Audio Manager Library Reference

QNX Software Systems Limited
1001 Farrar Road
Ottawa, Ontario
K2K 0B3
Canada

Voice: +1 613 591-0931
Fax: +1 613 591-3579
Email: info@qnx.com
Web: http://www.qnx.com/

Electronic edition published: Friday, February 7, 2014

Table of Contents

About this Reference

The audio manager is used to control audio stream routing behavior.

To find out about:	See:
What the audio manager does	*Audio Manager Overview* (p. 9)
How to control audio stream routing	*Audio Manager Routing Priority* (p. 11)
Routing priorities for audio stream types	*Audio Stream Routing Priorities by Type* (p. 13)
Linking to the audio manager library	*Linking with Audio Manager* (p. 17)
The audio manger configuration file and how to use it	*Audio Manager Configuration File* (p. 19)
The Audio Manager functions, data structures, constants and enumerated values	*Audio Manager API* (p. 31)

Typographical conventions

Throughout this manual, we use certain typographical conventions to distinguish technical terms. In general, the conventions we use conform to those found in IEEE POSIX publications.

The following table summarizes our conventions:

Reference	Example
Code examples	`if( stream == NULL )`
Command options	`-lR`
Commands	`make`
Environment variables	***PATH***
File and pathnames	`/dev/null`
Function names	*exit()*
Keyboard chords	**Ctrl –Alt –Delete**
Keyboard input	`Username`
Keyboard keys	`Enter`
Program output	`login:`
Variable names	*stdin*
Parameters	*parm1*
User-interface components	**Navigator**
Window title	**Options**

We use an arrow in directions for accessing menu items, like this:

You'll find the Other... menu item under **Perspective → Show View** .

We use notes, cautions, and warnings to highlight important messages:

Notes point out something important or useful.

Cautions tell you about commands or procedures that may have unwanted or undesirable side effects.

Warnings tell you about commands or procedures that could be dangerous to your files, your hardware, or even yourself.

Note to Windows users

In our documentation, we use a forward slash (/) as a delimiter in all pathnames, including those pointing to Windows files. We also generally follow POSIX/UNIX filesystem conventions.

Technical support

Technical assistance is available for all supported products.

To obtain technical support for any QNX product, visit the Support area on our website (*www.qnx.com*). You'll find a wide range of support options, including community forums.

Chapter 1
Audio Manager Overview

The audio manager is used to control audio stream routing, volume, and ducking behavior.

About the audio manager

The audio manager and its library of functions and data structures allows client applications to set audio properties such as routing, volume, and concurrency, and to receive events notifying them of changes to these properties for audio streams for other applications.

The audio manager provides:

- automatic routing, and manual routing of the PCM *preferred* path
- audio stream type identification
- audio concurrency policy (ducking) management
- audio device monitoring and mounting (e.g., headset, A2DP, HDMI)

Applications should notify the audio manager of the types of audio streams that are opening or closing, and can link specific PCM streams with these types in order to support advanced concurrency use cases (e.g., using alerts to automatically trigger audio ducking in the media player). The application can also use the audio manager to override the default preferred routing table for an audio stream type, thus allowing the user to choose the output device (e.g., the loudspeaker rather than the wired headset).

Using the audio manager is optional. However, applications that require close integration with the audio system will need to use this service. A client application that doesn't use the audio manager will be managed as a “default” type audio stream, and the client application will have little control over this default stream.

Using the audio manager

To play audio, an application needs to open a PCM stream. After opening the PCM stream and getting a handle for the stream, the application should get an audio manager handle with the same audio type as the PCM stream. This can be done by calling *audio_manager_snd_pcm_open()* (p. 116), then *audio_manager_get_handle()* (p. 108); or by calling *audio_manager_snd_pcm_open_name()* (p. 118).

Audio types are defined in *audio_manager_audio_type_t* (p. 101).

Chapter 2
Audio Manager Routing Priority

The audio manager allows client applications to control the routing of their audio streams.

The main audio device attached to the system supports only a single *main mix*. This means that, unless client applications specify audio routing paths, the system automatically selects for output the audio stream from the connected device with the highest priority in the current mode of operation.

Every type of audio stream has a priority relative to other types of audio stream types. In the event that two or more audio streams require concurrent access to an output device, the system grants access to the output device to the stream with the highest priority type. For example, if AUDIO_TYPE_DEFAULT is the type of the active stream, and a second stream of type AUDIO_TYPE_VIDEO_CHAT begins, the routing, tuning, and attenuation policies of the AUDIO_TYPE_VIDEO_CHAT stream take effect, since this stream has the higher priority.

The table below lists the audio stream types, in descending order of priority (that is, the higher the audio type is in the table, the higher its priority), as well as the effect on each stream type if it is pre-empted. The pre-empted stream may be either *muted* or *fully attenuated*:

- *muted* — lower priority streams are muted for a non-transient event, such as a phone call. It might be appropriate for applications generating lower priority audio streams to pause for the duration of the event.
- *fully attenuated* — lower priority streams are muted for a transient event, such as a camera click. Other applications might not need to react in this case.

Name	Description	Concurrency Rule
AUDIO_TYPE_SOUND_EFFECT	Sound effects that can never be attenuated, such as the camera click.	Fully attenuates all lower priority audio streams.
AUDIO_TYPE_RINGTONE	Used for playback of ringtones when an incoming phone call occurs.	Fully attenuates all lower priority audio streams.
AUIDO_TYPE_VOICE_TONES	DTMF and call progress tones. Can also be used to playback non-tone-based audio during phone call however.	Fully attenuates all lower priority audio streams.

Name	Description	Concurrency Rule
AUDIO_TYPE_VOICE	Voice band related streams, and specific activities related to telephony (cellular or VOIP).	Mutes all lower priority audio streams.
AUDIO_TYPE_VIDEO_CHAT	Used by the video chat client. This type is not covered by AUDIO_TYPE_VOICE because of a difference in automatic routing policy.	Mutes all lower priority audio streams.
AUDIO_TYPE_PUSH_TO_TALK	Used to denote streams related to push-to-talk use cases.	Fully attenuates all lower priority audio streams.
AUDIO_TYPE_VOICE_RECOGNITION	Voice recognition services, such as VAD.	Mutes all lower priority audio streams.
AUDIO_TYPE_TEXT_TO_SPEECH	Text to speech services.	Attenuates lower priority audio streams by 40%.
AUDIO_TYPE_ALERT	Notifiers, such as for calendar events, email, SMS, instant messaging, and so on.	Attenuates lower priority audio streams by 40%.
AUDIO_TYPE_VOICE_RECORDING	Voice recording services, such as focal points.	No change to other streams.
AUDIO_TYPE_MULTIMEDIA	Used by media player applications.	No change to other streams.
AUDIO_TYPE_DEFAULT	Any unclassified audio stream is of type default.	No change to other streams.

Chapter 3
Audio Stream Routing Priorities by Type

An audio stream (sound effect, ringtone, etc.) is routed to an output device based on the priority ranking of the device for the audio stream type.

The following table lists audio stream types and their routing priorities. For each audio stream type, the output devices to which the stream can be routed are listed in descending order of priority. That is, for AUDIO_TYPE_PUSH_TO_TALK, the preferred output device is AUDIO_DEVICE_HEADSET; if this device is not available, then the stream will be routed to AUDIO_DEVICE_HEADPHONE, and so on.

Audio type	Output device routing
AUDIO_TYPE_SOUND_EFFECT	AUDIO_DEVICE_SPEAKER
AUDIO_TYPE_RINGTONE	AUDIO_DEVICE_SPEAKER
AUDIO_TYPE_VOICE	AUDIO_DEVICE_TTY
	AUDIO_DEVICE_HEADSET
	AUIDO_DEVICE_HEADPHONE
	AUDIO_DEVICE_LINEOUT
	AUDIO_DEVICE_BT_SCO
	AUDIO_DEVICE_HAC
	AUDIO_DEVICE_HANDSET
	AUDIO_DEVICE_SPEAKER
AUDIO_TYPE_VIDEO_CHAT	AUDIO_DEVICE_HEADSET
	AUDIO_DEVICE_HEADPHONE
	AUDIO_DEVICE_LINEOUT
	AUDIO_DEVICE_BT_SCO
	AUDIO_DEVICE_HDMI
	AUDIO_DEVICE_TOSLINK
	AUDIO_DEVICE_SPEAKER
	AUDIO_DEVICE_HANDSET
AUDIO_TYPE_PUSH_TO_TALK	AUDIO_DEVICE_HEADSET

Audio type	Output device routing
	AUDIO_DEVICE_HEADPHONE
	AUDIO_DEVICE_LINEOUT
	AUDIO_DEVICE_BT_SCO
	AUDIO_DEVICE_SPEAKER
	AUDIO_DEVICE_HANDSET
AUDIO_TYPE_VOICE_RECOGNITION	AUDIO_DEVICE_HEADSET
	AUDIO_DEVICE_BT_SCO
	AUDIO_DEVICE_HEADPHONE
	AUDIO_DEVICE_LINEOUT
	AUDIO_DEVICE_SPEAKER
AUDIO_TYPE_TEXT_TO_SPEECH	AUDIO_DEVICE_HEADSET
	AUDIO_DEVICE_HEADPHONE
	AUDIO_DEVICE_LINEOUT
	AUDIO_DEVICE_A2DP
	AUDIO_DEVICE_HDMI
	AUDIO_DEVICE_TOSLINK
	AUDIO_DEVICE_SPEAKER
	AUDIO_DEVICE_HANDSET
AUDIO_TYPE_VOICE_RECORDING	AUDIO_DEVICE_HEADSET
	AUDIO_DEVICE_BT_SCO
	AUDIO_DEVICE_HANDSET
	AUDIO_DEVICE_HEADPHONE
	AUDIO_DEVICE_SPEAKER
AUDIO_TYPE_MULTIMEDIA	AUDIO_DEVICE_HEADSET
AUDIO_TYPE_DEFAULT	AUDIO_DEVICE_HEADPHONE
	AUDIO_DEVICE_LINEOUT
	AUDIO_DEVICE_A2DP
	AUDIO_DEVICE_HDMI

Audio type	Output device routing
	AUDIO_DEVICE_TOSLINK
	AUDIO_DEVICE_SPEAKER
	AUDIO_DEVICE_HANDSET
AUDIO_TYPE_VOICE_TONES	AUDIO_DEVICE_HEADSET
	AUIDO_DEVICE_HEADPHONE
	AUDIO_DEVICE_LINEOUT
	AUDIO_DEVICE_SPEAKER
	AUDIO_DEVICE_HANDSET
AUDIO_TYPE_ALERT	AUDIO_DEVICE_SPEAKER
	AUDIO_DEVICE_HEADSET
	AUDIO_DEVICE_HEADPHONE
	AUDIO_DEVICE_LINEOUT
	AUDIO_DEVICE_A2DP
	AUDIO_DEVICE_HANDSET

Chapter 4
Linking with Audio Manager

Projects that use audio manager need to link against the library audio_manager (or libaudio_manager.so for the object name) to resolve the symbolic links.

If you use a makefile project, you can link with audio manager by adding the following code to the file `common.makefile`:

```
#  Add library dependency
LIBS+=audio_manager

#  Statically link libaudio_manager for debug builds
ifneq( $(origin DEBUG),undefined)
LIBPREF_audio_manager = -Bstatic
LIBPOST_audio_manager = -Bdynamic
endif
```

Chapter 5
Audio Manager Configuration File

The audio manager configuration file sets the behavior of audio devices and streams.

About the configuration file

The audio manager configuration file (`/audioman_default.conf`) sets the behavior of audio devices and streams, including device and audio type attributes, ducking rules, and audio routing rules and priorities.

You can change the configuration file to adjust how your system handles audio streams and the behavior of output devices. For example, you can set the default volume for music (as a percentage of the maximum) at system start for each type of output device (speaker, headset, headphones, etc.), or the amount of attenuation when the phone rings. Thus, by using different configuration files you can tune audio behavior for specific devices or implementations (phone, in-vehicle system, etc.) without changing or recompiling code.

Editing the configuration file

The configuration file can't be changed by applications or the audio management service. To change values in the configuration file:

1. Stop the audio manager.
2. Edit the file.
3. Delete the `/pps/services/audio/*` PPS objects.
4. Restart the service so that it loads the new configuration into the PPS objects.

Organization and syntax

The audio manager configuration file is a plain text file that uses a simple syntax with hierarchical containers of values for attributes.

The syntax for a container is as follows:

```
"value":{
    "attribute":"value",
}
```

For example, the following configures two channels for headphones:

```
"headphone":{
      "numchans":"2",
    }
```

And the following configures the volume for an alert to 90 percent of the maximum for the output device:

```
"Ducking Rules":{
    "alert":"90",
  },
```

Note that the value in a configuration can be a pair of brackets, "{ ...}" and their contents, in which case the contents inside the brackets are the value for the attribute. For example, the file begins by defining the language used for managing audio, the file version number. It then begins configuring devices, as shown in the code snippet below.

```
AudioManagerConfiguration:json:{
  "Version":"0.0.1",
  "Devices":{
    "Description":"Specify the default values for the audio device attribut
    "speaker":{
      "supported":"true",
      ...
      }
   }
}
```

Loading the audio management configuration

At startup, the audio manager looks for the `audioman_default.conf` file at `/etc/system/config/audio/`. If it doesn't find this file, it uses its compiled values. If the audio manager finds a configuration file, it writes the values in the file to its PPS objects at `/pps/services/audio/`. If any configurations are missing from the configuration file, the audio manager uses the compiled values for these configurations.

Configuration details

The following snippet from an `audioman_default.conf` file shows a configuration for a headset.

```
"headset":{
      "supported":"true",
      "path":"/dev/snd/pcmPreferredp",
      "input.path":"/dev/snd/pcmPreferredc",
      "connected":"false",
      "numchans":"2",
      "order":"FL,FR",
      "audioconfig":"2.0",
      "inchans":"1",
      "hwinchans":"1",
      "volumecontrol":"percentage",
      "dependency":"",
      "keepalive":"false",
      "suspended":"false",
      "controlpps":"",
      "audioprocessing":"false",
      "vadsupport":"",
      "public":"false",
```

```
            "mutable":"true",
            "mixer":"Master",
            "input.mixer":"Input Gain",
            "Buttons":"1"
        },
```

Device attributes

The configuration file can be used to configure device attributes.

Valid device types are defined in *Definitions in audio_manager_device.h* (p. 38).

