SFX System - User's Guide

Quick Start and Practical Examples

Version: 1.0 **Last Updated:** 2025-10-19 **Difficulty:** Beginner to Intermediate

---

Table of Contents

---

1. Introduction

What is the SFX System?

The SFX System is a professional audio solution for Unity that makes sound implementation easy and powerful. Instead of manually managing AudioSources, you'll create audio **Events** that play **Containers** through **Buses**.

Why Use This System?

Before (Standard Unity):

```
public AudioClip footstepClip;
AudioSource.PlayClipAtPoint(footstepClip, transform.position);
```

Problems: - No volume control - No variations - No pooling (performance issues) - Hard to manage

After (SFX System):

```
public AudioEvent footstepEvent;
footstepEvent.Post(gameObject, transform.position);
```

Benefits: - Volume control via buses - Built-in variations - Automatic pooling - Easy to manage

Key Features You'll Love

- **One-click sound setup** - Create containers and events in the Unity Editor
- **Automatic variations** - No more repeating sounds
- **Volume control** - Adjust SFX, music, and dialogue separately
- **3D spatial audio** - Sounds correctly positioned in 3D space
- **Performance** - Automatically manages memory and CPU usage

---

2. Getting Started in 10 Minutes

Step 1: Setup AudioManager (2 minutes)

1. Create a new empty GameObject in your scene
2. Name it "AudioManager"
3. Add the `AudioManager` component from `AudioSystem` namespace
4. The AudioManager will persist between scenes automatically

Inspector Settings (defaults are fine for now): - Master Volume: 1.0 - Max Real Voices: 32 - Max Virtual Voices: 64

Step 2: Create Folder Structure (1 minute)

```
Assets/Audio/
├── Resources/
│   └── Audio/
│       ├── Events/     ← Create this (audio events load from here)
│       └── States/     ← Create this (states load from here)
├── Containers/         ← Create this (container assets)
├── Buses/              ← Create this (bus assets)
└── AudioClips/         ← Your actual .wav/.mp3 files
    ├── SFX/
    ├── Music/
    └── Ambience/
```

Step 3: Create Your First Bus (2 minutes)

1. Right-click in `Assets/Audio/Buses/`
2. Select `Create > Audio System > Audio Bus`
3. Name it "SFX_Bus"
4. In Inspector:
    - Bus Name: "SFX"
    - Volume Db: 0

Step 4: Create Your First Container (2 minutes)

1. Right-click in `Assets/Audio/Containers/`
2. Select `Create > Audio System > Routing Container`
3. Name it "TestSound_RC"
4. In Inspector:
    - Container Name: "TestSound"
    - Audio Clips: Drag in an AudioClip
    - Volume: 1.0

Step 5: Create Your First Event (2 minutes)

1. Right-click in `Assets/Audio/Resources/Audio/Events/`
2. Select `Create > Audio System > Audio Event`

3. Name it "Play_TestSound"
4. In Inspector:
    - Event Name: "Play_TestSound"
    - Actions → Size: 1
    - Action [0]:
        - Type: Play
        - Container: TestSound_RC
        - Target Bus: SFX_Bus

Step 6: Play the Sound (1 minute)

Create a test script:

```
using UnityEngine;
using AudioSystem;

public class TestAudio : MonoBehaviour
{
    [SerializeField] private AudioEvent testEvent;

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            testEvent.Post(gameObject, transform.position);
            Debug.Log("Playing test sound!");
        }
    }
}
```

1. Attach script to a GameObject
2. Drag the `Play_TestSound` event into the script's `Test Event` field
3. Press Play in Unity
4. Press Space to hear your sound!

Congratulations! You've set up the complete audio pipeline.

---

3. Basic Concepts

The Audio Chain

```
Your Code → AudioEvent → Container → Voice → Bus → Speaker
```

1. **Your Code**: Triggers an audio event
2. **AudioEvent**: Defines what happens (play, stop, change volume, etc.)
3. **Container**: Organizes audio clips (random, sequence, switch, etc.)
4. **Voice**: The actual playing sound instance
5. **Bus**: Volume and mixing control
6. **Speaker**: Final output

The Three Core Assets

1. Containers (How sounds are organized)

- **RoutingContainer**: Simple playback, one or more clips simultaneously
- **RandomContainer**: Picks one random clip per play
- **SequenceContainer**: Plays clips in order
- **SwitchContainer**: Switches between different containers based on game state
- **BlendContainer**: Plays multiple containers with crossfading

