How to Stream FreeEEG32 (BrainFlow) to LSL

The Challenge: The Architectural Gap

It is currently not possible to have a **BrainFlow device** (such as the **FreeEEG32**) function as a direct, "out-of-the-box" **LSL server** using only the standard BrainFlow library.

- **BrainFlow** is architected as a **device driver layer**; its job is to manage hardware communication (Serial/BLE) and provide a unified API for data acquisition.
- **LSL (Lab Streaming Layer)** is a **network protocol layer**; it requires an active application to "push" data into the network fabric.

To bridge this gap, you must use an intermediary script. While you can do this manually, using BrainFlow's built-in `add_streamer` function is the most robust method for decoupling hardware acquisition from network distribution.

The Solution: The "Streaming Board" Bridge

This method uses two components:

1. **The Producer:** Your main script that connects to the FreeEEG32 and broadcasts data to a local socket.
2. **The Consumer (LSL Bridge):** A lightweight script that picks up that socket data and converts it into an LSL stream.

Prerequisites

bash

pip install brainflow pylsl

Wees voorzichtig met code.

Step 1: Initialize the Hardware Stream (Producer)

In your main acquisition code, add the `streaming_board` parameter. This offloads the data transmission to BrainFlow's optimized C++ background thread.

python

```python
from brainflow.board_shim import BoardShim, BrainFlowInputParams, BoardIds, BrainFlowPresets

params = BrainFlowInputParams()
params.serial_port = 'COM3'  # Update to your actual port
board_id = BoardIds.FREEEEG32_BOARD.value

board = BoardShim(board_id, params)
board.prepare_session()

# This tells BrainFlow to broadcast data to a local UDP port
board.add_streamer("streaming_board://225.1.1.1:6677", BrainFlowPresets.DEFAULT_PRESET)

board.start_stream()
print("Hardware streaming to socket 225.1.1.1:6677...")
```

Wees voorzichtig met code.

Step 2: The LSL Bridge Script (Consumer)

Run this script simultaneously. It connects to the "virtual" board created by the streamer and pushes that data to LSL.

python

```python
import time
from brainflow.board_shim import BoardShim, BrainFlowInputParams, BoardIds
from pylsl import StreamInfo, StreamOutlet

def run_lsl_bridge():
    # 1. Connect to the data being broadcasted by the Producer script
    params = BrainFlowInputParams()
    params.ip_address = "225.1.1.1"
    params.ip_port = 6677
    # We tell BrainFlow that the incoming data belongs to a FreeEEG32
    params.master_board = BoardIds.FREEEEG32_BOARD.value

    # Use the special STREAMING_BOARD ID to receive the data
    bridge_board = BoardShim(BoardIds.STREAMING_BOARD.value, params)
    bridge_board.prepare_session()
    bridge_board.start_stream()

    # 2. Setup LSL Metadata
    srate = BoardShim.get_sampling_rate(BoardIds.FREEEEG32_BOARD.value)
    eeg_chans = BoardShim.get_eeg_channels(BoardIds.FREEEEG32_BOARD.value)

    info = StreamInfo('FreeEEG32_LSL', 'EEG', len(eeg_chans), srate, 'float32', 'feeg32_id_001')
    outlet = StreamOutlet(info)

    print(f"LSL Bridge Active. Relaying {len(eeg_chans)} channels at {srate}Hz.")

    try:
        while True:
            # Pull data from the local socket buffer
            data = bridge_board.get_board_data()
            if data.any():
                # Extract only EEG channels and push to LSL
                eeg_data = data[eeg_chans]
                outlet.push_chunk(eeg_data.T.tolist())
            time.sleep(0.01) # Prevents CPU spikes
    finally:
        bridge_board.release_session()

if __name__ == "__main__":
    run_lsl_bridge()
```

Why use this method?

1. **Fault Tolerance:** If your LSL script or network crashes, the hardware connection (Producer) remains alive.
2. **Distributed Processing:** You can run the **Producer** on a small Raspberry Pi attached to the FreeEEG32 and the **LSL Bridge** on a powerful workstation for processing.
3. **Low Latency:** Using `add_streamer` leverages BrainFlow's internal ring buffers, which are more efficient than handling raw Python lists.