You can configure the following device attributes:

Attribute	Description
audioconfig	Configuration of the audio output channels (e.g., "2.0", "5.1"). Note that this is used only by the `hdmi` device.
audioprocessing	Indicates whether the device can do some audio processing that the system won't need to handle (e.g., a headset may be able to do noise cancellation): `true` or `false`.
BUTTONS	The number of buttons the device supports. If not present, the default `0` is assumed.
connected	This type of device is connected to the system: `true` or `false`.
controlpps	The path of the device's PPS control object. If this object exists, then the device is controlled by a PPS interface instead of an actual audio driver.
dependency	Indicates whether this device depends on another device (this device has no effect unless the other is also connected).
hwinchans	Total number of input channels on the hardware.
inchans	Default number of channels that the client should use for multimedia audio capture. For example, for a device with four microphones, the client might use two for multimedia, in which case *inchans* would be 2 (*hwinchans* would be 4).
input.mixer	Name of the mixer group implemented by the input device for volume control. Values depend on the particular audio drivers and

Attribute	Description
	on the Audio Manager configuration. Default names are: • `BT A2DP In` • `BT SCO In` • `HDMI In` • `Input Gain` • `Tones In` • `TosLink In` • `USB In` • `Voice In` • `WIFI In` For details about audio drivers, see the `io-audio` manager and the `deva-*` entries in the QNX Neutrino *Utilities Reference*.
input.path	Path for the audio stream input; for example `/dev/snd/pcmPreferredc`.
keepalive	Indicates whether the output device is to be kept alive when no audio stream is active: `true` or `false`.
mixer	Name of the mixer group implemented by the output device for volume control. As for *input.mixer*, values depend on the particular audio drivers and on the Audio Manager configuration. Default names are: • `BT A2DP Out` • `BT SCO Out` • `HDMI Out` • `Output GaOut` • `Tones Out` • `TosLOutk Out` • `USB Out` • `Voice Out` • `WIFI Out` For details about audio drivers, see the `io-audio` manager and the `deva-*`

Attribute	Description
	entries in the QNX Neutrino *Utilities Reference*.
mutable	The output device can be muted: `true` or `false`.
numchans	Number of audio channels supported.
order	The first audio channel; see list of audio channels definitions in *Definitions in audio_manager_device.h* (p. 38).
path	Path for the audio stream output; for example, `/dev/snd/pcmPreferredp`.
public	Indicates whether the device can be heard publicly (e.g., the value for a speaker would be `true`).
supported	Set to `true` if this device type is supported: `true` or `false`.
suspended	Indicates whether the device is temporarily disabled by the system.: `true` or `false`.
vadsupport	Voice Activity Detection (VAD) is supported: `true` or `false`.
volumecontrol	The type of volume control; see `AUDIO_VOLUME_CONTROL_NAMES` in *Definitions in audio_manager_device.h* (p. 38).

Audio status list attributes

The configuration file can be used to specify default values for the attributes of the audio status list.

The “Status” configuration value contains configuration details that value specify the default values for the audio status list attributes. It sets values for *Attributes* (p. 25) and *Devices* (p. 26).

Attributes

The “Attributes” value defines default values of audio streams.

Attribute	Description
audio.mode	The type of stream. Supported values include “audio”, “record”, “video”, and “voice”.
input.gain	Gain to apply to input, as percentage of maximum possible.
input.muted	Default state for the input stream: “true” (muted) or “false” (not muted).
hpoverride.supported	Audio-boost override is or is not supported: “true” (supported) or “false” (not supported).
hpoverride	Audio-boost override is on: “true” (on) or “false” (off).
hpboostlevel	The unsafe volume level for headphones, as percentage of maximum possible volume. The unsafe volume range for headphones. How is this expressed?
output.available	Available output device; must be a device type defined in *Definitions in audio_manager_device.h* (p. 38).
hpoutput.regulated	Set to “true” if the headphone output volume is limited by regulations (e.g., the setting may be 100% by an app, but the regulation limits the volume to 90%).

Attribute	Description
hpoutput.unregulatedlevel	The volume level, as a percantage of maximum possible, when output is not regulated.

Devices

The “Devices” value defines default input and output behaviors, by device type. For a list of device types, see *Definitions in audio_manager_device.h* (p. 38).

Attribute	Description
output.speaker.volume	The deafult volume, measured as defined by the the device type “volumecontrol” attribute. For example, if the “volumecontrol” attribute value is “percentage”, then a value of “75” sets the deafult output speak volume to 75 percent of maximum.
output.speaker.muted	Default state for the output speaker “true” (muted) or “false” (not muted).
input.speaker.gain	The amount of gain applied to the input speaker, as a percentage of the maximum possible gain.
input.speaker.muted	Default state for the input speaker “true” (muted) or “false” (not muted).

Routing rules

The configuration file can be used to specify routing behavior for the different types of audio streams.

The “Routing Rules” configuration value specifies the routing of audio streams, in order of priority. It contains the audio stream types listed in priority. Supported audio stream types, in alphabetical order, are: “alert”, “default”, “inputfeedback”, “multimedia”, “pushtotalk”, “ringtone”, “ soundeffect”, “ texttospeech”, “ videochat”, “ voice”, “ voicerecognition”, “voicerecording”, and “ voicetones”.

Under each audio stream type, “Devices” attribute specify pairs of output and input devices, in order of priority for the audio stream type:

Attribute	Description
Devices	The input and output devices where this audio stream type is routed.
Output	The output route for the audio stream.
Input	The input route for the audio stream.

For a list of routing paths and priorities by audio stream type, see *Supported routing paths* (p. 13). For a list of device types, see *Definitions in audio_manager_device.h* (p. 38).

Example

The codes snippet below taken from the “Routing Rules” section of a configuration file shows that for this implementation:

- A ringtone has a higher priority than a videochat (it appears earlier in the list of audio stream types).
- Output for a ringtone will be routed to the speaker.
- Output for the videochat will be routed to the headset, and if no headset is available to the headphones, etc.
- The system will look for ringtones coming in, first through the USB connection, then from the handset.
- The system will look for videchat input, first from the headset, then from a USB connection.

```
"ringtone":{
      "Devices":{
        "Output":"speaker",
        "Input":"usb"
      },
      "Devices":{
        "Output":"speaker",
```

```
            "Input":"handset"
          },
          ...
    "videochat":{
          "Devices":{
            "Output":"headset",
            "Input":"headset"
          },
          "Devices":{
            "Output":"headphone",
            "Input":"usb"
          },
          ...
```

Ducking rules

The configuration file can be used to specify ducking behavior for the different types of audio streams.

The “Ducking Rules” configuration value specifies the behavior of audio streams when they are affected by other streams. The configuration file lists the audio stream types in order of priority, from highest to lowest, and the behavior of each audio stream type when it is pre-empted by a higher-priority stream.

Every audio stream can be set to one the following:

- a value from 1 to 100 specifying the volume level to which the stream with be set when it is interrupted; this value is a percentage of the current volume.
- “mute” (-1). The stream will mute if interrupted by a higher-priority stream.
- “no effect” (INT_MIN). The stream will not be affected by any other stream.

Example

The code snippet below shows an example of ducking rules. Note that these rules are not relevent for some audio streams, such as voice recordings.

```
"Devices":{
    "Description":"Specify the default values of the attributes of the audi
    "speaker":{
      "supported":"true",
      "path":"/dev/snd/pcmPreferredp",
      "input.path":"/dev/snd/pcmPreferredc",
      "connected":"true",
      "numchans":"1",
      "order":"FL",
      "audioconfig":"2.0",
      "inchans":"0",
      "hwinchans":"0",
      "volumecontrol":"percentage",
      "dependency":"",
      "keepalive":"false",
      "suspended":"false",
      "controlpps":"",
      "audioprocessing":"false",
      "vadsupport":"",
      "public":"true",
      "mutable":"true",
      "mixer":"Master",
      "input.mixer":"Input Gain"
    },
    ...
```

Chapter 6
Audio Manager API

The audio manager API includes functions and data structures for defining and managing concurrency policies and device properties, and for controlling audio routing, concurrency, and volume.

audio_manager_concurrency.h

Definitions for supported audio concurrency policies.

The audio manager maintains the audio concurrency policies for the supported audio types. This file defines concurrency properties and provides functions for them.

Data types in *audio_manager_concurrency.h*

Data structures, typedefs, and enumerations for managing audio concurrency.

audio_manager_attenuation_extra_option_t

Supported extra attenuation options.

Synopsis:

```
#include <audio/audio_manager_concurrency.h>


typedef enum {
      ATTENUATION_VOICE_UPLINK = (1u<<0)
      ATTENUATION_VOICE_DOWNLINK = (1u<<1)
} audio_manager_attenuation_extra_option_t;
```

Data:

ATTENUATION_VOICE_UPLINK

Uplink attenuation option.

ATTENUATION_VOICE_DOWNLINK

Downlink attenuation option.

Library:

`libaudio_manager`

Description:

This enumeration defines the supported extra attenuation options.

audio_manager_attenuation_params_t

Supported audio attenuation parameters.

Synopsis:

```
typedef struct  {
    int attenuation ;
    int attenuation_extra_options ;
}audio_manager_attenuation_params_t;
```

Data:

int attenuation

Attenuation value from 0-100 and enum audio_manager_attenuation_type_t.

int attenuation_extra_options

Extra attenuation options, represented by a bitmask of audio_manager_attenuation_extra_option_t.

Library:

`libaudio_manager`

Description:

This structure defines the parameters for setting the attenuation effect of an audio manager handle.

audio_manager_attenuation_type_t

Supported audio attenuation types.

Synopsis:

```
#include <audio/audio_manager_concurrency.h>


typedef enum {
      AUDIO_ATTENUATION_MUTE = -1
      AUDIO_ATTENUATION_NO_EFFECT = 100
      AUDIO_ATTENUATION_DEFAULT = INT_MAX
} audio_manager_attenuation_type_t;
```

Data:

AUDIO_ATTENUATION_MUTE

Mute attenuation type.

AUDIO_ATTENUATION_NO_EFFECT

No effect attenuation type.

AUDIO_ATTENUATION_DEFAULT

Default attenuation type.

Library:

`libaudio_manager`

Description:

This enumeration defines the supported audio attenuation types.

audio_manager_concurrency_t

Audio concurrency settings.

Synopsis:

```
typedef struct  {
    bool attenuated ;
    bool muted ;
    audio_manager_audio_type_t muted_by ;
    pid_t muted_by_pid ;
}audio_manager_concurrency_t;
```

Data:

bool attenuated

true if the audio type is currently being attenuated, false otherwise.

bool muted

true if the audio type is currently being fully muted, false otherwise.

audio_manager_audio_type_t muted_by

The audio type causing the mute policy to be applied.

pid_t muted_by_pid

The ID of the process causing the mute policy to be applied.

Library:

`libaudio_manager`

Description:

This structure defines the audio concurrency settings.

Functions in *audio_manager_concurrency.h*

Functions for managing audio concurrency.

audio_manager_get_audio_type_concurrency_status()

Get the audio concurrency status of a given audio type.

Synopsis:

```
#include <audio/audio_manager_concurrency.h>

int audio_manager_get_audio_type_concurrency_status(audio_manager_audio_type_t
 type, audio_manager_concurrency_t *status)
```

Arguments:

type

The audio type to query.

status

The audio concurrency status.

Library:

`libaudio_manager`

Description:

The audio_manager_get_audio_type_concurrency_status() function returns the audio concurrency status audio_manager_concurrency_t of a given audio type.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_current_audio_handle_concurrency_status()

Get the audio concurrency status of a given audio manager handle.

Synopsis:

```
#include <audio/audio_manager_concurrency.h>

int audio_manager_get_current_audio_handle_concurrency_status(unsigned int
audioman_handle, audio_manager_concurrency_t *status)
```

Arguments:

audioman_handle

The audio mananger handle to query.

status

The audio concurrency status.

Library:

`libaudio_manager`

Description:

The audio_manager_get_current_audio_handle_concurrency_status() function returns the current audio concurrency status audio_manager_concurrency_t of a given audio manager handle.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_set_handle_attenuation()

Override the attenuation that's associated with the type of the given audio manager handle.

Synopsis:

```
#include <audio/audio_manager_concurrency.h>

int audio_manager_set_handle_attenuation(unsigned int audioman_handle,
audio_manager_attenuation_params_t params)
```

Arguments:

audioman_handle

The audio manager handle returned by `audio_mananger_get_handle`.

params

The parameters for the handle attenuation to apply.

Library:

`libaudio_manager`

Description:

The audio_manager_set_handle_attenuation() function overrides the attenuation of the given audio manager handle applied to the lower ducking priority audio sources.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_device.h

Definitions for supported audio devices and their properties.

The audio manager maintains a list of supported devices on the target and their properties. This file defines device properties and provides get and set functions for them.

Constants in *audio_manager_device.h*

Constants for managing audio devices.

Definitions in *audio_manager_device.h*

Preprocessor macro definitions for the audio_manager_device.h header file in the libaudio_manager library.

Definitions:

```
#define AUDIO_DEVICE_NAMES {                        \
    "speaker",              \
    "headset",              \
    "headphone",            \
    "a2dp",                 \
    "handset",              \
    "hac",                  \
    "btsco",                \
    "hdmi",                 \
    "toslink",              \
    "tty",                  \
    "lineout",              \
    "usb",                  \
    "tones",                \
    "voice",                \
    "miracast",             \
    "mirrorlink",           \
    "audioshare",           \
}
```

Names for supported audio devices.

```
#define AUDIO_CHANNEL_NAMES {                        \
    "",                     \
    "FL",                   \
    "FC",                   \
    "FR",                   \
    "RL",                   \
    "RR",                   \
    "LFE",                  \
}
```

Names for audio channels.

```
#define AUDIO_VOLUME_CONTROL_NAMES {                                \
    "unavailable",                  \
    "simple",                       \
    "percentage",                   \
}
```

Names for volume control types.

```
#define AUDIO_HEADSET_BUTTON_NAMES {                                \
    "button_play_pause",            \
```

```
    "button_vol_up",                \
    "button_vol_down",              \
    "button_forward",               \
    "button_back",                  \
}
```

Names for headset buttons.

```
#define AUDIO_HEADSET_BUTTON_STATE_NAMES {
        \
    "pressed",                                  \
    "released",                                 \
}
```

Names for headset button states.

Library:

`libaudio_manager`

Data types in *audio_manager_device.h*

Data structures, typedefs, and enumerations for managing audio devices.

audio_manager_channel_t

Supported audio channels.

Synopsis:

```
#include <audio/audio_manager_device.h>


typedef enum {
      AUDIO_CHANNEL_UNAVAILABLE
      AUDIO_CHANNEL_FRONT_LEFT
      AUDIO_CHANNEL_FRONT_CENTER
      AUDIO_CHANNEL_FRONT_RIGHT
      AUDIO_CHANNEL_REAR_LEFT
      AUDIO_CHANNEL_REAR_RIGHT
      AUDIO_CHANNEL_LOW_FREQ_EFFECTS
      AUDIO_CHANNEL_COUNT
} audio_manager_channel_t;
```

Data:

AUDIO_CHANNEL_UNAVAILABLE

No output channel is supported.

AUDIO_CHANNEL_FRONT_LEFT

Front left channel.

AUDIO_CHANNEL_FRONT_CENTER

Front center channel.

AUDIO_CHANNEL_FRONT_RIGHT

Front right channel.

AUDIO_CHANNEL_REAR_LEFT

Rear left channel.

AUDIO_CHANNEL_REAR_RIGHT

Rear right channel.

AUDIO_CHANNEL_LOW_FREQ_EFFECTS

Low frequency effects channel (subwoofer).

AUDIO_CHANNEL_COUNT

The total number of audio channels supported.

Library:

`libaudio_manager`

Description:

This enumeration defines the supported types of audio channels (2.0/5.1).

audio_manager_channel_config_t

Supported audio device configurations.

Synopsis:

```
#include <audio/audio_manager_device.h>


typedef enum {
      AUDIO_CHANNEL_CONFIG_STEREO = (1u<<0)
      AUDIO_CHANNEL_CONFIG_5_1 = (1u<<1)
} audio_manager_channel_config_t;
```

Data:

AUDIO_CHANNEL_CONFIG_STEREO

The audio device supports stereo channels.

