Rapture3D for Unity Developer Guide

v3.4.2

Table of Contents

Table of Contents

1 Introduction

1.1 Welcome to Rapture3D for Unity

Rapture3D for Unity is a set of native plugin libraries and a C# "wrapper" layer for use in Unity. This allows high quality 3D audio to be added to your Unity scene. Rapture3D for Unity is a part of Rapture3D Universal.

Rapture3D Universal is a substantial rewrite of the original OpenAL-based Rapture3D game engine, reusing the best bits. It has been reorganised to be more self-contained, to be simpler and more flexible to use, to run faster and to produce even better audio. New features have been added, particularly around the handling of pre-rendered multichannel audio "beds". Features include:

- Efficient placement of large numbers of sources (mono sounds) and beds (multichannel sounds) into a convincing 3D sound scene.
- High quality binaural stereo output for headphones, including a choice of HRTF.
- High quality 3D rendering for speakers, including stereo, 5.1 and 7.1.
- Support for beds using pre-rendered ambisonic material, including support for 3D material mastered with Blue Ripple Sound's O3A studio tools.
- Advanced passive upmixing for beds from multichannel sound in conventional formats (e.g. 5.1).
- Beds can be moved around and reoriented so the player can "walk into" a pre-rendered audio scene and turn their head, or be presented with "near" and "far" field elements with slightly different parallax depending on head location.
- Broad hardware support and scalable quality settings allowing use with a wide range of devices.
- Internal Higher Order Ambisonic (HOA) bus which can scale from first to fifth order.
- Extremely fast SSE/NEON-optimised floating-point signal path.
- Self-contained library which does not rely on external data files or software.
- High quality sample rate conversion (also used for Doppler simulation).
- 3D spatial reverb with directivity controls, and filters for distance and occlusion simulation.

This user guide assumes that you are familiar with Unity audio and basic Unity C# scripting.

The underlying signal processing is provided using a fast native plugin which also has a direct C/C++ binding. If you wish to use Rapture3D Universal directly in C/C++ (or other compatible languages), separate documentation is available.

Please note that this edition of Rapture3D is not the same as the older OpenAL driver used by the Rapture3D "Game", "User" and "Advanced" editions.

1.2 Basic Concepts

This version of Rapture3D uses a single listener which renders audio based on the player's perspective in an audio scene. The audio output takes into account the location and orientation of the listener, so it is usually placed on the same rigid body as the Unity camera, along with the standard Unity `AudioListener`.

To place sounds in the audio scene there are three options. Generally, you'll want to use an audio adapter, which takes audio from the main Unity audio system and feeds it into Rapture3D for 3D placement. There are also mono sources and multichannel beds available, which are simpler in some ways and have some special features. However, these are less well integrated into the standard Unity audio pipeline, so adapters are usually best.

Reverb is also available and can be controlled using reverb zones.

These features work as individual C# scripts which can be added to game objects as components:

R3dAudioListener.cs	The audio listener controls the player's perspective on the audio scene and produces Rapture3D's final rendered output.
R3dAudioAdapter.cs	Audio adapters place mono audio from Unity's normal audio pipeline into the Rapture3D mix.
R3dAudioSource.cs	Audio sources place mono sound into the Rapture3D mix. Streaming assets are supported.
R3dAudioBed.cs	Audio beds place multichannel sound into the Rapture3D mix, including pre-rendered ambisonic assets. Streaming assets are supported.
R3dAudioReverbZone.cs	Reverb zones controls the Rapture3D reverb using settings that depend on the audio and listener locations in the audio scene.

When Rapture3D for Unity reads audio files (rather than Unity itself doing it), this is done through a library called "BSR", which can stream audio from your Unity project's `Assets/StreamingAssets/` directory. There is more about this later.

1.3 Getting Started

1.3.1 Import the R3DU Unity Package

Rapture3D comes as a Unity "package" which will be in your installation directory, generally `C:/Program Files/Rapture3D for Unity/` on Windows or `/Applications/Rapture3D for Unity/` on macOS. The package itself is **`R3DU.unitypackage`**.

So, first of all, import this package into your Unity project. To do this, pull down the Unity "Assets" menu, select "Import Package" and then "Custom Package...". Find the Unity package above and click "Open".

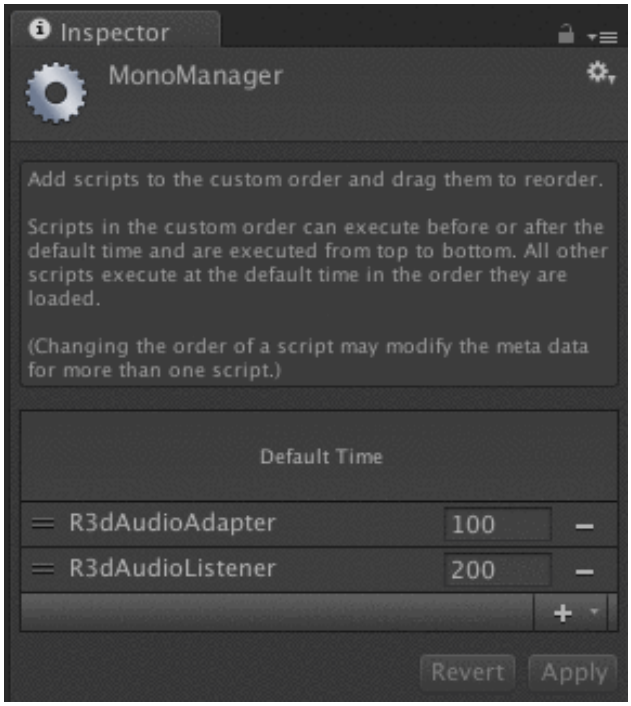
Once imported, your project should contain a number of new scripts under `Assets/Scripts/`, and native plugins under `Assets/Plugins/`.

1.3.2 Configure Script Ordering

Your Unity project needs to be configured to ensure that when Rapture3D scripts are run, two in particular are run in the right order. Forgetting to do this can produce some strange results, so please try to remember!

To do this, pull down the Unity "Edit" menu, select "Project Settings" and then "Script Execution Order". This should bring up the "MonoManager" inspector.

Press the plus sign button and add the `R3dAudioAdapter` and `R3dAudioListener` scripts, in that order, so the audio adapter is before the audio listener. Click "Apply" to save your change.



1.3.3 Preparing the Listener

Here we need to start to make creative decisions. For now, let's assume you want the audio perspective to be the same as the main camera perspective, so sounds in front of the camera should also sound in front of the player (and so on). This is generally a sensible choice!

Find the main camera in your scene, and bring up its inspector. Then press the "Add Component" button, select "Scripts", and finally select "R 3d Audio Listener".

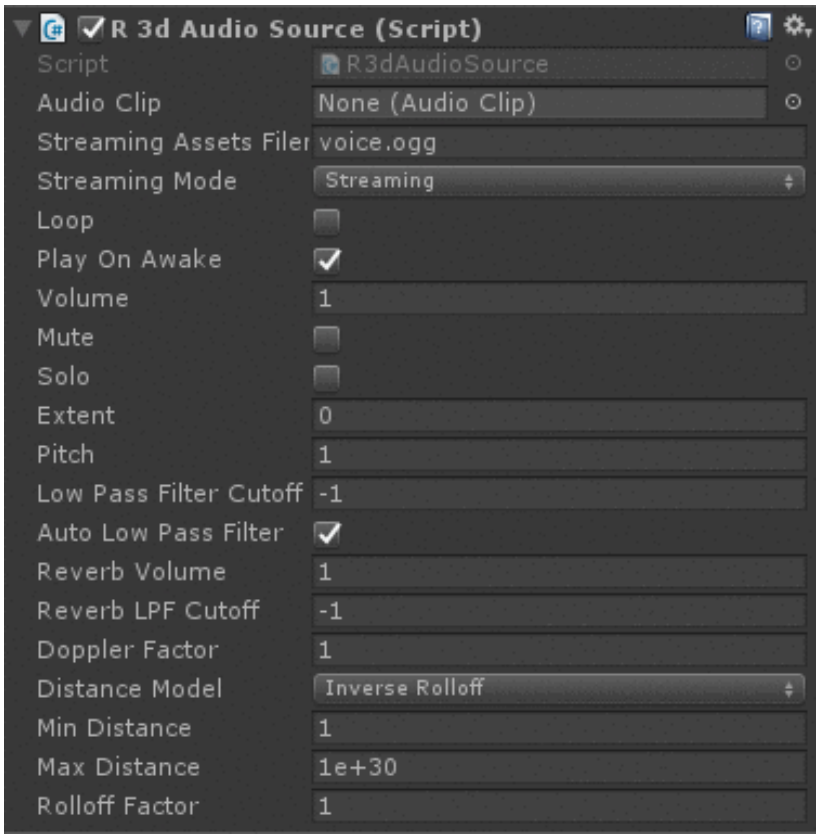
The main Unity audio listener should *also* be present, and ordering matters. The Unity "Audio Listener" must appear *above* the Rapture3D "R 3d Audio Listener". This arranges for the audio output from Rapture3D to be fed into Unity's normal audio pipeline.

1.3.4 Try a Test Sound

Put a mono (single-channel) WAV or OGG file into `Assets/StreamingAssets/` inside your Unity project, for instance `Assets/StreamingAssets/voice.ogg`. If this directory does not exist, create it with exactly this name; Unity has special handling for this directory.

Choose (or put) a game object near the camera, and attach a "R 3d Audio Source" script.

In the game object inspector, find the new component and set its "Streaming Assets Filename" to the file mentioned above, but without the full path (for instance "`voice.ogg`").



1.3.5 Play!

Set like this (with "Play on Awake" enabled) the sound will play automatically when the Unity scene is loaded, so hopefully all you need to do now is run your project.

1.4 Upgrading Rapture3D for Unity

If you have a previously imported Rapture3D assets into your Unity project, you may need to remove the following files to stop Unity from importing a new version as new assets. Please be careful doing this, to make sure you do not lose any of your own work. We recommend you back up your project before upgrading.

```
Assets/Gizmos/R3d*.png
Assets/Plugins/Android/libs/*/libBRS.so
Assets/Plugins/Android/libs/*/libR3DU.so
Assets/Plugins/iOS/libBRS.a
Assets/Plugins/iOS/libR3DU.a
Assets/Plugins/*BRS.bundle
Assets/Plugins/*R3DU.bundle
Assets/Plugins/*/*BSR.dll
```

```
Assets/Plugins/*/*R3DU.dll
Assets/Scripts/R3D/R3d*.cs
```

1.5 C and C#

Under the hood, you are using "Rapture3D Universal". This provides a high performance native C/C++ 3D rendering library with a control interface in C. But on top of this, we provide some C# scripts which define the components listed above, which make the library easy to use in Unity.

The library is "native", in the sense that it targets the low-level instruction set used by particular hardware directly. This is different from C#, where the same code is intended to run everywhere without modification. Native code allows access to hardware-specific features that can make things run very, very fast, but it also means that you need the right native libraries for the platforms you are currently targetting. On Windows, these native libraries are typically `.dll` files. On Android, `.so` files. And so on.

Rapture3D for Unity uses a second native library called "BSR" to perform buffered stream reading of audio files from disk (or other storage).

