

Pooled Audio Event System — Manual

Publisher: STL Dynamics

Namespace: `STLDynamics.PooledAudio`

1. Overview

The **Pooled Audio Event System** is a lightweight, event-driven audio workflow for Unity. Instead of placing a new `AudioSource` everywhere and manually wiring clips, volumes, mixer groups, distance settings, and cleanup logic, you author reusable **SoundEvent** assets and play them through a single **PooledAudioManager**.

The system is built around three main pieces:

PIECE	PURPOSE
<code>SoundEvent</code>	A ScriptableObject that defines what a sound is: clips, volume/pitch randomization, mixer bus, 2D/3D settings, distance culling, voice limits, distance layers, and loop behavior.
<code>PooledAudioManager</code>	The scene/service object that owns the AudioSource pool, plays one-shots and loops, handles music playback, manages mixer volume controls, and exposes the main code API.
<code>SoundEventPlayer</code>	An optional component wrapper that feels similar to an <code>AudioSource</code> , but still uses pooled playback behind the scenes. Useful for designers, UI buttons, animation events, triggers, and simple scene wiring.

The goal is simple: **author sounds once, play them anywhere, and avoid AudioSource spam.**

2. When to Use This System

Use this system for:

- UI clicks, menu sounds, and global 2D feedback.
- Positional one-shots such as impacts, footsteps, weapons, doors, explosions, pickups, and creature sounds.
- Tracked loops such as engines, machines, ambience emitters, alarms, fires, or magic effects.
- Music playback with fade in, fade out, and crossfade support.
- Distance-aware sounds that should use different clips at near/mid/far ranges.
- Projects that need simple voice limits and distance culling before an `AudioSource` is leased.

This package does **not** replace Unity's entire audio pipeline. It still uses Unity `AudioSource`, `AudioClip`, `AudioMixer`, `AudioMixerGroup`, rolloff settings, spatial settings, and custom curves. It just gives you a cleaner pooled workflow on top.

3. Core Concepts

SoundEvent

A `SoundEvent` is the data definition for a sound. It answers questions like:

- What clips can this sound choose from?
- Is this a 2D or 3D sound?
- What mixer bus should it route to?
- How loud should it be?
- Should pitch vary?
- How far away can the listener be before the sound is skipped?
- How many copies of this sound may play at once?
- Should a loop keep following a transform?
- Should near/mid/far clips be selected based on listener distance?

Think of `SoundEvent` as a reusable audio preset plus clip list.

PooledAudioManager

The `PooledAudioManager` is the central runtime service. Create one from **GameObject > STL Dynamics > Audio > Create Pooled Audio Manager**, usually in a boot scene, persistent scene, or gameplay test scene. It creates and reuses pooled `AudioSource` objects as sounds are played.

It provides the main API:

```
PooledAudioManager.Instance.PlayAt(explosionEvent, transform.position);
PooledAudioManager.Instance.Play2D(uiClickEvent);
PooledAudioManager.Instance.PlayMusic(menuMusicEvent, 1f);
```

For loops, it returns a `PooledAudioHandle`:

```
PooledAudioHandle engineLoop =
    PooledAudioManager.Instance.PlayLoopFollowing(engineHumEvent, vehicleTransform);

PooledAudioManager.Instance.StopLoop(engineLoop);
```

SoundEventPlayer

`SoundEventPlayer` is a convenience component. It gives you an `AudioSource`-like workflow without needing an actual `AudioSource` on the object.

Use it when you want to:

- Assign a `SoundEvent` in the inspector.
- Call `Play()` from code.
- Call methods from `UnityEvents`.
- Trigger sounds from animation events.
- Use Play On Awake / Play On Enable style behavior.
- Let non-programmers wire sounds quickly.

Example:

```
[SerializeField] private SoundEventListener footstepsPlayer;

public void AnimationEvent_Footstep()
{
    footstepsPlayer.PlayOneShot();
}
```

4. Basic Setup

1. In the Unity menu bar, choose **GameObject > STL Dynamics > Audio > Create Pooled Audio Manager**.
2. Keep only one active manager unless you intentionally want scene-local audio behavior.
3. Select the created **PooledAudioManager** GameObject in the Hierarchy.
4. Enable **Dont Destroy On Load** if the manager should persist across scenes.
5. Assign an `AudioMixer` if you want runtime volume control.
6. Assign mixer groups for the buses you plan to use.
7. Set the pool size.
8. Assign a listener override if your project does not use `Camera.main` as the listener position.

Optional: To add a ready-to-wire volume slider canvas, choose **GameObject > STL Dynamics > Audio > Spawn Volume Slider**. The slider prefab is already a canvas, so it is created directly in the scene.

Recommended First Setup

For most projects, start with:

SETTING	RECOMMENDED STARTING VALUE
Dont Destroy On Load	Enabled
Initial Pool Size	32
Max Pool Size	64
Source Prefab	Empty unless you need a custom AudioSource prefab
Listener Override	Your player/camera listener transform, or leave empty to use <code>Camera.main</code>
Fallback To Camera Main	Enabled

The manager creates inactive pooled AudioSources on startup. When sounds play, sources are leased from the pool, configured from the `SoundEvent`, played, then returned to the pool when finished.