AUDIO_CHANNEL_CONFIG_5_1

The audio device supports 5.1 surround sound.

Library:

`libaudio_manager`

Description:

This enumeration defines bit masks for supported audio device configurations.

audio_manager_device_t

Supported audio devices.

Synopsis:

```
#include <audio/audio_manager_device.h>


typedef enum {
      AUDIO_DEVICE_SPEAKER
      AUDIO_DEVICE_HEADSET
      AUDIO_DEVICE_HEADPHONE
      AUDIO_DEVICE_A2DP
      AUDIO_DEVICE_HANDSET
      AUDIO_DEVICE_HAC
      AUDIO_DEVICE_BT_SCO
      AUDIO_DEVICE_HDMI
      AUDIO_DEVICE_TOSLINK
      AUDIO_DEVICE_TTY
      AUDIO_DEVICE_LINEOUT
      AUDIO_DEVICE_USB
      AUDIO_DEVICE_TONES
      AUDIO_DEVICE_VOICE
      AUDIO_DEVICE_WIFI_DISPLAY
      AUDIO_DEVICE_MIRRORLINK
      AUDIO_DEVICE_AUDIO_SHARE
      AUDIO_DEVICE_COUNT
      AUDIO_DEVICE_DEFAULT = 0xFF
} audio_manager_device_t;
```

Data:

AUDIO_DEVICE_SPEAKER

The main speaker(s).

AUDIO_DEVICE_HEADSET

The mono/stereo headset with microphone.

AUDIO_DEVICE_HEADPHONE

The mono/stereo output only headphone.

AUDIO_DEVICE_A2DP

The Bluetooth A2DP streaming service.

AUDIO_DEVICE_HANDSET

The phone receiver.

AUDIO_DEVICE_HAC

The hearing aid compatibility device.

AUDIO_DEVICE_BT_SCO

The Bluetooth hands-free profile service for voice calls.

AUDIO_DEVICE_HDMI

The HDMI connection.

AUDIO_DEVICE_TOSLINK

The TOSLINK connection.

AUDIO_DEVICE_TTY

The telecommunications device for the hearing challenged.

AUDIO_DEVICE_LINEOUT

The lineout connection through the headset jack.

AUDIO_DEVICE_USB

The USB connection.

AUDIO_DEVICE_TONES

The virtual tones port that is used for system tones.

AUDIO_DEVICE_VOICE

The virtual voice port that is used for voice stream processing such as VoIP.

AUDIO_DEVICE_WIFI_DISPLAY

The audio device exposed by the Wi-Fi display connection.

AUDIO_DEVICE_MIRRORLINK

The audio device exposed by the MirrorLink connection.

AUDIO_DEVICE_AUDIO_SHARE

The virtual audio device exposed by video share.

AUDIO_DEVICE_COUNT

The total number of devices supported.

AUDIO_DEVICE_DEFAULT

The current active output device that audio is playing out of.

Library:

`libaudio_manager`

Description:

This enumeration defines the types of audio devices supported.

audio_manager_device_audio_config_t

Audio configuration settings for a given audio device.

Synopsis:

```
typedef struct  {
    int num_out_channels ;
    int num_in_channels ;
    audio_manager_channel_t channel_order [AUDIO_CHANNEL_COUNT];
    audio_manager_channel_config_t channel_config_mask ;
}audio_manager_device_audio_config_t;
```

Data:

int num_out_channels

The number of output channels supported.

int num_in_channels

The number of input channels supported.

audio_manager_channel_t channel_order[AUDIO_CHANNEL_COUNT]

The output channels of the audio device, listed in setup order.

audio_manager_channel_config_t channel_config_mask

The audio device configuration of the audio device.

Library:

`libaudio_manager`

Description:

This structure defines the audio configuration of a given audio device.

audio_manager_device_capabilities_t

Audio capabilities that can be queried.

Synopsis:

```
#include <audio/audio_manager_device.h>


typedef enum {
      AUDIO_DEVICE_PROPERTY_NUM_OUT_CHANNELS
      AUDIO_DEVICE_PROPERTY_NUM_IN_CHANNELS
      AUDIO_DEVICE_PROPERTY_CHANNEL_ORDER
      AUDIO_DEVICE_PROPERTY_CHANNEL_CONFIG
      AUDIO_DEVICE_PROPERTY_SUPPORTED
      AUDIO_DEVICE_PROPERTY_CONNECTED
      AUDIO_DEVICE_PROPERTY_SUSPENDED
      AUDIO_DEVICE_PROPERTY_VOLUME_CONTROL
      AUDIO_DEVICE_PROPERTY_PUBLIC
      AUDIO_DEVICE_PROPERTY_NUM_HW_IN_CHANNELS
      AUDIO_DEVICE_PROPERTY_COUNT
} audio_manager_device_capabilities_t;
```

Data:

AUDIO_DEVICE_PROPERTY_NUM_OUT_CHANNELS

The number of output channels supported; of type int.

AUDIO_DEVICE_PROPERTY_NUM_IN_CHANNELS

The number of input channels supported; of type int.

AUDIO_DEVICE_PROPERTY_CHANNEL_ORDER

The output channels, listed in setup order; of type audio_manager_channel_t[].

This parameter is only permitted if AUDIO_DEVICE_PROPERTY_NUM_OUT_CHANNELS is requested as well. If the number of channels is greater than the input value for AUDIO_DEVICE_PROPERTY_NUM_OUT_CHANNELS then, at most, the input value will be written.

AUDIO_DEVICE_PROPERTY_CHANNEL_CONFIG

The audio device configuration of the audio device; of type audio_manager_channel_config_t.

The value may be multiple values from this enum OR'ed together.

AUDIO_DEVICE_PROPERTY_SUPPORTED

Whether the device is supported on this platform; of type bool.

AUDIO_DEVICE_PROPERTY_CONNECTED

Whether the device is currently connected; of type bool.

For example, in the case of headphones, indicates whether the headphones are plugged in.

AUDIO_DEVICE_PROPERTY_SUSPENDED

Whether the device is suspended; of type bool.

AUDIO_DEVICE_PROPERTY_VOLUME_CONTROL

The type of volume control supported by the device; of type audio_manager_device_volume_control_t.

AUDIO_DEVICE_PROPERTY_PUBLIC

Whether the device is public; of type bool.

This means a device may be listened to by multiple listeners, such as a speaker, as opposed to a private device listened to by one person, such as headphones.

AUDIO_DEVICE_PROPERTY_NUM_HW_IN_CHANNELS

The number of input channels supported by the device.

This enumerator represents the true number of input channels that a device supports and therefore differs from the `AUDIO_DEVICE_PROPERTY_NUM_IN_CHANNELS` enumerator.

AUDIO_DEVICE_PROPERTY_COUNT

The total number of device properties supported.

Library:

`libaudio_manager`

Description:

This enumeration defines the audio capabilities that can be queried.

audio_manager_device_capability_t

Supported audio device I/O.

Synopsis:

```
#include <audio/audio_manager_device.h>


typedef enum {
      AUDIO_OUTPUT = (1u<<0)
      AUDIO_INPUT = (1u<<1)
} audio_manager_device_capability_t;
```

Data:

AUDIO_OUTPUT

The audio device supports output (playback).

AUDIO_INPUT

The audio device supports input (recording).

Library:

`libaudio_manager`

Description:

This enumeration defines bit masks for the supported audio device I/O.

audio_manager_device_headset_button_t

Supported audio headset button types.

Synopsis:

```
#include <audio/audio_manager_device.h>


typedef enum {
      AUDIO_HEADSET_PLAY_PAUSE
      AUDIO_HEADSET_VOLUME_UP
      AUDIO_HEADSET_VOLUME_DOWN
      AUDIO_HEADSET_FORWARD
      AUDIO_HEADSET_BACKWARD
      AUDIO_HEADSET_BUTTON_COUNT
} audio_manager_device_headset_button_t;
```

Data:

AUDIO_HEADSET_PLAY_PAUSE

The play/pause button or the mute/unmute button.

AUDIO_HEADSET_VOLUME_UP

The volume up button.

AUDIO_HEADSET_VOLUME_DOWN

The volume down button.

AUDIO_HEADSET_FORWARD

The track forward button.

AUDIO_HEADSET_BACKWARD

The track backward button.

AUDIO_HEADSET_BUTTON_COUNT

The total number of headset button types supported.

Library:

`libaudio_manager`

Description:

This enumeration defines the supported types of the audio headset buttons.

audio_manager_device_volume_control_t

Supported audio volume control types.

Synopsis:

```
#include <audio/audio_manager_device.h>


typedef enum {
      AUDIO_VOLUME_CONTROL_UNAVAILABLE
      AUDIO_VOLUME_CONTROL_SIMPLE
      AUDIO_VOLUME_CONTROL_PERCENT
      AUDIO_VOLUME_CONTROL_COUNT
} audio_manager_device_volume_control_t;
```

Data:

AUDIO_VOLUME_CONTROL_UNAVAILABLE

Volume control is not supported.

AUDIO_VOLUME_CONTROL_SIMPLE

The simple (up/down) volume control is supported.

AUDIO_VOLUME_CONTROL_PERCENT

The precise volume control in percentage is supported.

AUDIO_VOLUME_CONTROL_COUNT

The total number of volume control types supported.

Library:

`libaudio_manager`

Description:

This enumeration defines the supported types of the audio volume control types.

audio_manager_headset_button_state_t

The states of the headset buttons.

Synopsis:

```
typedef struct  {
    bool button_state [AUDIO_HEADSET_BUTTON_COUNT];
}audio_manager_headset_button_state_t;
```

Data:

bool button_state[AUDIO_HEADSET_BUTTON_COUNT]

The state of the audio headset button.

Library:

`libaudio_manager`

Description:

This structure defines the supported states of the audio headset buttons.

Functions in *audio_manager_device.h*

Functions for managing audio devices.

audio_manager_get_default_device()

Get the current default audio output device picked by audio manager.

Synopsis:

```
#include <audio/audio_manager_device.h>

int audio_manager_get_default_device(audio_manager_device_t *dev)
```

Arguments:

dev

The default audio device.

Library:

`libaudio_manager`

Description:

The audio_manager_get_default_device() function returns the type of the default audio output device picked by the audio manager.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_default_input_device()

Get the current default audio input device picked by audio manager.

Synopsis:

```
#include <audio/audio_manager_device.h>

int audio_manager_get_default_input_device(audio_manager_device_t *dev)
```

Arguments:

dev

The default audio input device.

Library:

`libaudio_manager`

Description:

The audio_manager_get_default_input_device() function returns the type of the default audio input device picked by the audio manager.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_device_audio_capabilities()

Get the audio capabilities of a given audio device.

Synopsis:

```
#include <audio/audio_manager_device.h>

int audio_manager_get_device_audio_capabilities(audio_manager_device_t dev,
audio_manager_device_capabilities_t *in, void **out, int count)
```

Arguments:

dev

The type of the audio device to query.

in

A list of capabilities requested.

out

The output pointers must match the type required for each input.

count

The number of elements in each of the `in` and `out` arrays.

Library:

`libaudio_manager`

Description:

The audio_manager_get_device_audio_capabilities() function returns the requested capabilities of the given audio device.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_get_device_audio_config()

Get the audio configuration of a given audio device.

Synopsis:

```
#include <audio/audio_manager_device.h>

int audio_manager_get_device_audio_config(audio_manager_device_t dev,
audio_manager_device_audio_config_t *config)
```

Arguments:

dev

The type of the audio device to query. If `AUDIO_DEVICE_DEFAULT` is passed in, the `num_in_channels` returned represents the number of input channels of the current default input device. The rest of the fields represent the settings of the current default output device.

config

The audio configuration.

Library:

`libaudio_manager`

Description:

The audio_manager_get_device_audio_config() function returns the audio configuration of the given audio device, using audio_manager_device_audio_config_t.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_get_device_audio_public()

Get whether an audio output device is public.

Synopsis:

```
#include <audio/audio_manager_device.h>

int audio_manager_get_device_audio_public(audio_manager_device_t dev, bool
*pub)
```

Arguments:

dev

The type of the audio device to query.

pub

`true` if the device is public, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_get_device_audio_public() function returns whether an audio output device is public, which means it can be heard by many people.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_device_button_states()

Get the current button states of a given output device.

Synopsis:

```
#include <audio/audio_manager_device.h>

int audio_manager_get_device_button_states(audio_manager_device_t dev,
audio_manager_headset_button_state_t *state)
```

Arguments:

dev

The default audio device.

state

The current button states.

Library:

`libaudio_manager`

Description:

The audio_manager_get_device_button_states() function returns the states of all the supported buttons of a given device. Currently, only headset is supported.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_device_capability()

Get the capabilities of a given audio device.

Synopsis:

```
#include <audio/audio_manager_device.h>

int audio_manager_get_device_capability(audio_manager_device_t dev,
audio_manager_device_capability_t *cap_mask)
```

Arguments:

dev

The type of the audio device to query.

cap_mask

The capabilities of the audio device, given as a mask of audio_manager_device_capability_t.

Library:

`libaudio_manager`

Description:

The audio_manager_get_device_capability() function returns the capabilities of a given audio device indicating whether it is capable of output and/or input.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_device_dependency()

Get the dependency of a given audio device.

Synopsis:

```
#include <audio/audio_manager_device.h>

int audio_manager_get_device_dependency(audio_manager_device_t dev,
audio_manager_device_t *dev_dependent)
```

Arguments:

dev

The type of the audio device to query.

dev_dependent

The associated dependent audio device.

Library:

`libaudio_manager`

Description:

The audio_manager_get_device_dependency() function returns the dependency of a given audio device. An audio device with an inactive (suspended/unsupported/disconnected) dependency is not picked by the audio manager as a valid routing destination/source.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_device_from_name()

Get the audio device given the name of the device.

Synopsis:

```
#include <audio/audio_manager_device.h>

audio_manager_device_t audio_manager_get_device_from_name(const char
*device_name)
```

Arguments:

device_name

The name of the device being returned.

Library:

`libaudio_manager`

Description:

The audio_manager_get_device_from_name() function returns the audio manager device given the name of the device.

Returns:

The device identifier.

audio_manager_get_device_name()

Get the audio device name given the type of the device.

Synopsis:

```
#include <audio/audio_manager_device.h>

const char* audio_manager_get_device_name(audio_manager_device_t device)
```

Arguments:

device

The type of the device being returned.

Library:

`libaudio_manager`

Description:

The audio_manager_get_device_name() function returns the audio manager device name given the type of the device.

Returns:

The device name.

audio_manager_get_device_volume_control()

Get the type of volume control of a given audio device.

Synopsis:

```
#include <audio/audio_manager_device.h>

int audio_manager_get_device_volume_control(audio_manager_device_t dev,
audio_manager_device_volume_control_t *control)
```

Arguments:

dev

The type of the audio device to query.

control

The type of the volume control.

Library:

`libaudio_manager`

Description:

The audio_manager_get_device_volume_control() function returns the type of the volume control of a given audio device.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_get_preferred_audio_input_path()

Get the preferred system audio input path.

Synopsis:

```
#include <audio/audio_manager_device.h>

int audio_manager_get_preferred_audio_input_path(char *path, int *size)
```

Arguments:

path

The audio path.

size

The size of the path buffer. If not sufficient, the minimum size to store the path is returned.

Library:

`libaudio_manager`

Description:

The audio_manager_get_preferred_audio_input_path() function returns the preferred system audio input path.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_get_preferred_audio_output_path()

Get the preferred system audio output path.

