Impact SFX Synthesizer Reference Guide

Support: `https://discord.gg/aCXxZmNvxn`

Contents

1. Getting Started

1.1 What Can This Plugin Do?

From a single light tap on your coffee cup to a plate shatters into tiny pieces, this plugin allows you to synthesize all kinds of impact sounds in real time effortlessly. Making infinite SFX variations for gunshots or explosions right inside Unreal has never been easier. And with just a few clicks, you can also turn them into SFX for electric motors, whooshes, sci-fi, and UI instantly.

But why synthesize instead of playback? While synthesizing requires a bit more computational cost than decoding an audio file, there are several key benefits:

- **Much lower memory cost**: both storage and runtime memory costs are reduced compared to using pure wave files.
- **More variations, customizability, and reusability**: you can easily change the speed, pitch, weight, and material (wood, metal, glass, etc.) of your SFX anytime with little to no additional computation cost at extremely high quality.
- **More controllable**: Spending time to record or find stock SFX that fit your current application is no simple task. Some sounds are too long, too many pieces, or not in the right pitches. With this plugin, you can separate each SFX into individual layers and adjust their parameters to create the effect you need. Syncing your SFX to VFX has become much simpler.

To help you get started, 110+ MetaSounds Source (prefix: **MSS**) graphs (Preview), which cover 6 SFX categories, are included with the plugin (more will be added in future updates.)

- Door: opening/closing generic and special doors.
- Footsteps: bare feet, sandals, and shoes.
- Melee weapons: punching, sword, spear, and light saber.
- Range weapons: pistol, rifle, minigun, bow, laser, and energy gun.
- UI: simple beeps and clicks, dialog, and processing effects.
- Simulation: engine power going up, hold, and down. Breaking/shattering SFX generation.

All these MSS are fully procedural and can easily be customized to suit your application. They can be used as is or be a part of bigger SFX graphs. We also have a YouTube playlist to explain them in detail.

Lastly, everything you synthesize using this plugin and its **built-in** models is 100% yours. This means you can write them to wave files using MetaSounds nodes. Then use them in other programs or even sell them on marketplaces without having to worry about legal binding issues.

1.2 How Does It Work?

To use this plugin effectively, you need to know how it works. You don't need to understand all the things discussed here. Just having a general concept is enough. You will understand it better when you use the plugin in MetaSounds graphs.

Impact sounds synthesizing isn't something new. If you are a sound designer, you might have encountered it in other commercial DAWs or plugins such as footsteps generators, rain simulators, etc. Implementation details might differ, but the core theory is the same in all software.

Each impact sound can be synthesized by mixing a **physical model** with a **noise model**. In this plugin and this document, the physical model is called **modal**. And the noise model is called **residual**.

Thus, there are two types of synthesizers in the plugin:

- Modal synthesizers: Read parameters from impact modal objects (prefix: **IMO**) and use them in a mathematical model to generate new signals.
- Residual synthesizers: Decode data from residual objects (prefix: **RO**) to create new signals.

Objects with different weights, materials, thicknesses will have different modal and residual data. Currently, this plugin has 100+ modal objects and 180+ residual objects.

1.2.1 Modal Synthesizer - IMO

Each **IMO** object contains **multiple** modals. Each modal has **3** parameters: amplitude, frequency, and decay rate, which are used in the following equation (t is the time variable):

$$y_{modal}(t) = Amplitude \times sin(2\pi \times Frequency \times t) \times e^{-Decay \times t} \tag{1.1}$$

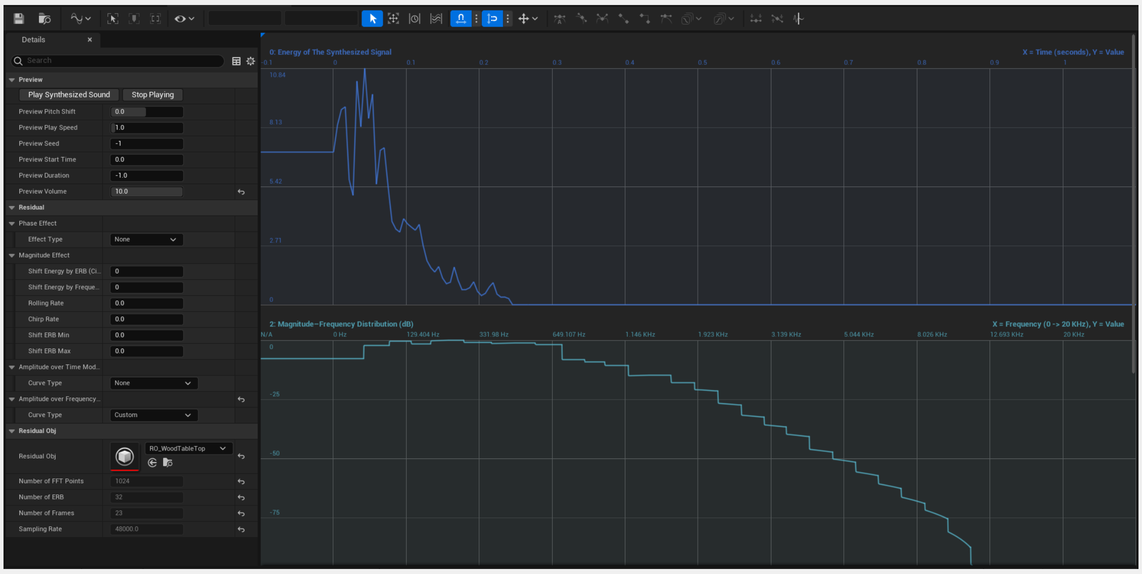
So if a modal synthesizer needs to synthesize an IMO object containing 16 modals, it must run through this equation 16 times and sum all their results together.

Modals capture most of the physical properties of an impact. Reducing decay rates will make an object sound more resonance/vibrating and increasing them will make it feel more solid/stiff. For example, you can change a wooden door knocking sound to a metallic one by simply changing the decay rate.

1.2.2 Residual Synthesizer - RD

Residual signals are those signals that couldn't be captured by our modals. They are usually left-over noises. In this plugin, we use an efficient noise encoding method to store their data (RO - **Residual Object**) and synthesize them by using a random generator.

