Audio I/O Software Use Cases

Sophia Poirier (spoirier@apple.com)
Frank Birbacher (frank.birbacher@gmail.com)
Timur Doumler (papers@timur.audio)

Document #: P2054R0
Date: 2020-01-13
Project: Programming Language C++
Audience: SG13

Abstract

This paper lists some use cases that could potentially be supported by a C++ API for audio I/O. Its purpose is to inform a discussion about whether (and in what form) to pursue a proposal to add such an API to the C++ standard library.

Introduction

[P1386R2] proposes to add a standard audio API to the C++ standard library. The proposal consists of a data type to represent raw audio data (based on multi-channel linear PCM), a type to represent an audio I/O device (physical or virtual), and an API to discover such audio I/O devices and to attach audio processing callbacks to them. This model follows the device-based APIs found in cross-platform audio libraries such as JUCE and PortAudio, which are existing practice on professional audio oriented desktop platforms.

A response paper [P1746R1] argues that, while the underlying audio data type is worth exploring, an device-based audio I/O API is not suitable for inclusion in the C++ standard. In particular, mobile platforms such as iOS do not provide direct access to audio devices, and can therefore not be adequately represented by a device-based API. Further, there are concerns about portability, audio routing, policies, and spatial audio.

[P1913R0] explores additional use cases that the author doesn't see supported by the API proposed in [P1386R2], such as interoperability with third-party audio libraries, persistent unique device identifiers, and pausing audio processing.

Before we can proceed with considering a concrete audio I/O API for inclusion in C++, we need to establish consensus on whether or not such a standard audio I/O API should be considered at all, and if so, whether it should be based on audio devices as proposed in [P1386R2] or follow some other design.

This paper attempts to provide a basis for this discussion by collecting different use cases that could potentially be supported by a C++ API for audio I/O. If we could reach an agreement on which use cases should be supported by such an API, this in turn could inform the discussion on the API design.

The approach to start with collecting use cases has recently proven to be productive for Contracts [P1995R0], another contentious C++ proposal. The authors hope that it will be helpful in this case as well.

Consumer Experience

- As an implementer I need to play application provided audio on hardware that was invented after the application was compiled and shipped—for instance on mobile platforms where applications from a store need to run on newer hardware,—using sample types and/or buffer layouts previously unknown, and…
    - …this needs to be as efficient as if these types and layouts were known before.
    - …I'm willing to pay for runtime overhead.

- As an operating system provider I want to pause and resume audio input and output of an application in order to moderate the use of real hardware between multiple applications.

- As an application developer, I want notification of when the operating system interrupts my audio session and pauses output as well as when the OS resumes my audio.

- As an implementer, when the operating system interrupts my audio session, I want to allow the OS the opportunity to appropriately fade out and fade back in, to circumvent audible glitches for the user.

- As an implementer, if there are restrictions on which application is allowed to access the microphone, these should be handled in the manner intended by the operating system, and an audio API should not be able to circumvent them.

- As an implementer, I want to at all times render audio I/O through the operating system designed appropriate input or output audio devices. For example, connecting headphones may supercede using a device's built-in speaker and microphone. I do not need to or want to know when device availability changes, simply for the appropriate I/O to be used, and as needed to avoid audible glitches, for audio fades to automatically be applied during device transitions.

- As an implementer, I want to be able to write software with audio I/O functionality for desktop and mobile platforms using standard library utilities with one implementation that handles all platforms uniformly.

- As an implementer of an audio playback application, I want to output the audio from audio media files. I want for the audio to render out of the system's default audio hardware and to handle potentially spatial audio in the optimal manner for that hardware. I want the audio media's sample rate to convert to match the audio hardware state. I want notification of any audio playback interruptions. I wish to leverage any operating system licensed decoding when required for encoded media files.

- As an implementer of a video playback application, I want to output the audio associated with the video being viewed. I want for the audio to render out of the system's default audio hardware and to handle spatial audio in the optimal manner for that hardware. I want the audio media's sample rate to convert to match the audio hardware state. I want notification of any audio playback interruptions. I wish to leverage any operating system licensed decoding when required for encoded media files. I require enough audio timestamp information to synchronize the render timing of the video.

- As an implementer of a consumer audio creation application, I want to output algorithmically generated audio. I want for the audio to render out of the system's default audio hardware and to handle potentially spatial audio in the optimal manner for that hardware. I want the generated audio's sample rate to convert to match the audio hardware state. I want low latency for a responsive user experience.

- As an implementer of a telecommunications application, I want to receive audio from the system's default audio input hardware and render to the system's default audio input. I want to operate in software at my preferred audio sample rate regardless of the hardware state and have the system provide any necessary sample rate conversion. I want a means to designate my audio session as one that should interrupt any other application audio on the machine. If I support video telecommunication, I require enough audio output timestamp information to synchronize the render timing of the video.

