Music Player
User Guide

Freakshow Studio

Contents

1 Introduction

Music Player is a scripting plugin for Unity that allows for easy playing of music in your projects.

It lets you configure playlists, for example for different parts of your game or different levels, and easily change between these at appropriate times. For example by having one playlist for the main menu, and a different one in game.

It is also possible to create playlists at runtime, if your audio clips are not available when you are editing or you are downloading the music at runtime.

Playlists can be played in order or shuffled, and tracks can be disabled. This can let users choose which tracks they want to hear.

2 Quickstart

First, a Music Player should be added to the scene. This is done with the menu option GameObject ➔ Audio ➔ Music Player. This will create a GameObject named *MusicPlayer* in the scene, with an *AudioSource* component and a *MusicPlayer* component.

It is possible to have several music players in a single scene, for example if you want to use them as positional (3D) audio sources. The most common scenario would however be to have just one, and then route the output of the *AudioSource* component to an *Audio Mixer*.

Once you have created a *Music Player*, you can set up playlists and tracks in its inspector. Set the *Play on Awake* option if you want it to automatically play the first playlist on start.

Once you have set up the playlists as you wish, use the public methods on the *Music Player* to control playback. These are *Play*, *Stop*, *Pause*, *UnPause*, *Next* and *Previous*.

3 Setting up the MusicPlayer

To function, *Music Player* needs to be added to the scene. A scene can contain more than one *Music Player*, and each instance needs its own Audio Source component. Setting up the Audio Source is left to the user as this allows setting it up as just a general music player routed directly to a mixer, or as one or more music players positioned in 3D space, or any other configuration you see fit.

3.1 Adding the MusicPlayer

To add a new *Music Player* to the current scene, select the menu option ***GameObject ➔ Audio ➔ Music Player***. This will create a new GameObject named MusicPlayer in the scene, with an Audio Source and a Music Player component.

You can also add the music player manually to a GameObject by adding the script MusicPlayer.cs to it.

3.2 Configuring the MusicPlayer

The Music Player itself only has a few options, which are *Play On Awake*, *Volume*, and *History Length*.

If *Play On Awake* is selected, the music player will automatically start playing the first playlist when the scene is loaded/started. Otherwise, you will need to call *Play* manually to start playing.

The Volume option sets the music volume, this directly controls the volume of the Audio Source, and so it is important that you don't try to adjust the volume on the Audio Source.

The *History Length* option sets the number of tracks it should be possible to skip backwards. This is stored internally as a list of integers, and therefore can be reasonably large without affecting memory usage.

See the chapter Playlists and tracks for information on how to set up playlists.

3.3 Configuring the Audio Source

When adding a new Music Player through the Game Object menu, the Audio Source will be configured automatically for use as a general music player intended to be routed directly to an audio mixer.

The Music Player does however support any configuration of the audio source you'd like; the only thing to keep in mind is that it should not be set to loop. If it is set to loop, it will never continue to the next track automatically.

3.4 Routing music to a mixer

A common scenario is routing the music to an Audio Mixer in Unity. There is nothing special with regard to this when using the Music Player, but it's documented here for completeness.

The first thing you need to do is create an Audio Mixer asset, if you haven't got one already. Then, in the Audio Mixer, add a new group and name it something like *Music*.

Finally, all you need to do is set the *Output* of the Audio Source on the Music Player to this Audio Mixer group.

You can then use this mixer group for setting the music volume, adding ducking and effects and so on like you normally would.

3.5 Controlling music volume

If you are routing the output to a mixer, it is recommended that you use the mixer for controlling the music volume.

Otherwise, it is important that you set the volume on the Music Player itself, and not on the Audio Source, as the Music Player will overwrite the volume on the Audio Source.

4 Playlists and tracks

The Music Player will play playlists, which can contain any number of tracks. Playlists can be played in order, or shuffled. Playlists can be created and played at runtime, see the chapter Managing playlists at runtime for more information about this. In most cases however, it will make more sense to set up the playlists beforehand, which is what we will cover here.

4.1 Setting up playlists

Playlists can be created and deleted in the *Music Player inspector*. To create a new playlist, click the *Add new playlist* button. To delete a playlist, click the '-' button.

Playlists can be moved up and down with the *Up* and *Dn* buttons. The order of the playlists is not important, except the first playlist which is the one that will be played on awake if *Play on Awake* is set, or when calling *Play* for the first time without a playlist parameter.

Use the foldout button on the playlist to reveal its options and tracks. Here you can set the name of the playlist, and if it should be shuffled when playing.

You can call *Play* with this name as a string to play a specific playlist.

4.2 Setting up tracks

To add a new track, click the *Add new track* button. The tracks have the same controls as the playlists, *Up* and *Dn* to move the track up or down, and '-' to delete.

Use the foldout to set the track options. A track can be enabled or disabled, when disabled it will not be played. If no tracks are enabled in a playlist, the playlist will not play.

A name for the track can also be set here. This is what will be returned by the property *MusicPlayer.CurrentTrackName* and the tracks *ToString* method.