Residual signals are what make a sound unique. Thus, the plugin allows you to edit them by using **Residual Data Editor** (**RD**):



Preview
Preview Pitch Shift 0.0
Preview Play Speed 1.0
Preview Seed -1
Preview Start Time 0.0
Preview Duration -1.0
Preview Volume 10.0
Residual
Phase Effect
Effect Type None
Magnitude Effect
Shift Energy by ERB (Ci... 0
Shift Energy by Freque... 0
Rolling Rate 0.0
Chirp Rate 0.0
Shift ERB Min 0.0
Shift ERB Max 0.0
Amplitude over Time Mod...
Curve Type None
Amplitude over Frequency...
Curve Type Custom
Residual Obj
Residual Obj RO_WoodTableTop
Number of FFT Points 1024
Number of ERB 32
Number of Frames 23
Sampling Rate 48000.0

Figure 1.1: Residual Data Editor.

Video tutorial. The top-right panel shows you the energy over time of the synthesized signal. The bottom-right panel shows the frequency distribution. Note that its frequency ranges are NOT divided linearly. Because normal people are less sensitive to changes in high frequencies, we use a non-linear function called ERB to divide frequency ranges and put more emphasis on the low and mid ranges.

On the left panels, from top to bottom:

- Preview section allows you to quickly preview the results with different pitches, speeds, and amplitudes.
- Residual section is the place where you can edit and add effects to the original data:
  - Phase Effect: use None for normal synthesizing. Constant and Saber when you need some sci-fi SFX.
  - Magnitude Effects: can dramatically transform an original SFX into a whole different kind of SFX.
    - Shift Energy by ERB: circularly shifts the entire frequency distribution of our signal. Its step size is based on ERB ranges.
    - Shift Energy by Frequency: circularly shifts the entire frequency distribution of our signal. Its step size is linear.
    - Rolling rate: a dynamic version of Shift Energy by Frequency. It circularly shifts the entire frequency distribution at every frame.
    - Chirp rate: linearly shifts the entire frequency distribution at every frame.
    - Shift ERB Min: shifts the lowest ERB value.
    - Shift ERB Max: shifts the highest ERB value.
  - Amplitude Over Time Modification Curve: allows you to modify the amplitude of the signal over time by using Unreal Rich Curve Editor.
  - Amplitude Over Frequency Modification Curve: allows you to modify the magnitude of each frequency. It's similar to an EQ. However, we use linear scale here and not dB. (To convert to dB, use $20 \times log_{10}(Scale)$.)
- Residual Obj section is the place to reference and display information about the original residual data.

All residual effects have minor to no additional computational costs.

You can turn your own wave files into residual objects inside Unreal by choosing **Create Residual Object** from their right mouse button menu. But note that:

- Residual synthesizer only works for noisy signals. SFX like explosions, gunshots, footsteps, wind, water, rocks falling, etc. usually have no problem because they're mostly noise. If your audio contains defining pitches, they will not be stored and can sound strange when synthesized.
- If you use stock audio from **another 3rd party**, please be aware that their license will still be applied even when you convert them to residual objects and synthesize them back to wave files. So make sure you don't do anything that violates their license (for example, selling the audio.)

1.2.3 Summary

A modal object captures the physical properties of an impact. Pitch and decay rate are parameters you can use to change the material and thickness of an impact sound.

Residual data captures the noisy part of an impact. Residual data is what makes a sound feel unique and realistic. You can do a lot of customization in **Residual Data Editor** to create your own SFX.

1.3 Audio Folder Structure

1. **ImpactData** stores all data needed for synthesizing:
   a. ModalObj (IMO): impact modal objects for modal synthesizing.
   b. MultiImpactData (MID): stores data used in MultiImpactSFXSynth node.
   c. ResidualData (RD): stores data that references ResidualObj and is used in residual synthesizer.
   d. ResidualObj (RO): original residual signals.
2. **MetaSoundPresets** stores all MetaSounds Source graphs:
   a. Doors: generic and special doors.
   b. FootSteps: **DryLeavesBranches** graphs simulate tree branches cracking sounds. In games, it's better to render these simulations to wave files.
   c. MeleeWeapons: whoosh sounds can be created through **Residual Data Editor**.
   d. Misc: utilities or experimental graphs.
   e. RangeWeapons: **MSS_Lyra_PistolFire** is an example of how to use the **Attenuation** interface of MetaSounds which is based on the pistol graph in Lyra (no asset from Lyra is used here.)
   f. Simulation: these graphs use MultiImpactSFXSynth node to simulate complex impacts such as a shattering plate or stepping on small tree branches.
   g. UI.

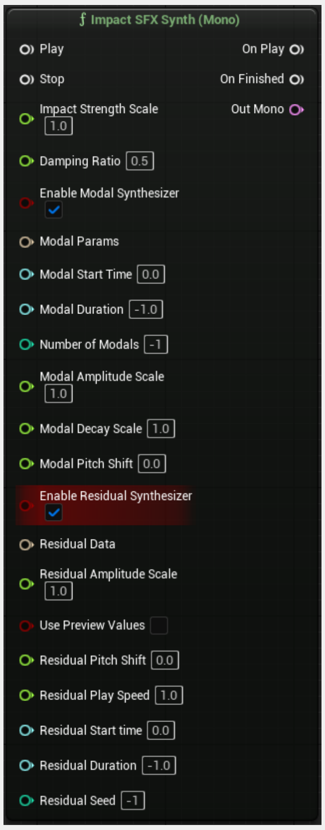
2. Core Nodes

2.1 Overview

Video Tutorials. Currently, this plugin adds 4 new native nodes into MetaSounds:

- **Impact SFX Synth**: synthesizes simple impact sounds.
- **Multi-Impact SFX Synth**: a more advanced version of the previous node. Optimized for complicated impact sounds such as a shattering plate.
- **Chirp Synth**: synthesizes sounds by using only modal parameters. It can be used for UI, sci-fi, and engine SFX.
- **Residual Synth**: synthesizes sounds by using only residual data. If you mix this node with Chirp Synth node, you have a less optimized version of ImpactSFXSynth node. Use this node for SFX where you don't need modal signals such as wind blowing, river streams flowing, earth quakes, engine noises, explosions, etc.

