

# **The ALSA Driver API**

---

# The ALSA Driver API

This document is free; 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 document 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

---

---

# Table of Contents

1. Management of Cards and Devices .....	1
Card Management .....	1
Device Components .....	9
Module requests and Device File Entries .....	12
Memory Management Helpers .....	16
2. PCM API .....	24
PCM Core .....	24
PCM Format Helpers .....	52
PCM Memory Management .....	66
3. Control/Mixer API .....	75
General Control Interface .....	75
AC97 Codec API .....	86
Virtual Master Control API .....	103
4. MIDI API .....	109
Raw MIDI API .....	109
MPU401-UART API .....	116
5. Proc Info API .....	120
Proc Info Interface .....	120
6. Miscellaneous Functions .....	129
Hardware-Dependent Devices API .....	129
Jack Abstraction Layer API .....	130
ISA DMA Helpers .....	134
Other Helper Macros .....	137

---

# **Chapter 1. Management of Cards and Devices**

## **Card Management**

## Name

`snd_card_new` — create and initialize a soundcard structure

## Synopsis

```
int snd_card_new (struct device * parent, int idx, const char * xid,  
struct module * module, int extra_size, struct snd_card ** card_ret);
```

## Arguments

<i>parent</i>	the parent device object
<i>idx</i>	card index (address) [0 ... (SNDRV_CARDS-1)]
<i>xid</i>	card identification (ASCII string)
<i>module</i>	top level module for locking
<i>extra_size</i>	allocate this extra size after the main soundcard structure
<i>card_ret</i>	the pointer to store the created card instance

## Description

Creates and initializes a soundcard structure.

The function allocates `snd_card` instance via `kzalloc` with the given space for the driver to use freely. The allocated struct is stored in the given `card_ret` pointer.

## Return

Zero if successful or a negative error code.

## Name

`snd_card_disconnect` — disconnect all APIs from the file-operations (user space)

## Synopsis

```
int snd_card_disconnect (struct snd_card * card);
```

## Arguments

*card*    soundcard structure

## Description

Disconnects all APIs from the file-operations (user space).

## Return

Zero, otherwise a negative error code.

## Note

The current implementation replaces all active file->f\_op with special dummy file operations (they do nothing except release).

## Name

`snd_card_set_id` — set card identification name

## Synopsis

```
void snd_card_set_id (struct snd_card * card, const char * nid);
```

## Arguments

*card*    soundcard structure

*nid*     new identification string

## Description

This function sets the card identification and checks for name collisions.

## Name

`snd_card_register` — register the soundcard

## Synopsis

```
int snd_card_register (struct snd_card * card);
```

## Arguments

*card* soundcard structure

## Description

This function registers all the devices assigned to the soundcard. Until calling this, the ALSA control interface is blocked from the external accesses. Thus, you should call this function at the end of the initialization of the card.

## Return

Zero otherwise a negative error code if the registration failed.



## Name

`snd_component_add` — add a component string

## Synopsis

```
int snd_component_add (struct snd_card * card, const char * component);
```

## Arguments

*card*            soundcard structure

*component*    the component id string

## Description

This function adds the component id string to the supported list. The component can be referred from the `alsa-lib`.

## Return

Zero otherwise a negative error code.

## Name

`snd_card_file_add` — add the file to the file list of the card

## Synopsis

```
int snd_card_file_add (struct snd_card * card, struct file * file);
```

## Arguments

*card* soundcard structure

*file* file pointer

## Description

This function adds the file to the file linked-list of the card. This linked-list is used to keep tracking the connection state, and to avoid the release of busy resources by hotplug.

## Return

zero or a negative error code.

## Name

`snd_card_file_remove` — remove the file from the file list

## Synopsis

```
int snd_card_file_remove (struct snd_card * card, struct file * file);
```

## Arguments

*card* soundcard structure

*file* file pointer

## Description

This function removes the file formerly added to the card via `snd_card_file_add` function. If all files are removed and `snd_card_free_when_closed` was called beforehand, it processes the pending release of resources.

## Return

Zero or a negative error code.

## Name

`snd_power_wait` — wait until the power-state is changed.

## Synopsis

```
int snd_power_wait (struct snd_card * card, unsigned int power_state);
```

## Arguments

*card*                      soundcard structure

*power\_state*    expected power state

## Description

Waits until the power-state is changed.

## Return

Zero if successful, or a negative error code.

## Note

the power lock must be active before call.

# Device Components

## Name

`snd_device_new` — create an ALSA device component

## Synopsis

```
int snd_device_new (struct snd_card * card, enum snd_device_type type,  
void * device_data, struct snd_device_ops * ops);
```

## Arguments

<i>card</i>	the card instance
<i>type</i>	the device type, <code>SNDRV_DEV_XXX</code>
<i>device_data</i>	the data pointer of this device
<i>ops</i>	the operator table

## Description

Creates a new device component for the given data pointer. The device will be assigned to the card and managed together by the card.

The data pointer plays a role as the identifier, too, so the pointer address must be unique and unchanged.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_device_free` — release the device from the card

## Synopsis

```
void snd_device_free (struct snd_card * card, void * device_data);
```

## Arguments

*card*                    the card instance

*device\_data*    the data pointer to release

## Description

Removes the device from the list on the card and invokes the callbacks, `dev_disconnect` and `dev_free`, corresponding to the state. Then release the device.

## Name

`snd_device_register` — register the device

## Synopsis

```
int snd_device_register (struct snd_card * card, void * device_data);
```

## Arguments

*card*                    the card instance

*device\_data*   the data pointer to register

## Description

Registers the device which was already created via `snd_device_new`. Usually this is called from `snd_card_register`, but it can be called later if any new devices are created after invocation of `snd_card_register`.

## Return

Zero if successful, or a negative error code on failure or if the device not found.

# Module requests and Device File Entries

## Name

`snd_request_card` — try to load the card module

## Synopsis

```
void snd_request_card (int card);
```

## Arguments

*card* the card number

## Description

Tries to load the module “snd-card-X” for the given card number via `request_module`. Returns immediately if already loaded.



## Name

`snd_lookup_minor_data` — get user data of a registered device

## Synopsis

```
void * snd_lookup_minor_data (unsigned int minor, int type);
```

## Arguments

*minor*    the minor number

*type*     device type (SNDRV\_DEVICE\_TYPE\_XXX)

## Description

Checks that a minor device with the specified type is registered, and returns its user data pointer.

This function increments the reference counter of the card instance if an associated instance with the given minor number and type is found. The caller must call `snd_card_unref` appropriately later.

## Return

The user data pointer if the specified device is found. NULL otherwise.

## Name

`snd_register_device_for_dev` — Register the ALSA device file for the card

## Synopsis

```
int snd_register_device_for_dev (int type, struct snd_card * card, int
dev, const struct file_operations * f_ops, void * private_data, const
char * name, struct device * device);
```

## Arguments

<i>type</i>	the device type, SNDRV_DEVICE_TYPE_XXX
<i>card</i>	the card instance
<i>dev</i>	the device index
<i>f_ops</i>	the file operations
<i>private_data</i>	user pointer for <i>f_ops</i> ->open
<i>name</i>	the device file name
<i>device</i>	the struct device to link this new device to

## Description

Registers an ALSA device file for the given card. The operators have to be set in *reg* parameter.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_unregister_device` — unregister the device on the given card

## Synopsis

```
int snd_unregister_device (int type, struct snd_card * card, int dev);
```

## Arguments

*type*    the device type, `SNDRV_DEVICE_TYPE_XXX`

*card*    the card instance

*dev*     the device index

## Description

Unregisters the device file already registered via `snd_register_device`.

## Return

Zero if successful, or a negative error code on failure.

# Memory Management Helpers

## Name

`copy_to_user_fromio` — copy data from mmio-space to user-space

## Synopsis

```
int copy_to_user_fromio (void __user * dst, const volatile void __iomem  
* src, size_t count);
```

## Arguments

*dst*      the destination pointer on user-space

*src*      the source pointer on mmio

*count*   the data size to copy in bytes

## Description

Copies the data from mmio-space to user-space.

## Return

Zero if successful, or non-zero on failure.

