NAME

axfer−transfer − transferrer of audio data frame for sound devices and nodes.

SYNOPSIS

axfer transfer *direction* [*common−options*] [*backend−options*] [*filepath*]

axfer transfer *direction* [*common−options*] [*backend−options*] *−I | −−separate−channels filepath ...*

direction = **capture | playback**

common−options = (read *OPTIONS* section)

backend−options = (read *OPTIONS* section)

filepaths = (read *OPTIONS* section)

DESCRIPTION

The **transfer** subcommand of **axfer** performs transmission of audio data frames for devices available in supported backends. This program is essentially designed to use alsa−lib APIs (libasound backend) to handle sound devices supported by Linux sound subsystem (ALSA).

OPTIONS

Direction

capture

Operates for capture transmission.

playback

Operates for playback transmission.

Filepath

Filepath is handled as a path relative to current working directory of run time if it's not full path from root directory.

The standard input or output is used if filepath is not specified or given as '−' .

For playback transmission, container format of given *filepath* is detected automatically and metadata is used for parameters of sample format, channels, rate, duration. If nothing detected, content of given file path is handled as raw data. In this case, the parameters should be indicated as options.

Multiple *filepaths* are allowed with *−I | −−separate−channels* option. In this case, standard input and output is not available. The same *filepath* is not allowed except for paths listed below:

- /dev/null
- /dev/zero
- /dev/full
- /dev/random
- /dev/urandom

Common options

−h, −−help

Print help messages and finish run time.

−q, −−quiet
Quiet mode. Suppress messages (not sound :))

−v, −−verbose
Verbose mode. Runtime dumps supplemental information according to the number of this option given in command line.

−d, −−duration=#
Interrupt after # seconds. A value of zero means infinity. The default is zero, so if this option is omitted then the transmission process will run until it is killed. Either *−d* or *−s* option is available exclusively.

−s, −−samples=#
Interrupt after transmission of # number of data frames. A value of zero means infinity. The default is zero, so if this options is omitted then the transmission process will run until it is killed. Either *−d* or *−s* option is available exclusively.

−f, −−format=FORMAT
Indicate format of audio sample. This is required for capture transmission, or playback transmission with files including raw audio data.

Available sample format is listed below:
- [S8|U8|S16|U16|S32|U32][_LE|_BE]
- [S24|U24][_LE|_BE]
- FLOAT[_LE|_BE]
- FLOAT64[_LE|_BE]
- IEC958_SUBFRAME[_LE|_BE]
- MU_LAW
- A_LAW
- [S20|U20][_LE|_BE]
- [S24|U24][_3LE|_3BE]
- [S20|U20][_3LE|_3BE]
- [S18|U18][_3LE|_3BE]
- DSD_U8
- DSD_[U16|U32][_LE|_BE]

If endian−ness is omitted, host endian−ness is used.

Some special formats are available:
- cd (16 bit little endian, 44100, stereo) [= −f S16_LE −c 2 −r 44100]
- cdr (16 bit big endian, 44100, stereo) [= −f S16_BE −c 2 −f 44100]
- dat (16 bit little endian, 48000, stereo) [= −f S16_LE −c 2 −r 48000]

If omitted, *U8* is used as a default. Actual available formats are restricted by each transmission backend.

Unavailable sample format is listed below. These format has size of data frame unaligned to byte unit.

- IMA_ADPCM
- MPEG
- GSM
- SPECIAL

- G723_24
- G723_24_1B
- G723_40
- G723_40_1B

−c, −−channels=#

Indicate the number of audio data samples per frame. This is required for capture transmission, or playback transmission with files including raw audio data. The value should be between *1 to 256* . If omitted, *1* is used as a default.

−r, −−rate=#

Indicate the number of audio data frame per second. This is required for capture transmission, or playback transmission with files including raw audio data. If the value is less than *1000* , it's interpreted by *kHz* unit. The value should be between *2000* and *192000* . If omitted, *8000* is used as a default.