2.2 Impact SFX Synth Node


Impact SFX Synth (Mono)

Play — On Play
Stop — On Finished
Impact Strength Scale 1.0 — Out Mono
Damping Ratio 0.5
Enable Modal Synthesizer ✓
Modal Params
Modal Start Time 0.0
Modal Duration -1.0
Number of Modals -1
Modal Amplitude Scale 1.0
Modal Decay Scale 1.0
Modal Pitch Shift 0.0
Enable Residual Synthesizer ✓
Residual Data
Residual Amplitude Scale 1.0
Use Preview Values
Residual Pitch Shift 0.0
Residual Play Speed 1.0
Residual Start time 0.0
Residual Duration -1.0
Residual Seed -1

Figure 2.1: Impact SFX Synth node (mono version).

This node has mono and stereo versions. It synthesizes simple impact sounds by mixing modal and residual synthesizers. Good for SFX like footsteps, opening/closing doors, explosions, gunshots, etc.

2.2.1 Input Pins

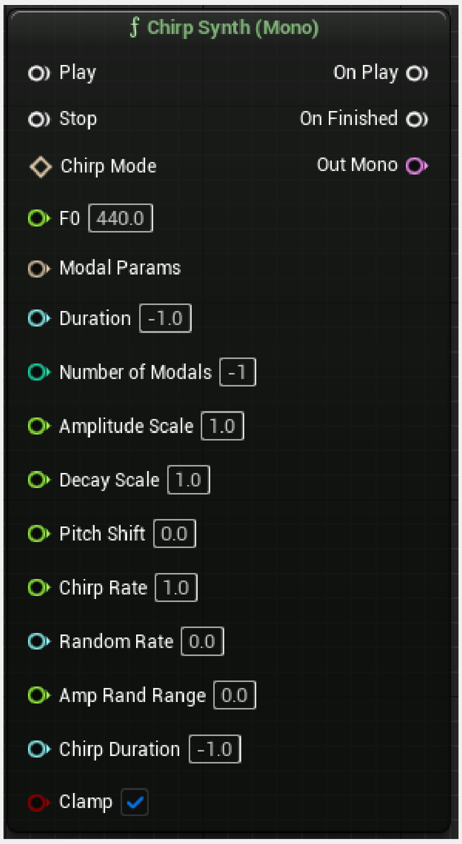
- **Play** (trigger): input event to start all synthesizers.
- **Stop** (trigger): input event to stop all synthesizers.
- **Impact Strength Scale** (float): scale the output of both modal and residual synthesizers.
- **Damping Ratio** (float): automatically scale the decay rate and residual play speed by combining this ratio with **Impact Strength Scale**. Solid/sitff objects should have this at 1.0. While hollow/resonance objects should be at 0. Leaving this at 0.5 is also fine for most applications.
- **Enable Modal Synthesizer** (bool): use modal synthesizer if true.
- **Modal Params** (ImpactModalObj): choose the ImpactModalObj you want to use.
- **Modal Start Time** (time): the start time in seconds. If negative, synthesize in reverse, which creates a pass by/whoosh effect.
- **Modal Duration** (time): determines how long modal synthesizer will run in seconds. If ≤ 0, synthesize until all modals decay to 0.
- **Number of Modals** (int): how many modals will be used. If ≤ 0, use all modals. Note that modals are sorted according to their total power. Thus, if an object has 16 modals and this is set at 8, modals from index 0 to 7 will be used.
- **Modal Amplitude Scale** (float): scale the amplitude of all modals. This must be larger than 0.
- **Modal Decay Scale** (float): scale the decay rate of all modals. Final decay rate is clamped between [0.1, 1000].
- **Modal Pitch Shift** (float): shift the pitch (frequency) of all modals in semitone. Range [-72, 72].
- **Enable Residual Synthesizer** (bool): use residual synthesizer if true.
- **Residual Data** (ResidualData): choose the ResidualData you want to use.
- **Residual Amplitude Scale** (float): scale the amplitude of ResidualData. Must be larger than 0.
- **Use Preview Values** (bool): if true, replace the values of all pins below this with those specified in **Residual Data Editor**. Useful for quick prototyping.
- **Residual Pitch Shift** (float): shift the pitch (frequency) in semitone. Range [-72, 72].
- **Residual Play Speed** (float): change the number of frames synthesized per second. Very similar to changing the playback speed of an audio file. Range [0.1, 10].
- **Residual Start Time** (time): the first frame to be synthesized in seconds. If negative, synthesize in reverse, which creates a pass-by/whoosh effect. Note that the start time (both negative and positive) can't exceed the final frame (automatically clamped internally.)
- **Residual Duration** (time): determines how long this residual synthesizer will run in seconds. If ≤ 0, it will run until it reaches the final frame.
- **Residual Seed** (int): the seed for the randomizer. If < 0, use a random seed each time it runs. This makes each sound has some minor perceptual variations.

For optimization reasons, changes to input values will only be registered when **Play** is triggered. In other words, you can't change the models, amplitude, play speed, etc. when the synthesizers are running.

2.2.2 Output Pins

- **On Play** (trigger): triggered on play event.
- **On Finished** (trigger): triggered when both modal and residual synthesizer have finished.
- **Out X** (float buffer): output the synthesized audio.

2.3 Chirp Synth Node



ƒ Chirp Synth (Mono)

Play — On Play
Stop — On Finished
Chirp Mode — Out Mono
F0 440.0
Modal Params
Duration -1.0
Number of Modals -1
Amplitude Scale 1.0
Decay Scale 1.0
Pitch Shift 0.0
Chirp Rate 1.0
Random Rate 0.0
Amp Rand Range 0.0
Chirp Duration -1.0
Clamp ✓

Figure 2.2: Chirp Synth node.

Aside from synthesizing modals, this node also allows you to change their frequency over time, which is usually called chirp effect in SFX. When pitches are outside of the [20Hz, 20000Hz] range, their amplitude is reduced to 0.

This node doesn't have a stereo version. For UI SFX, mono is usually enough. But if you want to create stereo audio and this node is the only node in your graph, a common trick is to add a delay node before the output of the second channel. However, if you mix it with other stereo synthesizer nodes, you can just plug this into the left and right channels directly.