## Name

`copy_from_user_toio` — copy data from user-space to mmio-space

## Synopsis

```
int copy_from_user_toio (volatile void __iomem * dst, const void __user  
* src, size_t count);
```

## Arguments

*dst*      the destination pointer on mmio-space

*src*      the source pointer on user-space

*count*    the data size to copy in bytes

## Description

Copies the data from user-space to mmio-space.

## Return

Zero if successful, or non-zero on failure.

## Name

`snd_malloc_pages` — allocate pages with the given size

## Synopsis

```
void * snd_malloc_pages (size_t size, gfp_t gfp_flags);
```

## Arguments

*size*            the size to allocate in bytes

*gfp\_flags*    the allocation conditions, GFP\_XXX

## Description

Allocates the physically contiguous pages with the given size.

## Return

The pointer of the buffer, or NULL if no enough memory.

## Name

`snd_free_pages` — release the pages

## Synopsis

```
void snd_free_pages (void * ptr, size_t size);
```

## Arguments

*ptr*     the buffer pointer to release

*size*    the allocated buffer size

## Description

Releases the buffer allocated via `snd_malloc_pages`.

## Name

`snd_dma_alloc_pages` — allocate the buffer area according to the given type

## Synopsis

```
int snd_dma_alloc_pages (int type, struct device * device, size_t size,  
struct snd_dma_buffer * dmab);
```

## Arguments

*type*      the DMA buffer type

*device*    the device pointer

*size*      the buffer size to allocate

*dmab*      buffer allocation record to store the allocated data

## Description

Calls the memory-allocator function for the corresponding buffer type.

## Return

Zero if the buffer with the given size is allocated successfully, otherwise a negative value on error.



## Name

`snd_dma_alloc_pages_fallback` — allocate the buffer area according to the given type with fallback

## Synopsis

```
int snd_dma_alloc_pages_fallback (int type, struct device * device,
size_t size, struct snd_dma_buffer * dmab);
```

## Arguments

*type*      the DMA buffer type

*device*    the device pointer

*size*      the buffer size to allocate

*dmab*      buffer allocation record to store the allocated data

## Description

Calls the memory-allocator function for the corresponding buffer type. When no space is left, this function reduces the size and tries to allocate again. The size actually allocated is stored in `res_size` argument.

## Return

Zero if the buffer with the given size is allocated successfully, otherwise a negative value on error.

## Name

`snd_dma_free_pages` — release the allocated buffer

## Synopsis

```
void snd_dma_free_pages (struct snd_dma_buffer * dmab);
```

## Arguments

*dmab* the buffer allocation record to release

## Description

Releases the allocated buffer via `snd_dma_alloc_pages`.

---

# Chapter 2. PCM API

## PCM Core

## Name

`snd_pcm_new_stream` — create a new PCM stream

## Synopsis

```
int  snd_pcm_new_stream (struct snd_pcm * pcm, int stream, int
                          substream_count);
```

## Arguments

<i>pcm</i>	the pcm instance
<i>stream</i>	the stream direction, <code>SNDRV_PCM_STREAM_XXX</code>
<i>substream_count</i>	the number of substreams

## Description

Creates a new stream for the pcm. The corresponding stream on the pcm must have been empty before calling this, i.e. zero must be given to the argument of `snd_pcm_new`.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_pcm_new` — create a new PCM instance

## Synopsis

```
int snd_pcm_new (struct snd_card * card, const char * id, int device,  
int playback_count, int capture_count, struct snd_pcm ** rpcm);
```

## Arguments

<i>card</i>	the card instance
<i>id</i>	the id string
<i>device</i>	the device index (zero based)
<i>playback_count</i>	the number of substreams for playback
<i>capture_count</i>	the number of substreams for capture
<i>rpcm</i>	the pointer to store the new pcm instance

## Description

Creates a new PCM instance.

The pcm operators have to be set afterwards to the new instance via `snd_pcm_set_ops`.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_pcm_new_internal` — create a new internal PCM instance

## Synopsis

```
int snd_pcm_new_internal (struct snd_card * card, const char * id, int
device, int playback_count, int capture_count, struct snd_pcm ** rpcm);
```

## Arguments

<i>card</i>	the card instance
<i>id</i>	the id string
<i>device</i>	the device index (zero based - shared with normal PCM's)
<i>playback_count</i>	the number of substreams for playback
<i>capture_count</i>	the number of substreams for capture
<i>rpcm</i>	the pointer to store the new pcm instance

## Description

Creates a new internal PCM instance with no userspace device or procfs entries. This is used by ASoC Back End PCM's in order to create a PCM that will only be used internally by kernel drivers. i.e. it cannot be opened by userspace. It provides existing ASoC components drivers with a substream and access to any private data.

The pcm operators have to be set afterwards to the new instance via `snd_pcm_set_ops`.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_pcm_set_ops` — set the PCM operators

## Synopsis

```
void snd_pcm_set_ops (struct snd_pcm * pcm, int direction, const struct  
snd_pcm_ops * ops);
```

## Arguments

<i>pcm</i>	the pcm instance
<i>direction</i>	stream direction, <code>SNDRV_PCM_STREAM_XXX</code>
<i>ops</i>	the operator table

## Description

Sets the given PCM operators to the pcm instance.

## Name

`snd_pcm_set_sync` — set the PCM sync id

## Synopsis

```
void snd_pcm_set_sync (struct snd_pcm_substream * substream);
```

## Arguments

*substream* the pcm substream

## Description

Sets the PCM sync identifier for the card.



## Name

`snd_interval_refine` — refine the interval value of configurator

## Synopsis

```
int  snd_interval_refine (struct  snd_interval  *  i,  const  struct
snd_interval  *  v);
```

## Arguments

*i* the interval value to refine

*v* the interval value to refer to

## Description

Refines the interval value with the reference value. The interval is changed to the range satisfying both intervals. The interval status (min, max, integer, etc.) are evaluated.

## Return

Positive if the value is changed, zero if it's not changed, or a negative error code.

## Name

`snd_interval_ratnum` — refine the interval value

## Synopsis

```
int snd_interval_ratnum (struct snd_interval * i, unsigned int
    rats_count, struct snd_ratnum * rats, unsigned int * nump, unsigned
    int * denp);
```

## Arguments

<i>i</i>	interval to refine
<i>rats_count</i>	number of <code>ratnum_t</code>
<i>rats</i>	<code>ratnum_t</code> array
<i>nump</i>	pointer to store the resultant numerator
<i>denp</i>	pointer to store the resultant denominator

## Return

Positive if the value is changed, zero if it's not changed, or a negative error code.

## Name

`snd_interval_list` — refine the interval value from the list

## Synopsis

```
int snd_interval_list (struct snd_interval * i, unsigned int count,  
const unsigned int * list, unsigned int mask);
```

## Arguments

<i>i</i>	the interval value to refine
<i>count</i>	the number of elements in the list
<i>list</i>	the value list
<i>mask</i>	the bit-mask to evaluate

## Description

Refines the interval value from the list. When mask is non-zero, only the elements corresponding to bit 1 are evaluated.

## Return

Positive if the value is changed, zero if it's not changed, or a negative error code.

## Name

`snd_pcm_hw_rule_add` — add the hw-constraint rule

## Synopsis

```
int snd_pcm_hw_rule_add (struct snd_pcm_runtime * runtime, unsigned int
cond, int var, snd_pcm_hw_rule_func_t func, void * private, int dep,
...);
```

## Arguments

<i>runtime</i>	the pcm runtime instance
<i>cond</i>	condition bits
<i>var</i>	the variable to evaluate
<i>func</i>	the evaluation function
<i>private</i>	the private data pointer passed to function
<i>dep</i>	the dependent variables
...	variable arguments

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_pcm_hw_constraint_mask64` — apply the given bitmap mask constraint

## Synopsis

```
int snd_pcm_hw_constraint_mask64 (struct snd_pcm_runtime * runtime,
snd_pcm_hw_param_t var, u_int64_t mask);
```

## Arguments

*runtime*    PCM runtime instance

*var*        hw\_params variable to apply the mask

*mask*       the 64bit bitmap mask

## Description

Apply the constraint of the given bitmap mask to a 64-bit mask parameter.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_pcm_hw_constraint_integer` — apply an integer constraint to an interval

## Synopsis

```
int snd_pcm_hw_constraint_integer (struct snd_pcm_runtime * runtime,  
snd_pcm_hw_param_t var);
```

## Arguments

*runtime*    PCM runtime instance

*var*        hw\_params variable to apply the integer constraint

## Description

Apply the constraint of integer to an interval parameter.

