Notes on status of CoreAudio Implementation of PortAudio

Document Last Updated December 9, 2005

There are currently two implementations of PortAudio for Mac Core Audio.

The original is in pa\_mac\_core\_old.c, and the newer, default implementation

is in pa\_mac\_core.c.

Only pa\_mac\_core.c is currently developed and supported as it uses apple's

current core audio technology. To select use the old implementation, replace

pa\_mac\_core.c with pa\_mac\_core\_old.c (eg. "cp pa\_mac\_core\_auhal.c

pa\_mac\_core.c"), then run configure and make as usual.

-------------------------------------------

Notes on Newer/Default AUHAL implementation:

by Bjorn Roche

Last Updated December 9, 2005

Principle of Operation:

This implementation uses AUHAL for audio I/O. To some extent, it also

operates at the "HAL" Layer, though this behavior can be limited by

platform specific flags (see pa\_mac\_core.h for details). The default

settings should be reasonable: they don't change the SR of the device and

don't cause interruptions if other devices are using the device.

Major Software Elements Used: Apple's HAL AUs provide output SR

conversion transparently, however, only on output, so this

implementation uses AudioConverters to convert the sample rate on input.

A PortAudio ring buffer is used to buffer input when sample rate

conversion is required or when separate audio units are used for duplex

IO. Finally, a PortAudio buffer processor is used to convert formats and

provide additional buffers if needed. Internally, interleaved floating

point data streams are used exclusively - the audio unit converts from

the audio hardware's native format to interleaved float PCM and

PortAudio's Buffer processor is used for conversion to user formats.

Simplex Input: Simplex input uses a single callback. If sample rate

conversion is required, a ring buffer and AudioConverter are used as

well.

Simplex output: Simplex output uses a single callback. No ring buffer or

audio converter is used because AUHAL does its own output SR conversion.

Duplex, one device (no SR conversion): When one device is used, a single

callback is used. This achieves very low latency.

Duplex, separate devices or SR conversion: When SR conversion is

required, data must be buffered before it is converted and data is not

always available at the same times on input and output, so SR conversion

requires the same treatment as separate devices. The input callback

reads data and puts it in the ring buffer. The output callback reads the

data off the ring buffer, into an audio converter and finally to the

buffer processor.

Platform Specific Options:

By using the flags in pa\_mac\_core.h, the user may specify several options.

For example, the user can specify the sample-rate conversion quality, and

the extent to which PA will attempt to "play nice" and to what extent it

will interrupt other apps to improve performance. For example, if 44100 Hz

sample rate is requested but the device is set at 48000 Hz, PA can either

change the device for optimal playback ("Pro" mode), which may interrupt

other programs playing back audio, or simple use a sample-rate coversion,

which allows for friendlier sharing of the device ("Play Nice" mode).

Additionally, the user may define a "channel mapping" by calling

paSetupMacCoreChannelMap() on their stream info structure before opening

the stream with it. See below for creating a channel map.

Known issues:

- Buffering: No buffering beyond that provided by core audio is provided

except where absolutely needed for the implementation to work. This may cause

issues with large framesPerBuffer settings and it also means that no additional

latency will be provided even if a large latency setting is selected.

- Latency: Latency settings are generally ignored. They may be used as a

hint for buffer size in paHostFramesPerBufferUnspecified, or the value may

be used in cases where additional buffering is needed, such as doing input and

output on seperate devices. Latency settings are always automatically bound

to "safe" values, however, so setting extreme values here should not be

an issue.

- Buffer Size: paHostFramesPerBufferUnspecified and specific host buffer sizes

are supported. paHostFramesPerBufferUnspecified works best in "pro" mode,

where the buffer size and sample rate of the audio device is most likely

to match the expected values. In the case of paHostFramesPerBuffer, an

appropriate framesPerBuffer value will be used that guarantees minimum

requested latency if that's possible.