By default, when the Unity package is imported into a project, native plugins are installed to the following paths:

Android ARM v7a	`Assets/Plugins/Android/libs/armeabi-v7a/libBSR.so` `Assets/Plugins/Android/libs/armeabi-v7a/libR3DU.so`
Android ARM x86	`Assets/Plugins/Android/libs/x86/libBSR.so` `Assets/Plugins/Android/libs/x86/libR3DU.so`
Android ARM-64 v8a	`Assets/Plugins/Android/libs/arm64-v8a/libBSR.so` `Assets/Plugins/Android/libs/arm64-v8a/libR3DU.so`
iOS (static library)	`Assets/Plugins/iOS/libBSR.a` `Assets/Plugins/iOS/libR3DU.a`
macOS	`Assets/Plugins/BSR.bundle` `Assets/Plugins/R3DU.bundle`
Windows 32bit	`Assets/Plugins/x86/BSR.dll` `Assets/Plugins/x86/R3DU.dll`
Windows 64bit	`Assets/Plugins/x86_64/BSR.dll` `Assets/Plugins/x86_64/R3DU.dll`

At the time of writing, the vast majority of support issues relating to Rapture3D turn out to be due to something changing around the location, naming or Unity settings for these plugins, so if something is not working you may wish to come back here. The plugins should have the names and locations as above and the Unity Inspector should show them targetting the correct platform and platform settings. For Windows and macOS, the platform settings should be "Editor" *and* "Standalone".

In Unity it is generally best to use the C# scripts, and leave it up to them to talk to the native code. You may modify the C# scripts, but modified scripts are not supported by Blue Ripple Sound.

1.6 Scripting

Using the game object inspector in Unity only takes you so far with audio for game design, because sounds are typically triggered when something happens in the game. For instance, when the trigger of a gun is pulled, a gunshot sound probably needs to start. In Unity, this sort of behaviour is normally managed through scripts.

As usual in Unity, the fields in the Rapture3D scripts that are shown in the inspector are available in scripts. In addition, some Rapture3D components have additional properties and methods that can be called from scripts. These are described later.

1.7 Building for Microsoft Windows

On Windows, Rapture3D Universal requires the Visual Studio C++ 2010 runtime libraries. You will probably need to install these with your product.

1.8 Building for iOS

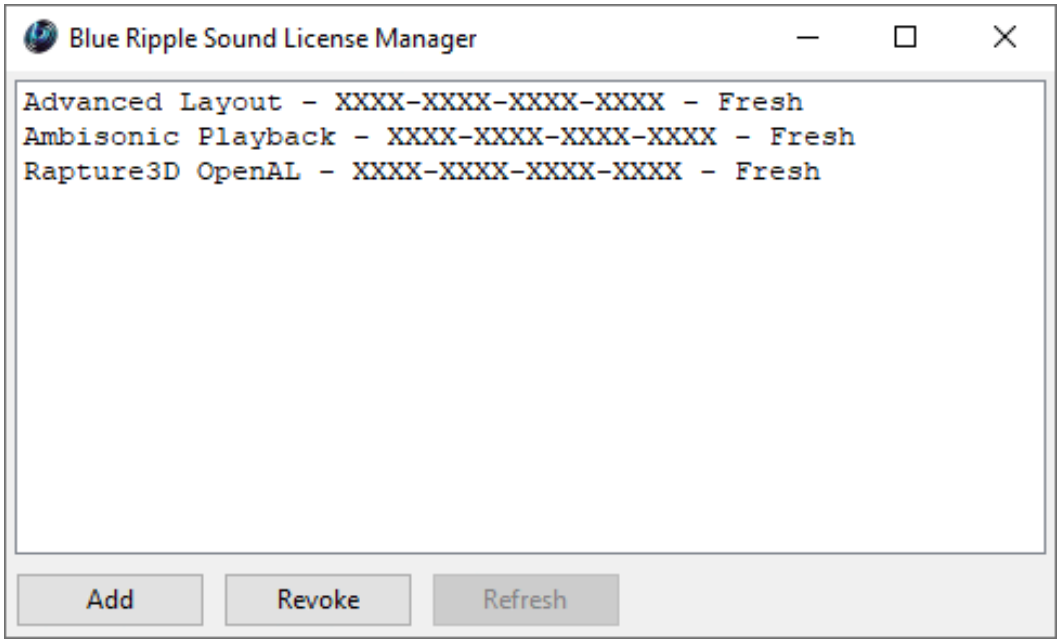
The iOS version of the native library uses the Apple "Accelerate" framework, which may need to be added to the Xcode project during the Unity build process.

1.9 Other Software Used

The BSR native library uses Ogg/Vorbis (see license provided).

1.10 The License Manager

The Blue Ripple Sound License Manager application can be used to move license keys around between computers.



The application is generally available in the Start Menu on Windows, and in your installation directory, which is typically:

- `C:\Program Files\Rapture3D Universal SDK\` on Windows, and
- `/Applications/Rapture3D Universal SDK/` on macOS.

License keys can be removed from a computer using the 'Revoke' button and added with 'Add'. If data is not 'Fresh' this probably indicates a network connectivity issue, in which case the 'Refresh' button may be used. Generally, licenses are refreshed automatically.

2 Rapture3D and the Unity Sound System

2.1 How Rapture3D Integrates With Unity

Rapture3D does sophisticated 3D rendering using Higher Order Ambisonics (HOA). The details of this are not visible in Unity; instead, audio for game sounds is fed into Rapture3D using sources, beds and adapters. The audio output resulting from the rendering is placed back into Unity's main audio pipeline by the `R3dAudioListener.cs` script.

For ordinary mono sound sources in the game world, it is generally best to use a Unity `AudioSource` combined with a Rapture3D audio adapter. However, Rapture3D audio sources and beds have special uses, particular when handling pre-rendered beds.

2.2 Sources and Beds

Rapture3D sources and beds do not use Unity's main audio engine directly at all, so Unity audio filters cannot be used. Instead, sources and beds push sound directly into the Rapture3D audio pipeline.

Sources and beds can read or stream their audio from the `Assets/StreamingAssets/` folder using BSR, or from certain kinds of Unity `AudioClip`.

A typical use for BSR streaming is playback of an HOA asset from the `Assets/StreamingAssets/` folder, for instance 3D audio scenes pre-rendered using Blue Ripple Sound's O3A studio tools. For mobile, it is usually best to compress such asset with Ogg/Vorbis.

2.2.1 Limitations of AudioClips

Sources and beds can use Unity's built-in `AudioClip` class, however there are two important things to know:

1. Unity's `AudioClip` only allows access to its audio data if the clip's load type is set to "decompress on load", so this needs to be set. If this isn't done, Unity will feed silence to Rapture3D.
2. The audio in Unity's `AudioClip` cannot be accessed from the audio thread. To work around this, the entire audio is copied into another buffer, which uses memory.

For these reasons, it is generally better to use audio adapters instead of sources or beds where possible.

2.3 Adapters

Adapters provide a different way for game objects to send audio to Rapture3D. Adapters work as Unity audio filters, so they can be used with ordinary Unity `AudioSource` objects and filters chains.

This means that the audio path is only "outside" Unity between the adapter and the listener, while the Rapture3D spatial audio processing happens.

There is a section on adapters below.

2.3.1 Configuration

If you are using adapters, don't forget that you will need to set the Unity Script Execution order as above.

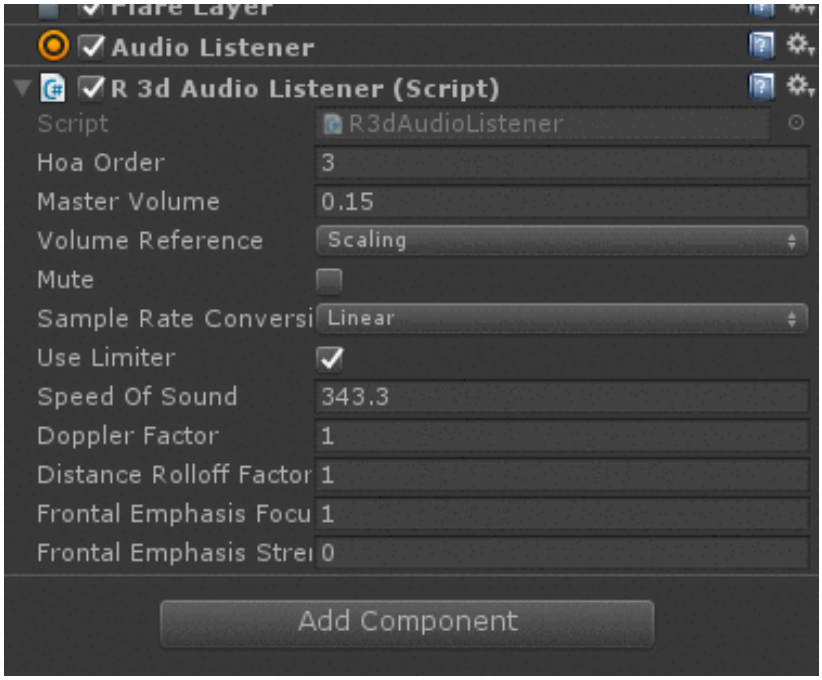
3 The Rapture3D Audio Listener

The Rapture3D Audio Listener outputs all audio for the game world. It can produce mono, stereo or multichannel audio, including synthetic binaural (using HRTFs). The component script for this is `R3dAudioListener.cs`.

The Rapture3D Audio Listener is a Unity audio filter. It produces the "rendered" output from Rapture3D and feeds this back into the standard Unity audio pipeline. Normally the listener will be placed just below a standard Unity `AudioListener` which will receive the audio and present it to the user.

Sounds are rendered to locations in 3D space in a way that takes the listener's location, velocity and orientation into account, so it is normal to attach the listener to the same rigid body, or other reliable source of geometry information, as the Unity camera.

3.1 Main Fields



3.1.1 HOA Order

```
int hoaOrder
```

The HOA (Higher Order Ambisonic) order can be thought of as a spatial quality setting. Whole numbers from 1 to 5 are accepted and 3 ("high quality") is generally a good option.

Higher settings use more CPU cycles than lower ones. If you are only using Rapture3D for ambisonic bed playback and the bed only ever needs to be rotated the order typically need not be set higher than the ambisonic bed's order.

Changing this setting will restart the Rapture3D sound engine, so this is only advisable on startup or in configuration screens.

3.1.2 Master Volume

```
float masterVolume
```

This is a linear gain affecting the whole mix. It is 0.15 by default.

3.1.3 Volume Reference

```
R3dGainReference volumeReference
```

The volume reference sets the reference level for the mix before master volume is applied. The options are:

Scaling	With this reference, overall gain increases with the number of speakers (as more headroom is available). This (with a gain of 0.15) is the default.
Studio	With this reference, overall gain remains roughly consistent regardless of the number of speakers used. Using this and a master volume of 1 will give similar decoder levels to those used by the Blue Ripple studio tools. This is particularly important if you are using Blue Ripple metering in the studio (e.g. "O3A Meter - Karma" for LUFS estimation) and want consistent results here.

3.1.4 Mute

```
bool mute
```

This mutes the output from the listener, so all Rapture3D output stops. Anything else playing is bypassing Rapture3D.