## Return

Positive if the value is changed, zero if it's not changed, or a negative error code.

## Name

`snd_pcm_hw_constraint_minmax` — apply a min/max range constraint to an interval

## Synopsis

```
int snd_pcm_hw_constraint_minmax (struct snd_pcm_runtime * runtime,
snd_pcm_hw_param_t var, unsigned int min, unsigned int max);
```

## Arguments

<i>runtime</i>	PCM runtime instance
<i>var</i>	hw_params variable to apply the range
<i>min</i>	the minimal value
<i>max</i>	the maximal value

## Description

Apply the min/max range constraint to an interval parameter.

## Return

Positive if the value is changed, zero if it's not changed, or a negative error code.

## Name

`snd_pcm_hw_constraint_list` — apply a list of constraints to a parameter

## Synopsis

```
int  snd_pcm_hw_constraint_list (struct snd_pcm_runtime * runtime,
unsigned int  cond,  snd_pcm_hw_param_t  var,  const struct
snd_pcm_hw_constraint_list * l);
```

## Arguments

<i>runtime</i>	PCM runtime instance
<i>cond</i>	condition bits
<i>var</i>	hw_params variable to apply the list constraint
<i>l</i>	list

## Description

Apply the list of constraints to an interval parameter.

## Return

Zero if successful, or a negative error code on failure.



## Name

`snd_pcm_hw_constraint_ratnums` — apply ratnums constraint to a parameter

## Synopsis

```
int      snd_pcm_hw_constraint_ratnums      (struct      snd_pcm_runtime      *  
runtime,      unsigned      int      cond,      snd_pcm_hw_param_t      var,      struct  
snd_pcm_hw_constraint_ratnums * r);
```

## Arguments

<i>runtime</i>	PCM runtime instance
<i>cond</i>	condition bits
<i>var</i>	hw_params variable to apply the ratnums constraint
<i>r</i>	struct snd_ratnums constraints

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_pcm_hw_constraint_ratdens` — apply ratdens constraint to a parameter

## Synopsis

```
int      snd_pcm_hw_constraint_ratdens      (struct      snd_pcm_runtime      *  
runtime,      unsigned      int      cond,      snd_pcm_hw_param_t      var,      struct  
snd_pcm_hw_constraint_ratdens * r);
```

## Arguments

<i>runtime</i>	PCM runtime instance
<i>cond</i>	condition bits
<i>var</i>	hw_params variable to apply the ratdens constraint
<i>r</i>	struct snd_ratdens constraints

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_pcm_hw_constraint_msbits` — add a hw constraint msbits rule

## Synopsis

```
int snd_pcm_hw_constraint_msbits (struct snd_pcm_runtime * runtime,
unsigned int cond, unsigned int width, unsigned int msbits);
```

## Arguments

*runtime*    PCM runtime instance

*cond*       condition bits

*width*      sample bits width

*msbits*     msbits width

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_pcm_hw_constraint_step` — add a hw constraint step rule

## Synopsis

```
int  snd_pcm_hw_constraint_step (struct snd_pcm_runtime * runtime,
unsigned int cond, snd_pcm_hw_param_t var, unsigned long step);
```

## Arguments

*runtime* PCM runtime instance

*cond* condition bits

*var* hw\_params variable to apply the step constraint

*step* step size

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_pcm_hw_constraint_pow2` — add a hw constraint power-of-2 rule

## Synopsis

```
int  snd_pcm_hw_constraint_pow2 (struct snd_pcm_runtime * runtime,
unsigned int cond, snd_pcm_hw_param_t var);
```

## Arguments

*runtime* PCM runtime instance

*cond* condition bits

*var* hw\_params variable to apply the power-of-2 constraint

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_pcm_hw_rule_noresample` — add a rule to allow disabling hw resampling

## Synopsis

```
int  snd_pcm_hw_rule_noresample (struct snd_pcm_runtime * runtime,
unsigned int base_rate);
```

## Arguments

*runtime*      PCM runtime instance

*base\_rate*    the rate at which the hardware does not resample

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_pcm_hw_param_value` — return *params* field *var* value

## Synopsis

```
int snd_pcm_hw_param_value (const struct snd_pcm_hw_params * params,
snd_pcm_hw_param_t var, int * dir);
```

## Arguments

*params*    the hw\_params instance

*var*       parameter to retrieve

*dir*       pointer to the direction (-1,0,1) or NULL

## Return

The value for field *var* if it's fixed in configuration space defined by *params*. -EINVAL otherwise.

## Name

`snd_pcm_hw_param_first` — refine config space and return minimum value

## Synopsis

```
int snd_pcm_hw_param_first (struct snd_pcm_substream * pcm, struct
snd_pcm_hw_params * params, snd_pcm_hw_param_t var, int * dir);
```

## Arguments

<i>pcm</i>	PCM instance
<i>params</i>	the hw_params instance
<i>var</i>	parameter to retrieve
<i>dir</i>	pointer to the direction (-1,0,1) or NULL

## Description

Inside configuration space defined by *params* remove from *var* all values > minimum. Reduce configuration space accordingly.

## Return

The minimum, or a negative error code on failure.



## Name

`snd_pcm_hw_param_last` — refine config space and return maximum value

## Synopsis

```
int snd_pcm_hw_param_last (struct snd_pcm_substream * pcm, struct
snd_pcm_hw_params * params, snd_pcm_hw_param_t var, int * dir);
```

## Arguments

<i>pcm</i>	PCM instance
<i>params</i>	the hw_params instance
<i>var</i>	parameter to retrieve
<i>dir</i>	pointer to the direction (-1,0,1) or NULL

## Description

Inside configuration space defined by *params* remove from *var* all values < maximum. Reduce configuration space accordingly.

## Return

The maximum, or a negative error code on failure.

## Name

`snd_pcm_lib_ioctl` — a generic PCM ioctl callback

## Synopsis

```
int snd_pcm_lib_ioctl (struct snd_pcm_substream * substream, unsigned
int cmd, void * arg);
```

## Arguments

*substream*    the pcm substream instance

*cmd*            ioctl command

*arg*            ioctl argument

## Description

Processes the generic ioctl commands for PCM. Can be passed as the ioctl callback for PCM ops.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_pcm_period_elapsed` — update the pcm status for the next period

## Synopsis

```
void snd_pcm_period_elapsed (struct snd_pcm_substream * substream);
```

## Arguments

*substream* the pcm substream instance

## Description

This function is called from the interrupt handler when the PCM has processed the period size. It will update the current pointer, wake up sleepers, etc.

Even if more than one periods have elapsed since the last call, you have to call this only once.

## Name

`snd_pcm_add_chmap_ctls` — create channel-mapping control elements

## Synopsis

```
int snd_pcm_add_chmap_ctls (struct snd_pcm * pcm, int stream, const
struct snd_pcm_chmap_elem * chmap, int max_channels, unsigned long
private_value, struct snd_pcm_chmap ** info_ret);
```

## Arguments

<i>pcm</i>	the assigned PCM instance
<i>stream</i>	stream direction
<i>chmap</i>	channel map elements (for query)
<i>max_channels</i>	the max number of channels for the stream
<i>private_value</i>	the value passed to each kcontrol's <code>private_value</code> field
<i>info_ret</i>	store struct <code>snd_pcm_chmap</code> instance if non-NULL

## Description

Create channel-mapping control elements assigned to the given PCM stream(s).

## Return

Zero if successful, or a negative error value.

## Name

`snd_pcm_stop` — try to stop all running streams in the substream group

## Synopsis

```
int snd_pcm_stop (struct snd_pcm_substream * substream, snd_pcm_state_t
state);
```

## Arguments

*substream*    the PCM substream instance

*state*        PCM state after stopping the stream

## Description

The state of each stream is then changed to the given state unconditionally.

## Return

Zero if successful, or a negative error code.