−t, −−file−type=TYPE

Indicate the type of file. This is required for capture transmission. Available types are listed below:
- wav: Microsoft/IBM RIFF/Wave format
- au, sparc: Sparc AU format
- voc: Creative Tech. voice format
- raw: raw data

When nothing is indicated, for capture transmission, the type is decided according to suffix of *filepath* , and *raw* type is used for fallback.

−I, −−separate−channels

Indicate this option when several files are going to be handled. For capture transmission, if one filepath is given as *filepath* , a list of *filepaths* is generated in a formula '<filepath>−<sequential number>[.suffix]'. The suffix is omitted when raw format of container is used.

−−dump−hw−params

Dump hardware parameters and finish run time if backend supports it.

−−xfer−backend=BACKEND

Select backend of transmission from a list below. The default is libasound.
- libasound
- libffado (optional if compiled)

Backend options for libasound

−D, −−device=NODE

This option is used to select PCM node in libasound configuration space. Available nodes are listed by *pcm* operation of *list* subcommand.

−N, −−nonblock

With this option, PCM substream is opened in non−blocking mode. When audio data frame is not available in buffer of the PCM substream, I/O operation immediately returns without blocking process. This option implicitly uses *−−waiter−type* option as well to prevent heavy consumption of CPU time.

−M, −−mmap

With this option, audio data frame is processed directly in buffer of PCM substream if selected node supports this operation. Without the option, temporary buffers are used to copy audio data frame for buffer of PCM substream. This option implicitly uses *−−waiter−type* option as well to prevent heavy consumption of CPU time.

−F, −−period−size=#

This option configures given value to *period_size* hardware parameter of PCM substream. The parameter indicates the number of audio data frame per period in buffer of the PCM substream. Actual number is decided as a result of interaction between each implementation of PCM plugin chained from the selected PCM node, and in−kernel driver or PCM I/O plugins.

Ideally, the same amount of audio data frame as the value should be handled in one I/O operation. Actually, it is not, depending on implementation of the PCM plugins, in−kernel driver, PCM I/O plugins and scheduling model. For 'hw' PCM plugin in 'irq' scheduling model, the value is used to decide intervals of hardware interrupt, thus the same amount of audio data frame as the value is expected to be available for one I/O operation.

−−period−time=#

This option configures given value to *period_time* hardware parameter of PCM substream. This option is similar to *−−period−size* option, however its unit is micro−second.

−B, −−buffer−size=#

This option configures given value to *buffer_size* hardware parameter of PCM substream. The parameter indicates the number of audio data frame in buffer of PCM substream. Actual number is decided as a result of interaction between each implementation of PCM plugin chained from the selected PCM node, and in−kernel driver or PCM I/O plugins.

Ideally, this is multiples of the number of audio data frame per period, thus the size of period. Actually, it is not, depending on implementation of the PCM plugins, in−kernel driver and PCM I/O plugins.

−−buffer−time=#

This option configures given value to *buffer_time* hardware parameter of PCM substream. This option is similar to *−−buffer−size* option, however its unit is micro−second.

−−waiter−type=TYPE

This option indicates the type of waiter for event notification. At present, four types are available; *default* , *select* , *poll* and *epoll* . With *default* type, 'snd_pcm_wait()' is used. With *select* type, 'select(2)' system call is used. With *poll* type, 'poll(2)' system call is used. With *epoll* type, Linux−specific 'epoll(7)' system call is used.

This option should correspond to one of *−−nonblock* or *−−mmap* options, or *timer* value of *−−sched−model* option. Neither this option nor *−−test−nowait* is available at the same time.

−−sched−model=MODEL

This option selects scheduling model for process of this program. One of *irq* or *timer* is available. In detail, please read 'SCHEDULING MODEL' section.

When nothing specified, *irq* model is used.

−A, −−avail−min=#

This option configures given value to *avail−min* software parameter of PCM substream. In blocking mode, the value is used as threshold of the number of available audio data frames in buffer of PCM substream to wake up process blocked by I/O operation. In non−blocking mode, any I/O operation returns −EAGAIN untill the available number of audio data frame reaches the threshold.

This option has an effect in cases neither *−−mmap* nor *timer* value of *−−sched−model* option is used.