2.3.1 Input Pins

- **Play** (trigger): input event to start the synthesizer.
- **Stop** (trigger): input event to stop the synthesizer.
- **Chirp Mode** (enum): Linear, sigmoid, and exponent sweeps are supported. Linear mode can be used

for sci-fi or UI effects. Exponent mode is used to simulate engine power going up or down. Sigmoid mode is similar to exponent but has an overshooting effect.

- **F0** (float): allows you to specify directly the frequency you want to use instead of having to use a modal object. Note that this pin is ignored if you plug a modal object into **Modal Params** pin.
- **Modal Params** (ImpactModalObj): choose the ImpactModalObj you want to use. This pin will override the value in **F0** pin.
- **Duration** (time): determines how long modal synthesizer will run in seconds. If ≤ 0, synthesize until all modals decay to 0.
- **Number of Modals** (int): how many modals will be used. If ≤ 0, use all modals. Note that modals are sorted according to their total power. Thus, if an object has 16 modals and this is set at 8, modals from index 0 to 7 will be used.
- **Amplitude Scale** (float): scale the amplitude of all modals. This must be larger than 0. Can be changed while synthesizing.
- **Decay Scale** (float): scale the decay rate of all modals. Must be ≥ 0. Can be changed while synthesizing.
- **Pitch Shift** (float): shift the pitch (frequency) of all modals in semitone. Range [-72, 72]. Can be changed while synthesizing.
- **Chirp Rate** (float): pitches changing rate of all modals. Can be changed while synthesizing.
- **Random Rate** (time): the time interval in seconds to add some randomization to each modal. This is usually used when simulating engine sound. If ≤ 0, no randomization is used.
- **Amp Rand Range** (float): Percentage in the range [0., 1.f]. the amount of randomization to add to or subtract from the amplitude of each modal. Each modal will be randomized by a different value within this range. This is only used if Random Rate > 0.
- **Chirp Duration** (time): the time to sweep all pitches. Can be changed while synthesizing. If $<$ 0, linear mode will sweep infinitely. But exponent and sigmoid mode will **NOT** sweep. For engine simulation, you can set **Chirp Rate** at 1, and use this to determine how long the power up/down duration will last.
- **Clamp** (bool): clamps the output between [-1, 1] or not.

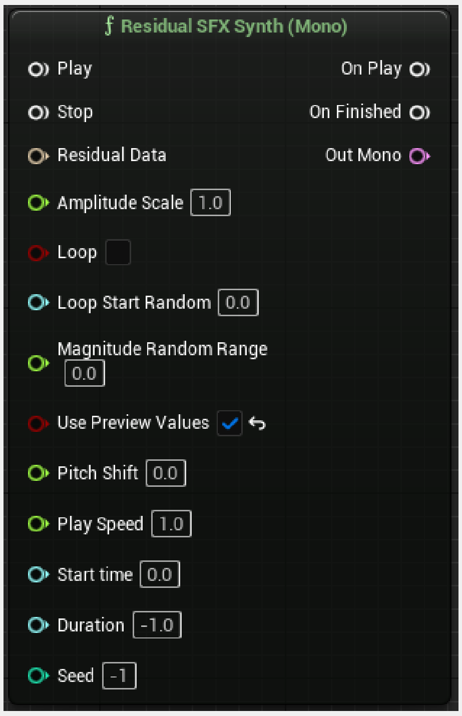
R This node is optimized so that what you don't use won't be calculated. For example, if you set **Decay Scale** at 0, then it won't calculate $e^{-Decay \times t}$. Similarly, if **Chirp Rate** is zero, then no sweeping equation is used.

R As you can change its amplitude and decay scale when this node is running, a pin for start time is unnecessary. Thus, start time is always 0.

2.3.2 Output Pins

- **On Play** (trigger): triggered on play event.
- **On Finished** (trigger): triggered when all modals decay to zero.
- **Out Mono** (float buffer): output the synthesized audio.

2.4 Residual SFX Synth Node



ƒ Residual SFX Synth (Mono)

Play — On Play
Stop — On Finished
Residual Data — Out Mono
Amplitude Scale 1.0
Loop
Loop Start Random 0.0
Magnitude Random Range 0.0
Use Preview Values ✓
Pitch Shift 0.0
Play Speed 1.0
Start time 0.0
Duration -1.0
Seed -1

Figure 2.3: Residual SFX Synth node.

This node has mono and stereo versions. It's useful when you want to add an extra noise layer or texture to your SFX.

2.4.1 Input Pins

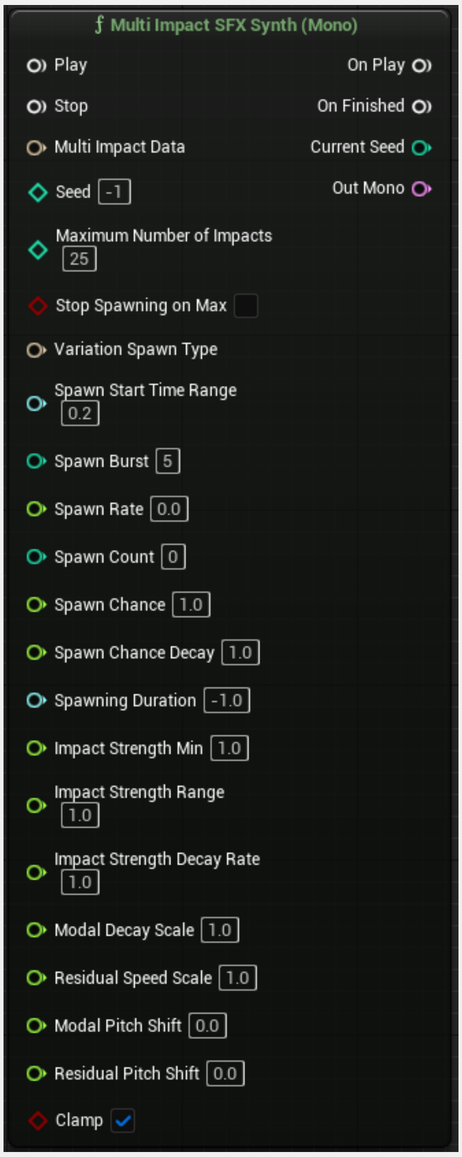
- **Play** (trigger): input event to start the synthesizer.
- **Stop** (trigger): input event to stop the synthesizer.
- **Residual Data** (ResidualData): choose the ResidualData you want to use.
- **Amplitude Scale** (float): scale the amplitude of output. Must be larger than 0.
- **Loop** (bool): if true, loop infinitely.
- **Loop Start Random** (time): randomize the start time in seconds. True Start Time = Start Time + Rand(0, Loop Start Random). This is very useful when looping to avoid audible repetition.
- **Magnitude Random Range** (float): the magnitude of each frequency band is scaled randomly in this range.
- **Use Preview Values** (bool): if true, replace values of all pins below this with the values specified in **Residual Data Editor**. Useful for quick prototyping.
- **Pitch Shift** (float): shift the pitch (frequency) in semitone. Range [-72, 72].
- **Play Speed** (float): change the number of frames synthesized per second. Very similar to changing the play back speed of an audio file. Range [0.1, 10].