2. Events (When/how to play)

- Triggers containers
- Can execute multiple actions (play, stop, fade, etc.)
- Has priority and instance limits

3. Buses (Volume and mixing)

- Hierarchical mixing
- Group controls (mute all SFX, lower music, etc.)
- Ducking (music quiets when dialogue plays)

Important Terminology

Term	Simple Explanation
Post an Event	Trigger a sound
Container	A group of audio clips
Voice	One sound that's currently playing
Bus	Volume control group (like a mixing board channel)
RTPC	Real-Time Parameter (dynamically change volume, pitch, etc.)
State	Game mode that changes audio (underwater, combat, menu)
Switch	Choose different sounds based on surface, character, etc.

---

4. Tutorial: Your First Sound

Let's create a simple button click sound.

Part A: Create the Container

1. Right-click `Assets/Audio/Containers/`
2. `Create > Audio System > Routing Container`
3. Name: "UI_ButtonClick_RC"
4. In Inspector:
    - Container Name: "UI_ButtonClick"
    - Audio Clips: Add your button click sound(s)
    - Volume: 0.8
    - Enable Volume Randomization: ☑
        - Volume Random Min: -1
        - Volume Random Max: 1

    - Enable Pitch Randomization: ☑
        - Pitch Random Min: -50
        - Pitch Random Max: 50

This adds subtle variation so repeated clicks don't sound identical.

Part B: Create the Bus

1. Right-click `Assets/Audio/Buses/`
2. `Create > Audio System > Audio Bus`
3. Name: "UI_Bus"
4. In Inspector:
    - Bus Name: "UI"
    - Volume Db: 0

Part C: Create the Event

1. Right-click `Assets/Audio/Resources/Audio/Events/`
2. `Create > Audio System > Audio Event`
3. Name: "Play_UI_ButtonClick"
4. In Inspector:
    - Event Name: "Play_UI_ButtonClick"
    - Priority: High
    - Max Instances: 3 (prevents spam)
    - Cooldown: 0.1
    - Actions → Size: 1
    - Action [0]:
        - Type: Play
        - Container: UI_ButtonClick_RC
        - Target Bus: UI_Bus

Part D: Hook Up to UI Button

```
using UnityEngine;
using UnityEngine.UI;
using AudioSystem;

public class ButtonSound : MonoBehaviour
{
    [SerializeField] private AudioEvent clickEvent;

    void Start()
    {
        GetComponent<Button>().onClick.AddListener(OnButtonClick);
    }

    void OnButtonClick()
    {
        clickEvent.Post();
    }
}
```

1. Attach this script to your Button GameObject
2. Drag "Play_UI_ButtonClick" into the Click Event field
3. Click the button and hear your sound!

---

5. Tutorial: Footsteps with Variations

Let's create a realistic footstep system with different surfaces.

Part A: Create Random Containers for Each Surface

Grass Footsteps

1. `Create > Audio System > Random Container`
2. Name: "Footsteps_Grass_RnC"
3. In Inspector:
    - Audio Clips: Add 4-6 grass footstep sounds
    - Set each Weight: 1.0

- Avoid Repeat Last: 2 (won't repeat last 2 sounds)
- Use Weighting: ☑
- Volume: 0.6
- Enable Volume Randomization: ☑ (±2 dB)
- Enable Pitch Randomization: ☑ (±100 cents)

Metal Footsteps

1. Repeat above steps
2. Name: "Footsteps_Metal_RnC"
3. Add metal footstep clips
4. Volume: 0.8 (metal is louder)

Wood Footsteps

1. Repeat above steps
2. Name: "Footsteps_Wood_RnC"
3. Add wood footstep clips
4. Volume: 0.7

Part B: Create Switch Container

1. `Create > Audio System > Switch Container`
2. Name: "Footsteps_Switch_SwC"
3. In Inspector:
    - Switch Group Name: "Surface_Type"
    - Switch Entries → Size: 3
        - [0] Switch Value: "Grass" → Container: Footsteps_Grass_RnC
        - [1] Switch Value: "Metal" → Container: Footsteps_Metal_RnC
        - [2] Switch Value: "Wood" → Container: Footsteps_Wood_RnC
    - Default Container: Footsteps_Grass_RnC
    - Volume: 1.0
    - Is3D: ☑
    - Min Distance: 1