−R, −−start−delay=#

This option configures given value to *start_threshold* software parameter of PCM substream. The value is used as threshold to start PCM substream automatically. At present, this option has an effect in cases neither *−−mmap* nor *timer* value of *−−sched−model* option is used.

For playback transmission, when the number of accumulated audio data frame in buffer of PCM substream to which this program writes out reaches the threshold, the PCM substream starts automatically without an explicit call of *snd_pcm_start()* to the PCM substream.

For capture transmission, this option is useless. The number of accumulated audio data frame is not increased without an explicit call of *snd_pcm_start()* to the PCM substream.

This option has an effect in cases neither *−−mmap* nor *timer* value of *−−sched−model* option is used.

−T, −−stop−delay=#

This option configures given value to *stop_threshold* software parameter of PCM substream. The value is used as threshold to stop PCM substream automatically. At present, this option has an effect in cases neither *−−mmap* nor *timer* value of *−−sched−model* option is used.

For capture transmission, when the number of accumulated audio data frame in buffer of PCM substream to which a driver or alsa−lib PCM plugins write reaches the threshold, the PCM substream stops automatically without an explicit call of *snd_pcm_stop()* to the PCM substream. This is a case that this program leaves the audio data frames without reading for a while.

For playback transmission, when the number available audio data frame in buffer of PCM substream from which a driver or alsa−lib PCM plugins read reaches the threshold, the PCM substream stops automatically without an explicit call of *snd_pcm_stop()* to the PCM substream. This is a case that this program leaves the audio data frames without writing for a while.

This option has an effect in cases neither *−−mmap* nor *timer* value of *−−sched−model* option is used.

−−disable−resample

This option has an effect for 'plug' plugin in alsa−lib to suppress conversion of sampling rate for audio data frame.

−−disable−channels

This option has an effect for 'plug' plugin in alsa−lib to suppress conversion of channels for audio data frame.

−−disable−format

This option has an effect for 'plug' plugin in alsa−lib to suppress conversion of sample format for audio data frame.

−−disable−softvol

This option has an effect for 'softvol' plugin in alsa−lib to suppress conversion of samples for audio data frame via additional control element.

−−fatal−errors

This option suppresses recovery operation from XRUN state of running PCM substream, then process of this program is going to finish as usual.

−−test−nowait

This option disables any waiter for I/O event notification. I/O operations are iterated till any of audio data frame is available. The option brings heavy load in consumption of CPU time.

Backend options for libffado

This backend is automatically available when configure script detects *ffado_streaming_init()* symbol in libffado shared object.

−p, −−port=#

This option uses given value to decide which 1394 OHCI controller is used to communicate. When Linux system has two 1394 OHCI controllers, *0* or *1* are available. Neither this option nor *−g* is available at the same time. If nothing specified, libffado performs to communicate to units on IEEE 1394 bus managed by all of 1394 OHCI controller available in Linux system.

−n, −−node=#

This option uses given value to decide which unit is used to communicate. This option requires *−p* option to indicate which 1394 OHCI controller is used to communicate to the specified unit.

−g, −−guid=HEXADECIMAL

This option uses given value to decide a target unit to communicate. The value should be prefixed with '0x' and consists of hexadecimal literal letters (0−9, a−f, A−F). Neither this option nor *−p* is available at the same time. If nothing specified, libffado performs to communicate to units on

IEEE 1394 bus managed by all of 1394 OHCI controller available in Linux system.

−−frames−per−period=#

This option uses given value to decide the number of audio data frame in one read/write operation. The operation is blocked till the number of available audio data frame exceeds the given value. As a default, 512 audio data frames is used.

−−periods−per−buffer=#

This option uses given value to decide the size of intermediate buffer between this program and libffado. As a default, 2 periods per buffer is used.

−−slave

This option allows this program to run slave mode. In this mode, libffado adds unit directory into configuration ROM of 1394 OHCI controller where Linux system runs. The unit directory can be found by the other node on the same bus. Linux system running on the node can transfer isochronous packet with audio data frame to the unit. This program can receive the packet and demultiplex the audio data frame.

