

## **Audio systems guide**

Note: This is not for submission, but a basis for guides that would be utilised for testing and getting people to set up their own scenes for using the systems. (Would be iterated over and refined before that point though!)

**Getting started: 2**

**One-shot and parameter static class references: 3-5**

**One-Shot example: 6 -9**

**Occlusion system: 11-16**

**Dialogue System: 17-20**

## Fmod for Unity sound designer tool package – User guide

Contents:

Introduction:

### Installing fmod for Unity:

While fmod support tool creations that work with fmod for Unity and the fmod studio application, re-distribution of fmod scripts is not permitted. Therefore, to utilise this package you must also install the fmod for unity package and manually link the banks provided to drive the audio in the test scenes which highlight use of the project.

1. Download fmod for Unity from the asset store:  
<https://assetstore.unity.com/packages/tools/audio/fmod-for-unity-161631>
2. Open a Unity project (2021 or above) and navigate to Window->Package manager then with My assets selected, search for 'Fmod for Unity' then install and import the package.
3. Disable the Unity built-in audio and follow the fmod set up wizard to your specifications (note: if wanting to use audio provided with the system examples, utilise single or multiple banks mode in the linking section of the wizard. The location to the example banks will be provided in the following section)
4. The Fmod for Unity package is now set up and the audio tools provided with this system can now be utilised.

### Getting started:

If unfamiliar with the systems as part of this package, then there are various test scenes that show example usage of the systems at work and the fmod banks utilised for these test scenes are provided by the package. If not wanting to create your own project from scratch, but first view the audio provided in the test scenes with the package, then you can link your unity project to the fmod banks provided. These banks are provided in the assets folder in a folder named 'Banks' and can be linked to through the fmod for Unity set-up wizard, in the linking section. Select single platform build and navigate to UnityProjectDirectory->Assets->Banks to link to the banks provided with the package.

The package comes with three scenes that aim to highlight simple example uses of the systems at work, in a lightweight fashion, without bloating your project with a large quantity of un-needed third-party assets, such as object meshes and materials. So, while the scenes are relatively bare bones, they provide examples on how to use the system, while keeping the package at a small memory size. These scenes serve both as an example to be used independently, and as a use case for a tutorial that will be provided within this documentation.

### Example scenes provided: (may need names changed)

-One-Shot and parameter control: highlights the use of the fire and forget with parameters system and utility script to control parameters of looping audio.

-Spatial audio system: highlights the use of the occlusion system provided with this package, as well as how it can be incorporated into other acoustical effects provided by fmod, such as reverb.

-Dialogue scene: highlights all capabilities of the dialogue system, as well as triggering methods. Also highlights how the developed player response system can be utilised to drive player response driven conversations through the implemented dialogue NPC database that comes with this system.

## **Adapting audio with the parameter's utility SDK**

Sound design is a vast topic, as is software development. While fmod provides an API that allows sound designers and implementers the ability to adapt the parameters of audio events, this requires knowledge of the workings of the fmod for Unity APIs, as well as the software knowledge to be able to implement it in ways which remove the need for repetition and support the breadth of functionality required for creating adaptable audio. This system aims to overcome this barrier by providing an SDK for controlling fmod for Unity event parameters. This SDK encapsulates all fmod for Unity API relating to event parameters into a global context, which can be used in any part of your code base! This SDK utilises different programming aspects to create a concise set of scripts and functions that can handle all parameter functionality, while removing the need to implement any of that API yourself. This system comes in the form of two static classes, contained across two scripts, which contain various functions. An SDK reference for these scripts and descriptions of them are provided below.

### **Scripts reference**

#### **Class: Audio Playback Script**

##### **Description**

This class can be utilised to handle any type of one-shot audio, in a fire and forget fashion. One-shots created through this class can be 3D or 2D and also support any number of parameter types, with support for all fmod parameter types. All event instantiation is abstracted, and all memory management is handled by the script.

##### **Public static methods:**

#### **-PlayOneShot(EventReference fmodEvent, Transform transformToAttachTo)**

**Description:** play a 2D or 3D audio event with no parameters.

**Return type:** void

##### **Arguments:**

fmodEvent: the event reference of the fmod studio event you would like to trigger. (Required argument!)

transformToAttachTo: the transform of the gameobject you would like to attach the fmod event to to make 3D. (Non-required argument: you can pass 'null' if you want the event to be 2D)

Example call: 'AudioPlayback.PlayOneShot(shootingEvent, this.transform);'

## **PlayOneShotWithParameters<T>(EventReference fmodEvent, Transform transformToAttachTo, params (string name, T value)[] parameters)**

**Description:** play a 2D or 3D audio event with any number of parameters, supporting all fmod built in parameter types (**int**, **string** and **float**)

Return type: void

### **Arguments:**

fmodEvent: the event reference of the fmod studio event you would like to trigger. (Required argument!)

transformToAttachTo: the transform of the gameobject you would like to attach the fmod event to to make 3D. (non-required argument: you can pass 'null' if you want the event to be 2D).

parameters: name: the name of the fmod parameter you would like to adapt. Value: the value you want to set the fmod parameter to.

Example call with one parameter type: 'AudioPlayback.PlayOneShotWithParameters(foodstepSFX, this.transform, ("Surface", "Rock"))';

Example call with varying parameter types:

'AudioPlayback.PlayOneShotWithParameters<dynamic>(foodstepSFX, this.transform, ("Surface", "Rock"), ("Health", 10));' Note: when using parameters that vary in type you will have to add '<dynamic>' at the end of the function name, this is not required if all parameters are of the same data type.

## **Class: FmodParameters**

**Description:** used to adapt parameter values for any looping fmod events, both local and global parameter scopes supported.

### **Public static methods:**

SetParamByLabelName(EventInstance eventInstance, string labelParamName, string labelParamState)

Description: used to set the value of a local labelled parameter for an fmod event instance.

Return type: void

### **Arguments**

eventInstance: the instance of an fmod event you would like to adapt a local parameter for (Required argument)

labelParamName: the name of the labelled parameter you would like to adapt (Required argument).

labelParamState: the value of the labelled parameter you would like to set.

Example call: 'FmodParams.SetParamByLabelName(npcAudioInstance, "State", "Engage");'

### **SetParamByName<T>(EventInstance eventInstance, [string](#) paramName, T paramValue)**

Description: used to adapt the parameter of a looping audio event. Can take handle and of the fmod builtin parameter types (float, int, string)

Arguments:

eventInstance: the instance of an fmod event you would like to adapt a local parameter for (Required argument)

paramName: the name of the fmod event instance you would like to adapt a variable for

paramValue: the value to which you would like to set an fmod event instance parameter value to.

Example call: 'FmodParams.SetParamByName(pistolEvent, "Ammo", 10);'

### **GetParamByName(EventInstance eventInstance, [string](#) paramName)**

Description: used to get a local parameter value for a continuous or discrete type.

Return type: float

Arguments:

eventInstance: the event instance of the parameter value you would like to retrieve.

paramName: the name of the local parameter you would like to adapt.

Example call: 'fmodParams.GetParamByName(playerHealth, 5);'

SetGlobalParamByLabelName([string](#) labelParamName, [string](#) labelParamState)

Description: used to set a global fmod labelled parameter.