- **Start Time** (time): the first frame to be synthesized in seconds. If negative, synthesize in reverse, which creates a pass-by/whoosh effect. Note that the start time (both negative and positive) can't exceed the final frame (automatically clamped internally).
- **Duration** (time): determines how long this residual synthesizer will run in seconds. If ≤ 0, it will run until we reach the final frame.
- **Seed** (int): the seed for the randomizer. If < 0, use a random seed each time it runs. This makes each sound has some minor perceptual variations.

2.4.2 Output Pins

- **On Play** (trigger): triggered on play event.
- **On Finished** (trigger): triggered when all modals decay to zero.
- **Out X** (float buffer): output the synthesized audio.

2.5 Multi Impact SFX Synth Node



ƒ Multi Impact SFX Synth (Mono)

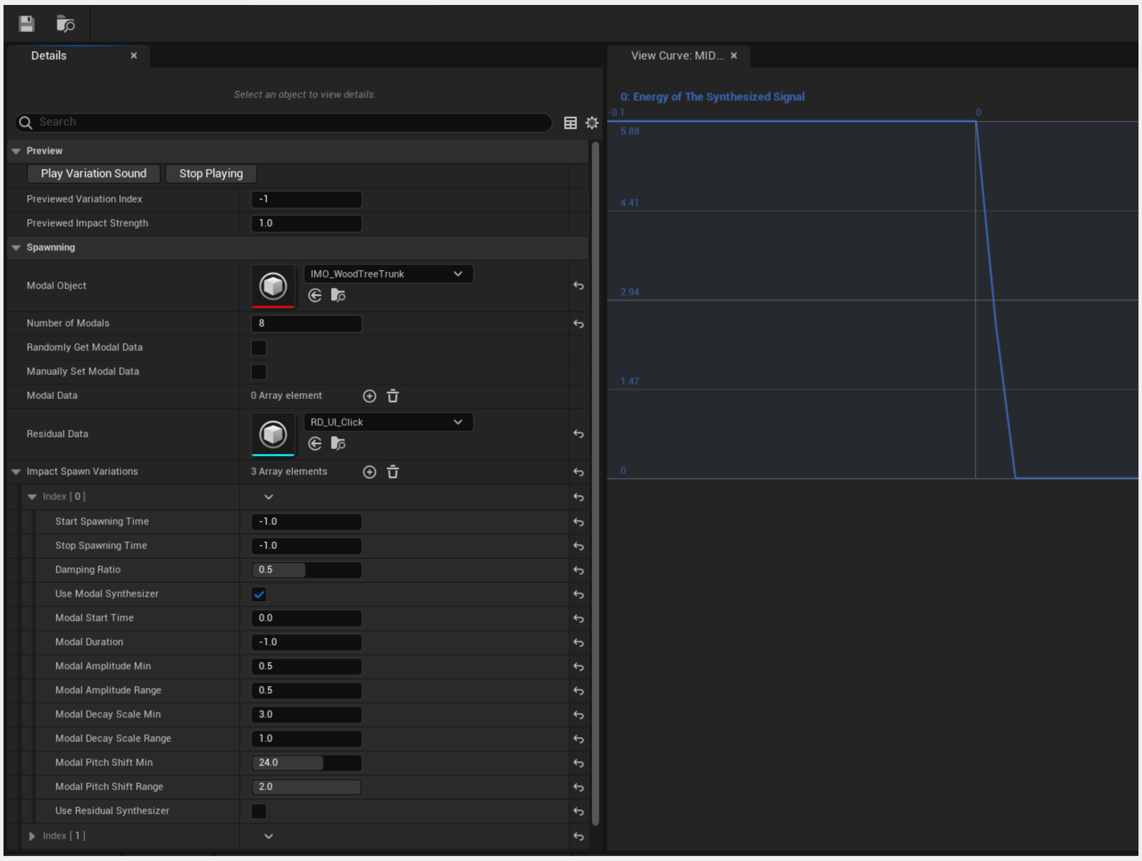
Play	On Play
Stop	On Finished
Multi Impact Data	Current Seed
Seed -1	Out Mono
Maximum Number of Impacts 25	
Stop Spawning on Max	
Variation Spawn Type	
Spawn Start Time Range 0.2	
Spawn Burst 5	
Spawn Rate 0.0	
Spawn Count 0	
Spawn Chance 1.0	
Spawn Chance Decay 1.0	
Spawning Duration -1.0	
Impact Strength Min 1.0	
Impact Strength Range 1.0	
Impact Strength Decay Rate 1.0	
Modal Decay Scale 1.0	
Residual Speed Scale 1.0	
Modal Pitch Shift 0.0	
Residual Pitch Shift 0.0	
Clamp ✓	

Figure 2.4: Multi Impact SFX Synth node.

A lot of impact sounds have multiple impacts instead of just one, e.g., sword clashes, gun reloading, etc. While you can try and use multiple ImpactSFXSynth nodes to create multiple impacts, it's not optimized and makes the final graph become a mess. By using this node, you avoid those issues.