Synopsis:

```
#include <audio/audio_manager_device.h>

int audio_manager_get_preferred_audio_output_path(char *path, int *size)
```

Arguments:

path

The audio path.

size

The size of the path buffer. If not sufficient, the minimum size to store the path is returned.

Library:

`libaudio_manager`

Description:

The audio_manager_get_preferred_audio_output_path() function returns the preferred system audio output path.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_is_device_connected()

Check whether a given audio device is connected to the system.

Synopsis:

```
#include <audio/audio_manager_device.h>

int audio_manager_is_device_connected(audio_manager_device_t dev, bool
*connected)
```

Arguments:

dev

The type of the audio device to check against.

connected

`true` if the specified audio device is connected, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_is_device_connected() function checks whether a given audio device is currently connected to the system. Disconnected audio devices are not picked by the audio manager as a valid routing destination/source.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_is_device_kept_alive()

Check whether a given audio device is kept alive by the system.

Synopsis:

```
#include <audio/audio_manager_device.h>

int audio_manager_is_device_kept_alive(audio_manager_device_t dev, bool
*keep_alive)
```

Arguments:

dev

The type of the audio device to check against.

keep_alive

`true` if the specified audio device is kept alive, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_is_device_kept_alive() function checks whether a given audio device is currently kept alive by the system even when no audio streams are active. This is typically done to avoid excessive audio artifacts caused by hardware transitions.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_is_device_supported()

Check whether a given audio device is supported by the system.

Synopsis:

```
#include <audio/audio_manager_device.h>

int audio_manager_is_device_supported(audio_manager_device_t dev, bool
*supported)
```

Arguments:

dev

The type of the audio device to check against.

supported

`true` if the specified audio device is supported, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_is_device_supported() function checks whether a given audio device is currently supported by the system. Unsupported audio devices would result in errors when used against other audio manager interfaces, such as volume, event, and concurrency.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_is_device_suspended()

Check whether a given audio device is suspended by the system.

Synopsis:

```
#include <audio/audio_manager_device.h>

int audio_manager_is_device_suspended(audio_manager_device_t dev, bool
*suspended)
```

Arguments:

dev

The type of the audio device to check against.

suspended

`true` if the specified audio device is suspended, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_is_device_suspended() function checks whether a given audio device is currently suspended by the system. Suspended audio devices are not picked by the audio manager as a valid routing destination/source.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_is_hdmi_in_mirror_mode()

Check whether the mirror mode of the HDMI settings is on.

Synopsis:

```
#include <audio/audio_manager_device.h>

int audio_manager_is_hdmi_in_mirror_mode(bool *mirror_mode)
```

Arguments:

mirror_mode

`true` if the HDMI is set to mirror mode, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_is_hdmi_in_mirror_mode() function checks whether the HDMI is currently set to mirror mode.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_set_device_connected()

Set any audio device as connected.

Synopsis:

```
#include <audio/audio_manager_device.h>

int audio_manager_set_device_connected(audio_manager_device_t dev, bool
connected)
```

Arguments:

dev

The type of the audio device.

connected

`true` if the given audio device is to be marked as connected, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_set_device_connected() function sets the connected status of a given audio device to help the audio manager allocate proper audio devices as default.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_set_device_suspended()

Set a supported audio device as suspended.

Synopsis:

```
#include <audio/audio_manager_device.h>

int audio_manager_set_device_suspended(audio_manager_device_t dev, bool
suspended)
```

Arguments:

dev

The type of audio device.

suspended

`true` if the given audio device is to be marked as suspended, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_set_device_suspended() function sets the suspended status of a given audio device to help the audio manager allocate proper audio devices as default.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_set_hdmi_mirror_mode()

Set HDMI audio in mirror mode.

Synopsis:

```
#include <audio/audio_manager_device.h>

int audio_manager_set_hdmi_mirror_mode(bool mirror_mode)
```

Arguments:

mirror_mode

`true` if the HDMI is to be put in mirror mode, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_set_hdmi_mirror_mode() function puts the HDMI audio in mirror mode, which routes audio automatically to the HDMI.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_event.h

Definitions for supported audio manager events.

The audio manager controls audio routing, concurrency, and volume control automatically. A client can subscribe to specific events to listen for changes that are related to audio manager activities. The events are broadcast to all clients and are asynchronous. The clients that respond to the events are subject to act within a reasonable time frame in order to have audio transition without artifacts.

Data types in *audio_manager_event.h*

Data structures, typedefs, and enumerations for managing audio events.

audio_manager_concurrency_change_t

The audio concurrency change event.

Synopsis:

```
typedef struct  {
    audio_manager_audio_type_t audio_type ;
    audio_manager_concurrency_t status ;
}audio_manager_concurrency_change_t;
```

Data:

audio_manager_audio_type_t audio_type

The audio type that the event is triggered for.

audio_manager_concurrency_t status

The concurrency policy status of the given audio type.

Library:

`libaudio_manager`

Description:

This structure defines the changes to the audio concurrency policy of a given audio type.

audio_manager_device_button_change_t

The headset button change event.

Synopsis:

```
typedef struct  {
    audio_manager_device_t dev ;
    audio_manager_headset_button_state_t state ;
}audio_manager_device_button_change_t;
```

Data:

audio_manager_device_t dev

The audio device that the event is triggered for.

audio_manager_headset_button_state_t state

The current device button states.

Library:

`libaudio_manager`

Description:

This structure defines the changes to the headset button presses.

audio_manager_device_change_t

The audio device change event.

Synopsis:

```
typedef struct  {
    audio_manager_device_t dev ;
    char * diff ;
}audio_manager_device_change_t;
```

Data:

audio_manager_device_t dev

The audio device that the event is triggered for.

char * diff

The change in format of the audio device.

Library:

`libaudio_manager`

Description:

This structure defines the changes to the attribute(s) of a given audio device.

audio_manager_event_context_t

The event loop context.

Synopsis:

```
#include <audio/audio_manager_event.h>

typedef struct audio_manager_event_context audio_manager_event_context_t;
```

Library:

`libaudio_manager`

Description:

This structure defines the audio event context.

audio_manager_event_type_t

The supported audio events.

Synopsis:

```
#include <audio/audio_manager_event.h>

typedef enum {
      AUDIO_ROUTING_CHANGE
      AUDIO_CONCURRENCY_CHANGE
      AUDIO_DEVICE_CHANGE
      AUDIO_VOLUME_CHANGE
      AUDIO_HEADPHONE_BOOST_CHANGE
      AUDIO_VOICE_VOLUME_CHANGE
      AUDIO_VOICE_SERVICE_CHANGE
      AUDIO_DEVICE_BUTTON_CHANGE
      AUDIO_HEADPHONE_UNSAFE_ZONE_CHANGE
      AUDIO_STAT_CHANGE
      AUDIO_VOICE_OPTION_CHANGE
      AUDIO_HEADPHONE_OUTPUT_VOLUME_REGULATION_CHANGE
      AUDIO_EVENT_TYPE_COUNT
} audio_manager_event_type_t;
```

Data:

AUDIO_ROUTING_CHANGE

Changes to the audio input/output path.

AUDIO_CONCURRENCY_CHANGE

Changes to the audio concurrent policy of a given audio type.

AUDIO_DEVICE_CHANGE

Changes to the attribute(s) of a given audio device.

AUDIO_VOLUME_CHANGE

Changes to the volume of a given audio device.

AUDIO_HEADPHONE_BOOST_CHANGE

Changes to the headphone volume boost settings.

AUDIO_VOICE_VOLUME_CHANGE

Changes to the volume of a given audio device during voice calls.

AUDIO_VOICE_SERVICE_CHANGE

Changes to the status of a given voice service.

AUDIO_DEVICE_BUTTON_CHANGE

Changes to the button state of a given audio device.

AUDIO_HEADPHONE_UNSAFE_ZONE_CHANGE

Changes to the button state of a given audio device.

AUDIO_STAT_CHANGE

Changes to the statistics of audio usage.

AUDIO_VOICE_OPTION_CHANGE

Changes to the audio options of the voice services.

AUDIO_HEADPHONE_OUTPUT_VOLUME_REGULATION_CHANGE

Changes to the headphone output volume regulation.

AUDIO_EVENT_TYPE_COUNT

Total of the event types.

Library:

`libaudio_manager`

Description:

This enumeration defines the audio events currently supported.

audio_manager_routing_change_t

The changes of the audio input/output path.

Synopsis:

```
typedef struct  {
    audio_manager_device_t dev_prev ;
    audio_manager_device_t dev_now ;
    audio_manager_device_t dev_input_prev ;
    audio_manager_device_t dev_input_now ;
}audio_manager_routing_change_t;
```

Data:

audio_manager_device_t dev_prev

The audio output device that was previously active.

audio_manager_device_t dev_now

The audio output device currently being used as primary.

audio_manager_device_t dev_input_prev

The audio input device that was previously active.

audio_manager_device_t dev_input_now

The audio input device currently being used as primary.

Library:

`libaudio_manager`

Description:

This structure defines the audio routing change event.

audio_manager_stat_change_t

The audio statistics change event.

Synopsis:

```
typedef struct  {
    char name [64];
    uint64_t value ;
}audio_manager_stat_change_t;
```

Data:

char name[64]

The name of the statistic entry to be monitored.

uint64_t value

The value of the statistic entry.

Library:

`libaudio_manager`

Description:

This structure defines the changes to the audio statistics.

audio_manager_status_headphone_boost_change_t

The audio headphone boost change event.

Synopsis:

```
typedef struct  {
    int headphone_boost_level ;
    bool headphone_override ;
    audio_manager_headphone_volume_override_status_t headphone_boost ;
}audio_manager_status_headphone_boost_change_t;
```

Data:

int headphone_boost_level

The limit level of the headphone volume without boost enabled.

bool headphone_override

The current setting of headphone boost to override the limit.

audio_manager_headphone_volume_override_status_t headphone_boost

The status of the headphone volume boost.

Library:

`libaudio_manager`

Description:

This structure defines the changes to the headphone volume boost settings.

audio_manager_status_headphone_output_volume_regulation_change_t

The audio headphone output volume regulation change event.

Synopsis:

```
typedef struct  {
    audio_manager_headphone_output_regulation_t status ;
}audio_manager_status_headphone_output_volume_regulation_change_t;
```

Data:

audio_manager_headphone_output_regulation_t status

The status of the headphone output level regulation.

Library:

`libaudio_manager`

Description:

This structure defines the changes to the headphone output volume regulation status.

audio_manager_status_headphone_unsafe_zone_change_t

The audio headphone volume unsafe-zone change event.

Synopsis:

```
typedef struct  {
    audio_manager_headphone_volume_override_status_t headphone_unsafe_zone ;
}audio_manager_status_headphone_unsafe_zone_change_t;
```

Data:

audio_manager_headphone_volume_override_status_t headphone_unsafe_zone

The status of the headphone volume unsafe zone.

Library:

`libaudio_manager`

Description:

This structure defines the changes to the headphone volume boost settings.

audio_manager_status_volume_change_t

The audio volume change event.

Synopsis:

```
typedef struct  {
    audio_manager_device_t dev ;
    double output_level ;
    double input_level ;
    bool output_mute ;
    bool input_mute ;
}audio_manager_status_volume_change_t;
```

Data:

audio_manager_device_t dev

The audio device that the event is triggered for.

double output_level

The current output volume level (percentage) of the given audio device.

double input_level

The current input volume level (percentage) of the given audio device.

bool output_mute

The current output mute status of the given audio device.

bool input_mute

The current input mute status of the given audio device.

Library:

`libaudio_manager`

Description:

This structure defines the changes to the volume of a given audio device.

audio_manager_voice_audio_option_change_t

The voice audio options change event.

Synopsis:

```
typedef struct  {
    audio_manager_voice_service_t service ;
    audio_manager_device_t dev ;
```

```
    audio_manager_voice_option_t option ;
}audio_manager_voice_audio_option_change_t;
```

Data:

audio_manager_voice_service_t service

The voice service that the event is triggered for.

audio_manager_device_t dev

The audio device that the event is triggered for.

audio_manager_voice_option_t option

The current audio option selected.

Library:

`libaudio_manager`

Description:

This structure defines the changes to the audio options of a given voice service.

audio_manager_voice_service_change_t

The audio voice service change event.

Synopsis:

```
typedef struct  {
    audio_manager_voice_service_t service ;
    audio_manager_voice_service_status_t status ;
}audio_manager_voice_service_change_t;
```

Data:

audio_manager_voice_service_t service

The voice service that the event is triggered for.

audio_manager_voice_service_status_t status

The current status of the voice service.

Library:

`libaudio_manager`

Description:

This structure defines the changes to the status of a given voice service.

audio_manager_voice_status_volume_change_t

The audio voice volume change event.

Synopsis:

```
#include <audio/audio_manager_event.h>

typedef audio_manager_status_volume_change_t
audio_manager_voice_status_volume_change_t;
```

Library:

`libaudio_manager`

Description:

This structure defines the changes to the volume of a given audio device during voice calls.

Functions in *audio_manager_event.h*

Functions for managing audio events.

audio_manager_add_concurrency_change_event()

Add a concurrency change event to the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_add_concurrency_change_event(audio_manager_event_context_t
*context, audio_manager_audio_type_t audio_type)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

audio_type

The audio type that the event is triggered for.

Library:

`libaudio_manager`

Description:

The audio_manager_add_concurrency_change_event() function adds the change event to the event list, which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_add_device_button_change_event()

Add a device button change event to the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_add_device_button_change_event(audio_manager_event_context_t
 *context, audio_manager_device_t dev)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

dev

The audio device that the event is triggered for.

Library:

`libaudio_manager`

Description:

The audio_manager_add_device_button_change_event() function adds the change event to the event list, which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_add_device_change_event()

Add a device change event to the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_add_device_change_event(audio_manager_event_context_t
*context, audio_manager_device_t dev)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

dev

The audio device that the event is triggered for.

Library:

`libaudio_manager`

Description:

The audio_manager_add_device_change_event() function adds the change event to the event list, which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_add_routing_change_event()

Add a routing change event to the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_add_routing_change_event(audio_manager_event_context_t
*context)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

Library:

`libaudio_manager`

Description:

The audio_manager_add_routing_change_event() function adds the change event to the event list, which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_add_stat_change_event()

Add an audio statistics change event to the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_add_stat_change_event(audio_manager_event_context_t *context,
 const char *name)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

name

The voice service that the event is triggered for.

Library:

`libaudio_manager`

Description:

The audio_manager_add_stat_change_event() function adds the change event to the event list, which is monitored by calls to audio_manager_get_event(). The function

interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_add_status_hp_boost_change_event()

Add a headphone volume boost change event to the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_add_status_hp_boost_change_event(audio_manager_event_context_t
 *context)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

Library:

`libaudio_manager`

Description:

The audio_manager_add_status_hp_boost_change_event() function adds the change event to the event list, which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_add_status_hp_output_volume_regulation_change_event()

Add a headphone output volume regulation change event to the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>
```

```
int
audio_manager_add_status_hp_output_volume_regulation_change_event(audio_manager_event_context_t
 *context)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

Library:

`libaudio_manager`

Description:

The audio_manager_add_status_hp_output_volume_regulation_change_event() function adds the change event to the event list, which is monitored by calls to audio_manager_get_event().

