Article

Porting your audio code to Apple silicon

Eliminate issues in your audio-specific code when running on Apple silicon Mac computers.

Overview

Include time in your porting plans to migrate code that uses the Core Audio family of frameworks. In particular, update your Audio Units to support Apple silicon, and optimize any real-time code to run efficiently on all Mac computers.

Create Universal Versions of Your Host App and Audio Units

Always create universal versions of your audio host app and Audio Unit plug-ins. Universal binaries ensure that your code runs natively on all platforms, which gives you the opportunity to optimize your code for each platform.

Because the host app controls the execution environment, providing universal Audio Units is particularly important. If your Audio Unit contains code for both `arm64` and `x86_64` architectures, a host app can load your Audio Unit either in-process or out-of-process. The ability to load Audio Units in-process is still important for some apps, such as those that require minimal latency.

To learn how to create a universal binary, see Building a universal macOS binary.

Support the Modern Audio Component Architecture

For the `arm64` architecture, always use the Audio Component API to load Audio Units, codecs, and other code modules into your app. The Audio Component API is the modern way to search for loadable code modules, and it's available in macOS 10.6 and later. Apps that target the `arm64` architecture or link against the macOS 11 SDK cannot use the legacy Carbon Component Manager

API to open Audio Units. If your app uses the Carbon Component Manager, plan to migrate off of it when porting your app.

If you develop Audio Units or codecs, update your code to support the Audio Component or Audio Unit Extension APIs if you haven't already done so. When you link an Audio Unit or codec against the macOS 11 (or later) SDK, use one of these modern APIs instead of the Carbon Component API.

> **Note**
>
> To preserve compatibility, apps that link against the macOS 10.15 (or earlier) SDK may continue to use the Carbon Component Manager to load Audio Units and codecs. Similarly, Audio Units and codecs built using the macOS 10.15 (or earlier) SDK may continue to include separate resources and entry points to support older APIs.

For information about the Audio Component API, see the Audio Toolbox framework.

Prioritize Realtime Threads Using Workgroups

Apps that perform realtime audio processing need to ensure that their threads run at regular intervals. In some cases, your app or Audio Unit may also need to coordinate with threads from the audio server or a host app to ensure timely processing of audio. For both of these situations, use workgroups to communicate the scheduling needs of your realtime threads to the system.

Each Core Audio device provides a workgroup that other realtime threads can join using the `os_workgroup_join_self` function. Joining the audio device workgroup tells the system that your app's realtime threads are working toward the same deadline as the device's thread. You access the workgroup associated with a device in one of several ways:

- Fetch the `kAudioDevicePropertyIOThreadOSWorkgroup` property of the device.
- Fetch the `kAudioOutputUnitProperty_OSWorkgroup` property of an audio I/O unit (AUHAL or AURemoteIO).
- Get it from the `osWorkgroup` property of an `AUAudioUnit` object that acts as an input/output unit.

If your Audio Unit creates its own realtime audio processing threads for rendering, coordinate the activity of those threads with the host app by joining the threads to the host app's workgroup. To obtain the host app's workgroup, return an `AURenderContextObserver` block from the `renderContextObserver` property of your `AUAudioUnit` object. (For v2 Audio Units, return the `AURenderContextObserver` block as the data for your Audio Unit's `kAudioUnitProperty_RenderContextObserver` property.) When the system executes your block, retrieve the host app's workgroup from the provided `AudioUnitRenderContext` object. The host's workgroup may change between rendering calls. If it does, update your realtime audio threads to join the workgroup that the system passes to your block, and to leave their previous workgroup.

If your realtime audio threads operate on different deadlines than Core Audio threads, create your own interval workgroup using the `AudioWorkIntervalCreate` function. For workgroups you create, join your threads to the workgroup and call the `os_workgroup_interval_start` function from one thread. When you call that function, specify the time at which you expected your thread to wake up and process the audio. Upon completion of the work, call the `os_workgroup_interval_finish` function to tell the system that you finished the work associated with the current deadline.

For information about the Core Audio workgroups, see the reference for Audio Toolbox.

See Also

General porting tips

Addressing architectural differences in your macOS code

Fix problems that stem from architectural differences between Apple silicon and Intel-based Mac computers.

Porting just-in-time compilers to Apple silicon

Update your just-in-time (JIT) compiler to work with the Hardened Runtime capability, and with Apple silicon.