🍑HolyAudio2D

A comprehensive and simple-to-implement 2D Audio Manager for the Unity Engine

Version: **0.1.0** (August 30, 2023)
Unity Editor Version: **2021.3.11f1** (Note: This is the version of the editor I used to make this tool. It will most likely work on later version although I have not tried)

Repository: 🍑HolyAudio2D
Author: holypeach
E-mail: frankberm3@gmail.com

Welcome to the documentation for this little project of mine, I hope it is of use to you in your endeavors. I made this Audio Manager to fit my own specifications and what I think I will need in my future projects. I will most likely keep updating it as I test it, add new features I might need, and improve on some design ideas. If you have any recommendations or find any bugs feel free to contact me or open an issue on the repository.

Table of Contents

Overview and Design

The manager consists of 2 major MonoBehaviour scripts. The first is the HolyAudioManager which is to be used with player, music, and UI sounds. The second script is the HolyLocalAudio which is to be used for environmental and enemy sounds. Apart from these 2 major scripts, there is 1 that is a MonoBehaviour and 5 other classes that are not MonoBehaviour.

Monobehaviour:
The one MonoBehaviour is HolyLocalTemp which is used in the PlayInPlace() method of the HolyLocalAudio script.

Non-Monobehaviour:
HolyMixerInfo holds a Mixer and Mixer UpdateMode, which is used by the Managers when setting up Mixers on the inspector. Similarly HolySound and HolySourceSounds hold the information for an audio clip and its AudioSource. HolyAudioData is the object that is stored as a binary to save volume levels. Finally, HolyAudioSaver is a static class that is in charge of saving and loading the data to and from the binary save file.

The HolyAudioManager is to be inserted as a component of an empty GameObject. The HolyLocalAudio is to be inserted in entities such as enemies and environmental objects that might be created or destroyed; these will handle their own sounds internally.
Sounds (HolySound) are objects that contain information for the AudioSource. When the game runs, an AudioSource component is added for each sound to the object holding the audio manager (both HolyAudioManager and HolyLocalAudio). A SourceSound (HolySourceSounds) is a reference to an existing AudioSource component in the object.
When you play a sound using the Play method, Play(“Theme”), the name of the clip that you pass is compared to the sounds stored in the Sounds and SourceSounds Dictionaries. Meaning that we look for the KeyValuePair that has the key, which is the string we pass. If the clip is found it is played.

Quick Reference

Terms

- **Sound**: Refers to a HolySound object that is a part of the Sounds list on the inspector of either the HolyAudioManager or the HolyLocalAudio.
- **SourceSound**: Refers to a HolySourceSound object that is a part of the SourceSoundslist on the inspector of either the HolyAudioManager or the HolyLocalAudio.
- **Raw Value of Volume**: Refers to the volume value from a Mixer or a MixerGroup expressed in decibels.
- **mixerGroupNameVolume**: Refers to the exposed parameter from a mixer that belongs to the volume of a mixer group.

Must Know

1. Enable Debug during development.
2. Follow the Mixer and MixerGroup setup. Here is a summary:
    a. Name the Volume parameter for the Master of all Mixers “MasterVolume” and name groups NameOfGroup+Volume (i.e. MusicVolume, SoundEffectsVolume).
    b. Make sure no two groups have the same name, even across multiple mixers.

3. After setting up mixers and mixer groups properly, check true the “Are Mixers Setup Properly” field. This will enable saving of audio settings.
4. Make sure that the Sound names, and SourceSound names are unique for the object. No 2 Sounds should have the same name and no Sound should have the same name as a SourceSound.
5. To get and set Mixers’ and MixerGroups’ Volumes, you can use their respective methods from the HolyAudioManager.
6. Before building the game for release, set the “Enable Debug” field in the HolyAudioManager and all HolyLocalAudio to false (uncheck it).

HolyAudioManager Properties

```
public static HolyAudioManager HolyAudioManagerInstance {get; private
set;}
This variable stores a static instance of the global HolyAudioManager. It
is static so the HolyLocalAudio script and all of the objects that use
this script, can access the global script. Keep in mind that this variable
is initialized in the Awake method. Meaning that this variable will be
null if another script references HolyAudioManagerInstance during Awake().
     For a more detailed explanation see "Debugging and Error Messages"
-> "Discussion on Possible Errors" -> "HolyAudioManagerInstance and
GlobalHolyAudioManager".
```

```
public string GameAudioVersion;
Stores a string passed from the inspector. See LoadSettings() method in
the HolyAudioManager Methods section.
```

HolyAudioManager Methods

Play

```
public void Play(string clipName)
Plays a sound that matches the passed clipName from either Sounds or
SourceSounds. This method will NOT STOP the clip if it was playing already
in the Sound’s source, it will play the sound again on top of the existing
sound.
```

```
public void PlayRepeat(string clipName, int iterations)
```

```
Plays a sound that matches the passed clipName from either Sounds or
SourceSounds. The sound will repeat as many times as the value of the
passed iterations argument.
```

```
public void PlayOnce(string clipName)
Plays a sound that matches the passed clipName from either Sounds or
SourceSounds. This method WILL STOP the clip if it was playing already in
the sound’s source, and play it again from the beginning.
```

Pause

```
public void Pause(string clipName)
Pauses a sound that matches the passed clipName from either Sounds or
SourceSounds.
```

```
public void PauseAllButMixerGroup(string mixerGroupName)
Pauses all Sounds and SourceSounds EXCEPT for the ones in the MixerGroup.
```

```
public void PauseAllFromMixerGroup(string mixerGroupName)
Pauses all Sounds and SourceSounds from a specific MixerGroup.
```

```
public void PauseAll()
Pauses all Sounds and SourceSounds.
```

Unpause

```
public void Unpause(string clipName)
Unpauses a sound that matches the passed clipName from either Sounds or
SourceSounds.
```

```
public void UnpauseAllButMixerGroup(string mixerGroupName)
Unpauses all the Sounds and SourceSounds EXCEPT for the ones in the
MixerGroup.
```

```
public void UnpauseAllFromMixerGroup(string mixerGroupName)
Pauses all the Sounds and SourceSoundsfrom a specific MixerGroup.
```

```
public void UnpauseAll()
Unpauses all Sounds and SourceSounds.
```

Stop

```
public void Stop(string clipName)
Stops a sound that matches the passed clipName from either Sounds or
SourceSounds.
```

```
public void StopAllButMixerGroup(string mixerGroupName)
Stops all the Sounds and SourceSounds EXCEPT for the ones in the
MixerGroup.
```

```
public void StopAllFromMixerGroup(string mixerGroupName)
Stops all the Sounds and SourceSounds from a specific MixerGroup.
```

```
public void StopAll()
Stops all Sounds and SourceSounds.
```

Getters for Mixers and Mixer Groups

```
public AudioMixer GetMixer(string mixerName)
Returns the AudioMixer that matches the passed mixerName.
```

```
public Dictionary<string, AudioMixer> GetAllMixers()
Returns a Dictionary that contains Audio Mixers. The key being the name of
the Mixer.
```

```
public AudioMixerGroup GetMixerGroup(string mixerGroupName)
Returns an AudioMixerGroup that matches the passed mixerGroupName.
```

```
public Dictionary<string, AudioMixerGroup> GetAllMixerGroups()
Returns a Dictionary that contains all Mixer Groups. The key being the
name of the MixerGroup
```

Saving and Loading Audio Settings

```
public void SaveSettings()
Saves audio settings (HolyAudioData object) to a binary file at
Application.persistentDataPath. For more information see HolyAudioSaver.cs
and the Saving section of this document.
```

```
public void LoadSettings()
```

```
Retrieves the audio settings from the saved binary file if it exists and
if the GameAudioVersion (HolyAudioManager) matches the AudioVersion
(HolyAudioData) from the binary save file. For more information, see
HolyAudioSaver.cs and the Saving section of this document.
```