Mixer Setup

The manager can route events through logical buses:

```
SoundBus.Master
SoundBus.Music
SoundBus.SFX
SoundBus.UI
SoundBus.Gameplay
SoundBus.Ambience
SoundBus.Cinematic
```

```
SoundBus.Foley
SoundBus.Footsteps
SoundBus.Weapons
SoundBus.Voice
SoundBus.Vehicles
SoundBus.Background
SoundBus.Emitters
SoundBus.Menu
```

Each `SoundEvent` chooses a `SoundBus`. The manager maps that bus to an `AudioMixerGroup`.

If a bus group is not assigned, the manager falls back to the Master group. If no group is assigned, Unity will still play the `AudioSource` using default output routing, but mixer-based volume control will not affect it.

Mixer Volume Parameters

The volume API expects the `AudioMixer` to expose these parameter names:

```
MasterVolume
MusicVolume
SFXVolume
UIVolume
GameplayVolume
AmbienceVolume
CinematicVolume
FoleyVolume
FootstepsVolume
WeaponsVolume
VoiceVolume
VehiclesVolume
BackgroundVolume
EmittersVolume
MenuVolume
```

Each volume method takes a `0..1` linear value and converts it to decibels internally.

Example settings UI:

```
public void SetSfxVolume(float value)
{
    PooledAudioManager.Instance.SetSFXVolume(value);
}

public void SetMusicVolume(float value)
{
    PooledAudioManager.Instance.SetMusicVolume(value);
}
```

5. Creating a SoundEvent

Create a new `SoundEvent` from:

```
Assets > Create > STL Dynamics > Pooled Audio > Sound Event
```

A `SoundEvent` asset is where most sound authoring happens.

Identity

FIELD	MEANING
Event Name	Friendly name used by debug tools. If empty, the asset name is used.
Bus	The default mixer bus for this event.

Use clear names like:

```
SE_UI_Click
SE_Door_Open
SE_Explosion_Small
SE_Footstep_Concrete
SE_Engine_Loop
SE_Music_Menu
```

Good names make debugging way less cursed.

Clips

Assign one or more default clips.

When multiple clips are assigned, the system chooses a random clip each time the event plays.

Use default clips for:

- UI sounds.
- 2D sounds.
- Music events.
- Simple positional one-shots.
- Fallback sounds when no distance layer matches.

Important: **Global 2D and UI playback use the default clip list.** If an event only has distance-layer clips and no default clips, it is meant for positional playback.

Volume and Pitch Randomization

FIELD	MEANING
Volume Min / Max	Random volume range for each playback.
Pitch Min / Max	Random pitch range for each playback.

Examples:

SOUND TYPE	VOLUME	PITCH
UI click	0.9 - 1.0	0.98 - 1.02
Footstep	0.75 - 0.95	0.92 - 1.08
Explosion	0.95 - 1.0	0.9 - 1.05
Creature chirp	0.8 - 1.0	0.85 - 1.2

Small pitch and volume variation helps repeated sounds feel less robotic.

AudioSource Output Settings

The SoundEvent applies common AudioSource settings when a pooled source is leased:

- Mute
- Priority
- Stereo pan
- Velocity update mode
- Bypass effects
- Bypass listener effects
- Bypass reverb zones
- Ignore listener pause
- Ignore listener volume

Use these the same way you would use them on a normal Unity `AudioSource`.

3D Audio Settings

FIELD	MEANING
Spatial	If enabled, the event plays as positional 3D audio unless the play call forces 2D.
Spatial Blend	0 = 2D, 1 = fully 3D.
Min Distance	Distance where the sound begins attenuating depending on rolloff.
Max Distance	Distance used by Unity's 3D audio rolloff.
Rolloff Mode	Unity rolloff mode.
Spread	Stereo spread for 3D sound.
Doppler Level	Doppler effect amount.
Reverb Zone Mix	Reverb zone influence.
Spatialize	Enables Unity spatialization if available.
Spatialize Post Effects	Applies spatialization after source effects.

For normal positional SFX, use:

```
Spatial: Enabled
Spatial Blend: 1
Min Distance: 1
Max Distance: 30-60
Rolloff Mode: Linear or Logarithmic
```

For UI sounds, use:

```
Spatial: Disabled
```

Custom Audio Curves

SoundEvents support optional custom curves for:

- Volume rolloff
- Spatial blend
- Spread
- Reverb zone mix

Enable these only when you need more control than Unity's default fields provide.

Transform Following

FIELD	MEANING
Follow Transform One Shot	Positional one-shots with a target can follow that target until the clip ends.
Follow Transform While Looping	Positional loops with a target follow that target while active.

Use follow behavior for moving emitters:

- Cars
- Projectiles
- Characters
- Weapons
- Creatures
- Machines
- Magic effects

If follow is disabled, playback uses the starting position only.

Looping and Voice Control

FIELD	MEANING
Loop	Default loop flag applied to the leased AudioSource. PlayLoop calls also force looping.
Max Voices	Maximum simultaneous audible instances of this exact SoundEvent. 0 means unlimited.
Priority	Custom gameplay priority value reserved for user extensions.