## Name

`snd_pcm_suspend` — trigger SUSPEND to all linked streams

## Synopsis

```
int snd_pcm_suspend (struct snd_pcm_substream * substream);
```

## Arguments

*substream* the PCM substream

## Description

After this call, all streams are changed to SUSPENDED state.

## Return

Zero if successful (or *substream* is NULL), or a negative error code.

## Name

`snd_pcm_suspend_all` — trigger SUSPEND to all substreams in the given pcm

## Synopsis

```
int snd_pcm_suspend_all (struct snd_pcm * pcm);
```

## Arguments

*pcm* the PCM instance

## Description

After this call, all streams are changed to SUSPENDED state.

## Return

Zero if successful (or *pcm* is NULL), or a negative error code.

# PCM Format Helpers

## Name

`snd_pcm_format_signed` — Check the PCM format is signed linear

## Synopsis

```
int snd_pcm_format_signed (snd_pcm_format_t format);
```

## Arguments

*format* the format to check

## Return

1 if the given PCM format is signed linear, 0 if unsigned linear, and a negative error code for non-linear formats.



## Name

`snd_pcm_format_unsigned` — Check the PCM format is unsigned linear

## Synopsis

```
int snd_pcm_format_unsigned (snd_pcm_format_t format);
```

## Arguments

*format* the format to check

## Return

1 if the given PCM format is unsigned linear, 0 if signed linear, and a negative error code for non-linear formats.

## Name

`snd_pcm_format_linear` — Check the PCM format is linear

## Synopsis

```
int snd_pcm_format_linear (snd_pcm_format_t format);
```

## Arguments

*format* the format to check

## Return

1 if the given PCM format is linear, 0 if not.

## Name

`snd_pcm_format_little_endian` — Check the PCM format is little-endian

## Synopsis

```
int snd_pcm_format_little_endian (snd_pcm_format_t format);
```

## Arguments

*format* the format to check

## Return

1 if the given PCM format is little-endian, 0 if big-endian, or a negative error code if endian not specified.

## Name

`snd_pcm_format_big_endian` — Check the PCM format is big-endian

## Synopsis

```
int snd_pcm_format_big_endian (snd_pcm_format_t format);
```

## Arguments

*format* the format to check

## Return

1 if the given PCM format is big-endian, 0 if little-endian, or a negative error code if endian not specified.

## Name

`snd_pcm_format_width` — return the bit-width of the format

## Synopsis

```
int snd_pcm_format_width (snd_pcm_format_t format);
```

## Arguments

*format* the format to check

## Return

The bit-width of the format, or a negative error code if unknown format.

## Name

`snd_pcm_format_physical_width` — return the physical bit-width of the format

## Synopsis

```
int snd_pcm_format_physical_width (snd_pcm_format_t format);
```

## Arguments

*format* the format to check

## Return

The physical bit-width of the format, or a negative error code if unknown format.

## Name

`snd_pcm_format_size` — return the byte size of samples on the given format

## Synopsis

```
ssize_t snd_pcm_format_size (snd_pcm_format_t format, size_t samples);
```

## Arguments

*format*     the format to check

*samples*   sampling rate

## Return

The byte size of the given samples for the format, or a negative error code if unknown format.

## Name

`snd_pcm_format_silence_64` — return the silent data in 8 bytes array

## Synopsis

```
const unsigned char * snd_pcm_format_silence_64 (snd_pcm_format_t
format);
```

## Arguments

*format* the format to check

## Return

The format pattern to fill or NULL if error.



## Name

`snd_pcm_format_set_silence` — set the silence data on the buffer

## Synopsis

```
int snd_pcm_format_set_silence (snd_pcm_format_t format, void * data,  
unsigned int samples);
```

## Arguments

*format*     the PCM format

*data*       the buffer pointer

*samples*   the number of samples to set silence

## Description

Sets the silence data on the buffer for the given samples.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_pcm_limit_hw_rates` — determine rate\_min/rate\_max fields

## Synopsis

```
int snd_pcm_limit_hw_rates (struct snd_pcm_runtime * runtime);
```

## Arguments

*runtime* the runtime instance

## Description

Determines the rate\_min and rate\_max fields from the rates bits of the given runtime->hw.

## Return

Zero if successful.

## Name

`snd_pcm_rate_to_rate_bit` — converts sample rate to `SNDRV_PCM_RATE_XXX` bit

## Synopsis

```
unsigned int snd_pcm_rate_to_rate_bit (unsigned int rate);
```

## Arguments

*rate* the sample rate to convert

## Return

The `SNDRV_PCM_RATE_XXX` flag that corresponds to the given rate, or `SNDRV_PCM_RATE_KNOT` for an unknown rate.

## Name

`snd_pcm_rate_bit_to_rate` — converts `SNDRV_PCM_RATE_xxx` bit to sample rate

## Synopsis

```
unsigned int snd_pcm_rate_bit_to_rate (unsigned int rate_bit);
```

## Arguments

*rate\_bit* the rate bit to convert

## Return

The sample rate that corresponds to the given `SNDRV_PCM_RATE_xxx` flag or 0 for an unknown rate bit.

## Name

`snd_pcm_rate_mask_intersect` — computes the intersection between two rate masks

## Synopsis

```
unsigned int snd_pcm_rate_mask_intersect (unsigned int rates_a, unsigned
int rates_b);
```

## Arguments

*rates\_a*    The first rate mask

*rates\_b*    The second rate mask

## Description

This function computes the rates that are supported by both rate masks passed to the function. It will take care of the special handling of `SNDRV_PCM_RATE_CONTINUOUS` and `SNDRV_PCM_RATE_KNOT`.

## Return

A rate mask containing the rates that are supported by both `rates_a` and `rates_b`.

# PCM Memory Management

## Name

`snd_pcm_lib_preallocate_free_for_all` — release all pre-allocated buffers on the pcm

## Synopsis

```
int snd_pcm_lib_preallocate_free_for_all (struct snd_pcm * pcm);
```

## Arguments

*pcm* the pcm instance

## Description

Releases all the pre-allocated buffers on the given pcm.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_pcm_lib_preallocate_pages` — pre-allocation for the given DMA type

## Synopsis

```
int    snd_pcm_lib_preallocate_pages    (struct    snd_pcm_substream    *  
substream, int type, struct device * data, size_t size, size_t max);
```

## Arguments

<i>substream</i>	the pcm substream instance
<i>type</i>	DMA type (SNDRV_DMA_TYPE_*)
<i>data</i>	DMA type dependent data
<i>size</i>	the requested pre-allocation size in bytes
<i>max</i>	the max. allowed pre-allocation size

## Description

Do pre-allocation for the given DMA buffer type.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_pcm_lib_preallocate_pages_for_all` — pre-allocation for continuous memory type (all substreams)

## Synopsis

```
int snd_pcm_lib_preallocate_pages_for_all (struct snd_pcm * pcm, int
type, void * data, size_t size, size_t max);
```

## Arguments

*pcm*     the pcm instance

*type*    DMA type (SNDRV\_DMA\_TYPE\_\*)

*data*    DMA type dependent data

*size*    the requested pre-allocation size in bytes

*max*     the max. allowed pre-allocation size

## Description

Do pre-allocation to all substreams of the given pcm for the specified DMA type.

## Return

Zero if successful, or a negative error code on failure.



## Name

`snd_pcm_sgbuf_ops_page` — get the page struct at the given offset

## Synopsis

```
struct page * snd_pcm_sgbuf_ops_page (struct snd_pcm_substream *  
substream, unsigned long offset);
```

## Arguments

*substream* the pcm substream instance

*offset* the buffer offset

## Description

Used as the page callback of PCM ops.

## Return

The page struct at the given buffer offset. NULL on failure.

## Name

`snd_pcm_lib_malloc_pages` — allocate the DMA buffer

## Synopsis

```
int snd_pcm_lib_malloc_pages (struct snd_pcm_substream * substream,
size_t size);
```

## Arguments

*substream* the substream to allocate the DMA buffer to

*size* the requested buffer size in bytes

## Description

Allocates the DMA buffer on the BUS type given earlier to `snd_pcm_lib_preallocate_XXX_pages`.

## Return

1 if the buffer is changed, 0 if not changed, or a negative code on failure.

