Android Native Audio Music

Android Native Audio Music (ANA Music) 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

ANAMusic & AndroidNativeAudio

ANA Music is a whole new feature in Android Native Audio 2.0. 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\Android Native Audio.pdf` for details on how to use the original.

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 ANAMusic`.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 Music

ANA Music is simple to use, but it works in ways that are different from Unity's audio system. (It is a direct call to `android.media.MediaPlayer` in Java.)

The life cycle of ANA Music works like this:

1. Load a file
2. Play the file
3. Optionally modify the music (pause, volume, stop, etc)
4. Release the file

Android Music Files

Audio files for ANA Music 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 Music 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 Vorbis (OGG) files.

Play In Background

By default, ANA Music will pause all playing tracks when the application loses focus and resume them when the application regains focus. But it can also play tracks while the application is in the background. To do this, use the `playInBackground` flag on the `load` method or use the `setPlayInBackground` method on loaded music. See below for details.

Examples

When the scene loads:

```
int musicID = ANAMusic.load("Music Native.ogg");
```

Later, play the file:

```
ANAMusic.play(musicID);
```

Other operations are very intuitive:

```
ANAMusic.pause(musicID);
ANAMusic.seekTo(musicID, 10000);
ANAMusic.setVolume(musicID, 0.5f);
```

When you're done, you should release the file:

```
ANAMusic.release(musicID);
```

It's that easy!

Tips

- `ANAMusic` is intended for longer tracks such as music. For short, small sound effects, use `AndroidNativeAudio`.
- 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 play at once. And be sure to release files 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 Music & Non-Android Environments

Editor

ANA Music 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 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
	ANAMusic.play(musicID);
#else
	audioSource.Play();
#endif
```

This will use the `ANAMusic` 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.ogg");
while (!www.isDone) { }
audioSource.clip = www.GetAudioClip(false, false,
	AudioType.OGGVORBIS);
```

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

`ANAMusic.`**`getCurrentPosition`**`(int musicID)`
Gets the current playback position.
musicID - The ID of the music to use.
Returns: `int` - The current position in milliseconds.

`ANAMusic.`**`getDuration`**`(int musicID)`
Gets the duration of the file.
`musicID` - The ID of the music to use.
Returns: `int` - The duration in milliseconds.

`ANAMusic.`**`isLooping`**`(int musicID)`
Checks whether the MediaPlayer is looping or non-looping.
`musicID` - The ID of the music to use.
Returns: `bool` - True if the MediaPlayer is currently looping, false otherwise.

`ANAMusic.`**`isPlaying`**`(int musicID)`
Checks whether the MediaPlayer is playing.
musicID - The ID of the music to use.
Returns: `bool` - True if currently playing, false otherwise.

`ANAMusic.`**`load`**`(string audioFile, bool usePersistentDataPath = false, bool loadAsync = false, Action<int> loadedCallback = null, bool playInBackground = false)`
Loads a music file into a native Android media player.
`audioFile` - The path to the music file, relative to `Assets\StreamingAssets`.
`usePersistentDataPath` - If true, audioFile is relative to `Application.persistentDataPath`. If false, it is relative to `Assets\StreamingAssets`.
`loadAsync` - If true, the file is loaded asynchronously and the method immediately returns. If false, the method will not return until the load has completed.
`loadedCallback` - If given, the method to call when the load is complete. Must take one int parameter which is the loaded music ID.
`playInBackground` - If true, the music will continue playing when the game is not active. If false, the music will be paused when the game is not active and resumed when it becomes active again.
Returns: `int` - The ID of the loaded music.

`ANAMusic.`**`OnApplicationPause`**`(bool isPaused)`
Used for automatically handling music when the game is paused or resumed. You shouldn't need to use this manually.
`isPaused` - Whether or not the application is currently paused.

`ANAMusic.`**`pause`**`(int musicID)`
Pauses playback. Call `play()` to resume.
`musicID` - The ID of the music to use.

`ANAMusic.`**`pauseAll`**`()`
Pauses all currently playing music. Use `resumeAll()` to resume only music that was paused with `pauseAll()`.

`ANAMusic.`**`play`**`(int musicID, Action<int> completionCallback = null)`
Starts or resumes playback.
`musicID` - The ID of the music to use.
`completionCallback` - If given, the method to call when playback is complete.

`ANAMusic.`**`release`**`(int musicID)`
Releases resources associated with this MediaPlayer object.
`musicID` - The ID of the music to use.

`ANAMusic.`**`resumeAll`**`()`
Resumes play of all music paused with pauseAll().

`ANAMusic.`**`seekTo`**`(int musicID, int msec)`
Seeks to specified time position.
`musicID` - The ID of the music to use.
`msec` - the offset in milliseconds from the start to seek to

`ANAMusic.`**`setLooping`**`(int musicID, bool looping)`
Sets the player to be looping or non-looping.
`musicID` - The ID of the music to use.
`looping` - whether to loop or not

`ANAMusic.`**`setPlayInBackground`**`(int musicID, bool playInBackground)`
Sets whether or not the music will play when the game is inactive.
`musicID` - The ID of the music to use.
`playInBackground` - If true, the music will continue playing when the game is not active. If false, the music will be paused when the game is not active and resumed when it becomes active again.

`ANAMusic.`**`setVolume`**`(int musicID, float leftVolume, float rightVolume = -1)`
Sets the volume on this player.
`musicID` - The ID of the music to use.
`leftVolume` - left volume scalar (0.0 to 1.0) If `rightVolume` is omitted, this value will be used for both.
`rightVolume` - right volume scalar (0.0 to 1.0) Defaults to `leftVolume`.