Audio I/O Software Use Cases

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

Document #: P2054R1
Date: 2020-03-02
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, a 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.

Generic applications with audio input and/or output

- **[dev.generic.output_interruption_notification]** As a developer of an application with continuous audio output, I want notification of when the operating system interrupts my audio session and pauses output as well as when the OS resumes my audio, e.g., when a phone call interrupts all other audio.

- **[dev.generic.output_interruption_fade]** As a developer of an application with continuous audio output, 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.

- **[dev.generic.input_interruption_notification]** As a developer of an application that records audio input, I want notification of when the operating system interrupts my audio recording session.

- **[dev.generic.input_access]** As a developer of an application that records audio input, if there are restrictions on which application is allowed to access the microphone, I want these to be handled in the manner intended by the operating system.

- **[dev.generic.output_access]** As a developer of an application with audio output, I wish to be notified if the OS mutes the output device to which I am rendering audio.

- **[dev.generic.inter_app_audio]** As a developer of an application that records audio, I wish to be able to access another application’s audio output as my audio input source.

- **[dev.generic.input_retain_output]** As a developer of an application that records audio input, I want to persist any other application's audio output during my audio recording session, e.g. start a recording from my mobile's microphone while listening to music on my earphones.

- **[dev.generic.spatial_virtualization]** As a developer of an application with continuous audio output, I want to output the audio from audio media files. I want 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.

- **[dev.generic.follow_default]** As a developer of an application with audio input and/or output, I want to at all times render audio I/O through the operating system designated input or output audio devices. For example, connecting headphones may supersede the previous speaker routing. I do not need to or want to know when device availability changes, simply for the appropriate I/O to be used. As necessary to avoid audible glitches, audio fades should automatically be applied during device transitions.

- **[dev.generic.cross_platform]** As a developer of an application with audio input and/or output, 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.

- **[dev.generic.event_sounds]** As a developer of an application without audio-centered features, I wish to occasionally output sounds associated with application events by playing back short audio files (e.g., sent or received email sound events). I want 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.

- **[dev.generic.device_selection]** As a developer of an application with audio input and/or output, 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.

- **[dev.generic.always_a_device]** As a developer of an application with audio input and/or output, I want to avoid the hassle of checking for "no audio I/O devices available" in a number of contexts: when instantiating my audio related classes, when persisting a choice to disk, when loading everything from disk, and possibly when running tests.

- **[dev.generic.decode_common_formats]** As a developer of an application with audio output, I want to be able to open audio files on disk using common (compressed or uncompressed) audio file formats such as WAV, AIFF, MP3, and FLAC, and decode them into raw audio data which I can then manipulate and play back.

- **[dev.generic.encode_common_formats]** As a developer of an application with audio input, I want to be able to encode recorded raw audio data into one of the common (compressed or uncompressed) audio file formats such as WAV, AIFF, MP3, and FLAC, and to write it into an audio file on disk.

Media playback applications

- **[dev.playback.encoded_packets]** As a developer of an audio media playback application, I wish to be able to represent audio in a data type that preserves encoded packets from encoded audio formats. Encoded formats can include compressed packet formats like MP3 or FLAC, proprietary encodings for copy protection, spatial encodings like Dolby Pro Logic or Dolby Atmos, etc. The goal is to be able to pass along the data as opaque blocks or bitstreams, not that a standard library would be expected to encode or decode these. In cases where the OS holds licensing to decode or encode, I expect to be able to leverage that functionality. In cases of copy-protected encoding (e.g., Blu-ray audio), decoding is done directly to audio output, which is only possible by the OS.

- **[dev.playback.media_category]** As a developer of an audiobook playback application, I want to be able to identify my audio stream to the operating system as "spoken word" material so that audio interruptions may occur with more appropriate behavior (such as pausing my audio stream during the interruption rather than ducking its volume).

- **[dev.playback.video_general]** As a developer of a video playback application, I want to output the audio associated with the video being viewed, created, or edited. I want 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.

- **[dev.playback.video_audio_sync]** As a developer of a video playback application, I require enough audio timestamp information to synchronize the render timing of the video and compensate for any perceptual latency.

