# 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 ..... 13
    - Module requests and Device File Entries ..... 17
    - Memory Management Helpers ..... 21
2. PCM API ..... 29
    - PCM Core ..... 29
    - PCM Format Helpers ..... 111
    - PCM Memory Management ..... 125
    - PCM DMA Engine API ..... 133
3. Control/Mixer API ..... 147
    - General Control Interface ..... 147
    - AC97 Codec API ..... 165
    - Virtual Master Control API ..... 182
4. MIDI API ..... 188
    - Raw MIDI API ..... 188
    - MPU401-UART API ..... 197
5. Proc Info API ..... 201
    - Proc Info Interface ..... 201
6. Compress Offload ..... 208
    - Compress Offload API ..... 208
7. ASoC ..... 227
    - ASoC Core API ..... 227
    - ASoC DAPM API ..... 307
    - ASoC DMA Engine API ..... 335
8. Miscellaneous Functions ..... 339
    - Hardware-Dependent Devices API ..... 339
    - Jack Abstraction Layer API ..... 340
    - ISA DMA Helpers ..... 357
    - Other Helper Macros ..... 360

# Chapter 1. Management of Cards and Devices

## Card Management

# Name

snd_device_initialize — Initialize struct device for sound devices

## Synopsis

```
void snd_device_initialize (struct device * dev, struct snd_card * card);
```

## Arguments

*dev* device to initialize

*card* card to assign, optional

# 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

| | |
|---|---|
| *parent* | the parent device object |
| *idx* | card index (address) [0 ... (SNDRV_CARDS-1)] |
| *xid* | card identification (ASCII string) |
| *module* | top level module for locking |
| *extra_size* | allocate this extra size after the main soundcard structure |
| *card_ret* | 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_free_when_closed — Disconnect the card, free it later eventually

## Synopsis

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

## Arguments

*card* soundcard structure

## Description

Unlike `snd_card_free`, this function doesn't try to release the card resource immediately, but tries to disconnect at first. When the card is still in use, the function returns before freeing the resources. The card resources will be freed when the refcount gets to zero.

# Name

snd_card_free — frees given soundcard structure

## Synopsis

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

## Arguments

*card* soundcard structure

## Description

This function releases the soundcard structure and the all assigned devices automatically. That is, you don't have to release the devices by yourself.

This function waits until the all resources are properly released.

## Return

Zero. Frees all associated devices and frees the control interface associated to given soundcard.

# 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_add_dev_attr — Append a new sysfs attribute group to card

## Synopsis

```
int snd_card_add_dev_attr (struct snd_card * card, const struct
attribute_group * group);
```

## Arguments

*card* card instance

*group* attribute group to append

# 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

| | |
|---|---|
| *card* | the card instance |
| *type* | the device type, SNDRV_DEV_XXX |
| *device_data* | the data pointer of this device |
| *ops* | 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_disconnect — disconnect the device

## Synopsis

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

## Arguments

*card* the card instance

*device_data* the data pointer to disconnect

## Description

Turns the device into the disconnection state, invoking dev_disconnect callback, if the device was already registered.

Usually called from `snd_card_disconnect`.

## Return

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

# 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 — 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, struct device
* device);
```

## Arguments

| | |
|---|---|
| *type* | the device type, SNDRV_DEVICE_TYPE_XXX |
| *card* | the card instance |
| *dev* | the device index |
| *f_ops* | the file operations |
| *private_data* | user pointer for f_ops->open |
| *device* | the device to register |

## 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 (struct device * dev);
```

## Arguments

*dev* the device instance

## 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_format_name — Return a name string for the given PCM format

## Synopsis

```
const char * snd_pcm_format_name (snd_pcm_format_t format);
```

## Arguments

*format* PCM format

# 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

| | |
|---|---|
| *pcm* | the pcm instance |
| *stream* | the stream direction, SNDRV_PCM_STREAM_XXX |
| *substream_count* | 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

| | |
|---|---|
| *card* | the card instance |
| *id* | the id string |
| *device* | the device index (zero based) |
| *playback_count* | the number of substreams for playback |
| *capture_count* | the number of substreams for capture |
| *rpcm* | 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

| | |
|---|---|
| *card* | the card instance |
| *id* | the id string |
| *device* | the device index (zero based - shared with normal PCMs) |
| *playback_count* | the number of substreams for playback |
| *capture_count* | the number of substreams for capture |
| *rpcm* | 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 PCMs 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_notify — Add/remove the notify list

## Synopsis

```
int snd_pcm_notify (struct snd_pcm_notify * notify, int nfree);
```

## Arguments

*notify* PCM notify list

*nfree* 0 = register, 1 = unregister

## Description

This adds the given notifier to the global list so that the callback is called for each registered PCM devices. This exists only for PCM OSS emulation, so far.

# 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

| | |
|---|---|
| *pcm* | the pcm instance |
| *direction* | stream direction, SNDRV_PCM_STREAM_XXX |
| *ops* | 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, const struct snd_ratnum * rats, unsigned int * nump, unsigned int * denp);
```

## Arguments

| | |
|---|---|
| *i* | interval to refine |
| *rats_count* | number of ratnum_t |
| *rats* | ratnum_t array |
| *nump* | pointer to store the resultant numerator |
| *denp* | 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* the interval value to refine

*count* the number of elements in the list

*list* the value list

*mask* 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_interval_ranges — refine the interval value from the list of ranges

## Synopsis

```
int snd_interval_ranges (struct snd_interval * i, unsigned int count,
const struct snd_interval * ranges, unsigned int mask);
```

## Arguments

| | |
|---|---|
| *i* | the interval value to refine |
| *count* | the number of elements in the list of ranges |
| *ranges* | the ranges list |
| *mask* | the bit-mask to evaluate |

## Description

Refines the interval value from the list of ranges. 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

| | |
|---|---|
| *runtime* | the pcm runtime instance |
| *cond* | condition bits |
| *var* | the variable to evaluate |
| *func* | the evaluation function |
| *private* | the private data pointer passed to function |
| *dep* | 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

*runtime* PCM runtime instance

*var* hw_params variable to apply the range

*min* the minimal value

*max* 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

*runtime* PCM runtime instance

*cond* condition bits

*var* hw_params variable to apply the list constraint

*l* 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_ranges — apply list of range constraints to a parameter

## Synopsis

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

## Arguments

*runtime* PCM runtime instance

*cond* condition bits

*var* hw_params variable to apply the list of range constraints

*r* ranges

## Description

Apply the list of range 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, const struct snd_pcm_hw_constraint_ratnums * r);
```

## Arguments

*runtime* PCM runtime instance

*cond* condition bits

*var* hw_params variable to apply the ratnums constraint

*r* struct snd_ratnums constriants

## 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, const struct snd_pcm_hw_constraint_ratdens * r);
```

## Arguments

*runtime* PCM runtime instance

*cond* condition bits

*var* hw_params variable to apply the ratdens constraint

*r* struct snd_ratdens constriants

## 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

## Description

This constraint will set the number of most significant bits (msbits) if a sample format with the specified width has been select. If width is set to 0 the msbits will be set for any sample format with a width larger than the specified msbits.

## 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

*pcm* PCM instance

*params* the hw_params instance

*var* parameter to retrieve

*dir* 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

| | |
|---|---|
| *pcm* | PCM instance |
| *params* | the hw_params instance |
| *var* | parameter to retrieve |
| *dir* | 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

| | |
|---|---|
| *pcm* | the assigned PCM instance |
| *stream* | stream direction |
| *chmap* | channel map elements (for query) |
| *max_channels* | the max number of channels for the stream |
| *private_value* | the value passed to each kcontrol's private_value field |
| *info_ret* | store struct snd_pcm_chmap 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_stream_lock — Lock the PCM stream

## Synopsis

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

## Arguments

*substream* PCM substream

## Description

This locks the PCM stream's spinlock or mutex depending on the nonatomic flag of the given substream. This also takes the global link rw lock (or rw sem), too, for avoiding the race with linked streams.

# Name

snd_pcm_stream_unlock — Unlock the PCM stream

## Synopsis

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

## Arguments

*substream* PCM substream

## Description

This unlocks the PCM stream that has been locked via `snd_pcm_stream_lock`.

# Name

snd_pcm_stream_lock_irq — Lock the PCM stream

## Synopsis

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

## Arguments

*substream* PCM substream

## Description

This locks the PCM stream like `snd_pcm_stream_lock` and disables the local IRQ (only when nonatomic is false). In nonatomic case, this is identical as `snd_pcm_stream_lock`.

# Name

snd_pcm_stream_unlock_irq — Unlock the PCM stream

## Synopsis

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

## Arguments

*substream* PCM substream

## Description

This is a counter-part of `snd_pcm_stream_lock_irq`.

# Name

snd_pcm_stream_unlock_irqrestore — Unlock the PCM stream

## Synopsis

```
void snd_pcm_stream_unlock_irqrestore (struct snd_pcm_substream * substream, unsigned long flags);
```

## Arguments

*substream* PCM substream

*flags* irq flags

## Description

This is a counter-part of `snd_pcm_stream_lock_irqsave`.

# 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_stop_xrun — stop the running streams as XRUN

## Synopsis

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

## Arguments

*substream* the PCM substream instance

## Description

This stops the given running substream (and all linked substreams) as XRUN. Unlike `snd_pcm_stop`, this function takes the substream lock by itself.

## 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.

# Name

snd_pcm_lib_default_mmap — Default PCM data mmap function

## Synopsis

```
int snd_pcm_lib_default_mmap (struct snd_pcm_substream * substream,
struct vm_area_struct * area);
```

## Arguments

*substream* PCM substream

*area* VMA

## Description

This is the default mmap handler for PCM data. When mmap pcm_ops is NULL, this function is invoked implicitly.

# Name

snd_pcm_lib_mmap_iomem — Default PCM data mmap function for I/O mem

## Synopsis

```
int snd_pcm_lib_mmap_iomem (struct snd_pcm_substream * substream, struct
vm_area_struct * area);
```

## Arguments

*substream* PCM substream

*area* VMA

## Description

When your hardware uses the iomapped pages as the hardware buffer and wants to mmap it, pass this function as mmap pcm_ops. Note that this is supposed to work only on limited architectures.

# Name

snd_pcm_stream_linked — Check whether the substream is linked with others

## Synopsis

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

## Arguments

*substream* substream to check

## Description

Returns true if the given substream is being linked with others.

# Name

snd_pcm_stream_lock_irqsave — Lock the PCM stream

## Synopsis

```
snd_pcm_stream_lock_irqsave ( substream, flags);
```

## Arguments

*substream* PCM substream

*flags* irq flags

## Description

This locks the PCM stream like `snd_pcm_stream_lock` but with the local IRQ (only when nonatomic is false). In nonatomic case, this is identical as `snd_pcm_stream_lock`.

# Name

snd_pcm_group_for_each_entry — iterate over the linked substreams

## Synopsis

```
snd_pcm_group_for_each_entry ( s, substream);
```

## Arguments

| | |
|---|---|
| *s* | the iterator |
| *substream* | the substream |

## Description

Iterate over the all linked substreams to the given *substream*. When *substream* isn't linked with any others, this gives returns *substream* itself once.

# Name

snd_pcm_running — Check whether the substream is in a running state

## Synopsis

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

## Arguments

*substream* substream to check

## Description

Returns true if the given substream is in the state RUNNING, or in the state DRAINING for playback.

# Name

bytes_to_samples — Unit conversion of the size from bytes to samples

## Synopsis

```
ssize_t bytes_to_samples (struct snd_pcm_runtime * runtime, ssize_t size);
```

## Arguments

*runtime* PCM runtime instance

*size* size in bytes

# Name

bytes_to_frames — Unit conversion of the size from bytes to frames

## Synopsis

```
snd_pcm_sframes_t bytes_to_frames (struct snd_pcm_runtime * runtime,
ssize_t size);
```

## Arguments

*runtime* PCM runtime instance

*size* size in bytes

# Name

samples_to_bytes — Unit conversion of the size from samples to bytes

## Synopsis

```
ssize_t samples_to_bytes (struct snd_pcm_runtime * runtime, ssize_t size);
```

## Arguments

*runtime* PCM runtime instance

*size* size in samples

# Name

frames_to_bytes — Unit conversion of the size from frames to bytes

## Synopsis

```
ssize_t frames_to_bytes (struct snd_pcm_runtime * runtime,
snd_pcm_sframes_t size);
```

## Arguments

*runtime* PCM runtime instance

*size* size in frames

# Name

frame_aligned — Check whether the byte size is aligned to frames

## Synopsis