Getters for Volume

```
public float GetMixerMasterVolume(string mixerName)
public float GetMixerMasterVolume(AudioMixer mixer)
public float GetMixerMasterVolume(int index)
Returns the raw value of the Mixer parameter "MasterVolume" from the
passed mixerName, mixer, or index of element from the Mixers array found
on the Inspector.
```

```
public float GetMixerGroupVolume(string mixerGroupName)
public float GetMixerGroupVolume(AudioMixerGroup mixerGroup)
public float GetMixerGroupVolume(int index)
Returns the raw value of the Mixer parameter "mixerGroupNameVolume" from
the passed mixerGroupName, mixerGroup, or index of element from the
MixerGroups array found on the Inspector.
```

Setters for Volume

```
public void SetMixerMasterVolume(string mixerName, float rawVolume)
public void SetMixerMasterVolume(string mixerName, int percentVolume)
public void SetMixerMasterVolume(AudioMixer mixer, float rawVolume)
public void SetMixerMasterVolume(AudioMixer mixer, int percentVolume)
public void SetMixerMasterVolume(int index, float rawVolume)
public void SetMixerMasterVolume(int index, int percentVolume)
Sets the Volume of a mixer's master. You must provide it a mixerName,
mixer, or index from the MixersInfo array. Must also provide a rawVolume
or percentVolume.
```

```
public void SetMixerGroupVolume(string mixerGroupName, float rawVolume)
public void SetMixerGroupVolume(string mixerGroupName, int percentVolume)
public void SetMixerGroupVolume(AudioMixerGroup mixerGroup, float
rawVolume)
public void SetMixerGroupVolume(AudioMixerGroup mixerGroup, int
percentVolume)
public void SetMixerGroupVolume(int index, int percentVolume)
public void SetMixerGroupVolume(int index, float rawVolume)
Sets the Volume of a mixer group. You must provide it a mixerGroupName,
mixerGroup, or index from the MixerGroups array. Must also provide a
rawVolume or percentVolume.
```

Volume Value Conversions

```
public float DeciblesToPercent(float rawVolume)
Converts the passed rawVolume into a percent volume value and returns said
percent value.
```

```
public float PercentToDecibles(float percentVolume)
Converts the passed percentVolume into a raw volume value and returns said
raw value.
```

Does X Exist

```
public bool DoesMixerExist(string mixerName)
Checks if a mixer by the passed mixerName exists in the Mixers Dictionary.
```

```
public bool DoesMixerGroupExist(string mixerGroupName)
Checks if a mixer group by the passed mixerName exists in the MixerGroups
Dictionary.
```

```
public bool DoesSoundExist(string soundName)
Checks if a Sound by soundName exists in the Sounds Dictionary.
```

```
public bool DoesSourceSoundExist(string sourceSoundName)
Checks if a SourceSound by sourceSoundName exists in the SourceSounds
Dictionary.
```

HolyLocalAudio Properties

```
public HolyAudioManager GlobalHolyAudioManager
This variable stores a reference to the HolyAudioManagerInstance property
of the HolyAudioManager. Keep in mind that this variable is initialized in
the Start() method. Meaning that this variable will be null if another
script references GlobalHolyAudioManager during Awake().
```

```
     For a more detailed explanation see "Debugging and Error Messages"
-> "Discussion on Possible Errors" -> "HolyAudioManagerInstance and
GlobalHolyAudioManager".
```

HolyLocalAudio Methods

Play

```
public void Play(string clipName)
Plays a sound that matches the passed clipName from either Sounds or
SourceSounds. This method will NOT STOP the clip if it was playing already
in the Sound’s source, it will play the sound again on top of the existing
sound.
```

```
public void PlayRepeat(string clipName, int iterations)
Plays a sound that matches the passed clipName from either Sounds or
SourceSounds. The sound will repeat as many times as the value of the
passed iterations argument.
```

```
public void PlayOnce(string clipName)
Plays a sound that matches the passed clipName from either Sounds or
SourceSounds. This method WILL STOP the clip if it was playing already in
the sound’s source, and play it again from the beginning.
```

```
public void PlayInPlace(string clipName, int iterations)
Allows you to Play a sound even if the object holding the HolyLocalAudio
component is destroyed by creating an empty object where the original
object was. You can also pass it the number of times the sound should
play.
```

Pause

```
public void Pause(string clipName)
Pauses a sound that matches the passed clipName from either Sounds or
SourceSounds.
```

```
public void PauseAllButMixerGroup(string mixerGroupName)
Pauses all Sounds and SourceSounds EXCEPT for the ones in the MixerGroup.
```

```
public void PauseAllFromMixerGroup(string mixerGroupName)
Pauses all Sounds and SourceSounds from a specific MixerGroup.
```

```
public void PauseAll()
Pauses all Sounds and SourceSounds.
```

Unpause

```
public void Unpause(string clipName)
Unpauses a sound that matches the passed clipName from either Sounds or
SourceSounds.
```

```
public void UnpauseAllButMixerGroup(string mixerGroupName)
Unpauses all the Sounds and SourceSounds EXCEPT for the ones in the
MixerGroup.
```

```
public void UnpauseAllFromMixerGroup(string mixerGroupName)
Pauses all the Sounds and SourceSoundsfrom a specific MixerGroup.
```

```
public void UnpauseAll()
Unpauses all Sounds and SourceSounds.
```

Stop

```
public void Stop(string clipName)
Stops a sound that matches the passed clipName from either Sounds or
SourceSounds.
```

```
public void StopAllButMixerGroup(string mixerGroupName)
Stops all the Sounds and SourceSounds EXCEPT for the ones in the
MixerGroup.
```

```
public void StopAllFromMixerGroup(string mixerGroupName)
Stops all the Sounds and SourceSounds from a specific MixerGroup.
```

```
public void StopAll()
Stops all Sounds and SourceSounds.
```

Getters for Mixers and Mixer Groups

```
public AudioMixer GetMixer(string mixerName)
Returns the AudioMixer that matches the passed mixerName.
```

```
public Dictionary<string, AudioMixer> GetAllMixers()
Returns a Dictionary that contains all Audio Mixers. The key being the
name of the Mixer.
```

```
public AudioMixerGroup GetMixerGroup(string mixerGroupName)
Returns an AudioMixerGroup that matches the passed mixerGroupName.
```

```
public Dictionary<string, AudioMixerGroup> GetAllMixerGroups()
Returns a Dictionary that contains all Mixer Groups. The key being the
name of the MixerGroup.
```

Volume Value Conversions

```
public float DeciblesToPercent(float rawVolume)
Converts the passed rawVolume into a percent volume value and returns said
percent value.
```

```
public float PercentToDecibles(float percentVolume)
Converts the passed percentVolume into a raw volume value and returns said
raw value.
```

Does X Exist

```
public bool DoesMixerExist(string mixerName)
Checks if a mixer by the passed mixerName exists in the Mixers Dictionary.
```

```
public bool DoesMixerGroupExist(string mixerGroupName)
Checks if a mixer group by the passed mixerName exists in the MixerGroups
Dictionary.
```

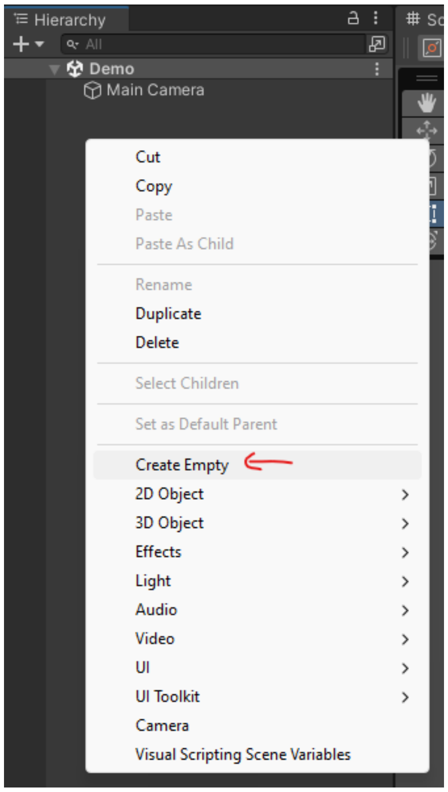
```
public bool DoesSoundExist(string soundName)
Checks if a Sound by soundName exists in the Sounds Dictionary.
```