3.1.5 Sample Rate Conversion Mode

```
R3dSRCMode sampleRateConversionMode
```

Sample rate conversion is used when audio rates in assets do not match the Unity sample rate (under "Project Settings" and then "Audio") or when Doppler Shift is in use. Like HOA Order, some options are higher quality but use more CPU cycles. The options are:

Linear	Basic linear interpolation. Not great, but cheap on CPU.
Sinc Low	Short length truncated sinc interpolation.
Sinc Medium	Medium length truncated sinc interpolation.
Sinc High	Long length truncated sinc interpolation. Note that this is not available in the iOS and Android versions of the R3D library and Sinc Medium is used instead.

Increasing the quality setting has a significant cost in CPU use, so depending on your project, it is well worth having a careful listen when choosing.

For direct scripting, the relevant enumeration is defined in script `R3dSRCMode.cs`.

Changing this setting will restart the Rapture3D sound engine.

3.1.6 Use Limiter

```
bool useLimiter
```

When this is enabled, loud audio will be automatically reduced in level to attempt to avoid clipping. This is always recommended, except when tuning the audio levels themselves.

Changing this setting will restart the Rapture3D sound engine.

3.1.7 Speed of Sound

```
float speedOfSound
```

The speed of sound is used in Doppler calculations and the default value (343.3) assumes that units in the game world are metres. If other units are in use, this setting should be changed to correspond. For instance, if the units are centimetres then a value of 34330 might be suitable.

3.1.8 Doppler Factor

```
float dopplerFactor
```

The Doppler Factor can be used to reduce or increase the strength of the Doppler simulation throughout the game world. A value of zero effectively disables Doppler.

Doppler calculations need a listener velocity, as calculated by Unity. The C# scripts attempt to find this by looking for a `Rigidbody`, `CharacterController` or `Camera` component (in that order). See `R3dAudioListener.cs`.

3.1.9 Distance Rolloff Factor

```
float distanceRolloffFactor
```

The Distance Rolloff Factor can be used to reduce or increase the strength of distance rolloff through the game world.

3.1.10 Frontal Emphasis Focus

```
float frontalEmphasisFocus
```

Sounds in front of the listener can be emphasised by lowering the level of sounds in other directions. Focus determines the shape of the emphasis, with higher values using a more focussed (narrow) region. Values are between 0 and 5. The HOA Order also limits how focussed the emphasis can be, so if you chose an HOA Order of 3, the focus effectively will be limited at a value of 3.

3.1.11 Frontal Emphasis Strength

```
float frontalEmphasisStrength
```

Sounds in front of the listener can be emphasised by lowering the level of sounds in other directions (see Frontal Emphasis Focus above). The Frontal Emphasis Strength determines how aggressively sounds not in the focussed region are reduced in level. Values are between 0 (off) and 1 (full).

Frontal emphasis is off by default (by setting this to 0). **We recommend it be left off during normal operation.** It is intended for use sparingly as an effect.

3.2 Scripting Properties

3.2.1 Decoder

```
R3dDecoder decoder
```

This property allows the active decoder to be checked or changed.

The decoder controls the final output rendering produced by Rapture3D. There are a number of Rapture3D decoders available, depending on the version and installation of Rapture3D in use. They can be enumerated using `R3dDecoder.GetDecoders()`. Decoders have the following public properties:

`channelCount`	The number of output channels in the layout in use, for instance two for stereo, or six for 5.1. This must match the active Unity setting for the decoder to work.
`layoutName`	The English name of the layout, for instance "Headphone Stereo", "Stereo" or "Surround 5.1".
`methodName`	The English name of the method used to produce output for this layout. For instance, there are a number of different HRTF methods for headphones.
`uri`	A resource identifier for the decoder.

You may wish to localise some or all of this data and offer it to the user in an audio options screen.

If no decoder is selected then a default is chosen using the current Unity channel count and `AudioSpeakerMode`. However, a number of decoders are typically available for a particular speaker mode.

Setting the `R3dAudioListener.decoder` property to one of these decoders will switch the Rapture3D renderer to the new decoder, assuming the channel count is compatible. This will restart the Rapture3D sound engine.

3.2.2 Group Delay Samples

```
int groupDelaySamples [read-only]
```

This property is read-only and provides the nominal group delay of the current Rapture3D decoder. In some ways this can be thought of as a processing latency and is typically only needed for tight A/V sync. It is measured in samples, so you can divide it by the Unity sample rate to find the group

delay in seconds.

3.2.3 Tail Samples

```
int tailSamples [read-only]
```

This property is also read-only and estimates the time (in samples) between audio input to Rapture3D stopping and audio output stopping. This is typically only relevant in specialised shutdown scenarios.

3.2.4 Pause

```
bool pause
```

This provides a simple way to pause or un-pause sounds in the Rapture3D audio engine.

Note that it is possible for sources and beds to exclude themselves from this by setting their `ignoreListenerPause` property. This, for instance, can allow background music to continue while the main game is paused to bring up a configuration screen.

Pause does **not** affect apapters, which are receiving audio from Unity's audio pipeline. Instead, Unity's audio pipeline needs to be paused.

The outputs of reverbs are also not paused.

3.2.5 Listener Position

```
Vector3 listenerPosition [read-only]
```

This is a short-cut to find the listener's current location.

3.3 Multiple Listeners

In principle, the underlying Rapture3D native library can support multiple listeners (for instance for split-screen play on platforms with multiple audio outputs). However, in practice this is relatively difficult to set up in Unity so use of just one listener is generally recommended.

Multiple listeners require all adapters, sources, beds and reverbs to be configured with the relevant listener *before* any use, using the respective scripts' `audioListener` property.

After this, it is up to the game to decide where to route the rendered *output* of the relevant listeners.

3.4 SteamVR

When using Rapture3D with the SteamVR Unity plugin, note that the camera "Expand" button moves the main Unity `AudioListener` to the ears, but not the `R3dAudioListener`, which may even be moved to the eyes. If you use this button, move the `R3dAudioListener` to the ears too.

3.5 Decoder List

This version of the underlying Rapture3D Universal native library contains the following decoders. This list may change between releases. A definitive list can be found at runtime using `R3dDecoder.GetDecoders()`, see above.

Please note that typically not all decoders can actually be used, depending on the host. In particular, in Unity, the host application needs to be persuaded to run using the required channel count and **currently eight channels is the limit** (configured as "7.1").

Layout	Ch	Method	URI	Un Outp
B-Format 1	4	Base	`r3dlocal:bformat1,base`	Yes
B-Format 2	9	Base	`r3dlocal:bformat2,base`	**No**
B-Format 3	16	Base	`r3dlocal:bformat3,base`	**No**
Cube	8	Basic	`r3dlocal:cube,basic`	Yes
Cube	8	Reconstruction	`r3dlocal:cube,reconstruction`	Yes
Cube	8	Tinted Reconstruction	`r3dlocal:cube,tintedreconstruction`	Yes
Headphone Stereo	2	HRTF Amber	`r3dlocal:headphonestereo,hrtfamber`	Yes
Headphone Stereo	2	HRTF Blue	`r3dlocal:headphonestereo,hrtfblue`	Yes
Headphone Stereo	2	HRTF Green	`r3dlocal:headphonestereo,hrtfgreen`	Yes
Headphone Stereo	2	HRTF Orange	`r3dlocal:headphonestereo,hrtforange`	Yes
Headphone Stereo	2	HRTF Purple	`r3dlocal:headphonestereo,hrtfpurple`	Yes
Headphone Stereo	2	HRTF Red	`r3dlocal:headphonestereo,hrtfred`	Yes
Headphone Stereo	2	HRTF Simplified	`r3dlocal:headphonestereo,hrtfsimplified`	Yes
Headphone Stereo	2	HRTF Yellow	`r3dlocal:headphonestereo,hrtfyellow`	Yes
Hexagon 1	6	Basic	`r3dlocal:hexagon1,basic`	Yes
Hexagon 1	6	Reconstruction	`r3dlocal:hexagon1,reconstruction`	Yes
Hexagon 1	6	Tinted Reconstruction	`r3dlocal:hexagon1,tintedreconstruction`	Yes
Mono	1	Base	`r3dlocal:mono,base`	Yes
N3D 01	4	Base	`r3dlocal:n3d01,base`	Yes
N3D 02	9	Base	`r3dlocal:n3d02,base`	**No**
N3D 03	16	Base	`r3dlocal:n3d03,base`	**No**
N3D 04	25	Base	`r3dlocal:n3d04,base`	**No**
N3D 05	36	Base	`r3dlocal:n3d05,base`	**No**
Octagon 1	8	Basic	`r3dlocal:octagon1,basic`	Yes
Octagon 1	8	Reconstruction	`r3dlocal:octagon1,reconstruction`	Yes
Octagon 1	8	Tinted Reconstruction	`r3dlocal:octagon1,tintedreconstruction`	Yes
Quad	4	Basic	`r3dlocal:quad,basic`	Yes
Quad	4	Reconstruction	`r3dlocal:quad,reconstruction`	Yes
Quad	4		`r3dlocal:quad,tintedreconstruction`	Yes

Layout	Ch	Method	URI	Un Outp
		Tinted Reconstruction		
SN3D 01	4	Base	r3dlocal:sn3d01,base	Yes
SN3D 02	9	Base	r3dlocal:sn3d02,base	**No**
SN3D 03	16	Base	r3dlocal:sn3d03,base	**No**
SN3D 04	25	Base	r3dlocal:sn3d04,base	**No**
SN3D 05	36	Base	r3dlocal:sn3d05,base	**No**
Stereo	2	Panner	r3dlocal:stereo,panner	Yes
Surround 3D7.1 (FC)	8	Basic	r3dlocal:surround3d71fc,basic	Yes
Surround 3D7.1 (FC)	8	Reconstruction	r3dlocal:surround3d71fc,reconstruction	Yes
Surround 3D7.1 (FC)	8	Tinted Reconstruction	r3dlocal:surround3d71fc,tintedreconstruction	Yes
Surround 5.0 (FC)	5	Basic	r3dlocal:surround50fc,basic	**No**
Surround 5.0 (FC)	5	Reconstruction	r3dlocal:surround50fc,reconstruction	**No**
Surround 5.0 (FC)	5	Tinted Reconstruction	r3dlocal:surround50fc,tintedreconstruction	**No**
Surround 5.0 (ITU, FC)	5	Basic	r3dlocal:surround50itufc,basic	**No**
Surround 5.0 (ITU, FC)	5	Reconstruction	r3dlocal:surround50itufc,reconstruction	**No**
Surround 5.0 (ITU, FC)	5	Tinted Reconstruction	r3dlocal:surround50itufc,tintedreconstruction	**No**
Surround 5.1 (FC)	6	Basic	r3dlocal:surround51fc,basic	Yes
Surround 5.1 (FC)	6	Pro Logic IIz	r3dlocal:surround51fc,prologiciiz	Yes
Surround 5.1 (FC)	6	Reconstruction	r3dlocal:surround51fc,reconstruction	Yes
Surround 5.1 (FC)	6	Tinted Reconstruction	r3dlocal:surround51fc,tintedreconstruction	Yes
Surround 5.1 (ITU, FC)	6	Basic	r3dlocal:surround51itufc,basic	Yes
Surround 5.1 (ITU, FC)	6	Pro Logic IIz	r3dlocal:surround51itufc,prologiciiz	Yes
Surround 5.1 (ITU, FC)	6	Reconstruction	r3dlocal:surround51itufc,reconstruction	Yes
Surround 5.1 (ITU, FC)	6	Tinted Reconstruction	r3dlocal:surround51itufc,tintedreconstruction	Yes
Surround 5.1.2 (FC)	8	Basic	r3dlocal:surround512fc,basic	Yes

