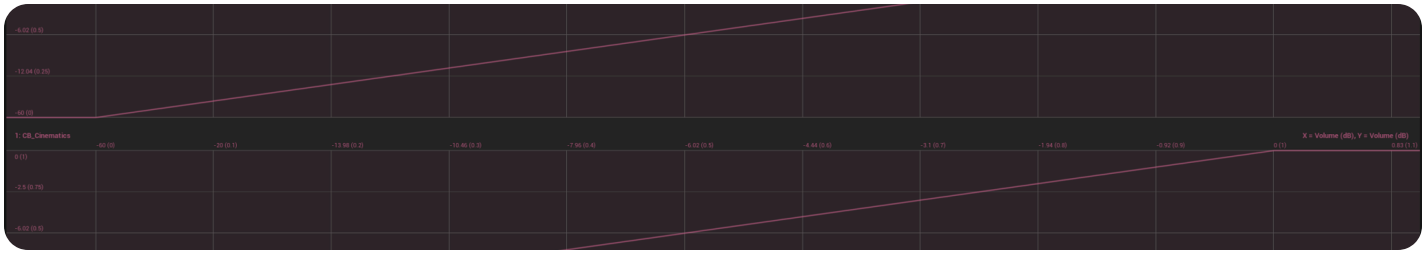
Audio Engine Overview

An overview of features in the Unreal Audio Engine.



The **Unreal Audio Engine** is a robust audio engine that supports a wide variety of features across all platforms supported by **Unreal Engine**.

Combined with the power of Blueprints and a new multi-platform audio mixer that enables audio digital signal processing (DSP), procedural synthesis, a customizable submix graph, and a flexible C++ API, the audio engine is fully capable of shipping high-quality audio on any project, and at any scale.

This overview presents a snapshot of various subsystems and features that are utilized to solve high-level audio issues.

Source Asset Management

Overall asset management is straightforward. The guiding principle in asset management is to do things as directly as possible, and handle things automatically whenever possible.

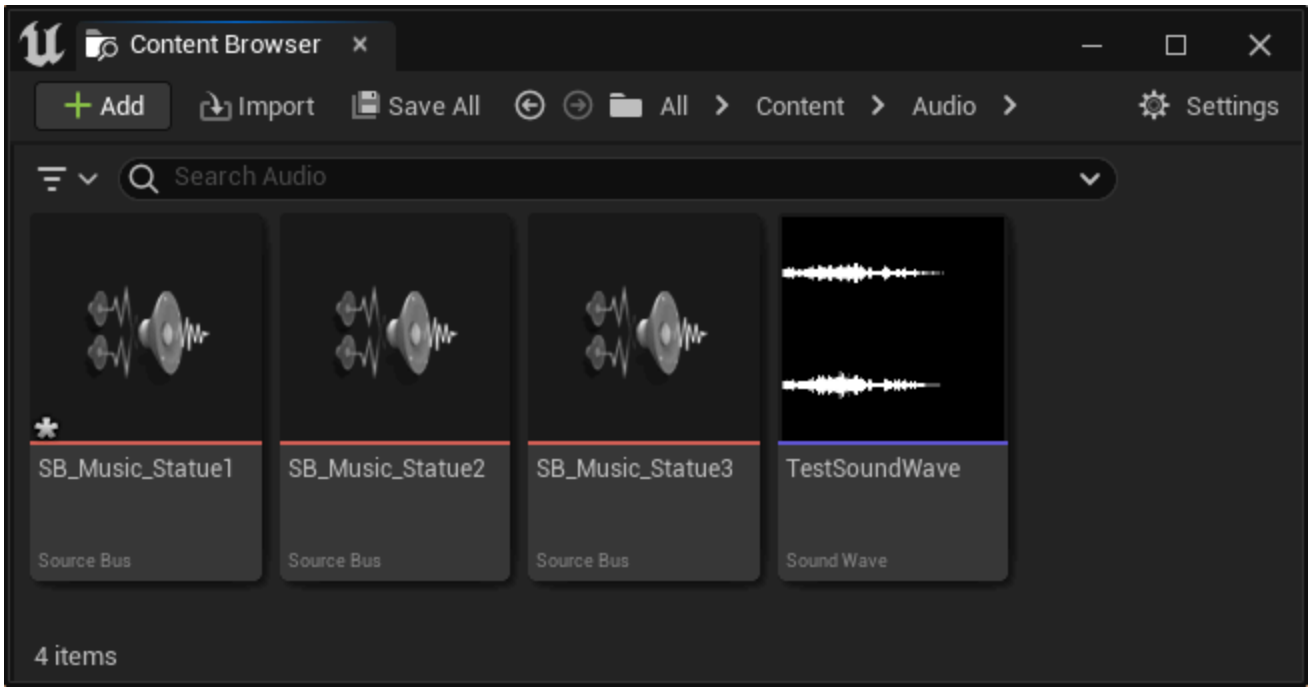
Importing Assets

Unreal Engine supports direct import of audio assets into the Content Browser from the desktop by use of the file import dialog, or by dragging audio assets directly into the browser.