## Name

`snd_pcm_lib_free_pages` — release the allocated DMA buffer.

## Synopsis

```
int snd_pcm_lib_free_pages (struct snd_pcm_substream * substream);
```

## Arguments

*substream* the substream to release the DMA buffer

## Description

Releases the DMA buffer allocated via `snd_pcm_lib_malloc_pages`.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_pcm_lib_free_vmalloc_buffer` — free vmalloc buffer

## Synopsis

```
int snd_pcm_lib_free_vmalloc_buffer (struct snd_pcm_substream *  
    substream);
```

## Arguments

*substream* the substream with a buffer allocated by `snd_pcm_lib_alloc_vmalloc_buffer`

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_pcm_lib_get_vmalloc_page` — map vmalloc buffer offset to page struct

## Synopsis

```
struct page * snd_pcm_lib_get_vmalloc_page (struct snd_pcm_substream *  
substream, unsigned long offset);
```

## Arguments

*substream*    the substream with a buffer allocated by `snd_pcm_lib_alloc_vmalloc_buffer`

*offset*        offset in the buffer

## Description

This function is to be used as the page callback in the PCM ops.

## Return

The page struct, or NULL on failure.

---

# **Chapter 3. Control/Mixer API**

## **General Control Interface**

## Name

`snd_ctl_new1` — create a control instance from the template

## Synopsis

```
struct snd_kcontrol * snd_ctl_new1 (const struct snd_kcontrol_new *  
ncontrol, void * private_data);
```

## Arguments

*ncontrol*            the initialization record

*private\_data*    the private data to set

## Description

Allocates a new struct `snd_kcontrol` instance and initialize from the given template. When the access field of `ncontrol` is 0, it's assumed as `READWRITE` access. When the count field is 0, it's assumes as one.

## Return

The pointer of the newly generated instance, or `NULL` on failure.

## Name

`snd_ctl_free_one` — release the control instance

## Synopsis

```
void snd_ctl_free_one (struct snd_kcontrol * kcontrol);
```

## Arguments

*kcontrol* the control instance

## Description

Releases the control instance created via `snd_ctl_new` or `snd_ctl_new1`. Don't call this after the control was added to the card.



## Name

`snd_ctl_add` — add the control instance to the card

## Synopsis

```
int snd_ctl_add (struct snd_card * card, struct snd_kcontrol * kcontrol);
```

## Arguments

*card*            the card instance

*kcontrol*       the control instance to add

## Description

Adds the control instance created via `snd_ctl_new` or `snd_ctl_new1` to the given card. Assigns also an unique numid used for fast search.

It frees automatically the control which cannot be added.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_ctl_replace` — replace the control instance of the card

## Synopsis

```
int snd_ctl_replace (struct snd_card * card, struct snd_kcontrol *  
kcontrol, bool add_on_replace);
```

## Arguments

<i>card</i>	the card instance
<i>kcontrol</i>	the control instance to replace
<i>add_on_replace</i>	add the control if not already added

## Description

Replaces the given control. If the given control does not exist and the `add_on_replace` flag is set, the control is added. If the control exists, it is destroyed first.

It frees automatically the control which cannot be added or replaced.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_ctl_remove` — remove the control from the card and release it

## Synopsis

```
int snd_ctl_remove (struct snd_card * card, struct snd_kcontrol *  
kcontrol);
```

## Arguments

*card*            the card instance

*kcontrol*       the control instance to remove

## Description

Removes the control from the card and then releases the instance. You don't need to call `snd_ctl_free_one`. You must be in the write lock - `down_write(card->controls_rwsem)`.

## Return

0 if successful, or a negative error code on failure.

## Name

`snd_ctl_remove_id` — remove the control of the given id and release it

## Synopsis

```
int snd_ctl_remove_id (struct snd_card * card, struct snd_ctl_elem_id  
* id);
```

## Arguments

*card*    the card instance

*id*      the control id to remove

## Description

Finds the control instance with the given id, removes it from the card list and releases it.

## Return

0 if successful, or a negative error code on failure.

## Name

`snd_ctl_activate_id` — activate/inactivate the control of the given id

## Synopsis

```
int snd_ctl_activate_id (struct snd_card * card, struct snd_ctl_elem_id  
* id, int active);
```

## Arguments

*card*      the card instance

*id*        the control id to activate/inactivate

*active*    non-zero to activate

## Description

Finds the control instance with the given id, and activate or inactivate the control together with notification, if changed.

## Return

0 if unchanged, 1 if changed, or a negative error code on failure.

## Name

`snd_ctl_rename_id` — replace the id of a control on the card

## Synopsis

```
int snd_ctl_rename_id (struct snd_card * card, struct snd_ctl_elem_id  
* src_id, struct snd_ctl_elem_id * dst_id);
```

## Arguments

*card*      the card instance

*src\_id*    the old id

*dst\_id*    the new id

## Description

Finds the control with the old id from the card, and replaces the id with the new one.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_ctl_find_numid` — find the control instance with the given number-id

## Synopsis

```
struct snd_kcontrol * snd_ctl_find_numid (struct snd_card * card,
unsigned int numid);
```

## Arguments

*card*    the card instance

*numid*   the number-id to search

## Description

Finds the control instance with the given number-id from the card.

The caller must down `card->controls_rwsem` before calling this function (if the race condition can happen).

## Return

The pointer of the instance if found, or `NULL` if not.

## Name

`snd_ctl_find_id` — find the control instance with the given id

## Synopsis

```
struct snd_kcontrol * snd_ctl_find_id (struct snd_card * card, struct
snd_ctl_elem_id * id);
```

## Arguments

*card* the card instance

*id* the id to search

## Description

Finds the control instance with the given id from the card.

The caller must down `card->controls_rwsem` before calling this function (if the race condition can happen).

## Return

The pointer of the instance if found, or `NULL` if not.



## Name

`snd_ctl_enum_info` — fills the info structure for an enumerated control

## Synopsis

```
int snd_ctl_enum_info (struct snd_ctl_elem_info * info, unsigned int
channels, unsigned int items, const char *const names[]);
```

## Arguments

<i>info</i>	the structure to be filled
<i>channels</i>	the number of the control's channels; often one
<i>items</i>	the number of control values; also the size of <i>names</i>
<i>names</i> []	an array containing the names of all control values

## Description

Sets all required fields in *info* to their appropriate values. If the control's accessibility is not the default (readable and writable), the caller has to fill *info->access*.

## Return

Zero.

# AC97 Codec API

## Name

`snd_ac97_write` — write a value on the given register

## Synopsis

```
void snd_ac97_write (struct snd_ac97 * ac97, unsigned short reg, unsigned short value);
```

## Arguments

*ac97*     the ac97 instance

*reg*     the register to change

*value*   the value to set

## Description

Writes a value on the given register. This will invoke the write callback directly after the register check. This function doesn't change the register cache unlike `#snd_ca97_write_cache`, so use this only when you don't want to reflect the change to the suspend/resume state.

## Name

`snd_ac97_read` — read a value from the given register

## Synopsis

```
unsigned short snd_ac97_read (struct snd_ac97 * ac97, unsigned short  
reg);
```

## Arguments

*ac97*    the ac97 instance

*reg*    the register to read

## Description

Reads a value from the given register. This will invoke the read callback directly after the register check.

## Return

The read value.

## Name

`snd_ac97_write_cache` — write a value on the given register and update the cache

## Synopsis

```
void snd_ac97_write_cache (struct snd_ac97 * ac97, unsigned short reg,  
unsigned short value);
```

## Arguments

*ac97*     the ac97 instance

*reg*     the register to change

*value*   the value to set

## Description

Writes a value on the given register and updates the register cache. The cached values are used for the cached-read and the suspend/resume.

## Name

`snd_ac97_update` — update the value on the given register

## Synopsis

```
int snd_ac97_update (struct snd_ac97 * ac97, unsigned short reg, unsigned short value);
```

## Arguments

*ac97*     the *ac97* instance

*reg*     the register to change

*value*   the value to set

## Description

Compares the value with the register cache and updates the value only when the value is changed.

## Return

1 if the value is changed, 0 if no change, or a negative code on failure.