The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_add_status_hp_unsafe_zone_change_event()

Add a headphone volume unsafe-zone change event to the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>

int
audio_manager_add_status_hp_unsafe_zone_change_event(audio_manager_event_context_t
 *context)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

Library:

`libaudio_manager`

Description:

The audio_manager_add_status_hp_unsafe_zone_change_event() function adds the change event to the event list, which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_add_voice_audio_option_change_event()

Add a voice audio option change event to the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>


int
audio_manager_add_voice_audio_option_change_event(audio_manager_event_context_t
 *context, audio_manager_voice_service_t service, audio_manager_device_t dev)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

service

The voice service that the event is triggered for.

dev

The audio device the event is triggered for.

Library:

`libaudio_manager`

Description:

The audio_manager_add_voice_audio_option_change_event() function adds the change event to the event list, which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_add_voice_service_change_event()

Add a voice service change event to the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_add_voice_service_change_event(audio_manager_event_context_t
 *context, audio_manager_voice_service_t service)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

service

The voice service that the event is triggered for.

Library:

`libaudio_manager`

Description:

The audio_manager_add_voice_service_change_event() function adds the change event to the event list, which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_add_voice_volume_change_event()

Add a voice volume change event to the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_add_voice_volume_change_event(audio_manager_event_context_t
 *context, audio_manager_device_t dev)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

dev

The audio device that the event is triggered for.

Library:

`libaudio_manager`

Description:

The audio_manager_add_voice_volume_change_event() function adds the change event to the event list, which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_add_volume_change_event()

Add a volume change event to the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_add_volume_change_event(audio_manager_event_context_t
*context, audio_manager_device_t dev)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

dev

The audio device that the event is triggered for.

Library:

`libaudio_manager`

Description:

The audio_manager_add_volume_change_event() function adds the change event to the event list, which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_dispatch_event()

Dispatch the change event.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_dispatch_event(audio_manager_event_context_t *context,
audio_manager_event_type_t event_type, void *event_params)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

event_type

The type of the event returned by audio_manager_peek_event().

event_params

The parameters of the event returned by audio_manager_peek_event().

Library:

`libaudio_manager`

Description:

The audio_manager_dispatch_event() function marks an event as dispatched and it no longer returns from audio_manager_peek_event() or audio_manager_dispatch_event().

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_event()

Get the next change event.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_get_event(audio_manager_event_context_t *context,
audio_manager_event_type_t *event_type, void **event_params)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

event_type

The type of the event triggered.

event_params

The parameters of the event triggered. The caller is responsible for casting the pointer to the proper change event structure.

Library:

`libaudio_manager`

Description:

The audio_manager_get_event() function blocks until the next change event is available. This function is used for typical message get/dispatch event loops. The event returned is cleared by calling the audio_manager_dispatch_event() function.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_event_context()

Allocate the event context.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_get_event_context(audio_manager_event_context_t **context)
```

Arguments:

context

The context returned to the caller.

Library:

`libaudio_manager`

Description:

The audio_manager_get_event_context() function allocates the context for events to be delivered.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_get_event_fd()

Get the file descriptors of the events.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_get_event_fd(audio_manager_event_context_t *context, int
*fd)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

fd

The file descriptor. This can be passed to `select()` or `io_notify()`.

Library:

libaudio_manager

Description:

The audio_manager_get_event_fd() function returns the current file descriptors of the events added to the event list.

Returns:

EOK upon success, negative errno upon failure.

audio_manager_peek_event()

Peek the next change event.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_peek_event(audio_manager_event_context_t *context, int fd,
audio_manager_event_type_t *event_type, void **event_params)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

fd

The file descriptor returned by audio_manager_get_event_fd().

event_type

The type of the event triggered.

event_params

The parameters of the event triggered. The caller is responsible for casting the pointer to the proper change event structure.

Library:

libaudio_manager

Description:

The audio_manager_peek_event() function checks whether any event has occurred and returns the event if one is found. If an event is found, the event is cleared by calling the audio_manager_dispatch_event() function.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_release_event_context()

Free the event context.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_release_event_context(audio_manager_event_context_t **context)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

Library:

`libaudio_manager`

Description:

The audio_manager_release_event_context() function frees the context allocated by audio_manager_get_event_context().

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_remove_concurrency_change_event()

Remove a concurrency change event from the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_remove_concurrency_change_event(audio_manager_event_context_t
 *context, audio_manager_audio_type_t audio_type)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

audio_type

The audio type that the event is triggered for.

Library:

`libaudio_manager`

Description:

The audio_manager_remove_concurrency_change_event() function removes the change event from the event list, which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_remove_device_button_change_event()

Remove a device button change event from the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>

int
audio_manager_remove_device_button_change_event(audio_manager_event_context_t
 *context, audio_manager_device_t dev)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

dev

The audio device that the event is triggered for.

Library:

`libaudio_manager`

Description:

The audio_manager_remove_device_button_change_event() function removes the change event from the event list, which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_remove_device_change_event()

Remove a device change event from the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_remove_device_change_event(audio_manager_event_context_t
*context, audio_manager_device_t dev)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

dev

The audio device that the event is triggered for.

Library:

`libaudio_manager`

Description:

The audio_manager_remove_device_change_event() function removes the change event from the event list, which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_remove_routing_change_event()

Remove a routing change event from the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_remove_routing_change_event(audio_manager_event_context_t
*context)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

Library:

`libaudio_manager`

Description:

The audio_manager_remove_routing_change_event() function removes the change event from the event list, which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_remove_stat_change_event()

Remove an audio statistics change event from the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_remove_stat_change_event(audio_manager_event_context_t
*context, const char *name)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

name

The voice service that the event is triggered for.

Library:

`libaudio_manager`

Description:

The audio_manager_remove_stat_change_event() function removes the event from the event list which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_remove_status_hp_boost_change_event()

Remove a headphone volume boost change event from the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>

int
audio_manager_remove_status_hp_boost_change_event(audio_manager_event_context_t
 *context)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

Library:

`libaudio_manager`

Description:

The audio_manager_add_status_hp_boost_change_event() function removes the event from the event list, which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being

blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

EOK upon success, negative errno upon failure.

audio_manager_remove_status_hp_unsafe_zone_change_event()

Remove a headphone volume unsafe-zone change event from the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>


int
audio_manager_remove_status_hp_unsafe_zone_change_event(audio_manager_event_context_t
 *context)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

Library:

libaudio_manager

Description:

The audio_manager_remove_status_hp_unsafe_zone_change_event() function removes the event from the event list, which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

EOK upon success, negative errno upon failure.

audio_manager_remove_voice_audio_option_change_event()

Remove a voice audio option change event from the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>


int
audio_manager_remove_voice_audio_option_change_event(audio_manager_event_context_t
 *context, audio_manager_voice_service_t service, audio_manager_device_t dev)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

service

The voice service that the event is triggered for.

dev

The audio device that the event is triggered for.

Library:

`libaudio_manager`

Description:

The audio_manager_remove_voice_audio_option_change_event() function removes the event from the event list which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_remove_voice_service_change_event()

Remove a voice service change event from the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>


int
audio_manager_remove_voice_service_change_event(audio_manager_event_context_t
 *context, audio_manager_voice_service_t service)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

service

The voice service that the event is triggered for.

Library:

`libaudio_manager`

Description:

The audio_manager_remove_voice_service_change_event() function removes the event from the event list which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_remove_voice_volume_change_event()

Remove a voice volume change event from the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_remove_voice_volume_change_event(audio_manager_event_context_t
 *context, audio_manager_device_t dev)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

dev

The audio device that the event is triggered for.

Library:

`libaudio_manager`

Description:

The audio_manager_remove_voice_volume_change_event() function removes the event from the event list, which is monitored by calls to audio_manager_get_event(). The

function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_remove_volume_change_event()

Remove a volume change event from the event list.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_remove_volume_change_event(audio_manager_event_context_t
*context, audio_manager_device_t dev)
```

Arguments:

context

The context returned by audio_manager_get_event_context().

dev

The audio device that the event is triggered for.

Library:

`libaudio_manager`

Description:

The audio_manager_remove_volume_change_event() function removes the change event from the event list, which is monitored by calls to audio_manager_get_event(). The function interrupts the audio_manager_get_event() function if it is already being blocked in another thread. The audio_manager_get_event() function is then called to include the new event.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_wait_for_initialization()

Block until the audio manager completes initialization.

Synopsis:

```
#include <audio/audio_manager_event.h>

int audio_manager_wait_for_initialization()
```

Arguments:

Library:

`libaudio_manager`

Description:

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_routing.h

Definitions for supported audio routing configurations.

The audio manager maintains the audio routing logic based on registered audio sources. This file defines routing properties and provides functions for them. The following are examples of using audio routing.

Setting an audio type from libasound

This example shows how to get an audio handle and set its type for a libasound channel. The example is for playback, but the audio manager specific steps are the same for recording. The example acquires an audio manager handle of type AUDIO_TYPE_ALERT, and sets it to become active only when PCM audio is actually playing (this is done via the suspended flag). The audio manager handle is bound to a libasound PCM handle, and freed when the program ends.

```
#include <audio/audio_manager_routing.h>
#include <sys/asoundlib.h>

void main(void)
{
  unsigned int audioman_handle;
  snd_pcm_t *pcm_handle;

  // Acquire an audioman handle. Start it "suspended", which means that its
  // ducking and routing rules only become active when the PCM channel is
  // actually playing.
  if ( audio_manager_get_handle ( AUDIO_TYPE_ALERT,    // Audio Type
                                  0,                   // pid that owns handle. 0 = this
                                  true,                // Start "suspended"
                                  &audioman_handle ) < 0 ) {
    printf("Could not get an audioman handle of the requested type\n");
    return;
  }

  // Acquire a pcm channel from libasound for playback
  snd_pcm_open ( &pcm_handle, "/dev/snd/pcmPreferred", SND_PCM_OPEN_PLAYBACK );

  // Bind the pcm_handle with the audioman handle.
  if ( snd_pcm_set_audioman_handle( pcm_handle, audioman_handle ) < 0 ) {
    printf("Could not set the pcm_handle with the audioman_handle\n");
  }

  // Do what you would normally do here with libasound...
  // ...

  // Cleanup
  snd_pcm_close( pcm_handle );
  audio_manager_free_handle( audioman_handle );
}
```

The audio type AUDIO_TYPE_ALERT, when active, will follow the routing of other active audio because it is lowest priority in the routing table. If nothing else is playing, it will force routing to be through the loud speaker, regardless of what other devices are connected.

For concurrency, if there is an AUDIO_TYPE_DEFAULT playing concurrently (which is the case for any stream that does not explicitly use the audio manager), when the

AUDIO_TYPE_ALERT becomes active, it will cause the AUDIO_TYPE_DEFAULT stream to become attenuated.

Alernatively, you can use utility functions in `audio_manager_routing.h` to implicitly handle the setup in one call. Note that the audioman_handle starts up suspended:

```
#include <audio/audio_manager_routing.h>
#include <sys/libasound.h>


void main(void)
{
  unsigned int audioman_handle;
  snd_pcm_t *pcm_handle;

  if (audio_manager_snd_pcm_open_name( AUDIO_TYPE_ALERT,
                       &pcm_handle,
                       &audioman_handle,
                       "/dev/snd/pcmPreferred",
                       SND_PCM_OPEN_PLAYBACK ) < 0 ) {
    printf("Failed to open an audioman pcm channel\n");
    return;
  }


  // Do what you would normally do with libasound...
  // ...


  // Cleanup
  snd_pcm_close( pcm_handle );
  audio_manager_free_handle( audioman_handle );
}
```

Setting an audio type from mm-renderer

This example shows how to set the audio type for an mm-renderer application. The code that follows sets the audio type of the mm-renderer context to AUDIO_TYPE_ALERT. Currently, the "audio_type" output parameter in mm-renderer is supported for playback only.

```
#include <mm/renderer.h>
#include <audio/audio_manager_routing.h>

{
    mmr_connection_t* connection;
    connection = mmr_connect(NULL);


    // Create a new mm-renderer context
    mmr_context_t* mmr_context;
    mmr_context = mmr_context_create(connection, context_name, 0, S_IRWXU|S

    // Set audio type by setting "audio_type" when calling mmr_output_param
    strm_dict_t* dict = strm_dict_new();
    dict = strm_dict_set( dict, "audio_type", audio_manager_get_name_from_t
    mmr_output_parameters( context, output_id, dict );

    //
```

```
    // Attach input here
    //
    // Play audio
    mmr_play( context );

    // ....


    //
    // Cleanup
    //
    strm_dict_destroy( dict );
    mmr_context_destroy(context);
    mmr_disconnect(connection);

}
```

Setting an audio manager handle from mm-renderer

There may be times when an application needs to do some advanced routing. For example, an application may want to do its own custom audio routing, instead of using the default audio routing provided by audio manager. In that case, an application may request a handle from audio manager and then pass this handle to mm-renderer. This way, the mm-renderer context is bound to the audio manager handle.

Your application can pass the audio manager handle to mm-renderer by calling the function mmr_output_parameters() and specifying "audioman_handle". Note that setting "audioman_handle" parameter on mm-renderer is supported for playback only.

mm-renderer will not activate or suspend the audio manager handle. It is the responsibility of the application to activate and suspend audio manager. The audio routing and audio ducking will be in effect only when an audio manager handle is active. In other words; when audio manager handle is suspended, no audio routing and audio ducking effects will happen, even if the audio is already playing.

You can activate an audio manager handle by:

- setting the suspended parameter to false when you request an audio manager handle by calling *audio_manager_get_handle()*. This will activate audio manager immediately.
- calling *audio_manager_activate_handle()*

You can suspend an audio manager handle by:

- freeing the audio manager handle by calling *audio_manager_free_handle()*
- calling *audio_manager_suspend_handle()*

```
#include <mm/renderer.h>
#include <audio/audio_manager_routing.h>

{
    mmr_connection_t* connection;
```

```
    connection = mmr_connect(NULL);

    // Create a new mm-renderer context
    mmr_context_t* mmr_context;
    mmr_context = mmr_context_create(connection, context_name, 0, S_IRWXU|S_IRWXG|S_IRWXO

    // Get an audio manager handle from Audio Manager handle and then convert it to a str
    // so that it can be passed to mm-renderer.
    // This call will activate audio manager handle immediately (notice the suspended par
    // set to false).
    uint32_t audioman_handle; audio_manager_get_handle(AUDIO_TYPE_DEFAULT, getpid(), fals
    char audioman_handle_str[15];
    itoa(audioman_handle, audioman_handle_str, 10);

    // Set audio manager handle by setting "audioman_handle" when calling mmr_output_para
    strm_dict_t* dict = strm_dict_new();
    dict = strm_dict_set( dict, "audioman_handle", audioman_handle_str);
    mmr_output_parameters( context, output_id, dict );

    //
    // Attach input here
    //

    // Play audio mmr_play( context );

    // ....

    //
    // Cleanup
    //
    strm_dict_destroy( dict );
    mmr_context_destroy(context);
    mmr_disconnect(connection); }
```

Forcing the audio routing

This example shows how to override the default routing for an audio type. In this example, because we are not actually binding a libasound pcm handle to the audioman handle, we start the audioman handle as not suspended so that the routing and concurrency policy changes take effect right away. This will force the output routing to speaker, and leave the input routing up to the default routing table. The routing override is in effect as long as the audioman_handle's type is the highest priority routing type currently active. Note that if there are multiple handles of the same type active at the same time, then it is a last-one-wins policy.

```
#include <audio/audio_manager_routing.h>

void main(void)
{
    unsigned int audioman_handle;

    // Acquire an audioman handle. Start it "unsuspended", which means that its
    // ducking and routing rules take effect immediately.
    if (audio_manager_get_handle(AUDIO_TYPE_DEFAULT, // Type default
                                 0,                  // This pid
                                 false,              // Start "unsuspended"
                                 &audioman_handle) == EOK) {

        // Use audio_manager_set_handle_type to also override the routing paths
        if (audio_manager_set_handle_type(audioman_handle, // audioman handle
                  AUDIO_TYPE_DEFAULT,              // Use the same type
                  AUDIO_DEVICE_SPEAKER,            // Force routing to loud speaker for c
                  AUDIO_DEVICE_DEFAULT ) == EOK) { // No preference for the input routing

            // Do what you would normally do here with libasound...
            // ...

        }
        audio_manager_free_handle(audioman_handle);
```

```
    } else {
        // Handle error as desired.
        // You may want to proceed with what you would normally do here
        // with libasound...
        // ...

    }
}
```

Constants in *audio_manager_routing.h*

Constants for managing audio routing.

