40.1. sunaudiodev — Access to Sun audio hardware¶

Platforms: SunOS

Deprecated since version 2.6: The `sunaudiodev` module has been deprecated for removal in Python 3.0.

This module allows you to access the Sun audio interface. The Sun audio hardware is capable of recording and playing back audio data in u-LAW format with a sample rate of 8K per second. A full description can be found in the *audio(7I)* manual page.

The module `SUNAUDIODEV` defines constants which may be used with this module.

This module defines the following variables and functions:

exception `sunaudiodev.error`¶

This exception is raised on all errors. The argument is a string describing what went wrong.

`sunaudiodev.open`(*mode*)¶

This function opens the audio device and returns a Sun audio device object. This object can then be used to do I/O on. The *mode* parameter is one of `'r'` for record-only access, `'w'` for play-only access, `'rw'` for both and `'control'` for access to the control device. Since only one process is allowed to have the recorder or player open at the same time it is a good idea to open the device only for the activity needed. See *audio(7I)* for details.

As per the manpage, this module first looks in the environment variable `AUDIODEV` for the base audio device filename. If not found, it falls back to `/dev/audio`. The control device is calculated by appending “ctl” to the base audio device.

40.1.1. Audio Device Objects¶

The audio device objects are returned by `open()` define the following methods (except `control` objects which only provide `getinfo()`, `setinfo()`, `fileno()`, and `drain()`):

`audio device.close()`

This method explicitly closes the device. It is useful in situations where deleting the object does not immediately close it since there are other references to it. A closed device should not be used again.

`audio device.fileno()`

Returns the file descriptor associated with the device. This can be used to set up `SIGPOLL` notification, as described below.

`audio device.drain()`

This method waits until all pending output is processed and then returns. Calling this method is often not necessary: destroying the object will automatically close the audio device and this will do an implicit drain.

`audio device.flush()`

This method discards all pending output. It can be used avoid the slow response to a user’s stop request (due to buffering of up to one second of sound).

`audio device.getinfo()`

This method retrieves status information like input and output volume, etc. and returns it in the form of an audio status object. This object has no methods but it contains a number of attributes describing the current device status. The names and meanings of the attributes are described in `<sun/audioio.h>` and in the *audio(7I)* manual page. Member names are slightly different from their C counterparts: a status object is only a single structure. Members of the `play` substructure have `o_` prepended to their name and members of the `record` structure have `i_`. So, the C member `play.sample_rate` is accessed as `o_sample_rate`, `record.gain` as `i_gain` and `monitor_gain` plainly as `monitor_gain`.

`audio device.ibufcount()`

This method returns the number of samples that are buffered on the recording side, i.e. the program will not block on a `read()` call of so many samples.

`audio device.obufcount()`

This method returns the number of samples buffered on the playback side. Unfortunately, this number cannot be used to determine a number of samples that can be written without blocking since the kernel output queue length seems to be variable.

`audio device.read(size)`

This method reads *size* samples from the audio input and returns them as a Python string. The function blocks until enough data is available.

`audio device.setinfo(status)`

This method sets the audio device status parameters. The *status* parameter is an device status object as returned by `getinfo()` and possibly modified by the program.

```
audio device.write(samples)
```

Write is passed a Python string containing audio samples to be played. If there is enough buffer space free it will immediately return, otherwise it will block.

The audio device supports asynchronous notification of various events, through the SIGPOLL signal. Here's an example of how you might enable this in Python:

```
def handle_sigpoll(signum, frame):
   print 'I got a SIGPOLL update'

import fcntl, signal, STROPTS

signal.signal(signal.SIGPOLL, handle_sigpoll)
fcntl.ioctl(audio_obj.fileno(), STROPTS.I_SETSIG, STROPTS.S_MSG)
```

40.2. SUNAUDIODEV — Constants used with sunaudiodev¶

Platforms: SunOS

Deprecated since version 2.6: The SUNAUDIODEV module has been deprecated for removal in Python 3.0.

This is a companion module to sunaudiodev which defines useful symbolic constants like MIN_GAIN, MAX_GAIN, SPEAKER, etc. The names of the constants are the same names as used in the C include file <sun/audioio.h>, with the leading string AUDIO_ stripped.

Previous topic

40. SunOS Specific Services

Next topic

41. Undocumented Modules

This Page

- Show Source

Navigation

- index
- modules |
- next |
- previous |
- Python v2.6.4 documentation »
- The Python Standard Library »
- 40. SunOS Specific Services »

© Copyright 1990-2010, Python Software Foundation.
The Python Software Foundation is a non-profit corporation. Please donate.
Last updated on Feb 26, 2010. Created using Sphinx 0.6.3.