Return type: void

Arguments:

labelParamName: the name of the labelled parameter you would like to adapt.

labelParamState: the value or state you would like to set a global labelled parameter to.

Example call: 'fmodParams.SetGlobalParamByLabelName("PlayerState", "Injured");'

SetGlobalParamByName<T>([string](#) globalParamName, T globalParamValue)

Description: allows you to set a global parameter of any of the supported fmod data types.

Return type: void

Arguments:

globalParamName: the name of the global parameter you would like to adapt.

globalParamValue: the value you would like to set the global parameter value to.

Example call: `fmodParams.SetGlobalParamByName("Day-Cycle", 2);`

### Tutorial: Utilising the SDK to create adaptable footsteps and shell casing impacts

While the utility scripts have a reference, as shown above, which highlight all aspects of the provided utility scripts. A practical example is provided with the package. This package utilises a free package from the asset store called EasyFPS, available here: <https://assetstore.unity.com/packages/3d/characters/humanoids/sci-fi/easy-fps-73776>. The package is a very basic FPS example that allows you to walk around and shoot a gun. The package utilised non modulated or automated Unity audio which tonal aspects were static, in that there is nothing adapting the sounds. This example scene swapped the Unity audio for FMOD, utilising the scripts provided with the parameter SDK to highlight how utilising these scripts can provide the easy adaptation of any parameter type, globally.

The test scene highlights the use of the scripts for adapting footstep sounds, as well as shell casing impact sounds, utilising adaptive audio design.

#### Create a parameter inside fmod

For this tutorial a local parameter type called surface is utilised, this is a labelled parameter and will be utilised to detect what surface type the player is on, to adapt footstep and shell impact audio events. Create an fmod event for footsteps which has multiple multi-instruments on different tracks, each multi-instrument on a track will represent a surface type. Create a labelled local parameter type, in this case the parameter name is called 'Surface' and is used to trigger on of the footsteps multi-instruments, respective of its surface. The parameter used in the example was created as shown in Figure 1.

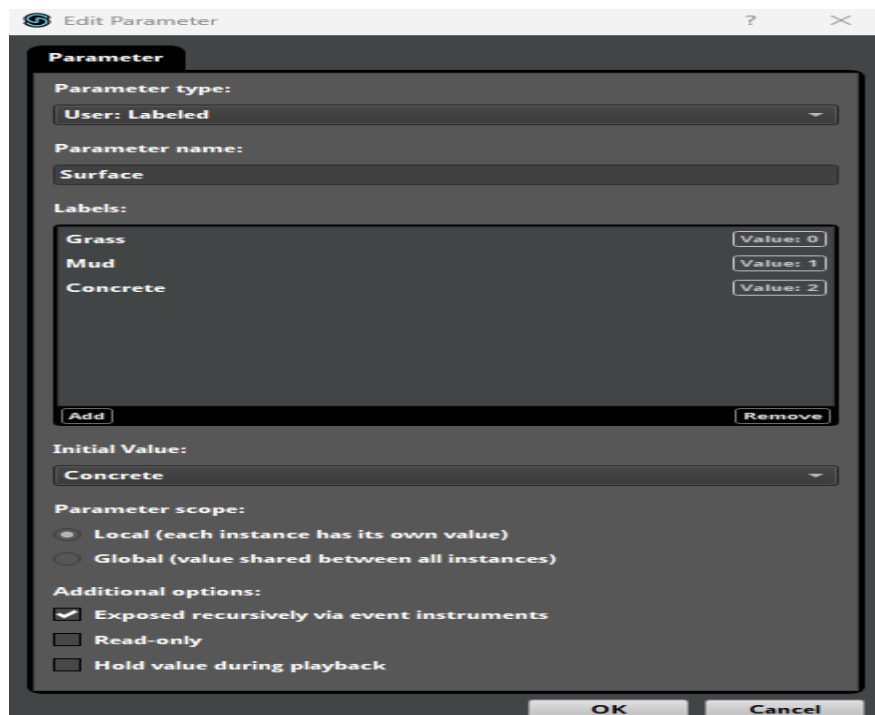


Figure 1 Parameter created for the example scene provided with the package

With this parameter in place, you can now provide the parameter as a trigger condition for each of the multi-instruments, this is shown in Figure 2. Set the trigger condition to the labelled param respective of the surface, this will mean that only the surface type the player is situated on will trigger and not all of the footstep's sound for each multi-instrument. It should be noted, that while

this is not a sound design tutorial, it is recommended to use asynchronous trigger mode for footsteps so they can trigger over each other, without requiring a previous instance to finish first.

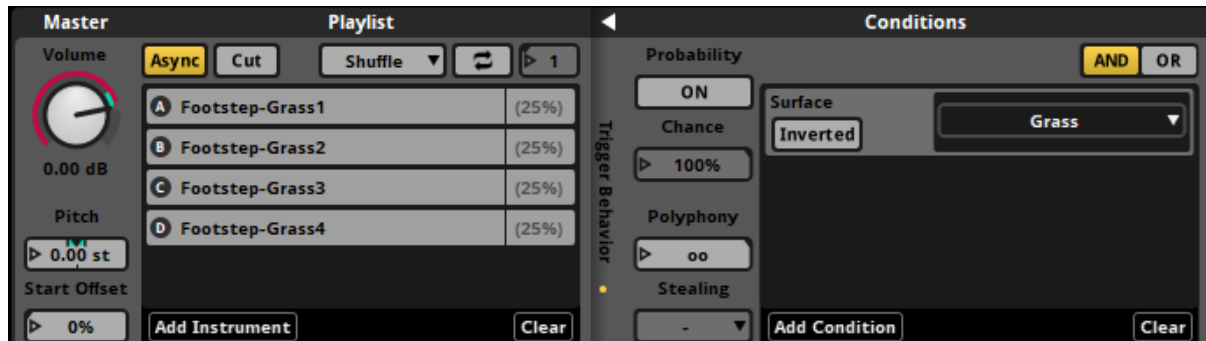


Figure 2 Using the created parameter to dictate when a multi-instrument should be triggered

While this is only being used to adapt which type of footstep sound to play, the tutorial shows a more adapted use of the parameter with the shell casing audio event provided within the banks provided with the package. This use case utilises an FMOD low-pass filter to adapt the cut-off based upon which surface the player is on. If the player is on a hard surface, the shells will clatter on the floor, but on a soft surface these impacts will be dulled or completely in-audible. This implementation utilises the same surface parameter, but instead of using it to determine whether the casing impact should be triggered it is used to adapt the cut off of a low-pass filter attached to the audio track of the shell casing as highlighted in Figure 3. This is not done on the master of the event as we only want to adapt the shell casings, not the gunshot, which are situated within the same FMOD event. Therefore, create two tracks for the event, one for the gunshot and one for the casing, again make them asynchronous on the instrument and attach a low pass filter to the post fader side of the effects strip (right of the volume effect). Right click on the cut-off of the low pass filter and apply automation, use the 'Surface' parameter to feed the automation and set the different surface cut-off values as desired.