Layout	Ch	Method	URI	Un Outp
Surround 5.1.2 (FC)	8	Reconstruction	r3dlocal:surround512fc,reconstruction	Yes
Surround 5.1.2 (FC)	8	Tinted Reconstruction	r3dlocal:surround512fc,tintedreconstruction	Yes
Surround 5.1.2 (ITU, FC)	8	Basic	r3dlocal:surround512itufc,basic	Yes
Surround 5.1.2 (ITU, FC)	8	Reconstruction	r3dlocal:surround512itufc,reconstruction	Yes
Surround 5.1.2 (ITU, FC)	8	Tinted Reconstruction	r3dlocal:surround512itufc,tintedreconstruction	Yes
Surround 7.1 (FC)	8	Basic	r3dlocal:surround71fc,basic	Yes
Surround 7.1 (FC)	8	Pro Logic IIz	r3dlocal:surround71fc,prologiciiz	Yes
Surround 7.1 (FC)	8	Reconstruction	r3dlocal:surround71fc,reconstruction	Yes
Surround 7.1 (FC)	8	Tinted Reconstruction	r3dlocal:surround71fc,tintedreconstruction	Yes
Surround 7.1.2 (Dolby Atmos)	10	Basic	r3dlocal:surround712dolbyatmos,basic	**No**
Surround 7.1.2 (Dolby Atmos)	10	Reconstruction	r3dlocal:surround712dolbyatmos,reconstruction	**No**
Surround 7.1.2 (Dolby Atmos)	10	Tinted Reconstruction	r3dlocal:surround712dolbyatmos,tintedreconstruction	**No**
Surround Stereo	2	Pro Logic	r3dlocal:surroundstereo,prologic	Yes
Twisted Cube	8	Basic	r3dlocal:twistedcube,basic	Yes
Twisted Cube	8	Reconstruction	r3dlocal:twistedcube,reconstruction	Yes
Twisted Cube	8	Tinted Reconstruction	r3dlocal:twistedcube,tintedreconstruction	Yes

3.5.1 HRTF Decoder Colours

The HRTF-based decoders above are labelled with "colours". Amber is generally recommended.

Colour	Notes
Amber	The underlying data used for this is a processed form of the IRCAM LISTEN HRTF data set, available at http://recherche.ircam.fr/equipes/salles/listen/index.html.
Blue	The underlying data used for this is a processed form of the CIAIR HRTF data set by Takanori Nishino, Shoji Kajita, Kazuya Takeda and Fumitada Itakura, available at http://www.sp.m.is.nagoya-u.ac.jp/HRTF/database.html.
Green	The underlying data used for this is a processed form of the CIAIR HRTF data set by Takanori Nishino, Shoji Kajita, Kazuya Takeda and Fumitada Itakura, available at http://www.sp.m.is.nagoya-u.ac.jp/HRTF/database.html.
Orange	The underlying data used for this is a processed form of the IRCAM LISTEN HRTF data set, available at http://recherche.ircam.fr/equipes/salles/listen/index.html.
Purple	The underlying data used for this is a set of measurements of a KU100 dummy head. This is from the SADIE II data set (v1.1), produced by the University of York. The data set is available at https://www.york.ac.uk/sadie-project/index.html.
Red	The underlying data used for this is a processed form of the MIT KEMAR data set by Bill Gardner and Keith Martin, available at http://sound.media.mit.edu/KEMAR.html.
Yellow	The underlying data used for this is a processed form of the MIT KEMAR data set by Bill Gardner and Keith Martin, available at http://sound.media.mit.edu/KEMAR.html.

4 Rapture3D Audio Adapters

Audio adapters can be added to game objects and act as Unity audio filters. They take the main Unity audio on the object and send it to Rapture3D for rendering in 3D. The component script for audio adapters is `R3dAudioAdapter.cs`.

The adapter needs to be placed *above* any Unity `AudioSource` and other filters (in the Inspector). The audio fed to the adapter leaves the Unity audio pipeline as mono and is sent to Rapture3D directly.

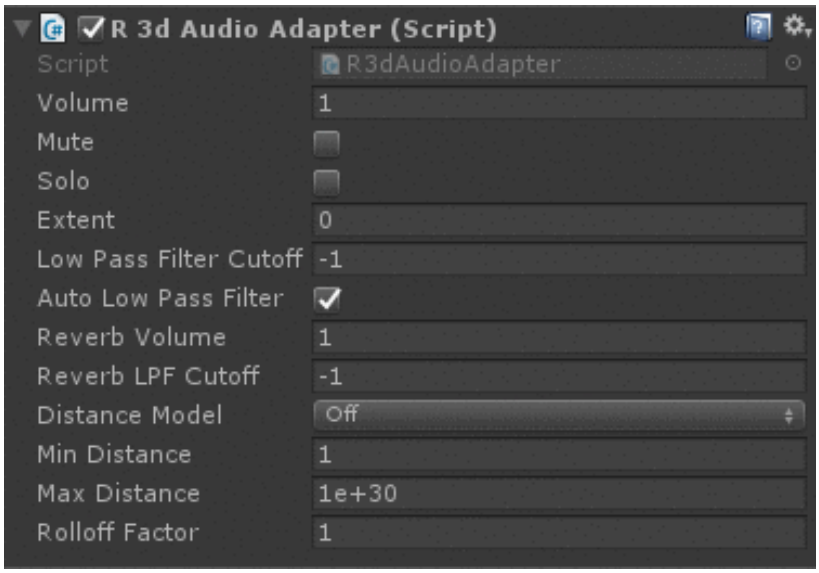
The Rapture3D Audio Adapter does not support Doppler as it cannot control the speed of the audio played to it from the Unity audio pipeline. However, if you are using a Unity `AudioSource` Doppler can be introduced there. Also, both the adapter and Unity's `AudioSource` provide distance models, so the adapter's one is off by default.

Other than Unity's Doppler and distance models, Unity `AudioSource` spatialisation parameters should not be used. In particular we recommend not using "Stereo Pan", "Spatial Blend", "Spread" or "Reverb Zone Mix". Instead, you might consider Rapture3D Audio Adapter's "Extent" and "Reverb Volume".

4.1 Unity Configuration

If you are using Rapture3D Audio Adapters, you must configure Unity's script order (see above).

4.2 Main Fields



4.2.1 Volume

```
float volume
```

This is a linear gain affecting the audio passing through the adapter.

4.2.2 Mute

```
bool mute
```

Enabling this silences the audio passing through the adapter.

4.2.3 Solo

```
bool solo
```

Enabling this silences all other Rapture3D audio except audio passing through the adapter (or other soloed objects).

4.2.4 Extent

```
float extent
```

This is the apparent size of the audio source, in game distance units. The default value is zero, indicating a narrow point source. Using larger sizes result in the impression of a larger object, particularly when up close or within the object.

If a distance model is in use, extent can increase the effective minimum distance in use.

4.2.5 Low Pass Filter Cutoff

```
float lowPassFilterCutoff
```

This is the -3dB filter cutoff of a -6dB/octave filter that is applied to audio passing through the adapter. This has various purposes, but is typically used to help simulate distance or occlusion. The filter is *not* applied to any reverb send (see "Reverb LPF Cutoff" below).

Negative values disable the low pass filter.

Note that if "auto low pass filter" is enabled, this cutoff will be set automatically.

4.2.6 Auto Low Pass Filter

```
bool autoLowPassFilter
```

If this is set, the "Low Pass Filter Cutoff" and "Reverb LPF Cutoff" fields are set automatically using the distance between the game object and the listener, and the Rolloff Factor. The setting makes distant sounds muffled, and is intended to produce a simple simulation of air filtering over distance.

Note that if this is used, other scripts that attempt to set these cutoff fields may conflict.

4.2.7 Reverb Volume

```
float reverbVolume
```

When a reverb zone is active, this linear gain affects the amount of audio sent from this adapter to the reverb. Note that this volume setting is independent of the main volume setting above.

Reverb levels can also be controlled using the reverb zone, so it is normally best to leave this set to one when reverb is wanted.

4.2.8 Reverb LPF Cutoff

```
float reverbLPFCutoff
```

When a reverb zone is active, this cutoff frequency controls a low pass filter which modifies the signal sent to the reverb.

Negative values disable the low pass filter.

Note that if "auto low pass filter" is enabled, this cutoff will be set automatically.

4.2.9 Distance Model, Min Distance, Max Distance and Rolloff Factor

```
R3dDistanceModel distanceModel
float minDistance
float maxDistance
float rolloffFactor
```

The distance model determines how the level of the sound changes as its distance from the listener changes. Behaviour of the various distance models is described in a later section.

Note that the Unity `AudioSource` also provides a distance model, which is on by default. Because of this, the adapter distance model defaults to "Off" to avoid having two distance models in use at once. You may wish to reconfigure this.

4.3 Scripting Properties

There is only one (minor) property available here. Most other behaviour will be controlled through other Unity audio components that are connected to the adapter.

4.3.1 Audio Listener

```
R3dAudioListener audioListener
```

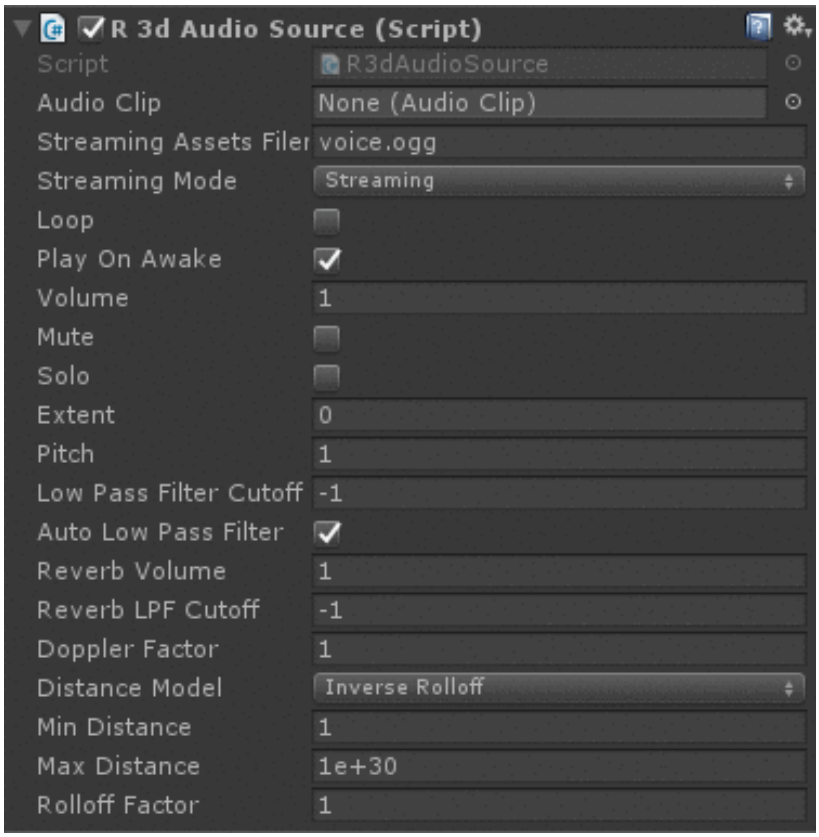
This property is used to specify which `R3dAudioListener` should be used in a multiple-listener scenario. Otherwise this normally can be ignored.

