STL DYNAMICS

Quick Start

Pooled Audio Event System — Quick Start

This guide gets you from an empty scene to working pooled audio using the **PooledAudioManager**, **SoundEvent**, and **SoundEventPlayer** workflow.

You will use two playback paths:

1. **SoundEventPlayer components** — best for designers, prefabs, interactables, animation events, UI buttons, and scene objects.
2. **PooledAudioManager API calls** — best for gameplay code, procedural systems, spawned effects, damage events, and one-off audio triggers.

Most projects use both: SoundEventPlayers for object-owned audio, and manager calls for system-owned audio.

1. Create the PooledAudioManager

Before using any SoundEvents, create one **PooledAudioManager** in the active scene.

1. Open your scene.
2. In the Unity menu bar, choose **GameObject > STL Dynamics > Audio > Create Pooled Audio Manager**.
3. Select the created **PooledAudioManager** GameObject in the Hierarchy.
4. Press Play once to confirm the manager initializes without setup warnings.

Only one **PooledAudioManager** is usually needed per scene.

> Tip: Create the manager in your boot scene, persistent scene, or gameplay test scene depending on how your project loads levels.

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.

Part 1 — Getting Started with SoundEventPlayers

Use **SoundEventPlayer** when the sound belongs to a GameObject.

This is the easiest workflow for prefabs, props, weapons, interactables, animation events, UI buttons, doors, vehicles, characters, and environmental objects.

2. Create a SoundEvent

A **SoundEvent** stores the sound settings used by the pooling system.

1. In the Project window, right-click.
2. Choose **Create > STL Dynamics > Pooled Audio > Sound Event**.
3. Name it clearly, such as:

```
SE_Explosion_Large
SE_Button_Click
SE_Door_Open
SE_Gunshot
SE_Footstep_Concrete
```

4. Assign one or more AudioClips.
5. Configure the basics:
   - Volume
   - Pitch / random pitch
   - 2D or 3D behavior
   - Looping if needed
   - Voice limit if this sound can spam
   - Distance layers for larger 3D sounds
   - Mixer bus or optional mixer override

Start simple. Add one clip, press Play, then tune the advanced settings after the sound works.

---

3. Add a SoundEventPlayer

1. Select a GameObject.
2. Add the **SoundEventPlayer** component.
3. Assign your **SoundEvent**.
4. Choose the playback mode:
   - **OneShot** for normal fire-and-forget sounds.
   - **Loop** for engine hums, ambience emitters, machines, etc.
   - **Music** for music playback through the manager music system.
5. Choose the output mode:
   - **Positional** for 3D world audio.
   - **Global2D** for non-positional sounds.
   - **UI** for interface sounds.

Example setup:

```
Explosion prefab  -> SoundEventPlayer -> SE_Explosion_Large
Door prefab       -> SoundEventPlayer -> SE_Door_Open
UI button         -> SoundEventPlayer -> SE_Button_Click
Enemy prefab      -> SoundEventPlayer -> SE_Enemy_Hit
Vehicle prefab    -> SoundEventPlayer -> SE_Engine_Loop
```

For 3D sounds, place the GameObject where the sound should come from.

---

4. Basic SoundEventPlayer Script Example

```
using UnityEngine;
using STLDynamics.PooledAudio;

public class ExampleTriggerSound : MonoBehaviour
{
    [SerializeField] private SoundEventPlayer soundPlayer;

    private void OnMouseDown()
    {
        soundPlayer.Play();
    }
}
```

Use this for interactables, pickups, buttons, doors, weapons, and small gameplay objects.

5. Collision Sound Example

```
using UnityEngine;
using STLDynamics.PooledAudio;

public class ExampleCollisionSound : MonoBehaviour
{
    [SerializeField] private SoundEventPlayer impactSound;
    [SerializeField] private float minimumImpactSpeed = 2f;

    private void OnCollisionEnter(Collision collision)
    {
        if (collision.relativeVelocity.magnitude < minimumImpactSpeed)
            return;

        impactSound.PlayOneShot();
    }
}
```

Good for physics props, debris, vehicles, projectiles, and destructible objects.