You can import a wide variety of asset types and channel formats (see Importing Audio Files).

Internally, Unreal Engine stores the imported audio file in a 16-bit, uncompressed `.wav` format.

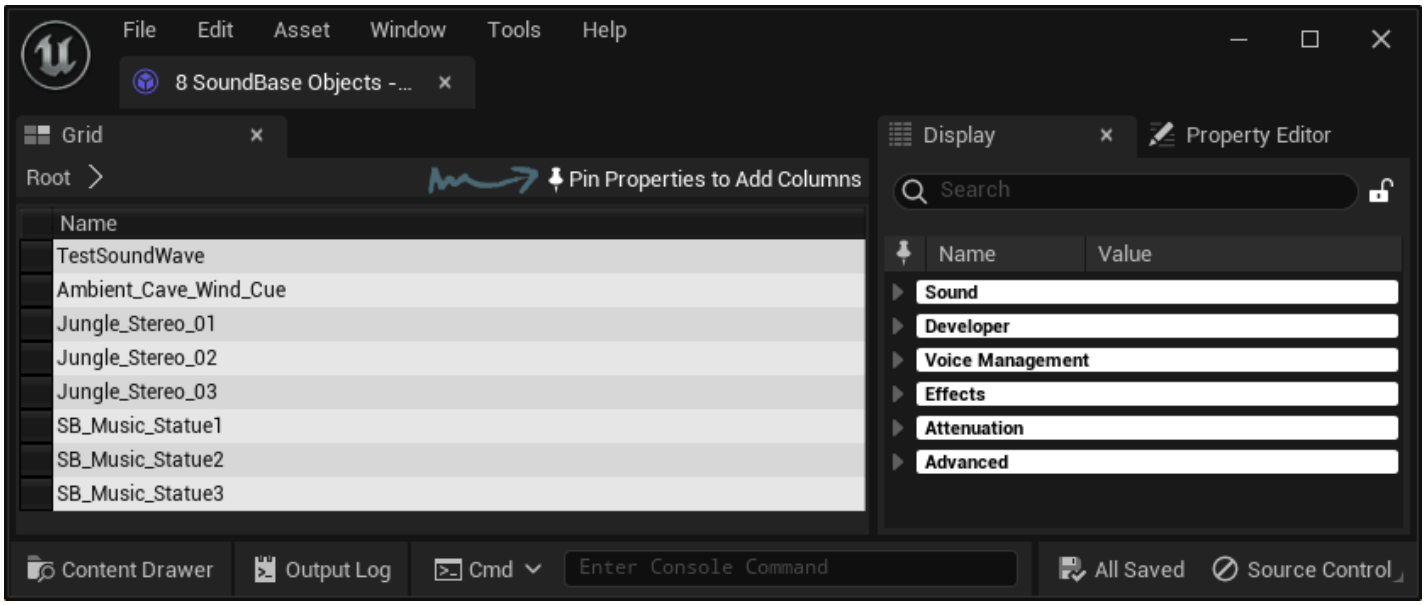
The `UAsset` that contains the imported data is called a `USoundWave`, and it can be played directly from all Unreal Engine audio gameplay APIs. The imported asset can also be previewed in the Content Browser by clicking the Play button on the asset or pressing the spacebar while the asset is selected.



When an audio asset is imported into the Content Browser, the engine stores it in a 16-bit, uncompressed .wav format.

Asset Management Tools

Apart from a well-organized Content Browser folder structure, well-designed bookmarks, and useful general Content Browser tools, audio assets in particular benefit greatly from the **Property Matrix** tool. To make mass edits to audio assets in Unreal Engine, select the assets you want to edit, then open the Property Matrix tool, where you can make sweeping adjustments on up to hundreds of assets simultaneously.



The Property Matrix tool can be used to make broad changes to assets at the same time.