5 Rapture3D Audio Sources

The component script for audio sources is `R3dAudioSource.cs`.

This component has several similarities to the built-in Unity `AudioSource` and for *some* uses it can replace it. However, In general **it is better to use the Unity `AudioSource` and a `R3dAudioAdapter.cs` components**.

5.1 Main Fields



5.1.1 Audio Clip

```
AudioClip audioClip
```

This accepts a standard Unity audio clip for playback. However, the clip *must* be set to "decompress on load" and a copy will be loaded into memory (see above). It also must be mono (single-channel).

Only one of "Audio Clip", "Streaming Assets Filename" or the `bsrReader` property (see below) may be set.

5.1.2 Streaming Assets Filename

```
string streamingAssetsFilename
```

This is the name of a mono WAV or OGG file in your project's `Assets/StreamingAssets/` directory. Note that files placed in this folder are copied into the build outputs and may be accessible to users.

When importing such files into Unity, it is normally best to **copy the file into the streaming assets directory *outside* Unity** (e.g. in File Explorer or Finder). This stops Unity from preprocessing the file.

The filename should not include the streaming assets path, but it should include the filename suffix (e.g. ".wav" or ".ogg"). Please note that on some platform file systems the filename is case-sensitive.

Only one of "Audio Clip", "Streaming Assets Filename" or the `bsrReader` property (see below) may be set.

Audio loaded this way is managed by BSR (see above). The exact streaming mechanism is determined by the "Streaming Mode":

5.1.3 Streaming Mode

```
R3dBSRReader.StorageMode streamingMode
```

This field is only used when the "Streaming Assets Filename" is set. The following modes are available:

`Memory`	The audio is streamed and is held in memory once streaming has completed.
`Streaming`	The audio is streamed into memory and a small buffer of future audio is held in memory. This option uses less memory, but needs regular disk I/O.
`Automatic`	Depending on the size of the audio file, one of the previous options is selected. Small files use the "Memory" option and others use "Streaming". The threshold for this choice is one second on iOS and Android, and five seconds on other platforms.

The enumeration is defined in script `R3dBSRReader.cs`.

5.1.4 Loop

```
bool loop
```

If this is set, then when playback reaches the end of the sound, playback continues from the start.

5.1.5 Play On Awake

```
bool playOnAwake
```

If this is set, then when a game object with this script is woken for use, playback starts automatically. This can be useful for looped background sounds.

It is more common to trigger sounds for playback using scripting (see below).

5.1.6 Volume

```
float volume
```

This is a linear gain affecting the audio level.

5.1.7 Mute

```
bool mute
```

Enabling this silences the source's audio.

5.1.8 Solo

```
bool solo
```

Enabling this silences all other Rapture3D audio except audio produced by the source (or other soloed objects).

5.1.9 Extent

```
float extent
```

This is the apparent size of the audio source, in game distance units. The default value is zero, indicating a narrow point source. Using larger sizes result in the impression of a larger object, particularly when up close or within the object.

If a distance model is in use, extent can increase the effective minimum distance in use.

5.1.10 Low Pass Filter Cutoff

```
float lowPassFilterCutoff
```

This is the -3dB filter cutoff of a -6dB/octave filter that is applied to the source's audio. This has various purposes, but is typically used to help simulate distance or occlusion. The filter is *not* applied to any reverb send (see "Reverb LPF Cutoff" below).

Negative values disable the low pass filter.

Note that if "auto low pass filter" is enabled, this cutoff will be set automatically.

5.1.11 Auto Low Pass Filter

```
bool autoLowPassFilter
```

If this is set, the "Low Pass Filter Cutoff" and "Reverb LPF Cutoff" fields are set automatically using the distance between the game object and the listener, and the Rolloff Factor. The setting makes distant sounds muffled, and is intended to produce a simple simulation of air filtering over distance.

Note that if this is used, other scripts that attempt to set these cutoff fields may conflict.

5.1.12 Reverb Volume

```
float reverbVolume
```

When a reverb zone is active, this linear gain affects the amount of audio sent from the source to the reverb. Note that this volume setting is independent of the main volume setting above.

Reverb levels can also be controlled using the reverb zone, so it is normally best to leave this set to one when reverb is wanted.

5.1.13 Reverb LPF Cutoff

```
float reverbLPFCutoff
```

When a reverb zone is active, this cutoff frequency controls a low pass filter which modifies the signal sent to the reverb.

Negative values disable the low pass filter.

Note that if "auto low pass filter" is enabled, this cutoff will be set automatically.

5.1.14 Doppler Factor

```
float dopplerFactor
```

The Doppler Factor can be used to reduce or increase the strength of the Doppler simulation for this object. A value of zero effectively disables Doppler and a value of one is the most physically realistic

(assuming the Doppler Factor setting on the listener is also set to one).

Doppler calculations need a source velocity, as calculated by Unity. The C# scripts attempt to find this by looking for a `Rigidbody` or `CharacterController` component (in that order). See `R3dAudioBase.cs`.

5.1.15 Distance Model, Min Distance, Max Distance and Rolloff Factor

```
R3dDistanceModel distanceModel
float minDistance
float maxDistance
float rolloffFactor
```

The distance model determines how the level of the sound changes as its distance from the listener changes. Behaviour of the various distance models is described in a later section.

5.2 Scripting Properties

5.2.1 Audio Listener

```
R3dAudioListener audioListener
```

This property is used to specify which `R3dAudioListener` should be used in a multiple-listener scenario. Otherwise this normally can be ignored.

5.2.2 Is Playing

```
bool isPlaying [read-only]
```

This property indicates if the source is currently playing.

5.2.3 Time

```
float time
```

This can be used to get or set the current playback point in the current audio input. Time is measured in seconds.

5.2.4 Time Samples

```
int timeSamples
```

This can be used to get or set the current playback point in the current audio input. Time is measured in samples (at the current Unity sample rate).

5.2.5 Ignore Listener Pause

```
bool ignoreListenerPause
```

This setting means that if the listener's `pause` property is set, this sound will continue regardless.

5.2.6 BSR Reader

```
R3dBSRReader bsrReader
```

This allows a BSR stream reader to be set or got directly, without going through the `audioClip` or `streamingAssetsFilename` fields.

5.2.7 Is Audio Clip Ready

```
bool isAudioClipReady
```

This property indicates if the audio for an audio clip has already been loaded into the source.

5.3 Scripting Methods

5.3.1 Play

```
void Play()
```

This method starts a source playing, if it wasn't playing already.

5.3.2 Play Delayed

```
void PlayDelayed(float seconds)
```

This method sets the source into a 'pending' state, where sound will start at a point in the future, specified in seconds from the current time.

5.3.3 Pause

```
void Pause()
```

This method causes audio output to cease, but does not reset the playback point in the audio input, so playback can continue from that point later.

5.3.4 Stop

```
void Stop()
```

This method causes audio output to cease and resets the playback point in the audio input back to the start.

5.3.5 Play At Point

`static void PlayAtPoint(AudioClip clip, Vector3 position, float volume = 1.0f, R3dAudioListener listener = null)`
`static void PlayAtPoint(R3dBSRReader reader, Vector3 position, float volume = 1.0f, R3dAudioListener listener = null)`
`static void PlayAtPoint(string streamingAssetsFilename, Vector3 position, float volume = 1.0f, R3dAudioListener listener = null)`

These static routines create a game object, attach an `R3dAudioSource.cs` to it, drop the game object into the world at the `position` specified, and start playback. The object is destroyed automatically after playback stops.

Optional `volume` and `listener` parameters can be used to set the corresponding fields on the script.

Playback can use a Unity `AudioClip`. a `R3dBSRReader` (see BSR for details) or a streaming assets filename.

5.3.6 Play "One Shot"

`void PlayOneShot(AudioClip clip, float volume = 1.0f)`
`void PlayOneShot(R3dBSRReader reader, float volume = 1.0f)`
`void PlayOneShot(string streamingAssetsFilename, float volume = 1.0f)`

These routines attach an additional `R3dAudioSource.cs` script to the relevant game object and start playback. The script is destroyed automatically when playback stops.

An optional `volume` parameter can be used to set the corresponding field on the script. Any listener setting is copied from "this".

Playback can use a Unity `AudioClip`. a `R3dBSRReader` (see BSR for details) or a streaming assets filename.

6 Rapture3D Audio Beds

The component script for audio beds is `R3dAudioBed.cs`.

Beds allow various types of multichannel audio asset to be loaded from audio clips or (more typically) streamed using BSR.

A unique feature of this component is the ability to stream Third Order Ambisonic assets, for instance prepared with Blue Ripple Sound's O3A plugins. This allow a complete 3D audio scene to be encoded into a multichannel file.

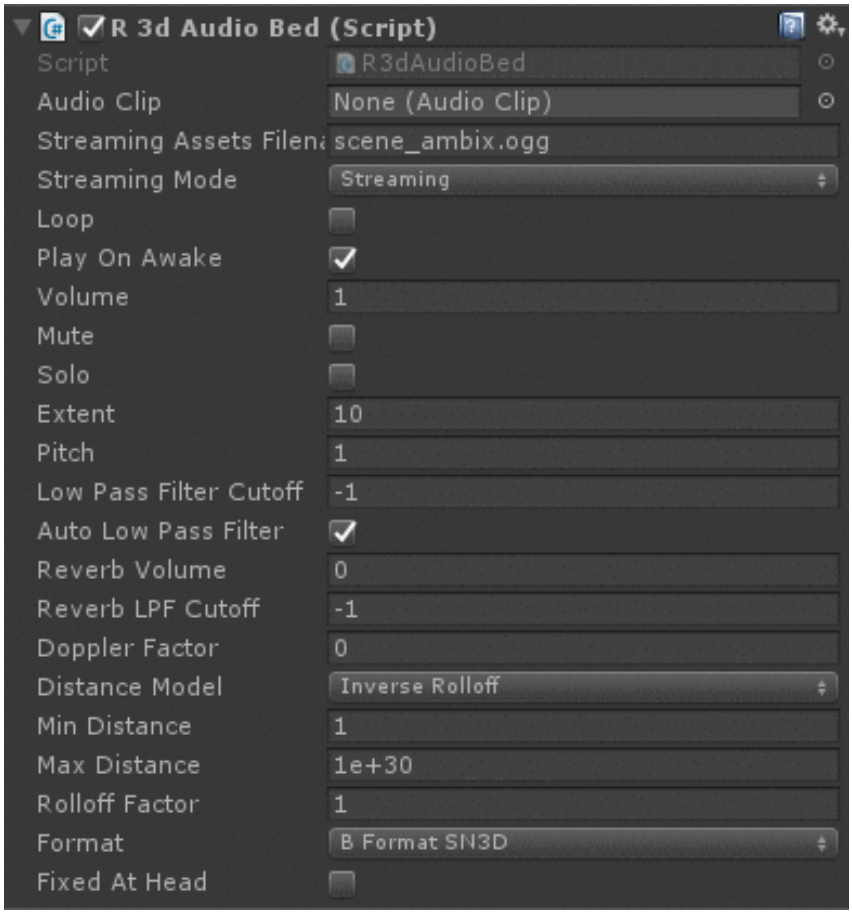
Beds fed to Rapture3D are not necessarily static around the head. With Rapture3D, you can actually place them in the overall game scene, with an extent, in a similar way to sources. You can then walk around and through them!