6. Animation Event Example

For animation events, add a public method to a script on the animated object:

```
using UnityEngine;
using STLDynamics.PooledAudio;

public class ExampleAnimationAudio : MonoBehaviour
{
    [SerializeField] private SoundEventPlayer footstepSound;

    public void PlayFootstep()
    {
        footstepSound.PlayOneShot();
    }
}
```

Then add an Animation Event and call:

```
PlayFootstep
```

This works well for footsteps, attacks, reloads, impacts, and character actions.

Part 2 — Getting Started with PooledAudioManager

Use **PooledAudioManager** directly when audio comes from code instead of a specific scene object.

This is best for procedural systems, damage events, spawned effects, one-off world sounds, UI controllers, and gameplay managers.

---

7. Play a 2D Sound from Code

Use 2D playback for UI, notifications, rewards, errors, menu sounds, and non-positional feedback.

```
using UnityEngine;
using STLDynamics.PooledAudio;

public class ExamplePlay2DSound : MonoBehaviour
{
    [SerializeField] private SoundEvent buttonClickSound;

    public void OnButtonPressed()
    {
        PooledAudioManager.Instance.Play2D(buttonClickSound);
    }
}
```

For UI-specific sounds, you can also use:

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

---

8. Play a 3D Sound at a Position

Use positional playback when the sound should come from a world position.

```
using UnityEngine;
using STLDynamics.PooledAudio;

public class ExampleExplosionAudio : MonoBehaviour
{
    [SerializeField] private SoundEvent explosionSound;

    public void Explode(Vector3 position)
    {
        PooledAudioManager.Instance.PlayAt(explosionSound, position);
    }
}
```

Good for explosions, gunshots, impacts, animal sounds, vehicle sounds, destruction, and environmental events.

9. Play a Sound at a Transform

```
using UnityEngine;
using STLDynamics.PooledAudio;

public class ExampleTransformAudio : MonoBehaviour
{
    [SerializeField] private SoundEvent spawnSound;

    private void Start()
    {
        PooledAudioManager.Instance.PlayAt(spawnSound, transform);
    }
}
```

This plays the sound at the transform position. If the SoundEvent is configured to follow one-shots, the sound can follow the transform while it plays.

To force following from code:

```
PooledAudioManager.Instance.PlayFollowing(spawnSound, transform);
```

10. Play and Stop a Loop from Code

Loops return a **PooledAudioHandle** so they can be stopped or updated later.

```
using UnityEngine;
using STLDynamics.PooledAudio;

public class ExampleLoopAudio : MonoBehaviour
{
    [SerializeField] private SoundEvent engineLoop;

    private PooledAudioHandle loopHandle;

    private void OnEnable()
    {
        loopHandle = PooledAudioManager.Instance.PlayLoopFollowing(engineLoop, transform);
    }

    private void OnDisable()
    {
        PooledAudioManager.Instance.StopLoop(loopHandle);
    }
}
```

Use loops for engines, machines, ambience emitters, warning alarms, energy fields, and other continuous sounds.

11. Update a Loop at Runtime

```
using UnityEngine;
using STLDynamics.PooledAudio;

public class ExampleLoopControl : MonoBehaviour
{
    [SerializeField] private SoundEvent loopSound;

    private PooledAudioHandle loopHandle;

    private void Start()
    {
        loopHandle = PooledAudioManager.Instance.PlayLoop(loopSound, transform.position);
    }

    private void Update()
    {
        PooledAudioManager.Instance.SetLoopPosition(loopHandle, transform.position);
        PooledAudioManager.Instance.SetLoopParams(loopHandle, volume: 0.75f, pitch: 1.1f);
    }

    private void OnDestroy()
    {
        PooledAudioManager.Instance.StopLoop(loopHandle);
    }
}
```

For moving objects, **PlayLoopFollowing** or a **SoundEventPlayer** is usually cleaner.

Part 3 — Overrides

Overrides let you change what plays or how it plays without rebuilding the whole setup.

Use overrides when a prefab has a default sound, but gameplay sometimes needs a different sound.