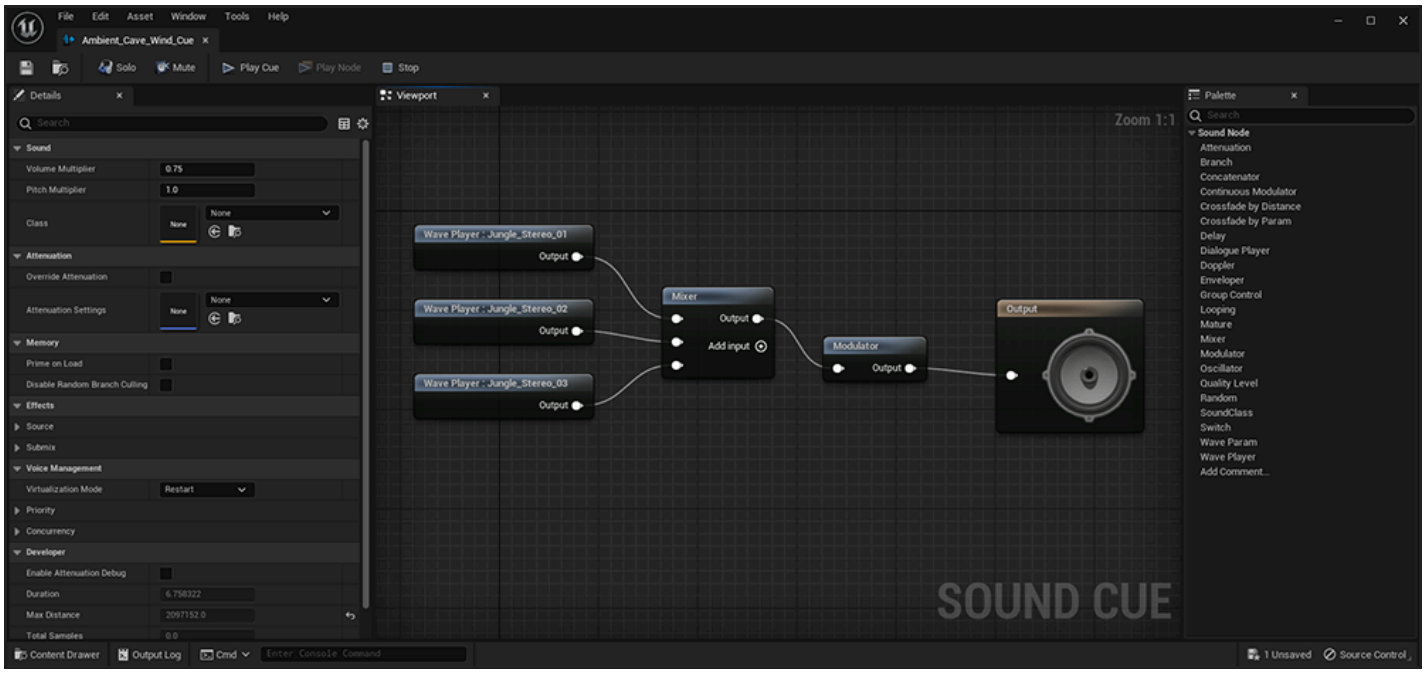
Sound Cues

In the Unreal Audio Engine, the asset that is the conceptual aggregation of multiple sound assets into an abstract Sound Object is the **Sound Cue**. It is not literally the sound itself, but *the abstract concept of that sound*.

A Sound Cue is runtime-evaluated, that is, a bundle of potential—not actual—sound. Each time a Sound Cue plays, it may sound different, depending on the context in which it is used.

Sound Cues are one of the oldest tools in Unreal Engine. The Sound Cue Editor gives sound designers a way to create their own sound-design topologies that translate a simple play-sound event into an arbitrarily complex, nuanced, sound-designed event.

At a high level, Sound Cues support asset randomization, gameplay-driven parameter mapping, custom logic and branching for distance attenuation, volume attenuation, pitch shifting, and a large number of other features.



