FMOD+Unity Integration BASIC CODE

What you need to know to play and manipulate FMOD-driven audio events in a Unity game.

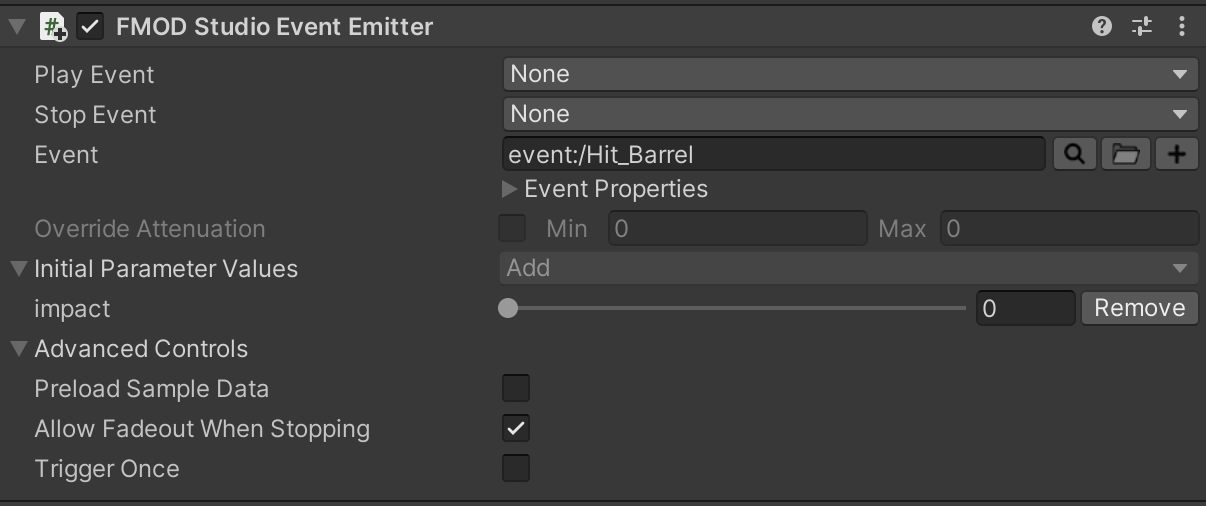
FMOD Integration Package, Event Emitters, and C# Code

It is possible to attach FMOD Events to Unity game objects using Unity's Inspector, but you can gain greater flexibility when you "write" some code. Why "write"? Because it's usually a simple process to copy/paste and change Event and variable names to suit your particular needs. These snippets should provide the code you need to get started with parameter-based sound effects and music designs.

Copy these code snippets and paste them into the C# files for your game. Names in green are simply placeholders and should be changed to reflect the unique Event names and Parameters you define in your own project.

Play FMOD Event with a Studio Event Emitter Component

The Studio Event Emitter Component will be available in Unity after you install the FMOD Integration package (steps covered in the video tutorials linked below). Attach the component to an appropriate game object and use the available menu options to configure its playback:



* **Play Event:** what event cues the sound
* **Stop Event:** what event stops the sound
* Options for Play and Stop include Object Start, Object Destroy, Trigger Enter, Trigger Exit, Trigger Enter 2D, Trigger Exit 2D, Collision Enter, Collision Exit, Collision Enter 2D, Collision Exit 2D, Oject Enable, Object Disable, and None (default).
* **Event:** which event from the FMOD Studio project should play
* Additional aspects of a sound that can be further modulated by conditions in the game as it is played

Cue an Attached Emitter Sound with C# code

This technique is useful in situations where an identical C# script is used multiple times; attached to several different game objects that all must have unique sounds. For example, if your game had many different targets in a shooting gallery game, each could have the same "hit target and fall over" behavior script but would need to play a unique sound when hit by a projectile: rubber duck squeaks, car honks, bottle shatters, and so on.

Declare the emitter reference

Do this in the class definition—the same place where variables are declared at the top of a C# script.

FMODUnity.StudioEventEmitter eventEmitterRef;

Retrieve the Event Emitter Component

By definition for this technique it is attached to a game object. Here, the Awake() method ensures that this happens when the script instance is loaded.

void Awake()

{

eventEmitterRef = GetComponent<FMODUnity.StudioEventEmitter>();

}

Play the sound

This example shows how to do it when two game objects collide.

void OnTriggerEnter(Collider other)

{

if (other.gameObject.CompareTag("Projectile"))

{

eventEmitterRef.Play();

}

}

Play FMOD Event: Play a FMOD Event with C# scripting in Unity.

Declare an Event instance

Do this in the class definition—the same place where variables are declared at the top of a C# script. The declaration will allow you to manage the behavior of the event: play, modulate parameters, and stop.

FMOD.Studio.EventInstance gameMusic;

Create the instance

Once created, associate it with an event from your FMOD project. The Start() method is a good place to do this so that the instance is ready to go immediately when the game loads

void Start()

{

gameMusic = FMODUnity.RuntimeManager.CreateInstance("event:/game\_music\_path");

}

Start and stop the FMOD Event

Use start() to play the event instance.

// use this just about anytime when playback should commence...

gameMusic.start();

// ...such as when the player presses an input key

if (Input.GetButtonDown ("Horizontal"))

{

gameMusic.start();

}

Use stop() to halt the event instance.

if (Input.GetButtonDown ("Horizontal"))

{

gameMusic.stop();

}

FMOD's AHDSR Envelope is one of the best techniques to stop a sound with a smooth transition to silence. Rather than stop the sound effect or music track with an abrupt silence, you can use the AHDSR Envelope to gradually lower the volume over time. STOP\_MODE.ALLOWFADEOUT uses the AHDSR modulation envelope on the FMOD Studio Event's Master volume knob to set a fade time and shape.

void ResetGameLevel()

{

gameMusic.stop(FMOD.Studio.STOP\_MODE.ALLOWFADEOUT);

}

Modulate the Event using a parameter

Once the sound is playing, you can use data from the game to control parameters in real-time. "ParameterName" is the name you've assigned in FMODStudio; parameter\_value is a number (int or float) derived from the game (health, score, remaining enemies, and so on).

void UpdateScore ()

{

gameMusic.setParameterByName("ParameterName", parameter\_value);

}

Play a OneShot Sound

OneShot sounds are "fire and forget"—they play immediately and are released from memory when done. When cueing a 3D sound you can use an optional second argument to have the sound play at its current position in the game world (transform.position).

// spawn new; play sound

void SpawnNewRay()

{

FMODUnity.RuntimeManager.PlayOneShot("event:/sound\_event\_path");

}

// spawn new; play sound at the position of the game object to which this script is attached

void SpawnNewRayInWorld()

{

FMODUnity.RuntimeManager.PlayOneShot("event:/sound\_event\_path", transform.position);

}

It's also possible to expose the sound in the Unity Inspector and reference with FMOD's full path. This approach allows Events to be changed in Unity without having to touch the related C# file. First create a string variable to store sound event full path. This goes at the top of the script along with all other variable declarations.

public string oneShotForRaySound;

An empty text field will then appear in Unity Inspector. Copy the full path (something like event:/sound\_event) from the FMOD Event tab and paste it into the field. Playing the sound elsewhere in the script is simply a matter of using the variable name within the PlayOneShot method.

void SpawnNewRay()

{

FMODUnity.RuntimeManager.PlayOneShot(oneShotForRaySound);

}