Audio Modulation Reference Guide

A reference guide for Audio Modulation.

The **Audio Modulation** plugin provides dynamic control over floating-point audio parameters, such as volume and pitch, from the **Component** and **Blueprint** systems.

The Audio Modulation plugin is disabled by default. To enable it, follow these steps:

1. Select **Edit > Plugins** to open the **Plugin** panel.
2. Use the search bar to find the plugin.
3. Enable the corresponding checkbox.
4. Restart the Unreal Editor.

Asset Types

There are five different asset types that provide Audio Modulation's functionality.

Asset Type	Description
Modulation Parameters	Parameters that provide context for how a modulation value normalizes, displays, and mixes with other values.
Control Buses	Connectors that reference a Modulation Parameter.
Control Bus Mixes	Modulators that can apply multiple Control Bus values at once.
Parameter Patches	Hybrid connector-modulators that remap multiple Control Bus values into a single output value.
Modulation Generators	Hybrid connector-modulators that generate values over time.

You can use these assets to create the foundation for your modulation pipeline before attaching it to an endpoint Modulation Destination.

To create an Audio Modulation asset, follow these steps:

1. In the **Content Browser**, click the **Add** button.
2. Under **Audio > Modulation**, select the desired asset type.

Modulation Parameters

Modulation Parameters provide context for how a modulation value normalizes, displays, and mixes with other values.

Each Modulation Parameter has the following base properties (in addition to others depending on class):

- **Default Parameter Value**: The default modulation value. This functions as a bypass value that should be set so it results in no change to an input when a modulator referencing this parameter is mixed with another. Mathematically, the value should be selected so the mix function is effectively reduced to an identity function when set as an input.
- **Unit Display Name**: The unit to display on references in the Unreal Editor. This information is for development purposes only, as it is removed from packaged builds.

There are several built-in Modulation Parameter classes to support a variety of use cases, including volume mixing and frequency manipulation.

The Audio Modulation plugin includes ready-made Modulation Parameter assets for the most common parameters, such as Volume and Pitch.

To view them in the **Content Browser**, follow these steps:

1. Click the **Settings** button.
2. Enable **Show Engine Content** and **Show Plugin Content**.
3. Navigate to **All/Engine/Plugins/Audio Modulation Content**.

About Modulation Values

Modulation values are transformed between two value spaces:

- **Unit**: Value in units associated with the Modulation Parameter. For example, decibels (dB) for a Volume Modulation Parameter. Unit values are mapped to normalized values.
- **Normalized**: Value between a normalized 0.0 to 1.0 range for uniform passing and mixing.

Modulation values mix when multiple modulators link to a single Modulation Destination. The way they combine depends on the associated Modulation Parameter. Here are some examples of how some of the built-in Modulation Parameter types handle mixing:

- **Volume**: The volume reductions are additive. For example, if two Control Buses have -6 dB and -12 dB values, respectively, the Modulation Destination's volume will be set to -18 dB.
- **High-Pass Filter Frequency**: The maximum value is selected. For example, if two Control Buses have 1000 Hz and 500 Hz values, respectively, the Modulation Destination's high-pass filter frequency is set to 1000 Hz.
- **Low-Pass Filter Frequency**: The minimum value is selected. For example, if two Control Buses have the 1000 Hz and 500 Hz values, respectively, the Modulation Destination's low-pass filter frequency is set to 500 Hz.

Create Custom Modulation Parameter Classes

You can create your own Modulation Parameter class by inheriting from the `USoundModulationParameter` base class and implementing the following functions:

- `GetMixFunction`: Expected to return a function that mixes an input and output value.
- `GetUnitConversionFunction`: Expected to return a function that transforms a value to unit space.
- `GetNormalizedConversionFunction`: Expected to return a function that transforms a value to normalized space.

For more detailed information, see the Audio Modulation C++ API Reference.

Control Buses

Control Buses are connectors that reference a Modulation Parameter. They can be:

- Used by Control Bus Mixes and Modulation Generators to modulate.
- Referenced and remapped by Parameter Patches.

It's generally a good idea to create a Control Bus for each category of sounds that you want to modulate. The category can be general, like sound effects (`CB_SFX`), or specific, like attack sounds (`CB_Attacks`). In this example, a sword swing sound could have both Control Buses in its modulation pipeline.

Control Bus Properties

Property	Description
Bypass	If enabled, the Control Bus sends the parameter's default value to any attached Modulation Destinations. The Control Bus remains active in either case.
Parameter	The Modulation Parameter to use. The default value of the given Modulation Parameter is used as the default value for this Control Bus.
Override Address	If enabled, you can change the **Address** property.

Property	Description
Address	The internal reference name (used by some Blueprint functions). This defaults to the asset's name but may be changed by enabling **Override Address**.
Generators	An array of Modulation Generators that drive the bus algorithmically.