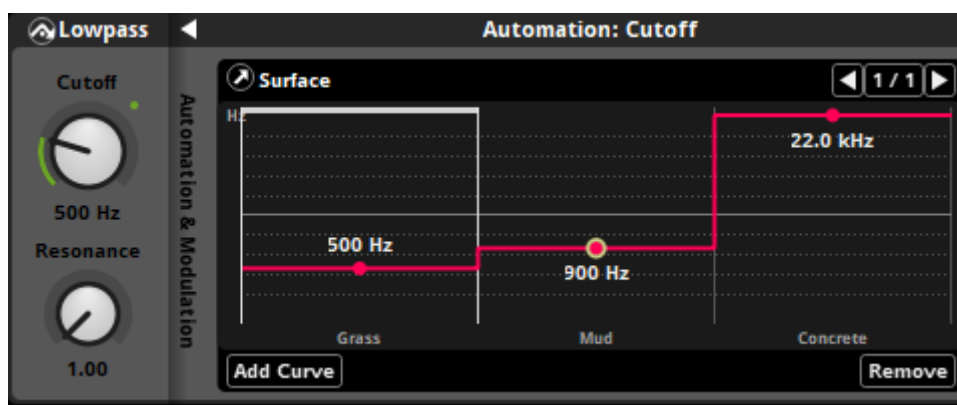
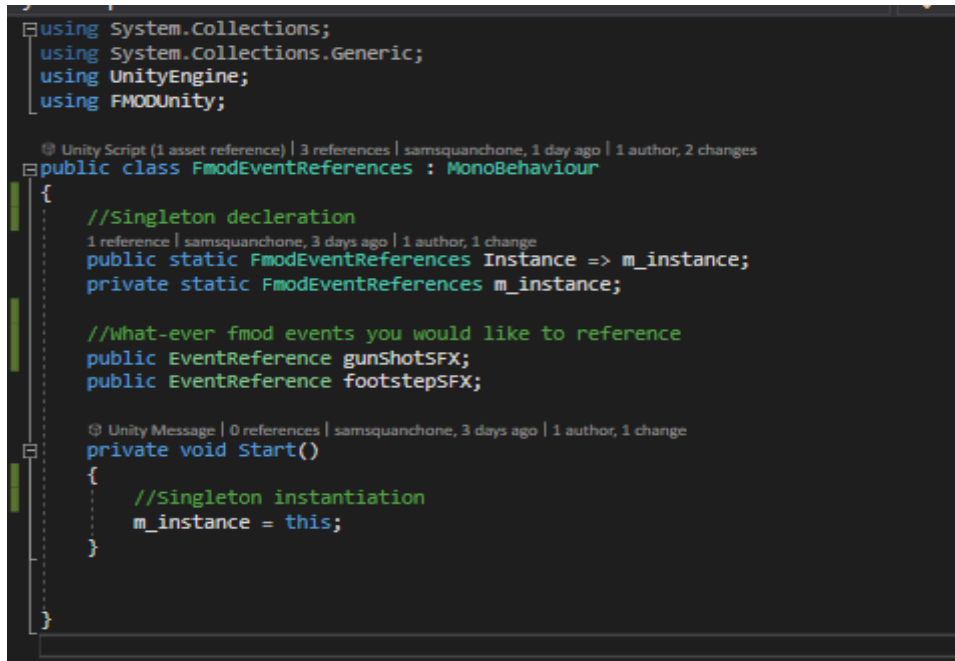


Figure 3 cut-off automation for the shell casing, based off surface parameter value. Part of the audio provided with the test scenes of the package

## Triggering a one-shot with parameters inside of Unity

Now with some insight for how the audio event parameters were created for this tutorial, as well as how the parameter was used to achieved different things: filtering and trigger conditions, you can now utilise the parameter SDK inside of Unity to adapt your audio events! It should be noted that you will still need to create a reference of an audio event you would like to trigger. There are many methodologies to achieving this, such as creating an fmod event reference in the script where you are triggering audio to provide to the function call for triggering one shots. However, for the example scene provided as part of this approach a singleton method was utilised. A game object was created called fmod references, this had a script attached to it called FMOD event references, this script is a singleton and allows the easy creation of event references while being able to be accessible by any script being used within the same scene. The implementation of that simple script is highlighted in Figure 3. Note, this method does not have to be utilised to work with the system and any method of event referencing can be used.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using FMODUnity;

public class FmodEventReferences : MonoBehaviour
{
    //Singleton declaration
    public static FmodEventReferences Instance => m_instance;
    private static FmodEventReferences m_instance;

    //What-ever fmod events you would like to reference
    public EventReference gunShotSFX;
    public EventReference footstepSFX;

    private void Start()
    {
        //Singleton instantiation
        m_instance = this;
    }
}
```

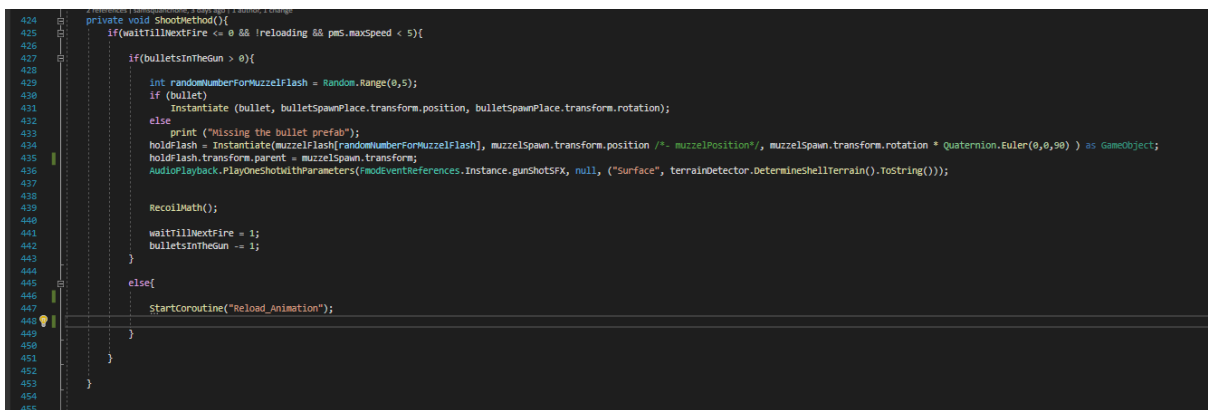
Figure 3 The method used for referencing audio events for the test scene provided with this package

With a method in place for accessing fmod event references, the SDK can now be utilised to trigger one-shot fire and forget audio with any quantity of parameters, of any of the fmod supported parameter types. For this tutorial it is utilised to adapt footstep and bullet impact, depending on what surface the player is on. The parameter value is calculated by firing a raycast below the player to determine what surface the player is on and then adapt the audio depending on the value. While the methodology is outside of the scope of the tutorial, it is provided in the test scene with the package, however that will not be discussed in detail as this tutorial is about using the parameters SDK provided and not creating system to generate parameter values. As this test scene replaced the Unity audio of the scene provided with the EasyFPS package, calls to the parameter SDK are executed



within scripts provided by the FPS package, to illustrate how quickly adaptive audio can be implemented with this SDK.

For the shell casing the call to the parameter SDK for handling one-shot with parameters is handled in the 'GunScript' provided by the EasyFPS package. Previously, within this script Unity audio was being utilised to trigger gunshot sounds, this was replaced with the fmod package utilising the fmod parameters SDK provided with this tool. On the playerObject there is a script called 'DetermineTerrain' this is just used to fire a raycast at a given time to determine what layer the player is on and set an enum value for the respective layer. It should be noted, this requires setting up layer masks to determine your layer types, for ease of use the enum values for surface type are the same as what the surfaces are called in the fmod surface parameter for use in this tutorial. This means that the enum string values can be used to pass to the one-shot with parameters function in the AudioPlayback script as part of the SDK, this is highlighted in Figure 4. on line 436. The singleton instance is also used to get the reference to the fmod event, without needing to reference it within this gun script.