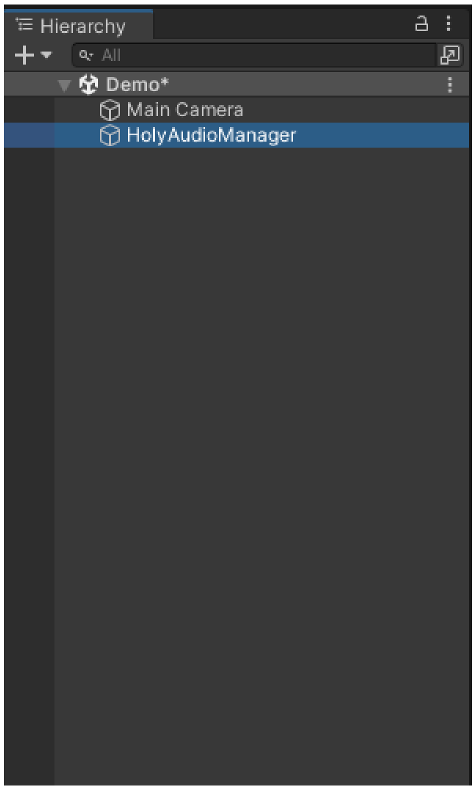
```
public bool DoesSourceSoundExist(string sourceSoundName)
Checks if a SourceSound by sourceSoundName exists in the SourceSounds
Dictionary.
```

Setup

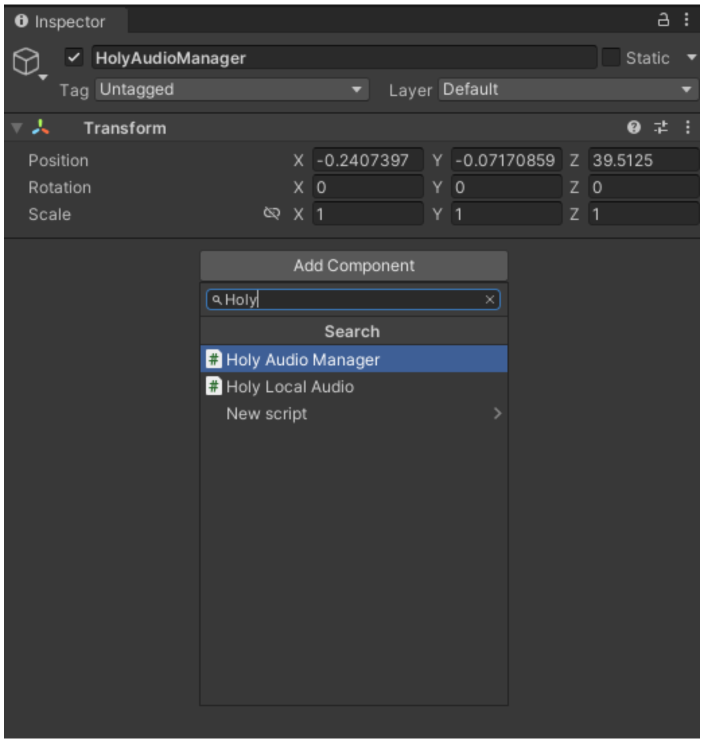
Scene/**HolyAudioManager** Setup

1. Create a new, empty Object and name it.





2. Attach the HolyAudioManager script to it.


Inspector

HolyAudioManager — Static

Tag Untagged — Layer Default

Transform

	X	Y	Z
Position	-0.2407397	-0.07170859	39.5125
Rotation	0	0	0
Scale	1	1	1

Add Component

Holy

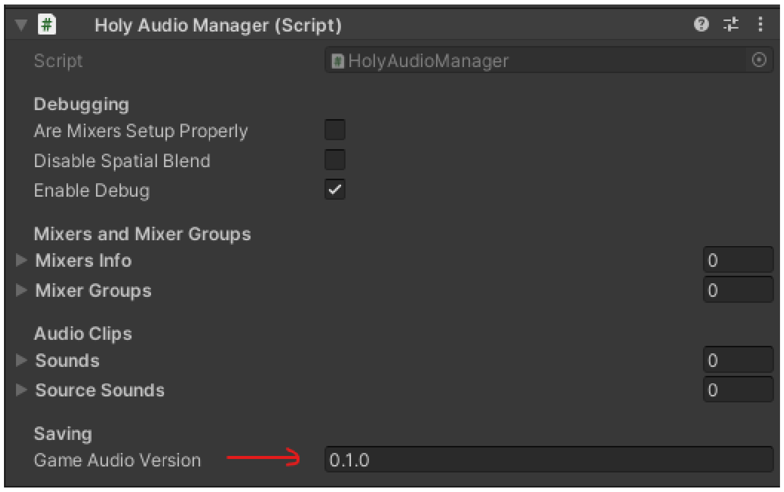
Search

Holy Audio Manager

Holy Local Audio

New script

3. Fill out the Game Audio Version field.


Holy Audio Manager (Script)