This is a very advanced node and can be hard to use for new users, especially those without experience in sound design. Please always look for MSS in the simulation folder first to see if you can reuse them to avoid setting up everything from scratch. In future updates, we will add more MSS for this node.

2.5.1 Multi Impact Data Editor



Preview	
Previewed Variation Index	-1
Previewed Impact Strength	1.0
Spawnning	
Modal Object	IMO_WoodTreeTrunk
Number of Modals	8
Randomly Get Modal Data	☐
Manually Set Modal Data	☐
Modal Data	0 Array element
Residual Data	RD_UI_Click
Impact Spawn Variations	3 Array elements
Index [0]	
Start Spawning Time	-1.0
Stop Spawning Time	-1.0
Damping Ratio	0.5
Use Modal Synthesizer	✓
Modal Start Time	0.0
Modal Duration	-1.0
Modal Amplitude Min	0.5
Modal Amplitude Range	0.5
Modal Decay Scale Min	3.0
Modal Decay Scale Range	1.0
Modal Pitch Shift Min	24.0
Modal Pitch Shift Range	2.0
Use Residual Synthesizer	☐
Index [1]	

Figure 2.5: Multi Impact Data Editor.

To use MultiImpactSFXSynth node, you must set up a multi impact data object which is an object used to encapsulate both modal object and residual data, before passing them into MetaSounds.

The most important section in Multi Impact Data Editor is **Impact Spawn Variations**, which is actually an array. Each variation stores parameters for both modal and residual synthesizers. In MetaSounds graph, these variations will be retrieved, and synthesizers will be created from their parameters to create different impact sounds.

Similar to Residual Data Editor, the right panel shows a synthesized signal from **Impact Spawn Variations** section. On the left panel, from top to bottom:

- **Preview** section allows you to quickly preview a variation. If variation index < 0, a random variation from **Impact Spawn Variations** is synthesized.
- **Spawning** section:
  - **Modal Object**: the modal object to use for all variations.
  - **Number of Modals**: number of modals from object will be used.
  - **Randomly Get Modal Data**: If true, instead of always starting from modal 0, modals are retrieved randomly each time a variation is spawned.
  - **Manually Set Modal Data**: use modal parameters specified in **Modal Data** field below instead of retrieving them from a modal object. Useful for UI sounds when you need to control the frequency directly.
  - **Modal Data**: an array of modal parameters that you can set manually.

  - **Residual Data**: the residual data to use for all variations.

- **Impact Spawn Variations**: an array of variations that decide how modal and residual data are synthesized. Each variation will have the following fields:
  - **Start Spawning Time**: this variation can only be spawned after this time. If < 0, always spawn.
  - **Stop Spawning Time**: this variation won't be spawned after this time. If < 0, always spawn.
  - **Damping Ratio**: same meaning as in ImpactSFXSynth node.
  - **Use Modal Synthesizer**: will this variation use a modal synthesizer or not.
    - **Modal Start Time**: starting time of the modal synthesizer using this variation in seconds.
    - **Modal Duration**: how long a modal synthesizer will run for this variation in seconds.
    - **Modal Amplitude Min**: the minimum amplitude scale of this variation.
    - **Modal Amplitude Range**: amplitude scale is randomized in this range. Modal Amplitude Scale = Modal Amplitude Min + Rand(0, Modal Amplitude Range).
    - **Modal Decay Min**: the minimum decay scale of this variation.
    - **Modal Decay Range**: decay scale is randomized in this range. Modal Decay Scale = Modal Decay Min + Rand(0, Modal Decay Range).
    - **Modal Pitch Min**: the minimum pitch shift of this variation.
    - **Modal Pitch Range**: Pitch shift is randomized in this range. Modal Pitch Shift = Modal Pitch Min + Rand(0, Modal Pitch Range).
  - **Use Residual Synthesizer**: will this variation use a residual synthesizer or not.
    - **Residual Start Time**: starting frame in seconds.
    - **Residual Duration**: duration of a residual synthesizer using this variation in seconds.
    - **Residual Amplitude Min**: the minimum amplitude scale of this variation.
    - **Residual Amplitude Range**: amplitude scale is randomized in this range. Residual Amplitude Scale = Residual Amplitude Min + Rand(0, Residual Amplitude Range).
    - **Residual Speed Min**: the minimum speed scale of this variation.
    - **Residual Speed Range**: speed scale is randomized in this range. Residual Speed Scale = Residual Speed Min + Rand(0, Residual Speed Range).
    - **Residual Pitch Min**: the minimum pitch shift of this variation.
    - **Residual Pitch Range**: pitch shift is randomized in this range. Residual Pitch Shift = Residual Pitch Min + Rand(0, Residual Pitch Range).

R The maximum value of **Pitch Range** is limited to 2 because it's much better to add more variations with different minimum pitch shifts than using a big shift range. For example, if you want your variations to shift pitches in the range [0, 10]. Then it's better to make 5 variations with the following minimum pitch shift: 0, 2, 4, 6, 8, and a pitch shift range of 2. This helps each synthesizer sound more distinctive. Otherwise, if we make a variation with a minimum pitch shift at 0 and a pitch shift range of 10, then the synthesized audio will sound muddy, chaotic, and unrealistic.

2.5.2 Input Pins

- **Play** (trigger): input event to start all synthesizers.
- **Stop** (trigger): input event to stop all synthesizers.
- **Multi Impact Data** (MultiImpactData): choose the MultiImpactData you want to use.
- **Seed** (int): the seed to be used in the randomizer when spawning synthesizers. This seed is NOT passed down to residual synthesizers. If < 0, a random seed is used.