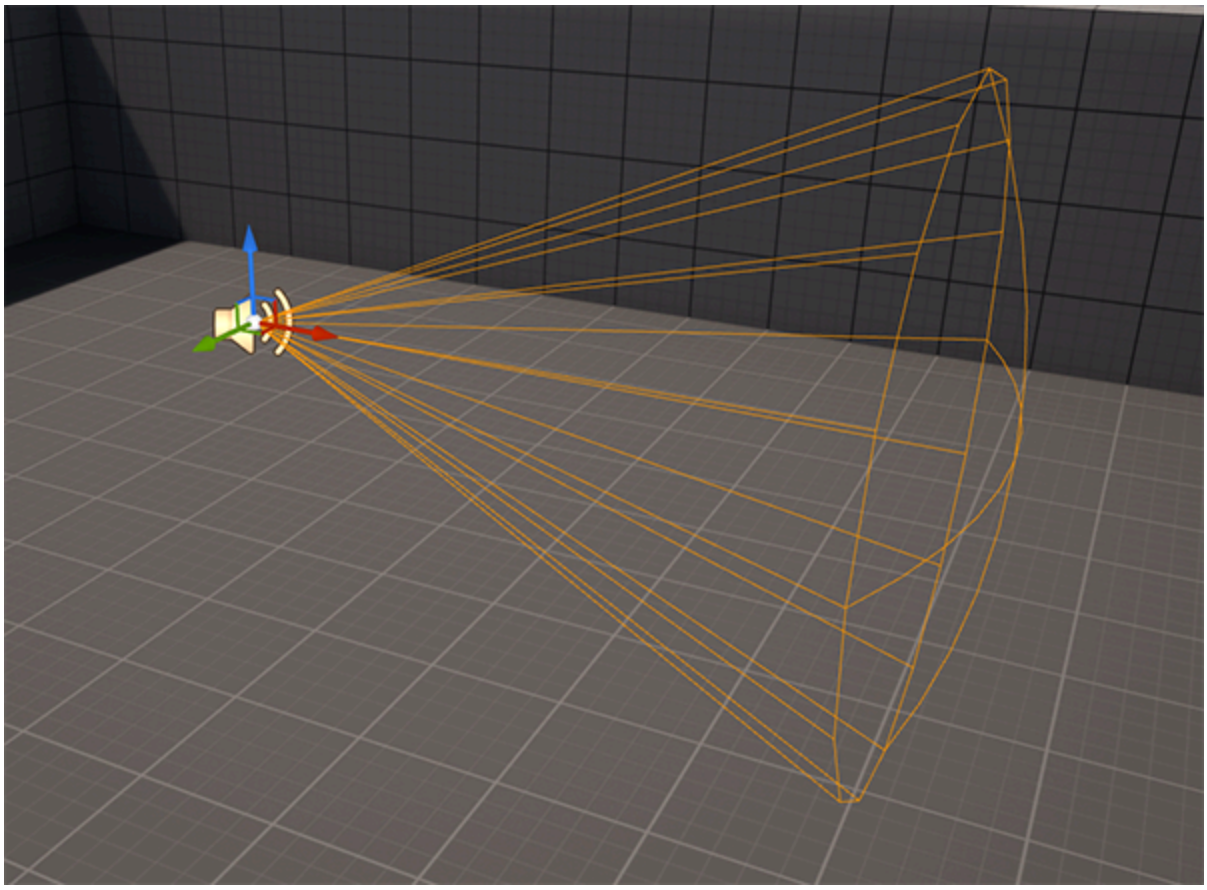
The Sound Cue Editor provides a way of translating a simple play-sound event into a complex, nuanced, sound-designed event.

Sound Cues can also be built using the **Sound Cue Templates** plugin, which is a simple C++ interface that sound designers can use to name specific sound cues, graph topologies, and logic to be automatically constructed and used from the Matrix Property Editor and the Content Browser. In this way, a common Sound Cue graph can be defined, and the sound designer can use this to optimize the workflow.

See the Sound Cue Editor for more information.

Spatialization and Attenuation

Spatialization and attenuation in games is important. In the Unreal Audio Engine, you can define custom spatialization and attenuation behavior.



Custom spatialization and attenuation can be done in various ways.

In general, sound and attenuation settings are defined using a **Sound Attenuation Settings** asset, and *hooked up* to sounds by sound designers. There are a number of places where sound attenuation can be hooked up:

- Directly on sound assets
- Within Sound Cues
- Overridden in Blueprint

For more information, see Sound Attenuation.

Non-Spatialized Sounds

Non-spatialized sounds are usually referred to as **2D sounds** in the context of audio engines. Unreal Audio supports playback of 2D assets by disabling spatialization in the sound's attenuation setting asset, or by not providing a sound attenuation asset. Such 2D sounds are useful for music, UI feedback sounds, and any other number of cases where the position of the listener is static.

Even though a sound asset might be played without a special spatialization method, the asset can contain spatial information that is baked into it.

- **Multichannel sound sources** (such as quad, 5.1, and 7.1 sound sources) are only supported for 2D playback. These are typically used for musical or ambient beds.
- **Stereo sound sources** can be played back in 2D but can also be spatialized.

Spatialized Sounds

By default, when spatialization is enabled by attenuation settings, sounds are spatialized using panning.

Panning is the process of changing the amplitude of a sound in physical output speaker channels based on the angle of that sound relative to the orientation of the listener.

Unreal Audio supports panning for multiple output configurations: stereo, 5.1, and 7.1, and for both mono and stereo source assets.

For stereo assets, there is a stereo-spread parameter that defines the virtual distance between left and right input channels for the source. In this way, the stereo source left and right channels are panned analogously to a mono source, and relative to the listener.

There are two **global panning modes** that sound designers can configure to use in projects: linear panning and equal-power panning.