Voice limits are counted per **SoundEvent** . If **SE_Footstep_Concrete** has **Max Voices = 8** , only eight audible instances of that specific event can play at once.

Virtualized loops do not count against the voice limit until they become audible and lease a source.

Distance Culling

FIELD	MEANING
Skip If Listener Too Far	If enabled, positional playback can be skipped before using a source.
Cull Distance	Distance beyond which positional one-shots are skipped and loops are virtualized.

Keep **Cull Distance** greater than or equal to **Max Distance** for most sounds.

Example:

Max Distance: 40

Cull Distance: 50 or 60

One-shots beyond the cull distance are skipped. Loops beyond the cull distance stay tracked but release their AudioSource until they return to range.

Mixer Override

A SoundEvent can optionally assign a direct `AudioMixerGroup` override. If set, that override wins over the selected `SoundBus`.

Use this for special cases. For most events, prefer the bus dropdown so the project stays organized.

6. Distance-Based Sound Layers

Distance sound layers let a positional SoundEvent choose different clips based on listener distance.

Example weapon shot setup:

LAYER	DISTANCE	CLIPS
Near	0 - 20	Loud close gunshot clips
Mid	20 - 75	Crack / outdoor shot clips
Far	75 - 200	Muffled distant shot clips

Each distance layer can define:

- Layer name
- Min distance
- Max distance
- Clips
- Volume multiplier
- Pitch multiplier
- Optional mixer group override

How Selection Works

When positional playback starts:

1. The manager checks listener distance.
2. The SoundEvent looks for the first layer containing that distance.
3. If a matching layer exists, it chooses a random clip from that layer.
4. If no matching layer exists, it can either use default clips or the closest layer depending on the fallback setting.

Fallback Behavior

SETTING	BEHAVIOR

Fallback To Default Clips When No Distance Layer Matches = Enabled	Use the default <code>Clips</code> list when no layer matches.
Fallback To Default Clips When No Distance Layer Matches = Disabled	Use the closest layer with clips.

Auto Enable Layers

If **Auto Enable Distance Sound Layers When Clips Assigned** is enabled, the `SoundEvent` turns distance layer selection on during validation when layer clips exist.

This is useful because it prevents the classic “I filled out layers but forgot to enable the feature” problem. Tiny quality-of-life win. We take those.

Distance Layers and Loops

Tracked loops can use distance layers too. The loop mode controls what happens after the loop starts.

MODE	BEHAVIOR
Lock On Start	Pick one layer at loop start and keep it until stopped.
Reevaluate On Reenter	Pick on start, then pick again if a virtualized loop re-enters audible range. Good default for ambience and machines.
Reevaluate While Moving	Re-check while audible. If the listener/source enters a different layer, the clip is swapped directly.

In the current version, moving loop layer changes use a direct clip swap, not a crossfade. For smooth ambience beds, prefer `Reevaluate On Reenter`.

7. Using the PooledAudioManager API

Add this namespace at the top of scripts that use the system:

```
using STLDynamics.PooledAudio;
```

Playing a Basic One-Shot

```
[SerializeField] private SoundEvent explosionEvent;

public void Explode()
{
    PooledAudioManager.Instance.PlayAt(explosionEvent, transform.position);
}
```

This plays a positional one-shot at the world position.

Playing at a Transform

```
[SerializeField] private SoundEvent pickupEvent;
```

```
public void PlayPickupSound(Transform pickup)
{
    PooledAudioManager.Instance.PlayAt(pickupEvent, pickup);
}
```

If the SoundEvent has **Follow Transform One Shot** enabled, the sound follows the target while it plays.

Forcing a Positional Sound to 2D

```
PooledAudioManager.Instance.PlayAt(alertEvent, transform.position, force2D: true);
```

This still uses the world position for distance checks, but the leased AudioSource plays with `spatialBlend = 0`.

Playing a Following One-Shot

```
PooledAudioManager.Instance.PlayFollowing(jumpEvent, playerTransform);
```

This explicitly makes the one-shot follow the transform while it is alive.

Playing Global 2D Audio

```
PooledAudioManager.Instance.Play2D(notificationEvent);
```

Global 2D sounds ignore distance culling.

Use this for:

- UI feedback
- Notifications
- Global game state sounds
- Non-positional feedback

Playing UI Audio

```
PooledAudioManager.Instance.PlayUI(buttonClickEvent);
```

`PlayUI` is currently an alias for `Play2D`. Use it anyway when the intent is UI, because it makes your code easier to read.

Starting a Positional Loop

```
[SerializeField] private SoundEvent alarmLoop;

private PooledAudioHandle _alarmHandle;

public void StartAlarm()
{
    _alarmHandle = PooledAudioManager.Instance.PlayLoop(alarmLoop, transform.position);
}

public void StopAlarm()
{
    PooledAudioManager.Instance.StopLoop(_alarmHandle);
}
```

```
_alarmHandle = PooledAudioHandle.Invalid;  
}
```

Loops return a `PooledAudioHandle`. Store it if you need to stop or modify the loop later.