- **Maximum Number of Impacts** (int): the maximum number of impact synthesizers available in the pool.
- **Stop Spawning on Max** (bool): If true, when we've reached the allowed maximum number of impacts, no new impact will be spawned until an old impact has finished. If false, the weakest impact in the current pool will be removed to spawn a new impact.
- **Variation Spawn Type** (enum): determines the way variations are retrieved from MultiImpactData. 4 types are supported: Random, Increment, Decrement, and Ping Pong.
- **Spawn Start Time Range** (time): when a new synthesizer is spawned, it might be delayed with a random duration before it can actually start creating signals. This is done to make impacts sound more organic. This pin determines the maximum range of this delay in seconds.
- **Spawn Burst** (int): the number of impacts to spawn when starting this node.
- **Spawn Rate** (float): spawning interval per second. A spawn rate of 10 means we spawn new impacts every 1/10.0 = 0.1 second.
- **Spawn Count** (int): the number of impacts to spawn at each spawning interval.
- **Spawn Chance** (float): Range [0, 1]. The chance at which a new impact is spawned. For example, if we reach a spawning interval and the spawn count is 5, then a spawn chance of 0.5 means only 2.5 impacts are actually spawned on average.
- **Spawn Chance Decay** (float): ≥ 0. Spawn chance is reduced over time by this decay rate. New Spawn Chance = Spawn Chance $\times e^{-Decay \times t}$.
- **Spawning Duration** (float): duration to spawn new impacts in seconds. If < 0, spawning until spawn chance reaches 0, or spawn rate reaches 0 if you change its value when running, or when impact strength decays to 0.
- **Impact Strength Min** (float): the minimum impact strength applies to modal and residual synthesizers.
- **Impact Strength Range** (float): each variation impact strength = Impact Strength Min + Rand(0, Impact Strength Range).
- **Impact Strength Decay Rate** (float): the decay rate of impact strength. New Impact Strength = Impact Strength $\times e^{-Decay \times t}$.
- **Modal Decay Scale** (float): the global decay scale applied to the modal synthesizer of all variations. This allows you to shift the decay rate of all variations without having to make a new multi impact data.
- **Residual Speed Scale** (float): the global speed scale applied to the residual synthesizer of all variations.
- **Modal Pitch Shift** (float): the global pitch shift applied to the modal synthesizer of all variations.
- **Residual Pitch Shift** (float): the global pitch shift applied to the residual synthesizer of all variations.
- **Clamp** (bool): clamp the output between [-1, 1] or not. This should normally be true, as the sum of all synthesizers can be really loud.

2.5.3 Output Pins

- **On Play** (trigger): triggered on play event.

- **On Finished** (trigger): triggered when no new synthesizer can be spawned (reaching Spawning Duration, or Spawning Rate is 0, or Spawning Chance decays to 0, or Impact Strength decays to 0) and all current synthesizers have finished.
- **Current Seed** (int): the seed used to synthesize the previous audio. Useful if you want to play randomly and find a seed with a desirable result. Need to connect this node to a Print Int node if you want to read its value.
- **Out X** (float buffer): output the synthesized audio.

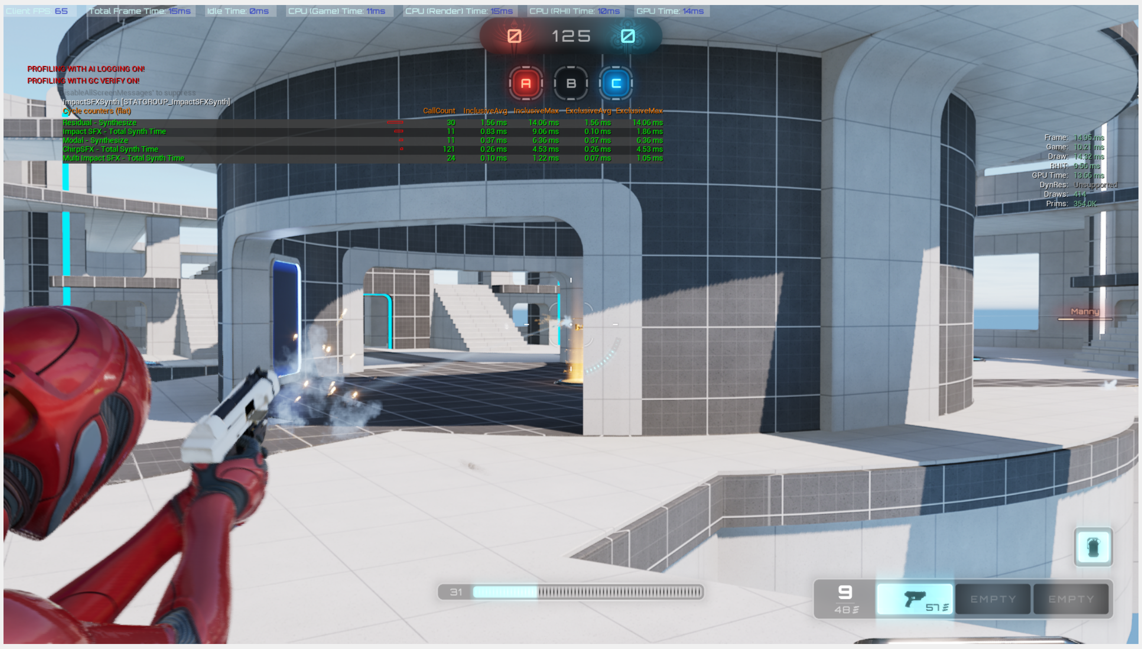
3. Performance Optimization

3.1 Impact SFX Synth Stats

3.1.1 CPU Load

MetaSounds renders audio on the audio thread instead of the game thread, which allows it to do some heavy stuff without tanking your FPS (as long as you don't build a graph with 80% CPU load). Starting with Unreal Engine 5.3, Epic lets you see your CPU load when a MetaSounds graph is playing. This is a very good indicator of how synthesizers perform.

Additionally, when playing your game in the editor or in a development build, you can use the console command: **stat ImpactSFXSynth** to view the computational costs of all running synthesizers.



```
Client FPS: 65   Total Frame Time: 15ms   Idle Time: 0ms   CPU (Game) Time: 11ms   CPU (Render) Time: 15ms   CPU (RHI) Time: 10ms   GPU Time: 14ms

PROFILING WITH AI LOGGING ON!
PROFILING WITH GC VERIFY ON!
isableAllScreenMessages' to suppress
ImpactSFXSynth [STATGROUP_ImpactSFXSynth]
Cycle counters (flat)                          CallCount  InclusiveAvg  InclusiveMax  ExclusiveAvg  ExclusiveMax
Residual - Synthesize                                 30       1.56 ms      14.06 ms       1.56 ms      14.06 ms
Impact SFX - Total Synth Time                         11       0.83 ms       9.06 ms       0.10 ms       1.86 ms
Modal - Synthesize                                    11       0.37 ms       6.36 ms       0.37 ms       6.36 ms
ChirpSFX - Total Synth Time                          121       0.26 ms       4.53 ms       0.26 ms       4.53 ms
Multi Impact SFX - Total Synth Time                   24       0.10 ms       1.22 ms       0.07 ms       1.05 ms

    Frame:  14.95 ms
     Game:  10.21 ms
     Draw:  14.32 ms
     RHIT:  9.56 ms
 GPU Time:  13.66 ms
  DynRes:  Unsupported
    Draws:  414
    Prims:  354.0K
```

Figure 3.1: Impact SFX Synth plugin stress test on a development build of Lyra project. Original SFX for footsteps, gunshots, guns reloading, bullet whiz, and hits are replaced with graphs using the plugin. (All of them were playing when the screenshot was taken.)

