**Effects System - Pipeline and AudioPlayer Design**

**Overview**

This document outlines the architecture of the **EffectPipeline** and **AudioPlayer** subsystems, which together will form the foundation for audio playback and real-time effect processing.

**Core Design Goals**

* **Separation of Concerns:** Playback logic and effect processing are decoupled.
* **Extensibility:** Easy to add, remove, and reorder effects.
* **Simplicity:** Initial implementation is clean and minimal for testing.
* **Real-Time Safe:** No dynamic allocations or blocking calls during audio processing.

**Key Components**

**1. EffectPipeline**

**Purpose:**

* Manage and apply a series of effects to an audio stream.

**Responsibilities:**

* Hold a list of EffectBase pointers (owned via std::unique\_ptr).
* Apply prepare(), processFrame(), and reset() methods to all effects in sequence.
* Provide methods to add and clear effects.

**Methods:**

* void addEffect(std::unique\_ptr<EffectBase> effect);
* void clear();
* void prepareAll(double sampleRate, int samplesPerBlock);
* void processAll(float\* data, int numSamples);
* void resetAll();

**2. AudioPlayer**

**Purpose:**

* Manage audio file loading, playback state (play/pause/stop), and streaming audio through the EffectPipeline.

**Responsibilities:**

* Load an audio file into memory.
* Provide Play, Pause, and Stop controls.
* Keep track of current playback position.
* Pass audio frames through the EffectPipeline before output.
* Handle sample rate and buffer size properly.

**Methods:**

* bool loadAudioFile(const juce::File& file);
* void play();
* void pause();
* void stop();
* void prepareToPlay(int samplesPerBlockExpected, double sampleRate);
* void getNextAudioBlock(juce::AudioBuffer<float>& buffer);
* void releaseResources();

**Playback States:**

* Stopped
* Playing
* Paused

**3. PassThroughEffect**

**Purpose:**

* Serve as a basic, no-op effect for testing.

**Responsibilities:**

* Implement EffectBase.
* processFrame copies input to output unchanged.

**Methods:**

* Standard prepare(), processFrame(), reset(), getName(), and getMetadata() overrides.

**Data Flow Diagram**

[Audio File Loaded] → [AudioPlayer Buffer] → [EffectPipeline] → [Each Effect in Series] → [Audio Output]

* AudioPlayer loads and manages the raw audio data.
* AudioPlayer streams blocks to EffectPipeline.
* EffectPipeline applies all registered effects to the block.
* Final processed block is sent to the audio output device.

**Folder and File Layout Proposal**

/Effects

EffectBase.h

EffectMetadata.h

EffectPipeline.h

EffectPipeline.cpp

PassThroughEffect.h

PassThroughEffect.cpp

/Audio

AudioPlayer.h

AudioPlayer.cpp

**First Milestone Goal**

* Load an audio file.
* Add a PassThroughEffect to the pipeline.
* Play, pause, and stop audio with effects applied.
* Verify end-to-end data flow without modifying the audio signal.

This document defines the initial step towards a flexible, modular audio processing system.