Starting a Loop That Follows a Transform

```
[SerializeField] private SoundEvent engineLoop;  
  
private PooledAudioHandle _engineHandle;  
  
public void StartEngineAudio(Transform vehicle)  
{  
    _engineHandle = PooledAudioManager.Instance.PlayLoopFollowing(engineLoop, vehicle);  
}  
  
public void StopEngineAudio()  
{  
    PooledAudioManager.Instance.StopLoop(_engineHandle);  
    _engineHandle = PooledAudioHandle.Invalid;  
}
```

Use this for vehicles, machines, creatures, characters, and moving objects.

Updating a Loop Position

```
PooledAudioManager.Instance.SetLoopPosition(_alarmHandle, newWorldPosition);
```

This switches the loop to a fixed world position.

Updating a Loop Follow Target

```
PooledAudioManager.Instance.SetLoopFollowTarget(_engineHandle, newVehicleTransform);
```

This makes the loop follow a different transform.

Changing Loop Volume or Pitch

```
PooledAudioManager.Instance.SetLoopParams(_engineHandle, volume: 0.75f);  
PooledAudioManager.Instance.SetLoopParams(_engineHandle, pitch: 1.15f);  
PooledAudioManager.Instance.SetLoopParams(_engineHandle, volume: 0.8f, pitch: 1.1f);
```

Values are clamped internally:

```
Volume: 0..1  
Pitch: -3..3
```

Muting a Loop

```
PooledAudioManager.Instance.SetLoopMuted(_engineHandle, true);  
PooledAudioManager.Instance.SetLoopMuted(_engineHandle, false);
```

Request-level mute combines with event-level mute. If the SoundEvent is muted, the final AudioSource is muted even if the request is not.

Checking Loop State

```
bool tracked = PooledAudioManager.Instance.IsLoopActive(_engineHandle);
bool audible = PooledAudioManager.Instance.IsLoopAudible(_engineHandle);
```

These are intentionally different:

METHOD	MEANING
IsLoopActive	The loop handle is still tracked. It may be audible or virtualized.
IsLoopAudible	The loop currently owns a pooled AudioSource and is actually audible.

A loop can be active but not audible when it is distance-culled, blocked by voice limits, or waiting for a free pooled source.

Stopping All Pooled Audio

```
PooledAudioManager.Instance.StopAllPooledAudio();
```

This stops pooled one-shots and loops, clears tracked loops, clears voice counts, and resets active source counts.

It does not replace scene unloading or full audio cleanup logic, but it is useful for hard transitions, debug tools, and scene resets.

8. Music Playback

The manager has a dedicated two-source music player for fade in, fade out, and crossfade behavior.

Playing Music

```
[SerializeField] private SoundEvent menuMusic;

public void PlayMenuMusic()
{
    PooledAudioManager.Instance.PlayMusic(menuMusic, fadeTime: 1f);
}
```

Music playback:

- Chooses a random clip from the SoundEvent.
- Plays it as 2D audio.
- Forces looping.
- Applies SoundEvent volume and pitch randomization.
- Uses the SoundEvent override group if one exists, otherwise uses the manager's music group/bus routing.

Stopping Music

```
PooledAudioManager.Instance.StopMusic(fadeTime: 0.5f);
```

Silencing Music Without Stopping It

```
PooledAudioManager.Instance.SetMusicSilenced(true, fadeTime: 0.25f);  
PooledAudioManager.Instance.SetMusicSilenced(false, fadeTime: 0.25f);
```

This fades the active music source down to zero or back to its last target volume. It is useful for pause menus, cutscenes, or temporary ducking-style behavior.

9. Runtime Stats and Debugging

The manager exposes:

```
PooledAudioStats stats = PooledAudioManager.Instance.Stats;
```

Available fields:

```
stats.poolSize  
stats.activeSources  
stats.activeLoops  
stats.trackedLoops  
stats.virtualizedLoops  
stats.peakActiveSources  
stats.skippedInvalidRequests  
stats.skippedByVoiceLimit  
stats.skippedByDistance  
stats.skippedByPoolLimit
```

Use these for debug panels, stress tests, and tuning.

Resetting Debug Stats

```
PooledAudioManager.Instance.ResetDebugStats();
```

This resets peak/skipped counters while preserving currently active playback.

What the Counters Mean

COUNTER	MEANING
Pool Size	Current number of pooled AudioSources created.
Active Sources	Pooled sources currently in use.
Active Loops	Loops currently audible.
Tracked Loops	Loop handles currently tracked, audible or virtualized.

Virtualized Loops	Loops tracked but not currently audible.
Peak Active Sources	Highest active source count since stats were reset.
Skipped Invalid Requests	Missing event, no clips, or invalid clip selection.
Skipped By Voice Limit	Playback blocked by the SoundEvent max voice count.
Skipped By Distance	Positional one-shots skipped or loops virtualized due to distance culling.
Skipped By Pool Limit	No pooled source available and max pool size reached.

10. Using SoundEventPlayer

`SoundEventPlayer` is the inspector-friendly wrapper. It is best when you want audio behavior on a `GameObject` without writing manager calls everywhere.

Basic Setup

1. Add `SoundEventPlayer` to a `GameObject`.
2. Assign a `SoundEvent`.
3. Choose a **Playback Mode**.
4. Choose an **Output Mode**.
5. Call `Play()`, enable Play On Awake, or wire a `UnityEvent`/animation event.