```
int frame_aligned (struct snd_pcm_runtime * runtime, ssize_t bytes);
```

## Arguments

*runtime* PCM runtime instance

*bytes* size in bytes

# Name

snd_pcm_lib_buffer_bytes — Get the buffer size of the current PCM in bytes

## Synopsis

```
size_t snd_pcm_lib_buffer_bytes (struct snd_pcm_substream * substream);
```

## Arguments

*substream* PCM substream

# Name

snd_pcm_lib_period_bytes — Get the period size of the current PCM in bytes

## Synopsis

```
size_t snd_pcm_lib_period_bytes (struct snd_pcm_substream * substream);
```

## Arguments

*substream* PCM substream

# Name

snd_pcm_playback_avail — Get the available (writable) space for playback

## Synopsis

```
snd_pcm_uframes_t snd_pcm_playback_avail (struct snd_pcm_runtime * runtime);
```

## Arguments

*runtime* PCM runtime instance

## Description

Result is between 0 ... (boundary - 1)

# Name

snd_pcm_capture_avail — Get the available (readable) space for capture

## Synopsis

```
snd_pcm_uframes_t snd_pcm_capture_avail (struct snd_pcm_runtime *runtime);
```

## Arguments

*runtime* PCM runtime instance

## Description

Result is between 0 ... (boundary - 1)

# Name

snd_pcm_playback_hw_avail — Get the queued space for playback

## Synopsis

```
snd_pcm_sframes_t snd_pcm_playback_hw_avail (struct snd_pcm_runtime *
runtime);
```

## Arguments

*runtime* PCM runtime instance

# Name

snd_pcm_capture_hw_avail — Get the free space for capture

## Synopsis

```
snd_pcm_sframes_t snd_pcm_capture_hw_avail (struct snd_pcm_runtime *
runtime);
```

## Arguments

*runtime* PCM runtime instance

# Name

snd_pcm_playback_ready — check whether the playback buffer is available

## Synopsis

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

## Arguments

*substream* the pcm substream instance

## Description

Checks whether enough free space is available on the playback buffer.

## Return

Non-zero if available, or zero if not.

# Name

snd_pcm_capture_ready — check whether the capture buffer is available

## Synopsis

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

## Arguments

*substream* the pcm substream instance

## Description

Checks whether enough capture data is available on the capture buffer.

## Return

Non-zero if available, or zero if not.

# Name

snd_pcm_playback_data — check whether any data exists on the playback buffer

## Synopsis

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

## Arguments

*substream* the pcm substream instance

## Description

Checks whether any data exists on the playback buffer.

## Return

Non-zero if any data exists, or zero if not. If stop_threshold is bigger or equal to boundary, then this function returns always non-zero.

# Name

snd_pcm_playback_empty — check whether the playback buffer is empty

## Synopsis

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

## Arguments

*substream* the pcm substream instance

## Description

Checks whether the playback buffer is empty.

## Return

Non-zero if empty, or zero if not.

# Name

snd_pcm_capture_empty — check whether the capture buffer is empty

## Synopsis

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

## Arguments

*substream* the pcm substream instance

## Description

Checks whether the capture buffer is empty.

## Return

Non-zero if empty, or zero if not.

# Name

snd_pcm_trigger_done — Mark the master substream

## Synopsis

```
void snd_pcm_trigger_done (struct snd_pcm_substream * substream, struct
snd_pcm_substream * master);
```

## Arguments

*substream* the pcm substream instance

*master* the linked master substream

## Description

When multiple substreams of the same card are linked and the hardware supports the single-shot operation, the driver calls this in the loop in `snd_pcm_group_for_each_entry` for marking the substream as “done”. Then most of trigger operations are performed only to the given master substream.

The trigger_master mark is cleared at timestamp updates at the end of trigger operations.

# Name

params_channels — Get the number of channels from the hw params

## Synopsis

```
unsigned int params_channels (const struct snd_pcm_hw_params * p);
```

## Arguments

*p* hw params

# Name

params_rate — Get the sample rate from the hw params

## Synopsis

```
unsigned int params_rate (const struct snd_pcm_hw_params * p);
```

## Arguments

*p* hw params

# Name

params_period_size — Get the period size (in frames) from the hw params

## Synopsis

```
unsigned int params_period_size (const struct snd_pcm_hw_params * p);
```

## Arguments

*p* hw params

# Name

params_periods — Get the number of periods from the hw params

## Synopsis

```
unsigned int params_periods (const struct snd_pcm_hw_params * p);
```

## Arguments

*p* hw params

# Name

params_buffer_size — Get the buffer size (in frames) from the hw params

## Synopsis

```
unsigned int params_buffer_size (const struct snd_pcm_hw_params * p);
```

## Arguments

*p* hw params

# Name

params_buffer_bytes — Get the buffer size (in bytes) from the hw params

## Synopsis

```
unsigned int params_buffer_bytes (const struct snd_pcm_hw_params * p);
```

## Arguments

*p* hw params

# Name

snd_pcm_hw_constraint_single — Constrain parameter to a single value

## Synopsis

```
int snd_pcm_hw_constraint_single (struct snd_pcm_runtime * runtime,
snd_pcm_hw_param_t var, unsigned int val);
```

## Arguments

*runtime* PCM runtime instance

*var* The hw_params variable to constrain

*val* The value to constrain to

## Return

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

# Name

snd_pcm_format_cpu_endian — Check the PCM format is CPU-endian

## Synopsis

```
int snd_pcm_format_cpu_endian (snd_pcm_format_t format);
```

## Arguments

*format* the format to check

## Return

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

# Name

snd_pcm_set_runtime_buffer — Set the PCM runtime buffer

## Synopsis

```
void snd_pcm_set_runtime_buffer (struct snd_pcm_substream * substream,
struct snd_dma_buffer * bufp);
```

## Arguments

*substream* PCM substream to set

*bufp* the buffer information, NULL to clear

## Description

Copy the buffer information to runtime->dma_buffer when *bufp* is non-NULL. Otherwise it clears the current buffer information.

# Name

snd_pcm_gettime — Fill the timespec depending on the timestamp mode

## Synopsis

```
void snd_pcm_gettime (struct snd_pcm_runtime * runtime, struct timespec
* tv);
```

## Arguments

*runtime* PCM runtime instance

*tv* timespec to fill

# Name

snd_pcm_lib_alloc_vmalloc_buffer — allocate virtual DMA buffer

## Synopsis

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

## Arguments

*substream* the substream to allocate the buffer to

*size* the requested buffer size, in bytes

## Description

Allocates the PCM substream buffer using `vmalloc`, i.e., the memory is contiguous in kernel virtual space, but not in physical memory. Use this if the buffer is accessed by kernel code but not by device DMA.

## Return

1 if the buffer was changed, 0 if not changed, or a negative error code.

# Name

snd_pcm_lib_alloc_vmalloc_32_buffer — allocate 32-bit-addressable buffer

## Synopsis

```
int snd_pcm_lib_alloc_vmalloc_32_buffer (struct snd_pcm_substream * substream, size_t size);
```

## Arguments

*substream* the substream to allocate the buffer to

*size* the requested buffer size, in bytes

## Description

This function works like `snd_pcm_lib_alloc_vmalloc_buffer`, but uses `vmalloc_32`, i.e., the pages are allocated from 32-bit-addressable memory.

## Return

1 if the buffer was changed, 0 if not changed, or a negative error code.

# Name

snd_pcm_sgbuf_get_addr — Get the DMA address at the corresponding offset

## Synopsis

```
dma_addr_t snd_pcm_sgbuf_get_addr (struct snd_pcm_substream * substream, unsigned int ofs);
```

## Arguments

*substream* PCM substream

*ofs* byte offset

# Name

snd_pcm_sgbuf_get_ptr — Get the virtual address at the corresponding offset

## Synopsis

```
void * snd_pcm_sgbuf_get_ptr (struct snd_pcm_substream * substream,
unsigned int ofs);
```

## Arguments

*substream* PCM substream

*ofs* byte offset

# Name

snd_pcm_sgbuf_get_chunk_size — Compute the max size that fits within the contig. page from the given size

## Synopsis

```
unsigned int snd_pcm_sgbuf_get_chunk_size (struct snd_pcm_substream *
substream, unsigned int ofs, unsigned int size);
```

## Arguments

*substream* PCM substream

*ofs* byte offset

*size* byte size to examine

# Name

snd_pcm_mmap_data_open — increase the mmap counter

## Synopsis

```
void snd_pcm_mmap_data_open (struct vm_area_struct * area);
```

## Arguments

*area* VMA

## Description

PCM mmap callback should handle this counter properly

# Name

snd_pcm_mmap_data_close — decrease the mmap counter

## Synopsis

```
void snd_pcm_mmap_data_close (struct vm_area_struct * area);
```

## Arguments

*area* VMA

## Description

PCM mmap callback should handle this counter properly

# Name

snd_pcm_limit_isa_dma_size — Get the max size fitting with ISA DMA transfer

## Synopsis

```
void snd_pcm_limit_isa_dma_size (int dma, size_t * max);
```

## Arguments

*dma* DMA number

*max* pointer to store the max size

# Name

snd_pcm_stream_str — Get a string naming the direction of a stream

## Synopsis

```
const char * snd_pcm_stream_str (struct snd_pcm_substream * substream);
```

## Arguments

*substream* the pcm substream instance

## Return

A string naming the direction of the stream.

# Name

snd_pcm_chmap_substream — get the PCM substream assigned to the given chmap info

## Synopsis

```
struct snd_pcm_substream * snd_pcm_chmap_substream (struct
snd_pcm_chmap * info, unsigned int idx);
```

## Arguments

*info* chmap information

*idx* the substream number index

## Name

pcm_format_to_bits — Strong-typed conversion of pcm_format to bitwise

### Synopsis

```
u64 pcm_format_to_bits (snd_pcm_format_t pcm_format);
```

### Arguments

*pcm_format* PCM format

# 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

*substream* the pcm substream 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 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.

# PCM DMA Engine API

# Name

snd_hwparams_to_dma_slave_config — Convert hw_params to dma_slave_config

## Synopsis

```
int snd_hwparams_to_dma_slave_config (const struct snd_pcm_substream
* substream, const struct snd_pcm_hw_params * params, struct
dma_slave_config * slave_config);
```

## Arguments

| | |
|---|---|
| *substream* | PCM substream |
| *params* | hw_params |
| *slave_config* | DMA slave config |

## Description

This function can be used to initialize a dma_slave_config from a substream and hw_params in a dmaengine based PCM driver implementation.

# Name

snd_dmaengine_pcm_set_config_from_dai_data — Initializes a dma slave config using DAI DMA data.

## Synopsis

```
void    snd_dmaengine_pcm_set_config_from_dai_data    (const    struct
snd_pcm_substream * substream, const struct snd_dmaengine_dai_dma_data
* dma_data, struct dma_slave_config * slave_config);
```

## Arguments

| | |
|---|---|
| *substream* | PCM substream |
| *dma_data* | DAI DMA data |
| *slave_config* | DMA slave configuration |

## Description

Initializes the {dst,src}_addr, {dst,src}_maxburst, {dst,src}_addr_width and slave_id fields of the DMA slave config from the same fields of the DAI DMA data struct. The src and dst fields will be initialized depending on the direction of the substream. If the substream is a playback stream the dst fields will be initialized, if it is a capture stream the src fields will be initialized. The {dst,src}_addr_width field will only be initialized if the addr_width field of the DAI DMA data struct is not equal to DMA_SLAVE_BUSWIDTH_UNDEFINED.

# Name

snd_dmaengine_pcm_trigger — dmaengine based PCM trigger implementation

## Synopsis

```
int snd_dmaengine_pcm_trigger (struct snd_pcm_substream * substream,
int cmd);
```

## Arguments

*substream* PCM substream

*cmd* Trigger command

## Description

Returns 0 on success, a negative error code otherwise.

This function can be used as the PCM trigger callback for dmaengine based PCM driver implementations.

# Name

snd_dmaengine_pcm_pointer_no_residue — dmaengine based PCM pointer implementation

## Synopsis

```
snd_pcm_uframes_t     snd_dmaengine_pcm_pointer_no_residue     (struct
snd_pcm_substream * substream);
```

## Arguments

*substream* PCM substream

## Description

This function is deprecated and should not be used by new drivers, as its results may be unreliable.

# Name

snd_dmaengine_pcm_pointer — dmaengine based PCM pointer implementation

## Synopsis