- Max Distance: 15

Part C: Create the Event

1. `Create > Audio System > Audio Event`
2. Name: "Play_Footstep"
3. Move to `Resources/Audio/Events/`
4. In Inspector:
    - Event Name: "Play_Footstep"
    - Priority: Low
    - Max Instances: 4
    - Actions → Size: 1
    - Action [0]:
        - Type: Play
        - Container: Footsteps_Switch_SwC
        - Target Bus: SFX_Bus

Part D: Player Script

```
using UnityEngine;
using AudioSystem;

public class PlayerFootsteps : MonoBehaviour
{
    [SerializeField] private AudioEvent footstepEvent;
    [SerializeField] private LayerMask groundLayers;

    // Call this from your animation events or at fixed intervals
    public void PlayFootstep()
    {
        // Detect surface type
        string surface = GetSurfaceType();

        // Set the switch value
        AudioManager.Instance.SetSwitch("Surface_Type", surface);

        // Play the footstep
        footstepEvent.Post(gameObject, transform.position);
    }

    private string GetSurfaceType()
    {
        RaycastHit hit;
        if (Physics.Raycast(transform.position, Vector3.down, out hit, 2f, groundLayers))
        {
            // Check by tag
            if (hit.collider.CompareTag("Metal")) return "Metal";
            if (hit.collider.CompareTag("Wood")) return "Wood";

            // Or check by material name
            if (hit.collider.sharedMaterial != null)
            {
                string matName = hit.collider.sharedMaterial.name.ToLower();
                if (matName.Contains("metal")) return "Metal";
                if (matName.Contains("wood")) return "Wood";
            }
        }

        return "Grass"; // Default
    }
}
```

Part E: Animation Event Setup

1. Open your walk/run animation in the Animation window
2. At each foot contact frame, add an Animation Event
3. Set Function: `PlayFootstep`
4. Done! Footsteps will play automatically with correct surface sounds

---

6. Tutorial: Background Music

Let's set up looping background music with intro and loop sections.

Part A: Create Music Containers

Music Intro

1. `Create > Audio System > Routing Container`
2. Name: "Music_MainTheme_Intro_RC"
3. In Inspector:
    - Audio Clips: Your intro clip
    - Volume: 1.0
    - Loop: ☐ (intro plays once)
    - Is3D: ☐ (music is 2D)

Music Loop

1. `Create > Audio System > Routing Container`
2. Name: "Music_MainTheme_Loop_RC"
3. In Inspector:
    - Audio Clips: Your looping music
    - Volume: 1.0
    - Loop: ☑ (loops forever)
    - Is3D: ☐

Part B: Create Music Bus

1. `Create > Audio System > Audio Bus`
2. Name: "Music_Bus"
3. In Inspector:
    - Bus Name: "Music"
    - Volume Db: -6 (leave headroom)

Part C: Create Events

Start Music Event

1. `Create > Audio System > Audio Event`
2. Name: "Music_Start_MainTheme"
3. In Inspector:
    - Event Name: "Music_Start_MainTheme"
    - Priority: Medium
    - Max Instances: 1
    - Actions → Size: 2
    - Action [0]:
        - Type: Play
        - Container: Music_MainTheme_Intro_RC
        - Target Bus: Music_Bus
        - Delay: 0
    - Action [1]:
        - Type: Play
        - Container: Music_MainTheme_Loop_RC
        - Target Bus: Music_Bus
        - Delay: 8.5 (length of intro in seconds)

Stop Music Event

1. `Create > Audio System > Audio Event`

2. Name: "Music_Stop_All"
3. Actions → Size: 1
    - Type: Stop
    - Container: Music_MainTheme_Loop_RC
    - Fade Duration: 2.0

Part D: Music Manager Script

```
using UnityEngine;
using AudioSystem;

public class MusicManager : MonoBehaviour
{
    [SerializeField] private AudioEvent startMusicEvent;
    [SerializeField] private AudioEvent stopMusicEvent;

    void Start()
    {
        // Start music when scene loads
        startMusicEvent.Post();
    }

    void OnDestroy()
    {
        // Fade out when scene unloads
        stopMusicEvent?.Post();
    }

    // Optional: Control from other scripts
    public void SetMusicVolume(float volume01)
    {
        float volumeDb = AudioExtensions.LinearToDb(volume01);
        AudioManager.Instance.SetBusVolume("Music", volumeDb, 0.5f);
    }
}
```

---

7. Tutorial: 3D Spatial Audio

Let's create a fire crackling sound that exists in 3D space.