−−snoop

This option allows this program to run snoop mode. In this mode, libffado listens isochronous channels to which device transfers isochronous packet. When isochronous communication starts by any unit on the same bus, the packets can be handled by this program.

−−sched−priority=#

This option executes *pthread_setschedparam()* in a call of *ffado_streaming_init()* to configure scheduling policy and given value as its priority for threads related to isochronous communication. The given value should be within *RLIMIT_RTPRIO* parameter of process. Please read *getrlimit(2)* for details.

POSIX SIGNALS

During transmission, *SIGINT* and *SIGTERM* will close handled files and PCM substream to be going to finish run time.

SIGTSTP will suspend PCM substream and *SIGCONT* will resume it. No XRUNs are expected. With libffado backend, the suspend/resume is not supported and runtime is aboeted immediately.

The other signals perform default behaviours.

EXAMPLES

```
$ axfer transfer playback -d 1 something
```

The above will transfer audio data frame in 'something' file for playback during 1 second. The sample format is detected automatically as a result to parse 'something' as long as it's compliant to one of Microsoft/IBM RIFF/Wave, Sparc AU, Creative Tech. voice formats. If nothing detected, *−r* , *−c* and *−f* should be given, or *−f* should be given with special format.

```
$ axfer transfer playback -r 22050 -c 1 -f S16_LE -t raw something
```

The above will transfer audio data frame in 'something' file including no information of sample format, as sample format of 22050 Hz, monaural, signed 16 bit little endian PCM for playback. The transmission continues till catching *SIGINT* from keyboard or *SIGTERM* by *kill(1)* .

```
$ axfer transfer capture −d 10 −f cd something.wav
```

The above will transfer audio data frame to 'something.wav' file as sample format of 44.1 kHz, 2 channels, signed 16 bit little endian PCM, during 10 seconds. The file format is Microsoft/IBM RIFF/Wave according to suffix of the given *filepath* .

```
$ axfer transfer capture −s 1024 −r 48000 −c 2 −f S32_BE −I −t au channels
```

The above will transfer audio data frame as sample format of 48.0 kHz, 2 channels, signed 32 bit big endian PCM for 1,024 number of data frames to files named 'channels−1.au' and 'channels−2.au'.

SCHEDULING MODEL

In a design of ALSA PCM core, runtime of PCM substream supports two modes; *period−wakeup* and *no−period−wakeup.* These two modes are for different scheduling models.

IRQ−based scheduling model

As a default, *period−wakeup* mode is used. In this mode, in−kernel drivers should operate hardware to generate periodical notification for transmission of audio data frame. The interval of notification is equivalent to the same amount of audio data frame as one period of buffer, against actual time.

In a handler assigned to the notification, a helper function of ALSA PCM core is called to update a position to head of hardware transmission, then compare it with a position to head of application operation to judge overrun/underrun (XRUN) and to wake up blocked processes.

For this purpose, hardware IRQ of controller for serial audio bus such as Inter−IC sound is typically used. In this case, the controller generates the IRQ according to transmission on the serial audio bus. In the handler assigned to the IRQ, direct media access (DMA) transmission is requested between dedicated host memory and device memory.

If target hardware doesn't support this kind of mechanism, the periodical notification should be emulated by any timer; e.g. hrtimer, kernel timer. External PCM plugins generated by PCM plugin SDK in alsa−lib should also emulate the above behaviour.

In this mode, PCM applications are programmed according to typical way of I/O operations. They execute blocking system calls to read/write audio data frame in buffer of PCM substream, or blocking system calls to wait until any audio data frame is available. In *axfer* , this is called *IRQ−based* scheduling model and a default behaviour. Users can explicitly configure this mode by usage of *−−sched−model* option with *irq* value.

Timer−based scheduling model

The *no−period−wakeup* mode is an optional mode of runtime of PCM substream. The mode assumes a specific feature of hardware and assist of in−kernel driver and PCM applications. In this mode, in−kernel drivers don't operate hardware to generate periodical notification for transmission of audio data frame. The hardware should automatically continue transmission of audio data frame without periodical operation of the drivers; e.g. according to auto−triggered DMA transmission, a chain of registered descriptors.