```
snd_pcm_uframes_t snd_dmaengine_pcm_pointer (struct snd_pcm_substream
* substream);
```

## Arguments

*substream* PCM substream

## Description

This function can be used as the PCM pointer callback for dmaengine based PCM driver implementations.

# Name

snd_dmaengine_pcm_request_channel — Request channel for the dmaengine PCM

## Synopsis

```
struct dma_chan * snd_dmaengine_pcm_request_channel (dma_filter_fn
filter_fn, void * filter_data);
```

## Arguments

*filter_fn* Filter function used to request the DMA channel

*filter_data* Data passed to the DMA filter function

## Description

Returns NULL or the requested DMA channel.

This function request a DMA channel for usage with dmaengine PCM.

# Name

snd_dmaengine_pcm_open — Open a dmaengine based PCM substream

## Synopsis

```
int snd_dmaengine_pcm_open (struct snd_pcm_substream * substream, struct
dma_chan * chan);
```

## Arguments

*substream* PCM substream

*chan* DMA channel to use for data transfers

## Description

Returns 0 on success, a negative error code otherwise.

The function should usually be called from the pcm open callback. Note that this function will use private_data field of the substream's runtime. So it is not available to your pcm driver implementation.

# Name

snd_dmaengine_pcm_open_request_chan — Open a dmaengine based PCM substream and request channel

## Synopsis

```
int snd_dmaengine_pcm_open_request_chan (struct snd_pcm_substream * substream, dma_filter_fn filter_fn, void * filter_data);
```

## Arguments

| | |
|---|---|
| *substream* | PCM substream |
| *filter_fn* | Filter function used to request the DMA channel |
| *filter_data* | Data passed to the DMA filter function |

## Description

Returns 0 on success, a negative error code otherwise.

This function will request a DMA channel using the passed filter function and data. The function should usually be called from the pcm open callback. Note that this function will use private_data field of the substream's runtime. So it is not available to your pcm driver implementation.

# Name

snd_dmaengine_pcm_close — Close a dmaengine based PCM substream

## Synopsis

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

## Arguments

*substream* PCM substream

# Name

snd_dmaengine_pcm_close_release_chan — Close a dmaengine based PCM substream and release channel

## Synopsis

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

## Arguments

*substream* PCM substream

## Description

Releases the DMA channel associated with the PCM substream.

# Name

snd_pcm_substream_to_dma_direction — Get dma_transfer_direction for a PCM substream

## Synopsis

```
enum dma_transfer_direction snd_pcm_substream_to_dma_direction (const
struct snd_pcm_substream * substream);
```

## Arguments

*substream* PCM substream

# Name

struct snd_dmaengine_dai_dma_data — DAI DMA configuration data

## Synopsis

```
struct snd_dmaengine_dai_dma_data {
  dma_addr_t addr;
  enum dma_slave_buswidth addr_width;
  u32 maxburst;
  unsigned int slave_id;
  void * filter_data;
  const char * chan_name;
  unsigned int fifo_size;
};
```

## Members

| | |
|---|---|
| addr | Address of the DAI data source or destination register. |
| addr_width | Width of the DAI data source or destination register. |
| maxburst | Maximum number of words(note: words, as in units of the src_addr_width member, not bytes) that can be send to or received from the DAI in one burst. |
| slave_id | Slave requester id for the DMA channel. |
| filter_data | Custom DMA channel filter data, this will usually be used when requesting the DMA channel. |
| chan_name | Custom channel name to use when requesting DMA channel. |
| fifo_size | FIFO size of the DAI controller in bytes |

# Name

struct snd_dmaengine_pcm_config — Configuration data for dmaengine based PCM

# Synopsis

```
struct snd_dmaengine_pcm_config {
  int (* prepare_slave_config) (struct snd_pcm_substream *substream,struct snd_pcm
  struct dma_chan *(* compat_request_channel) (struct snd_soc_pcm_runtime *rtd,str
  dma_filter_fn compat_filter_fn;
  struct device * dma_dev;
  const char * chan_names[SNDRV_PCM_STREAM_LAST + 1];
  const struct snd_pcm_hardware * pcm_hardware;
  unsigned int prealloc_buffer_size;
};
```

# Members

| | |
|---|---|
| prepare_slave_config | Callback used to fill in the DMA slave_config for a PCM substream. Will be called from the PCM drivers hwparams callback. |
| compat_request_channel | Callback to request a DMA channel for platforms which do not use devicetree. |
| compat_filter_fn | Will be used as the filter function when requesting a channel for platforms which do not use devicetree. The filter parameter will be the DAI's DMA data. |
| dma_dev | If set, request DMA channel on this device rather than the DAI device. |
| chan_names[SNDRV_PCM_STREAM_LAST + 1] | If set, these custom DMA channel names will be requested at registration time. |
| pcm_hardware | snd_pcm_hardware struct to be used for the PCM. |
| prealloc_buffer_size | Size of the preallocated audio buffer. |

# Note

If both compat_request_channel and compat_filter_fn are set compat_request_channel will be used to request the channel and compat_filter_fn will be ignored. Otherwise the channel will be requested using dma_request_channel with compat_filter_fn as the filter function.

# Chapter 3. Control/Mixer API

## General Control Interface

# Name

snd_ctl_notify — Send notification to user-space for a control change

## Synopsis

```
void snd_ctl_notify (struct snd_card * card, unsigned int mask, struct
snd_ctl_elem_id * id);
```

## Arguments

*card* the card to send notification

*mask* the event mask, SNDRV_CTL_EVENT_*

*id* the ctl element id to send notification

## Description

This function adds an event record with the given id and mask, appends to the list and wakes up the user-space for notification. This can be called in the atomic context.

# 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

| | |
|---|---|
| *card* | the card instance |
| *kcontrol* | the control instance to replace |
| *add_on_replace* | 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. The given ID data is filled with full information.

## 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_register_ioctl — register the device-specific control-ioctls

## Synopsis

```
int snd_ctl_register_ioctl (snd_kctl_ioctl_func_t fcn);
```

## Arguments

*fcn* ioctl callback function

## Description

called from each device manager like pcm.c, hwdep.c, etc.

# Name

snd_ctl_register_ioctl_compat — register the device-specific 32bit compat control-ioctls

## Synopsis

```
int snd_ctl_register_ioctl_compat (snd_kctl_ioctl_func_t fcn);
```

## Arguments

*fcn* ioctl callback function

# Name

snd_ctl_unregister_ioctl — de-register the device-specific control-ioctls

## Synopsis

```
int snd_ctl_unregister_ioctl (snd_kctl_ioctl_func_t fcn);
```

## Arguments

*fcn* ioctl callback function to unregister

# Name

snd_ctl_unregister_ioctl_compat — de-register the device-specific compat 32bit control-ioctls

## Synopsis

```
int snd_ctl_unregister_ioctl_compat (snd_kctl_ioctl_func_t fcn);
```

## Arguments

*fcn* ioctl callback function to unregister

# Name

snd_ctl_boolean_mono_info — Helper function for a standard boolean info callback with a mono channel

## Synopsis

```
int snd_ctl_boolean_mono_info (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_info * uinfo);
```

## Arguments

*kcontrol* the kcontrol instance

*uinfo* info to store

## Description

This is a function that can be used as info callback for a standard boolean control with a single mono channel.

# Name

snd_ctl_boolean_stereo_info — Helper function for a standard boolean info callback with stereo two channels

## Synopsis

```
int snd_ctl_boolean_stereo_info (struct snd_kcontrol * kcontrol, struct snd_ctl_elem_info * uinfo);
```

## Arguments

*kcontrol* the kcontrol instance

*uinfo* info to store

## Description

This is a function that can be used as info callback for a standard boolean control with stereo two channels.

# 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

*info* the structure to be filled

*channels* the number of the control's channels; often one

*items* the number of control values; also the size of *names*

*names[]* 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

| | |
|---|---|
| *card* | the card instance |
| *num* | the bus number |
| *ops* | the bus callbacks table |
| *private_data* | private data pointer for the new instance |
| *rbus* | 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, const struct ac97_quirk * quirk, const char * override);
```

## Arguments

*ac97* the ac97 instance

*quirk* quirk list

*override* 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

| | |
|---|---|
| *bus* | the ac97 bus instance |
| *pcms_count* | count of PCMs to be assigned |
| *pcms* | 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

This is a variant of `snd_rawmidi_transmit_peek` without spinlock.

# 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

This is a variant of `__snd_rawmidi_transmit_ack` without spinlock.

# 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

| | |
|---|---|
| *card* | the card instance |
| *id* | the id string |
| *device* | the device index |
| *output_count* | the number of output streams |
| *input_count* | the number of input streams |
| *rrawmidi* | 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

| | |
|---|---|
| *card* | the card instance |
| *device* | the device index, zero-based |
| *hardware* | the hardware type, MPU401_HW_XXXX |
| *port* | the base address of MPU401 port |
| *info_flags* | bitflags MPU401_INFO_XXX |
| *irq* | the ISA irq number, -1 if not to be allocated |
| *rrawmidi* | 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 rawmidi->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_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_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.

# 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. Compress Offload

## Compress Offload API

# Name

snd_compress_register — register compressed device

## Synopsis

```
int snd_compress_register (struct snd_compr * device);
```

## Arguments

*device* compressed device to register

# Name

struct snd_compressed_buffer — compressed buffer

## Synopsis

```
struct snd_compressed_buffer {
  __u32 fragment_size;
  __u32 fragments;
};
```

## Members

| | |
|---|---|
| fragment_size | size of buffer fragment in bytes |
| fragments | number of such fragments |

# Name

struct snd_compr_params — compressed stream params

## Synopsis

```
struct snd_compr_params {
  struct snd_compressed_buffer buffer;
  struct snd_codec codec;
  __u8 no_wake_mode;
};
```

## Members

| | |
|---|---|
| buffer | buffer description |
| codec | codec parameters |
| no_wake_mode | dont wake on fragment elapsed |

# Name

struct snd_compr_tstamp — timestamp descriptor

## Synopsis

```
struct snd_compr_tstamp {
  __u32 byte_offset;
  __u32 copied_total;
  __u32 pcm_frames;
  __u32 pcm_io_frames;
  __u32 sampling_rate;
};
```

## Members

| | |
|---|---|
| byte_offset | Byte offset in ring buffer to DSP |
| copied_total | Total number of bytes copied from/to ring buffer to/by DSP |
| pcm_frames | Frames decoded or encoded by DSP. This field will evolve by large steps and should only be used to monitor encoding/decoding progress. It shall not be used for timing estimates. |
| pcm_io_frames | Frames rendered or received by DSP into a mixer or an audio output/input. This field should be used for A/V sync or time estimates. |
| sampling_rate | sampling rate of audio |

# Name

struct snd_compr_avail — avail descriptor

## Synopsis

```
struct snd_compr_avail {
  __u64 avail;
  struct snd_compr_tstamp tstamp;
};
```

## Members

avail Number of bytes available in ring buffer for writing/reading

tstamp timestamp information

# Name

struct snd_compr_caps — caps descriptor

## Synopsis

```
struct snd_compr_caps {
  __u32 num_codecs;
  __u32 direction;
  __u32 min_fragment_size;
  __u32 max_fragment_size;
  __u32 min_fragments;
  __u32 max_fragments;
  __u32 codecs[MAX_NUM_CODECS];
  __u32 reserved[11];
};
```

## Members

| | |
|---|---|
| num_codecs | number of codecs supported |
| direction | direction supported. Of type snd_compr_direction |
| min_fragment_size | minimum fragment supported by DSP |
| max_fragment_size | maximum fragment supported by DSP |
| min_fragments | min fragments supported by DSP |
| max_fragments | max fragments supported by DSP |
| codecs[MAX_NUM_CODECS] | pointer to array of codecs |
| reserved[11] | reserved field |

# Name

struct snd_compr_codec_caps — query capability of codec

## Synopsis

```
struct snd_compr_codec_caps {
  __u32 codec;
  __u32 num_descriptors;
  struct snd_codec_desc descriptor[MAX_NUM_CODEC_DESCRIPTORS];
};
```

## Members

| | |
|---|---|
| codec | codec for which capability is queried |
| num_descriptors | number of codec descriptors |
| descriptor[MAX_NUM_CODEC_DESCRIPTORS] | array of codec capability descriptor |