Part A: Create 3D Container

1. `Create > Audio System > Random Container`
2. Name: "Fire_Crackle_RnC"
3. In Inspector:
    - Audio Clips: Add 3-5 fire crackling sounds
    - Avoid Repeat Last: 2
    - Volume: 0.7
    - Loop: ☑
    - **Is3D:** ☑ ← Important!
    - Min Distance: 2
    - Max Distance: 20
    - Rolloff Mode: Logarithmic

Part B: Create Event

1. `Create > Audio System > Audio Event`
2. Name: "Play_Fire_Crackle"
3. In Inspector:
    - Event Name: "Play_Fire_Crackle"
    - Priority: Low
    - Max Instances: 10 (allow multiple fires)
    - Actions → Size: 1
    - Action [0]:
        - Type: Play
        - Container: Fire_Crackle_RnC
        - Target Bus: SFX_Bus

Part C: Fire Script

```
using UnityEngine;
using AudioSystem;

public class FireSound : MonoBehaviour
{
    [SerializeField] private AudioEvent fireEvent;
    private AudioHandle fireHandle;

    void OnEnable()
    {
        // Start fire sound at this object's position
        fireHandle = fireEvent.Post(gameObject, transform.position);
    }

    void OnDisable()
    {
        // Stop fire sound with fade out
        fireHandle?.Stop(1f);
    }

    // Optional: Adjust volume based on fire intensity
    public void SetIntensity(float intensity)
    {
        if (fireHandle != null)
        {
            fireHandle.SetVolume(intensity);
        }
    }
}
```

Part D: Test It!

1. Attach FireSound script to a GameObject (e.g., campfire model)
2. Drag in the Play_Fire_Crackle event
3. Press Play and walk around the fire
4. You'll hear the sound get louder as you approach and quieter as you move away
5. The sound will also pan left/right based on position

---

8. Tutorial: Dynamic Music System

Create an adaptive music system that responds to combat intensity.

Part A: Create Layered Music Containers

Create three music containers with different intensity levels:

1. **Music_Ambient_RC**: Calm exploration music (loop)
2. **Music_Tension_RC**: Tense percussion layer (loop)
3. **Music_Combat_RC**: Full combat music (loop)

Part B: Create Blend Container

1. `Create > Audio System > Blend Container`
2. Name: "Music_Dynamic_BC"
3. In Inspector:
    - Blend Parameter Name: "CombatIntensity"
    - Blend Entries → Size: 3

    Entry [0]: Ambient Layer

    - Container: Music_Ambient_RC
    - Volume Curve:
        - Key at (0, 1): Full volume at 0 intensity
        - Key at (1, 0): Silent at max intensity

    Entry [1]: Tension Layer

    - Container: Music_Tension_RC
    - Volume Curve:
        - Key at (0, 0): Silent at 0 intensity
        - Key at (0.5, 1): Full volume at medium intensity
        - Key at (1, 0): Silent at max intensity

Entry [2]: Combat Layer

- Container: Music_Combat_RC
- Volume Curve:
  - Key at (0, 0): Silent at 0 intensity
  - Key at (1, 1): Full volume at max intensity
- Volume: 1.0
- Loop: ☑

Part C: Create Event

1. `Create > Audio System > Audio Event`
2. Name: "Start_DynamicMusic"
3. Actions → Size: 1
   - Type: Play
   - Container: Music_Dynamic_BC
   - Target Bus: Music_Bus

