STL DYNAMICS

Multiplayer

Pooled Audio Event System - FishNet Multiplayer

This guide adds **FishNet multiplayer replication** to the Pooled Audio Event System (PAES) using **PooledAudioFishNetReplicator**, **PooledAudioNetworkCatalog**, and **PooledAudioNetworkHandle**.

The integration keeps pooled playback local on every client. The server sends stable catalog ids and playback state; each client resolves the id to its local **SoundEvent** and plays it through its own **PooledAudioManager**.

You will use two main network paths:

1. **Server-authored playback** - best for weapons, impacts, abilities, interactions, world events, and other gameplay the server already validates.
2. **Optional client one-shot requests** - useful when an owning client needs to request an approved sound through an owned FishNet **NetworkObject**.

---

1. Enable FishNet Support

PAES keeps FishNet optional. The core package still compiles when FishNet is not installed.

1. Install FishNet in the Unity project.
2. Open **Project Settings > Player > Other Settings > Scripting Define Symbols**.
3. Add `PAES_FISHNET`.
4. Apply the change and allow Unity to recompile.
5. Add or select a **PooledAudioFishNetReplicator**. With FishNet enabled, it becomes the full FishNet **NetworkBehaviour** implementation.

> **Important:** Only define `PAES_FISHNET` in projects where FishNet is installed. Without the define, PAES uses a dependency-free placeholder so the rest of the audio package does not throw missing FishNet type errors.

Part 1 - Network Catalog Setup

The network catalog gives every replicated SoundEvent a stable integer identity. Server and client builds must resolve the same ids to the same sounds.

2. Create a PooledAudioNetworkCatalog

1. In the Project window, right-click.
2. Choose **Create > STL Dynamics > Pooled Audio > Networking > FishNet Sound Catalog**.
3. Name the catalog clearly, such as `PAES_NetworkCatalog`.
4. Add every SoundEvent that needs to cross the network.

Field	Purpose
Protocol Version	Increment when intentionally making a network-incompatible catalog change.
Network Id	Positive, unique integer sent across the network. Keep an existing id attached to the same sound.
Key	Optional human-readable lookup key for code and diagnostics.
Sound Event	The local PAES SoundEvent resolved by this catalog entry.
Allow Client Request	Explicitly permits the optional owner-client one-shot request path for this entry.
Minimum Client Request Interval	Per-connection cooldown for client-requestable playback.

> **Catalog rule:** Network ids must be unique, positive, and stable. Do not reuse an id for a different sound while compatible clients may still connect.

---

3. Add the FishNet Replicator

1. Create or select a networked GameObject.
2. Add a FishNet **NetworkObject**.
3. Add **PooledAudioFishNetReplicator**.
4. Assign the same **PooledAudioNetworkCatalog** used by the build.
5. Leave **Local Audio Manager** empty in normal setups. The replicator will use `PooledAudioManager.Instance` on that client.

Client Request Security

- Allow Client One Shot Requests
- Maximum Client Request Position Offset
- Minimum Client Request Interval
- Allow Client Force 2D

Diagnostics

- Log Warnings
- Log Playback
- `ServerReady` reports server/catalog readiness.
- `ClientReady` also requires catalog compatibility and a local manager.

Part 2 - Server-Authored One-Shots

For normal gameplay, let the server validate the action first and then replicate the sound. This is the preferred route for authoritative gameplay.

4. Play a Server One-Shot

The replicator can start a one-shot by **SoundEvent**, catalog **key**, or integer **network id**.

```
using UnityEngine;
using STLDynamics.PooledAudio;
using STLDynamics.PooledAudio.Networking;

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

    public void PlayExplosion(Vector3 worldPosition)
    {
        PooledAudioFishNetReplicator.Active
            .ServerPlayOneShot(explosionSound, worldPosition);
    }
}
```

Equivalent lookup paths:

```
replicator.ServerPlayOneShot(explosionSound, worldPosition);
replicator.ServerPlayOneShot("explosion.large", worldPosition);
replicator.ServerPlayOneShot(12, worldPosition);
```

Each call returns **true** when the server can resolve and replicate the request, or **false** when the server is not ready, the position is invalid, or the catalog entry cannot be resolved.

---

5. Force 2D Playback

Pass `force2D: true` when the replicated sound should use non-spatial output after the request has passed its positional checks.

```
replicator.ServerPlayOneShot(
    alertSound,
    player.transform.position,
    force2D: true);
```

---

6. Play a One-Shot for One Client

Use the targeted route when only one **NetworkConnection** should hear the replicated event.

```
using FishNet.Connection;
using STLDynamics.PooledAudio.Networking;

public void PlayPrivateNotification(NetworkConnection target, Vector3 position)
{
    PooledAudioFishNetReplicator.Active
        .ServerPlayOneShotFor(target, notificationSound, position);
}
```

> **Quick rule:** If gameplay logic is already on the server, call the `Server*` API directly after validation. Do not route ordinary server-owned gameplay through the client request path.

