Android Native Audio

Android Native Audio (ANA) is an asset for Unity 4.3.4 - 5.x on Android. It provides easy access to the native Android audio system for low-latency audio playback.

Note that audio latency can have many contributing factors, including hardware and Android itself. ANA only removes most of the latency generated by Unity. This will be different on every device. On some devices the reduction in latency will be large, on others it will be smaller.

Thank you!

These creations are a labor of love for me. It's very rewarding to know that what I make goes out into the world and helps people. Please feel free to contact me if you need any assistance or have any feedback. Thank you for your purchase, and I hope you enjoy using Android Native Audio!

Christopher

If you like Android Native Audio, please tell everyone!

Leave a review on the Asset Store.

If you don't like Android Native Audio, please tell me!

Let me know how I can help: support@ChristopherCreates.com

AndroidNativeAudio & ANAMusic

ANA version 2.0 brings you a whole new feature: ANA Music. Where the `AndroidNativeAudio` class is for short clips like sound effects, the `ANAMusic` class is for long tracks like music. See `Assets\Android Native Audio\ANA Music.pdf` for details on how to use it.

Examples

There are two included example scenes to show you how ANA works and let you quickly test the difference in latency between Unity and ANA. They are located at `Assets\Android Native Audio\Examples`.

Split Application Binary (OBB)

ANA supports split application binaries (APK + OBB files), and will automatically find and load your audio files in either location.

PlayMaker Support

ANA includes a full set of PlayMaker actions and two example scenes. These are located in `Assets\Android Native Audio\PlayMaker.zip`. Just unzip that file anywhere in your `Assets` folder to use them. For more information on PlayMaker and using actions, see the official website.

http://hutonggames.com/

Debug Logging

At the top of `AndroidNativeAudio.cs` you will find a `DEBUG` variable. Set it to `true` to enable logging of ANA activity. You can monitor Unity logging on Android by running "`<Android SDK Path>\platform-tools\adb logcat -s Unity`". If you are on Windows and your Android SDK is installed to the default location, you can double-click `Assets\Android Native Audio\Windows Android Logcat.cmd` to automatically open the log console.

Understanding Android Audio

ANA is simple to use, but it works in ways that are different from Unity's audio system. There are three core concepts to know:

Pool: The pool is the central Android resource that manages everything else. (It is a direct call to `android.media.SoundPool` in Java.)
File: A file is an individual audio file loaded into memory. A file is only loaded, played, and unloaded.
Stream: A stream is an audio channel currently playing a file. All other operations such as pause and resume are performed on streams.

The life cycle of ANA works like this:

1. Make the pool
2. Load a file
3. Play the file
4. Optionally modify the stream (pause, volume, stop, etc)
5. Unload the file
6. Release the pool

Android Audio Files

Audio files for ANA must be placed in `Assets/StreamingAssets`. This is a special Unity folder that makes raw files available directly in the APK (or OBB). You can create sub-folders under `StreamingAssets` to organize your files. (You can also place files in `Application.persistentDataPath` at run time, see the `load` method below.)

All audio files for ANA must be in an Android-compatible format. See the official documentation for details:
http://developer.android.com/guide/appendix/media-formats.html#core

For best performance, I suggest mono WAV (PCM) files at 16 bits and 44,100 Hz.

Examples

When the scene loads:

```
AndroidNativeAudio.makePool(1);
int fileID = AndroidNativeAudio.load("Effect.wav");
```

Later, play the file:

```
int streamID = AndroidNativeAudio.play(fileID);
```

Other operations are very intuitive:

```
AndroidNativeAudio.pause(streamID);
AndroidNativeAudio.resume(streamID);
AndroidNativeAudio.setVolume(streamID, 0.5f);
```

When you're done, you should unload the file(s) and release the pool:

```
AndroidNativeAudio.unload(fileID);
AndroidNativeAudio.releasePool();
```

It's that easy!

Tips

- `AndroidNativeAudio` is intended for short, small sound effects. Use `ANAMusic` (or Unity's normal audio) for longer tracks such as background music.
- Make sure your audio files are in the right place and the right format (see above).
- Remember that most Android devices don't have much memory. Be mindful of how many files you load and streams you play at once. And be sure to unload unneeded files and release the pool when you're done.
- Loading a file takes some amount of time. Be sure to load them ahead of when you need to play them. Use a callback method if you need to be sure it's finished (see the `load` method below).

ANA & Non-Android Environments

Editor

ANA can't play while in the editor (because it's not Android). Instead, it logs to the console window to let you know what it would be doing if it were on Android. Note that return values such as `fileID` won't be correct in the log, as it needs Android to generate the real data.

Multi-Platform

If you want to use the same code for both Android and non-Android platforms, you can make calls like this.

```
#if UNITY_ANDROID && !UNITY_EDITOR
      streamID = AndroidNativeAudio.play(fileID);
#else
      audioSource.Play();
#endif
```

This will use the `AndroidNativeAudio` call when on android, and the `audioSource` call everywhere else (including in the editor when set for Android).