Script: HolyAudioManager

Debugging

Are Mixers Setup Properly: ☐

Disable Spatial Blend: ☐

Enable Debug: ✓

Mixers and Mixer Groups

Mixers Info: 0

Mixer Groups: 0

Audio Clips

Sounds: 0

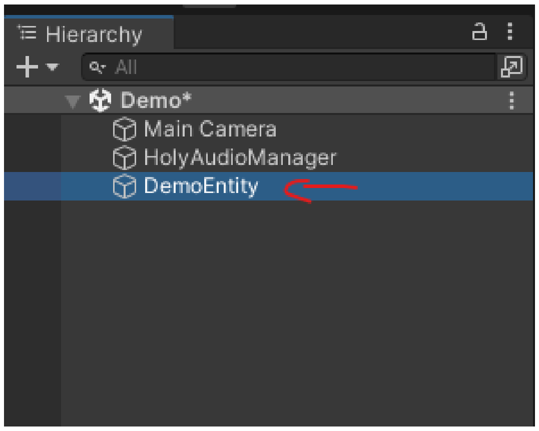
Source Sounds: 0

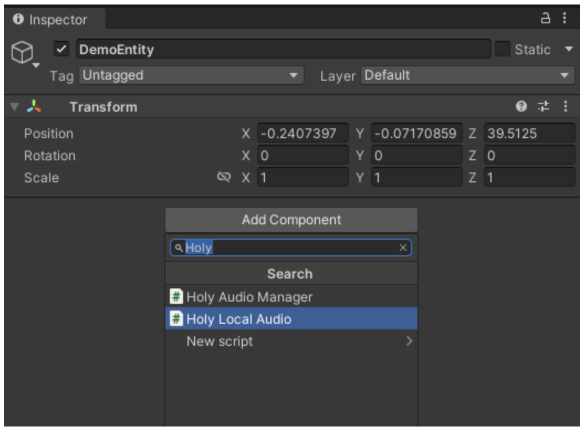
Saving

Game Audio Version: 0.1.0

Entity/**HolyLocalAudio** Setup

1. Create your entity, select it and add the HolyLocalAudio script to it.

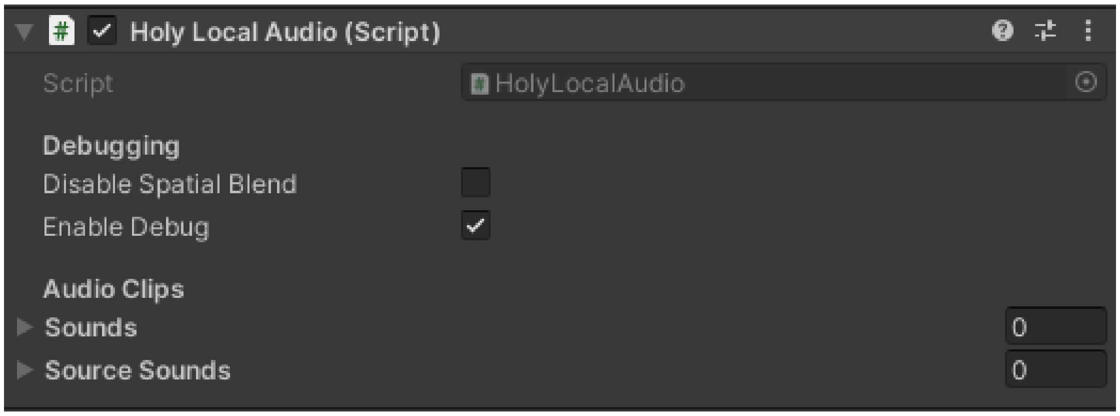




DemoEntity
Tag Untagged · Layer Default

	X	Y	Z
Position	-0.2407397	-0.07170859	39.5125
Rotation	0	0	0
Scale	1	1	1

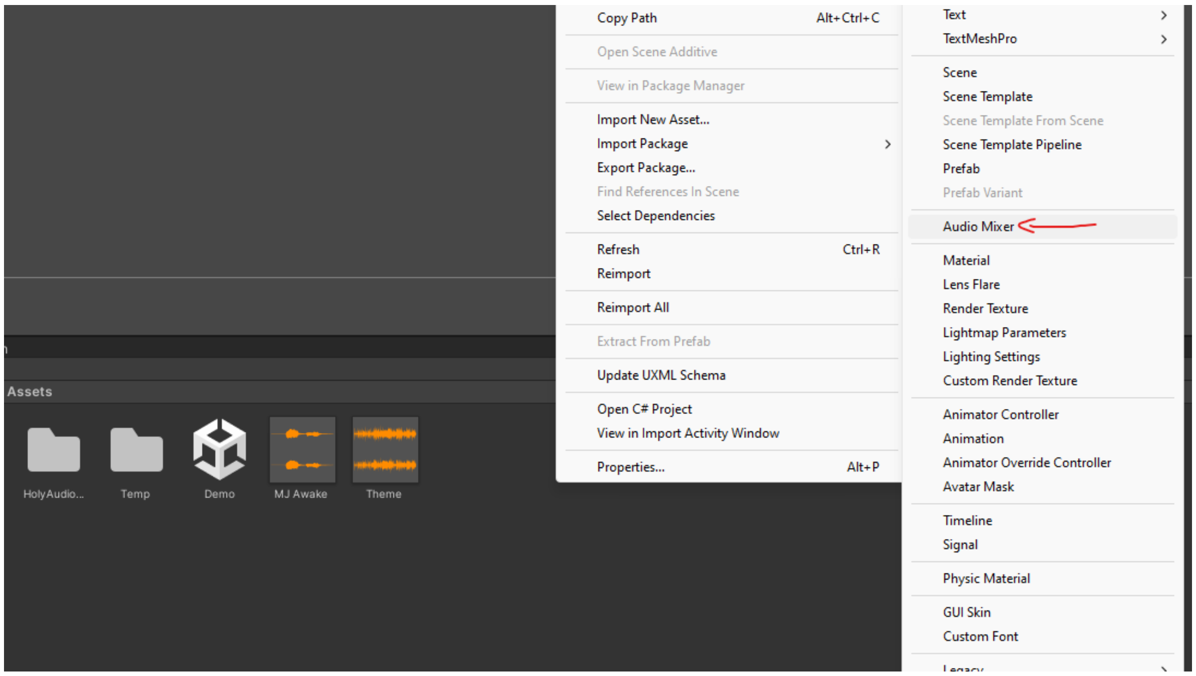
Search: Holy



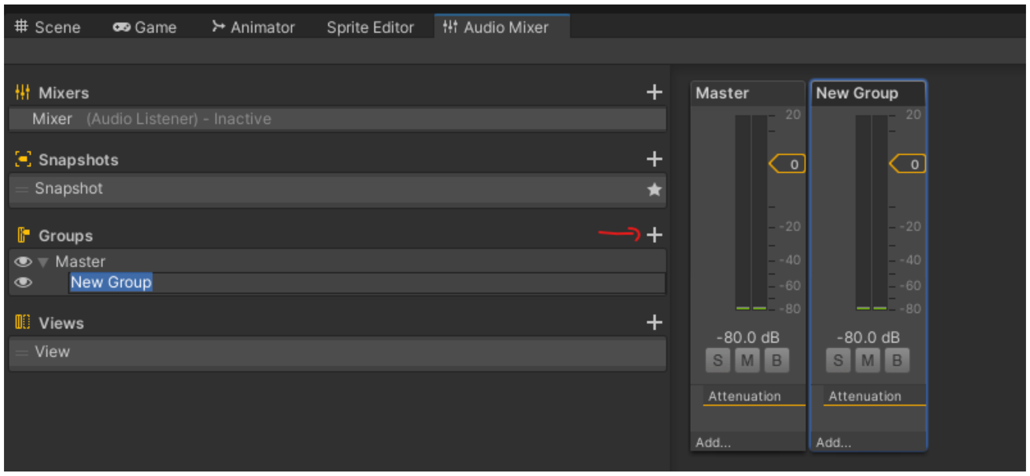
Script: HolyLocalAudio
Disable Spatial Blend: ☐
Enable Debug: ✓
Sounds: 0
Source Sounds: 0

AudioMixer and **AudioMixerGroups** Setup

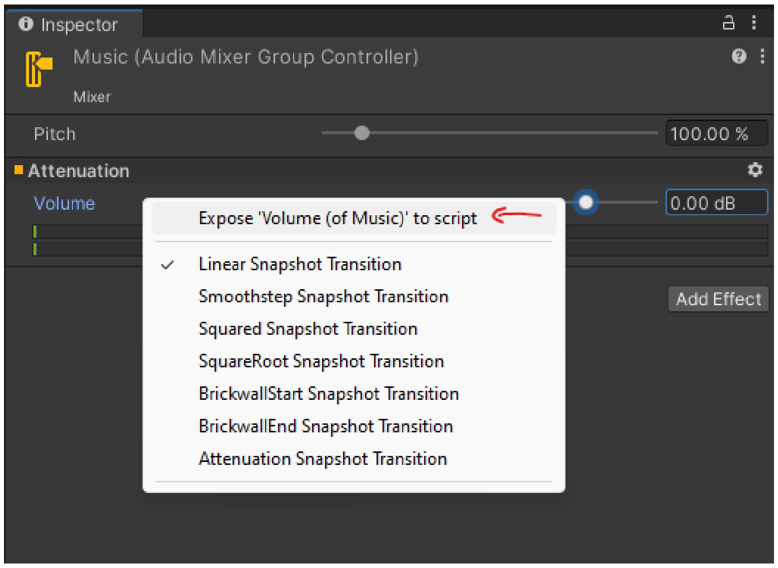
1. Right click on the assets panel, create a new AudioMixer, and name it.