In this mode, nothing wakes up blocked processes, therefore PCM applications should be programmed without any blocking operation. For this reason, this mode is enabled when the PCM applications explicitly configure hardware parameter to runtime of PCM substream, to prevent disorder of existing applications. Additionally, nothing maintains timing for transmission of audio data frame, therefore the PCM

applications should voluntarily handle any timer to queue audio data frame in buffer of the PCM substream for lapse of time. Furthermore, instead of driver, the PCM application should call a helper function of ALSA PCM core to update a position to head of hardware transmission and to check XRUN.

In *axfer* , this is called *timer−based* scheduling model and available as long as hardware/driver assists *no−period−wakeup* runtime. Users should explicitly set this mode by usage of *−−sched−model* option with *timer* value.

In the scheduling model, PCM applications need to care of available space on PCM buffer by lapse of time, typically by yielding CPU and wait for rescheduling. For the yielding, timeout is calculated for preferable amount of PCM frames to process. This is convenient to a kind of applications, like sound servers. when an I/O thread of the server wait for the timeout, the other threads can process audio data frames for server clients. Furthermore, with usage of rewinding/forwarding, applications can achieve low latency between transmission position and handling position even if they uses large size of PCM buffers.

Advantages and issues

Ideally, timer−based scheduling model has some advantages than IRQ−based scheduling model. At first, no interrupt context runs for PCM substream. The PCM substream is handled in any process context only. No need to care of race conditions between IRQ and process contexts. This reduces some concerns for some developers of drivers and applications. Secondary, CPU time is not used for handlers on the interrupt context. The CPU time can be dedicated for the other tasks. This is good in a point of Time Sharing System. Thirdly, hardware is not configured to generate interrupts. This is good in a point of reduction of overall power consumption possibly.

In either scheduling model, the hardware should allow drivers to read the number of audio data frame transferred between the dedicated memory and the device memory for audio serial bus. However, in timer−based scheduling model, fine granularity and accuracy of the value is important. Actually hardware performs transmission between dedicated memory and device memory for a small batch of audio data frames or bytes. In a view of PCM applications, the granularity in current transmission is required to decide correct timeout for each I/O operation. As of Linux kernel v4.21, ALSA PCM interface between kernel/userspace has no feature to report it.

COMPATIBILITY TO APLAY

The **transfer** subcommand of **axfer** is designed to keep compatibility to aplay(1). However some options below are not compatible due to several technical reasons.

−I, −−separate−channels

This option is supported just for files to store audio data frames corresponding to each channel. In aplay(1) implementation, this option has an additional effect to use PCM buffer aligned to non−interleaved order if a target device supports. As of 2018, PCM buffer of non−interleaved order is hardly used by sound devices.

−A, −−avail−min=#

This option indicates threshold to wake up blocked process in a unit of audio data frame. Against aplay(1) implementation, this option has no effect with *−−mmap* option as well as *timer* of *−−sched−model* option.

−R, −−start−delay=#

This option indicates threshold to start prepared PCM substream in a unit of audio data frame. Against aplay(1) implementation, this option has no effect with *−−mmap* option as well as *timer* of *−−sched−model* option.

−T, −−stop−delay=#

This option indicates threshold to stop running PCM substream in a unit of audio data frame. Against aplay(1) implementation, this option has no effect with *−−mmap* option as well as *timer* of *−−sched−model* option.

−−max−file−time=#

This option is unsupported. In aplay(1) implementation, the option has an effect for capture transmission to save files up to the same number of data frames as the given value by second unit, or the maximum number of data frames supported by used file format. When reaching to the limitation, used file is closed, then new file is opened and audio data frames are written. However, this option requires extra handling of files and shall increase complexity of main loop of axfer.

−−use−strftime=FORMAT

This option is unsupported. In aplay(1) implementation, the option has an effect for capture transmission to generate file paths according to given format in which some extra formats are available as well as formats supported by strftime(3). However, this option requires extra string processing for file paths and it's bothersome if written in C language.

