Audio API - Documentation & Reference Manual

by William Chilcote

Contents

1. Advantages & Implementation
2. Database Editor
3. Audio Types
    1. Sound
    2. Ambient
    3. Music

Advantages & Implementation

The default Audio System in Unity makes use of `AudioSource` and `AudioListener` components, that play `AudioClip` assets. This audio system builds upon, rather than replaces, this default system in order to provide higher level functions that are more specifically suited to any 2D RPG.

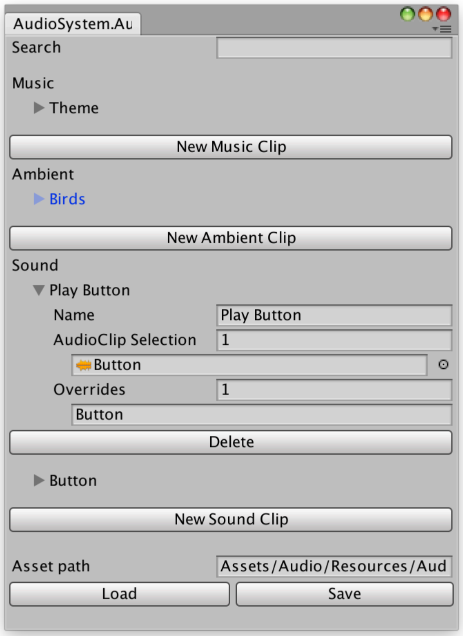
This Audio System breaks down audio into three main types, each with methods from the `Audio` class that can be used to play and stop these audio types from any script. This provides several advantages over the standard `AudioSource` method, such as:

- Fully static access
- Scene-persistent audio
- Single asset database with custom editor
- Multiple `AudioClip` assets for each database entry
- Same-frame duplicate filtering
- Audio clip overrides
- Additional type-specific behaviors, as explained in sections below

All methods and properties that can be used publicly are exposed through the `Audio` class, and are all static. While `Audio` is a static class, on its first use, it generates a singleton instance of `AudioSystem.AudioManager`. This is used to process audio effects on a frame-to-frame basis in its `Update` loop. All classes that are used internally and not publicly accessible through this Audio API are contained in the `AudioSystem` namespace, as to not overpopulate the global namespace.

Database Editor

All calls to play any audio through the API require a reference to an entry in the audio database. To add new audio entires, edit the `AudioDB` assets through the custom editor. Only one `AudioDB` asset can be used at a time. The asset must be stored in the top level of a “`Resources`” directory, and be named “`AudioDB.asset`”. The custom editor can be found under Window > Audio Database. The editor includes buttons to create new audio entries for each type, a field to search entries by name, buttons regarding the saving and loading of the `AudioDB` asset, and a list of expandable information for each existing entry.



AudioSystem.Au

Search

Music
- ▶ Theme

Ambient
- ▶ Birds

Sound
- ▼ Play Button
  - Name: Play Button
  - AudioClip Selection: 1
    - Button
  - Overrides: 1
    - Button

- ▶ Button

Asset path: Assets/Audio/Resources/Aud

The information for each entry includes a text field for the entry’s name, an array of variable size containing references to `AudioClip` assets that can be played when the entry is called, and a secondary array of variable size containing strings that reference the names of other entries of the same type that this entry will override if called during the same frame. The entry name must be unique to its type. If the `AudioClip` selection array contains more than one element, a checkbox labeled “Pick random” will be displayed as well. If this is checked, the entry will play a random `AudioClip` each time is is called. If it is not checked, it will play the `AudioClip` assets in order, starting from a random index. While these settings are available for all three audio types, they are not necessarily recommended or even implemented for each type.

The asset must be saved before exiting the Unity Editor or compiling in order for any changes to be retained.

Audio Types

The Audio System makes use of three different types of audio, each with their own behaviors, settings, and relevant methods and classes. All calls to play each audio type can be done through `Audio.Play()`, as the method is overloaded with several argument combinations to accommodate all types.

Generally, a class inheriting from `AudioSystem.ClipSettings` should be passed to the method. This includes `SoundClip`, `AmbientClip`, and `MusicClip` for each type. While each inheriting class contains its own information specific to the type, their base class, `ClipSettings`, contains type-independent information, such as the `pitch` and `volume` properties, which are used to directly set the similarly named properties on their resulting `AudioSource` component, and both default to 1. The first required argument to the constructor of any `ClipSettings` inheriting class is a string representing the name of the desired entry within the relevant type. Each inheriting type has different constructors, utilizing different combinations of arguments, but all properties are public and are therefore able to be changed manually after the constructor call.

Sound

Calls for audio of the Sound type should generally be used for short, non-looping clips of sound.