Because of this, please pay attention to the `Rotation` transform of the game object carrying the bed, as this will set the bed's orientation. The bed will be rotated automatically when the listener's orientation changes.

If you do not want this behaviour, you can enable "Fixed At Head".

When using this class, **the "Format" field must be set correctly**, or audio will play incorrectly or not at all.

6.1 Main Fields



6.1.1 Audio Clip

```
AudioClip audioClip
```

This accepts a standard Unity audio clip for playback. However, the clip *must* be set to "decompress on load" and a copy will be loaded into memory (see above).

Only one of "Audio Clip", "Streaming Assets Filename" or the `bsrReader` property (see below) may be set.

6.1.2 Streaming Assets Filename

```
string streamingAssetsFilename
```

This is the name of a mono WAV or OGG file in your project's `Assets/StreamingAssets/` directory. Note that files placed in this folder are copied into the build outputs and may be accessible to users.

When importing such files into Unity, it is normally best to **copy the file into the streaming assets directory *outside* Unity** (e.g. in File Explorer or Finder). This stops Unity from preprocessing the file.

The filename should not include the streaming assets path, but it should include the filename suffix (e.g. ".wav" or ".ogg"). Please note that on some platform file systems the filename is case-sensitive.

Only one of "Audio Clip", "Streaming Assets Filename" or the `bsrReader` property (see below) may be set.

Audio loaded this way is managed by BSR (see above). The exact streaming mechanism is determined by the "Streaming Mode":

6.1.3 Streaming Mode

```
R3dBSRReader.StorageMode streamingMode
```

This field is only used when the "Streaming Assets Filename" is set. The following modes are available:

`Memory`	The audio is streamed and is held in memory once streaming has completed.
`Streaming`	The audio is streamed into memory and a small buffer of future audio is held in memory. This option uses less memory, but needs regular disk I/O.
`Automatic`	Depending on the size of the audio file, one of the previous options is selected. Small files use the "Memory" option and others use "Streaming". The threshold for this choice is one second on iOS and Android, and five seconds on other platforms.

6.1.4 Loop

```
bool loop
```

If this is set, then when playback reaches the end of the sound, playback continues from the start.

6.1.5 Play On Awake

```
bool playOnAwake
```

If this is set, then when a game object with this script is woken for use, playback starts automatically. This can be useful for looped background sounds.

It is more common to trigger sounds for playback using scripting (see below).

6.1.6 Volume

```
float volume
```

This is a linear gain affecting the audio level.

6.1.7 Mute

```
bool mute
```

Enabling this silences the bed's audio.

6.1.8 Solo

```
bool solo
```

Enabling this silences all other Rapture3D audio except audio produced by the bed (or other soloed objects).

6.1.9 Extent

```
float extent
```

This is the apparent size of the audio bed, in game distance units. The default value is 10. Using larger sizes result in the impression of a larger object, particularly when up close or within the object.

If extent is set negative, it is assumed that the bed should have "infinite" extent. This can be useful to present an overall audio backdrop (like a "skybox") to a scene.

If a distance model is in use, extent can increase the effective minimum distance in use.

6.1.10 Low Pass Filter Cutoff

```
float lowPassFilterCutoff
```

This is the -3dB filter cutoff of a -6dB/octave filter that is applied to the bed's audio. This has various purposes, but is typically used to help simulate distance or occlusion. The filter is *not* applied to any reverb send (see "Reverb LPF Cutoff" below).

Negative values disable the low pass filter.

Note that if "auto low pass filter" is enabled, this cutoff will be set automatically.

6.1.11 Auto Low Pass Filter

```
bool autoLowPassFilter
```

If this is set, the "Low Pass Filter Cutoff" and "Reverb LPF Cutoff" fields are set automatically using the distance between the game object and the listener, and the Rolloff Factor. The setting makes distant sounds muffled, and is intended to produce a simple simulation of air filtering over distance.

Note that if this is used, other scripts that attempt to set these cutoff fields may conflict.

6.1.12 Reverb Volume

```
float reverbVolume
```

When a reverb zone is active, this linear gain affects the amount of audio sent from the bed to the reverb. Note that this volume setting is independent of the main volume setting above.

Reverb levels can also be controlled using the reverb zone, so it is normally best to leave this set to one when reverb is wanted. Reverb is off (zero) by default for beds.

6.1.13 Reverb LPF Cutoff

```
float reverbLPFCutoff
```

When a reverb zone is active, this cutoff frequency controls a low pass filter which modifies the signal sent to the reverb.

Negative values disable the low pass filter.

Note that if "auto low pass filter" is enabled, this cutoff will be set automatically.

6.1.14 Doppler Factor

```
float dopplerFactor
```

The Doppler Factor can be used to reduce or increase the strength of the Doppler simulation for this object. A value of zero effectively disables Doppler and a value of one is the most physically realistic

(assuming the Doppler Factor setting on the listener is also set to one).

Doppler calculations need a bed velocity, as calculated by Unity. The C# scripts attempt to find this by looking for a `Rigidbody` or `CharacterController` component (in that order). See `R3dAudioBase.cs`.

6.1.15 Distance Model, Min Distance, Max Distance and Rolloff Factor

`R3dDistanceModel distanceModel`
`float minDistance`
`float maxDistance`
`float rolloffFactor`

The distance model determines how the level of the sound changes as its distance from the listener changes. Behaviour of the various distance models is described in a later section.

6.1.16 Format

`R3dBedFormat format`

This field tells Rapture3D the type of audio that is present in the audio being played. It must be match the audio or playback will not be correct.

`Stereo`	The input is two-channel stereo audio.
`Quad`	The input is four-channel "quad" audio. The channel ordering must be Front Left, Front Right, Back Left, Back Right.
`Surround51`	The input is six-channel "5.1" audio. The channel ordering must be Front Left, Front Right, Front Centre, LFE, Side Surround Left, Side Surround Right. The LFE channel is not used.
`Surround71`	The input is eight-channel "7.1" audio. The channel ordering must be Front Left, Front Right, Front Centre, LFE, Rear Surround Left, Rear Surround Right, Side Surround Left, Side Surround Right. The LFE channel is not used.
`Cube`	The input is eight-channel audio targetting an array of eight speakers at the corners of a cube. The channel ordering must be Lower Front Left, Lower Front Right, Lower Back Right, Lower Back Left, Upper Front Left, Upper Front Right, Upper Back Right, Upper Back Left.
`BFormatFuMa`	The input uses ambisonic or higher order ambisonic (HOA) B-Format, encoded according to the FuMa channel convention (which extends classic WXYZ). The ambisonic order is inferred from the channel count, which must be 4, 9 or 16 (for first, second or third order).
`BFormatN3D`	The input uses ambisonic or higher order ambisonic (HOA) B-Format, encoded according to the N3D channel convention in ACN order. The ambisonic order is inferred from the channel count, which must be 4, 9, 16, 25 or 36 (first to fifth order).
`BFormatSN3D`	The input uses ambisonic or higher order ambisonic (HOA) B-Format, encoded according to the SN3D channel convention in ACN order. **Use this if you have third order ambisonic material prepared with Blue Ripple Sound's O3A tools.** The ambisonic order is inferred from the channel count, which must be 4, 9, 16, 25 or 36 (first to fifth order).

This enumeration is defined in script `R3dBedFormat.cs`.

6.1.17 Fixed At Head

```
bool fixedAtHead
```

If this is set, the game object's position and orientation, along with the listener's position and orientation, are entirely ignored and the sound plays relative to the head.

(Note that this is not quite the same as "Head Relative" in Rapture3D Universal's C/C++ binding.)

6.2 Scripting Properties

6.2.1 Audio Listener

```
R3dAudioListener audioListener
```

This property is used to specify which `R3dAudioListener` should be used in a multiple-listener scenario. Otherwise this normally can be ignored.

6.2.2 Is Playing

```
bool isPlaying [read-only]
```

This property indicates if the bed is currently playing.

6.2.3 Time

```
float time
```

This can be used to get or set the current playback point in the current audio input. Time is measured in seconds.

6.2.4 Time Samples

```
int timeSamples
```

This can be used to get or set the current playback point in the current audio input. Time is measured in samples (at the current Unity sample rate).

6.2.5 Ignore Listener Pause

```
bool ignoreListenerPause
```

This setting means that if the listener's `pause` property is set, this sound will continue regardless.

6.2.6 BSR Reader

```
R3dBSRReader bsrReader
```

This allows a BSR stream reader to be set or got directly, without going through the `audioClip` or `streamingAssetsFilename` fields.

6.2.7 Is Audio Clip Ready

```
bool isAudioClipReady
```

This property indicates if the audio for an audio clip has already been loaded into the source.

6.3 Scripting Methods

6.3.1 Play

```
void Play()
```

This method starts a bed playing, if it wasn't playing already.

6.3.2 Play Delayed

```
void PlayDelayed(float seconds)
```

This method sets the bed into a 'pending' state, where sound will start at a point in the future, specified in seconds from the current time.

6.3.3 Pause

```
void Pause()
```

This method causes audio output to cease, but does not reset the playback point in the audio input, so playback can continue from that point later.

6.3.4 Stop

```
void Stop()
```

This method causes audio output to cease and resets the playback point in the audio input back to the start.

6.3.5 Play At Point

`static void PlayAtPoint(AudioClip clip, R3dBedFormat format, Vector3 position, Quaternion rotation, float extent, float volume = 1.0f, R3dAudioListener listener = null)`
`static void PlayAtPoint(R3dBSRReader reader, R3dBedFormat format, Vector3 position, Quaternion rotation, float extent, float volume = 1.0f, R3dAudioListener listener = null)`
`static void PlayAtPoint(string streamingAssetsFilename, R3dBedFormat format, Vector3 position, Quaternion rotation, float extent, float volume = 1.0f, R3dAudioListener listener = null)`

These static routines create a game object, attach an `R3dAudioBed.cs` to it, drop the game object into the world at the `position` specified, and start playback of the specified asset. The game object's orientation (and thus the bed's orientation) is set to the value of the `rotation` parameter. The object is destroyed automatically after playback stops.

The `format` and `extent` parameters set the corresponding fields on the script, as may the optional `volume` and `listener` parameters.

Playback can use a Unity `AudioClip`. a `R3dBSRReader` (see BSR for details) or a streaming assets filename.

6.3.6 Play At Head

`static void PlayAtHead(AudioClip clip, R3dBedFormat format, float volume = 1.0f, R3dAudioListener listener = null)`
`static void PlayAtHead(R3dBSRReader reader, R3dBedFormat format, float volume = 1.0f, R3dAudioListener listener = null)`
`static void PlayAtHead(string streamingAssetsFilename, R3dBedFormat format, float volume = 1.0f, R3dAudioListener listener = null)`

These static routines create a game object, attach an `R3dAudioBed.cs` to it and start playback of the specified asset. The bed is set to play back with `fixedAtHead` set, so positions and orientations become irrelevant. The object is destroyed automatically after playback stops.