Control Bus Mixes

Control Bus Mixes are modulators that help you to:

- Influence multiple Control Bus values at once.
- Modulate with Control Buses shared with other Control Bus Mixes.
- Use mix profiles to test and tweak audio in real time.

Control Bus Mix Properties

Property	Description
Activate Mix	[Button] Activates the mix at the given **Profile Index** in all active worlds.
Deactivate All Mixes	[Button] Deactivates all mixes in all active worlds.
Deactivate Mix	[Button] Deactivates the mix at the given **Profile Index** in all active worlds.
Load Mix from Profile	[Button] Loads the mix at the given **Profile Index**. This overwrites any settings that you currently have in the mix. Mix profiles are loaded from `.ini` files at `<ProjectDirectory>/Saved/Config/AudioModulation`.
Save Mix to Profile	[Button] Saves the mix to the given **Profile Index**. This overwrites any settings present at the index. Mix profiles are saved to

Property	Description
	`.ini` files at `<ProjectDirectory>/Saved/Config/AudioModulation`.
Solo Mix	[Button] Deactivates all other mixes and activates the mix at the given **Profile Index** in all active worlds. For in-editor testing only.
Profile Index	The index (integer) of the mix profile targeted by the mix profile buttons. Each mix profile has an array of **Mix Stages** that determine how attached Modulation Destinations are modulated.
Mix Stages	An array of Control Bus references and associated mix information.
Mix Stages > Bus	The Control Bus to send the **Value** through.
Mix Stages > Value	The value to send through the Control Bus. You can set the value in units or the normalized value directly.
Mix Stages > Attack Time	The duration (in seconds) to go from the Modulation Parameter's default value to the given **Value** when the mix is activated.
Mix Stages > Release Time	The duration (in seconds) to go from the given **Value** to the Modulation Parameter's default value when the mix is deactivated.

The mix profile buttons provided in the Control Bus Mix **Details** panel are helpful for adjusting and testing your mix in **Play In Editor (PIE)** in real time. However, your final mix control implementation should be in Blueprint.

Parameter Patches

Parameter Patches are hybrid connector-modulators that are helpful in two primary ways:

- They transform one or more Control Bus values into a single output value.
- You can use them to customize Control Bus value curves, which is helpful for remapping values and creating different fade effects.

Assigning a Parameter Patch with multiple Control Buses to a Modulation Destination is functionally similar to setting each of those Control Buses to the Modulation Destination individually.

Parameter Patch Properties

Property	Description
Bypass	If enabled, the patch sends the parameter's default value to any attached Modulation Destinations. The Parameter Patch remains active in either case.
Parameter	The Modulation Parameter that determines how the Control Buses (in **Inputs**) mix.
Inputs	An array of Control Bus references and associated transform information. The output of each of these **Inputs** is mixed using the Modulation Parameter patch mix function. If there are no Control Buses in the array, the patch sends the parameter's default value to attached Modulation Destinations.
Inputs > Sample-And-Hold	If enabled, stores the Control Bus value when the Parameter Patch initializes and holds it for the patch's lifetime.
Inputs > Curve Type	The curve to apply when remapping the Control Bus' normalized values [0.0 - 1.0]. Options include: • Presets, such as Linear (Ramp In) and Sine (360 deg). • **Shared**: Reference a shared **Curve** asset. • **Custom**: Design a custom curve. For more information about curve editing, see Curve Editor.

Property	Description
Inputs > Bus	The input Control Bus.

Modulation Generators

Modulation Generators are hybrid connector-modulators that generate values over time. You can modulate with them by setting them on Control Bus or Modulation Destination properties.

The plugin provides implementations for the following types:

- **Low-Frequency Oscillator (LFO)**: Generates values based on oscillations of a given waveform, such as sine or square.
- **Envelope Follower**: Generates values based on the amplitude of a given Audio Bus.
- **AD Envelope**: Generates values based on a given attack-decay envelope.

Create Custom Modulation Generators

You can create your own Modulation Generators with the code generation templates provided for the `USoundModulationGenerator` class, which generates the boilerplate code necessary for a generator to be created and registered.

To create a `USoundModulationGenerator` class, follow these steps:

1. Select **Tools > New C++ Class**.
2. Select **All Classes** from the **Choose Parent Class** window.
3. Select **SoundModulatorGenerator** under **SoundModulatorBase**.

You must implement the following functions:

- `GetValue`: Expected to return a cached, normalized, unitless value between 0.0 and 1.0. The value is provided to listening Modulation Destinations.
- `Update`: Expected to implement calculation of the next frame's cached value.

You can optionally implement the following functions:

- `IsBypassed`: Expected to return a boolean value based on whether the generator value should be included in Destination computations. By default, it forwards the value from the generated `USoundModulationGenerator Bypass` property to the Unreal Audio Engine.

- `GetDebugValues`: Expected to provide an array of strings per generator instance to be displayed at runtime using the `au.Debug.SoundModulators` family of debug commands in non-shipped builds.
- `GetDebugCategories`: Expected to supply a static array of field names corresponding to the values provided by these instance values.

