



Table of Contents

- [MicroAudio 2.0](#)
- [Contact](#)
- [About](#)
- [Setup and Installation](#)
- [Cheatsheet](#)
- [Features](#)
 - [Minimal Allocation](#)
 - [MicroAudio Manager](#)
 - [MicroHandle](#)
 - [MicroSource](#)
- [Channels](#)
- [Sounds](#)
- [Music](#)
- [MicroFade](#)
- [MicroDelay](#)
- [Crossfade](#)
- [Save/Load](#)

MicroAudio 2.0

Powerful and streamlined library with minimal memory allocation for managing audio in game.

Contact

Discord: <https://discord.gg/a6EP9WVPU3>

Website: <https://www.microlightgames.com>

X: <https://x.com/microlightgames>

E-mail: contact@microlightgames.com

Ko-fi: <https://ko-fi.com/microlightgames>

About

MicroAudio manager eases the management of audio in your game. It saves You time by doing core audio features for You.

- Minimal memory allocation
- Manages audio channels
- Save/Load volume and mute settings
- Play music and sounds easily from anywhere in the project
- Crossfade tracks
- Support for custom audio sources
- Customizable audio channels
- Source code
- ~~Infinity sounds which replace monotonous loops~~

Setup & Installation

Asset Store

MicroAudio can be found on the Unity Asset Store: <https://assetstore.unity.com/packages/slug/272359>

Raw git Download

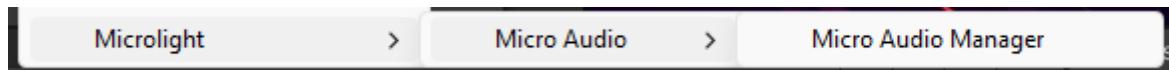
MicroAudio can be found on GitLab: <https://gitlab.com/microlight-unity/micro-audio>

Download '**Assets/Microlight**' folder and place it inside '**Assets**' folder of your project.

Scene Setup

There are 3 options:

1. Right-click on the hierarchy window > **Microlight** > **Micro Audio** > **MicroAudio Manager**.
2. Top toolbar of Unity under > **Plugins** > **Microlight** > **Micro Audio** > **MicroAudio Manager**.
3. Or drag&drop prefab from the **Prefabs** folder into the scene.



MicroAudio manager is required to be present in the scene.

Only one is needed in the starting scene and then it will persist between scenes because the manager is set as **DontDestroyOnLoad**.

Video tutorial can be found here: <https://youtu.be/LryjQARKRMs>
(video tutorial outdated and use of [Cheatsheet](#) is recommended)

Cheatsheet

Inside **MicroAudio** folder is the **cheatsheet.png** image which gives easy visual representation of the API and how to use the library.

Demo

The project contains a demo scene where manager features are presented and how to use the API. Feel free to explore the scene. **DemoSceneManager.cs** script showcases how to use the manager.

Sounds from kenney.nl by Kenney
UI graphics from kenney.nl by Kenney

Background from freepik.com by tohamina



Features

Minimal Allocation

Minimal allocation means that all of the objects that are used by the MicroAudio are cached and pooled. Objects get allocated memory when created and then they hold onto it and are reused. So when sound finishes reproduction it is cleared and sent back to the pool. For those reasons MicroAudio has safety system in place through the MicroHandle system which will give control of the sound to the user only for that use and wont be valid if the sound gets reused again. But MicroHandle also allows for directly referencing MicroSource, AudioSource, MicroFade and MicroDelay for advanced use but in that case user could cause unexpected behaviour. Best way to prevent such cases is to subscribe to the OnEnded event in MicroSource and not use the reference after that point, same goes for the MicroFade and MicroDelay references while AudioSource has the same lifespan as the MicroSource.

MicroAudio Manager

MicroAudio manager is singleton which allows it to be controlled from anywhere in the project. Manager is mainly hub for branching to each channel or module in MicroAudio. Every channel has its own volume and mute settings which can be always changed. Parent audio channel also affects every child audio channel.

Methods

SaveSettings	Stores current values for the volume and mute of the every audio channel
LoadSettings	Loads values for the volume and mute from the storage for the every audio channel
PauseAllSounds	
ResumeAllSounds	
StopAllSounds	
Properties (channels)	
Master	Master channel which is used for controlling settings of every channel
Music	Music channel is for playing music clips and playlists
SoundsMaster	Master channel but only for the sounds without affecting the music channel
SFX	4 sound channels for more control over sounds
UI	
Ambience	
Dialogue	

MicroHandle

MicroHandle is the main way of interacting with sounds that are playing. It is the safety system implemented so that user is safe from the exceptions or unintended use of MicroAudio.

Methods

IsValid	Is the handle still valid? If not, commands won't do anything
GetMicroSource	
Get AudioSource	
Get AudioClip	
Get Delay	
Get Fade	
Get Length	Returns the length of the clip modified by the pitch
Get Progress	Returns value between 0 and 1 of the current clip progress
Is Paused	
Is Playing	
Stop	
Pause	
Resume	
Fade	Fades the sound to the desired volume
Fade Out At End	Before the sound finishes playing, sound will start to fade out

MicroSource

MicroAudio uses special MicroSource wrapper around AudioSource which allows for more information about the AudioSource lifecycle. It is used mainly for internal use but events are exposed.