Playback Modes

MODE	BEHAVIOR
OneShot	Plays the SoundEvent once.
Loop	Starts a tracked loop and stores its handle inside the component.
Music	Sends the SoundEvent to the manager music player.

Output Modes

MODE	BEHAVIOR
Positional	Uses this <code>GameObject</code> or the Audio Origin Override as the sound position.
Global2D	Plays as global 2D audio and ignores distance culling.
UI	Same playback behavior as Global2D, but clearer intent for UI wiring.

Play On Awake and Play On Enable

FIELD	MEANING
Play On Awake	Plays from <code>Start()</code> , giving the manager time to initialize.

Play On Enable	Plays whenever the component becomes enabled.
Stop On Disable	Stops tracked loop/music when disabled.
Stop On Destroy	Stops tracked loop/music when destroyed.

Delay and Cooldown

FIELD	MEANING
Start Delay	Delay before playback starts.
Cooldown	Minimum time between successful play calls.

Cooldown is useful for things like:

- Footsteps
- Door sounds
- Buttons
- Damage ticks
- Trigger zones
- Repeated interactions

Note: the cooldown uses unscaled time. Delayed playback uses Unity's `WaitForSeconds`, so it follows normal scaled time.

Audio Origin

By default, a `SoundEventPlayer` uses its own transform as the audio origin.

You can override this with **Audio Origin Override**, or at runtime:

```
soundEventPlayer.SetAudioOrigin(muzzleTransform);
```

Clear the override:

```
soundEventPlayer.ClearAudioOrigin();
```

Force 2D

```
soundEventPlayer.SetForce2D(true);
```

For positional playback, this forces the sound to play as 2D after positional culling and layer selection.

Follow Toggles

```
soundEventPlayer.SetFollowTransformOneShot(true);
soundEventPlayer.SetFollowTransformWhileLooping(true);
```

These let the component override or enforce following behavior when playing through the manager.

11. Using SoundEventListener From Code

Common Play Methods

```
player.Play();
player.Play(otherEvent);

player.PlayOneShot();
player.PlayOneShot(otherEvent);

player.PlayLoop();
player.PlayLoop(otherEvent);

player.PlayMusic();
player.PlayMusic(otherEvent);

player.PlayDelayed(0.25f);
player.PlayDelayed(otherEvent, 0.25f);
```

`Play(SoundEvent overrideEvent)` does not permanently replace the assigned SoundEvent. It only uses the override for that play request.

Stop, Restart, Pause, Resume

```
player.Stop();
player.Restart();
player.Restart(otherEvent);

player.Pause();
player.UnPause();
player.Resume();

player.Toggle();
```

Important behavior:

- One-shots are fire-and-forget. `Stop()` clears the component's local one-shot playing estimate, but it does not stop an already leased one-shot source.
- Loops can be stopped because the component stores the loop handle.
- Music is stopped through the manager's music player.
- Pause is emulated. For loops, it stops the loop and remembers the event. For music, it silences the music instead of hard-stopping it.

UnityEvent and Animation Event Helpers

These methods exist because UnityEvents and animation events are happier with simple `void` methods:

```
player.PlayFromEvent();
player.StopFromEvent();
player.RestartFromEvent();
player.ToggleFromEvent();
player.PlayOneShotFromEvent();
player.PlayLoopFromEvent();
player.MuteFromEvent();
player.UnMuteFromEvent();
player.ToggleMuteFromEvent();
```

Use these directly in the Inspector.

Common Runtime Configuration

```
player.SetSoundEvent(nextEvent);

player.SetMute(true);
player.SetMute(false);
player.ToggleMute();

player.SetOutputMode(SoundEventPlayer.OutputMode.Global2D);
player.SetPlaybackMode(SoundEventPlayer.PlaybackMode.Loop);

player.SetForce2D(true);
player.SetFollowTransformOneShot(true);
player.SetFollowTransformWhileLooping(true);
```

Loop Control Through SoundEventPlayer

If the player started a loop, you can modify it:

```
player.SetLoopPosition(position);
player.SetLoopFollowTarget(target);

player.SetLoopVolume(0.6f);
player.SetLoopPitch(1.2f);
player.SetLoopParams(0.6f, 1.2f);

player.SetLoopMuted(true);
```

Useful Properties

```
SoundEvent assigned = player.SoundEvent;
player.SoundEvent = nextEvent;

bool muted = player.Muted;
player.Muted = true;

bool playing = player.IsPlaying;
bool audible = player.IsLoopAudible;

PooledAudioHandle handle = player.CurrentLoopHandle;
bool hasHandle = player.HasValidLoopHandle;
```

For one-shots, `IsPlaying` is an estimate based on the longest clip length because one-shots are not individually tracked after being fired.

12. Common Recipes

UI Button Click

SoundEvent setup:

```
Spatial: Disabled
Bus: UI
```

Clips: one or more click clips
Max Voices: 8

Code:

```
public void OnButtonClicked()
{
    PooledAudioManager.Instance.PlayUI(buttonClickEvent);
}
```

Or use a `SoundEventPlayer`:

Playback Mode: OneShot
Output Mode: UI

Wire `PlayFromEvent()` to the button's OnClick event.