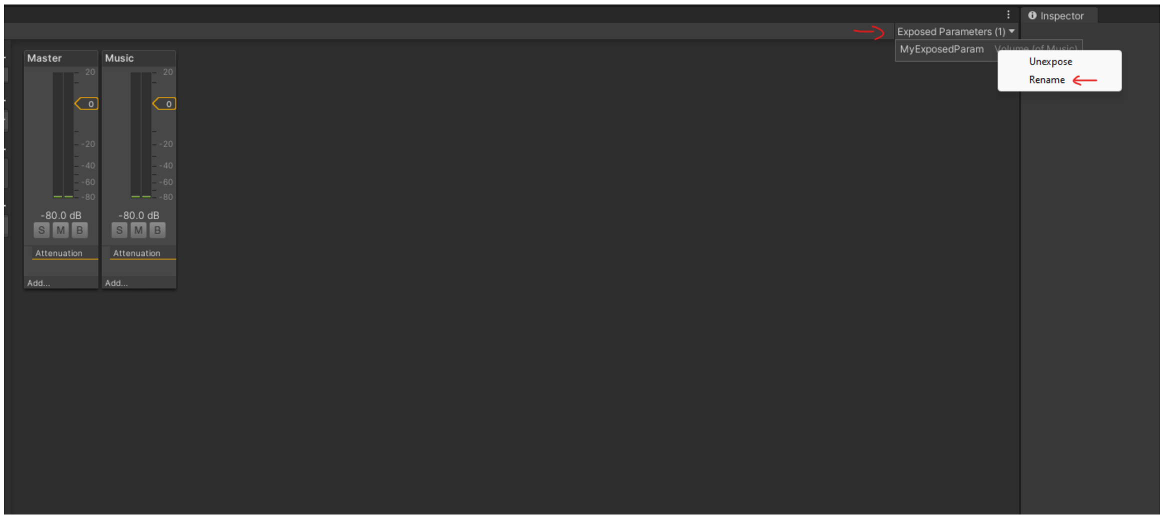
2. Double click on the Mixer to open the Audio Mixer panel.
3. Repeat steps 1 and 2 if you want to make other Mixers.
4. To create a MixerGroup click on the + icon in the Groups tab. Name your group.

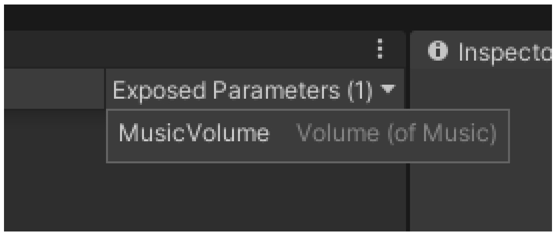


5. To expose the volume parameters click on the group (in this case Master or the new group which I called Music), then in the inspector right click on the Volume property and click on “Expose to script.”

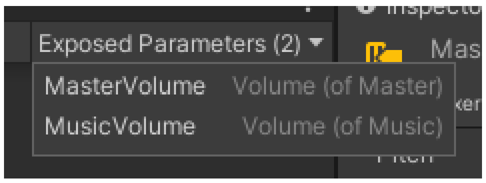

Inspector
Music (Audio Mixer Group Controller)
Mixer
Pitch
100.00 %
Attenuation
Volume
0.00 dB
Expose 'Volume (of Music)' to script
Linear Snapshot Transition
Smoothstep Snapshot Transition
Squared Snapshot Transition
SquareRoot Snapshot Transition
BrickwallStart Snapshot Transition
BrickwallEnd Snapshot Transition
Attenuation Snapshot Transition
Add Effect

6. Then on the top right of the Audio Mixer panel, there is a drop down menu called “Exposed Parameters.” Click on the Exposed Parameters menu and right click on the newly exposed parameter. **Rename it using the following format: NameOfGroupMixer + Volume. In this case it would be “MusicVolume”.** This is very important! You’ll get errors if you don’t do this correctly.

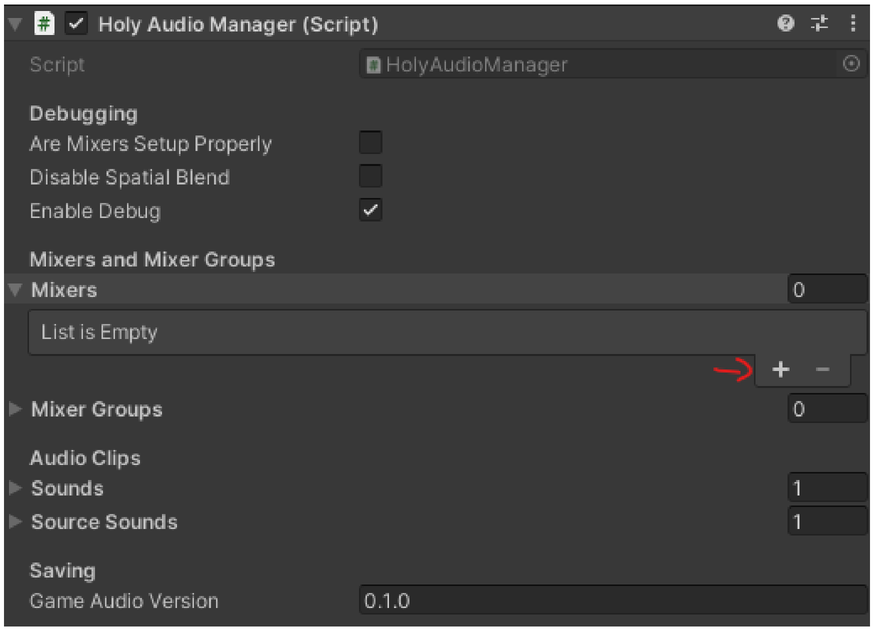




7. Do the same for the Volume parameter of the Master.


Exposed Parameters (2)
MasterVolume Volume (of Master)
MusicVolume Volume (of Music)
Mas

8. Do the same for the Volume parameters of other mixers. **Make sure to expose the Master of the other mixer groups and call them MasterVolume as well. Make sure to NOT have repeat names for MixerGroups.** For example don't do the following:
    a. Mixer1's Groups: Music, Player
    b. Mixer2's Groups: Music, Enemy
9. Add the Mixers and MixerGroups to the HolyAudioManager. No need to add the Master Groups, only add the child group mixers.



Holy Audio Manager (Script)

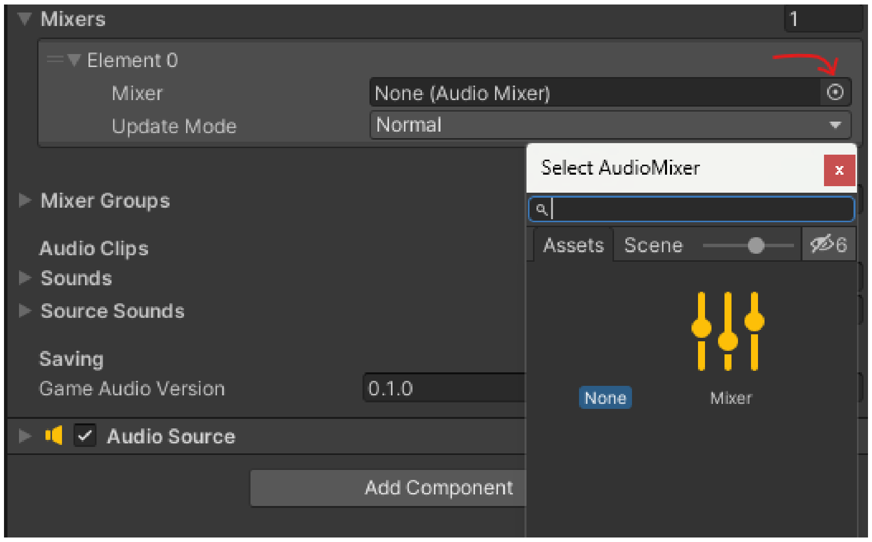
Script: HolyAudioManager

Debugging
- Are Mixers Setup Properly: ☐
- Disable Spatial Blend: ☐
- Enable Debug: ✓

