Sample Code

Creating an audio unit extension using the vDSP library

Add biquadratic filter audio-effect processing to apps like Logic Pro X and GarageBand with the Accelerate framework.

Download

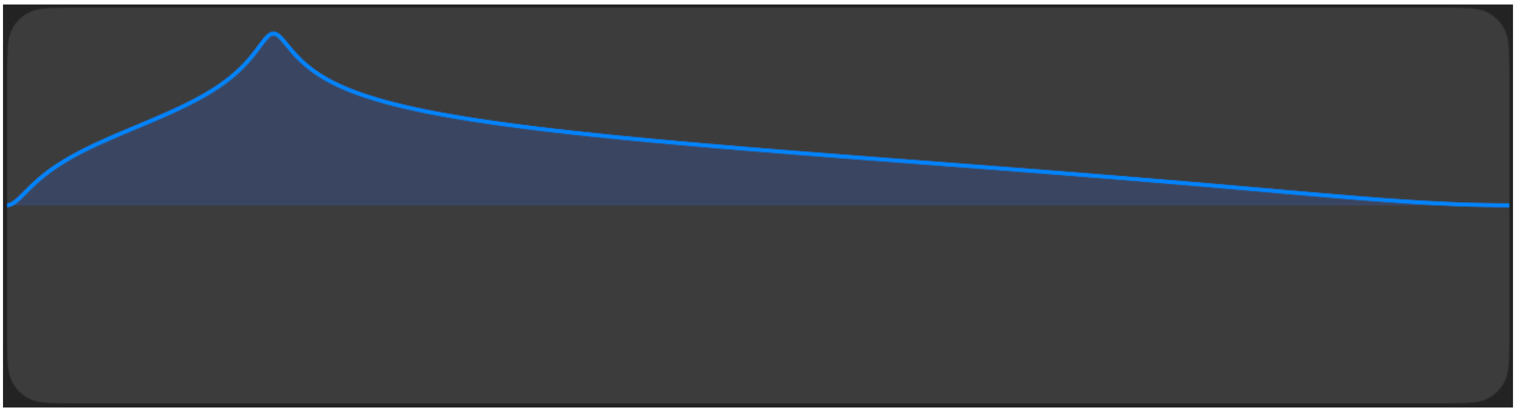
macOS 14.0+ | Xcode 15.1+

Overview

An audio unit extension provides a way to create or modify audio and MIDI data in an iOS or macOS app that uses sound — including music-production apps. It contains the audio unit and, optionally, a user interface to control the audio unit. The audio unit is a custom plug-in where you generate audio or implement an audio-processing algorithm.

You can shape the output of an audio signal, such as by boosting or cutting the bass or treble of a music track, with the single-channel and multichannel biquadratic filters that the vDSP library provides.

The image below shows an example of a magnitude response curve that boosts low frequencies:



This sample code project is a peaking EQ filter implemented with a vDSP biquadratic filter that's delivered as an audio unit extension. You can use the code in this project as the basis for writing audio units that use the vDSP library.

This project is based on the Audio Unit Extension App Xcode template and uses the *Effect* audio unit type. This type of audio unit accepts an audio input and produces an audio output. The template provides an audio pass-through effect with a signal parameter to adjust the gain of the audio that passes through the audio unit.

For more information about creating audio unit extensions, see Creating an audio unit extension.

Add a new parameter address

The peaking EQ filter requires three parameters: the center frequency, the Q value (which controls the shape of the response curve), and the decibel gain.