Footstep From Animation Event

SoundEvent setup:

Spatial: Enabled
Bus: Footsteps
Volume variation: small
Pitch variation: small
Max Voices: 8-16
Cull Distance: 25-40

Component setup:

SoundEventPlayer on character
Playback Mode: OneShot
Output Mode: Positional
Cooldown: optional

Animation event calls:

```
public void Footstep()
{
    footstepPlayer.PlayOneShot();
}
```

Explosion at World Position

```
[SerializeField] private SoundEvent explosion;

public void Detonate(Vector3 position)
{
    PooledAudioManager.Instance.PlayAt(explosion, position);
}
```

SoundEvent setup:

```
Spatial: Enabled
Bus: SFX or Gameplay
Max Voices: 4-8
Max Distance: 80
Cull Distance: 100
```

Optional distance layers:

```
Near: 0-30
Mid: 30-100
Far: 100-250
```

Vehicle Engine Loop

```
[SerializeField] private SoundEvent engineLoop;

private PooledAudioHandle _engine;

public void StartEngine(Transform vehicle)
{
    _engine = PooledAudioManager.Instance.PlayLoopFollowing(engineLoop, vehicle);
}

public void UpdateEngine(float throttle)
{
    float volume = Mathf.Lerp(0.35f, 1f, throttle);
    float pitch = Mathf.Lerp(0.85f, 1.35f, throttle);

    PooledAudioManager.Instance.SetLoopParams(_engine, volume, pitch);
}

public void StopEngine()
{
    PooledAudioManager.Instance.StopLoop(_engine);
    _engine = PooledAudioHandle.Invalid;
}
```

SoundEvent setup:

```
Spatial: Enabled
Loop: Enabled
Follow Transform While Looping: Enabled
Max Voices: based on how many vehicles can be audible
Cull Distance: slightly higher than Max Distance
Loop Distance Variant Mode: Reevaluate On Reenter
```

Area Ambience Loop

```
private PooledAudioHandle _ambience;

public void EnterArea()
{
    _ambience = PooledAudioManager.Instance.PlayLoop(ambienceEvent, transform.position);
}

public void ExitArea()
{

```

```
        PooledAudioManager.Instance.StopLoop(_ambience);  
    }  
}
```

For large ambience emitters, distance virtualization keeps the loop handle alive while releasing the AudioSource when the listener is too far away.

Music Crossfade

```
public void PlayCombatMusic()  
{  
    PooledAudioManager.Instance.PlayMusic(combatMusicEvent, fadeTime: 1.5f);  
}  
  
public void ReturnToExplorationMusic()  
{  
    PooledAudioManager.Instance.PlayMusic(explorationMusicEvent, fadeTime: 2f);  
}
```

Calling `PlayMusic` while music is already playing crossfades between the active and inactive music sources.

13. Best Practices

Use SoundEvents as Your Source of Truth

Avoid spreading raw clips and random AudioSource settings across gameplay scripts. Let scripts ask for an event:

```
[SerializeField] private SoundEvent hitSound;
```

Then let the event own the audio behavior.

Use Manager Calls for Systems

For gameplay systems, AI, weapons, vehicles, destruction, and procedural events, call `PooledAudioManager` directly.

This keeps the logic clear and avoids unnecessary components.

Use SoundEventPlayer for Scene Wiring

Use `SoundEventPlayer` when designers need to wire behavior quickly:

- UI buttons
- Trigger zones
- Animator events
- Scene props
- Simple interactables

Keep 2D and Positional Events Separate

For clarity, make separate events when behavior differs:

```
SE_UI_Click  
SE_World_Click
```

This avoids confusion around spatial settings, culling, and distance layers.

Set Realistic Voice Limits

Voice limits prevent audio chaos when repeated events fire rapidly.

Suggested starting points:

SOUND TYPE	MAX VOICES
UI click	4-8
Footsteps	8-16
Impacts	8-16
Explosions	4-8
Weapon shots	8-24
Looped engines	Expected number of audible emitters

Tune Pool Size With Stats

If `Skipped By Pool Limit` increases during normal gameplay, raise `Max Pool Size`.

If `Peak Active Sources` is much lower than your max, your pool is probably fine.

Keep Cull Distance Honest

Cull distance should usually be just beyond the event's audible range.

Bad:

```
Max Distance: 20
Cull Distance: 500
```

That keeps too many sounds eligible.

Better:

```
Max Distance: 20
Cull Distance: 25-35
```

14. Troubleshooting

Nothing Plays

Check:

1. A `PooledAudioManager` exists in the active scene. Create one from **GameObject > STL Dynamics > Audio > Create Pooled Audio Manager**.
2. The `SoundEvent` is assigned.
3. The `SoundEvent` has clips.
4. For UI/Global2D playback, the event has default clips, not only distance-layer clips.

5. The event is not muted.
6. The manager's max pool size has not been reached.
7. The event's max voices are not already filled.
8. For positional sounds, the listener is within cull distance or distance culling is disabled.

Positional Sound Is Silent Far Away

This is likely distance culling.