# Name

enum sndrv_compress_encoder —

## Synopsis

```
enum sndrv_compress_encoder {
  SNDRV_COMPRESS_ENCODER_PADDING,
  SNDRV_COMPRESS_ENCODER_DELAY
};
```

## Constants

SNDRV_COMPRESS_ENCODER_PADDING
: Number of samples appended by the encoder at the end of the track

SNDRV_COMPRESS_ENCODER_DELAY
: Number of samples inserted by the encoder at the beginning of the track

# Name

struct snd_compr_metadata — compressed stream metadata

## Synopsis

```
struct snd_compr_metadata {
  __u32 key;
  __u32 value[8];
};
```

## Members

key key id

value[8] key value

# Name

SNDRV_COMPRESS_IOCTL_VERSION —

## Synopsis

```
SNDRV_COMPRESS_IOCTL_VERSION (void);
```

## Arguments

None

## SNDRV_COMPRESS_GET_CAPS

Query capability of DSP

## SNDRV_COMPRESS_GET_CODEC_CAPS

Query capability of a codec

## SNDRV_COMPRESS_SET_PARAMS

Set codec and stream parameters

## Note

only codec params can be changed runtime and stream params cant be

## SNDRV_COMPRESS_GET_PARAMS

Query codec params

## SNDRV_COMPRESS_TSTAMP

get the current timestamp value

## SNDRV_COMPRESS_AVAIL

get the current buffer avail value. This also queries the tstamp properties

## SNDRV_COMPRESS_PAUSE

Pause the running stream

## SNDRV_COMPRESS_RESUME

resume a paused stream

## SNDRV_COMPRESS_START

Start a stream

## SNDRV_COMPRESS_STOP

stop a running stream, discarding ring buffer content and the buffers currently with DSP

## SNDRV_COMPRESS_DRAIN

Play till end of buffers and stop after that

## SNDRV_COMPRESS_IOCTL_VERSION

Query the API version

# Name

struct snd_enc_vorbis —

## Synopsis

```
struct snd_enc_vorbis {
  __s32 quality;
  __u32 managed;
  __u32 max_bit_rate;
  __u32 min_bit_rate;
  __u32 downmix;
};
```

## Members

quality — Sets encoding quality to n, between -1 (low) and 10 (high). In the default mode of operation, the quality level is 3. Normal quality range is 0 - 10.

managed — Boolean. Set bitrate management mode. This turns off the normal VBR encoding, but allows hard or soft bitrate constraints to be enforced by the encoder. This mode can be slower, and may also be lower quality. It is primarily useful for streaming.

max_bit_rate — Enabled only if managed is TRUE

min_bit_rate — Enabled only if managed is TRUE

downmix — Boolean. Downmix input from stereo to mono (has no effect on non-stereo streams). Useful for lower-bitrate encoding.

## Description

These options were extracted from the OpenMAX IL spec and Gstreamer vorbisenc properties

For best quality users should specify VBR mode and set quality levels.

# Name

struct snd_enc_real —

## Synopsis

```
struct snd_enc_real {
  __u32 quant_bits;
  __u32 start_region;
  __u32 num_regions;
};
```

## Members

| | |
|---|---|
| quant_bits | number of coupling quantization bits in the stream |
| start_region | coupling start region in the stream |
| num_regions | number of regions value |

## Description

These options were extracted from the OpenMAX IL spec

# Name

struct snd_enc_flac —

## Synopsis

```
struct snd_enc_flac {
  __u32 num;
  __u32 gain;
};
```

## Members

num serial number, valid only for OGG formats needs to be set by application

gain Add replay gain tags

## Description

These options were extracted from the FLAC online documentation

## at http

//flac.sourceforge.net/documentation_tools_flac.html

To make the API simpler, it is assumed that the user will select quality profiles. Additional options that affect encoding quality and speed can be added at a later stage if needed.

By default the Subset format is used by encoders.

TAGS such as pictures, etc, cannot be handled by an offloaded encoder and are not supported in this API.

# Name

struct snd_compr_runtime —

## Synopsis

```
struct snd_compr_runtime {
  snd_pcm_state_t state;
  struct snd_compr_ops * ops;
  void * buffer;
  u64 buffer_size;
  u32 fragment_size;
  u32 fragments;
  u64 total_bytes_available;
  u64 total_bytes_transferred;
  wait_queue_head_t sleep;
  void * private_data;
};
```

## Members

| | |
|---|---|
| state | stream state |
| ops | pointer to DSP callbacks |
| buffer | pointer to kernel buffer, valid only when not in mmap mode or DSP doesn't implement copy |
| buffer_size | size of the above buffer |
| fragment_size | size of buffer fragment in bytes |
| fragments | number of such fragments |
| total_bytes_available | cumulative number of bytes made available in the ring buffer |
| total_bytes_transferred | cumulative bytes transferred by offload DSP |
| sleep | poll sleep |
| private_data | driver private data pointer |

# Name

struct snd_compr_stream —

## Synopsis

```
struct snd_compr_stream {
  const char * name;
  struct snd_compr_ops * ops;
  struct snd_compr_runtime * runtime;
  struct snd_compr * device;
  enum snd_compr_direction direction;
  bool metadata_set;
  bool next_track;
  void * private_data;
};
```

## Members

| | |
|---|---|
| name | device name |
| ops | pointer to DSP callbacks |
| runtime | pointer to runtime structure |
| device | device pointer |
| direction | stream direction, playback/recording |
| metadata_set | metadata set flag, true when set |
| next_track | has userspace signal next track transition, true when set |
| private_data | pointer to DSP private data |

# Name

struct snd_compr_ops —

## Synopsis

```
struct snd_compr_ops {
  int (* open) (struct snd_compr_stream *stream);
  int (* free) (struct snd_compr_stream *stream);
  int (* set_params) (struct snd_compr_stream *stream,struct snd_compr_params *par
  int (* get_params) (struct snd_compr_stream *stream,struct snd_codec *params);
  int (* set_metadata) (struct snd_compr_stream *stream,struct snd_compr_metadata
  int (* get_metadata) (struct snd_compr_stream *stream,struct snd_compr_metadata
  int (* trigger) (struct snd_compr_stream *stream, int cmd);
  int (* pointer) (struct snd_compr_stream *stream,struct snd_compr_tstamp *tstamp
  int (* copy) (struct snd_compr_stream *stream, char __user *buf,size_t count);
  int (* mmap) (struct snd_compr_stream *stream,struct vm_area_struct *vma);
  int (* ack) (struct snd_compr_stream *stream, size_t bytes);
  int (* get_caps) (struct snd_compr_stream *stream,struct snd_compr_caps *caps);
  int (* get_codec_caps) (struct snd_compr_stream *stream,struct snd_compr_codec_c
};
```

## Members

| | |
|---|---|
| open | Open the compressed stream This callback is mandatory and shall keep dsp ready to receive the stream parameter |
| free | Close the compressed stream, mandatory |
| set_params | Sets the compressed stream parameters, mandatory This can be called in during stream creation only to set codec params and the stream properties |
| get_params | retrieve the codec parameters, mandatory |
| set_metadata | Set the metadata values for a stream |
| get_metadata | retrieves the requested metadata values from stream |
| trigger | Trigger operations like start, pause, resume, drain, stop. This callback is mandatory |
| pointer | Retrieve current h/w pointer information. Mandatory |
| copy | Copy the compressed data to/from userspace, Optional Can't be implemented if DSP supports mmap |
| mmap | DSP mmap method to mmap DSP memory |
| ack | Ack for DSP when data is written to audio buffer, Optional Not valid if copy is implemented |
| get_caps | Retrieve DSP capabilities, mandatory |
| get_codec_caps | Retrieve capabilities for a specific codec, mandatory |

# Name

struct snd_compr —

## Synopsis

```
struct snd_compr {
  const char * name;
  struct device dev;
  struct snd_compr_ops * ops;
  void * private_data;
  struct snd_card * card;
  unsigned int direction;
  struct mutex lock;
  int device;
};
```

## Members

| | |
|---|---|
| name | DSP device name |
| dev | associated device instance |
| ops | pointer to DSP callbacks |
| private_data | pointer to DSP pvt data |
| card | sound card pointer |
| direction | Playback or capture direction |
| lock | device lock |
| device | device id |

# Chapter 7. ASoC

## ASoC Core API

# Name

struct snd_soc_jack_pin — Describes a pin to update based on jack detection

## Synopsis

```
struct snd_soc_jack_pin {
  struct list_head list;
  const char * pin;
  int mask;
  bool invert;
};
```

## Members

| | |
|---|---|
| list | internal list entry |
| pin | name of the pin to update |
| mask | bits to check for in reported jack status |
| invert | if non-zero then pin is enabled when status is not reported |

# Name

struct snd_soc_jack_zone — Describes voltage zones of jack detection

## Synopsis

```
struct snd_soc_jack_zone {
  unsigned int min_mv;
  unsigned int max_mv;
  unsigned int jack_type;
  unsigned int debounce_time;
  struct list_head list;
};
```

## Members

| | |
|---|---|
| min_mv | start voltage in mv |
| max_mv | end voltage in mv |
| jack_type | type of jack that is expected for this voltage |
| debounce_time | debounce_time for jack, codec driver should wait for this duration before reading the adc for voltages |
| list | internal list entry |

# Name

struct snd_soc_jack_gpio — Describes a gpio pin for jack detection

## Synopsis

```
struct snd_soc_jack_gpio {
  unsigned int gpio;
  unsigned int idx;
  struct device * gpiod_dev;
  const char * name;
  int report;
  int invert;
  int debounce_time;
  bool wake;
  int (* jack_status_check) (void *data);
};
```

## Members

| | |
|---|---|
| gpio | legacy gpio number |
| idx | gpio descriptor index within the function of the GPIO consumer device |
| gpiod_dev | GPIO consumer device |
| name | gpio name. Also as connection ID for the GPIO consumer device function name lookup |
| report | value to report when jack detected |
| invert | report presence in low state |
| debounce_time | debounce time in ms |
| wake | enable as wake source |
| jack_status_check | callback function which overrides the detection to provide more complex checks (eg, reading an ADC). |

# Name

snd_soc_component_to_codec — Casts a component to the CODEC it is embedded in

## Synopsis

```
struct snd_soc_codec * snd_soc_component_to_codec (struct
snd_soc_component * component);
```

## Arguments

*component* The component to cast to a CODEC

## Description

This function must only be used on components that are known to be CODECs. Otherwise the behavior is undefined.

# Name

snd_soc_component_to_platform — Casts a component to the platform it is embedded in

## Synopsis

```
struct snd_soc_platform * snd_soc_component_to_platform (struct
snd_soc_component * component);
```

## Arguments

*component* The component to cast to a platform

## Description

This function must only be used on components that are known to be platforms. Otherwise the behavior is undefined.

# Name

snd_soc_dapm_to_component — Casts a DAPM context to the component it is embedded in

## Synopsis

```
struct snd_soc_component * snd_soc_dapm_to_component (struct
snd_soc_dapm_context * dapm);
```

## Arguments

*dapm* The DAPM context to cast to the component

## Description

This function must only be used on DAPM contexts that are known to be part of a component (e.g. in a component driver). Otherwise the behavior is undefined.

# Name

snd_soc_dapm_to_codec — Casts a DAPM context to the CODEC it is embedded in

## Synopsis

```
struct snd_soc_codec * snd_soc_dapm_to_codec (struct
snd_soc_dapm_context * dapm);
```

## Arguments

*dapm* The DAPM context to cast to the CODEC

## Description

This function must only be used on DAPM contexts that are known to be part of a CODEC (e.g. in a CODEC driver). Otherwise the behavior is undefined.

# Name

snd_soc_dapm_to_platform — Casts a DAPM context to the platform it is embedded in

## Synopsis

```
struct snd_soc_platform * snd_soc_dapm_to_platform (struct
snd_soc_dapm_context * dapm);
```

## Arguments

*dapm* The DAPM context to cast to the platform.

## Description

This function must only be used on DAPM contexts that are known to be part of a platform (e.g. in a platform driver). Otherwise the behavior is undefined.