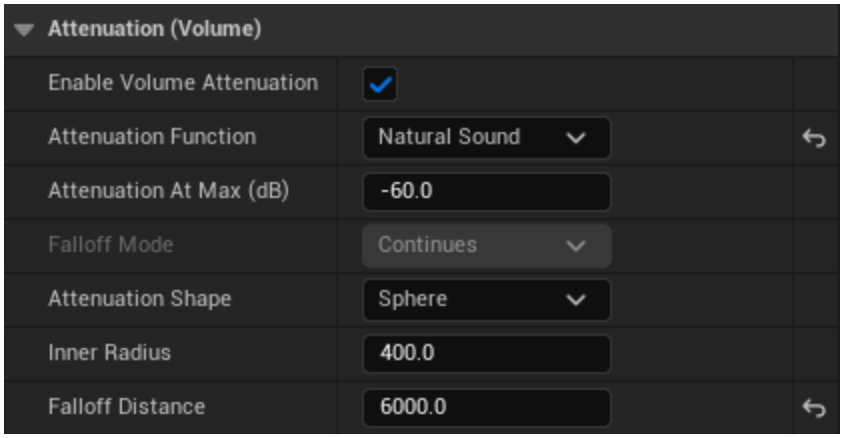
- **Linear panning** performs a simple linear crossfade between output channels.
- **Equal-power panning** maintains the power (amplitude squared) between output channels.

Since perception of volume is based on the power of the sound, maintaining equal power for panning results in a more perceptually constant volume pan. Linear panning results in sound appearing louder as it gets closer to the physical speaker location, and quieter when it is between speaker locations.

Unreal Audio also supports spatialization using third-party plugins with a robust and flexible C++ API. Third-party plugins can spatialize sounds however they want, including binaural/HRTF spatialization or ambisonics (soundfield) encoding and decoding. Plugins can provide their own asset settings that can be hooked up to the attenuation setting asset.

Distance Attenuation

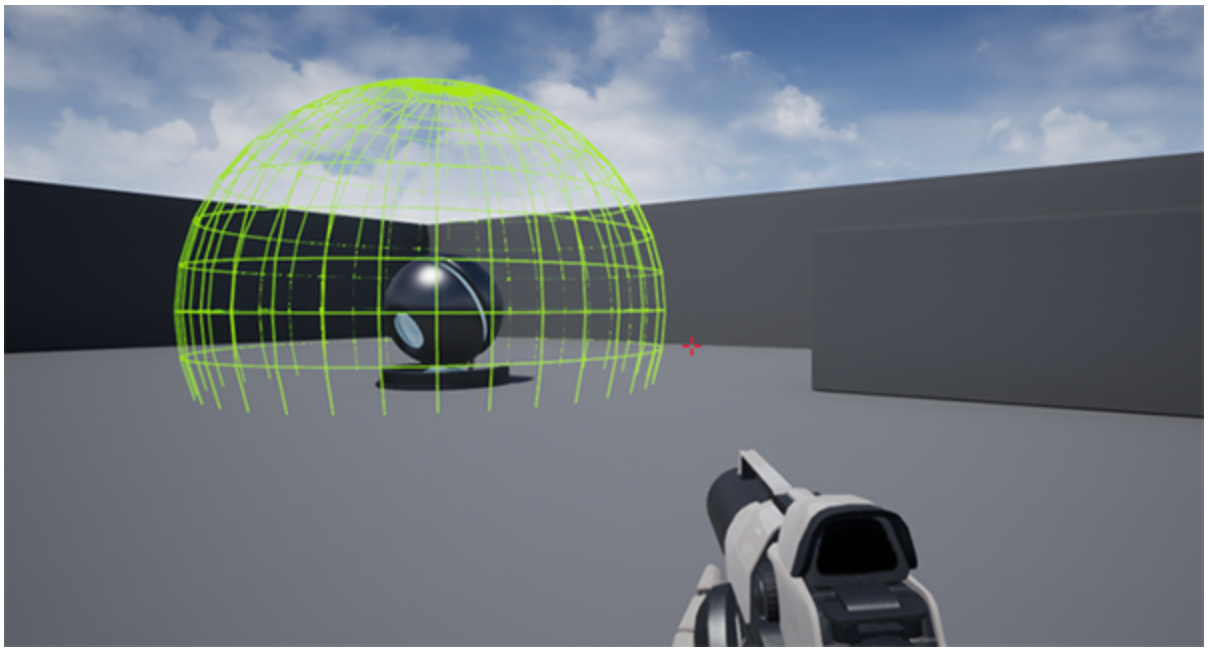
Distance-based sound attenuation is also defined in the Sound Attenuation Settings asset. In addition to providing a number of pre-designed functions and shapes for distance-based sound attenuation, sound designers can design their own custom distance-attenuation curves.