Part 3 - Optional Client One-Shot Requests

Client one-shot requests are intentionally restricted. They are **off by default** and should only be enabled for sounds that are safe for an owning client to request.

7. Configure an Approved Client Request

A request must pass all of these checks:

- ✓ The replicator has **Allow Client One Shot Requests** enabled.
- ✓ The catalog entry has **Allow Client Request** enabled.
- ✓ The request supplies a FishNet **NetworkObject** owned by the sending connection.
- ✓ The requested world position is within **Maximum Client Request Position Offset** of that owned object.
- ✓ The request passes both the global and per-entry rate limits.
- ✓ A requested 2D override is only honored when **Allow Client Force 2D** is enabled.

8. Request a One-Shot from an Owner Client

```
using FishNet.Object;
using UnityEngine;
using STLDynamics.PooledAudio;
using STLDynamics.PooledAudio.Networking;

public class NetworkFootstepRequest : MonoBehaviour
{
    [SerializeField] private SoundEvent footstepSound;
    [SerializeField] private NetworkObject ownedEmitter;

    public void RequestFootstep()
    {
        PooledAudioFishNetReplicator.Active.TryRequestOneShot(
            footstepSound,
            ownedEmitter,
            transform.position);
    }
}
```

If this code is running on a host/server instance, **TryRequestOneShot** directly uses the server one-shot route. On a client, it sends the request to the server for validation.

> **Security:** The PAES request checks protect the audio replication route, but they are not a substitute for gameplay validation. Damage, firing, ability use, inventory actions, and other authoritative state should still be validated by the game before sound replication is triggered.

Part 4 - Replicated Loops

Tracked loops use a server-issued **PooledAudioNetworkHandle**. The server owns the replicated loop state while every client owns its local pooled audio handle.

9. Start and Stop a Network Loop

Start a loop from the server and keep the returned network handle.

```
using UnityEngine;
using STLDynamics.PooledAudio;
using STLDynamics.PooledAudio.Networking;

public class NetworkEngineAudio : MonoBehaviour
{
    [SerializeField] private SoundEvent engineLoop;
    private PooledAudioNetworkHandle loopHandle;

    public void StartEngine()
    {
        loopHandle = PooledAudioFishNetReplicator.Active
            .ServerStartLoop(engineLoop, transform.position);
    }

    public void StopEngine()
    {
        PooledAudioFishNetReplicator.Active.ServerStopLoop(loopHandle);
        loopHandle = PooledAudioNetworkHandle.Invalid;
    }
}
```

Loops can also be started by catalog key or integer network id.

---

10. Update a Network Loop at Runtime

```
replicator.ServerSetLoopPosition(loopHandle, transform.position);
replicator.ServerSetLoopVolume(loopHandle, 0.8f);
replicator.ServerSetLoopPitch(loopHandle, 1.05f);
replicator.ServerSetLoopParameters(loopHandle, volume: 0.8f, pitch: 1.05f);
replicator.ServerSetLoopMuted(loopHandle, true);
replicator.ServerStopLoop(loopHandle);
replicator.ServerStopAllLoops();
```

Volume is clamped to 0-1. **Pitch** is clamped to -3 to 3 before replication.

---

11. Moving Loop Example

```
private void Update()
{
    if (!loopHandle.IsValid)
        return;

    PooledAudioFishNetReplicator.Active
        .ServerSetLoopPosition(loopHandle, transform.position);
}
```

> **Note:** This networking layer replicates loop state. PAES still performs the actual local pooling, playback, distance behavior, voice limiting, and output routing on each client.

Part 5 - Hosts, Late Joiners, and Catalog Compatibility

12. Late-Join Loop Snapshot

When a connection spawns on the server, the replicator sends a catalog signature and a snapshot of every currently tracked server loop.

That means a late-joining client can reconstruct active looping sounds with their current:

- Network sound id
- World position
- 2D override state
- Mute state
- Optional volume override
- Optional pitch override

> **One-shots are not historical state.** A client joining later receives the active loop snapshot, not old one-shot events that already happened.

---

13. Catalog Compatibility

Before accepting the loop snapshot, the client compares its local catalog signature with the server signature.

If the catalog is missing, invalid, or incompatible, the client logs a catalog mismatch error and disables network audio playback for that client instead of resolving ids against the wrong sound set.

```
Server catalog: PAES_NetworkCatalog
Client catalog: PAES_NetworkCatalog

Protocol version and network identity must remain compatible.
```

---

14. Host Behavior

A host can act as both server and client. When the client side starts after the server has already authored loops, the replicator rebuilds local client loop playback from the server's existing loop state.

---

15. Local Audio Requirements

Every listening client still needs a working **PooledAudioManager**. The replicator normally resolves `PooledAudioManager.Instance`, or you can assign a local manager override in the replicator inspector.

The normal PAES audio requirements still apply: valid SoundEvents, assigned clips, an AudioListener, correct 2D/3D behavior, usable distance settings, and sensible voice limits.

Part 6 - Optional Dependency Behavior

16. Projects Without FishNet

When `PAES_FISHNET` is not defined, **PooledAudioFishNetReplicator** compiles as a dependency-free **MonoBehaviour** placeholder.