# Name

snd_soc_component_get_dapm — Returns the DAPM context associated with a component

## Synopsis

```
struct snd_soc_dapm_context * snd_soc_component_get_dapm (struct
snd_soc_component * component);
```

## Arguments

*component* The component for which to get the DAPM context

# Name

snd_soc_codec_get_dapm — Returns the DAPM context for the CODEC

## Synopsis

```
struct snd_soc_dapm_context * snd_soc_codec_get_dapm (struct
snd_soc_codec * codec);
```

## Arguments

*codec* The CODEC for which to get the DAPM context

## Note

Use this function instead of directly accessing the CODEC's dapm field

# Name

snd_soc_codec_init_bias_level — Initialize CODEC DAPM bias level

## Synopsis

```
void snd_soc_codec_init_bias_level (struct snd_soc_codec * codec, enum
snd_soc_bias_level level);
```

## Arguments

*codec* The CODEC for which to initialize the DAPM bias level

*level* The DAPM level to initialize to

## Description

Initializes the CODEC DAPM bias level. See `snd_soc_dapm_init_bias_level`.

# Name

snd_soc_codec_get_bias_level — Get current CODEC DAPM bias level

## Synopsis

```
enum snd_soc_bias_level snd_soc_codec_get_bias_level (struct
snd_soc_codec * codec);
```

## Arguments

*codec* The CODEC for which to get the DAPM bias level

## Returns

The current DAPM bias level of the CODEC.

# Name

snd_soc_codec_force_bias_level — Set the CODEC DAPM bias level

## Synopsis

```
int snd_soc_codec_force_bias_level (struct snd_soc_codec * codec, enum
snd_soc_bias_level level);
```

## Arguments

*codec* The CODEC for which to set the level

*level* The level to set to

## Description

Forces the CODEC bias level to a specific state. See `snd_soc_dapm_force_bias_level`.

# Name

snd_soc_dapm_kcontrol_codec — Returns the codec associated to a kcontrol

## Synopsis

```
struct snd_soc_codec * snd_soc_dapm_kcontrol_codec (struct snd_kcontrol
* kcontrol);
```

## Arguments

*kcontrol* The kcontrol

## Description

This function must only be used on DAPM contexts that are known to be part of a CODEC (e.g. in a CODEC driver). Otherwise the behavior is undefined.

# Name

snd_soc_cache_sync — Sync the register cache with the hardware

## Synopsis

```
int snd_soc_cache_sync (struct snd_soc_codec * codec);
```

## Arguments

*codec* CODEC to sync

## Note

This function will call `regcache_sync`

# Name

snd_soc_codec_init_regmap — Initialize regmap instance for the CODEC

## Synopsis

```
void snd_soc_codec_init_regmap (struct snd_soc_codec * codec, struct
regmap * regmap);
```

## Arguments

*codec* The CODEC for which to initialize the regmap instance

*regmap* The regmap instance that should be used by the CODEC

## Description

This function allows deferred assignment of the regmap instance that is associated with the CODEC. Only use this if the regmap instance is not yet ready when the CODEC is registered. The function must also be called before the first IO attempt of the CODEC.

# Name

snd_soc_codec_exit_regmap — De-initialize regmap instance for the CODEC

## Synopsis

```
void snd_soc_codec_exit_regmap (struct snd_soc_codec * codec);
```

## Arguments

*codec* The CODEC for which to de-initialize the regmap instance

## Description

Calls `regmap_exit` on the regmap instance associated to the CODEC and removes the regmap instance from the CODEC.

This function should only be used if `snd_soc_codec_init_regmap` was used to initialize the regmap instance.

# Name

snd_soc_kcontrol_component — Returns the component that registered the control

## Synopsis

```
struct snd_soc_component * snd_soc_kcontrol_component (struct
snd_kcontrol * kcontrol);
```

## Arguments

*kcontrol* The control for which to get the component

## Note

This function will work correctly if the control has been registered for a component. Either with `snd_soc_add_codec_controls` or `snd_soc_add_platform_controls` or via table based setup for either a CODEC, a platform or component driver. Otherwise the behavior is undefined.

# Name

snd_soc_kcontrol_codec — Returns the CODEC that registered the control

## Synopsis

```
struct snd_soc_codec * snd_soc_kcontrol_codec (struct snd_kcontrol *
kcontrol);
```

## Arguments

*kcontrol* The control for which to get the CODEC

## Note

This function will only work correctly if the control has been registered with `snd_soc_add_codec_controls` or via table based setup of snd_soc_codec_driver. Otherwise the behavior is undefined.

# Name

snd_soc_kcontrol_platform — Returns the platform that registered the control

## Synopsis

```
struct snd_soc_platform * snd_soc_kcontrol_platform (struct
snd_kcontrol * kcontrol);
```

## Arguments

*kcontrol* The control for which to get the platform

## Note

This function will only work correctly if the control has been registered with `snd_soc_add_platform_controls` or via table based setup of a snd_soc_platform_driver. Otherwise the behavior is undefined.

# Name

snd_soc_runtime_set_dai_fmt — Change DAI link format for a ASoC runtime

## Synopsis

```
int snd_soc_runtime_set_dai_fmt (struct snd_soc_pcm_runtime * rtd,
unsigned int dai_fmt);
```

## Arguments

*rtd* The runtime for which the DAI link format should be changed

*dai_fmt* The new DAI link format

## Description

This function updates the DAI link format for all DAIs connected to the DAI link for the specified runtime.

## Note

For setups with a static format set the dai_fmt field in the corresponding snd_dai_link struct instead of using this function.

Returns 0 on success, otherwise a negative error code.

# Name

snd_soc_cnew — create new control

## Synopsis

```
struct snd_kcontrol * snd_soc_cnew (const struct snd_kcontrol_new *
_template, void * data, const char * long_name, const char * prefix);
```

## Arguments

| | |
|---|---|
| *_template* | control template |
| *data* | control private data |
| *long_name* | control long name |
| *prefix* | control name prefix |

## Description

Create a new mixer control from a template control.

Returns 0 for success, else error.

# Name

snd_soc_add_component_controls — Add an array of controls to a component.

## Synopsis

```
int snd_soc_add_component_controls (struct snd_soc_component * component, const struct snd_kcontrol_new * controls, unsigned int num_controls);
```

## Arguments

| | |
|---|---|
| *component* | Component to add controls to |
| *controls* | Array of controls to add |
| *num_controls* | Number of elements in the array |

## Return

0 for success, else error.

# Name

snd_soc_add_codec_controls — add an array of controls to a codec. Convenience function to add a list of controls. Many codecs were duplicating this code.

## Synopsis

```
int snd_soc_add_codec_controls (struct snd_soc_codec * codec, const
struct snd_kcontrol_new * controls, unsigned int num_controls);
```

## Arguments

| | |
|---|---|
| *codec* | codec to add controls to |
| *controls* | array of controls to add |
| *num_controls* | number of elements in the array |

## Description

Return 0 for success, else error.

# Name

snd_soc_add_platform_controls — add an array of controls to a platform. Convenience function to add a list of controls.

## Synopsis

```
int snd_soc_add_platform_controls (struct snd_soc_platform * platform,
const struct snd_kcontrol_new * controls, unsigned int num_controls);
```

## Arguments

| | |
|---|---|
| *platform* | platform to add controls to |
| *controls* | array of controls to add |
| *num_controls* | number of elements in the array |

## Description

Return 0 for success, else error.

# Name

snd_soc_add_card_controls — add an array of controls to a SoC card. Convenience function to add a list of controls.

## Synopsis

```
int snd_soc_add_card_controls (struct snd_soc_card * soc_card, const
struct snd_kcontrol_new * controls, int num_controls);
```

## Arguments

| | |
|---|---|
| *soc_card* | SoC card to add controls to |
| *controls* | array of controls to add |
| *num_controls* | number of elements in the array |

## Description

Return 0 for success, else error.

# Name

snd_soc_add_dai_controls — add an array of controls to a DAI. Convienience function to add a list of controls.

## Synopsis

```
int snd_soc_add_dai_controls (struct snd_soc_dai * dai, const struct
snd_kcontrol_new * controls, int num_controls);
```

## Arguments

| | |
|---|---|
| *dai* | DAI to add controls to |
| *controls* | array of controls to add |
| *num_controls* | number of elements in the array |

## Description

Return 0 for success, else error.

# Name

snd_soc_dai_set_sysclk — configure DAI system or master clock.

## Synopsis

```
int snd_soc_dai_set_sysclk (struct snd_soc_dai * dai, int clk_id,
unsigned int freq, int dir);
```

## Arguments

*dai* DAI

*clk_id* DAI specific clock ID

*freq* new clock frequency in Hz

*dir* new clock direction - input/output.

## Description

Configures the DAI master (MCLK) or system (SYSCLK) clocking.

# Name

snd_soc_codec_set_sysclk — configure CODEC system or master clock.

## Synopsis

```
int snd_soc_codec_set_sysclk (struct snd_soc_codec * codec, int clk_id,
int source, unsigned int freq, int dir);
```

## Arguments

*codec* CODEC

*clk_id* DAI specific clock ID

*source* Source for the clock

*freq* new clock frequency in Hz

*dir* new clock direction - input/output.

## Description

Configures the CODEC master (MCLK) or system (SYSCLK) clocking.

# Name

snd_soc_dai_set_clkdiv — configure DAI clock dividers.

## Synopsis

```
int snd_soc_dai_set_clkdiv (struct snd_soc_dai * dai, int div_id, int
div);
```

## Arguments

*dai* DAI

*div_id* DAI specific clock divider ID

*div* new clock divisor.

## Description

Configures the clock dividers. This is used to derive the best DAI bit and frame clocks from the system or master clock. It's best to set the DAI bit and frame clocks as low as possible to save system power.

# Name

snd_soc_dai_set_pll — configure DAI PLL.

## Synopsis

```
int snd_soc_dai_set_pll (struct snd_soc_dai * dai, int pll_id, int source, unsigned int freq_in, unsigned int freq_out);
```

## Arguments

| | |
|---|---|
| *dai* | DAI |
| *pll_id* | DAI specific PLL ID |
| *source* | DAI specific source for the PLL |
| *freq_in* | PLL input clock frequency in Hz |
| *freq_out* | requested PLL output clock frequency in Hz |

## Description

Configures and enables PLL to generate output clock based on input clock.

# Name

snd_soc_dai_set_bclk_ratio — configure BCLK to sample rate ratio.

## Synopsis

```
int snd_soc_dai_set_bclk_ratio (struct snd_soc_dai * dai, unsigned int
ratio);
```

## Arguments

*dai* DAI

*ratio* Ratio of BCLK to Sample rate.

## Description

Configures the DAI for a preset BCLK to sample rate ratio.

# Name

snd_soc_dai_set_fmt — configure DAI hardware audio format.

## Synopsis

```
int snd_soc_dai_set_fmt (struct snd_soc_dai * dai, unsigned int fmt);
```

## Arguments

*dai* DAI

*fmt* SND_SOC_DAIFMT_ format value.

## Description

Configures the DAI hardware format and clocking.

# Name

snd_soc_dai_set_tdm_slot — Configures a DAI for TDM operation

## Synopsis

```
int snd_soc_dai_set_tdm_slot (struct snd_soc_dai * dai, unsigned int
tx_mask, unsigned int rx_mask, int slots, int slot_width);
```

## Arguments

| | |
|---|---|
| *dai* | The DAI to configure |
| *tx_mask* | bitmask representing active TX slots. |
| *rx_mask* | bitmask representing active RX slots. |
| *slots* | Number of slots in use. |
| *slot_width* | Width in bits for each slot. |

## Description

This function configures the specified DAI for TDM operation. *slot* contains the total number of slots of the TDM stream and *slot_with* the width of each slot in bit clock cycles. *tx_mask* and *rx_mask* are bitmasks specifying the active slots of the TDM stream for the specified DAI, i.e. which slots the DAI should write to or read from. If a bit is set the corresponding slot is active, if a bit is cleared the corresponding slot is inactive. Bit 0 maps to the first slot, bit 1 to the second slot and so on. The first active slot maps to the first channel of the DAI, the second active slot to the second channel and so on.

TDM mode can be disabled by passing 0 for *slots*. In this case *tx_mask*, *rx_mask* and *slot_width* will be ignored.

Returns 0 on success, a negative error code otherwise.

# Name

snd_soc_dai_set_channel_map — configure DAI audio channel map

## Synopsis

```
int snd_soc_dai_set_channel_map (struct snd_soc_dai * dai, unsigned int
tx_num, unsigned int * tx_slot, unsigned int rx_num, unsigned int *
rx_slot);
```

## Arguments

*dai* DAI

*tx_num* how many TX channels

*tx_slot* pointer to an array which imply the TX slot number channel 0~num-1 uses

*rx_num* how many RX channels

*rx_slot* pointer to an array which imply the RX slot number channel 0~num-1 uses

## Description

configure the relationship between channel number and TDM slot number.