Part D: Combat Music Controller

```
using UnityEngine;
using AudioSystem;
using System.Collections.Generic;

public class DynamicMusicController : MonoBehaviour
{
    [SerializeField] private AudioEvent startMusicEvent;
    [SerializeField] private float transitionSpeed = 0.5f;
    [SerializeField] private float enemyDetectionRadius = 20f;
    [SerializeField] private LayerMask enemyLayer;

    private float targetIntensity = 0f;
    private float currentIntensity = 0f;

    void Start()
    {
        // Start the layered music
        startMusicEvent.Post();
        AudioManager.Instance.SetRTPC("CombatIntensity", 0f);
    }

    void Update()
    {
        // Detect nearby enemies
        int enemyCount = CountNearbyEnemies();

        // Calculate target intensity
        if (enemyCount == 0)
            targetIntensity = 0f;          // Ambient only
        else if (enemyCount <= 2)
            targetIntensity = 0.5f;        // Tension layer
        else
            targetIntensity = 1f;          // Full combat

        // Smooth transition
        if (Mathf.Abs(currentIntensity - targetIntensity) > 0.01f)
        {
            currentIntensity = Mathf.MoveTowards(
                currentIntensity,
                targetIntensity,
                transitionSpeed * Time.deltaTime
            );

            AudioManager.Instance.SetRTPC("CombatIntensity", currentIntensity);
        }
    }

    int CountNearbyEnemies()
    {
        Collider[] enemies = Physics.OverlapSphere(
```

```
            transform.position,
            enemyDetectionRadius,
            enemyLayer
        );
        return enemies.Length;
    }

    void OnDrawGizmos()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position, enemyDetectionRadius);
    }
}
```

Part E: Test It!

1. Attach the script to your player
2. Configure enemy detection settings
3. As enemies approach, music seamlessly transitions from ambient to tense to combat
4. Music layers crossfade smoothly based on threat level

---

9. Tutorial: Underwater Effects

Create an underwater state that modifies all audio.

Part A: Create Buses

Make sure you have these buses: - **Master_Bus**: Root bus (0 dB) - **SFX_Bus**: Child of Master (-3 dB) - **Music_Bus**: Child of Master (-6 dB) - **Ambience_Bus**: Child of Master (-6 dB)

Part B: Create Audio States

Normal State

1. `Create > Audio System > Audio State`
2. Name: "State_Normal"
3. Move to `Resources/Audio/States/`

4. In Inspector:
    - State Name: "Normal"
    - State Group: "Location"
    - (All volumes at default)

Underwater State

1. `Create > Audio System > Audio State`
2. Name: "State_Underwater"
3. Move to `Resources/Audio/States/`
4. In Inspector:
    - State Name: "Underwater"
    - State Group: "Location"
    - Bus Volumes → Size: 3
        - [0] Bus: SFX_Bus, Volume Db: -9
        - [1] Bus: Music_Bus, Volume Db: -12
        - [2] Bus: Ambience_Bus, Volume Db: -6
    - RTPC Values → Size: 1
        - [0] Parameter Name: "Underwater", Value: 1.0

Part C: Create Underwater Ambience

1. `Create > Audio System > Routing Container`
2. Name: "Ambience_Underwater_RC"
3. In Inspector:
    - Audio Clips: Underwater ambient sounds
    - Volume: 0.8
    - Loop: ☑
    - Is3D: ☐

Part D: Create Switch Container for Sounds

This allows sounds to have underwater variants.

1. `Create > Audio System > Switch Container`
2. Name: "Explosions_Switch_SwC"
3. In Inspector:
    - Switch Group Name: "Location"
    - Switch Entries → Size: 2
        - [0] Switch Value: "Normal" → Explosion_Normal_RC
        - [1] Switch Value: "Underwater" → Explosion_Muffled_RC
    - Default Container: Explosion_Normal_RC

Part E: Underwater Volume Script

```
using UnityEngine;
using AudioSystem;

public class UnderwaterVolume : MonoBehaviour
{
    [SerializeField] private AudioEvent underwaterAmbienceEvent;
    [SerializeField] private float transitionTime = 1.5f;

    private AudioHandle ambienceHandle;
    private bool isUnderwater = false;

    void OnTriggerEnter(Collider other)
    {
        if (!other.CompareTag("Player")) return;

        if (!isUnderwater)
        {
            EnterUnderwater();
        }
    }

    void OnTriggerExit(Collider other)
    {
        if (!other.CompareTag("Player")) return;

        if (isUnderwater)
        {
            ExitUnderwater();
        }
    }

    void EnterUnderwater()
    {
        isUnderwater = true;

        // Activate underwater state (affects all audio)
        AudioManager.Instance.SetState("Underwater", transitionTime);

        // Set switch for sound variants
        AudioManager.Instance.SetSwitch("Location", "Underwater");

        // Play underwater ambience
        ambienceHandle = underwaterAmbienceEvent.Post();

        Debug.Log("Entered underwater");
    }

    void ExitUnderwater()
    {
        isUnderwater = false;
```

```
        // Return to normal state
        AudioManager.Instance.SetState("Normal", transitionTime);

        // Set switch back to normal
        AudioManager.Instance.SetSwitch("Location", "Normal");

        // Stop underwater ambience
        ambienceHandle?.Stop(transitionTime);

        Debug.Log("Exited underwater");
    }
}
```

Part F: Setup in Scene

1. Create a large Box Collider for your underwater area
2. Check "Is Trigger"
3. Attach the UnderwaterVolume script
4. Assign the underwater ambience event
5. Test by swimming in and out - all audio will smoothly transition!

