NULL)
                continue;
        ...
        (void)mixer_close(m);
}
```

SEE ALSO

queue(3), sysctl(3), sound(4), mixer(8) and errno(2)

AUTHORS

Christos Margiolis <*christos@FreeBSD.org*>