Mixers and Mixer Groups
- Mixers: 0
  - List is Empty
- Mixer Groups: 0

Audio Clips
- Sounds: 1
- Source Sounds: 1

Saving
- Game Audio Version: 0.1.0

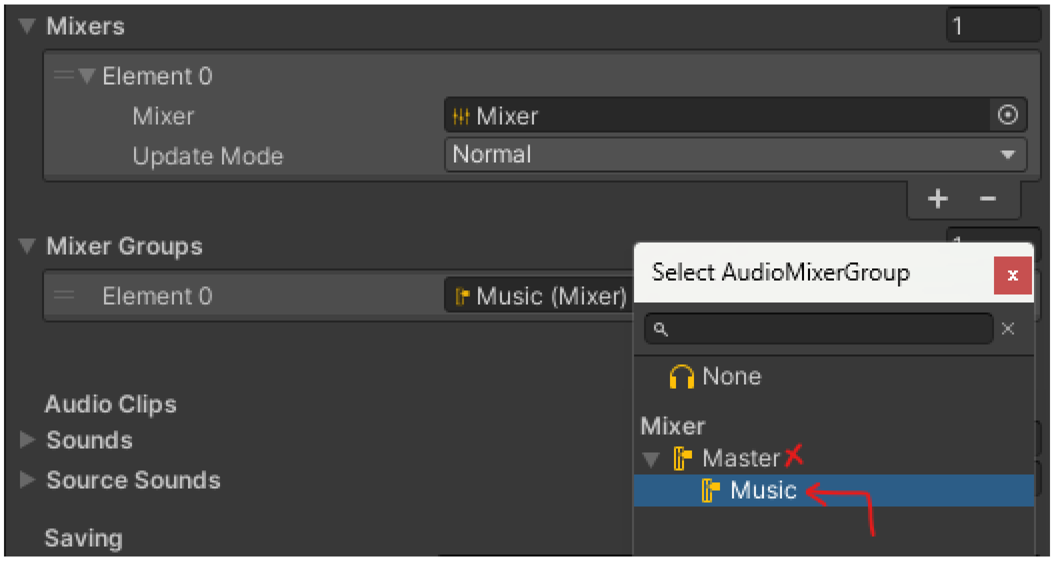


Mixers: 1
- Element 0
  - Mixer: None (Audio Mixer)
  - Update Mode: Normal

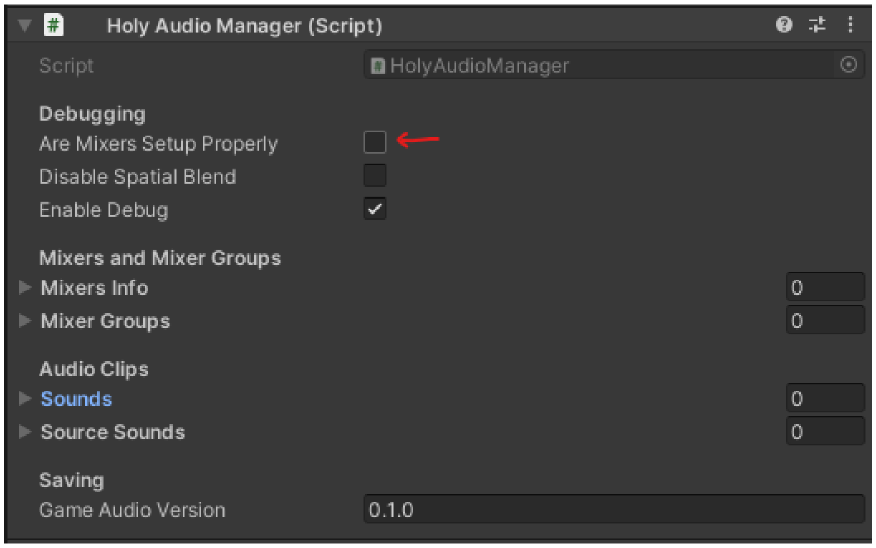
Mixer Groups

Audio Clips
- Sounds
- Source Sounds

Saving
- Game Audio Version: 0.1.0



10. Now that the Mixers and MixerGroups are set up properly you can enable saving by checking the “Are Mixers Setup Properly” property on the HolyAudioManager script in the inspector.

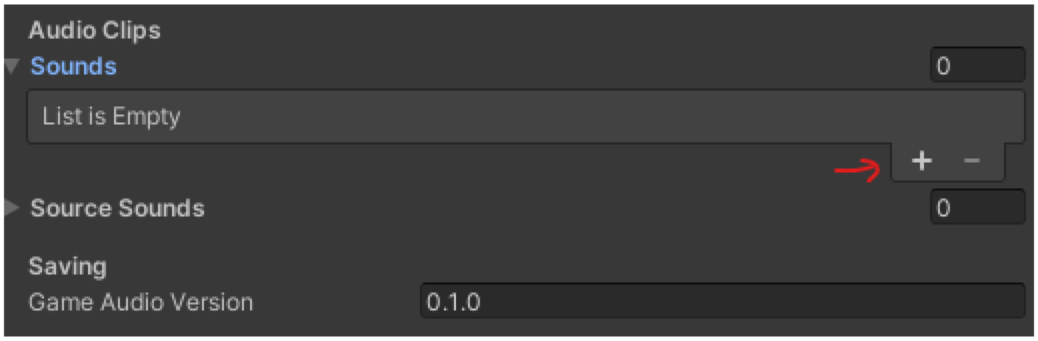

Holy Audio Manager (Script)

Script	HolyAudioManager
Debugging	
Are Mixers Setup Properly	☐
Disable Spatial Blend	☐
Enable Debug	✓
Mixers and Mixer Groups	
Mixers Info	0
Mixer Groups	0
Audio Clips	
Sounds	0
Source Sounds	0
Saving	
Game Audio Version	0.1.0

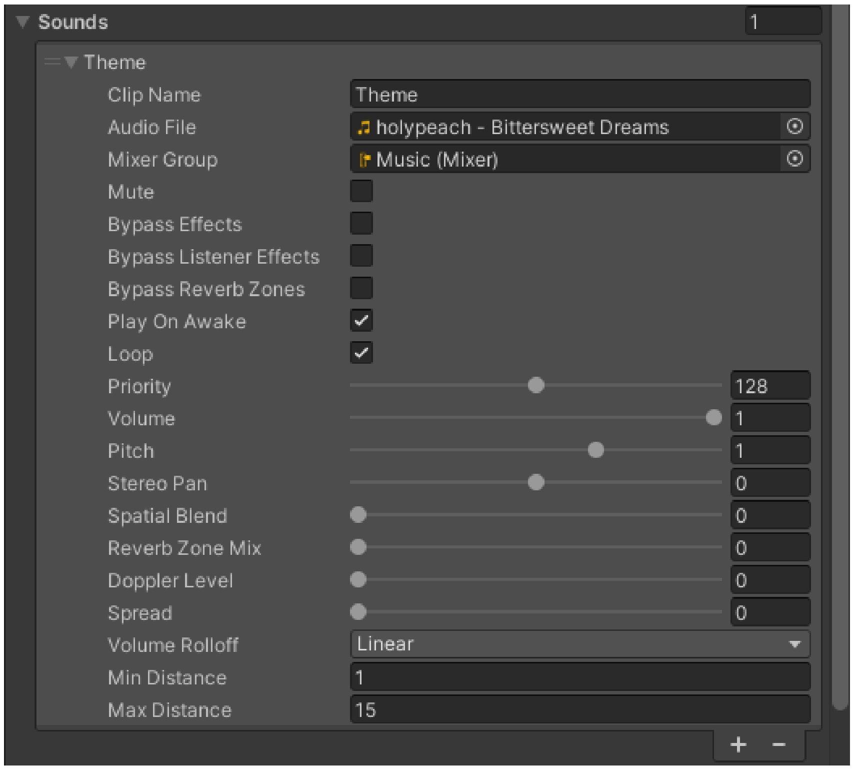
Sounds Setup

“Sounds” are the clips you can play when the game runs. A “Sound” in this context holds all the information for the AudioSource that plays the sound.

1. To add sounds to the HolyAudioManager (also applies to the HolyLocalAudio) go to the script and click on the Sounds section. Click the + button to add a new sound. Alternatively you can increase the number on the right rise which is the length of the array.