Check:

```
Skip If Listener Too Far  
Cull Distance  
Listener Override  
Fallback To Camera Main
```

If no listener is found, distance culling does not reject the sound. If a listener is found and the source is beyond cull distance, one-shots are skipped and loops are virtualized.

Loop Handle Is Valid But I Hear Nothing

That can be normal.

A valid loop handle means the loop is tracked. It does not guarantee the loop currently owns an AudioSource.

Use:

```
PooledAudioManager.Instance.IsLoopActive(handle);    // tracked  
PooledAudioManager.Instance.IsLoopAudible(handle);   // currently audible
```

UI Sound Does Not Use Distance Layers

Correct. Global 2D and UI playback intentionally ignore distance culling and distance layer selection. Use default clips for UI/global playback.

Music Volume Slider Does Nothing

Check that your AudioManager exposes the correct parameter name, such as:

```
MusicVolume
```

The group assignment alone is not enough. The exposed parameter must exist on the mixer.

Too Many Sounds Are Skipped By Voice Limit

Raise the SoundEvent's `Max Voices`, or reduce how often the sound is fired.

For spammy events like footsteps, a small cooldown on `SoundEventPlayer` can help.

Too Many Sounds Are Skipped By Pool Limit

Raise the manager's `Max Pool Size`, or reduce event spam.

The debug stats are your friend here. They tell you whether the bottleneck is invalid requests, voice limits, distance, or pool capacity.

A Following Loop Stops When the Target Is Destroyed

That is expected. If a loop was authored to follow a transform and the transform is lost, the loop is removed.

If you want the sound to continue at the last known position, switch the loop to a fixed position before destroying the target:

```
PooledAudioManager.Instance.SetLoopPosition(handle, lastKnownPosition);
```

Stop Does Not Stop a One-Shot From SoundEventPlayer

Correct. One-shots are fire-and-forget. The player does not retain a handle to each one-shot source.

Use a loop if you need reliable stop control.

15. API Reference

PooledAudioManager Properties

```
public static PooledAudioManager Instance { get; }

public PooledAudioStats Stats { get; }
public bool IsMusicPlaying { get; }
public AudioClip CurrentMusicClip { get; }
```

PooledAudioManager One-Shot API

```
public bool Play(SoundEvent soundEvent);

public bool PlayAt(
    SoundEvent soundEvent,
    Transform target,
    bool force2D = false,
    bool mute = false);

public bool PlayFollowing(
    SoundEvent soundEvent,
    Transform target,
    bool force2D = false,
    bool mute = false);

public bool Play2D(
    SoundEvent soundEvent,
    bool mute = false);

public bool PlayUI(
    SoundEvent soundEvent,
    bool mute = false);

public bool PlayAt(
    SoundEvent soundEvent,
    Vector3 worldPosition,
    bool force2D = false,
    bool mute = false);

public bool PlayAtFollowing(
    SoundEvent soundEvent,
    Transform target,
```

```
bool force2D = false,  
bool mute = false);
```

Return value:

```
true   = playback request was accepted  
false  = request was rejected or could not play
```

Possible false reasons include missing event, missing clips, voice limit, distance culling, or pool limit.

PooledAudioManager Loop API

```
public PooledAudioHandle PlayLoop(  
    SoundEvent soundEvent,  
    Vector3 worldPosition,  
    bool force2D = false,  
    bool mute = false);  
  
public PooledAudioHandle PlayLoopAt(  
    SoundEvent soundEvent,  
    Transform target,  
    bool force2D = false,  
    bool mute = false);  
  
public PooledAudioHandle PlayLoopFollowing(  
    SoundEvent soundEvent,  
    Transform target,  
    bool force2D = false,  
    bool mute = false);  
  
public PooledAudioHandle PlayLoop2D(  
    SoundEvent soundEvent,  
    bool mute = false);  
  
public void StopLoop(PooledAudioHandle handle);  
public bool IsLoopActive(PooledAudioHandle handle);  
public bool IsLoopAudible(PooledAudioHandle handle);  
  
public void SetLoopPosition(PooledAudioHandle handle, Vector3 worldPosition);  
public void SetLoopFollowTarget(PooledAudioHandle handle, Transform target);  
public void SetLoopParams(PooledAudioHandle handle, float? volume = null, float? pitch = null);  
public void SetLoopMuted(PooledAudioHandle handle, bool muted);  
  
public void StopAllPooledAudio();
```

PooledAudioManager Music API

```
public void PlayMusic(SoundEvent musicEvent, float fadeTime = 1f);  
public void StopMusic(float fadeTime = 0.5f);  
public void SetMusicSilenced(bool silenced, float fadeTime = 0.25f);
```

PooledAudioManager Volume API

```
public bool SetMasterVolume(float value);  
public bool SetMusicVolume(float value);  
public bool SetSFXVolume(float value);  
public bool SetUIVolume(float value);
```

```

public bool SetGameplayVolume(float value);

public bool SetAmbienceVolume(float value);
public bool SetCinematicVolume(float value);
public bool SetFoleyVolume(float value);
public bool SetFootstepsVolume(float value);
public bool SetWeaponsVolume(float value);
public bool SetVoiceVolume(float value);
public bool SetVehiclesVolume(float value);
public bool SetBackgroundVolume(float value);
public bool SetEmittersVolume(float value);
public bool SetMenuVolume(float value);

public bool SetBusVolume(SoundBus bus, float value);

```

Values are expected in the `0..1` range.