---

10. Common Recipes

Recipe 1: Simple Settings Menu

```
using UnityEngine;
using UnityEngine.UI;
using AudioSystem;

public class AudioSettings : MonoBehaviour
{
    [SerializeField] private Slider masterSlider;
    [SerializeField] private Slider sfxSlider;
    [SerializeField] private Slider musicSlider;

    void Start()
    {
        // Load saved settings
        masterSlider.value = PlayerPrefs.GetFloat("MasterVolume", 1f);
        sfxSlider.value = PlayerPrefs.GetFloat("SFXVolume", 1f);
        musicSlider.value = PlayerPrefs.GetFloat("MusicVolume", 1f);

        // Apply settings
        OnMasterVolumeChanged(masterSlider.value);
        OnSFXVolumeChanged(sfxSlider.value);
        OnMusicVolumeChanged(musicSlider.value);

        // Add listeners
        masterSlider.onValueChanged.AddListener(OnMasterVolumeChanged);
        sfxSlider.onValueChanged.AddListener(OnSFXVolumeChanged);
        musicSlider.onValueChanged.AddListener(OnMusicVolumeChanged);
    }

    void OnMasterVolumeChanged(float value)
    {
        AudioManager.Instance.SetMasterVolume(value);
        PlayerPrefs.SetFloat("MasterVolume", value);
    }

    void OnSFXVolumeChanged(float value)
    {
        float db = AudioExtensions.LinearToDb(value);
        AudioManager.Instance.SetBusVolume("SFX", db, 0.1f);
        PlayerPrefs.SetFloat("SFXVolume", value);
    }

    void OnMusicVolumeChanged(float value)
    {
        float db = AudioExtensions.LinearToDb(value);
        AudioManager.Instance.SetBusVolume("Music", db, 0.1f);
        PlayerPrefs.SetFloat("MusicVolume", value);
    }
}
```

Recipe 2: Vehicle Engine Sound

```
using UnityEngine;
using AudioSystem;

[RequireComponent(typeof(Rigidbody))]
public class VehicleEngineAudio : MonoBehaviour
{
    [SerializeField] private AudioEvent engineStartEvent;
    [SerializeField] private AudioEvent engineStopEvent;
    [SerializeField] private BlendContainer engineBlend;

    [Header("Settings")]
    [SerializeField] private float maxSpeed = 30f;
    [SerializeField] private float rpmTransitionSpeed = 3f;

    private Rigidbody rb;
    private AudioHandle engineHandle;
    private float currentRPM = 0f;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
        StartEngine();
    }

    void StartEngine()
    {
        engineHandle = engineStartEvent.Post(gameObject, transform.position);
    }

    void Update()
    {
        // Calculate RPM from speed
        float speed = rb.velocity.magnitude;
        float targetRPM = Mathf.Clamp01(speed / maxSpeed);

        // Smooth transition
        currentRPM = Mathf.Lerp(currentRPM, targetRPM, rpmTransitionSpeed * Time.deltaTime);

        // Update engine blend
        AudioManager.Instance.SetRTPC("EngineRPM", currentRPM);
    }

    void OnDestroy()
    {
        engineHandle?.Stop(0.5f);
    }
}
```

Recipe 3: Pause Menu

```
using UnityEngine;
using AudioSystem;

public class PauseMenu : MonoBehaviour
{
    private bool isPaused = false;

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (isPaused)
                Resume();
            else
                Pause();
        }
    }

    void Pause()
    {
        isPaused = true;
        Time.timeScale = 0f;

        // Activate pause state (muffles SFX, keeps UI sounds)
        AudioManager.Instance.SetState("Paused", 0.3f);

        // Or manually pause all SFX
        // AudioManager.Instance.PauseAll();
    }

    void Resume()
    {
        isPaused = false;
        Time.timeScale = 1f;

        // Return to playing state
        AudioManager.Instance.SetState("Playing", 0.3f);

        // Or manually unpause
        // AudioManager.Instance.UnpauseAll();
    }
}
```