Definitions in *audio_manager_routing.h*

Preprocessor macro definitions for the audio_manager_routing.h header file in the libaudio_manager library.

Definitions:

```
#define AUDIO_TYPE_NAMES_EXTENDED
```

Names for audio types.

```
#define AUDIO_TYPE_NAMES {                              \
    "voice",                    \
    "ringtone",                 \
    "voicerecognition",         \
    "texttospeech",             \
    "videochat",                \
    "voicerecording",           \
    "multimedia",               \
    "inputfeedback",            \
    "default",                  \
    "alert",                    \
    "voicetones",               \
    "soundeffect",              \
    "pushtotalk",               \
    "reserved_0",               \
    "cmas",                     \
    "alarm",                    \
    AUDIO_TYPE_NAMES_EXTENDED   \
}
```

Audio type names.

```
#define AUDIO_RUNTIME_NAMES {                               \
    "native",                   \
    "apkruntime"                \
}
```

Audio runtime names.

Library:

`libaudio_manager`

Data types in *audio_manager_routing.h*

Data structures, typedefs, and enumerations for managing audio routing.

audio_manager_audio_type_t

Supported audio types.

Synopsis:

```
#include <audio/audio_manager_routing.h>


typedef enum {
      AUDIO_TYPE_VOICE
      AUDIO_TYPE_RINGTONE
      AUDIO_TYPE_VOICE_RECOGNITION
      AUDIO_TYPE_TEXT_TO_SPEECH
      AUDIO_TYPE_VIDEO_CHAT
      AUDIO_TYPE_VOICE_RECORDING
      AUDIO_TYPE_MULTIMEDIA
      AUDIO_TYPE_INPUT_FEEDBACK
      AUDIO_TYPE_DEFAULT
      AUDIO_TYPE_ALERT
      AUDIO_TYPE_VOICE_TONES
      AUDIO_TYPE_SOUND_EFFECT
      AUDIO_TYPE_PUSH_TO_TALK
      AUDIO_TYPE_RESERVED_0
      AUDIO_TYPE_CMAS
      AUDIO_TYPE_ALARM
      AUDIO_TYPE_COUNT
} audio_manager_audio_type_t;
```

Data:

AUDIO_TYPE_VOICE

The audio type used by voice audio sources.

AUDIO_TYPE_RINGTONE

The audio type used by ringtone audio sources.

AUDIO_TYPE_VOICE_RECOGNITION

The audio type used by voice recognition audio sources.

AUDIO_TYPE_TEXT_TO_SPEECH

The audio type used by text-to-speech audio sources.

AUDIO_TYPE_VIDEO_CHAT

The audio type used by video chat audio sources.

AUDIO_TYPE_VOICE_RECORDING

The audio type used by voice recording audio sources.

AUDIO_TYPE_MULTIMEDIA

The audio type used by multimedia audio sources.

AUDIO_TYPE_INPUT_FEEDBACK

The audio type used by user input feedback.

AUDIO_TYPE_DEFAULT

The audio type used by default audio sources.

AUDIO_TYPE_ALERT

The audio type used by alert audio sources.

AUDIO_TYPE_VOICE_TONES

The audio type used by voice tone audio sources.

AUDIO_TYPE_SOUND_EFFECT

The audio type used by high priority sound effect.

AUDIO_TYPE_PUSH_TO_TALK

The audio type used by push-to-talk.

AUDIO_TYPE_RESERVED_0

The reserved audio type 0.

AUDIO_TYPE_CMAS

The audio type used by CMAS emergency broadcast systems.

AUDIO_TYPE_ALARM

The audio type used by alarms.

AUDIO_TYPE_COUNT

The total number of all audio types.

Library:

`libaudio_manager`

Description:

This enumeration defines the supported audio types.

audio_manager_runtime_t

Supported audio runtimes.

Synopsis:

```
#include <audio/audio_manager_routing.h>


typedef enum {
      AUDIO_RUNTIME_NATIVE
      AUDIO_RUNTIME_APKRUNTIME
      AUDIO_RUNTIME_COUNT
} audio_manager_runtime_t;
```

Data:

AUDIO_RUNTIME_NATIVE

The audio type used by clients designed for QNX directly.

AUDIO_RUNTIME_APKRUNTIME

The audio type used by clients designed for APK Runtime.

AUDIO_RUNTIME_COUNT

The total number of supported runtimes.

Library:

`libaudio_manager`

Description:

This enumeration defines the supported audio runtimes.

audio_manager_settings_reset_condition_t

Supported reset conditions of the routing preference settings.

Synopsis:

```
#include <audio/audio_manager_routing.h>


typedef enum {
      SETTINGS_NEVER_RESET = 0
      SETTINGS_RESET_ON_DEVICE_DISCONNECTION = 1
      SETTINGS_RESET_ON_DEVICE_CONNECTION = (1<<1)
      SETTINGS_RESET_ON_PREFERRED_DEVICE_DISCONNECTION = (1<<2)
      SETTINGS_RESET_ON_HIGHER_PRIORITY_DEVICE_CONNECTION = (1<<3)
} audio_manager_settings_reset_condition_t;
```

Data:

SETTINGS_NEVER_RESET

The preferences are never reset.

SETTINGS_RESET_ON_DEVICE_DISCONNECTION

The preferences are reset when any device disconnects.

SETTINGS_RESET_ON_DEVICE_CONNECTION

The preferences are reset when any device connects.

SETTINGS_RESET_ON_PREFERRED_DEVICE_DISCONNECTION

The preferences are reset when the preferred device disconnects.

SETTINGS_RESET_ON_HIGHER_PRIORITY_DEVICE_CONNECTION

The preferences are reset when the higher priority device connects.

Library:

`libaudio_manager`

Description:

This enumeration defines the supported reset conditions of the routing preferences set by audio_manager_set_handle_type().

Functions in *audio_manager_routing.h*

Functions for managing audio routing.

audio_manager_activate_bound_handle()

Activate the given audio manager handle and refresh audio ducking settings.

Synopsis:

```
#include <audio/audio_manager_routing.h>

int audio_manager_activate_bound_handle(unsigned int audioman_handle, bool
refresh_ducking)
```

Arguments:

audioman_handle

The audio manager handle that is being activated.

refresh_ducking

`true` if refresh ducking should be enabled, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_activate_bound_handle() function activates the given audio manager handle if it is already bound with a PCM handle by `snd_pcm_set_audioman_handle`.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_activate_handle()

Activate the given audio manager handle.

Synopsis:

```
#include <audio/audio_manager_routing.h>

int audio_manager_activate_handle(unsigned int audioman_handle)
```

Arguments:

audioman_handle

The audio manager handle that is being activated.

Library:

`libaudio_manager`

Description:

The audio_manager_activate_handle() function activates the given audio manager handle if it is not already bound with a PCM handle.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_check_autopause()

Check if a device switch should result in an autopause.

Synopsis:

```
#include <audio/audio_manager_routing.h>

int audio_manager_check_autopause(unsigned int audioman_handle,
audio_manager_device_t from, audio_manager_device_t to, bool *result)
```

Arguments:

audioman_handle

The audio manager handle on which to check.

from

The device that was active at the last point in time when the associated PCM channel was prepared.

to

The device that is now active.

result

`true` if the client should autopause, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_check_autopause() function indicates whether it is recommended that a client that has been forced to switch from one device to another should autopause.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_free_handle()

Free the given audio manager handle.

Synopsis:

```
#include <audio/audio_manager_routing.h>

int audio_manager_free_handle(unsigned int audioman_handle)
```

Arguments:

audioman_handle

The audio manager handle that is being freed.

Library:

`libaudio_manager`

Description:

The audio_manager_free_handle() function frees the given audio manager handle.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_alias_handle()

Get an audio manager handle that is an alias of another audio manager handle.

Synopsis:

```
#include <audio/audio_manager_routing.h>

int audio_manager_get_alias_handle(unsigned int target_audioman_handle, unsigned
 int *audioman_handle)
```

Arguments:

target_audioman_handle

The target audio manager handle that the alias is created from.

audioman_handle

The audio manager handle allocated.

Library:

`libaudio_manager`

Description:

The audio_manager_get_alias_handle() function returns the alias of a given audio manager handle. An alias audio manager handle has no audio policy effect.

Returns:

`EOK` upon success, negative `errno` upon failure

audio_manager_get_handle()

Get the audio manager handle for a given audio type.

Synopsis:

```
#include <audio/audio_manager_routing.h>

int audio_manager_get_handle(audio_manager_audio_type_t type, pid_t caller_pid,
 bool start_suspended, unsigned int *audioman_handle)
```

Arguments:

type

The type of the audio manager handle to query.

caller_pid

The ID of the process that the audio manager handle is allocated for (0 = the current process ID).

start_suspended

`true` if the audio manager handle is suspended after allocation, `false` otherwise.

audioman_handle

The audio manager handle allocated.

Library:

`libaudio_manager`

Description:

The audio_manager_get_handle() function returns the audio manager handle of a given type with a flag indicating whether the handle is immediately activated or not.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_handle_for_runtime()

Get the audio manager handle for a given audio type.

Synopsis:

```
#include <audio/audio_manager_routing.h>

int audio_manager_get_handle_for_runtime(audio_manager_runtime_t runtime,
audio_manager_audio_type_t type, pid_t caller_pid, bool start_suspended,
unsigned int *audioman_handle)
```

Arguments:

runtime

The type of the runtime of the audio manager to query.

type

The type of the audio manager handle to query.

caller_pid

The ID of the process that the audio manager handle is allocated for (0 = the current process ID).

start_suspended

`true` if the audio manager handle is suspended after allocation, `false` otherwise.

audioman_handle

The audio manager handle allocated.

Library:

`libaudio_manager`

Description:

The `audio_manager_get_handle_runtime()` function returns the audio manager handle of a given type with a flag indicating whether the handle is immediately activated or not.

This function should be used only by different runtimes which implement their own ducking rules. Applications should not use this function directly.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_get_handle_status()

Get the status of the given audio manager handle.

Synopsis:

```
#include <audio/audio_manager_routing.h>

int audio_manager_get_handle_status(unsigned int audioman_handle, bool
*suspended, bool *bound)
```

Arguments:

audioman_handle

The audio manager handle to query.

suspended

`true` if the audio manager handle is suspended, `false` otherwise.

bound

`true` if the audio manager handle is bound to a PCM handle, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_get_handle_status() function returns the activation status and binding status of a given audio manager handle.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_handle_type()

Retrieve the audio type of a given audio manager handle.

Synopsis:

```
#include <audio/audio_manager_routing.h>

int audio_manager_get_handle_type(unsigned int audioman_handle,
audio_manager_audio_type_t *type, audio_manager_device_t *pref_output,
audio_manager_device_t *pref_input)
```

Arguments:

audioman_handle

The audio manager handle that the new type is applied to.

type

The audio type that has been set on the given audio manager handle.

pref_output

The preferred output routing of the handle.

pref_input

The preferred input routing of the handle.

Library:

`libaudio_manager`

Description:

The audio_manager_get_handle_type() function gets the audio type of a given audio manager handle and the overrides of the default audio type routing policy for the preferred output and input audio devices.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_name_from_type()

Get the name of an audio manager type.

Synopsis:

```
#include <audio/audio_manager_routing.h>

const char* audio_manager_get_name_from_type(audio_manager_audio_type_t type)
```

Arguments:

type

The audio manager type to query.

Library:

`libaudio_manager`

Description:

The audio_manager_get_name_from_type() function returns the name of an audio manager type.

Returns:

The name of the audio manager type from `AUDIO_TYPE_NAMES`.

audio_manager_get_name_from_runtime()

Get the name of an audio manager runtime.

Synopsis:

```
#include <audio/audio_manager_routing.h>

const char* audio_manager_get_name_from_runtime(audio_manager_runtime_t runtime)
```

Arguments:

runtime

The audio manager runtime to query.

Library:

`libaudio_manager`

Description:

The audio_manager_get_name_from_runtime() function returns the name of an audio manager runtime.

Returns:

The name of the audio manager runtime from AUDIO_RUNTIME_NAMES.

audio_manager_get_runtime_from_name()

Get the audio handle runtime given the name of the runtime.

Synopsis:

```
#include <audio/audio_manager_routing.h>

audio_manager_runtime_t audio_manager_get_runtime_from_name(const char
*runtime_name)
```

Arguments:

runtime_name

The name of the runtime to query.

Library:

libaudio_manager

Description:

The audio_manager_get_runtime_from_name() function returns the audio manager handle runtime given the name of the runtime.

Returns:

The audio manager handle runtime.

audio_manager_get_type_from_name()

Get the audio handle type given the name of the type.

Synopsis:

```
#include <audio/audio_manager_routing.h>

audio_manager_audio_type_t audio_manager_get_type_from_name(const char
*type_name)
```

Arguments:

type_name

The name of the type to query.

Library:

`libaudio_manager`

Description:

The audio_manager_get_type_from_name() function returns the audio manager handle type given the name of the type.

Returns:

The audio manager handle type.

audio_manager_set_handle_routing_conditions()

Set the reset condition of the preferred input and output.

Synopsis:

```
#include <audio/audio_manager_routing.h>

int audio_manager_set_handle_routing_conditions(unsigned int audioman_handle,
 int routing_preference_reset_conditions)
```

Arguments:

audioman_handle

The audio manager handle that the conditions apply to.

routing_preference_reset_conditions

The bitmask of `audio_manager_settings_reset_condition_t` that specifies the condition(s) that the audio routing preference(s) gets set to.

Library:

`libaudio_manager`

Description:

The audio_manager_set_handle_routing_conditions() function sets the reset conditions of the preferred output and input routing path that are specified with audio_manager_set_handle_type().

Returns:

EOK upon success, negative errno upon failure.

audio_manager_set_handle_keep_alive()

Set the keep-alive status for the handle.

Synopsis:

```
#include <audio/audio_manager_routing.h>

int audio_manager_set_handle_keep_alive(unsigned int audioman_handle, int
direction)
```

Arguments:

audioman_handle

The audio manager handle that the keep-alive status applies to.

direction

A bitmask of AUDIO_INPUT or AUDIO_OUTPUT to control keep-alive direction. An empty bitmask will disable keep-alive.