```
424 private void ShootMethod(){
425     if(waitTillNextFire <= 0 && (reloading && pms.maxSpeed < 0){
426     }
427     if(bulletsInTheGun > 0){
428         int randomNumberForMuzzleFlash = Random.Range(0,5);
429         if (bullet)
430             Instantiate (bullet, bulletSpawnPlace.transform.position, bulletSpawnPlace.transform.rotation);
431         else
432             print ("Missing the bullet prefab");
433         holdFlash = Instantiate(muzzleFlash[randomNumberForMuzzleFlash], muzzleSpawn.transform.position /*- muzzlePosition*/, muzzleSpawn.transform.rotation * Quaternion.Euler(0,0,90) ) as GameObject;
434         holdFlash.transform.parent = muzzleSpawn.transform;
435         AudioPlayback.PlayOneShotWithParameters(FmodEventReferences.Instance.gunShotSFX, null, ("Surface", terrainDetector.DetermineShellTerrain().ToString()));
436
437         RecoilMath();
438
439         waitTillNextFire = 1;
440         bulletsInTheGun -= 1;
441     }
442     else{
443         StartCoroutine("Reload_Animation");
444     }
445 }
446
447
448
449
450
451
452
453
454
455 }
```

Figure 4 Trigger bullet casing impacts with parameters, as part of the tutorial scene provided with the package

A similar technique is utilised for triggering footsteps with parameters, the terrain is detected and then the one-shot is triggered with a parameter type. However, this is handled in a script called 'FootstepController' and allows the user to define a rigid body velocity threshold that must be reached to trigger a footstep, as well as the time delay between each trigger. While this footstep controller script is not a part of the main package, but a script to support the tutorial, it highlights how this SDK can be utilised to handle one-shots with parameter control. This footstep controller is not a main feature of the package as there are many ways player movement can be implemented and movement control is not a focus of this package, audio is. Therefore, this footstep controller serves as an example of how a footstep system could be created utilising the parameter SDK provided with the package.

## Occlusion and obstruction system

### Introduction

The second tool provided as part of this audio package is the audio occlusion and obstruction system. As it stands this system is currently completely inspector based. However, for further work there are plans to increase the systems functionality and expose some of this through an SDK, similar to that provided for the parameters control system. It should be noted that this occlusion system is designed for three-dimensional audio and the FMOD event you want this system to work with, must used the FMOD built in spatializer!

### Getting started

To utilise the occlusion system, attach the 'SpatialAudio' script to the event you would like to use with the occlusion system, this will utilise the objects transform to be the origin of the source. This system is completely inspector based and does not require any further programming to be utilised. An overview of the script from the inspector view is highlighted in Figure 5.

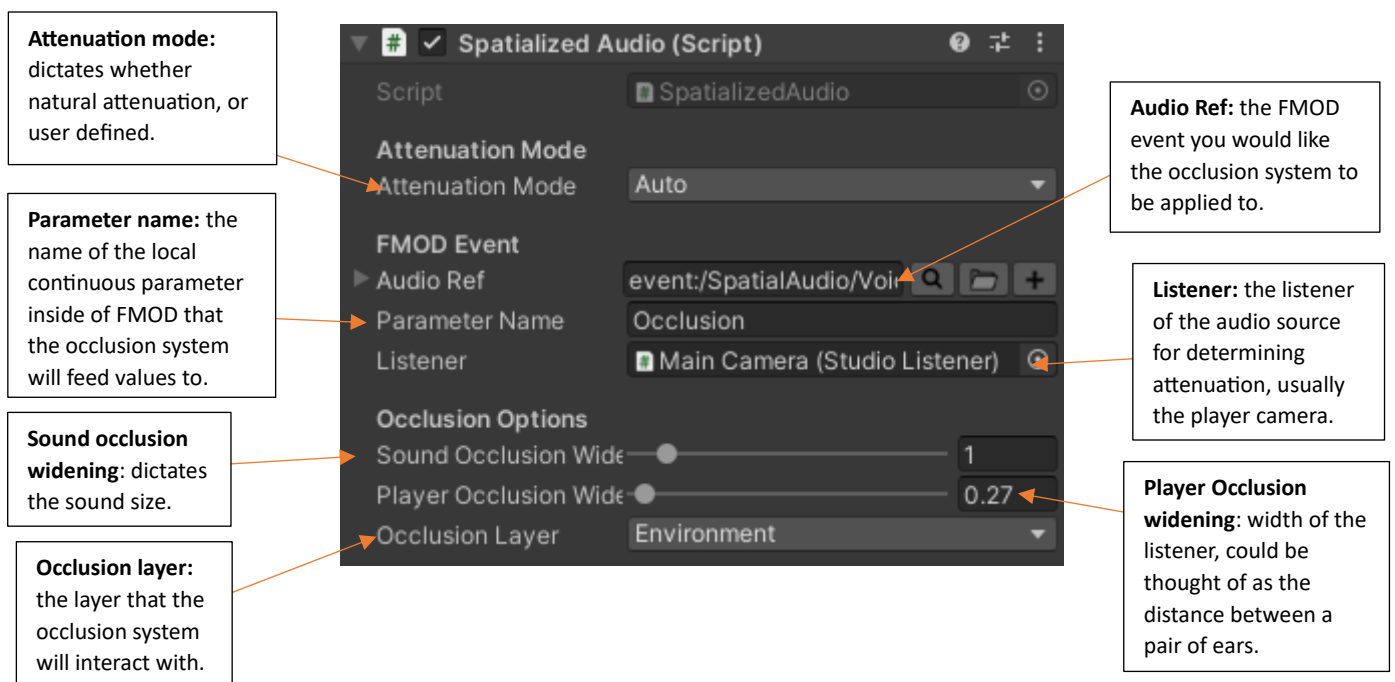


Figure 5 Inspector view of the script provided with this system, in auto attenuation mode

## Occlusion system: breakdown

The occlusion system has two primary operation modes, dictated by the attenuation mode and can be of an automatic or user defined type. The only difference between the two types is with user defined mode, a new variable will appear in the inspector called 'override max distance value' this is used to override the automatic attenuation and give you greater control over the distance at which the occlusion system functions.