Examples:

```
Same weapon prefab, different ammo sound
Same footstep player, different surface sound
Same UI button, different click/error/confirm sound
Same vehicle emitter, different engine loop
Same explosion prefab, different size variant
```

---

12. Play an Override SoundEvent

A **SoundEventPlayer** can play an override SoundEvent without permanently replacing its assigned default event.

```
using UnityEngine;
using STLDynamics.PooledAudio;

public class ExampleOverrideSound : MonoBehaviour
{
    [SerializeField] private SoundEventPlayer soundPlayer;
    [SerializeField] private SoundEvent defaultHitSound;
    [SerializeField] private SoundEvent criticalHitSound;
```

```
    public void PlayNormalHit()
    {
        soundPlayer.PlayOneShot(defaultHitSound);
    }

    public void PlayCriticalHit()
    {
        soundPlayer.PlayOneShot(criticalHitSound);
    }
}
```

The SoundEventPlayer keeps its assigned event. The override is only used for that call.

Supported override calls include:

```
soundPlayer.Play(overrideEvent);
soundPlayer.PlayOneShot(overrideEvent);
soundPlayer.PlayLoop(overrideEvent);
soundPlayer.PlayMusic(overrideEvent);
soundPlayer.PlayDelayed(overrideEvent, delay);
soundPlayer.Restart(overrideEvent);
```

13. Replace the Assigned SoundEvent at Runtime

Use **SetSoundEvent** when you want to permanently change what the player uses next.

```
using UnityEngine;
using STLDynamics.PooledAudio;

public class ExampleReplaceSound : MonoBehaviour
{
    [SerializeField] private SoundEventPlayer soundPlayer;
    [SerializeField] private SoundEvent woodFootstep;
    [SerializeField] private SoundEvent metalFootstep;

    public void SetWoodSurface()
    {
        soundPlayer.SetSoundEvent(woodFootstep);
    }

    public void SetMetalSurface()
    {
        soundPlayer.SetSoundEvent(metalFootstep);
    }

    public void PlayFootstep()
    {
        soundPlayer.PlayOneShot();
    }
}
```

Use temporary overrides for one-off changes. Use **SetSoundEvent** when the default should change.

14. Override Position, Output, and Mute Behavior

SoundEventPlayer also exposes runtime helpers for common playback changes.

```
using UnityEngine;
using STLDynamics.PooledAudio;

public class ExamplePlayerOverrides : MonoBehaviour
{
    [SerializeField] private SoundEventPlayer soundPlayer;
    [SerializeField] private Transform customOrigin;

    public void ConfigureAsUISound()
    {
        soundPlayer.SetOutputMode(SoundEventPlayer.OutputMode.UI);
        soundPlayer.SetForce2D(true);
    }

    public void UseCustomOrigin()
    {
        soundPlayer.SetAudioOrigin(customOrigin);
    }

    public void Mute(bool muted)
    {
        soundPlayer.SetMute(muted);
    }
}
```

Useful helpers:

```
soundPlayer.SetAudioOrigin(origin);
soundPlayer.ClearAudioOrigin();
soundPlayer.SetForce2D(true);
soundPlayer.SetOutputMode(SoundEventPlayer.OutputMode.Global2D);
soundPlayer.SetPlaybackMode(SoundEventPlayer.PlaybackMode.Loop);
soundPlayer.SetMute(true);
soundPlayer.ToggleMute();
```

15. Manager-Level Overrides

The manager API includes optional parameters for common one-off behavior.

```
// Play at a position but force 2D output after positional checks.
PooledAudioManager.Instance.PlayAt(soundEvent, transform.position, force2D: true);

// Play muted for this request only.
PooledAudioManager.Instance.PlayAt(soundEvent, transform.position, mute: true);

// Play a UI/global sound that ignores distance culling.
PooledAudioManager.Instance.PlayUI(soundEvent);

// Change the active listener used for distance checks.
PooledAudioManager.Instance.SetListenerOverride(listenerTransform);
```

For mixer routing, set the SoundEvent **Bus** or assign its optional **Override Group**. The override group takes priority over the default bus group.

Part 4 — Practical Settings

16. Voice Limits and Spam Protection

If a sound can play many times quickly, configure a voice limit on the **SoundEvent**.