Library:

libaudio_manager

Description:

The audio_manager_set_handle_keep_alive() function sets the keep-alive status of the handle. When audio is being routed according to this handle, the device being routed to will be kept ready, at a possible cost in power, even when no audio is being played or recorded.

Returns:

EOK upon success, negative errno upon failure.

audio_manager_set_handle_type()

Set the audio type of a given audio manager handle.

Synopsis:

```
#include <audio/audio_manager_routing.h>
```

```
int audio_manager_set_handle_type(unsigned int audioman_handle,
audio_manager_audio_type_t type, audio_manager_device_t pref_output,
audio_manager_device_t pref_input)
```

Arguments:

audioman_handle

The audio manager handle that the new type is applied to.

type

The new audio type to be set to the given audio manager handle. If AUDIO_TYPE_UNCHANGED is specified, the audio type is unchanged.

pref_output

The preferred output routing of the new audio type. If AUDIO_DEVICE_UNCHANGED is specified, the preferred output device is unchanged.

pref_input

The preferred input routing of the new audio type. If AUDIO_DEVICE_UNCHANGED is specified, the preferred input device is unchanged.

Library:

`libaudio_manager`

Description:

The audio_manager_set_handle_type() function sets the audio type of a given audio manager handle and gives the option to override the default audio type routing policy by specifying the preferred output and input audio devices.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_snd_pcm_open()

Open a PCM channel with a given audio type, given audio card, and device.

Synopsis:

```
#include <audio/audio_manager_routing.h>

int audio_manager_snd_pcm_open(audio_manager_audio_type_t type, snd_pcm_t
**handle, unsigned int *audioman_handle, int card, int device, int mode)
```

Arguments:

type

The audio type of the PCM channel being allocated.

handle

The handle of the PCM channel opened

audioman_handle

The audio manager handle allocated to the PCM channel.

card

The audio card to be used to open the PCM channel.

device

The audio device to be used to open the PCM channel.

mode

The PCM channel mode defined in `asoundlib.h`.

Library:

`libaudio_manager`

Description:

The audio_manager_snd_pcm_open() function combines the `snd_pcm_open()` and audio_manager_get_handle() functions and allows the allocation of a specific audio type PCM channel in one step.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_snd_pcm_open_name()

Open a PCM channel with a given audio type and a given name of the audio path.

Synopsis:

```
#include <audio/audio_manager_routing.h>

int audio_manager_snd_pcm_open_name(audio_manager_audio_type_t type, snd_pcm_t
 **handle, unsigned int *audioman_handle, char *name, int mode)
```

Arguments:

type

The audio type of the PCM channel being allocated.

handle

The handle of the PCM channel opened.

audioman_handle

The audio manager handle allocated to the PCM channel.

name

The name of the audio path to be used to open the PCM channel.

mode

The PCM channel mode defined in `asoundlib.h`.

Library:

`libaudio_manager`

Description:

The audio_manager_snd_pcm_open_name() function combines the `snd_pcm_open_name()` and audio_manager_get_handle() functions and allows the allocation a specific audio type PCM channel in one step.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_snd_pcm_open_preferred()

Open a preferred PCM channel with a given audio type.

Synopsis:

```
#include <audio/audio_manager_routing.h>


int audio_manager_snd_pcm_open_preferred(audio_manager_audio_type_t type,
snd_pcm_t **handle, unsigned int *audioman_handle, int *rcard, int *rdevice,
int mode)
```

Arguments:

type

The audio type of the PCM channel being allocated.

handle

The handle of the PCM channel opened.

audioman_handle

The audio manager handle allocated to the PCM channel.

rcard

The audio card used to open the PCM channel.

rdevice

The audio device used to open the PCM channel.

mode

The PCM channel mode defined in `asoundlib.h`.

Library:

`libaudio_manager`

Description:

The audio_manager_snd_pcm_open_preferred() function combines the `snd_pcm_open_preferred()` and audio_manager_get_handle() functions and allows the allocation of a specific audio type PCM channel in one step.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_suspend_bound_handle()

Activate the given audio manager handle and refresh audio ducking settings.

Synopsis:

```
#include <audio/audio_manager_routing.h>

int audio_manager_suspend_bound_handle(unsigned int audioman_handle)
```

Arguments:

audioman_handle

The audio manager handle that is being activated.

Library:

`libaudio_manager`

Description:

The audio_manager_activate_bound_handle() function activates the given audio manager handle if it is already bound with a PCM handle by `snd_pcm_set_audioman_handle`.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_suspend_handle()

Suspend the given audio manager handle.

Synopsis:

```
#include <audio/audio_manager_routing.h>

int audio_manager_suspend_handle(unsigned int audioman_handle)
```

Arguments:

audioman_handle

The audio manager handle that is being suspended.

Library:

`libaudio_manager`

Description:

The audio_manager_suspend_handle() function suspends the given audio manager handle if it is not already bound with a PCM handle.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_voice_service.h

Definitions for supported voice services.

The audio manager maintains the voice services status for devices. This file defines voice services status properties and provides functions for them.

Constants in *audio_manager_voice_service.h*

Constants for managing voice services.

Definitions in *audio_manager_voice_service.h*

Preprocessor macro definitions for the audio_manager_voice_service.h header file in the libaudio_manager library.

Definitions:

```
#define AUDIO_VOICE_SERVICE_NAMES {                                    \
    "cellular",                     \
    "voip",                         \
}
```

Audio voice service names.

```
#define AUDIO_VOICE_STATUS_NAMES {                                     \
    "off",                          \
    "ringer",                       \
    "on",                           \
}
```

Audio voice status names.

```
#define AUDIO_VOICE_OPTION_NAMES {                                     \
    "normal",                       \
    "boost_treble",                 \
    "boost_bass",                   \
}
```

Audio voice option names.

```
#define AUDIO_VOICE_CODEC_NAMES {                                      \
    "narrowband",                   \
    "wideband",                     \
}
```

Audio voice codec names.

Library:

`libaudio_manager`

Data types in *audio_manager_voice_service.h*

Data structures, typedefs, and enumerations for managing voice services.

audio_manager_voice_option_t

Supported audio options of the voice services.

Synopsis:

```
#include <audio/audio_manager_voice_service.h>


typedef enum {
      AUDIO_VOICE_OPTION_NORMAL
      AUDIO_VOICE_OPTION_TREBLE
      AUDIO_VOICE_OPTION_BASS
      AUDIO_VOICE_OPTION_COUNT
} audio_manager_voice_option_t;
```

Data:

AUDIO_VOICE_OPTION_NORMAL

The audio option for no particular audio tuning.

AUDIO_VOICE_OPTION_TREBLE

The audio option for treble audio tuning.

AUDIO_VOICE_OPTION_BASS

The audio option for bass audio tuning.

AUDIO_VOICE_OPTION_COUNT

The total number of audio options.

Library:

`libaudio_manager`

Description:

This enumeration defines the supported audio options of the voice services.

audio_manager_voice_service_t

The supported voice service types.

Synopsis:

```
#include <audio/audio_manager_voice_service.h>
```

```
typedef enum {
      AUDIO_VOICE_CELLULAR
      AUDIO_VOICE_VOIP
      AUDIO_VOICE_SERVICE_COUNT
} audio_manager_voice_service_t;
```

Data:

AUDIO_VOICE_CELLULAR

The cellular voice service.

AUDIO_VOICE_VOIP

The Voice over IP (VoIP) service.

AUDIO_VOICE_SERVICE_COUNT

The total of voice services supported.

Library:

`libaudio_manager`

Description:

This enumeration defines the supported voice service types.

audio_manager_voice_service_status_t

The supported status of the voice services.

Synopsis:

```
#include <audio/audio_manager_voice_service.h>


typedef enum {
      AUDIO_VOICE_OFF
      AUDIO_VOICE_RINGTONE
      AUDIO_VOICE_ON
      AUDIO_VOICE_STATUS_COUNT
} audio_manager_voice_service_status_t;
```

Data:

AUDIO_VOICE_OFF

The voice service is shut down.

AUDIO_VOICE_RINGTONE

The voice service is playing a ringtone.

AUDIO_VOICE_ON

The voice service is turned on.

AUDIO_VOICE_STATUS_COUNT

The total number of status types.

Library:

`libaudio_manager`

Description:

This enumeration defines the supported status of the voice services.

Functions in *audio_manager_voice_service.h*

Functions for managing voice services.

audio_manager_get_redirector_id()

Get the ID of redirector.

Synopsis:

```
#include <audio/audio_manager_voice_service.h>

int audio_manager_get_redirector_id(int *redirector_id)
```

Arguments:

redirector_id

The ID of redirector.

Library:

`libaudio_manager`

Description:

The audio_manager_get_redirector_id() function gets the ID of redirector.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_get_voice_service_from_name()

Get the audio voice service given the name of the service.

Synopsis:

```
#include <audio/audio_manager_voice_service.h>

audio_manager_voice_service_t audio_manager_get_voice_service_from_name(const
 char *name)
```

Arguments:

name

The name of the service to query.

Library:

`libaudio_manager`

Description:

The audio_manager_get_voice_service_from_name() function returns the audio voice service given the name of the service.

Returns:

The audio voice service identifier.

audio_manager_get_voice_service_name()

Get the audio voice service name given the type of the service.

Synopsis:

```
#include <audio/audio_manager_voice_service.h>

const char* audio_manager_get_voice_service_name(audio_manager_voice_service_t
 service)
```

Arguments:

service

The type of the service to query.

Library:

`libaudio_manager`

Description:

The audio_manager_get_voice_service_name() function returns the audio voice service name given the type of the service.

Returns:

The audio voice service name.

audio_manager_get_voice_service_option()

Get the current audio option of a given voice service type and a given audio device.

Synopsis:

```
#include <audio/audio_manager_voice_service.h>

int audio_manager_get_voice_service_option(audio_manager_voice_service_t
service, audio_manager_device_t dev, audio_manager_voice_option_t *option)
```

Arguments:

service

The voice service type to query.

dev

The audio device to query.

option

The audio option returned.

Library:

`libaudio_manager`

Description:

The audio_manager_get_voice_service_option() function returns the audio option of a given voice service type and a given audio device.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_voice_service_option_from_name()

Get the audio voice option given the name of the option.

Synopsis:

```
#include <audio/audio_manager_voice_service.h>

audio_manager_voice_option_t
audio_manager_get_voice_service_option_from_name(const char *name)
```

Arguments:

name

The name of the option to query.

Library:

`libaudio_manager`

Description:

The audio_manager_get_voice_service_option_from_name() function returns the enhanced audio option of a voice service given the name of the option.

Returns:

The option identifier.

audio_manager_get_voice_service_option_name()

Get the enhanced audio option name given the type of the option.

Synopsis:

```
#include <audio/audio_manager_voice_service.h>

const char*
audio_manager_get_voice_service_option_name(audio_manager_voice_option_t option)
```

Arguments:

option

The type of the option to query.

Library:

`libaudio_manager`

Description:

The audio_manager_get_voice_service_option_name() function returns the name of the enhanced audio option for the voice service given the type of the option.

Returns:

The option name.

audio_manager_get_voice_service_status()

Get the current status of a given voice service type.

Synopsis:

```
#include <audio/audio_manager_voice_service.h>

int audio_manager_get_voice_service_status(audio_manager_voice_service_t
service, audio_manager_voice_service_status_t *status)
```

Arguments:

service

The voice service type to query.

status

The status returned.

Library:

`libaudio_manager`

Description:

The audio_manager_get_voice_service_status() function returns the status of a given voice service type.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_voice_service_status_from_name()

Get the audio voice status given the name of the status.

Synopsis:

```
#include <audio/audio_manager_voice_service.h>
```

```
audio_manager_voice_service_status_t
audio_manager_get_voice_service_status_from_name(const char *name)
```

Arguments:

name

The name of the status to query.

Library:

`libaudio_manager`

Description:

The audio_manager_get_voice_service_status_from_name() function returns the voice status given the name of the status.

Returns:

The service status identifier.

audio_manager_get_voice_service_status_name()

Get the audio voice service status name given the type of the status.

Synopsis:

```
#include <audio/audio_manager_voice_service.h>


const char*
audio_manager_get_voice_service_status_name(audio_manager_voice_service_status_t
 status)
```

Arguments:

status

The type of the status to query.

Library:

`libaudio_manager`

Description:

The audio_manager_get_voice_service_status_name() function returns the audio voice service status name given the type of the status.

Returns:

The service status name.

audio_manager_get_voice_service_status_with_codec_settings()

Get the current status and the codec settings of a given voice service type.

Synopsis:

```
#include <audio/audio_manager_voice_service.h>


int
audio_manager_get_voice_service_status_with_codec_settings(audio_manager_voice_service_t
 service, audio_manager_voice_service_status_t *status, char *codec_name_buf,
 int buf_size, int *codec_rate)
```

Arguments:

service

The voice service type to query.

status

The status returned.

codec_name_buf

The buffer to store the name of the codec the service provider uses; NULL if not needed.

buf_size

The size of the `codec_name_buf`.

codec_rate

The pointer to store the audio sample rate of the service provider.

Library:

`libaudio_manager`

Description:

The audio_manager_get_voice_service_status_with_codec_settings() function returns the status and the codec settings of a given voice service type.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_set_voice_service_option()

Set the audio option of a given voice service type and a given audio device.

Synopsis:

```
#include <audio/audio_manager_voice_service.h>

int audio_manager_set_voice_service_option(audio_manager_voice_service_t
service, audio_manager_device_t dev_output, audio_manager_voice_option_t option)
```

Arguments:

service

The voice service type that the new audio option is applied to.

dev_output

The audio device that the new audio option is applied to.

option

The new audio option to be applied.

Library:

`libaudio_manager`

Description:

The audio_manager_set_voice_service_option() function sets the audio option of a given voice service type and a given audio device.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_set_voice_service_status()

Set the status of a given voice service type.

Synopsis:

```
#include <audio/audio_manager_voice_service.h>

int audio_manager_set_voice_service_status(audio_manager_voice_service_t
service, audio_manager_voice_service_status_t status)
```

Arguments:

service

The voice service type that the new status is applied to.

status

The status to be applied.

Library:

`libaudio_manager`

Description:

The audio_manager_set_voice_service_status() function sets the status of a given voice service type.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_set_voice_service_status_with_codec_settings()

Set the status and the codec settings of a given voice service type.

Synopsis:

```
#include <audio/audio_manager_voice_service.h>

int
audio_manager_set_voice_service_status_with_codec_settings(audio_manager_voice_service_t
```

```
service, audio_manager_voice_service_status_t status, const char *codec_name,
int codec_rate)
```

Arguments:

service

The voice service type that the new status is applied to.

status

The status to be applied.

codec_name

The name of the codec the service provider uses.

codec_rate

The audio sample rate the service provider uses.

Library:

`libaudio_manager`

Description:

The audio_manager_set_voice_service_status_with_codec_settings() function sets the status and the codec settings of a given voice service type.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_volume.h

Definitions for supported audio volume controls.

The audio manager maintains the volume control interfaces for all supported devices. This file defines audio volume control properties and provides functions for them.

Constants in *audio_manager_volume.h*

Constants for managing volume control.

Definitions in *audio_manager_volume.h*

Preprocessor macro definitions for the audio_manager_volume.h header file in the libaudio_manager library.