Telecommunication applications

- **[dev.comm.interruptor]** As a developer 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 output. I want a means to designate my audio session as one that should interrupt any other application audio on the machine.

- **[dev.comm.device_selection]** As a developer 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 continue using the speakers.

Games

- **[dev.game.soundtrack]** As a developer of a video game, 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. Latency is not a concern for soundtrack audio.

- **[dev.game.events]** As a developer of a video game, I 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.

- **[dev.game.spatial]** As a developer of a video game, I wish to position game event sounds in an abstract three-dimensional space, and for the OS to render that 3D audio image appropriately on the system's audio hardware, without my application requiring knowledge of the physical channel layout of the audio hardware.

- **[dev.game.mixing]** As a developer of a video game, I want to mix soundtrack, dialogue, and discrete game event sounds together before rendering them out of the user's audio hardware.

Audio analysis applications

- **[dev.analysis.fixed_sample_rate]** As a developer 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.

Music production applications

- **[dev.music.consumer]** As a developer of a consumer music creation application, I want to output algorithmically generated audio (e.g., a drum synthesizer). I want the audio to render out of the system's default audio hardware. I want low latency for a responsive user experience.

- **[dev.music.pro]** As a developer of a professional music production application, I want to give my users the ability to choose which audio hardware they want to use for rendering.

I want to give them direct control over that audio hardware. I want the minimal possible latency for a user experience that meets the requirements of live music performance. I want notification of any audio playback interruptions.

- **[dev.music.hardware_settings]** As a developer of a professional music production application, to find the best possible compromise between audio quality, low latency, and CPU load for my platform and audio hardware driver, I want to be able to configure the audio hardware's sample rate, rendering block size, and numeric format.

- **[dev.music.direct_rendering]** As a developer of a professional music production application, to fully control the sound quality and reduce latency, I want to render audio samples directly to audio hardware output channels, and receive audio samples directly from audio hardware input channels, without any converting mediation from the operating system.

- **[dev.music.exclusive_mode]** As a developer of a professional music production application, for better stability and performance, I want the application to take exclusive control of a connected audio I/O hardware device, and to block OS system sounds and sound from other programs to get mixed in, where the OS provides this capability (for example, Exclusive Mode on Windows).

- **[dev.music.backend_selection]** As a developer of a professional music production application, 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.

- **[dev.music.channel_mapping]** As a developer of a professional music production application, when working with multi-channel audio I/O hardware, I want to query its channel and format capabilities. I want to choose which of the available channels should be used for my app's audio input and output. I want a mechanism 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).

- **[dev.music.hardware_changes]** As a developer of a professional music production application, to make it easier for my users to use different audio hardware devices and to plug and unplug them while they are working, I want to receive notifications when any audio hardware device is connected or disconnected while my program is running, and to react to them in a way that makes sense for my program.

- **[dev.music.persistent_device_ids]** As a developer of a professional music production application, to make it easier for my users to use different audio hardware devices simultaneously, I want the audio devices to be programmatically identifiable with a stable

ID such that the choice of a particular device can be persisted in a file on disk and programmatically selected again on the next run of my application and after a system reboot.

- **[dev.music.midi_general]** As a developer of a professional music production application, to make it possible to interact with common musical controllers such as a MIDI keyboard, I want to send and receive MIDI data to and from connected MIDI devices.

- **[dev.music.midi_library]** As a developer of a professional music production application, to handle musical information inside my app, I want convenient library facilities for dealing with MIDI events (note-on, note-off, pitch bend, etc).

- **[dev.music.midi_integration]** As a developer of a professional music production application, to minimise latency and avoid having to manually synchronise audio and MIDI threads, I want the audio I/O and MIDI I/O to be integrated with each other in the audio library (for example, by providing the MIDI input buffer together with the audio I/O buffer in the audio processing callback).

Audio algorithm engineers

- **[dev.algo.generalized_format]** 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.

- **[dev.algo.hardware_format]** As an audio algorithm developer, in order 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. This includes types seen in hardware and audio files, but not currently captured by the C++ type system such as non-native-endian 16-bit, 24-bit, 24-bit encoded in 32-bit, etc.