The code below adds enumeration cases for the parameters to the `vDSP_audio_unitExtensionParameterAddresses.h` header:

```
typedef NS_ENUM(AUParameterAddress, vDSP_audio_unitExtensionParameterAddress) {
    frequency = 0,
    Q = 1,
    dbGain = 2
};
```

To allow the host app to interact with the parameters, the sample code project describes their default value, value range, name, and identifier in `Parameters.swift`. The identifier value you specify is what the audio unit uses to reference the parameter from the host app.

```
ParameterSpec(
    address: .frequency,
    identifier: "frequency",
    name: "Frequency",
    units: .hertz,
    valueRange: 20 ... 20_000,
    defaultValue: 100.0
)

ParameterSpec(
    address: .Q,
    identifier: "Q",
    name: "Q",
```

```
    units: .generic,
    valueRange: 0.1 ... 25,
    defaultValue: 1
)

ParameterSpec(
    address: .dbGain,
    identifier: "dbGain",
    name: "Decibel Gain",
    units: .linearGain,
    valueRange: -50 ... 50,
    defaultValue: 15
)
```

To expose each parameter for digital signal processing (DSP), the code below adds each custom member variable to the `setParameter` and `getParameter` functions:

```
void setParameter(AUParameterAddress address, AUValue value) {
    switch (address) {
        case vDSP_audio_unitExtensionParameterAddress::frequency:
            frequency = value;
            break;
        case vDSP_audio_unitExtensionParameterAddress::Q:
            Q = value;
            break;
        case vDSP_audio_unitExtensionParameterAddress::dbGain:
            dbGain = value;
            break;
    }
}

AUValue getParameter(AUParameterAddress address) {
    // Return the goal. It's not thread safe to return the ramping value.

    switch (address) {
        case vDSP_audio_unitExtensionParameterAddress::frequency:
            return (AUValue)frequency;
        case vDSP_audio_unitExtensionParameterAddress::Q:
            return (AUValue)Q;
        case vDSP_audio_unitExtensionParameterAddress::dbGain:
            return (AUValue)dbGain;
        default: return 0.f;
    }
```

```
}
```

Implement the biquadratic filter

The audio unit extension applies a peaking EQ filter with the vDSP_biquad_Setup filter. For more information about using biquadratic filters, see Applying biquadratic filters to a music loop.

The vDSP_audio_unitExtensionDSPKernel class provides the plug-in's DSP logic, and is written in C++ to ensure real-time safety. The code below initializes the DSP kernel by creating a vector of biquadratic filters with default, pass-though coefficients:

```
void initialize(int inputChannelCount, int outputChannelCount, double inSampleRate)
    mSampleRate = inSampleRate;

    // Default coefficients.
    double coefficients[5] = {1.0, 0.0, 0.0, 1.0, 0.0};

    for (int i = 0; i < inputChannelCount; i++) {

        biquads.push_back((Biquad){
            .setup = vDSP_biquad_CreateSetup(coefficients, 1)
        });

        for (int j = 0; j < 4; j++) {
            biquads[i].delay[j] = 0.0;
        }
    }
}
```

The vDSP_audio_unitExtensionDSPKernel::process() function applies the biquadratic filters to the input channels and writes the result to the output channels:

```
void process(std::span<float const*> inputBuffers,
             std::span<float *> outputBuffers,
             AUEventSampleTime bufferStartTime,
             AUAudioFrameCount frameCount) {

    if (mBypassed) {
        // Pass the samples through.
        for (UInt32 channel = 0; channel < inputBuffers.size(); ++channel) {
            std::copy_n(inputBuffers[channel], frameCount, outputBuffers[channel]);
        }
```

```
        return;
    }

    double coeffs[5];
    // Populate `coeffs` from the parameters.
    biquadCoefficientsFor(mSampleRate,
                          frequency,
                          Q,
                          dbGain,
                          coeffs);

    // For each channel, calculate and set the coefficients, and apply the
    // biquadratic filter.
    for (UInt32 channel = 0; channel < inputBuffers.size(); ++channel) {

        // Set the coefficients on the biquadratic object.
        vDSP_biquad_SetCoefficientsDouble(biquads[channel].setup,
                                          coeffs,
                                          0, 1);

        // Apply the biquadratic filter.
        vDSP_biquad(biquads[channel].setup,
                    biquads[channel].delay,
                    inputBuffers[channel], 1,
                    outputBuffers[channel], 1,
                    frameCount);
    }
}
```

See Also

Biquadratic filter essentials

{} Applying biquadratic filters to a music loop

Change the frequency response of an audio signal using a cascaded biquadratic filter.