Recipe 4: Footstep Timer (alternative to animation events)

```
using UnityEngine;
using AudioSystem;

public class FootstepTimer : MonoBehaviour
{
    [SerializeField] private AudioEvent footstepEvent;
    [SerializeField] private float walkStepInterval = 0.5f;
    [SerializeField] private float runStepInterval = 0.3f;
    [SerializeField] private float minSpeedForFootsteps = 0.1f;

    private CharacterController characterController;
    private float stepTimer = 0f;

    void Start()
    {
        characterController = GetComponent<CharacterController>();
    }

    void Update()
    {
        float speed = characterController.velocity.magnitude;

        if (speed > minSpeedForFootsteps && characterController.isGrounded)
        {
            // Determine step interval based on speed
            bool isRunning = speed > 5f;
            float interval = isRunning ? runStepInterval : walkStepInterval;

            stepTimer -= Time.deltaTime;

            if (stepTimer <= 0f)
            {
                PlayFootstep();
                stepTimer = interval;
            }
        }
        else
        {
            stepTimer = 0f;
        }
    }

    void PlayFootstep()
    {
        footstepEvent.Post(gameObject, transform.position);
    }
}
```

Recipe 5: Health-Based Heartbeat

```
using UnityEngine;
using AudioSystem;

public class HeartbeatAudio : MonoBehaviour
{
    [SerializeField] private AudioEvent heartbeatEvent;
    [SerializeField] private float lowHealthThreshold = 0.3f;

    private AudioHandle heartbeatHandle;
    private bool isPlaying = false;

    public void OnHealthChanged(float currentHealth, float maxHealth)
    {
        float healthPercent = currentHealth / maxHealth;

        if (healthPercent <= lowHealthThreshold && !isPlaying)
        {
            // Start heartbeat
            heartbeatHandle = heartbeatEvent.Post();
            isPlaying = true;
        }
        else if (healthPercent > lowHealthThreshold && isPlaying)
        {
            // Stop heartbeat
            heartbeatHandle?.Stop(1f);
            isPlaying = false;
        }

        // Adjust heartbeat speed based on health
        if (isPlaying)
        {
            float intensity = 1f - (healthPercent / lowHealthThreshold);
            AudioManager.Instance.SetRTPC("HeartbeatIntensity", intensity);
        }
    }
}
```

---

11. Tips and Tricks

Audio Design Tips

1. **Always Add Variation**
    - Use RandomContainer for sounds that repeat (footsteps, gunshots)
    - Enable pitch and volume randomization
    - Set "Avoid Repeat Last" to 2-3
2. **Leave Headroom**
    - Master bus: -3 to -6 dB
    - Individual buses: Don't max out at 0 dB
    - Prevents clipping when multiple sounds play
3. **Use Priorities Wisely**
    - Critical: UI feedback, dialogue
    - High: Player actions, important events
    - Medium: Regular gameplay
    - Low: Ambience, distant sounds
4. **Optimize for 3D**
    - Only use 3D for positioned sounds
    - Adjust Min/Max distances appropriately
    - Use Logarithmic rolloff for realistic sound
5. **Ducking for Clarity**
    - Duck music when dialogue plays
    - Duck SFX during important voiceover
    - Use short attack (0.05s) and longer release (0.5s)

Performance Tips

1. **Limit Instance Counts**

```
Explosion: maxInstances = 5
Footstep: maxInstances = 4
Gunshot: maxInstances = 10
```

2. **Use Voice Virtualization**
    - Enable LOD in AudioManager
    - Set appropriate real/virtual voice limits
    - Low-priority distant sounds will virtualize automatically
3. **Compress Audio Files**
    - UI sounds: PCM (uncompressed) - small files
    - SFX: Compressed in memory
    - Music: Streaming
4. **Don't Abuse Events**
    - Don't post events every frame
    - Use cooldown timers
    - Batch similar sounds

Organization Tips

1. **Naming Convention**

```
Containers: Category_Description_Type
Example: Weapon_RifleShot_RnC

Events: Action_Description
Example: Play_WeaponRifleShot

Buses: Category_Bus
Example: SFX_Weapons_Bus
```