## Name

`snd_ac97_update_bits` — update the bits on the given register

## Synopsis

```
int snd_ac97_update_bits (struct snd_ac97 * ac97, unsigned short reg,  
unsigned short mask, unsigned short value);
```

## Arguments

*ac97*    the `ac97` instance

*reg*     the register to change

*mask*    the bit-mask to change

*value*   the value to set

## Description

Updates the masked-bits on the given register only when the value is changed.

## Return

1 if the bits are changed, 0 if no change, or a negative code on failure.

## Name

`snd_ac97_get_short_name` — retrieve codec name

## Synopsis

```
const char * snd_ac97_get_short_name (struct snd_ac97 * ac97);
```

## Arguments

*ac97* the codec instance

## Return

The short identifying name of the codec.

## Name

`snd_ac97_bus` — create an AC97 bus component

## Synopsis

```
int  snd_ac97_bus (struct snd_card * card, int num, struct
snd_ac97_bus_ops * ops, void * private_data, struct snd_ac97_bus **
rbus);
```

## Arguments

<i>card</i>	the card instance
<i>num</i>	the bus number
<i>ops</i>	the bus callbacks table
<i>private_data</i>	private data pointer for the new instance
<i>rbus</i>	the pointer to store the new AC97 bus instance.

## Description

Creates an AC97 bus component. An struct `snd_ac97_bus` instance is newly allocated and initialized.

The ops table must include valid callbacks (at least read and write). The other callbacks, wait and reset, are not mandatory.

The clock is set to 48000. If another clock is needed, set `(*rbus)->clock` manually.

The AC97 bus instance is registered as a low-level device, so you don't have to release it manually.

## Return

Zero if successful, or a negative error code on failure.



## Name

`snd_ac97_mixer` — create an Codec97 component

## Synopsis

```
int snd_ac97_mixer (struct snd_ac97_bus * bus, struct snd_ac97_template  
* template, struct snd_ac97 ** rac97);
```

## Arguments

*bus*            the AC97 bus which codec is attached to

*template*    the template of ac97, including index, callbacks and the private data.

*rac97*        the pointer to store the new ac97 instance.

## Description

Creates an Codec97 component. An struct `snd_ac97` instance is newly allocated and initialized from the template. The codec is then initialized by the standard procedure.

The template must include the codec number (`num`) and address (`addr`), and the private data (`private_data`).

The `ac97` instance is registered as a low-level device, so you don't have to release it manually.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_ac97_update_power` — update the powerdown register

## Synopsis

```
int snd_ac97_update_power (struct snd_ac97 * ac97, int reg, int powerup);
```

## Arguments

*ac97*        the codec instance

*reg*        the rate register, e.g. `AC97_PCM_FRONT_DAC_RATE`

*powerup*    non-zero when power up the part

## Description

Update the AC97 powerdown register bits of the given part.

## Return

Zero.

## Name

`snd_ac97_suspend` — General suspend function for AC97 codec

## Synopsis

```
void snd_ac97_suspend (struct snd_ac97 * ac97);
```

## Arguments

*ac97* the ac97 instance

## Description

Suspends the codec, power down the chip.

## Name

`snd_ac97_resume` — General resume function for AC97 codec

## Synopsis

```
void snd_ac97_resume (struct snd_ac97 * ac97);
```

## Arguments

*ac97* the ac97 instance

## Description

Do the standard resume procedure, power up and restoring the old register values.

## Name

`snd_ac97_tune_hardware` — tune up the hardware

## Synopsis

```
int snd_ac97_tune_hardware (struct snd_ac97 * ac97, struct ac97_quirk  
* quirk, const char * override);
```

## Arguments

<i>ac97</i>	the ac97 instance
<i>quirk</i>	quirk list
<i>override</i>	explicit quirk value (overrides the list if non-NULL)

## Description

Do some workaround for each pci device, such as renaming of the headphone (true line-out) control as “Master”. The quirk-list must be terminated with a zero-filled entry.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_ac97_set_rate` — change the rate of the given input/output.

## Synopsis

```
int snd_ac97_set_rate (struct snd_ac97 * ac97, int reg, unsigned int rate);
```

## Arguments

*ac97* the ac97 instance

*reg* the register to change

*rate* the sample rate to set

## Description

Changes the rate of the given input/output on the codec. If the codec doesn't support VAR, the rate must be 48000 (except for SPDIF).

The valid registers are `AC97_PMC_MIC_ADC_RATE`, `AC97_PCM_FRONT_DAC_RATE`, `AC97_PCM_LR_ADC_RATE`, `AC97_PCM_SURR_DAC_RATE` and `AC97_PCM_LFE_DAC_RATE` are accepted if the codec supports them. `AC97_SPDIF` is accepted as a pseudo register to modify the SPDIF status bits.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_ac97_pcm_assign` — assign AC97 slots to given PCM streams

## Synopsis

```
int snd_ac97_pcm_assign (struct snd_ac97_bus * bus, unsigned short
    pcms_count, const struct ac97_pcm * pcms);
```

## Arguments

<i>bus</i>	the ac97 bus instance
<i>pcms_count</i>	count of PCMs to be assigned
<i>pcms</i>	PCMs to be assigned

## Description

It assigns available AC97 slots for given PCMs. If none or only some slots are available, `pcm->xxx.slots` and `pcm->xxx.rslots[]` members are reduced and might be zero.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_ac97_pcm_open` — opens the given AC97 pcm

## Synopsis

```
int snd_ac97_pcm_open (struct ac97_pcm * pcm, unsigned int rate, enum  
ac97_pcm_cfg cfg, unsigned short slots);
```

## Arguments

*pcm*      the ac97 pcm instance

*rate*     rate in Hz, if codec does not support VRA, this value must be 48000Hz

*cfg*      output stream characteristics

*slots*    a subset of allocated slots (`snd_ac97_pcm_assign`) for this pcm

## Description

It locks the specified slots and sets the given rate to AC97 registers.

## Return

Zero if successful, or a negative error code on failure.



## Name

`snd_ac97_pcm_close` — closes the given AC97 pcm

## Synopsis

```
int snd_ac97_pcm_close (struct ac97_pcm * pcm);
```

## Arguments

*pcm* the ac97 pcm instance

## Description

It frees the locked AC97 slots.

## Return

Zero.

## Name

`snd_ac97_pcm_double_rate_rules` — set double rate constraints

## Synopsis

```
int snd_ac97_pcm_double_rate_rules (struct snd_pcm_runtime * runtime);
```

## Arguments

*runtime* the runtime of the ac97 front playback pcm

## Description

Installs the hardware constraint rules to prevent using double rates and more than two channels at the same time.

## Return

Zero if successful, or a negative error code on failure.

# Virtual Master Control API

## Name

`snd_ctl_make_virtual_master` — Create a virtual master control

## Synopsis

```
struct snd_kcontrol * snd_ctl_make_virtual_master (char * name, const
unsigned int * tlv);
```

## Arguments

*name*    name string of the control element to create

*tlv*     optional TLV int array for dB information

## Description

Creates a virtual master control with the given name string.

After creating a vmaster element, you can add the slave controls via `snd_ctl_add_slave` or `snd_ctl_add_slave_uncached`.

The optional argument *tlv* can be used to specify the TLV information for dB scale of the master control. It should be a single element with `#SNDRV_CTL_TLVT_DB_SCALE`, `#SNDRV_CTL_TLV_DB_MINMAX` or `#SNDRV_CTL_TLVT_DB_MINMAX_MUTE` type, and should be the max 0dB.

## Return

The created control element, or `NULL` for errors (`ENOMEM`).

## Name

`snd_ctl_add_vmaster_hook` — Add a hook to a vmaster control

## Synopsis

```
int snd_ctl_add_vmaster_hook (struct snd_kcontrol * kcontrol, void
(*hook) (void *private_data, int), void * private_data);
```

## Arguments

*kcontrol*            vmaster kctl element

*hook*                the hook function

*private\_data*    the *private\_data* pointer to be saved

## Description

Adds the given hook to the vmaster control element so that it's called at each time when the value is changed.

## Return

Zero.