# Name

snd_soc_dai_set_tristate — configure DAI system or master clock.

## Synopsis

```
int snd_soc_dai_set_tristate (struct snd_soc_dai * dai, int tristate);
```

## Arguments

*dai* DAI

*tristate* tristate enable

## Description

Tristates the DAI so that others can use it.

# Name

snd_soc_dai_digital_mute — configure DAI system or master clock.

## Synopsis

```
int snd_soc_dai_digital_mute (struct snd_soc_dai * dai, int mute, int
direction);
```

## Arguments

| | |
|---|---|
| *dai* | DAI |
| *mute* | mute enable |
| *direction* | stream to mute |

## Description

Mutes the DAI DAC.

# Name

snd_soc_register_card — Register a card with the ASoC core

## Synopsis

```
int snd_soc_register_card (struct snd_soc_card * card);
```

## Arguments

*card* Card to register

# Name

snd_soc_unregister_card — Unregister a card with the ASoC core

## Synopsis

```
int snd_soc_unregister_card (struct snd_soc_card * card);
```

## Arguments

*card* Card to unregister

# Name

snd_soc_component_init_regmap — Initialize regmap instance for the component

## Synopsis

```
void snd_soc_component_init_regmap (struct snd_soc_component * component, struct regmap * regmap);
```

## Arguments

*component* The component for which to initialize the regmap instance

*regmap* The regmap instance that should be used by the component

## Description

This function allows deferred assignment of the regmap instance that is associated with the component. Only use this if the regmap instance is not yet ready when the component is registered. The function must also be called before the first IO attempt of the component.

# Name

snd_soc_component_exit_regmap — De-initialize regmap instance for the component

## Synopsis

```
void snd_soc_component_exit_regmap (struct snd_soc_component * component);
```

## Arguments

*component* The component for which to de-initialize the regmap instance

## Description

Calls `regmap_exit` on the regmap instance associated to the component and removes the regmap instance from the component.

This function should only be used if `snd_soc_component_init_regmap` was used to initialize the regmap instance.

# Name

snd_soc_unregister_component — Unregister a component from the ASoC core

## Synopsis

```
void snd_soc_unregister_component (struct device * dev);
```

## Arguments

*dev* The device to unregister

# Name

snd_soc_add_platform — Add a platform to the ASoC core

## Synopsis

```
int snd_soc_add_platform (struct device * dev, struct snd_soc_platform
* platform, const struct snd_soc_platform_driver * platform_drv);
```

## Arguments

| | |
|---|---|
| *dev* | The parent device for the platform |
| *platform* | The platform to add |
| *platform_drv* | The driver for the platform |

# Name

snd_soc_register_platform — Register a platform with the ASoC core

## Synopsis

```
int snd_soc_register_platform (struct device * dev, const struct
snd_soc_platform_driver * platform_drv);
```

## Arguments

*dev* The device for the platform

*platform_drv* The driver for the platform

# Name

snd_soc_remove_platform — Remove a platform from the ASoC core

## Synopsis

```
void snd_soc_remove_platform (struct snd_soc_platform * platform);
```

## Arguments

*platform* the platform to remove

# Name

snd_soc_unregister_platform — Unregister a platform from the ASoC core

## Synopsis

```
void snd_soc_unregister_platform (struct device * dev);
```

## Arguments

*dev* platform to unregister

# Name

snd_soc_register_codec — Register a codec with the ASoC core

## Synopsis

```
int snd_soc_register_codec (struct device * dev, const struct
snd_soc_codec_driver * codec_drv, struct snd_soc_dai_driver * dai_drv,
int num_dai);
```

## Arguments

| | |
|---|---|
| *dev* | The parent device for this codec |
| *codec_drv* | Codec driver |
| *dai_drv* | The associated DAI driver |
| *num_dai* | Number of DAIs |

# Name

snd_soc_unregister_codec — Unregister a codec from the ASoC core

## Synopsis

```
void snd_soc_unregister_codec (struct device * dev);
```

## Arguments

*dev* codec to unregister

# Name

devm_snd_soc_register_component — resource managed component registration

## Synopsis

```
int devm_snd_soc_register_component (struct device * dev, const struct
snd_soc_component_driver * cmpnt_drv, struct snd_soc_dai_driver *
dai_drv, int num_dai);
```

## Arguments

| | |
|---|---|
| *dev* | Device used to manage component |
| *cmpnt_drv* | Component driver |
| *dai_drv* | DAI driver |
| *num_dai* | Number of DAIs to register |

## Description

Register a component with automatic unregistration when the device is unregistered.

# Name

devm_snd_soc_register_platform — resource managed platform registration

## Synopsis

```
int devm_snd_soc_register_platform (struct device * dev, const struct
snd_soc_platform_driver * platform_drv);
```

## Arguments

| | |
|---|---|
| *dev* | Device used to manage platform |
| *platform_drv* | platform to register |

## Description

Register a platform driver with automatic unregistration when the device is unregistered.

# Name

devm_snd_soc_register_card — resource managed card registration

## Synopsis

```
int devm_snd_soc_register_card (struct device * dev, struct snd_soc_card
* card);
```

## Arguments

*dev* Device used to manage card

*card* Card to register

## Description

Register a card with automatic unregistration when the device is unregistered.

# Name

devm_snd_dmaengine_pcm_register — resource managed dmaengine PCM registration

## Synopsis

```
int devm_snd_dmaengine_pcm_register (struct device * dev, const struct
snd_dmaengine_pcm_config * config, unsigned int flags);
```

## Arguments

*dev* The parent device for the PCM device

*config* Platform specific PCM configuration

*flags* Platform specific quirks

## Description

Register a dmaengine based PCM device with automatic unregistration when the device is unregistered.

# Name

snd_soc_component_read — Read register value

## Synopsis

```
int snd_soc_component_read (struct snd_soc_component * component,
unsigned int reg, unsigned int * val);
```

## Arguments

| | |
|---|---|
| *component* | Component to read from |
| *reg* | Register to read |
| *val* | Pointer to where the read value is stored |

## Return

0 on success, a negative error code otherwise.

# Name

snd_soc_component_write — Write register value

## Synopsis

```
int snd_soc_component_write (struct snd_soc_component * component,
unsigned int reg, unsigned int val);
```

## Arguments

*component* Component to write to

*reg* Register to write

*val* Value to write to the register

## Return

0 on success, a negative error code otherwise.

# Name

snd_soc_component_update_bits — Perform read/modify/write cycle

## Synopsis

```
int snd_soc_component_update_bits (struct snd_soc_component * component, unsigned int reg, unsigned int mask, unsigned int val);
```

## Arguments

| | |
|---|---|
| *component* | Component to update |
| *reg* | Register to update |
| *mask* | Mask that specifies which bits to update |
| *val* | New value for the bits specified by mask |

## Return

1 if the operation was successful and the value of the register changed, 0 if the operation was successful, but the value did not change. Returns a negative error code otherwise.

# Name

snd_soc_component_update_bits_async — Perform asynchronous read/modify/write cycle

## Synopsis

```
int snd_soc_component_update_bits_async (struct snd_soc_component *
component, unsigned int reg, unsigned int mask, unsigned int val);
```

## Arguments

| | |
|---|---|
| *component* | Component to update |
| *reg* | Register to update |
| *mask* | Mask that specifies which bits to update |
| *val* | New value for the bits specified by mask |

## Description

This function is similar to `snd_soc_component_update_bits`, but the update operation is scheduled asynchronously. This means it may not be completed when the function returns. To make sure that all scheduled updates have been completed `snd_soc_component_async_complete` must be called.

## Return

1 if the operation was successful and the value of the register changed, 0 if the operation was successful, but the value did not change. Returns a negative error code otherwise.

# Name

snd_soc_component_async_complete — Ensure asynchronous I/O has completed

## Synopsis

```
void snd_soc_component_async_complete (struct snd_soc_component * component);
```

## Arguments

*component* Component for which to wait

## Description

This function blocks until all asynchronous I/O which has previously been scheduled using `snd_soc_component_update_bits_async` has completed.

# Name

snd_soc_component_test_bits — Test register for change

## Synopsis

```
int snd_soc_component_test_bits (struct snd_soc_component * component,
unsigned int reg, unsigned int mask, unsigned int value);
```

## Arguments

| | |
|---|---|
| *component* | component |
| *reg* | Register to test |
| *mask* | Mask that specifies which bits to test |
| *value* | Value to test against |

## Description

Tests a register with a new value and checks if the new value is different from the old value.

## Return

1 for change, otherwise 0.

# Name

snd_soc_update_bits — update codec register bits

## Synopsis

```
int snd_soc_update_bits (struct snd_soc_codec * codec, unsigned int reg,
unsigned int mask, unsigned int value);
```

## Arguments

*codec* audio codec

*reg* codec register

*mask* register mask

*value* new value

## Description

Writes new register value.

Returns 1 for change, 0 for no change, or negative error code.

# Name

snd_soc_test_bits — test register for change

## Synopsis

```
int snd_soc_test_bits (struct snd_soc_codec * codec, unsigned int reg,
unsigned int mask, unsigned int value);
```

## Arguments

*codec* audio codec

*reg* codec register

*mask* register mask

*value* new value

## Description

Tests a register with a new value and checks if the new value is different from the old value.

Returns 1 for change else 0.

# Name

snd_soc_set_runtime_hwparams — set the runtime hardware parameters

## Synopsis

```
int snd_soc_set_runtime_hwparams (struct snd_pcm_substream * substream,
const struct snd_pcm_hardware * hw);
```

## Arguments

*substream* the pcm substream

*hw* the hardware parameters

## Description

Sets the substream runtime hardware parameters.

# Name

snd_soc_info_enum_double — enumerated double mixer info callback

## Synopsis

```
int snd_soc_info_enum_double (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_info * uinfo);
```

## Arguments

*kcontrol* mixer control

*uinfo* control element information

## Description

Callback to provide information about a double enumerated mixer control.

Returns 0 for success.

# Name

snd_soc_get_enum_double — enumerated double mixer get callback

## Synopsis

```
int snd_soc_get_enum_double (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_value * ucontrol);
```

## Arguments

*kcontrol* mixer control

*ucontrol* control element information

## Description

Callback to get the value of a double enumerated mixer.

Returns 0 for success.

# Name

snd_soc_put_enum_double — enumerated double mixer put callback

## Synopsis

```
int snd_soc_put_enum_double (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_value * ucontrol);
```

## Arguments

*kcontrol* mixer control

*ucontrol* control element information

## Description

Callback to set the value of a double enumerated mixer.

Returns 0 for success.

# Name

snd_soc_info_volsw — single mixer info callback

## Synopsis

```
int snd_soc_info_volsw (struct snd_kcontrol * kcontrol, struct snd_ctl_elem_info * uinfo);
```

## Arguments

*kcontrol* mixer control

*uinfo* control element information

## Description

Callback to provide information about a single mixer control, or a double mixer control that spans 2 registers.

Returns 0 for success.

# Name

snd_soc_info_volsw_sx — Mixer info callback for SX TLV controls

## Synopsis

```
int snd_soc_info_volsw_sx (struct snd_kcontrol * kcontrol, struct snd_ctl_elem_info * uinfo);
```

## Arguments

*kcontrol* mixer control

*uinfo* control element information

## Description

Callback to provide information about a single mixer control, or a double mixer control that spans 2 registers of the SX TLV type. SX TLV controls have a range that represents both positive and negative values either side of zero but without a sign bit.

Returns 0 for success.

# Name

snd_soc_get_volsw — single mixer get callback

## Synopsis

```
int snd_soc_get_volsw (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_value * ucontrol);
```

## Arguments

*kcontrol* mixer control

*ucontrol* control element information

## Description

Callback to get the value of a single mixer control, or a double mixer control that spans 2 registers.

Returns 0 for success.

# Name

snd_soc_put_volsw — single mixer put callback

## Synopsis

```
int snd_soc_put_volsw (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_value * ucontrol);
```

## Arguments

*kcontrol* mixer control

*ucontrol* control element information

## Description

Callback to set the value of a single mixer control, or a double mixer control that spans 2 registers.

Returns 0 for success.

# Name

snd_soc_get_volsw_sx — single mixer get callback

## Synopsis

```
int snd_soc_get_volsw_sx (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_value * ucontrol);
```

## Arguments

*kcontrol* mixer control

*ucontrol* control element information

## Description

Callback to get the value of a single mixer control, or a double mixer control that spans 2 registers.

