MusicManager Manual

Tutorial and Reference

Add music to games or make an MP3 player in Unity

Table of Contents

1 Basic Usage: Wiring up an MP3 player

The simplest way to use the MusicManager script is to wire up an MP3 player. The asset collection contains a demo that is an MP3 player with basic functionality. Open the TestScene in the Demo folder in Unity and run the game to test the MP3 player. Click on the various levels in the Hierarchy and look at how the script is wired in the Inspector for a sample.

To create your own, create a new scene and put an empty game object in it. A good name for the game object is "MusicManager". Add the MusicManager.cs script to it as a component. Doing so should automatically add an AudioSource component. If not, just add one manually. The default AudioSource settings should be sufficient.

With the MusicManager game object selected in the Hierarchy, look at the properties in the Inspector. For an MP3 player, one should probably start with Repeat off, Auto Advance on, Recycle on, Play On Awake on, and Fade In Time set to 0.5 and Fade Out Time set to 1.0 (the last two can be adjusted to taste). Set the volume to where you want it. The Control Messages can be ignored for now, except that you might want to Log Control Messages for debugging purposes. Add whatever music files you want to the play list.

What these settings do is this: Repeat off means the clips don't loop (but you can wire it to a repeat button to allow the user to enable looping). Auto Advance on means when a clip is done, the next one plays, like with an MP3 player. (In a game, you probably want Repeat on and Auto Advance off). Recycle On means

when the end of the play list is reached, it starts again with the first song, again like most MP3 players. Play On Awake starts the first track as soon as the player is running. Fade In Time and Fade Out Time are so that when you press pause, and later press play to resume, the music fades out and in, respectively. This prevents noisy glitches. The volume setting is self-explanatory.

Then, build an MP3 player GUI however you wish, and wire the buttons up to the MusicManager properties and functions. The demo project is but one way it can be done.

Some hints: in the script, there are regions for properties ("Public Inspector Attributes"), methods that control the MP3 player ("Public Controls") and for finding information about the MusicManager object ("Public Queries"). All inspector attributes can be modified by public control methods that take no arguments, so they can easily be wired to buttons. The query methods can then be used to put information on the "LCD Screen" of the MP3 player.

The control methods used in the demo are as follows. You may prefer to use other control methods instead: PlayPauseToggle, ToggleRepeat, VolumeUp, VolumeDown, ToggleMute, Rewind, StopClip, Next, Previous, and Shuffle. Other methods that can be used in addition to or in place of these include MoveForward, MoveBackward, RewindClip, Play, and Pause. VolumeUp, VolumeDown, MoveForward, and MoveBackward can be called without arguments, or with a floating-point argument for the amount. In addition, other methods, some of which take arguments, are used to set Inspector attributes. These are found in the "Public Scripted Setup" region: SetFadeIn, SetFadeOut, SetVolume, AutoAdvance, NoAutoAdvance, Repeat, NoRepeat, Recycle, and NoRecycle.

Alternatively, the Inspector attributes already mentioned can be modified directly by buttons, sliders, and other GUI elements.

2 Intermediate Usage: Adding music to a game

When writing a game, you would write scripts that need to change the music clip or start and stop. The MusicManager class is a Singleton class by default (uncheck "Dont Destroy on Load" to change this, but then one cannot have music continue through scene changes without stopping), and thus it must be instantiated exactly once in a game. It persists on scene changes. Thus, it is recommended one writes a scene called "_Start" or "_Init" or similar that initializes the MusicManager, does other initializiation, and perhaps puts up a

splash screen, then immediately changes to the first "real" scene of the game and is never returned to again. One would put a GameObject in the initial scene, perhaps called "MusicManager" and attach the MusicManager.cs script to it, and verify that the required AudioSource is automatically added. In summary:

- Make a scene called, say, "_Init" that contains a GameObject called "MusicManager" and add the MusicManager.cs script to it as a component. Give it another script on a different component that changes to the first real scene (say, "TitleScreen" or "Level1").

Next, while in this initial scene, select the MusicManager object in the Hierarchy and look at it in the Inspector. Somewhat different defaults will be needed for gameplay as opposed to an MP3 player. Most likely, you want Repeat on, AutoAdvance off, Recycle doesn't matter, Play On Awake off unless you have Splash Screen music, but the fade controls should have similar values. The volume should be set so as not to overpower game sounds. Also, put your game music in the Play List (or in a Resources folder and check Load From Resources in the Inspector—a hint is putting track numbers before clip names, like "01 Clip One Song" causes the alphabetization to put the clips in the right order in the playlist).