## Name

`snd_ctl_sync_vmaster` — Sync the vmaster slaves and hook

## Synopsis

```
void snd_ctl_sync_vmaster (struct snd_kcontrol * kcontrol, bool  
hook_only);
```

## Arguments

*kcontrol*    vmaster kctl element

*hook\_only*   sync only the hook

## Description

Forcibly call the put callback of each slave and call the hook function to synchronize with the current value of the given vmaster element. NOP when NULL is passed to *kcontrol*.

## Name

`snd_ctl_add_slave` — Add a virtual slave control

## Synopsis

```
int snd_ctl_add_slave (struct snd_kcontrol * master, struct snd_kcontrol  
* slave);
```

## Arguments

*master*    vmaster element

*slave*     slave element to add

## Description

Add a virtual slave control to the given master element created via `snd_ctl_create_virtual_master` beforehand.

All slaves must be the same type (returning the same information via info callback). The function doesn't check it, so it's your responsibility.

Also, some additional limitations: at most two channels, logarithmic volume control (dB level) thus no linear volume, master can only attenuate the volume without gain

## Return

Zero if successful or a negative error code.

## Name

`snd_ctl_add_slave_uncached` — Add a virtual slave control

## Synopsis

```
int snd_ctl_add_slave_uncached (struct snd_kcontrol * master, struct
snd_kcontrol * slave);
```

## Arguments

*master*    vmaster element

*slave*     slave element to add

## Description

Add a virtual slave control to the given master. Unlike `snd_ctl_add_slave`, the element added via this function is supposed to have volatile values, and get callback is called at each time queried from the master.

When the control peeks the hardware values directly and the value can be changed by other means than the put callback of the element, this function should be used to keep the value always up-to-date.

## Return

Zero if successful or a negative error code.

---

# **Chapter 4. MIDI API**

## **Raw MIDI API**



## Name

`snd_rawmidi_receive` — receive the input data from the device

## Synopsis

```
int snd_rawmidi_receive (struct snd_rawmidi_substream * substream, const
unsigned char * buffer, int count);
```

## Arguments

*substream*    the rawmidi substream

*buffer*       the buffer pointer

*count*        the data size to read

## Description

Reads the data from the internal buffer.

## Return

The size of read data, or a negative error code on failure.

## Name

`snd_rawmidi_transmit_empty` — check whether the output buffer is empty

## Synopsis

```
int    snd_rawmidi_transmit_empty    (struct    snd_rawmidi_substream    *  
    substream);
```

## Arguments

*substream* the rawmidi substream

## Return

1 if the internal output buffer is empty, 0 if not.

## Name

`snd_rawmidi_transmit_peek` — copy data from the internal buffer

## Synopsis

```
int  snd_rawmidi_transmit_peek (struct snd_rawmidi_substream *  
    substream, unsigned char * buffer, int count);
```

## Arguments

*substream*    the rawmidi substream

*buffer*       the buffer pointer

*count*        data size to transfer

## Description

Copies data from the internal output buffer to the given buffer.

Call this in the interrupt handler when the midi output is ready, and call `snd_rawmidi_transmit_ack` after the transmission is finished.

## Return

The size of copied data, or a negative error code on failure.

## Name

`snd_rawmidi_transmit_ack` — acknowledge the transmission

## Synopsis

```
int snd_rawmidi_transmit_ack (struct snd_rawmidi_substream * substream,  
int count);
```

## Arguments

*substream*    the rawmidi substream

*count*        the transferred count

## Description

Advances the hardware pointer for the internal output buffer with the given size and updates the condition. Call after the transmission is finished.

## Return

The advanced size if successful, or a negative error code on failure.

## Name

`snd_rawmidi_transmit` — copy from the buffer to the device

## Synopsis

```
int snd_rawmidi_transmit (struct snd_rawmidi_substream * substream,  
unsigned char * buffer, int count);
```

## Arguments

*substream*    the rawmidi substream

*buffer*       the buffer pointer

*count*        the data size to transfer

## Description

Copies data from the buffer to the device and advances the pointer.

## Return

The copied size if successful, or a negative error code on failure.

## Name

`snd_rawmidi_new` — create a rawmidi instance

## Synopsis

```
int snd_rawmidi_new (struct snd_card * card, char * id, int device, int
output_count, int input_count, struct snd_rawmidi ** rrawmidi);
```

## Arguments

<i>card</i>	the card instance
<i>id</i>	the id string
<i>device</i>	the device index
<i>output_count</i>	the number of output streams
<i>input_count</i>	the number of input streams
<i>rrawmidi</i>	the pointer to store the new rawmidi instance

## Description

Creates a new rawmidi instance. Use `snd_rawmidi_set_ops` to set the operators to the new instance.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_rawmidi_set_ops` — set the rawmidi operators

## Synopsis

```
void snd_rawmidi_set_ops (struct snd_rawmidi * rmidi, int stream, struct  
snd_rawmidi_ops * ops);
```

## Arguments

*rmidi*     the rawmidi instance

*stream*   the stream direction, `SNDRV_RAWMIDI_STREAM_XXX`

*ops*       the operator table

## Description

Sets the rawmidi operators for the given stream direction.

# MPU401-UART API

## Name

`snd_mpu401_uart_interrupt` — generic MPU401-UART interrupt handler

## Synopsis

```
irqreturn_t snd_mpu401_uart_interrupt (int irq, void * dev_id);
```

## Arguments

*irq*        the irq number  
*dev\_id*    mpu401 instance

## Description

Processes the interrupt for MPU401-UART i/o.

## Return

IRQ\_HANDLED if the interrupt was handled. IRQ\_NONE otherwise.



## Name

`snd_mpu401_uart_interrupt_tx` — generic MPU401-UART transmit irq handler

## Synopsis

```
irqreturn_t snd_mpu401_uart_interrupt_tx (int irq, void * dev_id);
```

## Arguments

*irq*        the irq number  
*dev\_id*    mpu401 instance

## Description

Processes the interrupt for MPU401-UART output.

## Return

`IRQ_HANDLED` if the interrupt was handled. `IRQ_NONE` otherwise.

## Name

`snd_mpu401_uart_new` — create an MPU401-UART instance

## Synopsis

```
int snd_mpu401_uart_new (struct snd_card * card, int device, unsigned
short hardware, unsigned long port, unsigned int info_flags, int irq,
struct snd_rawmidi ** rrawmidi);
```

## Arguments

<i>card</i>	the card instance
<i>device</i>	the device index, zero-based
<i>hardware</i>	the hardware type, MPU401_HW_XXXX
<i>port</i>	the base address of MPU401 port
<i>info_flags</i>	bitflags MPU401_INFO_XXX
<i>irq</i>	the ISA irq number, -1 if not to be allocated
<i>rrawmidi</i>	the pointer to store the new rawmidi instance

## Description

Creates a new MPU-401 instance.

Note that the rawmidi instance is returned on the *rrawmidi* argument, not the mpu401 instance itself. To access to the mpu401 instance, cast from *rrawmidi*->*private\_data* (with struct `snd_mpu401` magic-cast).

## Return

Zero if successful, or a negative error code.

---

# **Chapter 5. Proc Info API**

## **Proc Info Interface**

## Name

`snd_iprintf` — printf on the procfs buffer

## Synopsis

```
int snd_iprintf (struct snd_info_buffer * buffer, const char * fmt, ...);
```

## Arguments

*buffer*    the procfs buffer

*fmt*        the printf format

*...*        variable arguments

## Description

Outputs the string on the procfs buffer just like `printf`.

## Return

The size of output string, or a negative error code.

## Name

`snd_info_get_line` — read one line from the procfs buffer

## Synopsis

```
int snd_info_get_line (struct snd_info_buffer * buffer, char * line,  
int len);
```

## Arguments

*buffer*    the procfs buffer

*line*      the buffer to store

*len*        the max. buffer size

## Description

Reads one line from the buffer and stores the string.

## Return

Zero if successful, or 1 if error or EOF.

## Name

`snd_info_get_str` — parse a string token

## Synopsis

```
const char * snd_info_get_str (char * dest, const char * src, int len);
```

## Arguments

*dest*    the buffer to store the string token

*src*     the original string

*len*     the max. length of token - 1

## Description

Parses the original string and copy a token to the given string buffer.

## Return

The updated pointer of the original string so that it can be used for the next call.

