Notes about WDM-KS host API

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

Status history

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

16th January 2011:

Added support for WaveRT device API (Vista and later) for even lesser

latency support.

10th November 2005:

Made following changes:

\* OpenStream: Try all PaSampleFormats internally if the the chosen

format is not supported natively. This fixed several problems

with soundcards that did not take kindly to using 24-bit 3-byte formats.

\* OpenStream: Make the minimum framesPerHostIBuffer (and framesPerHostOBuffer)

the default frameSize for the playback/recording pin.

\* ProcessingThread: Added a switch to only call PaUtil\_EndBufferProcessing

if the total input frames equals the total output frames

5th September 2004:

This is the first public version of the code. It should be considered

an alpha release with zero guarantee not to crash on any particular

system. So far it has only been tested in the author's development

environment, which means a Win2k/SP2 PIII laptop with integrated

SoundMAX driver and USB Tascam US-428 compiled with both MinGW

(GCC 3.3) and MSVC++6 using the MS DirectX 9 SDK.

It has been most widely tested with the MinGW build, with most of the

test programs (particularly paqa\_devs and paqa\_errs) passing.

There are some notable failures: patest\_out\_underflow and both of the

blocking I/O tests (as blocking I/O is not implemented).

At this point the code needs to be tested with a much wider variety

of configurations and feedback provided from testers regarding

both working and failing cases.

What is the WDM-KS host API?

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

PortAudio for Windows currently has 3 functional host implementations.

MME uses the oldest Windows audio API which does not offer good

play/record latency.

DirectX improves this, but still imposes a penalty

of 10s of milliseconds due to the system mixing of streams from

multiple applications.

ASIO offers very good latency, but requires special drivers which are

not always available for cheaper audio hardware. Also, when ASIO

drivers are available, they are not always so robust because they

bypass all of the standardised Windows device driver architecture

and hit the hardware their own way.

Alternatively there are a couple of free (but closed source) ASIO

implementations which connect to the lower level Windows

"Kernel Streaming" API, but again these require special installation

by the user, and can be limited in functionality or difficult to use.

This is where the PortAudio "WDM-KS" host implementation comes in.

It directly connects PortAudio to the same Kernel Streaming API which

those ASIO bridges use. This avoids the mixing penatly of DirectX,

giving at least as good latency as any ASIO driver, but it has the

advantage of working with ANY Windows audio hardware which is available

through the normal MME/DirectX routes without the user requiring

any additional device drivers to be installed, and allowing all

device selection to be done through the normal PortAudio API.

Note that in general you should only be using this host API if your

application has a real requirement for very low latency audio (<20ms),

either because you are generating sounds in real-time based upon

user input, or you a processing recorded audio in real time.

The only thing to be aware of is that using the KS interface will

block that device from being used by the rest of system through

the higher level APIs, or conversely, if the system is using

a device, the KS API will not be able to use it. MS recommend that

you should keep the device open only when your application has focus.

In PortAudio terms, this means having a stream Open on a WDMKS device.

Usage

-----

To add the WDMKS backend to your program which is already using

PortAudio, you must define PA\_USE\_WDMKS=1 in your build file,

and include the pa\_win\_wdmks\pa\_win\_wdmks.c into your build.

The file should compile in both C and C++.

You will need a DirectX SDK installed on your system for the

ks.h and ksmedia.h header files.

You will need to link to the system "setupapi" library.

Note that if you use MinGW, you will get more warnings from

the DX header files when using GCC(C), and still a few warnings

with G++(CPP).