- As an implementer of an application without audio-centered features, I wish to occasionally output sounds associated with application events by playing back short audio files. I want for the audio to render out of the system's default audio hardware. I want the audio media's sample rate to convert to match the audio hardware state. I want to "fire and forget" upon initiated playback and need no notification of audio interruptions.

- As an implementer of a video game, I want to output algorithmically generated audio. I want to play back static soundtrack music that renders out of the system's default audio hardware and to handle potentially spatial audio in the optimal manner for that hardware. I want the generated audio's sample rate to convert to match the audio hardware state. Latency is not a concern for sound track audio. I also want to output discrete audio sounds associated with game and user events. I require low latency for all event sounds for a responsive user experience. I wish to position these event sounds in three-dimensional space to render appropriately on the system's audio hardware without concern for the physical channel layout of the audio hardware.

- As an implementer of an audio note creation application, I want to record the audio from the microphone input device. I want to receive audio at my desired sample rate for data persistence regardless of the device's physical sample rate.

- As an implementer of an environmental audio analysis application, I want to receive the audio data from the microphone input device. I want to receive audio at my desired sample rate for my audio algorithms regardless of the device's physical sample rate. I do not need low latency or even to receive the audio data within a real-time context.

- As an implementer of an audio communication application on a desktop platform, I want my user to choose which audio I/O device the app should be used. I want to be able to choose one that is different from the audio device currently selected in the OS settings. (For example, the app should use the connected headset for a call while all other apps and system sounds keep using the speakers for their audio output.) When the headset is connected, I want the user to be notified, and to be offered a choice between using the headset or to keep using the speakers.

- As an application developer I want to allow my users to select an audio device from a list of all devices, as far as supported by my application, where the list has human readable names or descriptions such that the choice can be persisted and programmatically selected again on the next run of my application.

- As an application developer I want to avoid the hassle around checking for "no devices" in a number of places: when instantiating my audio related classes, when persisting a choice to disk, when loading everything from disk, and possibly when running tests.

Professional Experience

- As an implementer of a professional music production application, I want direct control over the audio hardware. I want to query its channel and format capabilities. I want to be able to configure its sample rate and rendering block size. I want to render audio samples directly to and from input and output audio hardware channels without any converting mediation from the operating system. I want low latency for a responsive user experience. I want notification of any audio playback interruptions.

- As an implementer of a video production application, I want to output the audio associated with the video being created or edited. I want for the audio to render out of the system's default audio hardware and to handle spatial audio in the optimal manner for that hardware. I want the audio media's sample rate to convert to match the audio hardware state. I want notification of any audio playback interruptions. I require enough audio timestamp information to synchronize the render timing of the video.

- As an audio engineer, I want to choose which audio backend supported by my OS should be used for audio I/O on a concrete sound card. For example, on Windows, sound card 1 might support ASIO, which offers low latency, but sound card 2 needs to use WASAPI instead because the vendor did not supply an ASIO driver.

- As an audio engineer, when working with multi-channel audio I/O hardware, I want to choose which of the available channels should be used for my app's audio input and output. I want to use a channel mask to specify the channel mapping. I want to distinguish between different channel layouts, even if they have the same number of channels (for example, LRC vs. LCR layouts for three channels).

- As an audio algorithm developer, to reduce unnecessary code duplication, I want to write an algorithm manipulating raw audio data using a single common numeric type for audio samples, such as `float`, and I want this algorithm to work across all platforms and audio devices.

- As an audio algorithm developer, to optimise performance and reduce audible artifacts, I want to be notified what the best numeric format and endianness is to use for the current audio I/O device, and I want to select an appropriate version of my algorithm that I programmed to work optimally for this specific numeric format and endianness.

- As an audio developer for embedded platforms, I want to be able to program against an audio I/O device that does not support floating point operations.

Library

- As a library developer, I want to provide audio data processing or handling APIs with data types that are consistent between other C++ audio libraries.

- As a library developer, I want to provide audio file rendering and recording APIs that leverage common standard library mechanisms for audio hardware I/O to allow reliable use across platforms.

- As a modern C++ developer writing audio processing code, I want to be able to code against an audio library that is easy to install, learn, and use, and does not have any additional dependencies.

Education

- As a student learning about audio and C++, I want to be able to easily write code that algorithmically generates audio, algorithmically processes existing audio, plays back existing audio files, records microphone input, and saves it into an audio file, given just the tools that come out of the box with any modern C++ compiler.

References

- [P1386R2] http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p1386r2.pdf
- [P1746R1] http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p1746r1.html
- [P1913R0] http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p1913r0.pdf
- [P1995R0] http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p1995r0.html