For more detailed information, see the Audio Modulation C++ API Reference.

Modulation Destinations

A **Modulation Destination** is an endpoint attached to an audio object that provides a base float value (in the given units) to be modulated by Audio Modulation assets.

The following objects have Modulation Destinations:

- Sound Waves
- MetaSound Sources
- Audio Components
- Synth Components
- Sound Classes
- Submixes
- Source Effects (BitCrusher and Chorus only)

You can find the Modulation Destinations for each supported audio object in their **Details** panel under **Modulation**. Each object lists its supported Modulation Destination types that determine which property, such as volume or pitch, is modulated.

To use a Modulation Destination, you must enable the associated **Modulate** checkbox and then add the desired connector (Control Bus, Modulation Generator, or Parameter Patch) to the array.

Modulation Destination Routing Options

Some Modulation Destinations, such as those on Audio Components, provide inheritance-based routing options. A child object may inherit modulation routing from a parent object. Specifically, Audio Components can inherit from the attached Sound asset, and Sounds can inherit from the attached Sound Class.

Routing Option	Description
Disable	Disables modulation routing.
Inherit	Inherits modulation routing from the parent object. This option is the default because Sound Classes are the most common place to set up modulation.
Override	Overrides the parent object's modulation routing with the given settings.
Union	Uses both the inherited modulation routing from the parent object and the given settings. If the same modulator is in both sets, it calculates only once.

MetaSounds Integration

The Audio Modulation plugin provides the following MetaSound nodes:

- **Get Modulator Value**: Returns the given modulator value, either normalized or in unit space.
- **Mix Modulators**: Returns a mixed parameter value between two given modulators, either normalized or in unit space.

For more information on the MetaSound system, see the MetaSounds documentation.

Blueprint API Summary

With Blueprint, you can create and control your modulation pipeline dynamically and use it to drive other systems, such as gameplay or visual effects.

Most of the Blueprint functions related to Audio Modulation are under **Audio > Modulation**. See the table below for a high-level overview of the functions.

Function	Variants	Description
Activate...	**Control Bus**, **Control Bus Mix**, **Modulation Generator**	Activates the provided object.

Function	Variants	Description
Clear...	**All Global Control Bus Mix Values**, **Global Control Bus Mix Value**, **Modulator**	Resets the provided bus' parameters to their default values.
Create...	**AD Envelope Generator**, **Control Bus**, **Control Bus Mix**, **Control Bus Mix Stage**, **Envelope Follower Generator**, **LFO Generator**, **Modulator Destination**	Creates an object with the provided default values.
Deactivate...	**All Control Bus Mixes, Control Bus**, **Control Bus Mix**, **Modulation Generator**	Deactivates the provided object.
Get Modulator	N/A	Gets the modulator of the given Modulation Destination.
Get Modulator Value	N/A	Gets the normalized value of the given modulator.
Get Modulators From Destination	N/A	Gets the list of modulators currently applied to a Modulation Destination.
Get Watched Modulator Value	N/A	Gets the value of the watched modulator value of the given Modulation Destination.
Is Control Bus Mix Active	N/A	Checks if the given Control Bus Mix is active.

Function	Variants	Description
Load Control Bus Mix From Profile	N/A	Loads control bus mix from a profile into UObject mix definition, deserialized from an `.ini` file.
Save Control Bus Mix To Profile	N/A	Saves a control bus mix to a profile, serialized to an `.ini` file. If the mix is loaded, it uses the state of the current proxy. If not, it uses the default UObject representation.
Set...	**Control Bus Mix**, **Control Bus Mix By Filter**, **Global Control Bus Mix Value**, **Watched Modulator**	Sets properties on the provided object. With **Set Control Bus Mix By Filter**, you can use OSC-style address filtering.
Update...	**Control Bus Mix**, **Modulator**	Commits updates from the provided object to the audio thread.

There are helpful functions outside of the **Audio > Modulation** group as well. In particular, the **Get Modulators** and **Set Modulation Routing** functions provide overloads for Audio Component and Synth Component and can be used to get or set a component's Modulation Destinations.

For more detailed information, see the Audio Modulation Blueprint API Reference.

About UObjects and Proxies

When working with Audio Modulation components and assets, you typically edit UObject values. However, Audio Modulation does not directly use UObjects for processing for two reasons.

- UObjects run on the game thread.

- UObjects can be garbage collected at times that are out of the Audio Modulation system's control.

As a result, the set values in UObjects are sent to objects in the Audio Modulation system on the audio render thread. This is important to remember because some Blueprint functions edit the UObject, and others modify the proxy object in the Audio Modulation system.

When a function only modifies the UObject value, you need to call an **Update** Blueprint function to send those changes to the proxy. Also, if you call a function that does not modify the UObject value and then call a function to update the proxy, the UObject value will overwrite the proxy.

> Although Audio Modulation runs on the audio render thread, it runs at block rate instead of sample rate.