Sound types can be played by passing a `SoundClip` instance to `Audio.Play()`. The `SoundClip` class contains the `delay` property, which represents the amount of time in seconds that will pass between when the call is made, and when the sound will play, and defaults to `0`. Additionally, a string containing the name of the Sound entry can be passed to `Audio.Play()` to play the Sound entry with no additional settings.

There are two available constructors for the `SoundClip` class:

```
SoundClip (string name)
SoundClip (string name, float delay)
```

Examples of syntax to call a Sound type audio entry are as follows:

```
// Example 1
Audio.Play(new SoundClip("Button"));

// Example 2
SoundClip sound = new SoundClip("Button");
sound.delay += 0.3f;
Audio.Play(sound);

// Example 3
Audio.Play(new SoundClip("Button", 0.5f) {
    pitch = 2,
    volume = 1.2f
} );

// Example 4
Audio.Play("Button");
```

Ambient

Ambient audio types are used to play looping background audio clips that play depending on the player's location, with variable linear decay. Ambient audio will be started from a random time in the `AudioClip`. If the player is in range of multiple Ambient types of the same name and settings, only one will play, and will have the volume of the loudest entry instance out of all merged instances.

Ambient types can be played by passing a `AmbientClip` instance to `Audio.Play()`. The `AmbientClip` class contains the properties `bounds` and `rolloff`. `bounds` represents the `Collider2D` that player must have entered for the audio to play. `rolloff` is the distance in Unity units that the player must be inside the collider from the nearest edge before the volume reaches 100%. Audio decay is linear, starting at 0% from the collider's edge, and 100% at a distance `rolloff` from the nearest edge. Audio is 0% anywhere outside the collider, and 100% everywhere inside that is not within the `rolloff` area. If `rolloff` is set to `0`, the audio will instantly start and stop as the player crosses the edge of the collider.

As calls to play an `AmbientClip` simply attach an `AudioSystem.AmbientTrigger` component to the `GameObject` of the specified `Collider2D`, the same can be accomplished in the Unity Editor by manually attaching the component to the `GameObject` in the scene hierarchy. If the component is manually added, values for the entry's name and `rolloff` setting will need to be specified in the Inspector.

There are two ways to remove Ambient audio types:

- The first is to pass the name of an Ambient type's entry to `Audio.StopAmbient()`. This will remove all Ambient types currently in the scene with the specified name.
- The second is to pass a `Collider2D` reference to `Audio.StopAmbient()`. This will remove all Ambient types associated with the collider, of any name.

As this simply removes the `AudioSystem.AmbientTrigger` component from the relevant `GameObject`, this too can be done manually if needed.

There are two available constructors for the `AmbientClip` class:

```
AmbientClip(string name, Collider2D bounds)
AmbientClip(string name, Collider2D bounds, float rolloff)
```

Examples of syntax to call an Ambient type audio entry are as follows:

```
Collider2D collider = GetComponent<Collider2D>();

// Example 1
Audio.Play(new AmbientClip("Birds", collider));

// Example 2
AmbientClip ambient = new AmbientClip("Birds", collider);
ambient.rolloff = 2;
Audio.Play(ambient);

// Example 3
Audio.Play(new AmbientClip("Birds", collider, 1.5f) {
    pitch = 0.8f
});
```

Music

Music audio types are used to set the single audio track that loops indefinitely until stopped or changed. Music types will always gradually fade in or out when started or stopped. There can only be one Music type playing at any given time, so if any calls to play a Music type are made when another Music type is currently playing, the currently playing entry will fade out first.

Music types can be played by passing a `MusicClip` instance to `Audio.Play()`. The `MusicClip` class contains no additional custom properties. Additionally, a string containing the name of the Music entry can also be passed to `Audio.Play()` to play the Music entry, provided there is no Sound type in the database with the same name.

There are two ways to remove Music audio types:

- The first is to call `Audio.StopMusic()`. This will fade out the currently playing Music entry and replace it with nothing.
- The second is to call `Audio.Play()` and pass a null `MusicClip` reference into it. This has the same effect as `Audio.StopMusic()`.

`Audio` also contains two properties that can be used as settings for Music types. `Audio.playMusic` is a boolean property, that if set to `false`, will mute all calls to play Music types. This should generally be set using a settings UI frontend. `Audio.fadeSpeed` is a multiplier applied to the default rate at which audio fades in and out when stopping and starting Music types. The default rate is 100% per second.

There is one available constructor for the `MusicClip` class:

```
MusicClip (string name)
```

Examples of syntax to call a Music type audio entry are as follows:

```
// Example 1
Audio.Play(new MusicClip("Theme"));

// Example 2
MusicClip music = new MusicClip("Theme");
Audio.Play(music);

// Example 3
MusicClip music = new MusicClip("Theme") {
    pitch = 2.0f
};
Audio.Play(music);

// Example 4
Audio.Play("Theme");
```