PixelJam — Development Journal

The Full Story: PlatformIO → Arduino IDE, ESP32Synth → ESP32-audioI2S

This document tells the complete story of the toolchain and library struggles encountered while building the PixelJam audio system. Every dead end, every confusing error, and every decision to pivot is documented here — honestly and in detail — so others building similar systems can avoid the same traps.

Part 1: The PlatformIO Experience

The project started in PlatformIO — a professional embedded development environment built into VS Code. It offers dependency management, multi-board support, and better project structure than Arduino IDE. On paper, it was the right choice. In practice, it became the first major obstacle.

1.1 Unknown Board ID

The first attempt used a **platformio.ini** with the board set to **seeed_xiao_esp32s3**. PlatformIO immediately threw:

> ✗ **UnknownBoard: Unknown board ID 'seeed_xiao_esp32s3'**
>
> PlatformIO could not find the board definition for the Seeed XIAO ESP32-S3 in its registry. The platform version installed (espressif32 default) was too old to include this board.

The fix was to pin the platform to a specific version that included the board definition:

```
platform = espressif32@6.9.0
```

> ✓ Pinning to espressif32@6.9.0 resolved the unknown board error and allowed compilation to proceed.

1.2 The Missing Header: driver/i2s_pdm.h

With the board recognised, the next build attempt failed immediately on the **ESP32Synth** library:

✗ **fatal error: driver/i2s_pdm.h: No such file or directory**

The ESP32Synth library includes driver/i2s_pdm.h which is part of ESP-IDF v5.x. PlatformIO was using the arduino-esp32 core v2.x which bundles ESP-IDF v4.x — the header simply did not exist.

Understanding this error required knowing the dependency chain:

Layer	Version needed	Version installed
ESP32Synth library	Arduino ESP32 Core 3.0+	—
Arduino ESP32 Core 3.0+	ESP-IDF v5.x	—
PlatformIO espressif32@6.9.0	Arduino ESP32 Core 2.x	✗ Wrong
ESP-IDF v4.x (included)	No i2s_pdm.h	✗ Wrong

The README for ESP32Synth clearly states:

⚠ Requires ESP32 Board Core version 3.0.0 or newer.

This was missed initially. The fix required forcing PlatformIO to use arduino-esp32 v3.x by adding a platform_packages override:

```
platform_packages =
  framework-arduinoespressif32 @ https://github.com/espressif/arduino-esp32.git#3.0.7
```

This is where the real trouble began.

1.3 The Toolchain Download Problem

Forcing PlatformIO to use arduino-esp32 v3.x triggered a fresh toolchain download. The toolchain is the compiler, linker, and all platform tools — several gigabytes in total. PlatformIO began downloading and almost immediately started failing:

✗ **Tool Manager: Error: Please read https://bit.ly/package-manager-ioerror**

Connection broken: IncompleteRead(23138529 bytes read, 226316800 more expected) PlatformIO's package mirror dropped the connection mid-download, repeatedly, after downloading only ~23MB of a ~250MB package.

PlatformIO automatically tried different mirrors each time, but every mirror dropped the connection at a similar point. The download counter would reset to 1% and start again:

```
Downloading  [###-------------------------------]    9%  01:23:07
Tool Manager: Warning! Package Mirror: Connection broken...
Tool Manager: Looking for another mirror...
Downloading  [##--------------------------------]    7%  01:09:15
```

Several approaches were tried to work around this:

Attempt 1 — Manual curl download

curl was used with the -C - resume flag which should resume an interrupted download:

```
curl -L -C - "https://github.com/espressif/arduino-
esp32/releases/download/3.0.7/..." -o framework.zip
```

The URL turned out to be wrong — the download returned a 9-byte file (just a redirect response), not the actual framework.

Attempt 2 — Symlink to Arduino IDE installation

Arduino IDE was installed separately and successfully downloaded ESP32 core 3.3.7. The idea was to point PlatformIO directly at this already-downloaded installation using a symlink:

```
platform_packages =
  framework-arduinoespressif32 @
symlink:///Users/apple/Library/Arduino15/packages/esp32/hardware/esp32/3.3.7
```

> ✗ **missing SConscript file 'platformio-build.py'**
>
> The Arduino IDE installation does not include PlatformIO's build script. PlatformIO's framework package has additional files (like platformio-build.py) that Arduino IDE does not ship. The symlink approach cannot work.