Returns 0 for success.

# Name

snd_soc_put_volsw_sx — double mixer set callback

## Synopsis

```
int snd_soc_put_volsw_sx (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_value * ucontrol);
```

## Arguments

*kcontrol* mixer control

*ucontrol* control element information

## Description

Callback to set the value of a double mixer control that spans 2 registers.

Returns 0 for success.

# Name

snd_soc_info_volsw_range — single mixer info callback with range.

## Synopsis

```
int snd_soc_info_volsw_range (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_info * uinfo);
```

## Arguments

*kcontrol* mixer control

*uinfo* control element information

## Description

Callback to provide information, within a range, about a single mixer control.

returns 0 for success.

# Name

snd_soc_put_volsw_range — single mixer put value callback with range.

## Synopsis

```
int snd_soc_put_volsw_range (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_value * ucontrol);
```

## Arguments

*kcontrol* mixer control

*ucontrol* control element information

## Description

Callback to set the value, within a range, for a single mixer control.

Returns 0 for success.

# Name

snd_soc_get_volsw_range — single mixer get callback with range

## Synopsis

```
int snd_soc_get_volsw_range (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_value * ucontrol);
```

## Arguments

*kcontrol* mixer control

*ucontrol* control element information

## Description

Callback to get the value, within a range, of a single mixer control.

Returns 0 for success.

# Name

snd_soc_limit_volume — Set new limit to an existing volume control.

## Synopsis

```
int snd_soc_limit_volume (struct snd_soc_card * card, const char * name,
int max);
```

## Arguments

*card* where to look for the control

*name* Name of the control

*max* new maximum limit

## Description

Return 0 for success, else error.

# Name

snd_soc_info_xr_sx — signed multi register info callback

## Synopsis

```
int snd_soc_info_xr_sx (struct snd_kcontrol * kcontrol, struct snd_ctl_elem_info * uinfo);
```

## Arguments

*kcontrol* mreg control

*uinfo* control element information

## Description

Callback to provide information of a control that can span multiple codec registers which together forms a single signed value in a MSB/LSB manner.

Returns 0 for success.

# Name

snd_soc_get_xr_sx — signed multi register get callback

## Synopsis

```
int snd_soc_get_xr_sx (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_value * ucontrol);
```

## Arguments

*kcontrol* mreg control

*ucontrol* control element information

## Description

Callback to get the value of a control that can span multiple codec registers which together forms a single signed value in a MSB/LSB manner. The control supports specifying total no of bits used to allow for bitfields across the multiple codec registers.

Returns 0 for success.

# Name

snd_soc_put_xr_sx — signed multi register get callback

## Synopsis

```
int snd_soc_put_xr_sx (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_value * ucontrol);
```

## Arguments

*kcontrol* mreg control

*ucontrol* control element information

## Description

Callback to set the value of a control that can span multiple codec registers which together forms a single signed value in a MSB/LSB manner. The control supports specifying total no of bits used to allow for bitfields across the multiple codec registers.

Returns 0 for success.

# Name

snd_soc_get_strobe — strobe get callback

## Synopsis

```
int snd_soc_get_strobe (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_value * ucontrol);
```

## Arguments

*kcontrol* mixer control

*ucontrol* control element information

## Description

Callback get the value of a strobe mixer control.

Returns 0 for success.

# Name

snd_soc_put_strobe — strobe put callback

## Synopsis

```
int snd_soc_put_strobe (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_value * ucontrol);
```

## Arguments

*kcontrol* mixer control

*ucontrol* control element information

## Description

Callback strobe a register bit to high then low (or the inverse) in one pass of a single mixer enum control.

Returns 1 for success.

## Name

snd_soc_new_compress — create a new compress.

## Synopsis

```
int snd_soc_new_compress (struct snd_soc_pcm_runtime * rtd, int num);
```

## Arguments

*rtd* The runtime for which we will create compress

*num* the device index number (zero based - shared with normal PCMs)

## Return

0 for success, else error.

# ASoC DAPM API

# Name

snd_soc_dapm_kcontrol_widget — Returns the widget associated to a kcontrol

## Synopsis

```
struct snd_soc_dapm_widget * snd_soc_dapm_kcontrol_widget (struct
snd_kcontrol * kcontrol);
```

## Arguments

*kcontrol* The kcontrol

# Name

snd_soc_dapm_kcontrol_dapm — Returns the dapm context associated to a kcontrol

## Synopsis

```
struct snd_soc_dapm_context * snd_soc_dapm_kcontrol_dapm (struct
snd_kcontrol * kcontrol);
```

## Arguments

*kcontrol* The kcontrol

## Note

This function must only be used on kcontrols that are known to have been registered for a CODEC. Otherwise the behaviour is undefined.

# Name

snd_soc_dapm_force_bias_level — Sets the DAPM bias level

## Synopsis

```
int snd_soc_dapm_force_bias_level (struct snd_soc_dapm_context * dapm,
enum snd_soc_bias_level level);
```

## Arguments

*dapm* The DAPM context for which to set the level

*level* The level to set

## Description

Forces the DAPM bias level to a specific state. It will call the bias level callback of DAPM context with the specified level. This will even happen if the context is already at the same level. Furthermore it will not go through the normal bias level sequencing, meaning any intermediate states between the current and the target state will not be entered.

Note that the change in bias level is only temporary and the next time `snd_soc_dapm_sync` is called the state will be set to the level as determined by the DAPM core. The function is mainly intended to be used to used during probe or resume from suspend to power up the device so initialization can be done, before the DAPM core takes over.

# Name

snd_soc_dapm_sync_unlocked — scan and power dapm paths

## Synopsis

```
int snd_soc_dapm_sync_unlocked (struct snd_soc_dapm_context * dapm);
```

## Arguments

*dapm* DAPM context

## Description

Walks all dapm audio paths and powers widgets according to their stream or path usage.

Requires external locking.

Returns 0 for success.

# Name

snd_soc_dapm_sync — scan and power dapm paths

## Synopsis

```
int snd_soc_dapm_sync (struct snd_soc_dapm_context * dapm);
```

## Arguments

*dapm* DAPM context

## Description

Walks all dapm audio paths and powers widgets according to their stream or path usage.

Returns 0 for success.

# Name

snd_soc_dapm_add_routes — Add routes between DAPM widgets

## Synopsis

```
int snd_soc_dapm_add_routes (struct snd_soc_dapm_context * dapm, const
struct snd_soc_dapm_route * route, int num);
```

## Arguments

*dapm* DAPM context

*route* audio routes

*num* number of routes

## Description

Connects 2 dapm widgets together via a named audio path. The sink is the widget receiving the audio signal, whilst the source is the sender of the audio signal.

Returns 0 for success else error. On error all resources can be freed with a call to `snd_soc_card_free`.

# Name

snd_soc_dapm_del_routes — Remove routes between DAPM widgets

## Synopsis

```
int snd_soc_dapm_del_routes (struct snd_soc_dapm_context * dapm, const
struct snd_soc_dapm_route * route, int num);
```

## Arguments

*dapm* DAPM context

*route* audio routes

*num* number of routes

## Description

Removes routes from the DAPM context.

# Name

snd_soc_dapm_weak_routes — Mark routes between DAPM widgets as weak

## Synopsis

```
int snd_soc_dapm_weak_routes (struct snd_soc_dapm_context * dapm, const
struct snd_soc_dapm_route * route, int num);
```

## Arguments

*dapm* DAPM context

*route* audio routes

*num* number of routes

## Description

Mark existing routes matching those specified in the passed array as being weak, meaning that they are ignored for the purpose of power decisions. The main intended use case is for sidetone paths which couple audio between other independent paths if they are both active in order to make the combination work better at the user level but which aren't intended to be “used”.

Note that CODEC drivers should not use this as sidetone type paths can frequently also be used as bypass paths.

# Name

snd_soc_dapm_new_widgets — add new dapm widgets

## Synopsis

```
int snd_soc_dapm_new_widgets (struct snd_soc_card * card);
```

## Arguments

*card* card to be checked for new dapm widgets

## Description

Checks the codec for any new dapm widgets and creates them if found.

Returns 0 for success.

# Name

snd_soc_dapm_get_volsw — dapm mixer get callback

## Synopsis

```
int snd_soc_dapm_get_volsw (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_value * ucontrol);
```

## Arguments

*kcontrol* mixer control

*ucontrol* control element information

## Description

Callback to get the value of a dapm mixer control.

Returns 0 for success.

# Name

snd_soc_dapm_put_volsw — dapm mixer set callback

## Synopsis

```
int snd_soc_dapm_put_volsw (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_value * ucontrol);
```

## Arguments

*kcontrol* mixer control

*ucontrol* control element information

## Description

Callback to set the value of a dapm mixer control.

Returns 0 for success.

# Name

snd_soc_dapm_get_enum_double — dapm enumerated double mixer get callback

## Synopsis

```
int snd_soc_dapm_get_enum_double (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_value * ucontrol);
```

## Arguments

*kcontrol* mixer control

*ucontrol* control element information

## Description

Callback to get the value of a dapm enumerated double mixer control.

Returns 0 for success.

# Name

snd_soc_dapm_put_enum_double — dapm enumerated double mixer set callback

## Synopsis

```
int snd_soc_dapm_put_enum_double (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_value * ucontrol);
```

## Arguments

*kcontrol* mixer control

*ucontrol* control element information

## Description

Callback to set the value of a dapm enumerated double mixer control.

Returns 0 for success.

# Name

snd_soc_dapm_info_pin_switch — Info for a pin switch

## Synopsis

```
int snd_soc_dapm_info_pin_switch (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_info * uinfo);
```

## Arguments

*kcontrol* mixer control

*uinfo* control element information

## Description

Callback to provide information about a pin switch control.

# Name

snd_soc_dapm_get_pin_switch — Get information for a pin switch

## Synopsis

```
int snd_soc_dapm_get_pin_switch (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_value * ucontrol);
```

## Arguments

*kcontrol* mixer control

*ucontrol* Value

# Name

snd_soc_dapm_put_pin_switch — Set information for a pin switch

## Synopsis

```
int snd_soc_dapm_put_pin_switch (struct snd_kcontrol * kcontrol, struct
snd_ctl_elem_value * ucontrol);
```

## Arguments

*kcontrol* mixer control

*ucontrol* Value

# Name

snd_soc_dapm_new_controls — create new dapm controls

## Synopsis

```
int snd_soc_dapm_new_controls (struct snd_soc_dapm_context * dapm, const
struct snd_soc_dapm_widget * widget, int num);
```

## Arguments

*dapm* DAPM context

*widget* widget array

*num* number of widgets

## Description

Creates new DAPM controls based upon the templates.

Returns 0 for success else error.

# Name

snd_soc_dapm_enable_pin_unlocked — enable pin.

## Synopsis

```
int snd_soc_dapm_enable_pin_unlocked (struct snd_soc_dapm_context * dapm, const char * pin);
```

## Arguments

*dapm* DAPM context

*pin* pin name

## Description

Enables input/output pin and its parents or children widgets iff there is a valid audio route and active audio stream.

Requires external locking.

## NOTE

`snd_soc_dapm_sync` needs to be called after this for DAPM to do any widget power switching.

# Name

snd_soc_dapm_enable_pin — enable pin.

## Synopsis

```
int snd_soc_dapm_enable_pin (struct snd_soc_dapm_context * dapm, const
char * pin);
```

## Arguments

*dapm* DAPM context

*pin* pin name

## Description

Enables input/output pin and its parents or children widgets iff there is a valid audio route and active audio stream.

## NOTE

`snd_soc_dapm_sync` needs to be called after this for DAPM to do any widget power switching.

# Name

snd_soc_dapm_force_enable_pin_unlocked — force a pin to be enabled

## Synopsis

```
int snd_soc_dapm_force_enable_pin_unlocked (struct snd_soc_dapm_context
* dapm, const char * pin);
```

## Arguments

*dapm* DAPM context

*pin* pin name

## Description

Enables input/output pin regardless of any other state. This is intended for use with microphone bias supplies used in microphone jack detection.

Requires external locking.

## NOTE

`snd_soc_dapm_sync` needs to be called after this for DAPM to do any widget power switching.

# Name

snd_soc_dapm_force_enable_pin — force a pin to be enabled

## Synopsis

```
int snd_soc_dapm_force_enable_pin (struct snd_soc_dapm_context * dapm,
const char * pin);
```

## Arguments

*dapm* DAPM context

*pin* pin name

## Description

Enables input/output pin regardless of any other state. This is intended for use with microphone bias supplies used in microphone jack detection.