−−process−id−file=FILEPATH

This option is unsupported. In aplay(1) implementation, the option has an effect to create a file for given value and write out process ID to it. This file allows users to get process ID and send any POSIX signal to aplay process. However, this idea has some troubles for file locking when multiple aplay processes run with the same file.

−V, −−vumeter=TYPE

This option is not supported at present. In aplay(1) implementation, this option has an effect to occupy stdout with some terminal control characters and display vumeter for monaural and stereo channels. However, some problems lay; this feature is just for audio data frames with PCM format, this feature brings disorder of terminal after aborting, stdout is not available for pipeline.

−i, −−interactive

This option is not supported at present. In aplay(1) implementation, this option has an effect to occupy stdin for key input and suspend/resume PCM substream according to pushed enter key. However, this feature requires an additional input handling in main loop and leave bothersome operation to maintain PCM substream.

−m, −−chmap=CH1,CH2,...

ALSA PCM core and control core doesn't support this feature, therefore remapping should be done in userspace. This brings overhead to align audio data frames, especially for mmap operation. Furthermore, as of alsa-lib v1.1.8, some plugins don't support this feature expectedly, thus this option is a lack of transparent operation. At present, this option is not supported yet not to confuse users.

SIGTSTP, SIGCONT

This performs suspend/resume of PCM substream. In aplay(1) implementation, these operations bring XRUN state to the substream, and suspend/resume is done in interactive mode in the above. Some developers use the signal for recovery test from XRUN. At present, no alternative is supported for the test.

SIGUSR1

This is not supported. In aplay(1) implementation, this signal is assigned to a handler to close a current file to store audio data frame and open a new file to continue processing. However, as well as −−*max−file−time* option, this option should increase complexity of main loop of axfer.

DESIGN

Modular structure

This program consists of three modules; *xfer* , *mapper* and *container* . Each module has an abstraction layer to enable actual implementation.

```
        --------    ----------    -------------
device <-> | xfer | <-> | mapper | <-> | container | <-> file
        --------    ----------    -------------
         libasound   single        wav
         libffado    multiple      au
                                   voc
                                   raw
```

The *xfer* module performs actual transmission to devices and nodes. The module can have several transmission backends. As a default backend, *libasound* backend is used to perform transmission via alsa−lib APIs. The module allows each backend to parse own command line options.

The *container* module performs to read/write audio data frame via descriptor for file/stream of multimedia container or raw data. The module automatically detect type of multimedia container and parse parameters in its metadata of data header. At present, three types of multimedia containers are supported; Microsoft/IBM RIFF/Wave (*wav*), Sparc AU (*au*) and Creative Technology voice (*voc*). Additionally, a special container is prepared for raw audio data (*raw*).

The *mapper* module handles buffer layout and alignment for transmission of audio data frame. The module has two implementations; *single* and *multiple* . The *single* backend uses one container to construct the buffer. The *multiple* backend uses several containers to construct it.

Care of copying audio data frame

Between the *xfer* module and *mapper* module, a pointer to buffer including audio data frames is passed. This buffer has two shapes for interleaved and non−interleaved order. For the former, the pointer points to one buffer. For the latter, the pointer points to an array in which each element points to one buffer. Between the *mapper* module and *container* module, a pointer to one buffer is passed because supported media containers including raw type store audio data frames in interleaved order.

In passing audio data frame between the modules, axfer is programmed to avoid copying between a buffer to another buffer as much as possible. For example, in some scenarios below, no copying occurs between modules.

- xfer(mmap/interleaved), mapper(single), container(any)
- xfer(mmap/non−interleaved), mapper(multiple), containers(any)

Unit test

For each of the *mapper* and *container* module, unit test is available. To run the tests, execute below command:

$ make test

Each test iterates writing to file and reading to the file for many times and it takes long time to finish. Please

take care of the execution time if running on any CI environment.

SEE ALSO

axfer(1), axfer−list(1), alsamixer(1), amixer(1)

AUTHOR

Takashi Sakamoto <o−takashi@sakamocchi.jp>