Finally, a track needs an associated Audio Clip for the actual music; this can be any imported Audio Clip that Unity supports.

5 Playing music

To interact with the *Music Player*, a reference to it is needed. You get this in the same way as any other component reference in Unity, normally assigned in the editor.

The music player lives in its own namespace, so to use it in code you need to import it in any source code where you want to use it. This is done with *using FreakLib.Music* in C# or with *import FreakLib.Music* in JavaScript.

The full API is documented in the included *API Reference* manual.

5.1 Playing a playlist

There are three methods to play a playlist. Calling *Play()* without arguments will begin playing the last active playlist, or the first one in the list if the current playlist has not yet been set.

To play a playlist by its name, you call the method with a string parameter, for example *Play("name")*. If a playlist with the given name is not found, a *UnityException* will be thrown.

A playlist can also be played with a reference to a Playlist object. Use *Play(playlist)* for this.

Note that calling *Play* will reset the history, even if called with the same playlist.

5.2 Stopping

To stop playing, simply call the *Stop* method. After calling *Stop*, the only way to start playing again is to call *Play*. As noted above, this will reset the history. If you don't want this, use *Pause* instead, optionally followed by seeking to the beginning of the track.

5.3 Pausing

To pause playing, call the *Pause()* method. If the player is currently paused, this will do nothing.

You can also use the method *PauseOrResume()*, this will pause if the player is currently playing, and will resume if called while the player is paused. This will do nothing if the player has been stopped instead of paused.

5.4 Skipping

Skip to the next or previous track by calling the methods *Next()* and *Previous()*.

If there are no more enabled tracks in the playlist, calling *Next()* will cause the music player to stop.

It is possible to skip backwards the number of tracks that are defined by the *History Length* setting. When this limit is reached, the music player will just continue playing the current track if *Previous()* is called.

5.5 Seeking

There are two properties on the player that can be used to read or control the current playtime, *Playtime* and *PlaytimeNormalized*.

Playtime will return the time since the start of the track in seconds and *PlaytimeNormalized* will return a value between 0 and 1 where 0 is the start of the track, and 1 is the end.

If you want the length of the current track, use *CurrentTrack.Length*.

Both *Playtime* and *PlaytimeNormalized* can be set to seek to a position in the track.

6 Events

The *Music Player* defines a number of Unity events that will be fired when specific things occur. These are set like normal Unity events, either in the inspector or the runtime, see the Unity documentation for further information about Unity events.

It is possible to get a reference to the *Music Player* that is sending the event in the callback, by having *MusicPlayer* as the single parameter in the method declaration.

6.1 OnPlay

The *OnPlay* event will fire when the player starts playing, triggered by *Play On Awake* or whenever a *Play* method is called.

6.2 OnStop

The *OnStop* event fires whenever the *Stop* method has been called, or when the player needs to stop because for example no more enabled tracks exist to play in a playlist.

6.3 OnPause

The *OnPause* event fires when the player is paused, by calling the methods *Pause* or *PauseOrResume*.

6.4 OnUnpause

The *OnUnpause* event fires when the player resumes from pause, when the methods *UnPause* or *PauseOrResume* are called. If the player is not paused when one of these methods is called, the event will not fire.

6.5 OnTrackChange

The *OnTrackChange* event will fire whenever the player changes track, either when this is done automatically when the current track ends, or when the *Next* or *Previous* methods are called. If there is only one enabled track in a playlist, this event will still be fired even if it will just re-start the same track.

This event will also fire when a playlist is first started.

6.6 OnPlaylistChange

The *OnPlaylistChange* event will fire when the player changes playlists, as a response to *Play On Awake* or the *Play* method being called. However if *Play* is called with the same playlist as the current one, this event will not fire.

7 Managing playlists at runtime

In addition to setting up playlists and tracks in the inspector, they can also be created at runtime.

This can be useful if you for example are downloading tracks at runtime that you want to play, or if you want to allow users to create their own playlists.

Remember to include the namespace *FreakLib.Music* in your code, with *import FreakLib.Music* in JavaScript or *using FreakLib.Music* in C#.

7.1 Creating a Track at runtime

Before creating a Playlist, it makes sense to first create a list of tracks that you want in your playlist.

Tracks are created at runtime with a normal constructor. If you use the constructor without parameters, you should take care to set the *clip* and *name* variables.

The *foldout* variable is only used for the inspector, and the *Plays* property is only used internally to track the number of plays of the track for the shuffle function.

7.2 Creating a Playlist at runtime

A *Playlist* is created the same way as a track, with a normal constructor. The *name* variable is the name of the playlist, and the *tracks* should be set to a generic list of tracks, *List<Track>*. The *shuffle* parameter determines if the playlist should be played in order, or shuffled.

7.3 Playing created playlists

Once you have created a *Playlist*, you can play it directly in the *Music Player* by passing it as an argument to the *Play* method.

You can also add it to the list of playlists in the player. This list is exposed as the *List<Playlist>* property *Playlists*. To add the playlist to this list, use the method *MusicPlayer.Playlists.Add()*. This will allow you to play the playlist by its name, with the *Play("name")* method.