PooledAudioManager Helpers

```

public void ResetDebugStats();

public AudioManagerGroup GetGroupForEvent(SoundEvent soundEvent);
public AudioManagerGroup GetGroupForBus(SoundBus bus);

public Transform GetListenerTransform();
public void SetListenerOverride(Transform listener);

```

16. SoundEventPlayer API Reference

Properties

```

public SoundEvent Event { get; set; }
public SoundEvent SoundEvent { get; set; }

public SoundEventPlayer.PlaybackMode Mode { get; set; }
public SoundEventPlayer.OutputMode Output { get; set; }

public bool Mute { get; set; }
public bool Muted { get; set; }

public PooledAudioHandle CurrentLoopHandle { get; }
public bool HasValidLoopHandle { get; }

public bool IsPlaying { get; }
public bool IsLoopAudible { get; }

public Transform AudioOrigin { get; }
public Vector3 Position { get; }

```

Playback

```

public bool Play();
public bool Play(SoundEvent overrideEvent);

public bool PlayOneShot();

```

```
public bool PlayOneShot(SoundEvent overrideEvent);

public bool PlayLoop();
public bool PlayLoop(SoundEvent overrideEvent);

public bool PlayMusic();
public bool PlayMusic(SoundEvent overrideEvent);

public bool PlayDelayed(float delay);
public bool PlayDelayed(SoundEvent overrideEvent, float delay);

public void Stop();
public bool Restart();
public bool Restart(SoundEvent overrideEvent);

public void Pause();
public bool UnPause();
public bool Resume();

public void Toggle();
```

UnityEvent / Animation Event Methods

```
public void PlayFromEvent();
public void StopFromEvent();
public void RestartFromEvent();
public void ToggleFromEvent();
public void PlayOneShotFromEvent();
public void PlayLoopFromEvent();

public void MuteFromEvent();
public void UnMuteFromEvent();
public void ToggleMuteFromEvent();
```

Runtime Settings

```
public void SetSoundEvent(SoundEvent nextEvent);

public void SetMute(bool value);
public void ToggleMute();

public void SetAudioOrigin(Transform origin);
public void ClearAudioOrigin();

public void SetForce2D(bool value);
public void SetFollowTransformOneShot(bool value);
public void SetFollowTransformWhileLooping(bool value);

public void SetOutputMode(SoundEventPlayer.OutputMode value);
public void SetPlaybackMode(SoundEventPlayer.PlaybackMode value);
```

Loop Control

```
public void SetLoopPosition(Vector3 worldPosition);
public void SetLoopFollowTarget(Transform target);

public void SetLoopParams(float volume, float pitch);
public void SetLoopVolume(float volume);
```

```
public void SetLoopPitch(float pitch);  
public void SetLoopMuted(bool muted);
```

17. Suggested Screenshot List

Use screenshots to make the manual feel premium and easy to follow.

Recommended screenshots:

1. Package folder layout.
2. Create menu for SoundEvent.
3. SoundEvent inspector overview.
4. SoundEvent clip / volume / pitch section.
5. SoundEvent 3D audio section.
6. SoundEvent distance layer setup.
7. GameObject menu for creating the PooledAudioManager.
8. PooledAudioManager inspector.
9. Mixer group assignments.
10. Runtime debug stats panel.
11. SoundEventPlayer inspector.
12. UnityEvent wired to `PlayFromEvent`.
13. Example scene showing positional emitter and listener distance.
14. Example of near/mid/far distance layer behavior.
15. Example settings menu using mixer volume sliders.

18. Quick Reference

Fastest Way to Play a Sound

```
PooledAudioManager.Instance.PlayAt(soundEvent, transform.position);
```

Fastest Way to Play UI

```
PooledAudioManager.Instance.PlayUI(soundEvent);
```

Fastest Way to Start and Stop a Loop

```
PooledAudioHandle handle =  
    PooledAudioManager.Instance.PlayLoopFollowing(loopEvent, transform);  
  
PooledAudioManager.Instance.StopLoop(handle);
```

Fastest Way to Play Music

```
PooledAudioManager.Instance.PlayMusic(musicEvent, 1f);
```

Fastest Designer-Friendly Setup

```
Add SoundEventListener
Assign SoundEvent
Choose Playback Mode
Choose Output Mode
Call PlayFromEvent from a UnityEvent
```

19. Final Notes

The Pooled Audio Event System is designed to stay out of your architecture. You do not need to change your character controller, UI system, save system, scene loader, or gameplay framework to use it.

Create one manager from **GameObject > STL Dynamics > Audio > Create Pooled Audio Manager**, create SoundEvents, then play them through code or the SoundEventListener component.

Use direct manager calls for gameplay systems. Use SoundEventListener for scene objects, buttons, triggers, and animation events.

That split keeps the workflow clean: powerful enough for programmers, simple enough for designers, and not bloated to death.