The `format` parameter sets the corresponding field on the script, as may the optional `volume` and `listener` parameters.

Playback can use a Unity `AudioClip`. a `R3dBSRReader` (see BSR for details) or a streaming assets filename.

6.3.7 Play "One Shot"

`void PlayOneShot(AudioClip clip, R3dBedFormat format, float extent, float volume = 1.0f)`
`void PlayOneShot(R3dBSRReader reader, R3dBedFormat format, float extent, float volume = 1.0f)`
`void PlayOneShot(string streamingAssetsFilename, R3dBedFormat format, float extent, float volume = 1.0f)`

These routines attach an additional `R3dAudioBed.cs` script to the relevant game object and start playback. The script is destroyed automatically when playback stops.

The `format` and `extent` parameters set the corresponding fields on the script, as may the optional `volume`. Any listener setting is copied from "this".

Playback can use a Unity `AudioClip` or a `R3dBSRReader` (see BSR for details).

7 Rapture3D Reverb

The component script for reverb zones is `R3dAudioReverbZone.cs`.

Reverb zones allow a single Rapture3D algorithmic reverb to change settings as the listener wanders around a scene, for instance as the listener passes from an outdoor scene into a building.

The zone management occurs entirely in the C# layer rather than the underlying native plugin. Like the other C# scripts you might even consider changing it to suit your game better, although this is definitely not supported and you may lose your changes if you import Rapture3D again!

7.1 A Basic Reverb Zone

A simple way to make a reverb zone work is to create an empty game object ("Create Empty" from the "GameObject" menu), attach a `R3dAudioReverbZone.cs` script to it, and place it somewhere in the game world. You will see two blue spheres around the same central point; the inner one is determined by the zone's "Min Distance" field and the outer by the "Max Distance".

Within the inner sphere, the reverb will be all around the listener at a fixed level for audio also in the inner sphere. When the listener leaves the inner sphere, the reverb level drops until the outer sphere, where it is silent. Also, when the listener is within this outer region, the reverb can be configured to become more directional, appearing to be coming from the inner zone.

When audio objects leave the inner sphere, they also drop in level until the outer sphere where they drop to the level given by the "External Sound Volume". Other settings (such as "Reverb Decay Time") change the character of the reverb when the listener is in the zone.

Note that each audio adapter, source and bed has a "Reverb Volume" which influences how much sound is sent to the reverb. It is generally best to leave these set to one where reverb is wanted and to control reverb level using reverb zones. A "Reverb LPF Cutoff" is also available.

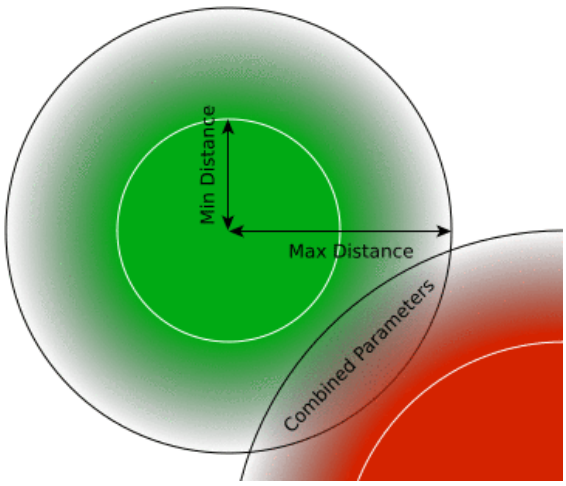
7.2 Interaction Between Reverb Zones

Multiple zones may be present in the same scene, in which case the reverb is reconfigured dynamically.

If reverb zones do not overlap, the reverb will simply use the settings for the zone the listener is currently in.

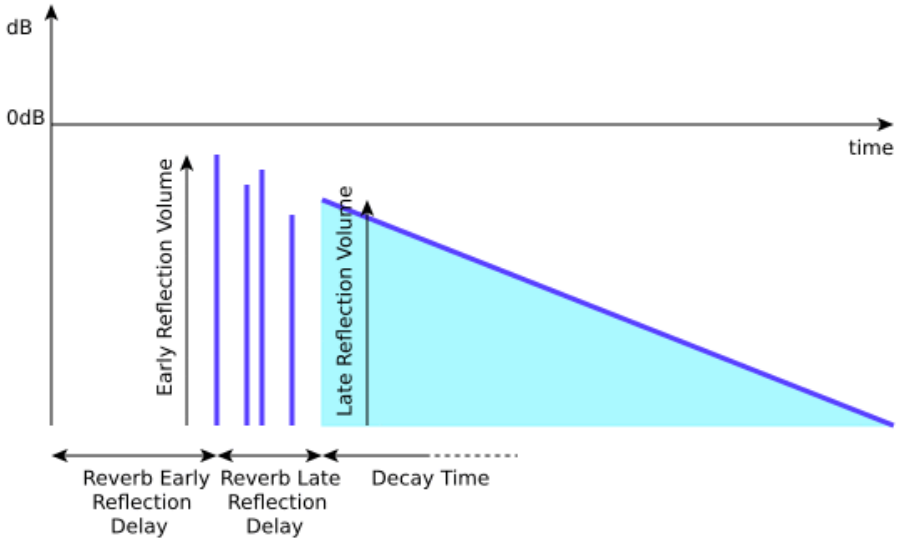
When reverb zones overlap, parameters are combined, using weightings based on how far into each zone the listener is and each zone's "Zone Weight" parameter. Tuning this parameter can be something of an art, but can lead to natural-sounding reverb transitions.

This means, for instance, you can have a low-weighted reverb set for an entire scene, but high-weighted local reverbs in particular spaces, overlapping where necessary.



7.3 Reverb Algorithm

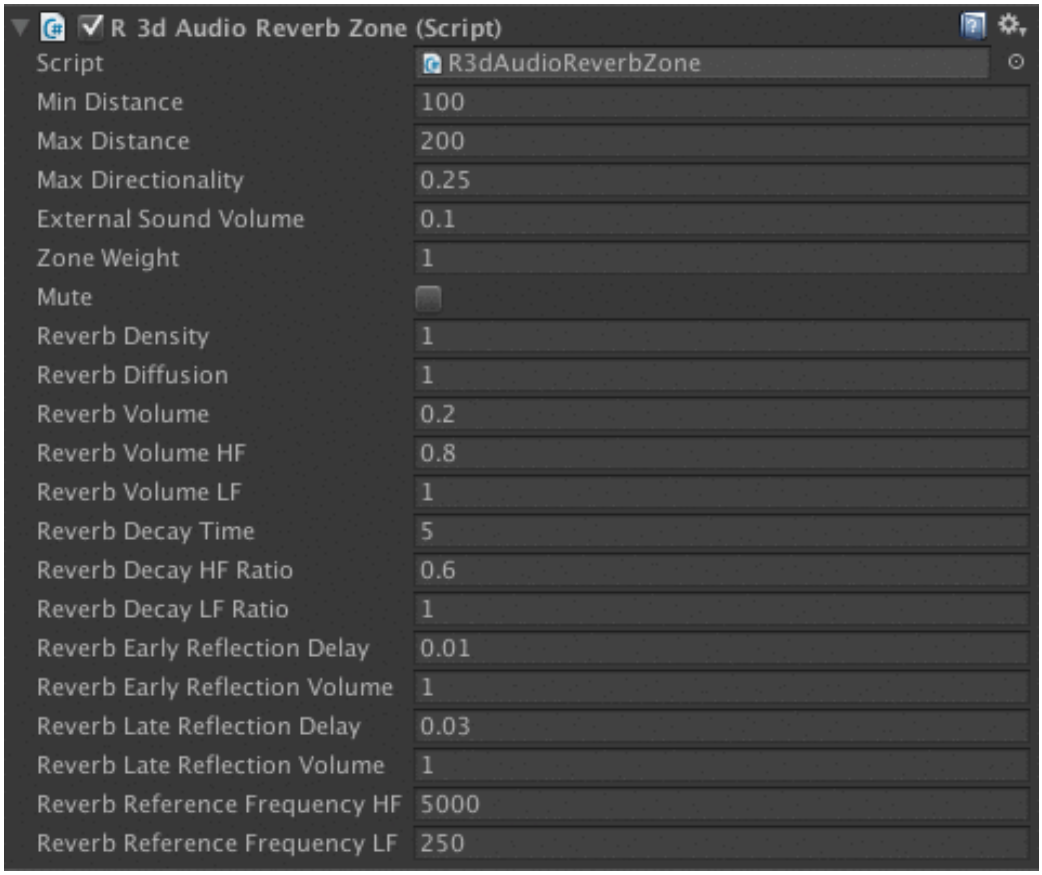
The algorithm is at heart a Moorer-Schroeder algorithmic reverb in style, although heavily modified to produce 3D output. It uses a set of discrete "early reflections" followed by a denser "late reflection" tail. The timing and levels of these phases is controllable.



The actual parameters controlling the reverb algorithm that are combined are those listed below, with names starting "Reverb".

This type of reverb can sound quite "ringy" if configured badly. It is generally a good idea to set the "Reverb Density" and "Reverb Diffusion" to high settings. Also, to simulate natural decays, do not be afraid to set high frequency gains to low values. This applies to both the overall "Reverb Volume HF" parameter and the "Reverb Decay HF Ratio".

7.4 Main Fields



7.4.1 Min Distance

This is the distance from the game object's centre defining the "inner" reverb region.

7.4.2 Max Distance

This is the distance from the game object's centre defining the "outer" reverb region.

7.4.3 Max Directionality

When the listener is outside the inner region, but inside the outer region, the reverb can become more directional as the listener moves further out. Values are between zero and one. Zero means no directionality, and one means full directionality.

7.4.4 External Sound Volume

When audio leaves the reverb zone that the listener is in, this parameter controls the amount of sound still sent to the reverb. A value of zero means the audio will fall silent when it is outside, whereas a value of one means that the sound will still be heard.

7.4.5 Zone Weight

When zones overlap, settings are combined by weighting and summing the zone reverb settings. The weights are given by this setting, modified by how far the listener is into each zone.

7.4.6 Mute

This can be used to switch off a particular reverb zone.

7.4.7 Reverb Density

This, and the following settings, are combined when zones overlap.

The "Reverb Density" is between zero and one and controls the density of simulated late reflections produced by the algorithm. Higher density values produce a larger number of reflections per second. Lower values can make a room sound more "ringy".

Be careful with this control, as low density values can sound unnatural.

7.4.8 Reverb Diffusion

The "Reverb Diffusion" is between zero and one and controls the diffuseness of the late reflections. Higher diffusion values introduce allpass filters which result in reflections with softer transients. Lower values can increase reverb "flutter".

Be careful with this control, as low diffusion values can sound unnatural.

7.4.9 Reverb Volume

This linear gain controls the overall volume level of the reverb. Values must be between 0 and 10.

7.4.10 Reverb Volume HF

This linear gain modifies the overall volume level of the reverb at high frequencies by use of a simple low-pass filter. The reference frequency can be set below. Values must be between 0.25 and 1.