2. **Folder Structure**

```
Containers/
├── UI/
├── Weapons/
├── Footsteps/
├── Ambience/
└── Music/
```

3. **Use Descriptive Names**
   - ✅ "Play_Player_Footstep_Run"
   - ❌ "Event1", "Sound2"

Debugging Tips

1. **Check Voice Count**

```
void OnGUI()
{
    var stats = AudioManager.Instance.GetStatistics();
    GUI.Label(new Rect(10, 10, 300, 20),
        $"Voices: {stats.activeVoices}/{stats.totalVoices}");
}
```

2. **Log Events**

```
AudioHandle handle = myEvent.Post();
handle.OnStarted += () => Debug.Log("Sound started");
handle.OnFinished += () => Debug.Log("Sound finished");
```

3. **Visualize 3D Sounds**

```
void OnDrawGizmos()
{
    // Draw sound radius
    Gizmos.color = Color.yellow;
    Gizmos.DrawWireSphere(transform.position, maxDistance);
}
```

---

12. FAQ

Setup Questions

Q: Do I need to put AudioManager in every scene? A: No! AudioManager uses `DontDestroyOnLoad`, so it persists between scenes. Add it once to your first scene.

Q: Why must events be in Resources/Audio/Events/? A: The AudioManager automatically loads events from this path at startup. This allows events to be referenced without direct asset references.

Q: Can I use Unity's AudioMixer with this system? A: Yes! Assign AudioMixerGroups to buses in the Inspector. The system will route audio through them.

Usage Questions

Q: How do I make a sound loop? A: Set `Loop = true` on the container (not the event).

Q: How do I stop a looping sound? A: Keep the AudioHandle and call `handle.Stop()`, or create a Stop event.

Q: Can I have multiple music tracks playing? A: Yes! Use a BlendContainer to layer tracks and crossfade between them with RTPCs.

Q: How do I make explosions louder than footsteps? A: Use bus hierarchy or container volume. Example:

```
SFX_Bus (0 dB)
├── Explosions_Bus (+3 dB) ← Louder
└── Footsteps_Bus (-12 dB) ← Quieter
```

Troubleshooting Questions

Q: My sounds aren't playing! A: Checklist: 1. Is AudioManager in the scene? 2. Is the event in Resources/Audio/Events/? 3. Does the container have audio clips assigned? 4. Is the bus volume set correctly (not -80 dB)? 5. Check the Console for errors

Q: I'm getting "No available voices" warnings A: Increase `maxRealVoices` in AudioManager, or set `maxInstances` on events to prevent spam.

Q: 3D sounds aren't spatial A: Make sure: 1. Container has `Is3D = true` 2. AudioListener exists in scene 3. Spatial Blend = 1 on AudioSource

Q: Switch containers aren't switching A: Make sure you call `AudioManager.Instance.SetSwitch(groupName, value)` BEFORE posting the event.

Q: Occlusion isn't working A: Check: 1. `AudioManager.enableOcclusion = true` 2. Occlusion mask includes the geometry layer 3. Sounds are 3D 4. There's an AudioListener in the scene

Performance Questions

Q: How many sounds can play at once? A: Depends on `maxRealVoices` (default 32). Additional sounds will virtualize or be voice-stolen.

Q: Does virtualization affect audio quality? A: No, virtualized voices are paused. When they become real again, they resume from the correct time.

Q: Is the system mobile-friendly? A: Yes! Reduce `maxRealVoices` to 16-24 for mobile, and be mindful of 3D spatial audio cost.

Next Steps

You've Completed the User's Guide!

You now know how to: - ✅ Set up the AudioManager - ✅ Create containers, events, and buses - ✅ Implement footsteps, music, 3D audio - ✅ Use states, switches, and RTPCs - ✅ Build dynamic music systems - ✅ Optimize performance

Where to Go from Here

1. **Read the Comprehensive Manual**: For deep technical details and advanced features
2. **Experiment**: Try creating your own complex audio systems
3. **Study the Examples**: Review the example scenes (if provided)
4. **Build Your Game**: Apply what you've learned!

Need More Help?

- Review the Comprehensive Manual for technical details
- Check the API Reference section for code documentation

- Look at the troubleshooting section for common issues

Happy Sound Design! 🎵

Created with the SFX System v1.0