- **[dev.algo.deinterleaved]** As an audio algorithm developer, I want to easily implement algorithms that process one channel at a time (such as an FFT). For this, I want to be able to work with deinterleaved audio buffers, which give me access to every channel as a contiguous block of samples, even if this is not the native audio buffer format provided by the audio driver. I expect this buffer format to be provided by the audio library I am using, rather than having to deal with the buffer format myself.

- **[dev.algo.flexible_buffer]** As an audio algorithm developer, I want to have a flexible and efficient audio buffer type that can represent different data layouts (interleaved or deinterleaved, contiguously allocated or using pointer-to-pointer, etc.). I require this flexibility to ensure I can store and pass through functions audio in the most efficient representation for a given context, without necessitating extra conversions.

- **[dev.algo.realtime_safe]** As an audio algorithm developer using an audio I/O library, I want to work with vocabulary types and functions that are safe to use in the context of real-time audio processing (i.e. non-blocking, non-allocating, etc).

- **[dev.algo.complete_library]** As an audio algorithm developer, 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.

- **[dev.algo.common_algorithms]** As an audio algorithm developer, to make implementing audio algorithms easier, I want to have a generic, efficient implementation of common audio algorithms such as interleave/deinterleave, channel extraction, basic filters (FIR, biquad), gain, and resampling.

Embedded platforms

- **[dev.embed.no_floating_point]** 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.

- **[dev.embed.fixed_format]** As an audio developer for embedded platforms, I know the rendering block size, sample rate, and numeric format at compile time, because they are fixed on my platform. I do not want to pay any runtime overhead for those values.

- **[dev.embed.bare_metal]** As an audio developer for embedded platforms, I am working on a bare metal platform with no OS, no threads, and no clocks. I am reading and writing audio data by implementing my own double buffering and using direct memory access. I would like to use an audio library that would be available on such platforms, so that I do not have to re-implement everything from scratch for every new platform and product.

- **[dev.embed.deinterleaved]** As an audio developer for embedded platforms, when developing for a platform with very constrained resources and computing speed, I want to process audio channel data in individual contiguous arrays, without having to deinterleave the audio signal, because this approach is much faster on my platform.

- **[dev.embed.interleaved]** As an audio developer for embedded platforms, to be compatible with typical digital audio transport formats such as I^2S or AES3, I want to interleave my audio data for purposes of transmission and storage.

Standard library implementers and vendors

- [**vendor.std.hardware_format**] As a vendor of the standard library on a real-time system, I need to play application-provided audio with maximal efficiency and using the hardware-native audio representation. I cannot afford to lose performance from translation to any intermediate audio representation. If new audio hardware is introduced using sample types and/or buffer layouts previously unknown, I need to require recompilation of the client application code.

- **[vendor.std.virtualized_device]** As a vendor of the standard library, I need to provide an implementation on a system that does not expose physical audio I/O devices to me. I need the audio API to support the concept of always-available input source and output destination that may vary in its underlying physical hardware mapping.

- **[vendor.std.interruption_policy]** As a vendor of the standard library, 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, or interrupt all audio with higher priority audio sources (for example, telecommunications or audio notifications).

- **[vendor.std.input_access]** As a vendor of the standard library, if the OS imposes restrictions on which application is allowed to access the microphone, I do not want an audio API to be able to circumvent them.

- **[vendor.std.backends]** As an implementer of a standard library implementation that targets an operating system with a variety of different native audio backend APIs, and different such APIs being present on different distributions of the OS (for example, Linux), I need to be able to provide a working implementation of my library to all my users.

Third-party libraries

- **[vendor.3rdparty.data_types]** As a developer of a third-party library, I want to provide audio data processing or handling APIs with data types that are consistent between other C++ audio libraries.

- **[vendor.3rdparty.file_playback]** As a developer of a third-party library, 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.

Education

- **[edu.general]** 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.

- **[edu.crossplatform]** As a teacher, in order to give my students working code as part of the class material, I want to be able to write a C++ program with audio capabilities that will compile and run on any desktop platform.

Acknowledgements

Thanks to Stefan Stenzel and Tom Waldron for providing additional information about embedded use cases.

Document history

- R0, 2020-01-13: initial version
- R1, 2020-03-02: added more use cases, arranged use cases into sections, added stable refs, cited examples

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