**Attenuation mode:** a drop-down selection that allows you to select which attenuation mode the system utilises. With an automatic type, no further considerations are needed for distance attenuation. However, with a user type the user must specify the maximum distance for attenuation, as set by the sound designer in fmod on the events spatializer.

**Audio ref:** the fmod audio event you would like to apply occlusion to, note the system supports both single trigger and looping audio. However, the event must be 3D and have a fmod spatializer attached to it.

**Parameter name:** the name of the continuous parameter inside of fmod that the occlusion system will feed data to. An example will be provided on how this was set up for the occlusion test scene, provided with the package.

**Listener:** the listener, this is utilised to calculate distance attenuation. In most use cases the listener will be the player camera.

**Sound occlusion widening:** this parameter is used to dictate how large the sound size is, or the width that the line casts are fired in. A smaller value for this parameter will create a narrower cast of sound, more likely to be fully occluded, while a higher value for this parameter means a sound is more likely to have some parts which travel around, or over geometry in between the source and listener.

**Player occlusion widening:** this parameter is used to dictate the listeners width and can be used to support player models of different sizes.

**Occlusion layer:** the layer you want the occlusion system to interact with. For geometry in your game scene you would like the occlusion system to detect, you must create a layer and add that as the geometry objects layer, then in the occlusion layer parameter select that layer.

**Override Max Distance value (user attenuation mode only):** this parameter will only appear in the inspector if you have selected the user attenuation mode. This is used when the sound designer in fmod is not utilising automatic attenuation, they must define the min and max attenuation value for the audio event, use the maximum attenuation value as this parameter value. **The inspector in this user attenuation mode can be seen in Figure 7.**

## Demonstration: Utilising the occlusion system to emulate different sound sizes

In the test scene provided with the package that utilises the occlusion system, the aim is to highlight how the system can be used not only to emulate different sound sizes, to give you greater acoustical

control over events, but also how the system can be used in conjunction with other acoustical effects, such as reverberation. As such, these points will be the main focus points for this tutorial.

Within the test scene are three audio sources, two of them are the loudspeakers and are representative of an emitter generating a sound of a large size. The third audio emitter is within the room at the back of the scene and is ambient chatter audio, aimed to emulate sound of a small size. The listener for this scene is the player, represented as the black capsule. The scene set up is highlighted in Figure 6.

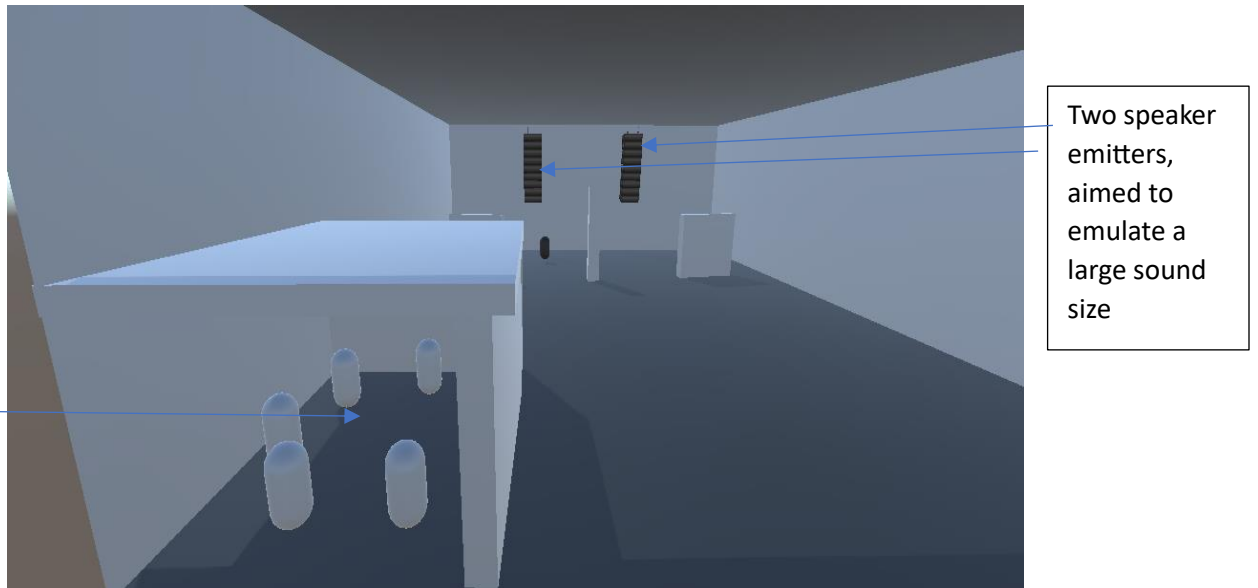


Figure 6 Test scene for the occlusion system, provided with the package

### Setting up the large sounds (Speaker emitters)

Within the scene there is a game object named 'Speaker Emitters', these contain two game objects for the loud speakers Line Array and Line Array(1), each of these objects have a child called emitter. On this emitter script there will be a script called 'Spatialized Audio', this script is responsible for setting up occlusion for the emitter. For the two loudspeaker emitters they will have the same settings for their occlusion scripts. They utilise the user attenuation mode, as the automatic attenuation mode attenuates the audio from the emitter too quickly and does not support a sound size typical of a loudspeaker. Therefore, the user attenuation mode was utilized and the fmod event for the music that the speaker emitters are playing were overridden and set to a max attenuation distance of 85, as seen in Figure 7.