Attempt 3 — Direct GitHub clone

PlatformIO was told to pull the framework directly from the arduino-esp32 GitHub repository via git clone, which handles interruptions better than HTTP downloads:

```
platform_packages =
  platformio/framework-arduinoespressif32 @
https://github.com/espressif/arduino-esp32.git#3.0.7
```

This would have worked but cloning the entire arduino-esp32 repository is even larger than the pre-built package — potentially taking 30+ minutes even on a good connection.

⚠ Key insight: PlatformIO and Arduino IDE use different package formats for the same underlying code. The Arduino IDE package cannot be used directly by PlatformIO — it is missing PlatformIO-specific build infrastructure files.

1.4 Decision to Leave PlatformIO

After multiple failed toolchain download attempts, the cost-benefit of staying in PlatformIO was reassessed:

Factor	PlatformIO	Arduino IDE
Toolchain status	Requires 250MB+ download	Already installed (3.3.7)
ESP32Synth compatibility	Needs specific platform_packages config	Works out of the box
Build time (first)	30+ min download	Ready immediately
Project structure	Better (src/, lib/, platformio.ini)	Single .ino file
Library management	Registry + lib_deps	Manual + Library Manager
Blocker	Network/toolchain	None

The decision was made to switch to Arduino IDE. The project structure is slightly less clean, but it was immediately usable with the already-installed ESP32 core 3.3.7 — no downloads needed.

✓ Switched to Arduino IDE with ESP32 core 3.3.7 already installed. Compilation succeeded immediately after moving ESP32Synth to ~/Documents/Arduino/libraries/ and adding WebSockets via Library Manager.

Part 2: The ESP32Synth Library Problems

With the toolchain resolved, compilation succeeded using **ESP32Synth**. The WebSocket worked, the SD card was detected, tracks were found — but there was no audio output. This section documents the full investigation.

2.1 What ESP32Synth Is

ESP32Synth is a polyphonic synthesizer library for ESP32. Its primary purpose is real-time audio synthesis — generating waveforms (sine, sawtooth, pulse, triangle, noise), applying envelopes, wavetables, arpeggios, and effects entirely in software. It also has a streaming feature that can play WAV files from an SD card.

For the PixelJam project, only the streaming feature was needed. This turned out to be where the library's limitations were most apparent.

ESP32Synth feature	Needed for PixelJam	Status
Oscillator synthesis (sine, saw, etc.)	No	Works well
Wavetable playback	No	Works well
SD card WAV streaming	Yes	Broken by default
Envelope / modulation	No	Works well
Seek / scrub position	Yes	Available but unreachable

2.2 The Root Cause: STREAM_BUF_SAMPLES = 0

The streaming system in ESP32Synth uses a ring buffer — a fixed-size memory area where the SD reader writes audio data and the audio task reads from it. The size of this buffer is set by a compile-time constant in ESP32Synth.h:

```
#define STREAM_BUF_SAMPLES 0    // Ring buffer size (Must be power of 2)
```

> ✗ **STREAM_BUF_SAMPLES = 0 disables streaming entirely**
>
> A ring buffer of size 0 has no storage. The streaming engine has nowhere to put audio data read from the SD card. playStream() reports success (returns 0) because the file opens correctly, but no audio is actually buffered or output. isStreamPlaying() returns false immediately.

The library header shows two configuration presets — the default and a low-RAM alternative:

```
// /* // default:
#define MAX_VOICES 80
#define MAX_STREAMS 4
#define STREAM_BUF_SAMPLES 0     // ← streaming disabled
// */

 /* // Low ram usage:
#define MAX_VOICES 1
#define MAX_STREAMS 1
#define STREAM_BUF_SAMPLES 2048  // ← streaming enabled
// */
```

The fix was straightforward — switch to the low-RAM preset by commenting out the default block and uncommenting the low-RAM block. However this required editing the library source file directly, which is fragile — any library update would overwrite the change.

⚠ This is a significant documentation gap in ESP32Synth. The default configuration silently disables its own streaming feature. A first-time user following the examples will see playStream() return 0 (success) and hear nothing — with no error message explaining why.