## NOTE

`snd_soc_dapm_sync` needs to be called after this for DAPM to do any widget power switching.

# Name

snd_soc_dapm_disable_pin_unlocked — disable pin.

## Synopsis

```
int snd_soc_dapm_disable_pin_unlocked (struct snd_soc_dapm_context *
dapm, const char * pin);
```

## Arguments

*dapm* DAPM context

*pin* pin name

## Description

Disables input/output pin and its parents or children widgets.

Requires external locking.

## NOTE

`snd_soc_dapm_sync` needs to be called after this for DAPM to do any widget power switching.

# Name

snd_soc_dapm_disable_pin — disable pin.

## Synopsis

```
int snd_soc_dapm_disable_pin (struct snd_soc_dapm_context * dapm, const
char * pin);
```

## Arguments

*dapm* DAPM context

*pin* pin name

## Description

Disables input/output pin and its parents or children widgets.

## NOTE

`snd_soc_dapm_sync` needs to be called after this for DAPM to do any widget power switching.

# Name

snd_soc_dapm_nc_pin_unlocked — permanently disable pin.

## Synopsis

```
int snd_soc_dapm_nc_pin_unlocked (struct snd_soc_dapm_context * dapm,
const char * pin);
```

## Arguments

*dapm* DAPM context

*pin* pin name

## Description

Marks the specified pin as being not connected, disabling it along any parent or child widgets. At present this is identical to `snd_soc_dapm_disable_pin` but in future it will be extended to do additional things such as disabling controls which only affect paths through the pin.

Requires external locking.

## NOTE

`snd_soc_dapm_sync` needs to be called after this for DAPM to do any widget power switching.

# Name

snd_soc_dapm_nc_pin — permanently disable pin.

## Synopsis

```
int snd_soc_dapm_nc_pin (struct snd_soc_dapm_context * dapm, const char
* pin);
```

## Arguments

*dapm* DAPM context

*pin* pin name

## Description

Marks the specified pin as being not connected, disabling it along any parent or child widgets. At present this is identical to `snd_soc_dapm_disable_pin` but in future it will be extended to do additional things such as disabling controls which only affect paths through the pin.

## NOTE

`snd_soc_dapm_sync` needs to be called after this for DAPM to do any widget power switching.

# Name

snd_soc_dapm_get_pin_status — get audio pin status

## Synopsis

```
int snd_soc_dapm_get_pin_status (struct snd_soc_dapm_context * dapm,
const char * pin);
```

## Arguments

*dapm* DAPM context

*pin* audio signal pin endpoint (or start point)

## Description

Get audio pin status - connected or disconnected.

Returns 1 for connected otherwise 0.

# Name

snd_soc_dapm_ignore_suspend — ignore suspend status for DAPM endpoint

## Synopsis

```
int snd_soc_dapm_ignore_suspend (struct snd_soc_dapm_context * dapm,
const char * pin);
```

## Arguments

*dapm* DAPM context

*pin* audio signal pin endpoint (or start point)

## Description

Mark the given endpoint or pin as ignoring suspend. When the system is disabled a path between two endpoints flagged as ignoring suspend will not be disabled. The path must already be enabled via normal means at suspend time, it will not be turned on if it was not already enabled.

## Name

snd_soc_dapm_free — free dapm resources

## Synopsis

```
void snd_soc_dapm_free (struct snd_soc_dapm_context * dapm);
```

## Arguments

*dapm* DAPM context

## Description

Free all dapm widgets and resources.

# ASoC DMA Engine API

# Name

snd_dmaengine_pcm_prepare_slave_config — Generic prepare_slave_config callback

## Synopsis

```
int snd_dmaengine_pcm_prepare_slave_config (struct snd_pcm_substream *
substream, struct snd_pcm_hw_params * params, struct dma_slave_config
* slave_config);
```

## Arguments

| | |
|---|---|
| *substream* | PCM substream |
| *params* | hw_params |
| *slave_config* | DMA slave config to prepare |

## Description

This function can be used as a generic prepare_slave_config callback for platforms which make use of the snd_dmaengine_dai_dma_data struct for their DAI DMA data. Internally the function will first call snd_hwparams_to_dma_slave_config to fill in the slave config based on the hw_params, followed by snd_dmaengine_set_config_from_dai_data to fill in the remaining fields based on the DAI DMA data.

# Name

snd_dmaengine_pcm_register — Register a dmaengine based PCM device

## Synopsis

```
int snd_dmaengine_pcm_register (struct device * dev, const struct
snd_dmaengine_pcm_config * config, unsigned int flags);
```

## Arguments

*dev* The parent device for the PCM device

*config* Platform specific PCM configuration

*flags* Platform specific quirks

# Name

snd_dmaengine_pcm_unregister — Removes a dmaengine based PCM device

## Synopsis

```
void snd_dmaengine_pcm_unregister (struct device * dev);
```

## Arguments

*dev* Parent device the PCM was register with

## Description

Removes a dmaengine based PCM device previously registered with snd_dmaengine_pcm_register.

# Chapter 8. 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

enum snd_jack_types — Jack types which can be reported

## Synopsis

```
enum snd_jack_types {
  SND_JACK_HEADPHONE,
  SND_JACK_MICROPHONE,
  SND_JACK_HEADSET,
  SND_JACK_LINEOUT,
  SND_JACK_MECHANICAL,
  SND_JACK_VIDEOOUT,
  SND_JACK_AVOUT,
  SND_JACK_LINEIN,
  SND_JACK_BTN_0,
  SND_JACK_BTN_1,
  SND_JACK_BTN_2,
  SND_JACK_BTN_3,
  SND_JACK_BTN_4,
  SND_JACK_BTN_5
};
```

## Constants

| | |
|---|---|
| SND_JACK_HEADPHONE | Headphone |
| SND_JACK_MICROPHONE | Microphone |
| SND_JACK_HEADSET | Headset |
| SND_JACK_LINEOUT | Line out |
| SND_JACK_MECHANICAL | Mechanical switch |
| SND_JACK_VIDEOOUT | Video out |
| SND_JACK_AVOUT | AV (Audio Video) out |
| SND_JACK_LINEIN | Line in |
| SND_JACK_BTN_0 | Button 0 |
| SND_JACK_BTN_1 | Button 1 |
| SND_JACK_BTN_2 | Button 2 |
| SND_JACK_BTN_3 | Button 3 |
| SND_JACK_BTN_4 | Button 4 |
| SND_JACK_BTN_5 | Button 5 |

## Description

These values are used as a bitmask.

Note that this must be kept in sync with the lookup table in sound/core/jack.c.

# Name

snd_jack_add_new_kctl — Create a new snd_jack_kctl and add it to jack

## Synopsis

```
int snd_jack_add_new_kctl (struct snd_jack * jack, const char * name,
int mask);
```

## Arguments

*jack* the jack instance which the kctl will attaching to

*name* the name for the snd_kcontrol object

*mask* a bitmask of enum snd_jack_type values that can be detected by this snd_jack_kctl object.

## Description

Creates a new snd_kcontrol object and adds it to the jack kctl_list.

## Return

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

# 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, bool initial_kctl, bool phantom_jack);
```

## 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. |
| *initial_kctl* | if true, create a kcontrol and add it to the jack list. |
| *phantom_jack* | Don't create a input device for phantom jacks. |

## 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

*jack* The jack to configure

*type* Jack report type for this key

*keytype* 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

# Name

snd_soc_card_jack_new — Create a new jack

## Synopsis

```
int snd_soc_card_jack_new (struct snd_soc_card * card, const char * id,
int type, struct snd_soc_jack * jack, struct snd_soc_jack_pin * pins,
unsigned int num_pins);
```

## Arguments

| | |
|---|---|
| *card* | ASoC card |
| *id* | an identifying string for this jack |
| *type* | a bitmask of enum snd_jack_type values that can be detected by this jack |
| *jack* | structure to use for the jack |
| *pins* | Array of jack pins to be added to the jack or NULL |
| *num_pins* | Number of elements in the *pins* array |

## Description

Creates a new jack object.

Returns zero if successful, or a negative error code on failure. On success jack will be initialised.

# Name

snd_soc_jack_report — Report the current status for a jack

## Synopsis

```
void snd_soc_jack_report (struct snd_soc_jack * jack, int status, int mask);
```

## Arguments

*jack* the jack

*status* a bitmask of enum snd_jack_type values that are currently detected.

*mask* a bitmask of enum snd_jack_type values that being reported.

## Description

If configured using `snd_soc_jack_add_pins` then the associated DAPM pins will be enabled or disabled as appropriate and DAPM synchronised.

## Note

This function uses mutexes and should be called from a context which can sleep (such as a workqueue).

# Name

snd_soc_jack_add_zones — Associate voltage zones with jack

## Synopsis

```
int snd_soc_jack_add_zones (struct snd_soc_jack * jack, int count,
struct snd_soc_jack_zone * zones);
```

## Arguments

*jack* ASoC jack

*count* Number of zones

*zones* Array of zones

## Description

After this function has been called the zones specified in the array will be associated with the jack.

# Name

snd_soc_jack_get_type — Based on the mic bias value, this function returns the type of jack from the zones declared in the jack type

## Synopsis

```
int snd_soc_jack_get_type (struct snd_soc_jack * jack, int micbias_voltage);
```

## Arguments

| | |
|---|---|
| *jack* | ASoC jack |
| *micbias_voltage* | mic bias voltage at adc channel when jack is plugged in |

## Description

Based on the mic bias value passed, this function helps identify the type of jack from the already declared jack zones

# Name

snd_soc_jack_add_pins — Associate DAPM pins with an ASoC jack

## Synopsis

```
int snd_soc_jack_add_pins (struct snd_soc_jack * jack, int count, struct
snd_soc_jack_pin * pins);
```

## Arguments

*jack* ASoC jack

*count* Number of pins

*pins* Array of pins

## Description

After this function has been called the DAPM pins specified in the pins array will have their status updated to reflect the current state of the jack whenever the jack status is updated.

# Name

snd_soc_jack_notifier_register — Register a notifier for jack status

## Synopsis

```
void snd_soc_jack_notifier_register (struct snd_soc_jack * jack, struct
notifier_block * nb);
```

## Arguments

*jack* ASoC jack

*nb* Notifier block to register

## Description

Register for notification of the current status of the jack. Note that it is not possible to report additional jack events in the callback from the notifier, this is intended to support applications such as enabling electrical detection only when a mechanical detection event has occurred.

# Name

snd_soc_jack_notifier_unregister — Unregister a notifier for jack status

## Synopsis

```
void snd_soc_jack_notifier_unregister (struct snd_soc_jack * jack,
struct notifier_block * nb);
```

## Arguments

*jack* ASoC jack

*nb* Notifier block to unregister

## Description

Stop notifying for status changes.

# Name

snd_soc_jack_add_gpios — Associate GPIO pins with an ASoC jack

## Synopsis

```
int snd_soc_jack_add_gpios (struct snd_soc_jack * jack, int count,
struct snd_soc_jack_gpio * gpios);
```

## Arguments

*jack* ASoC jack

*count* number of pins

*gpios* array of gpio pins

## Description

This function will request gpio, set data direction and request irq for each gpio in the array.

# Name

snd_soc_jack_add_gpiods — Associate GPIO descriptor pins with an ASoC jack

## Synopsis

```
int snd_soc_jack_add_gpiods (struct device * gpiod_dev, struct
snd_soc_jack * jack, int count, struct snd_soc_jack_gpio * gpios);
```

## Arguments

| | |
|---|---|
| *gpiod_dev* | GPIO consumer device |
| *jack* | ASoC jack |
| *count* | number of pins |
| *gpios* | array of gpio pins |

## Description

This function will request gpio, set data direction and request irq for each gpio in the array.

## Name

snd_soc_jack_free_gpios — Release GPIO pins' resources of an ASoC jack

## Synopsis

```
void snd_soc_jack_free_gpios (struct snd_soc_jack * jack, int count,
struct snd_soc_jack_gpio * gpios);
```

## Arguments

*jack* ASoC jack

*count* number of pins

*gpios* array of gpio pins

## Description

Release gpio and irq resources for gpio pins associated with an ASoC 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_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_BUG — give a BUG warning message and stack trace

## Synopsis

```
snd_BUG (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_BUG_ON — debugging check macro

## Synopsis

```
snd_BUG_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.