Figure 7 Spatializer for the music audio event, played by the speaker emitters

For the fmod reference, if using the test scene provided by this package, you can utilise the sound banks provided within this scene for the audio, or add you own if you don't like dance music. You can hit the search icon signified by the magnifying glass on the variable to manually search sound banks for the audio event, as shown in Figure 8. The audio event utilised for this demonstration is the 'Music-Occlusion' event located in the 'SpatialAudio' folder. For the parameter name, if utilising the banks provided as part of the tutorial, type in 'Occlusion' for this value, which is the parameter inside fmod utilised for this demonstration, further information on setting up this parameter will be provided later on in this tutorial/demonstration. For the listener place the camera nested within the player object in the game scene. For the sound occlusion width as the speaker emitters are trying to emulate the a large sound size, set this value to a high value to give a large width to the sound, for this demonstration it is set to 5.38, as shown in Figure 9. While it is encouraged to tweak with the different occlusion widening settings to gain a grasp on how they adapt the system, a generic set up is provided with the scene for demonstration, as shown in Figure 9.

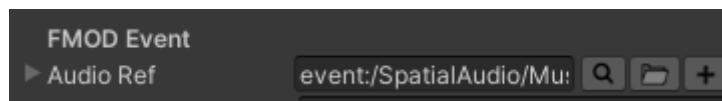


Figure 8 Audio reference example

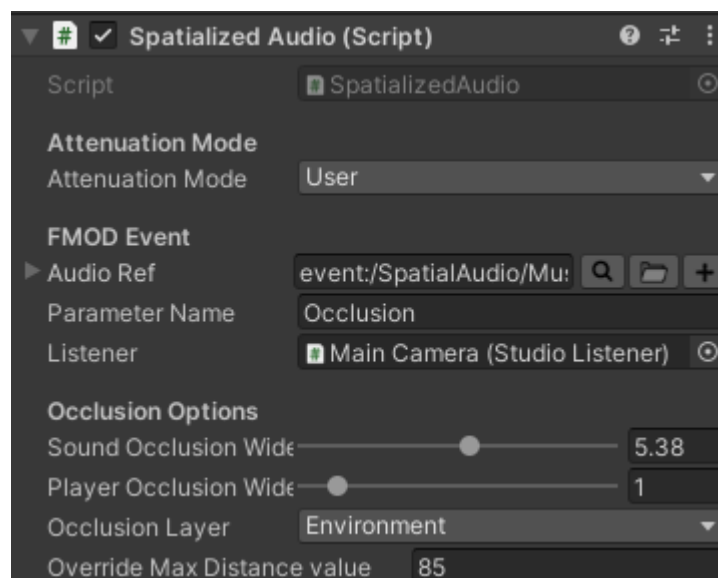


Figure 9 Occlusion system settings for both the loud speaker emitters, which emulate a sound of a large size

### Setting up the small sized sound (Ambient chatter)

As well as the two speaker sound sources, the scene also contains a third source, which is the ambient chatter of the room located at the back of the scene. This is aimed to highlight two different use cases for the system, aimed at re-creating that classic effect often found in nightclubs where you are in a separate room from the music but can still hear the music, predominantly the low frequencies. However, this would not be the same scenario for ambient chatter, which would mostly be in-audible if moving into a separate room from that source. Therefore, the test scene provided with the package for the occlusion system aims to highlight use of the system to re-create this phenomenon.

The emitter for this sound source can be found nested in the game object named 'SubRoom' on the object called 'Emitter', find the attached script called 'SpatialAudio'. Unlike the loudspeaker emitters, this system utilises the automatic attenuation type. This attenuation mode is utilised as the sound is trying to emulate a much smaller sound size than the loudspeaker emits. Therefore, the smaller maximum attenuation value provided by the automatic fmod spatialization mode is perfectly acceptable for this use case. For using the ambient chatter audio event, provided in the sound banks provided with this package select the 'Voices-Occlusion' event that is located within the 'SpatialAudio' folder. As done for the speaker emitters set the parameter name to 'Occlusion' and set the drag the MainCamera of the player into the listener parameter. Unlike the speaker emitters this sound will have a lower value for the sound occlusion widening parameter, to emulate a narrower propagation of sound. It is encouraged to tweak the widening settings to gain a grasp on how this adapts the occlusion system, however for demonstration purposes the settings provided in the test scene are shown in Figure 10.