Non-Redundant Files

In the editor, Unity won't allow you to use files in StreamingAssets as an AudioSource. So the basic strategy for multi-platform is to have two copies of you audio files, one for Android and one for everything else. This works fine, but takes up twice the space and means you have to manage two copies of your files. You can work around the problem with something like this.

```
var www = new WWW("file:" + Application.streamingAssetsPath +
      "/Native.wav");
while (!www.isDone) { }
audioSource.clip = www.GetAudioClip(false, false, AudioType.WAV);
```

This lets you load a `StreamingAssets` file into an `AudioSource` at run time. It's a bit clunky, but it will allow you to work on any platform with just one set of files.

Scripting Reference

The `AndroidNativeAudio` class is designed to closely follow the native `android.media.SoundPool`. If you'd like to know more about what's going on behind the scenes, the official documentation has all the details:
http://developer.android.com/reference/android/media/SoundPool.html

```
AndroidNativeAudio.load(string audioFile, bool usePersistentDataPath =
false, Action<int> callback = null)
```

Loads an audio file. Use `makePool` before loading.

`audioFile` - The path to the audio file, relative to `Assets\StreamingAssets`. (Unless using `usePersistentDataPath`, see below.)

`usePersistentDataPath` - Makes `audioFile` relative to `Application.persistentDataPath`.

`callback` - Method to call when load is complete. Must take one int parameter which is the loaded file ID.

Returns: `int` - The file ID if successful, -1 if the load fails.

```
AndroidNativeAudio.makePool(int maxStreams = 16)
```

Makes an Android native audio pool.

`maxStreams` - The maximum number of streams. (The maximum number of simultaneously playing files.)

```
AndroidNativeAudio.pause(int streamID)
```

Pauses a stream.

`streamID` - The ID of the stream to pause.

```
AndroidNativeAudio.pauseAll()
```

Pauses all playing streams. Call `resumeAll` to resume.

```
AndroidNativeAudio.play(int fileID, float leftVolume = 1, float rightVolume = -1, int priority = 1, int loop = 0, float rate = 1)
```

Plays a file. Use `load` before playing.

`fileID` - The ID of the file to play.

`leftVolume` - The left volume to play at (0.0 - 1.0). If `rightVolume` is omitted, this value will be used for both.

`rightVolume` - The right volume to play at (0.0 - 1.0). Defaults to `leftVolume`.

`priority` - The priority of this stream. If the number of simultaneously playing streams exceeds `maxStreams` in `makePool`, higher priority streams will play and lower priority streams will not.

`loop` - How many times to loop the audio. A value of 0 will play once, -1 will loop until stopped.

`rate` - The rate to play at. A value of 0.5 will play at half speed, 2 will play at double speed.

Returns: `int` - The stream ID if successful, -1 if the play fails.

```
AndroidNativeAudio.releasePool()
```

Releases the audio pool resources.

```
AndroidNativeAudio.resume(int streamID)
```

Resumes a paused stream.

`streamID` - The ID of the stream to resume.

```
AndroidNativeAudio.resumeAll()
```

Resumes all streams paused with `pauseAll`.

```
AndroidNativeAudio.setLoop(int streamID, int loop)
```

Sets the loop of a stream.

`streamID` - The ID of the stream to change.

`loop` - How many times to loop the audio. A value of 0 will play once, -1 will loop until stopped.

```
AndroidNativeAudio.setPriority(int streamID, int priority)
```

Sets the priority of a stream.

`streamID` - The ID of the stream to change.

`priority` - The priority of this stream. If the number of simultaneously playing streams exceeds `maxStreams` in `makePool`, higher priority streams will play and lower priority streams will not.

```
AndroidNativeAudio.setRate(int streamID, float rate)
```

Sets the rate of a stream.

`streamID` - The ID of the stream to change.

`rate` - The rate to play at. A value of 0.5 will play at half speed, 2 will play at double speed.

```
AndroidNativeAudio.setVolume(int streamID, float leftVolume, float
rightVolume = -1)
```

Sets the volume of a stream.

`streamID` - The ID of the stream to change.

`leftVolume` - The left volume to play at (0.0 - 1.0). If `rightVolume` is omitted, this value will be used for both.

`rightVolume` - The right volume to play at (0.0 - 1.0). Defaults to `leftVolume`.

```
AndroidNativeAudio.stop(int streamID)
```

Stops a stream.

`streamID` - The ID of the stream to stop.

```
AndroidNativeAudio.unload(int fileID)
```

Unloads a file from the pool. Returns true if unloaded, false if previously unloaded.

`fileID` - The ID of the file to unload.

Returns: `bool` - True if unloaded, false if previously unloaded.