Audio Clips
Sounds 0
List is Empty
Source Sounds 0
Saving
Game Audio Version 0.1.0

2. Fill out the fields for the Sound. Make sure to set the priority to 128, **Volume and Pitch to 1**, the Min Distance to 0 or 1, and the Max Distance to whatever you feel like (Set it to 15 for now). You can play around with these values later if you don't know what they do, as well as the Volume Rolloff. Here are some "Default" values.


Sounds 1
Theme

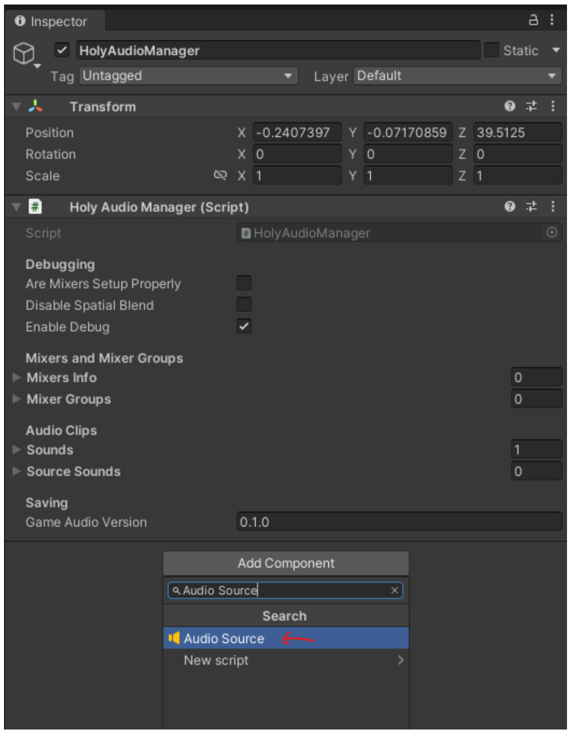
Clip Name	Theme
Audio File	holypeach - Bittersweet Dreams
Mixer Group	Music (Mixer)
Mute	☐
Bypass Effects	☐
Bypass Listener Effects	☐
Bypass Reverb Zones	☐
Play On Awake	✓
Loop	✓
Priority	128
Volume	1
Pitch	1
Stereo Pan	0
Spatial Blend	0
Reverb Zone Mix	0
Doppler Level	0
Spread	0
Volume Rolloff	Linear
Min Distance	1
Max Distance	15

3. **Spatial Blend** is a very important setting, it determines if the sound is going to be 2D or 3D, meaning will you be able to tell the direction of the sound or not as well as volume. If it is set to 0 it will behave like a 2D sound. You'll be able to hear it no matter where you are, I recommend this for UI, Player sounds, and Music. When setting up sounds for entities, I recommend switching to 3D to be able to hear where enemies are coming from! When doing this don't forget to set up the Volume Rolloff, Min and Max Distances properly.
4. Now you can access these Sounds through code!

SourceSounds Setup

SourceSounds are very similar to Sounds. The way to set them up is slightly different. I added this option because the only way to create custom Volume Rolloff Graphs is through the inspector. If you would like to use this feature use SourceSounds.

1. Add an AudioSource component to the object and set it up.



Inspector
HolyAudioManager — Static
Tag Untagged — Layer Default

Transform
Position X -0.2407397 Y -0.07170859 Z 39.5125
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Holy Audio Manager (Script)
Script HolyAudioManager

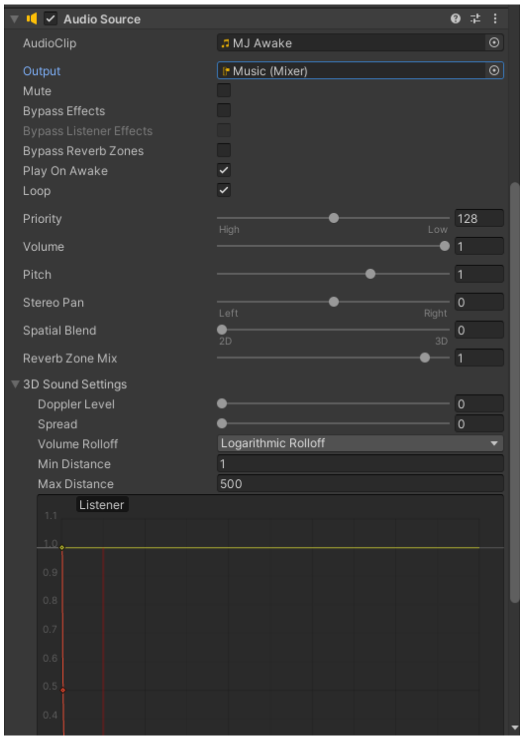
Debugging
Are Mixers Setup Properly ☐
Disable Spatial Blend ☐
Enable Debug ✓

Mixers and Mixer Groups
Mixers Info 0
Mixer Groups 0

Audio Clips
Sounds 1
Source Sounds 0

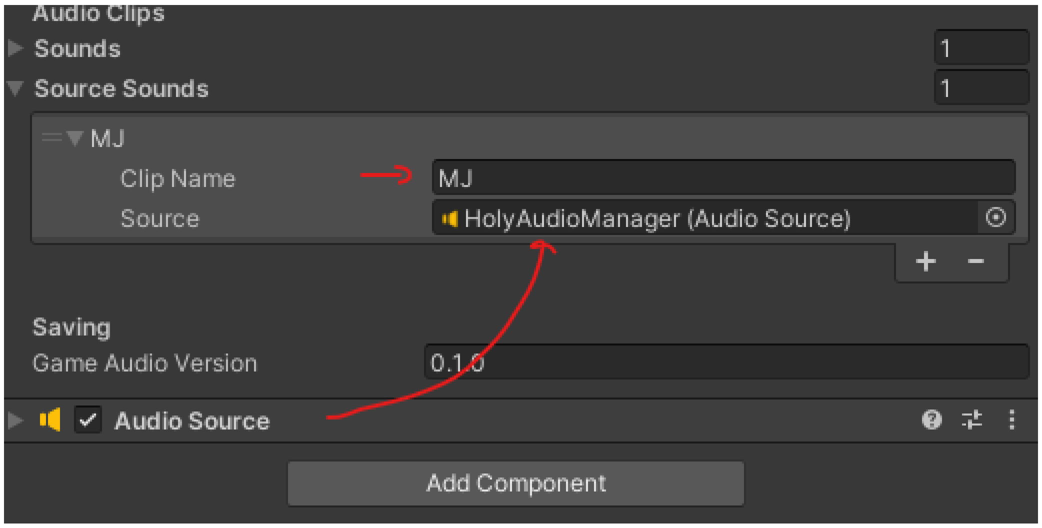
Saving
Game Audio Version 0.1.0

Add Component
Audio Source
Search
Audio Source
New script



Audio Source
AudioClip MJ Awake
Output Music (Mixer)
Mute ☐
Bypass Effects ☐
Bypass Listener Effects ☐
Bypass Reverb Zones ☐
Play On Awake ✓
Loop ✓
Priority 128 (High — Low)
Volume 1
Pitch 1
Stereo Pan 0 (Left — Right)
Spatial Blend 0 (2D — 3D)
Reverb Zone Mix 1
3D Sound Settings
Doppler Level 0
Spread 0
Volume Rolloff Logarithmic Rolloff
Min Distance 1
Max Distance 500

2. Open the Source Sounds section and add a SourceSound.
3. Give the SourceSound a unique name and drag the AudioSource you made to the corresponding field.



Audio Clips
Sounds 1
Source Sounds 1
MJ
Clip Name MJ
Source HolyAudioManager (Audio Source)

Saving
Game Audio Version 0.1.0

Audio Source

Add Component

4. Now you can access these SourceSounds through code!

Sounds and **SourceSounds**

Sounds

Refers to a HolySound object that is a part of the Sounds list on the inspector of either the HolyAudioManager or the HolyLocalAudio. “Sounds” are the clips you can play when the game runs. A “Sound” in this context holds all the information for the AudioSource that plays the sound.