2.3 API Parameter Confusion

Even before discovering the buffer issue, the playStream() call was being used with incorrect parameters. The function signature is:

```
int8_t playStream(uint16_t voice, fs::FS &fs, const char* path,
                  uint16_t volume = 255,
                  uint32_t rootFreqCentiHz = 26163,
                  bool loop = false);
```

The rootFreqCentiHz parameter is designed for pitch-matched sample playback, not plain WAV streaming. Passing 0 caused a crash:

✗ Backtrace crash when rootFreqCentiHz = 0

Passing 0 for rootFreqCentiHz caused a divide-by-zero or null pointer inside the streaming engine, resulting in a FreeRTOS stack backtrace and immediate reboot. The parameter must be 26163 (middle C in centi-Hz) for standard WAV playback.

The correct call for plain WAV file playback at normal pitch:

```
synth.playStream(0, SD, "/audio/001.wav", 255, 26163, false);
```

2.4 String vs const char* Type Mismatch

When auto-scanning the SD card, track paths were stored in a std::vector<String>. Passing a String directly to playStream() caused a compile error:

✗ cannot convert 'String' to 'const char*'

playStream() expects a const char* for the file path. std::vector<String> stores Arduino String objects. Passing trackPaths[i] directly does not compile — the types are incompatible.

Fix — use .c_str() to convert String to const char*:

```
synth.playStream(0, SD, trackPaths[i].c_str(), 255, 26163, false);
```

2.5 isStreamPlaying() Always Returning False

After fixing the type error and using the correct rootFreqCentiHz, playStream() returned 0 (success). But isStreamPlaying() returned 0 immediately after:

```
Starting stream...
Result: 0
Is playing: 0
```

This confirmed the STREAM_BUF_SAMPLES = 0 diagnosis. The stream opened the file successfully but had nowhere to buffer audio data, so the playing state was never set to true.

Part 3: Switching to ESP32-audioI2S

3.1 Why Not Just Fix ESP32Synth?

The STREAM_BUF_SAMPLES fix was technically simple — just edit two lines in a header file. But it was rejected as the final solution for several reasons:

- Editing library source files is fragile. Any update to ESP32Synth resets the change.
- ESP32Synth is fundamentally a synthesizer, not a media player. Its streaming feature is secondary and its default configuration treats it as such.
- ESP32Synth does not support MP3. All audio files would need to be converted to 16-bit 44100Hz WAV — adding an extra step for every track added to the SD card.
- The seek/scrub feature (seekStreamMs) existed in the API but was unreachable because the streaming engine was not running. Even after the buffer fix, its reliability for real-time scrubbing was uncertain.
- The PixelJam project needed a media player, not a synthesizer. Using ESP32Synth was like using a grand piano to play a CD.

3.2 What ESP32-audioI2S Offers

ESP32-audioI2S (by schreibfaul1 on GitHub) is purpose-built for streaming audio files from SD card or the internet to an I2S DAC. It was designed specifically for the use case PixelJam needed.

Capability	ESP32Synth	ESP32-audioI2S
WAV playback from SD	Yes (but buffer=0 by default)	Yes, works immediately
MP3 playback from SD	No — WAV only	Yes, native
Seek / scrub	seekStreamMs() — unreliable	setAudioPlayTime(sec) — reliable

Capability	ESP32Synth	ESP32-audioI2S
Song duration	Not available	Parsed from file metadata
Track end callback	Not available	audio_eof_mp3() callback
Pause / Resume	pauseStream() / resumeStream()	pauseResume() toggle
Volume control	setMasterVolume(0-1023)	setVolume(0-21)
Memory usage	Large (80 voices default)	Single stream, efficient
PSRAM support	Manual	Automatic when enabled
Primary purpose	Synthesizer	Media player

3.3 The Out of Memory Crash After Switching

After switching to ESP32-audioI2S and wiring up the WebSocket, the first play attempt produced a new error:

> ✗ **OOM: failed to allocate 720896 bytes for AudioBuffer**
>
> ESP32-audioI2S tries to allocate a ~700KB ring buffer for audio streaming. The default heap on the XIAO ESP32-S3 is approximately 320KB — far too small. The allocation failed silently and no audio played.