Events

On Started
On Paused
On Resumed
On Stopped
On Completed
On Ended
On Fade Out

Channels

MicroAudio has integrated mixers for easy control of volume in your games. By default, there are 7 channels (master, music, ambience, sounds, SFX, UI and dialogue) but this can be customized.

Lite Channels

If your game is simpler and needs only 2 channels (music and sounds), MicroAudio has very simple solution.

In `Microlight > MicroAudio > Scripts > Core > MicroAudio.Channels.cs` scripts, uncommenting first line `//#define LITE_MODE`, will turn on `Lite mode` which will make MicroAudio much lighter. After reloading project (usually done automatically when switching back to the editor), MicroAudio manager in your scene needs to be reconnected with the channel data scriptable objects. Channel data is located in `Microlight > MicroAudio > Prefabs > LiteMixer` which should be dragged onto the MicroAudio manager on the scene. After that MicroAudio will use lite version of the mixer.

All Channel Controls

Every channel has access to these controls but it is important to note that after changing settings, settings will not persist so they need to be saved.

Properties

`Muted`

`Volume`

Methods

`SaveSettings`

`LoadSettings`

Example: Change volume of the music channel and save its settings

```
MicroAudio.Music.Volume = 0.5f;  
MicroAudio.Music.SaveSettings();
```

Audio Channels Hierarchy

Parent channel settings also affect the child channels.

Full:

- Master
 - Music
 - Ambience
 - Sounds
 - SFX
 - UI
 - Dialogue

Lite:

- Master
 - Music
 - Sounds

Sounds

While there are many audio channels for the sounds, they all share the same controls

Methods

PlaySound Custom AudioSource can be passed so it will be used instead of pooled one

PauseAll

ResumeAll

StopAll

Example: Plays sound effect when the button is clicked

```
using Microlight.MicroAudio; // It is important to include MicroAudio namespace  
in scripts you wish to use MicroAudio  
  
public Button myButton;  
public AudioClip clickSound;  
  
void Start()  
{  
    myButton.onClick.AddListener(OnClicked);  
}  
  
void OnClicked()  
{  
    MicroAudio.UI.PlaySound(clickSound);  
}
```

Example: Displaying message when sound effect ends

```
MicroAudio.UI.PlaySound(transactionSound).OnComplete((x) =>
    TransactionSoundFinished);

...
void TransactionSoundFinished(MicroSource src)
{
    DisplayMessage("Transaction complete!");
}
```

Music

Music allows playing of one clip or the whole playlist. Many settings

Example: Looped main menu theme music

```
public AudioClip mainMenuTheme;

void Start()
{
    MicroAudio.Music.PlayTrack(mainMenuTheme, true);
}
```

Example: Tracking progress of music track

```
private void Update()
{
    trackProgressSlider.value = MicroAudio.Music.GetTrackProgress();
}
```

MicroFade

Properties

[IsPaused](#)

[IsFinished](#)

[GetStartVolume](#)

[GetEndVolume](#)

[GetDuration](#)

[GetElapsed Time](#)

Properties

`IsActive`

`GetMicroSource`

`GetProgress`**Methods**

`Kill`

`SkipToEnd`**Events**

`OnStarted`

`OnPaused`

`OnResumed`

`OnCompleted`

`OnEnded`

Example: Playing sound at 0 volume and then fading it in to full volume over 2 seconds

```
void Start()
{
    MicroAudio.SFX.PlaySound(introSound, 0f, 0f).Fade(1f, 2f);
}
```

MicroDelay

Properties

`IsPaused`

`IsFinished`

`GetDuration`

`GetElapsedTime`

`IsActive`

`GetMicroSource`

`GetProgress`**Methods**

`Kill`

`SkipToEnd`

Events

[OnPaused](#)

[OnResumed](#)

[OnCompleted](#)

[OnEnded](#)

Example: Playing sound after 2 seconds

```
void Start()
{
    MicroAudio.SFX.PlaySound(screamSound, 2f);
}
```

Crossfade

Crossfade allows the current song to gradually lower its volume at the end while the new song increases its volume and slowly takes over. The duration of the effect can be set by the [Music.SetCrossfadeDuration](#).

Example: Playing new playlist where songs will be crossfaded over 2 seconds

```
void Start()
{
    MicroAudio.Music.SetCrossfadeDuration(2f);
    MicroAudio.Music.PlayPlaylist(listOfSongs);
}
```

Save/Load

Saving and loading of volume settings is done through the [PlayerPrefs](#) feature from Unity. Simply call the [MicroAudio.SaveSettings](#) and [MicroAudio.LoadSettings](#) method to save all channels settings or [MicroAudio.Music.SaveSettings](#) for a specific channel. An example of saving and loading volume settings can be inspected in a pre-built demo scene. In the demo scene volume is saved as soon as it is changed but it can be saved on applying volume settings or when leaving the options screen.