## Name

`snd_info_create_module_entry` — create an info entry for the given module

## Synopsis

```
struct snd_info_entry * snd_info_create_module_entry (struct module *  
module, const char * name, struct snd_info_entry * parent);
```

## Arguments

*module*    the module pointer

*name*      the file name

*parent*    the parent directory

## Description

Creates a new info entry and assigns it to the given module.

## Return

The pointer of the new instance, or NULL on failure.

## Name

`snd_info_create_card_entry` — create an info entry for the given card

## Synopsis

```
struct snd_info_entry * snd_info_create_card_entry (struct snd_card *  
card, const char * name, struct snd_info_entry * parent);
```

## Arguments

*card*      the card instance

*name*      the file name

*parent*    the parent directory

## Description

Creates a new info entry and assigns it to the given card.

## Return

The pointer of the new instance, or NULL on failure.



## Name

`snd_card_proc_new` — create an info entry for the given card

## Synopsis

```
int snd_card_proc_new (struct snd_card * card, const char * name, struct
snd_info_entry ** entryp);
```

## Arguments

*card*      the card instance

*name*      the file name

*entryp*    the pointer to store the new info entry

## Description

Creates a new info entry and assigns it to the given card. Unlike `snd_info_create_card_entry`, this function registers the info entry as an ALSA device component, so that it can be unregistered/released without explicit call. Also, you don't have to register this entry via `snd_info_register`, since this will be registered by `snd_card_register` automatically.

The parent is assumed as `card->proc_root`.

For releasing this entry, use `snd_device_free` instead of `snd_info_free_entry`.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_info_free_entry` — release the info entry

## Synopsis

```
void snd_info_free_entry (struct snd_info_entry * entry);
```

## Arguments

*entry* the info entry

## Description

Releases the info entry. Don't call this after registered.

## Name

`snd_info_register` — register the info entry

## Synopsis

```
int snd_info_register (struct snd_info_entry * entry);
```

## Arguments

*entry* the info entry

## Description

Registers the proc info entry.

## Return

Zero if successful, or a negative error code on failure.

---

# **Chapter 6. Miscellaneous Functions**

## **Hardware-Dependent Devices API**

## Name

`snd_hwdep_new` — create a new hwdep instance

## Synopsis

```
int snd_hwdep_new (struct snd_card * card, char * id, int device, struct
snd_hwdep ** rhwdep);
```

## Arguments

*card*      the card instance

*id*        the id string

*device*    the device index (zero-based)

*rhwdep*    the pointer to store the new hwdep instance

## Description

Creates a new hwdep instance with the given index on the card. The callbacks (`hwdep->ops`) must be set on the returned instance after this call manually by the caller.

## Return

Zero if successful, or a negative error code on failure.

# Jack Abstraction Layer API

## Name

`snd_jack_new` — Create a new jack

## Synopsis

```
int snd_jack_new (struct snd_card * card, const char * id, int type,  
struct snd_jack ** jjack);
```

## Arguments

*card*     the card instance

*id*       an identifying string for this jack

*type*     a bitmask of enum `snd_jack_type` values that can be detected by this jack

*jjack*    Used to provide the allocated jack object to the caller.

## Description

Creates a new jack object.

## Return

Zero if successful, or a negative error code on failure. On success *jjack* will be initialised.

## Name

`snd_jack_set_parent` — Set the parent device for a jack

## Synopsis

```
void snd_jack_set_parent (struct snd_jack * jack, struct device *  
parent);
```

## Arguments

*jack*      The jack to configure

*parent*    The device to set as parent for the jack.

## Description

Set the parent for the jack devices in the device tree. This function is only valid prior to registration of the jack. If no parent is configured then the parent device will be the sound card.

## Name

`snd_jack_set_key` — Set a key mapping on a jack

## Synopsis

```
int snd_jack_set_key (struct snd_jack * jack, enum snd_jack_types type,
int keytype);
```

## Arguments

<i>jack</i>	The jack to configure
<i>type</i>	Jack report type for this key
<i>keytype</i>	Input layer key type to be reported

## Description

Map a `SND_JACK_BTN_` button type to an input layer key, allowing reporting of keys on accessories via the jack abstraction. If no mapping is provided but keys are enabled in the jack type then `BTN_n` numeric buttons will be reported.

If jacks are not reporting via the input API this call will have no effect.

Note that this is intended to be use by simple devices with small numbers of keys that can be reported. It is also possible to access the input device directly - devices with complex input capabilities on accessories should consider doing this rather than using this abstraction.

This function may only be called prior to registration of the jack.

## Return

Zero if successful, or a negative error code on failure.



## Name

`snd_jack_report` — Report the current status of a jack

## Synopsis

```
void snd_jack_report (struct snd_jack * jack, int status);
```

## Arguments

*jack*      The jack to report status for

*status*    The current status of the jack

# ISA DMA Helpers

## Name

`snd_dma_program` — program an ISA DMA transfer

## Synopsis

```
void snd_dma_program (unsigned long dma, unsigned long addr, unsigned  
int size, unsigned short mode);
```

## Arguments

*dma* the dma number

*addr* the physical address of the buffer

*size* the DMA transfer size

*mode* the DMA transfer mode, `DMA_MODE_XXX`

## Description

Programs an ISA DMA transfer for the given buffer.

## Name

`snd_dma_disable` — stop the ISA DMA transfer

## Synopsis

```
void snd_dma_disable (unsigned long dma);
```

## Arguments

*dma* the dma number

## Description

Stops the ISA DMA transfer.

## Name

`snd_dma_pointer` — return the current pointer to DMA transfer buffer in bytes

## Synopsis

```
unsigned int snd_dma_pointer (unsigned long dma, unsigned int size);
```

## Arguments

*dma*     the dma number

*size*    the dma transfer size

## Return

The current pointer in DMA transfer buffer in bytes.

## Other Helper Macros

## Name

`snd_register_device` — Register the ALSA device file for the card

## Synopsis

```
int snd_register_device (int type, struct snd_card * card, int dev,  
const struct file_operations * f_ops, void * private_data, const char  
* name);
```

## Arguments

<i>type</i>	the device type, SNDRV_DEVICE_TYPE_XXX
<i>card</i>	the card instance
<i>dev</i>	the device index
<i>f_ops</i>	the file operations
<i>private_data</i>	user pointer for <i>f_ops</i> ->open
<i>name</i>	the device file name

## Description

Registers an ALSA device file for the given card. The operators have to be set in *reg* parameter.

This function uses the card's device pointer to link to the correct struct device.

## Return

Zero if successful, or a negative error code on failure.

## Name

`snd_printk` — printk wrapper

## Synopsis

```
snd_printk ( fmt, args... );
```

## Arguments

*fmt*            format string

*args...*      variable arguments

## Description

Works like `printk` but prints the file and the line of the caller when configured with `CONFIG_SND_VERBOSE_PRINTK`.

## Name

snd\_printd — debug printk

## Synopsis

```
snd_printd ( fmt, args... );
```

## Arguments

*fmt*            format string

*args...*      variable arguments

## Description

Works like `snd_printk` for debugging purposes. Ignored when `CONFIG_SND_DEBUG` is not set.

## Name

`snd_DEBUG` — give a BUG warning message and stack trace

## Synopsis

```
snd_DEBUG (void);
```

## Arguments

None

## Description

Calls `WARN` if `CONFIG_SND_DEBUG` is set. Ignored when `CONFIG_SND_DEBUG` is not set.



## Name

`snd_printd_ratelimit` —

## Synopsis

```
snd_printd_ratelimit (void);
```

## Arguments

None

## Name

`snd_DEBUG_ON` — debugging check macro

## Synopsis

```
snd_DEBUG_ON ( cond );
```

## Arguments

*cond*   condition to evaluate

## Description

Has the same behavior as `WARN_ON` when `CONFIG_SND_DEBUG` is set, otherwise just evaluates the conditional and returns the value.

## Name

snd\_printdd — debug printk

## Synopsis

```
snd_printdd ( format, args... );
```

## Arguments

*format*     format string

*args...*   variable arguments

## Description

Works like `snd_printk` for debugging purposes. Ignored when `CONFIG_SND_DEBUG_VERBOSE` is not set.