SourceSounds

Refers to a HolySourceSound object that is a part of the SourceSoundslist on the inspector of either the HolyAudioManager or the HolyLocalAudio. SourceSounds are very similar to Sounds. The way to set them up is slightly different. I added this option because the only way to create custom Volume Rolloff Graphs is through the inspector. If you would like to use this feature use SourceSounds. But you can use both at the same time as long as you use unique names for both.

Having **Duplicate Names**

I’ve said multiple times that you should have unique names for your Sounds and SourceSounce, but you can as long as you are aware of its effects. There are 2 outcomes of using duplicate names. If you have 2 Sounds or 2 SourceSounds that have the same name whichever comes first in the list will play/play once/pause/unpause/stop, and only the first sound. This is because only the first sound will be stored internally in the AudioManagers. This is because when the Sounds and SourceSounds are being stored it checks for duplicates, if it detects this it displays an error in the Debug Log and doesn’t add the duplicate. So you will be alerted of this in a Debug message when the game loads if there are any duplicate names. This is also the case for Mixers and MixerGroups. But if you have 1 Sound that shares a name with a SourceSound there will be a warning but both sounds will play. This might have some use which is why I didn’t completely remove this effect.

Saving

HolyAudioData contains all the information that will be saved. The class itself is marked as [System.Serializable] meaning the data in the object can be deconstructed, stored, and later constructed from the stored file which can be used to restore the volume values that the player had set while the game was running.
The HolyAudioSaver class performs the creation of the file as well as its "reconstruction," but the methods in the HolyAudioManager actually create the HolyAudioData object and pass it to the HolyAudioSaver. The HolyAudioManager also retrieves the data object from the HolyAudioSaver and loads the data.

GameAudioVersion

This refers to the variable

```
public string GameAudioVersion;
```

in the HolyAudioManager which can be edited from the Inspector. This is important to make sure the right levels are loaded for each mixer and mixer group. If you change the order of either Mixers or MixerGroups in the HolyAudioManager arrays or add more of any of these I recommend that you change the version so the wrong values are not loaded. Not doing so will not result in errors but it might load the wrong values in the wrong place. Keep this in mind if you release a game and plan on updating it. Once again, it will not mess anything up but it will be unpleasant for the end user. If the version is changed the saved audio data will not be loaded and audio settings will be reset. This is because the "version of the game" does not match the file's "version of the game."

Save **File** and **Path**

By going to the HolyAudioSaver script you can edit the name of the file as well as its extension. You can also add the relative file path:

```
    private static string filePath = "/AudioSettings.holy";
```

The actual path used for the program is:

```
Application.persistentDataPath + filePath
```

If you don't know what this above is, visit this link.

Debugging and **Error Messages**

Debugging

Before building the game for release, set the "Enable Debug" field in the HolyAudioManager and all HolyLocalAudio to false.

Error **Messages**

All the Debug messages from this module follow the same general format, for example:

1.
```
HolyAudioManager|Play: A clip has been found in Sounds and SourceSounds
```
2.
```
HolyLocalAudio|Sound|PlayOnce: Clip has played!
```

First is the Script the log message is coming from. Then occasionally, a specific detail as seen in 2 above which says “Sound.” This is saying that a clip has played, but specifically a Sound. Then the Method the log message is coming from, and finally the actual message.

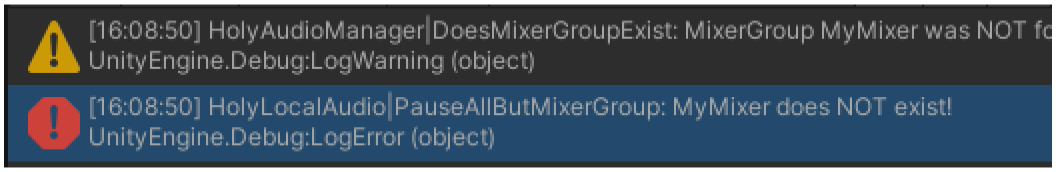
There are 3 debug message types. Normal logs (Debug.Log) that notify that something was performed; these can be disabled by turning off debugging, this is done by unchecking the "Enable Debug" field. Warning logs (Debug.LogWarning) that notify of something important that needs review, and Error logs (Debug.LogError) that indicate something went wrong with the program and should be fixed. Most warning logs will not be disabled by disabling debugging. Error logs are not affected by the debugging option.

Error **Return Values**

To avoid using nullable return types I’m returning a specific value, -999.9f, on getters for volume and clip length. I’m not 100% sure if this is the best move but it’s simple for now. I think creating custom exceptions would be safer but doing it this way is simpler, and as long as you read log messages you will be fine.

Some **Special Cases**

Two Debug Messages



```
[16:08:50] HolyAudioManager|DoesMixerGroupExist: MixerGroup MyMixer was NOT fo
UnityEngine.Debug:LogWarning (object)
[16:08:50] HolyLocalAudio|PauseAllButMixerGroup: MyMixer does NOT exist!
UnityEngine.Debug:LogError (object)
```

At times you might see two error messages. This is because I am using a method that already has debug messages such as DoesMixerGroupExist() from the HolyAudioManager. If you use a method such as PauseAllButMixerGroup() from a HolyLocalAudio script, and there is an error both methods will report their error. If you read the messages Unity actually tells you where the messages originate but for simplicity and readability I made sure to add a debug message for both.

Discussion on Possible **Errors**

HolyAudioManagerInstance and **GlobalHolyAudioManager**

This is one of the issues I encountered during the development of this project, it will most likely not be a problem during normal usage but I thought to include it here because I know it exists and it could potentially become a headache.

HolyAudioManagerInstance

```
public static HolyAudioManager HolyAudioManagerInstance {get; private
set;}
```

At some point during execution you may want to use the Global HolyAudioManager. To do this you can use the **HolyAudioManager.HolyAudioManagerInstance** property. Even though the variable is static it is instantiated when the Awake() method is run, meaning that it's not null only after the program starts.
During testing I encountered this issue in the form of trying to access HolyAudioManager.HolyAudioManagerInstance from the HolyLocalAudio script. Occasionally Unity would display an error telling me that the instance of the object I was trying to access was null. This came from trying to access the variable in the Awake() method of the HolyLocalAudio script. After building a project the audio from an object with a HolyLocalAudio script would not play. To solve this I moved the code that would access the HolyAudioManager.HolyAudioManagerInstance from the Awake() to the Start() method. This solved the issue and it's something to keep in mind when accessing this property.

GlobalHolyAudioManager

```
public HolyAudioManager GlobalHolyAudioManager;
```

Similarly to the **HolyAudioManagerInstance** (read previous entry), the **GlobalHolyAudioManager** property should only be accessed after the Start() method is called because it may be Null.
That being said, I recommend referencing **HolyAudioManager.HolyAudioManagerInstance** directly instead of using GlobalHolyAudioManager to avoid these issues if you are doing something that is "time sensitive," meaning before the normal Update() and FixedUpdate() loops. If you are referencing these only during normal execution then there should not be any problems.

Other Methods

In the **HolyLocalAudio** script there are several methods that are dependent on the availability of **HolyAudioManager.HolyAudioManagerInstance**, which are the following:

```
GetMixer(string mixerName)
GetAllMixers()
GetMixerGroup(string mixerGroupName)
GetAllMixerGroups()
DoesMixerExist(string mixerName)
DoesMixerGroupExist(string mixerGroupName)
```

Out of all of these only **DoesMixerGroupExist()** is used in HolyLocalAudio. Methods that use DoesMixerGroupExist() are:

```
public void PauseAllButMixerGroup(string mixerGroupName)
public void PauseAllFromMixerGroup(string mixerGroupName)
public void UnpauseAllButMixerGroup(string mixerGroupName)
public void UnpauseAllFromMixerGroup(string mixerGroupName)
public void StopAllButMixerGroup(string mixerGroupName)
public void StopAllFromMixerGroup(string mixerGroupName)
```