- Timing info. It reports on stream time, but I'm probably doing something

wrong since patest\_sine\_time often reports negative latency numbers. Also,

there are currently issues with some devices whehn plugging/unplugging

devices.

- xrun detection: The only xrun detection performed is when reading

and writing the ring buffer. There is probably more that can be done.

- abort/stop issues: stopping a stream is always a complete operation,

but latency should be low enough to make the lack of a separate abort

unnecessary. Apple clarifies its AudioOutputUnitStop() call here:

http://lists.apple.com/archives/coreaudio-api/2005/Dec/msg00055.html

- blocking interface: should work fine.

- multichannel: It has been tested successfully on multichannel hardware

from MOTU: traveler and 896HD. Also Presonus firepod and others. It is

believed to work with all Core Audio devices, including virtual devices

such as soundflower.

- sample rate conversion quality: By default, SR conversion is the maximum

available. This can be tweaked using flags pa\_mac\_core.h. Note that the AU

render quyality property is used to set the sample rate conversion quality

as "documented" here:

http://lists.apple.com/archives/coreaudio-api/2004/Jan/msg00141.html

- x86/Universal Binary: Universal binaries can be build.

Creating a channel map:

How to create the map array - Text taken From AUHAL.rtfd :

[3] Channel Maps

Clients can tell the AUHAL units which channels of the device they are interested in. For example, the client may be processing stereo data, but outputting to a six-channel device. This is done by using the kAudioOutputUnitProperty\_ChannelMap property. To use this property:

For Output:

Create an array of SInt32 that is the size of the number of channels of the device (Get the Format of the AUHAL's output Element == 0)

Initialize each of the array's values to -1 (-1 indicates that that channel is NOT to be presented in the conversion.)

Next, for each channel of your app's output, set:

channelMapArray[deviceOutputChannel] = desiredAppOutputChannel.

For example: we have a 6 channel output device and our application has a stereo source it wants to provide to the device. Suppose we want that stereo source to go to the 3rd and 4th channels of the device. The channel map would look like this: { -1, -1, 0, 1, -1, -1 }

Where the formats are:

Input Element == 0: 2 channels (- client format - settable)

Output Element == 0: 6 channels (- device format - NOT settable)

So channel 2 (zero-based) of the device will take the first channel of output and channel 3 will take the second channel of output. (This translates to the 3rd and 4th plugs of the 6 output plugs of the device of course!)

For Input:

Create an array of SInt32 that is the size of the number of channels of the format you require for input. Get (or Set in this case as needed) the AUHAL's output Element == 1.

Next, for each channel of input you require, set:

channelMapArray[desiredAppInputChannel] = deviceOutputChannel;

For example: we have a 6 channel input device from which we wish to receive stereo input from the 3rd and 4th channels. The channel map looks like this: { 2, 3 }

Where the formats are:

Input Element == 0: 2 channels (- device format - NOT settable)

Output Element == 0: 6 channels (- client format - settable)

----------------------------------------

Notes on Original implementation:

by Phil Burk and Darren Gibbs

Last updated March 20, 2002

WHAT WORKS

Output with very low latency, <10 msec.

Half duplex input or output.

Full duplex on the same CoreAudio device.

The paFLoat32, paInt16, paInt8, paUInt8 sample formats.

Pa\_GetCPULoad()

Pa\_StreamTime()

KNOWN BUGS OR LIMITATIONS

We do not yet support simultaneous input and output on different

devices. Note that some CoreAudio devices like the Roland UH30 look

like one device but are actually two different CoreAudio devices. The

Built-In audio is typically one CoreAudio device.

Mono doesn't work.

DEVICE MAPPING

CoreAudio devices can support both input and output. But the sample

rates supported may be different. So we have map one or two PortAudio

device to each CoreAudio device depending on whether it supports

input, output or both.

When we query devices, we first get a list of CoreAudio devices. Then

we scan the list and add a PortAudio device for each CoreAudio device

that supports input. Then we make a scan for output devices.