The XIAO ESP32-S3 has 8MB of PSRAM (Pseudo-Static RAM) available, but it must be explicitly enabled in the IDE settings. When PSRAM is disabled, the library can only use the internal heap (~320KB) and fails. When PSRAM is enabled, the library automatically detects and uses it for the audio buffer.

Fix — in Arduino IDE:

```
Tools → PSRAM → OPI PSRAM
```

> ✓ Enabling OPI PSRAM in Arduino IDE gave ESP32-audioI2S access to 8MB of additional RAM. The 700KB audio buffer was allocated successfully and audio played immediately.

3.4 Why ESP32Synth Did Not Have This Problem

Interestingly, ESP32Synth never crashed with an OOM error despite also needing memory for audio. This is because STREAM_BUF_SAMPLES = 0 meant **no streaming buffer was ever allocated**. The library silently did nothing instead of crashing. In a way, the bug masked a secondary problem — if streaming had been working, ESP32Synth would likely have hit the same OOM issue.

Part 4: Summary — What Went Wrong and Why

Problem	Root Cause	Resolution
PlatformIO: Unknown board	Default platform version too old for XIAO ESP32-S3	Pinned to espressif32@6.9.0
PlatformIO: i2s_pdm.h missing	ESP-IDF v4.x (bundled with old core) lacks this header; ESP32Synth needs v5.x	Required arduino-esp32 core 3.x
PlatformIO: toolchain download failures	Package mirrors dropped large downloads mid-transfer repeatedly	Switched to Arduino IDE (core already installed)
Symlink to Arduino packages failed	PlatformIO needs its own platformio-build.py which Arduino IDE packages don't include	Confirmed Arduino IDE as the build environment
ESP32Synth: no audio despite success return	STREAM_BUF_SAMPLES=0 disables ring buffer; streaming engine has nowhere to write data	Switched to ESP32-audioI2S
ESP32Synth: rootFreqCentiHz=0 crash	Divide-by-zero in streaming engine; parameter must be 26163 for standard playback	Documented; moot after library switch
ESP32Synth: String to const char* error	playStream() expects const char*; std::vector<String> returns String	Used .c_str() converter
ESP32-audioI2S: OOM crash	700KB audio buffer exceeds internal heap; PSRAM not enabled	Enabled OPI PSRAM in Arduino IDE Tools menu

Part 5: Key Lessons

For PlatformIO Users

- Always check that your platform version supports your exact board before starting. Use the PlatformIO boards search: pio boards | grep xiao
- When a library requires a specific core version, forcing PlatformIO to use it requires the platform_packages override — not just changing the platform version number.

- Large toolchain downloads over unstable connections are a real problem with PlatformIO. If the network is unreliable, Arduino IDE may be more practical for initial development.
- Arduino IDE and PlatformIO package formats are incompatible — you cannot symlink one into the other.

For ESP32Synth Users

- ESP32Synth is an excellent synthesizer library. For synthesis (waveforms, envelopes, effects) it works very well out of the box.
- For SD card WAV streaming, you must edit ESP32Synth.h and enable the low-RAM preset (STREAM_BUF_SAMPLES 2048). The default configuration (STREAM_BUF_SAMPLES 0) silently disables streaming.
- playStream() returning 0 does not mean audio is playing — it means the file opened successfully. Always check isStreamPlaying() to confirm the stream is active.
- ESP32Synth does not support MP3. If your audio files are MP3, convert to 16-bit 44100Hz WAV using ffmpeg first.

For ESP32-audioI2S Users

- This library is the right tool for media player use cases — it handles MP3, WAV, streaming, seeking, and track end callbacks reliably.
- Always enable PSRAM (Tools → PSRAM → OPI PSRAM in Arduino IDE) when using this library on boards that have PSRAM. Without it, the 700KB audio buffer allocation will fail.
- The audio.loop() call must be in the main loop() function — do not use delays or blocking code in loop() or audio will stutter.
- Song duration is available via the audio_info() callback — parse the Duration: string to get seconds. This is needed for implementing seek/scrub functionality.

Final note: The total time spent on toolchain and library issues was significantly more than the time spent on the actual audio feature. This is common in embedded development. Choosing the right tool for the specific use case from the start — and verifying library defaults before assuming they work — would have saved most of this time.