Good candidates:

```
Bullet impacts
Footsteps
Debris collisions
UI clicks
Enemy hit sounds
Coin pickups
Explosion chains
```

Recommended starting values:

```
UI click:        2-4 voices
Footstep:        4-8 voices
Small impact:    6-12 voices
Large explosion: 2-4 voices
Gunshot:         4-10 voices
```

Tune by ear. The goal is controlled chaos, not 97 rocks all screaming at once.

---

17. Distance Layers for 3D Sounds

Distance layers allow a SoundEvent to use different clips based on listener distance.

Example:

```
Gunshot near clip -> close range
Gunshot mid clip  -> medium range
Gunshot far clip  -> far range
```

Use distance layers for large, noticeable sounds:

```
Gunshots
Explosions
Thunder
Vehicle crashes
Large creature sounds
Destruction
```

For small sounds like UI clicks or item pickups, simple 2D or simple 3D playback is usually enough.

---

18. Common Setup Checklist

If no sound plays, check these first:

- A **PooledAudioManager** exists in the scene. Create one from **GameObject > STL Dynamics > Audio > Create Pooled Audio Manager**.
- The **SoundEvent** has at least one assigned AudioClip.
- The AudioClip volume is not silent.
- The SoundEvent volume is above 0.
- The scene has an AudioListener.
- 3D sounds are close enough to the listener.
- The SoundEventPlayer references the correct SoundEvent.
- The sound is not blocked by a strict voice limit.
- The GameObject calling Play is active.
- The correct output mode is selected: Positional, Global2D, or UI.
- The sound is not muted by the SoundEvent, SoundEventPlayer, or request.

Most setup issues are caused by a missing manager, missing clip, wrong 2D/3D setting, distance falloff, or voice limits.

---

19. Recommended First Test

To confirm the system is working:

1. Use the manager created in Section 1.
2. Create **SE_Test_Click**.
3. Assign one short AudioClip.
4. Set the sound to 2D or use a Global2D/UI output mode.
5. Add **SoundEventPlayer** to any GameObject.
6. Assign **SE_Test_Click**.
7. Enable Play On Awake or call `Play()` from script.
8. Press Play.

After that works, test a positional sound with:

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

---

20. Minimal API Reference

Common SoundEventPlayer calls:

```
soundEventPlayer.Play();
soundEventPlayer.PlayOneShot();
soundEventPlayer.PlayLoop();
soundEventPlayer.PlayMusic();
soundEventPlayer.PlayDelayed(0.5f);
soundEventPlayer.Stop();
soundEventPlayer.Restart();
soundEventPlayer.Toggle();
soundEventPlayer.SetMute(true);
```

Common PooledAudioManager calls:

```
PooledAudioManager.Instance.Play(soundEvent);
PooledAudioManager.Instance.Play2D(soundEvent);
PooledAudioManager.Instance.PlayUI(soundEvent);
```

```
PooledAudioManager.Instance.PlayAt(soundEvent, worldPosition);
PooledAudioManager.Instance.PlayAt(soundEvent, targetTransform);
PooledAudioManager.Instance.PlayFollowing(soundEvent, targetTransform);

PooledAudioHandle loop = PooledAudioManager.Instance.PlayLoop(soundEvent, worldPosition);
PooledAudioManager.Instance.StopLoop(handle);
PooledAudioManager.Instance.SetLoopPosition(handle, worldPosition);
PooledAudioManager.Instance.SetLoopFollowTarget(handle, targetTransform);
PooledAudioManager.Instance.SetLoopParams(handle, volume: 0.8f, pitch: 1.05f);
PooledAudioManager.Instance.SetLoopMuted(handle, true);

PooledAudioManager.Instance.PlayMusic(musicEvent, fadeTime: 1f);
PooledAudioManager.Instance.StopMusic(fadeTime: 0.5f);
```

21. Quick Rule

Use **SoundEventPlayer** when the sound lives on an object.

Use **PooledAudioManager** when the sound comes from a system.

Pass an override SoundEvent for one-time variations. Call **SetSoundEvent** when the default sound should change.

That rule gets you most of the way there.