Spatialization/Attenuation Orthogonality

One subtle aspect of the Unreal Audio spatialization and attenuation features is that they are treated as orthogonal properties. This means sound designers can choose to attenuate a sound by distance (or source-listener orientation) independent of spatialization, and vice versa.

Spatialization and attenuation settings are defined on the Sound Attenuation Settings asset.



In this image, when the listener is within the non-spatialized radius (represented by the green sphere), the sound bleeds to all channels of the speaker configuration. Outside of the radius, the sound is spatialized as normal.

Distance Filtering

Sound Attenuation settings also have an option for filtering sounds as a function of distance. Separate curves for low-pass and high-pass filters allow sound designers to emulate the effect of air absorption (low-pass filter), or to model frequency-dependent distance attenuation (high-pass filter).

Occlusion

Unreal Audio has an extremely inexpensive default async trace that performs occlusion checks for sounds. The sound attenuation setting has options that allow sound designers to enable occlusion, and to set a variety of parameters for filtering sounds based on whether they are occluded. More advanced occlusion solutions are implemented in third-party plugins utilizing Unreal Audio's Occlusion C++ API.

Listener-Based Attenuation

Sound Attenuation settings also have an option that allows sound designers to write volume attenuation, prioritization scaling, and other effects based on the orientation of the sound relative to the listener. In this way, sounds can become more "in-focus" or "out-of-focus" when they come within view of the listener.

For more information, see Attenuation and Listener Focus.

Distance-Based Reverb Sends

Unreal Audio also supports changing how much audio is sent to the master reverb submix as a function of distance. This mapping is defined similarly to the other distance-based parameter curves.

For more information, see Attenuation Reverb Send.

Gameplay Audio API

Unreal Audio has a simple and flexible gameplay API that allows Blueprint and C++ code to customize the behavior of the audio engine.

PlaySound and SpawnSound Blueprint APIs

There are a number of Blueprint functions that you can use to play sounds from Blueprints and gameplay C++ code, in two general categories:

- **PlaySound functions:** This includes PlaySoundAtLocation, PlaySound2D, PlayDialogue2D, and so on.
- **SpawnSound functions:** This includes SpawnSoundAtLocation, SpawnSound2D, and SpawnSoundAttached.
- **PlaySound APIs:** These play the indicated sound in a fire-and-forget mode. Once the sound is playing, you can't modify its playback from Blueprint, or attach it to objects. This type of playback is useful for simple one-shot types of sounds where you don't need dynamic control.
- **SpawnSound functions:** You can create an audio component to dynamically control parameters on a sound, attach the sound to other actors, and control looping sounds.

Audio components are a useful object in Unreal Engine for sounds, and can be used for complex interactive Blueprint systems for audio.

Most Unreal Engine audio systems have associated Blueprint APIs that allow for customization and control from Blueprint. For example, every source and submix DSP effect can be modified and controlled from Blueprint, as can sound mixes and sound classes.

Game Volume Mixing

Game mixing is one of the more challenging aspects of game audio. Unreal Audio provides a number of features for sound designers to use to define and control game mix. There is a wide variety of factors that contribute to the overall volume mix.

Direct Volume Adjustments

Individual assets, such as Sound Cues and Sound Waves, have parameters that provide volume control. Audio components also allow sound volume to be modified from Blueprint. The PlaySound and SpawnSound audio gameplay API makes it possible to select volumes at playback. Sound Cues can modify sound volumes according to dynamic gameplay parameters, or with other Sound Cues graph logic. Asset volume can also be set via a variety of distance attenuation options, including attenuation via listener orientation.

Sound Classes

A Sound Class is an asset that provides a bucket of common settings for assets that share some semantic meaning. One primary motivation for grouping sounds together into a common classification is to control volumes of sounds as a group. Sound Classes are created and modified in a separate Sound Class graph editor. In addition to volume, Sound Classes can be used to control other parameters on sounds in a class group.

Sound Mixes

A Sound Mix is an asset type that applies dynamic Sound Class volume and pitch adjusters to Sound Classes. Sound Mixes are the traditional method with games to perform Class-based

volume control, including ducking.

Volume ducking is achieved using a Passive Mix Modifier—a mechanism that indirectly applies a mix when a sound plays on a given Sound Class. In this way, a game can, for example, cause ducking on background ambience when a player talks or when a gun shoots.

Sound Parameter Modulation

A newer mixing method to control a game mix is with the audio **Parameter Modulation** plugin. This plugin generalizes the concept of changing audio parameters through a mix. Sounds can now associate their parameters as **modulation destinations** of a parameter bus. A **parameter bus** is an object that allows arbitrary parameter modulation sources to write into the bus. For example, a modulation source could be from a Blueprint, derived from a parameter mix, from some interactive system, or from a parameter modulation oscillator such as an LFO.