7.4.11 Reverb Volume LF

This linear gain modifies the overall volume level of the reverb at low frequencies by use of a simple high-pass filter. The reference frequency can be set below. Values must be between 0.25 and 1.

7.4.12 Reverb Decay Time

This determines how long it takes the reverb to drop roughly -60dB in level. It is measured in seconds and must be between 0 and 20.

7.4.13 Reverb Decay HF Ratio

This is a modifier to the reverb decay time at high frequencies. The reference frequency can be set below. Values must be between 0.1 and 2.

Many real-world acoustics decay extremely quickly at high frequencies, so don't be afraid to set this to a low value.

7.4.14 Reverb Decay LF Ratio

This is a modifier to the reverb decay time at low frequencies. The reference frequency can be set below. Values must be between 0.1 and 2.

7.4.15 Reverb Early Reflection Delay

The reverb algorithm used has a directionally-dependent early reflection module that generates the first few acoustic reflections that reach the listener.

This delay is measured in seconds and determines how long after the initial sound event the first early reflection is heard. Values are between 0 and 0.3.

7.4.16 Reverb Early Reflection Volume

This is a linear gain that changes the level of the early reflections. Values are between 0 and 10.

7.4.17 Reverb Late Reflection Delay

The early reflections are spread out over a short period of time before a separate late reflection algorithm takes over.

This delay is measured in seconds and is the interval between the early reflections starting and the late reflections starting. Values are between 0 and 0.1.

7.4.18 Reverb Late Reflection Volume

This is a linear gain that changes the level of the late reflections. Values are between 0 and 10.

7.4.19 Reverb Reference Frequency HF

This reference frequency is measured in Hertz and is used in high frequency (low-pass) filter calculations.

7.4.20 Reverb Reference Frequency LF

This reference frequency is measured in Hertz and is used in low frequency (high-pass) filter calculations.

8 Distance Models

The `R3dAudioAdapter.cs`, `R3dAudioSource.cs` and `R3dAudioBed.cs` component scripts all provide distance models, with four parameters.

Distance models controls how the volume level of a sound decreases as the sound gets further away. This is a useful cue to help give an impression of distance. Other useful distance cues are:

- Low pass filtering, for instance when automatic low pass filtering is enabled.
- The relative levels of the main (direct sound) and reverb volume levels. This is important because more distant sounds tend to be more reverberant.

Unity also provides distance models with Unity Audio Sources, so when using a Rapture3D audio adapter it generally makes sense to use either the Unity distance model or the Rapture3D one, but not both. If both are used, their distance model effects will combine, which can be confusing.

The four parameters used to describe the distance model follow. These parameters are available with Rapture3D sources, beds and adapters.

8.1 Distance Model

This selects the overall distance model in use for the sound.

Off	No distance model is used, so volume is not affected by distance. This is the default for Rapture3D audio adapters, because usually the Unity distance model will also be active.
Inverse Rolloff	Volume essentially follows the inverse square law when the rolloff factor is one. Changing the rolloff factor reduces or increases the effect, effectively by scaling the distance.
Linear Rolloff	This model reduces the volume linearly with distance, reaching silence at max distance when the rolloff factor is one. Changing the rolloff factor reduces or increases the effect. This is not physically realistic, but can be useful when constructing crossfades.
Logarithmic Rolloff	Volume essentially follows the inverse square law when the rolloff factor is one. Changing the rolloff factor reduces or increases the effect, effectively by scaling the gain change in decibels.

For scripting, this enumeration is defined in `R3dDistanceModel.cs`.

8.2 Min Distance

This controls the minimum distance value that is used for calculation. It effectively overrides distances that are less. This is particularly useful with the Inverse or Logarithmic Rolloff models as these otherwise become extremely loud when sounds are very close.

Note that the "extent" of adapters, sources or beds also have this effect, so the level does not change if the listener is inside the extent.

8.3 Max Distance

This controls the maximum distance value that is used for calculation. It effectively overrides distances that are more. Although not physically realistic, with the Inverse and Logarithmic models, this can be useful to stop important sounds that are extremely distant from fading to nothing.

With the Linear Rolloff model (only), the max distance is the distance at which the sound fades to silence when the rolloff factor is one. This can be useful for crossfades.

Please note that this maximum distance is the distance beyond which the volume stops changing. **In general, it is *not* the distance at which the sound becomes silent.** The exception here is Linear Rolloff with the rolloff factor set to one; this can be useful for crossfades.

8.4 Rolloff Factor

The rolloff factor by default is one. Changing this increases or reduces the effect of the distance model (see above). It also affects any automatic low pass filter calculation.

9 BSR

As well as the main Rapture3D rendering plugin, the Unity package also includes a second native plugin called BSR. This performs buffered streamed reading of audio from disk.

BSR provides an audio reading service to the C# code. It attempts to "pre-buffer" audio that is going to be needed by sources or beds by reading it off disk (or other storage) before it is needed. It is used when the `StreamingAssetsFilename` is set on a source or bed, or a `R3dBSRReader` object is set explicitly. BSR supports common forms of WAV and OGG Vorbis audio file.

Audio assets to be read with BSR *must* be placed in the Unity's special `Assets/StreamingAssets/` folder. Audio placed here will be copied as a file into the build outputs. This means that **individual assets may be accessible to users** which is not always a good idea.

Most of the routines below allow assets to be specified by filename, in which case BSR use happens "behind the scenes".

For more direct control, you can create your own `R3dBSRReader` objects to manage input from a file. These have the following properties and methods, which are defined in script `R3dBSRReader.cs`.

If some `R3dBSRReader` objects are stored in memory, it is generally best to create them through the R3dBSRReaderCache (more below).

9.1 BSR Reader Scripting Properties

9.1.1 Channel Count

```
int channelCount [read-only]
```

The number of files in the file being read.

9.1.2 Has BSR Length

```
bool hasBSRLength [read-only]
```

Finding the length of a file can be very expensive in CPU and I/O terms. If this property is set to true, the length is already known. If it is false, then the next call to `length` is potentially expensive.

9.1.3 Length

```
ulong length [read-only]
```

Find the length of the file, in samples. Note that for some file types this is best avoided as it can be a very expensive operation because the entire file may need to be decompressed!

9.1.4 Sample Rate

```
float sampleRate [read-only]
```

The sample rate of the file, in Hertz.

9.1.5 Storage Mode

```
StorageMode storageMode [read-only]
```

Storage mode will be one of the following values:

`Memory`	The audio is streamed and is held in memory once streaming has completed.
`Streaming`	The audio is streamed into memory and a small buffer of future audio is held in memory. This option obviously uses less memory, but needs regular disk I/O.
`Automatic`	Depending on the size of the audio file, one of the previous options is selected. Small files use the "Memory" option and others use "Streaming". The threshold for this choice is one second on iOS and Android, and five seconds on other platforms.

9.1.6 Filename

```
string filename [read-only]
```

The filename used to find the file. Please note this will be within the `Assets/StreamingAssets/` folder.

9.1.7 Last Use

```
System.DateTime lastUse [read-only]
```

The time this reader was last read. This is used by the BSR cache mechanism (more below).

9.1.8 Load Queue Length

```
int loadQueueLength [read-only]
```

The buffering state of the reader, measured in blocks, between 0 (fully buffered) and BLOCK_COUNT (see the section on tuning). Higher values indicate risk of starvation and audio dropout.

9.2 BSR Reader Scripting Methods

9.2.1 Constructor

```
R3dBSRReader(string filename, StorageMode storageMode)
```

This creates an `R3dBSRReader` that will read from the specified file. The `storageMode` parameter dictates how memory will be managed (see the `storageMode` property above).

If some `R3dBSRReader` objects are stored in memory, it is generally best to create them through the R3dBSRReaderCache (more below).

9.2.2 Read

```
ulong Read(System.IntPtr target, ulong point, ulong length)
```

This call is not intended for general use and should not be called with Rapture3D. However, if Rapture3D is *not* in use, this method allows access to the actual samples in the audio stream (this is how the Rapture3D source and bed objects work).

This call writes 32bit floating point audio **to the memory address specified by `target`**. Be careful doing this, because making a mistake will probably crash your program.

Audio is read from `point` samples into the file. Note that if the audio is not in memory and you (or another reader) do not access the samples in strict order, the underlying BSR library will probably need to skip within the file, which may cause awful I/O performance.

The number of samples read is `length` samples in time, so the total number of 32bit floats read is this multiplied by `channelCount`. Thus the total amount of memory overwritten is `length*4*channelCount` bytes.

9.2.3 BSR Reader Cache Methods

The `R3dBSRReaderCache` class can be used to manage readers that are stored in memory. When audio is asked for that is already being managed by a reader, the pre-existing reader can be returned, which saves both the loading time for the audio and memory storage.

Cached data is stored statically, so no cache object need be created explicitly.

9.2.3.1 Get Reader

```
static R3dBSRReader GetReader(string filename, R3dBSRReader.StorageMode
storageMode)
```

Create a reader, or grab one from the cache if a suitable one is present. Items are only cached if they are stored in memory, as other items are streamed and may be at different points in the same underlying file.

If you are using the cache at all, you should call the `Trim()` routine from time to time.

9.2.3.2 Trim

```
static void Trim(float seconds)
```

This will remove unused items from the cache. Items in the cache are considered in use (and are kept) if:

1. The reader is currently in use in any other scripts *OR*

2. The reader has been created or read from recently (the time threshold for this is determined by the `seconds` parameter).

9.2.3.3 Trim Heavy

```
static void TrimHeavy()
```

This routine is intended for use when the game state changes in a way that means the cache's content is probably not useful. It is similar to calling `Trim(0)`, which causes any readers not in current use in other scripts to be discarded.

It also goes further and discards the "recent use" history for all readers, which makes them more likely to be discarded in subsequent calls to `Trim()` until they are demonstrated to be in use through a new read.

9.3 Supported Files

The BSR readers have support for conventional PCM WAV files (`.wav`). These are not recommended on mobile platforms.

Single-stream Ogg Vorbis files (`.ogg` or `.oga`) are also supported.

Note that both of these formats can manage large numbers of channels. Of particular interest are sixteen-channel files carrying complete 3D audio scenes encoded using third order ambisonics (O3A). These can be produced using a digital audio workstation such as Reaper and Blue Ripple Sound's O3A plugins.

9.4 Tuning Buffer Settings

Buffering is controlled by settings near the end of the C# file `R3dBSRReader.cs`:

```
private const int BLOCK_SIZE = 8192;
private const int BLOCK_COUNT = 8;
```

These settings determine how much audio BSR will attempt to read into memory ahead of playback, measured as block size (in samples) and the number of blocks. At 48kHz, the settings above will actually buffer more than a second of audio. Depending on your application, this may be quite a high setting and you may wish to reduce it. For some applications, however, it is not, and you may even wish to increase it (generally by increasing the `BLOCK_COUNT`) if you encounter stuttering.

More buffering is typically needed when there is contention on the file system, for instance when a high resolution VR360 video is also being read. Also, specifically on Android, Unity assets are stored in a Jar file, which is a compressed format which needs to be navigated. On top of this, audio will typically be compressed with Ogg/Vorbis, so two levels of software decompression are running in the BSR thread.

The BSR thread runs with a normal thread priority, so be careful not to starve it by running higher priority threads.