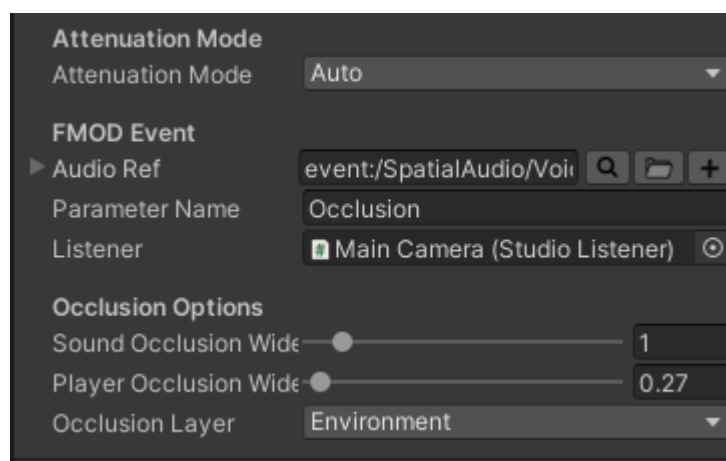


Figure 10 Set up for the occlusion system used on the ambient chatter sound source within the test scene

## Setting up the occlusion-parameter inside fmod

While the calculations for the occlusion parameter are automatically handled and calculated by the system provided with the package, a continuous local parameter need to be created within fmod and have a low-pass filter attached to the event to have the system be able to automate the filters cut-off depending on how much of the sound is occluded by geometry in the scene. The parameter can be named anything, if that name is reflected inside of Unity within the parameter name variable of the inspector for the system. However, the parameter must be of a local continuous type with a range of 0-1. The parameter created for use within this demonstration can be seen in figure 11.

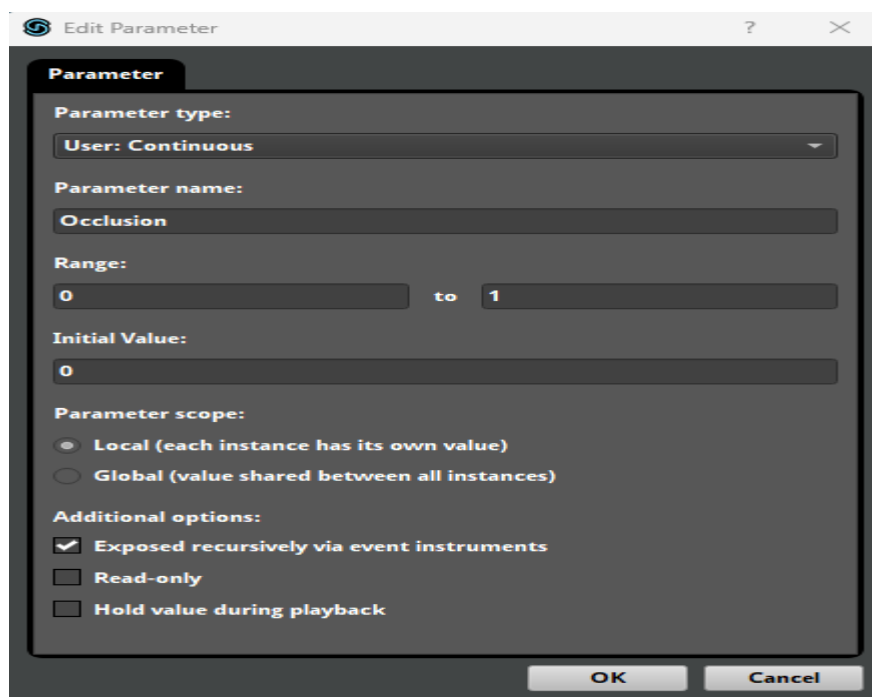
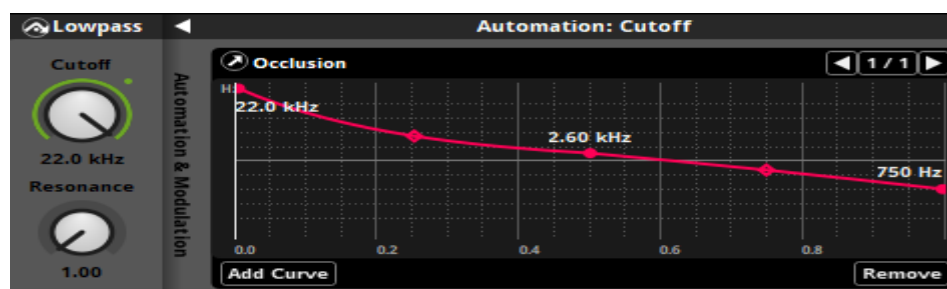


Figure 11 Occlusion parameter created in fmod for this test scene

With a parameter created that will interact with the occlusion system, effects can now be automated with values provided to the parameter. As lower frequencies are of greater wavelengths than higher frequencies, it makes sense to create an occlusion effect by utilising a low-pass filter and automating the cut-off based on the occlusion parameter value. The automation curve created between 0 (no obstruction) and 1 (occluded) can be defined by the sound designer in fmod, however for the demonstration provided within the scene the cut-off

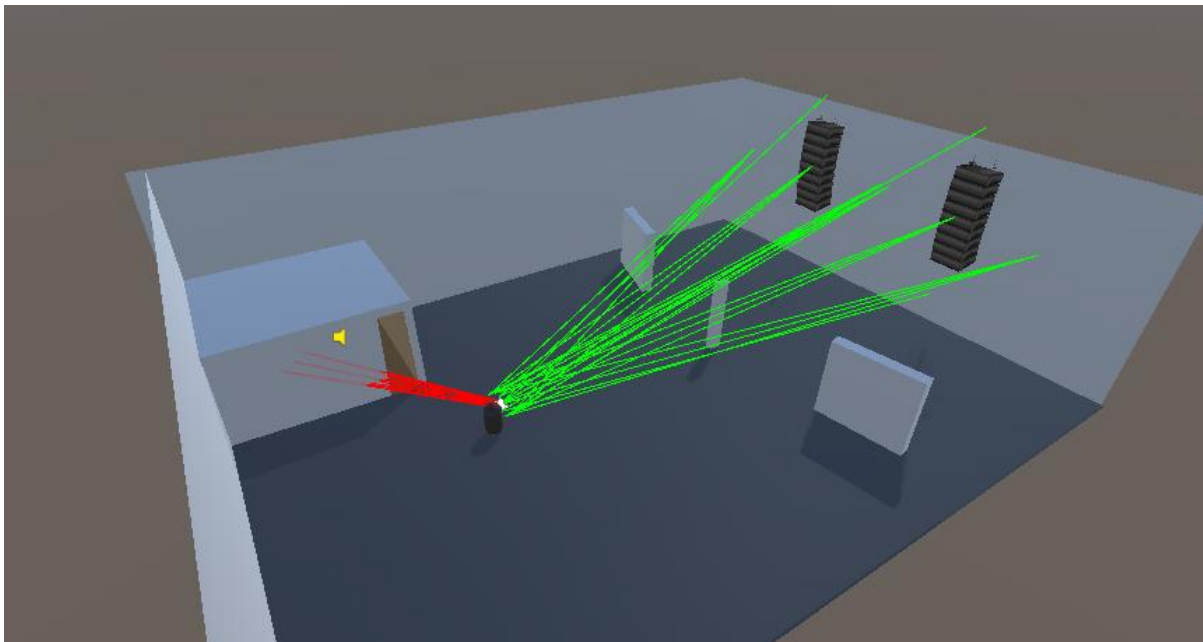


automation was set as shown in figure 12. The parameter can also be used to automate other effect parameters, such as volume.

*Figure 12 LPF occlusion automation curve created for the test scene. This curve is used for all emitter audio within the scene*

### **Result of set up**

As a result of the setup created for the demonstration scene, the result is two varying use cases of the system. The green lines in Figure 13 illustrate the propagation of sound from the speaker emitters, and being of a larger sound size and higher sound occlusion widening parameter value. While the red lines indicate sound coming from the ambient chatter emitter, having lower occlusion widening parameter value. The result is when you enter the separate room you can still hear the music but mostly the lower frequencies, as well as the ambient chatter. But when you leave the room the ambient chatter is very quickly occluded narrower propagation to the player.



*Figure 13 Result of the setup within the scene. Speakers being a larger sounds size and reaches the player in more locations than the ambient chatter, in the separate room*





## Dialogue system

### XML Application

The XML application allows easy authoring of XML files to work with the Unity dialogue system. Each line of dialogue can have as many conditions as required, however they are currently of an OR operator type. Full overview of all the fields and functionality within the XML application are provided in Figure 14. The authoring tool provides validation to ensure correct values are inputted.

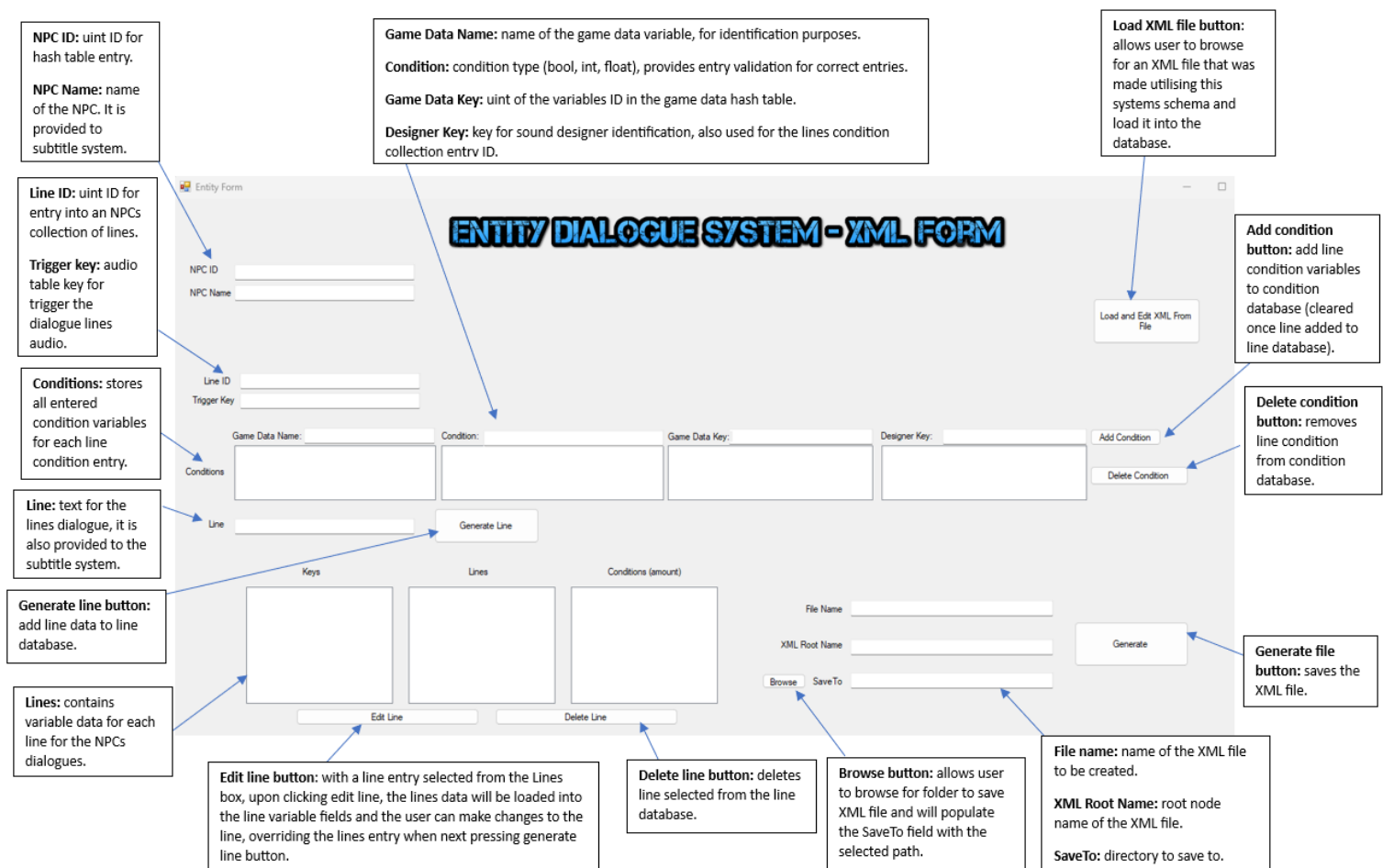


Figure 14 XML Application Functionality

However, if wanting to create your own Excel scheme the data structure for the XML can be described as so:

**Base Node(Attributes: uint 'ID', string 'Name')**