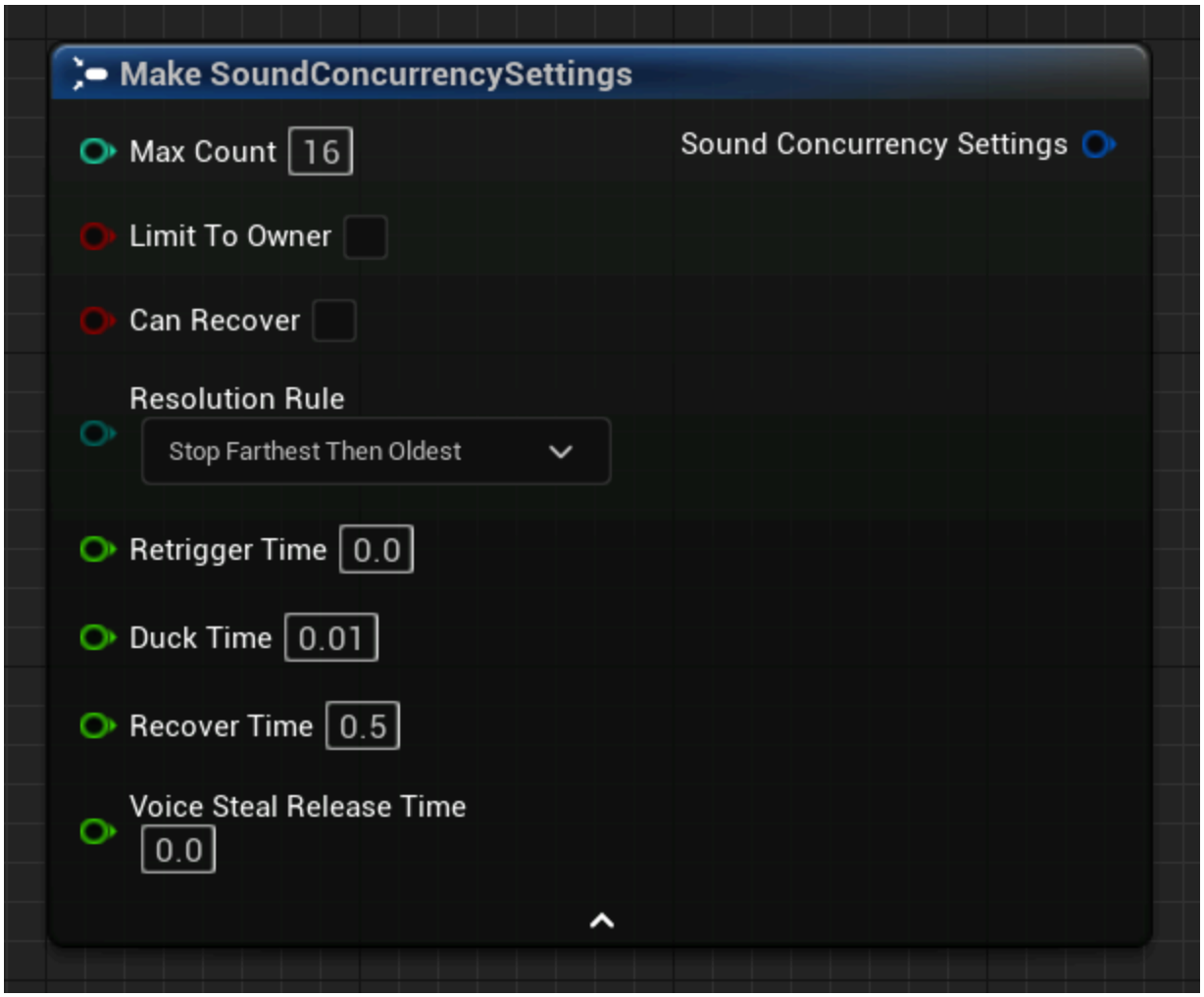
Instead of controlling only the volume parameter of a sound, the Parameter Modulation plugin also generalizes the concept of parameter modulation, and allows the same paradigm to control any number of output parameters (such as filter frequency cutoff and pitch modulation).

For more information, see Audio Modulation Overview.

Concurrency Management

An often overlooked aspect of game mixing is the issue of managing **sound concurrency**—essentially, how many sounds of a given type can play at the same time. Without careful management, it is easy for games to generate a flood of sounds of one type, such as guns and other enemy weapons.

The Unreal Audio Engine provides tools for sound designers to control their concurrency groups via the Sound Concurrency Asset. This asset defines the limit of how many sounds to allow in the group, and what to do once that limit is reached.