In the figure above, there are 5 stats:

- Modal - Synthesize: stats of modal synthesizers from ImpactSFXSynth and MultiImpactSFXSynth nodes.
- Residual - Synthesize: stats of residual synthesizers from ImpactSFXSynth, MultiImpactSFXSynth, and ResidualSFXSynth nodes.
- Impact SFX - Total Synth Time: stats for both modal and residual synthesizers of ImpactSFXSynth node.

- Multi Impact SFX - Total Synth Time: stats for both modal and residual synthesizers of MultiImpactSFXSynth node.
- ChirpSFX - Total Synth Time: stats of ChirpSynth node.

Most of the time, our synthesizers won't be the bottleneck for your FPS. On simple graphs, they can even be faster than playing wave files. For example, using ChirpSynth node for simple UI beeps is much faster than playing WAV.

MultiImpactSFXSynth is the node which can create performance issues, e.g., using this node to simulate the sound effects of bullets. In a shooting game with hundreds of gunshots per second, it's better to render those simulations into wave files. In the test above, we use this node for gun reloading and when bullets hit glasses.

3.1.2 Latency

Standard latency for audio hardware is below 20ms (except for Bluetooth headsets. Cheap ones have extremely bad latency). And normal people can detect an audio latency if it's above 40-50ms. In other words, if your SFX is delayed by more than 50ms compared to your VFX, then players will feel like they're out of sync.

Thus, latency is also a very important parameter. **InclusiveMax** might somewhat indicate the maximum latency of synthesizers (when only 1 SFX is played). Unfortunately, it isn't very accurate. The best way to measure latency is to record your game. Then view the delay between your VFX and SFX in a video editor. This ignores the latency of your own speakers/headphones.

Overall, through our stress test on Lyra (i5-6500 CPU @ 3.20GHz), there aren't any latency issues by using the plugin. Even with a complicated graph such as the pistol gunshot graph, which triggers 1 MultiImpactSFXSynth, 1 ImpactSFXSynth, 2 ResidualSFXSynth, and 2 ChirpSynth nodes. In a shooting game like Lyra, this graph is spawned multiple times per second. But if you need to improve the latency, here are a few things you can do (ordered from best to worst):

- In a graph, avoid triggering too many synthesizers at the same time. You can use a Trigger Delay node to add a few 10ms (block rate) delays to improve the performance, as long as this doesn't affect the quality of your SFX.
- If a graph has many residual synthesizers, you can merge them into one by mixing their output and writing them to a wave file. Then import that wave file back into Unreal, right-click, and choose **Create Residual Object**. Finally, replace those residual synthesizers with a node that uses the new file. This improves both your latency and computational costs.
- Currently, MetaSounds uses audio buffers with a block rate of 480, and the sampling frequency is 48000Hz. So for each audio frame, we have to synthesize 10ms (480 / 48000) of audio before playing. To improve latency, lower your block rate. For example, change it to 240 with this command: **au.MetaSound.BlockRate 240**. Though this comes with more CPU costs and isn't very effective for residual synthesizers due to how they decode. Also if the value is too small, there can be some pop noise.

R You might wonder why computing 5ms of audio costs more than 10ms. Well, if only synthesizers are considered, then synthesizing 10ms of audio is still a tiny bit faster than doing 5ms 2 times. Then, to play the audio, there is still a lot of low-level stuff like copying data to audio buffers, scheduling, etc., which is another deep rabbit hole and outside the scope of this document.

3.2 Performance Tips

These are just suggestions, not hard rules that you must follow all the time. Your performance won't take a sudden dip even if you multiply outputs with some float values.

- Triggering 3-4 synthesizers at the same time in a graph is fine. If more than that, you should consider merging or adding some trigger delay (> 10ms) to them to avoid hitches on playing. (This is also actually true even if you just play pure audio files.)

- The cost of a modal synthesizer increases linearly with its number of modals. Thus, on SFX that are common in your game or in MultiImpactSFXSynth, try to use the lowest number of modals without scarifying too much quality.

- All synthesizers allow you to change their gain/amplitude through the Amplitude Scale or Impact Strength pin. Amplitude is also used to drive some optimizations internally. So please use it to control the gain of your audio instead of multiplying the output with a float or through Audio Component (you can change MetaSounds input from C++ or Blueprints). For example, instead of specifying 0.5 gain on your mixer node, just put this value on the amplitude pin instead.

- If you use an amplitude curve over time in Residual Data Editor to shorten the duration (ex. scale amplitude to 0 after 100ms), in MetaSounds graph, you should also specify your desired residual duration (ex. 100ms), so it doesn't waste CPU resources by synthesizing silent frames.

- When encoding wave files to ResidualObj (by using **Create Residual Obj**), the plugin automatically removes silent frames from the start and end of your audio. If your files contain audible silent sections in the middle, it's better to split them into multiple files (using an external DAW) before encoding. Then you can use Trigger Delay nodes to control their timing when synthesizing. This not only makes your SFX more reusable but also avoids wasting CPU resources to synthesize silent frames.

- A common trick in Multi Impact Data Editor is to use modal and residual synthesizers alternatively. For example, the first variation only uses a modal synthesizer. The second variation only uses a residual synthesizer. Then repeat (it doesn't have to be in order if you use Random retrieve mode). This can actually make it sound more realistic in some situations while reducing the total number of synthesizers running at the same time.