```
{
    Line Node(Attributes: uint 'id')
    {
        Key element (string 'Key')
        Line Text element (string 'LinText')
        Condition Node(Attributes: uint 'id')
        {
            Game Data Name element(string 'GameDataName')
            Game Data Key element(uint 'GameDataKey')
            Trigger Condition element (string 'TriggerCondition')
        }
    }
}
```

### **Dialogue Entity Script**

The dialogue entity script allows you to use a created XML file to trigger dialogue and check dialogue conditions. Currently the system provides these sequence types: random one-shot, sequential and player response. Furthermore, the system supports these trigger types: trigger-exit, trigger-enter, collision, radius and player response. Overview of this script is provided in Figure 15.

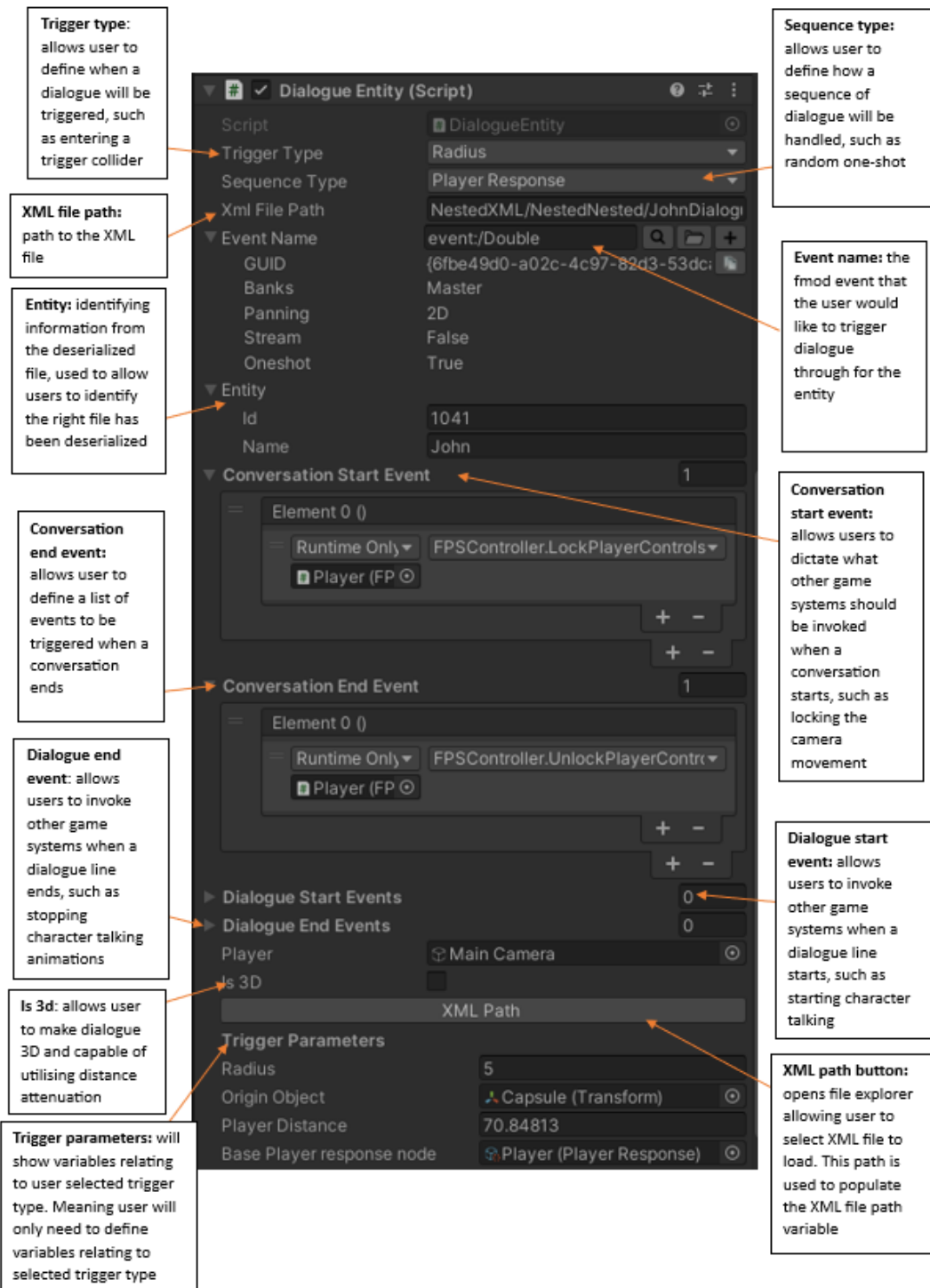


Figure 15 Dialogue entity script functionality

## Player response system

The player response system allows you to create new player response scriptable objects in your project assets. By creating a new player response object asset type. Full functionality is provided in Figure 16.

The screenshot shows the Unity Inspector for a **Player (Player Response)** scriptable object. The interface is divided into several sections with the following fields and callouts:

- Script:** PlayerResponse
- Player Name:** Alex
- Response Node Name:** GetQuest
- Npc ID:** 1041
- Player Responses:** 2
  - Hey, how is it going?**
    - Response Text:** Hey, how is it going?
    - Condition:** 0
    - Npc Line ID:** 1234
    - Transition Node:** None (Player Response)
    - Is Exit Node:** ☐
    - Events List:** 0
  - Know of anything I can do round here?**
    - Response Text:** Know of anything I can do round here?
    - Condition:** 0
    - Npc Line ID:** 4567
    - Transition Node:** PlayerNode2 (Player Response)
    - Is Exit Node:** ☐
    - Events List:** 0
- Fmod Event:** event:/Talking
- Transition To:** Transition (Player Response)
- Transition Condition:**
  - Game Data Condition Name:** QuestAccepted
  - Trigger Condition:** True
  - Condition To Parse:** True
  - Game Data Key:** 2

**Callouts and their descriptions:**

- Player name:** used to provide name of player to the subtitle system.
- NPC ID:** the dialogue hash table ID for the NPC the player is having a conversation with.
- Player response:** list of player response objects for the conversation (maximum of six player responses currently supported).
- NPC Line ID:** the id of the NPC dialogue line the player response will trigger.
- Events list:** user defined events that will be invoked after the NPC dialogue triggered from the player response.
- Transition to:** node to transition to when its condition is met.
- Response node name:** Used to help identify the nodes' purpose.
- Response text:** the text of the response, also provided to the subtitle system.
- Condition:** condition for the response to be triggerable (not visible but uses same data structure as the dialogue condition object).
- Transition node:** the node the response will transition to (if empty the conversation will return to this node after the NPC dialogue is sequenced).
- Is exit node:** used to dictate whether the response will exit the conversation after the NPC dialogue is triggered.
- FMOD event:** FMOD event used for NPC dialogue triggered from player response.
- Transition condition:** condition information for transitioning the node for the next conversation. The game data key is used to check the condition with the game data resolver.