- Set Repeat on, Auto Advance off, Play On Awake off. Fill the play list with game music clips.

Now, you need to arrange the music to change on each level. Thus, to each level scene, add a MonoBehaviour script component to a game object in the scene. The "Start()" method should then get the instance of MusicManager, and then set the appropriate clip. It is helpful to test if the same clip is already playing, and just let it keep going if so.

- In a script on a game object in each level, the Start method should execute

```
if(!MusicManager.instance.IsPlaying() || indexOfClipForThisScene !=
MusicManager.instance.CurrentTrackNumber()){
   MusicManager.instance.Pause(); //Stop any currently-playing clip
   MusicManager.instance.SetTrack(indexOfClipForThisScene);
   MusicManager.instance.Play();
}
```

3 Reference

3.1 Public Inspector Attributes

- **Play List**: List of clips to play. Instead of setting them by hand in the inspector, they can be loaded automatically from a Resources folder using the LoadClipsFromResources method in the "Public Scripted Setup" region.
- **Repeat**: Loop the currently-playing clip until told otherwise.
- **Auto Advance**: Automatically go to next clip when current clip finishes. Does not work if Repeat is on.
- **Recycle**: When end of play list is reached, go back to first clip.
- **Play On Awake**: Begin playing first clip as soon as the Music Manager starts.
- **Fade In Time**: If nonzero, take this many seconds to fade into the track on resume from pause or changing track from middle of song.
- **Fade Out Time**: If nonzero, take this many seconds to fade out the track on pause or changing track from middle of song.
- **Volume**: Adjust the volume of the clip.
- **Load From Resources**: Append clips from resources directories into playlist. This will not duplicate clips that are already there.
- **Dont Destroy On Load**: (sic) Make the Music Manager a singleton class that persists between scene changes.

3.2 Public Scripted Setup

- **ClearPlayList()**: Remove all clips from playlist.
- **AddToPlayList(AudioClip clip)**: Add a clip to the end of the playlist.
- **RemoveFromPlayList(int index)**: Remove selected track from playlist (0 is first track).
- **LoadClipFromResources(string name)**: Load named clip from resources folder(s) into playlist.

- **LoadClipsFromResources(string path="")**: Load all clips from resources folders, from named subdirectory if given. Automatically dedupes.
- **AudioClip GetPlayListClip(int index)**: Get the audioclip at specified track in playlist (0 is first track).
- **int GetPlayListLength()**: Get the number of tracks in the playlist.
- **SetFadeIn(float time)**: Set the fade-in time when resuming from pause.
- **SetFadeOut(float time)**: Set the fade-out time when pausing.
- **SetVolume(float amount)**: Adjust the volume control, from 0 to 1 (use 1 for "turn it up to eleven").
- **AutoAdvance()**/**NoAutoAdvance()**: Set/unset auto advance.
- **Repeat()**/**NoRepeat()**: Set/unset repeat.
- **Recycle()**/**NoRecycle()**: Set/unset recycle.

3.3 Public Controls

- **PlayPauseToggle()**: Toggle between playing and pausing of current track.
- **ToggleRepeat()**:Toggle the Repeat (loop this track) flag.
- **ToggleRecycle()**: Toggle the Recycle (loop the whole playlist) flag.
- **VolumeUp(float amount = 0.05f)**: Increment the volume.
- **VolumeDown(float amount = 0.05f)**: Decrement the volume.
- **ToggleMute()**: Toggle between muting and playing at last-set volume.
- **Play()**: Start playing, or resume from pause.
- **Pause()**: Stop playing, but don't lose place in clip.
- **Rewind()**: Go back to beginning of playlist.
- **RewindClip()**: Go back to beginning of clip.
- **Stop()**: Stop playing and rewind to beginning of playlist.
- **StopClip()**: Stop playing and rewind to beginning of clip.
- **Next()**: Advance to next song on playlist.
- **Previous()**: Go back to previous song on playlist.

- **MoveForward(float seconds = 10f)**: Advance a bit in the song.
- **MoveBackward(float seconds = 10f)**: Go back a bit in the song.
- **Shuffle()**: Reorder the playlist randomly.

3.4 Public Queries

- **int CurrentTrackNumber()**: Track number playing (from 0 to number of tracks – 1).
- **AudioClip NowPlaying()**: Audio clip now playing (even if paused).
- **bool IsPlaying()**: True if there is a current clip and it is not paused.
- **bool IsPaused()**: True if there is a current clip and it is paused.
- **float TimeInSeconds()**: Time into the current clip.
- **float LengthInSeconds()**: Length of the current clip, or 0 if no clip.