Sound concurrency is a way to manage how many sounds of a specific type can play simultaneously.

For example, a sound designer could choose to stop the oldest sound when a new sound comes in, or alternatively, to reject new sounds and continue the ones that are already playing.

Sound Concurrency also allows groups to happen in a chain, where a hierarchy of **Concurrency Resolution** needs to pass for a sound to play.

For example, footsteps might exist in a concurrency, limiting the total number of footsteps. But that, in turn, could exist in a concurrency group of Foley sounds, and again in turn, might be in a concurrency group of SFX sounds, and so on.

For more information, see Make SoundConcurrencySetting.

Global Polyphony Management and Prioritization

Managing the number of sounds that can play at once in a game is called **polyphony management** or **voice management**. The primary cost for an audio engine is decoding and rendering sound sources, so one of the primary tools for reducing CPU cost is limiting the number of sounds that can play at the same time.

In Unreal Engine, this number is set in a project setting that you can change based on the target platform. The number can also be changed dynamically while the game is running, so in situations where performance is bottlenecked or tight, the audio engine can dynamically reduce the number of sound sources to be rendered.

Once the audio engine reaches the number of sounds it can render, it needs to decide which ones to play and which ones to reject or stop. This is done through a combination of the sound's volume total (the final volume of the sound after all stages are accounted for) and the sound's priority. For more information, see Priority.

DSP Effects and Synthesis

The Unreal Audio Engine also features a robust **digital signal processing (DSP)** effects processing pipeline.

Individual sound sources can specify a DSP effect chain to be applied to individual source instances. Settings for those effects can be changed at runtime in Blueprint, and will support live previewing changes while the Play in Editor (PIE) session runs.

Unreal Audio also has a Submix Graph Editor that allows sound designers to apply DSP effects and perform analysis on mixes of sound sources. You can construct submix effects in submix effect chains. Sound sources can be specified to play on any given submix and to *send* their audio as **send effects** to any other submix graph. Any sound source can dynamically route their audio to any submix from a Blueprint. Further, Sound Attenuation Settings support sending audio to submix effects as a function of distance. There is a huge variety of possibilities.

The Unreal Audio Engine is flexible enough to allow canonical master-effect setups like Master EQ, Master Compression, and Master Reverb, while also supporting experimentation and exploration of more custom DSP graphs and routing schemes.

A variety of source and submix effects have been implemented in the Synthesis and DSP Effects plugin, and new effects are always being added. With our growing DSP C++ library in

our Signal Processing Module, it is easy to add new effects via the Unreal Audio Engine's DSP and Synthesis C++ API.

With other robust tools and technology in Unreal Engine via automatic property reflection and hot-loading DLLs, Unreal Audio Engine is an ideal place to learn about writing DSP effects, or to serve as a platform for cutting-edge experimentation and research in game audio technology.

Asset Cooking

When cooking for a target platform, assets are compressed using a codec specific to both the platform being targeted and the feature the sound is using. For example, Unreal Engine will automatically encode to hardware-accelerated codecs on platforms where that is supported.

The `USoundWave` that wraps the imported asset has a quality slider to help cook-time compression schemes in relation to the quality the sound designer is targeting for the given asset. This quality value is interpreted differently, based on the specific codec used for a target platform.

Asset Compression Overrides: Automatic Quality Reduction

Unreal Engine has a wide variety of tools that sound designers and engineers can use to control automatic cook-time down-sampling and quality scale reductions for any given target platform. This allows a project to ship on as wide a variety of platforms as possible, and with as little platform-specific content redesign as possible.

For example, a project that ships on PC could automatically have the asset memory footprint reduced when targeting a mobile platform.

Debugging and Profiling

The Unreal Audio Engine is fully integrated with the rest of Unreal Engine, and so benefits from all other existing profiling tools used to analyze CPU usage and memory. Hooks into **LLM**

(low level memory) tracking are added throughout the Unreal Audio Engine, as well as **CSV CPU profiling**. Furthermore, the Unreal Insights tool has access to all audio CPU usage.

In addition to comprehensive CPU and memory profiling available to all Unreal Engine developers, there are a variety of debugging tools that can be used to view and analyze audio events as they happen in-game. For example:

- To view all playing Sound Wave instances (the objects that are actively generating sound), type **stat soundwave** in the dev console window.
- To view playing Sound Cues, type **stat soundcues** in the console window.
- To visualize playing 3D sounds in your game, type **Audio3DVisualize** in the console window.

There are also a variety of console variables that can be toggled to enable and disable various components of the audio engine, enable muting and soloing of sounds, and so on. See FAudioDevice for more information on debugging.