Definitions:

`#define UNSAFEZONE_TIMEOUT_ID "Audio.EU.headphone.boost.timer"`

Statistic entry name used with audio_manager_get_stat_counter() for EU headphone volume regulations.

This counter tracks the total time the user has spent in the EU unsafe volume zone with headphones.

`#define UNSAFEZONE_DEFAULT_EU_UNSAFE_LEVEL 75`

The default threshold in percentage for entering the EU unsafe volume zone for headphones.

`#define UNSAFEZONE_DEFAULT_EU_UNSAFE_TIMEOUT 72000000`

The default timeout of the EU unsafe volume zone for headphones.

Once the timer reaches this threshold, the user is required to acknowledge the EU regulations again to use headphones with volume level above the UNSAFEZONE_DEFAULT_EU_UNSAFE_LEVEL.

Library:

`libaudio_manager`

Data types in *audio_manager_volume.h*

Data structures, typedefs, and enumerations for managing audio volume.

audio_manager_headphone_output_regulation_t

The audio headphone output volume regulation status.

Synopsis:

```
typedef struct  {
    bool regulated ;
    double level ;
}audio_manager_headphone_output_regulation_t;
```

Data:

bool regulated

The current status of whether the headphone volume is regulated.

double level

The output level in percentage that the headphone volume is regulated at.

Library:

`libaudio_manager`

Description:

This structure defines the status of whether and at which level the headphone output volume is regulated.

audio_manager_headphone_volume_override_status_t

Headphone volume override status.

Synopsis:

```
typedef struct  {
    bool supported ;
    bool enabled ;
    int level ;
}audio_manager_headphone_volume_override_status_t;
```

Data:

bool supported

true if the extra volume override is supported by the device or region, false otherwise.

bool enabled

true if the extra volume range is allowed, false otherwise.

int level

The output level in percentage (0-100) that the headphone volume is limited to if the override is false.

Library:

`libaudio_manager`

Description:

This structure defines the status of the configuration of a given headphone volume override feature.

Functions in *audio_manager_volume.h*

Functions for managing audio volume.

audio_manager_adjust_input_level()

Adjust the audio input level of a given audio device.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_adjust_input_level(audio_manager_device_t dev, double level)
```

Arguments:

dev

The audio device the new input level is applied to.

level

The change in level of the audio input in percentage (e.g. 10.00 = 10% increase, -10.00 = 10% decrease).

Library:

`libaudio_manager`

Description:

The audio_manager_adjust_input_level() function adjusts the audio input level of a given audio device.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_adjust_output_level()

Adjust the audio output level of a given audio device.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_adjust_output_level(audio_manager_device_t dev, double level)
```

Arguments:

dev

The audio device that the new output level is applied to.

level

The change in level of the audio output in percentage (e.g. 10.00 = 10% increase, -10.00 = 10% decrease).

Library:

`libaudio_manager`

Description:

The audio_manager_adjust_output_level() function adjusts the audio output level of a given audio device.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_adjust_voice_output_level()

Adjust the audio output level of a given audio device during voice calls.

Synopsis:

```
#include <audio/audio_manager_volume.h>
```

```
int audio_manager_adjust_voice_output_level(audio_manager_device_t dev, double
 level)
```

Arguments:

dev

The audio device that the new output level is applied to.

level

The change in level of the audio output in percentage, 0.00 - 100.0. (e.g. 10.00 = 10% increase, -10.00 = 10% decrease)

Library:

`libaudio_manager`

Description:

The audio_manager_adjust_voice_output_level() function adjusts the audio output level of a given audio device during voice calls.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_clear_stat_counter()

Clear the statistic counter of a given statistic entry name.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_clear_stat_counter(const char *name)
```

Arguments:

name

The name of the statistic entry to clear.

Library:

`libaudio_manager`

Description:

The audio_manager_clear_stat_counter() function clears the counter of the given statistic entry.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_decrease_output_level()

Decrease the audio output level of a given audio device.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_decrease_output_level(audio_manager_device_t dev)
```

Arguments:

dev

The audio device that the decrease is applied to.

Library:

`libaudio_manager`

Description:

The audio_manager_decrease_output_level() function decreases the audio output level of a given audio device. The step of the output level decrease is defined by the particular audio device.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_decrease_voice_output_level()

Decrease the audio output level of a given audio device during voice calls.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_decrease_voice_output_level(audio_manager_device_t dev)
```

Arguments:

dev

The audio device that the decrease is applied to.

Library:

`libaudio_manager`

Description:

The audio_manager_decrease_voice_output_level() function decreases the audio output level of a given audio device during voice calls. The step of the output level decrease is defined by the particular audio device.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_get_headphone_boost_status()

Get the headphone volume boost status.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int
audio_manager_get_headphone_boost_status(audio_manager_headphone_volume_override_status_t
 *status)
```

Arguments:

status

The status of the volume boost as audio_manager_headphone_volume_override_status_t.

Library:

`libaudio_manager`

Description:

The audio_manager_get_headphone_boost_status() function returns the headphone volume boost status. The override allows extra volume boost to the headphone output.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_headphone_output_level_regulation_status()

Get whether the current headphone output level is restricted due to regulations.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int
audio_manager_get_headphone_output_level_regulation_status(audio_manager_headphone_output_regulation_t
 *status)
```

Arguments:

status

The status of the output level as audio_manager_headphone_output_regulation_t.

Library:

`libaudio_manager`

Description:

The `audio_manager_get_headphone_level_regulated()` function returns whether the level of the current headphone output level is regulated to a lower level than the user or application might have set.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_headphone_unsafe_zone_status()

Get the headphone volume unsafe zone status.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int
audio_manager_get_headphone_unsafe_zone_status(audio_manager_headphone_volume_override_status_t
 *status)
```

Arguments:

status

The status of the volume unsafe zone as audio_manager_headphone_volume_override_status_t.

Library:

`libaudio_manager`

Description:

The audio_manager_get_headphone_unsafe_zone_status() function returns the headphone volume unsafe zone status.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_input_level()

Get the audio input level of a given audio device.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_get_input_level(audio_manager_device_t dev, double *level)
```

Arguments:

dev

The audio device to query.

level

The input level being returned in percentage, 0.00 - 100.0 (e.g. 90.00 = 90%).

Library:

`libaudio_manager`

Description:

The audio_manager_get_input_level() function returns the audio input level of a given audio device.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_get_input_mute()

Get the mute status of the audio input of a given audio device.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_get_input_mute(audio_manager_device_t dev, bool *mute)
```

Arguments:

dev

The audio device to query.

mute

`true` if the input of the audio device is being muted, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_get_input_mute() function returns the mute status of the audio input of a given audio device.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_modem_output_mute()

Get the mute status of the audio output of the modem.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_get_modem_output_mute(bool *mute)
```

Arguments:

mute

`true` if the output of the modem is muted, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_get_modem_output_mute() function returns the mute status of the audio output of the modem.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_output_level()

Get the audio output level of a given audio device.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_get_output_level(audio_manager_device_t dev, double *level)
```

Arguments:

dev

The audio device to query.

level

The output level being returned in percentage, 0.00 - 100.0 (e.g. 90.00 = 90%).

Library:

`libaudio_manager`

Description:

The audio_manager_get_output_level() function returns the audio output level of a given audio device.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_get_output_mute()

Get the mute status of the audio output of a given audio device.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_get_output_mute(audio_manager_device_t dev, bool *mute)
```

Arguments:

dev

The audio device to query.

mute

`true` if the output of the audio device is being muted, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_get_output_mute() function returns the mute status of the audio output of a given audio device.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_output_volume_steps()

Get the number of available output volume steps.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_get_output_volume_steps(audio_manager_device_t dev, int
*steps)
```

Arguments:

dev

The audio device to query.

steps

The available volume steps returned.

Library:

`libaudio_manager`

Description:

The audio_manager_get_output_volume_steps() function returns the number of available volume steps.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_stat_counter()

Get the statistic counter of a given statistic entry name.

Synopsis:

```
#include <audio/audio_manager_volume.h>
```

```
int audio_manager_get_stat_counter(const char *name, uint64_t *counter)
```

Arguments:

name

The name of the statistic entry.

counter

The counter of the selected entry returned.

Library:

`libaudio_manager`

Description:

The audio_manager_get_stat_counter() function returns the counter of the given statistic entry.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_voice_input_mute()

Get the mute status of the audio input (to the far end) of the current voice call.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_get_voice_input_mute(bool *mute)
```

Arguments:

mute

`true` if the input of the voice call is being muted, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_get_voice_input_mute() function returns the mute status of the audio input (to the far end) of the current voice call.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_voice_output_level()

Get the audio output level of a given audio device during voice calls.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_get_voice_output_level(audio_manager_device_t dev, double
*level)
```

Arguments:

dev

The audio device to query.

level

The output level being returned in percentage, 0.00 - 100.0 (e.g. 90.00 = 90%).

Library:

`libaudio_manager`

Description:

The audio_manager_get_voice_output_level() function returns the audio output level of a given audio device during voice calls.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_get_voice_output_mute()

Get the mute status of the audio output of a given audio device during voice calls.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_get_voice_output_mute(audio_manager_device_t dev, bool *mute)
```

Arguments:

dev

The audio device to query.

mute

`true` if the output of the audio device is being muted, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_get_voice_output_mute() function returns the mute status of the audio output of a given audio device during voice calls.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_get_voice_output_volume_steps()

Get the number of available voice output volume steps.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_get_voice_output_volume_steps(audio_manager_device_t dev,
int *steps)
```

Arguments:

dev

The audio device to query.

steps

The available volume steps returned.

Library:

`libaudio_manager`

Description:

The audio_manager_get_voice_output_volume_steps() function returns the number of available volume steps for voice calls.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_increase_output_level()

Increase the audio output level of a given audio device.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_increase_output_level(audio_manager_device_t dev)
```

Arguments:

dev

The audio device that the increase is applied to.

Library:

`libaudio_manager`

Description:

The audio_manager_increase_output_level() function increases the audio output level of a given audio device. The step of the output level increase is defined by the particular audio device.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_increase_voice_output_level()

Increase the audio output level of a given audio device during voice calls.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_increase_voice_output_level(audio_manager_device_t dev)
```

Arguments:

dev

The audio device that the increase is applied to.

Library:

`libaudio_manager`

Description:

The audio_manager_increase_voice_output_level() function increases the audio output level of a given audio device during voice calls. The step of the output level increase is defined by the particular audio device.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_set_headphone_volume_boost()

Set the use of the headphone output volume boost.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_set_headphone_volume_boost(bool enable)
```

Arguments:

enable

`true` if the extra volume boost is allowed, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_set_headphone_volume_boost() function sets the enable status of the headphone output level to allow an extra volume boost to the headphone output.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_set_headphone_volume_unsafe_zone()

Set the use of the unsafe range of the headphone output volume.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_set_headphone_volume_unsafe_zone(bool enable)
```

Arguments:

enable

`true` if the extra volume range is allowed, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_set_headphone_volume_unsafe_zone() function sets the enable status of the unsafe volume range of the headphone output to allow an extra volume range to the headphone output.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_set_input_level()

Set the audio input level of a given audio device.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_set_input_level(audio_manager_device_t dev, double level)
```

Arguments:

dev

The audio device that the new input level is applied to.

level

The new input level in percentage, 0.00 - 100.00 (e.g. 90.00 = 90%).

Library:

`libaudio_manager`

Description:

The audio_manager_set_input_level() function sets the audio input level of a given audio device.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_set_input_mute()

Mute the audio input of a given audio device.

Synopsis:

```
#include <audio/audio_manager_volume.h>
```

```
int audio_manager_set_input_mute(audio_manager_device_t dev, bool mute)
```

Arguments:

dev

The audio device to mute input.

mute

`true` if the input of the audio device should be muted, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_set_input_mute() function mutes the audio input of a given audio device.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_set_modem_output_mute()

Mute the audio output of the modem.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_set_modem_output_mute(bool mute)
```

Arguments:

mute

`true` if the output of the modem should be muted, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_set_modem_output_mute() function mutes the audio output of the modem.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_set_output_level()

Set the audio output level of a given audio device.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_set_output_level(audio_manager_device_t dev, double level)
```

Arguments:

dev

The audio device that the new output level is applied to.

level

The new output level in percentage, 0.00 - 100.0 (e.g. 90.00 = 90%).

Library:

`libaudio_manager`

Description:

The audio_manager_set_output_level() function sets the audio output level of a given audio device.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_set_output_mute()

Mute the audio output of a given audio device.

Synopsis:

```
#include <audio/audio_manager_volume.h>
```

```
int audio_manager_set_output_mute(audio_manager_device_t dev, bool mute)
```

Arguments:

dev

The audio device to mute output.

mute

`true` if the output of the audio device should be muted, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_set_output_mute() function mutes the audio output of a given audio device.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_set_voice_input_mute()

Mute the audio input (to the far end) of the current voice call.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_set_voice_input_mute(bool mute)
```

Arguments:

mute

`true` if the input of the current voice call should be muted, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_set_voice_input_mute() function mutes the audio input (to the far end) of the current voice call.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

`EOK` upon success, negative `errno` upon failure.

audio_manager_set_voice_output_level()

Set the audio output level of a given audio device during voice calls.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_set_voice_output_level(audio_manager_device_t dev, double
level)
```

Arguments:

dev

The audio device that the new output level is applied to.

level

The new output level in percentage, 0.00 - 100.0 (e.g. 90.00 = 90%).

Library:

`libaudio_manager`

Description:

The audio_manager_set_voice_output_level() function sets the audio output level of a given audio device during voice calls.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_set_voice_output_mute()

Mute the audio output of a given audio device during voice calls.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_set_voice_output_mute(audio_manager_device_t dev, bool mute)
```

Arguments:

dev

The audio device to mute output.

mute

`true` if the output of the audio device should be muted, `false` otherwise.

Library:

`libaudio_manager`

Description:

The audio_manager_set_voice_output_mute() function mutes the audio output of a given audio device during voice calls.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_toggle_input_mute()

Toggle the audio input mute status of a given audio device.

Synopsis:

```
#include <audio/audio_manager_volume.h>
```

```
int audio_manager_toggle_input_mute(audio_manager_device_t dev)
```

Arguments:

dev

The audio device that the toggle is applied to.

Library:

`libaudio_manager`

Description:

The audio_manager_toggle_output_mute() function toggles the mute status of the audio input of a given audio device.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_toggle_output_mute()

Toggle the audio output mute status of a given audio device.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_toggle_output_mute(audio_manager_device_t dev)
```

Arguments:

dev

The audio device that the toggle is applied to.

Library:

`libaudio_manager`

Description:

The audio_manager_toggle_output_mute() function toggles the mute status of the audio output of a given audio device.

Returns:

EOK upon success, negative `errno` upon failure.

audio_manager_toggle_voice_output_mute()

Toggle the audio output mute status of a given audio device during voice calls.

Synopsis:

```
#include <audio/audio_manager_volume.h>

int audio_manager_toggle_voice_output_mute(audio_manager_device_t dev)
```

Arguments:

dev

The audio device that the toggle is applied to.

Library:

`libaudio_manager`

Description:

The audio_manager_toggle_voice_output_mute() function toggles the mute status of the audio output of a given audio device during voice calls.

A process must have either an effective user ID of root, or the authman capability of `access_audio_manager`, to use this function.

Note that this function is intended for use by system components. Therefore, it is not suitable for all applications.

Returns:

`EOK` upon success, negative `errno` upon failure.

Index

T