The placeholder keeps the catalog reference and basic readiness properties so PAES can exist in a project without FishNet installed. The full FishNet RPC and server playback API is only compiled inside the FishNet-enabled branch.

Configuration	Result
FishNet not installed, no define	PAES core and placeholder compile without FishNet references.
FishNet installed, no define	Placeholder remains active; FishNet replication is disabled.
FishNet installed + `PAES_FISHNET`	Full NetworkBehaviour/RPC implementation compiles.
No FishNet + `PAES_FISHNET`	Unity will report missing FishNet namespaces/types. Remove the define or install FishNet.

17. Guard FishNet-Specific Gameplay Code

If a script directly calls the full FishNet replicator API, that script should only compile when FishNet support is enabled.

```
#if PAES_FISHNET
using STLDynamics.PooledAudio.Networking;
#endif

public void PlayNetworkSound()
{
#if PAES_FISHNET
    PooledAudioFishNetReplicator.Active
        .ServerPlayOneShot(soundEvent, transform.position);
#endif
}
```

> **Why this matters:** The placeholder intentionally does not fake FishNet RPC methods. It only keeps the optional component dependency-safe. Game code that depends on FishNet should be optional too.

Part 7 - Troubleshooting

Use the checklist on the next page before chasing networking ghosts. Most failures come from a missing define, invalid catalog, missing local manager, or a request being correctly rejected by the security checks.

18. Common Multiplayer Setup Checklist

Problem	Check
FishNet fields/API are missing	Confirm FishNet is installed and `PAES_FISHNET` is in Scripting Define Symbols.
FishNet namespace/type compile errors	The define is enabled but FishNet is unavailable to the project. Install FishNet or remove the define.
ServerReady is false	The server has not started, or the assigned network catalog is missing/invalid.
ClientReady is false	The client has not started, the catalog is incompatible, or no local PooledAudioManager can be resolved.
Catalog mismatch error	Server and client builds are using incompatible catalog data. Verify protocol version, ids, keys, and SoundEvent assignments.
Client request does nothing	Check global request permission, entry opt-in, NetworkObject ownership, position offset, and request cooldowns.
Loop replicates but no sound plays	Verify the client has PooledAudioManager, the id resolves to a SoundEvent, and the local PAES sound is configured correctly.
3D sound is silent	Check AudioListener, listener distance, SoundEvent distance settings, mute state, and voice limits.

19. Recommended First Multiplayer Test

1. Confirm normal PAES local playback works first.
2. Create `SE_Network_Test` with one short AudioClip.
3. Add it to a new FishNet Sound Catalog with a unique positive network id.
4. Add **NetworkObject** + **PooledAudioFishNetReplicator** to a networked object.
5. Assign the catalog.
6. Start one host and one client.
7. From server-owned gameplay, call:

```
PooledAudioFishNetReplicator.Active.ServerPlayOneShot(
    networkTestSound,
    transform.position);
```

1. Confirm both host and client hear the sound locally.
2. Then test one replicated loop and verify a late-joining client receives the active loop.

> **Debug in layers:** First prove local PAES playback, then catalog resolution, then server one-shots, then loops, and only then enable optional client requests.

20. Minimal Multiplayer API Reference

Server one-shots:

```
replicator.ServerPlayOneShot(soundEvent, worldPosition);
replicator.ServerPlayOneShot(catalogKey, worldPosition);
replicator.ServerPlayOneShot(soundId, worldPosition);
replicator.ServerPlayOneShotFor(targetConnection, soundEvent, worldPosition);
```

Optional client request:

```
replicator.TryRequestOneShot(soundEvent, ownedNetworkObject, worldPosition);
```

Tracked loops:

```
PooledAudioNetworkHandle loop = replicator.ServerStartLoop(soundEvent, worldPosition);
replicator.ServerSetLoopPosition(loop, worldPosition);
replicator.ServerSetLoopVolume(loop, 0.8f);
replicator.ServerSetLoopPitch(loop, 1.05f);
replicator.ServerSetLoopParameters(loop, 0.8f, 1.05f);
replicator.ServerSetLoopMuted(loop, true);
replicator.ServerStopLoop(loop);
replicator.ServerStopAllLoops();
```

Readiness:

```
PooledAudioFishNetReplicator.Active
replicator.Catalog
replicator.ServerReady
replicator.ClientReady
```

---

21. Quick Rule

Let the server decide what happened. Let the replicator tell clients what sound state to reproduce. Let each client's PooledAudioManager handle the actual audio.

Use **ServerPlayOneShot** for authoritative fire-and-forget events.

Use **ServerStartLoop** plus a **PooledAudioNetworkHandle** for persistent replicated audio state.

Use **TryRequestOneShot** only for explicitly approved owner-client requests.

Keep network ids stable and use matching catalogs across compatible server and client builds.

> **Optional integration:** FishNet support is enabled with `PAES_FISHNET`. Without it, the PAES core package remains dependency-free and continues to compile normally.