Article

Creating an audio unit extension

Build an extension by using an Xcode template.

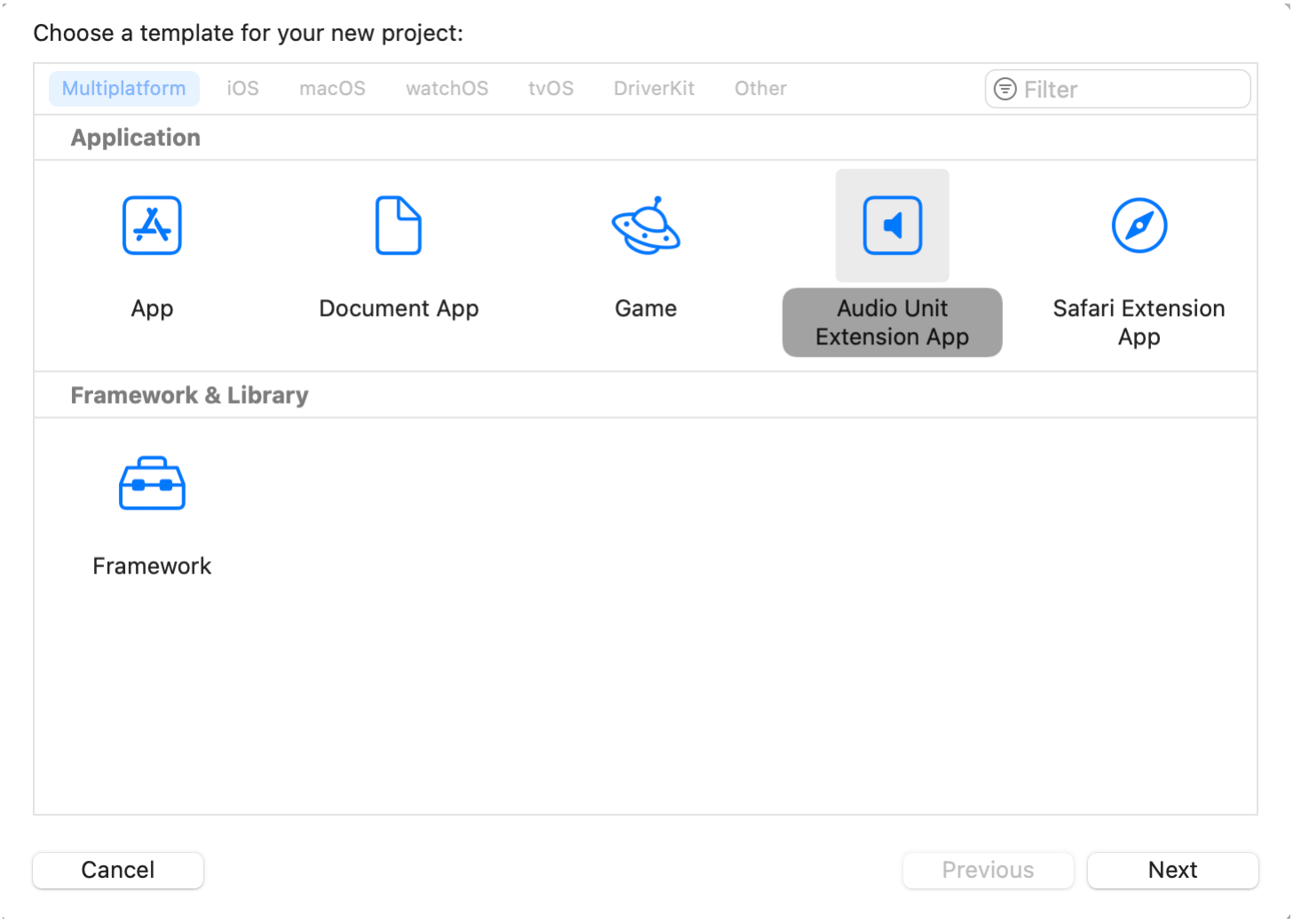
Overview

An audio unit extension provides a way to create or modify audio and MIDI data in a multiplatform 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.

To create an audio unit extension, you create an Xcode project using the Audio Unit Extension App template.

Create a new audio unit extension project

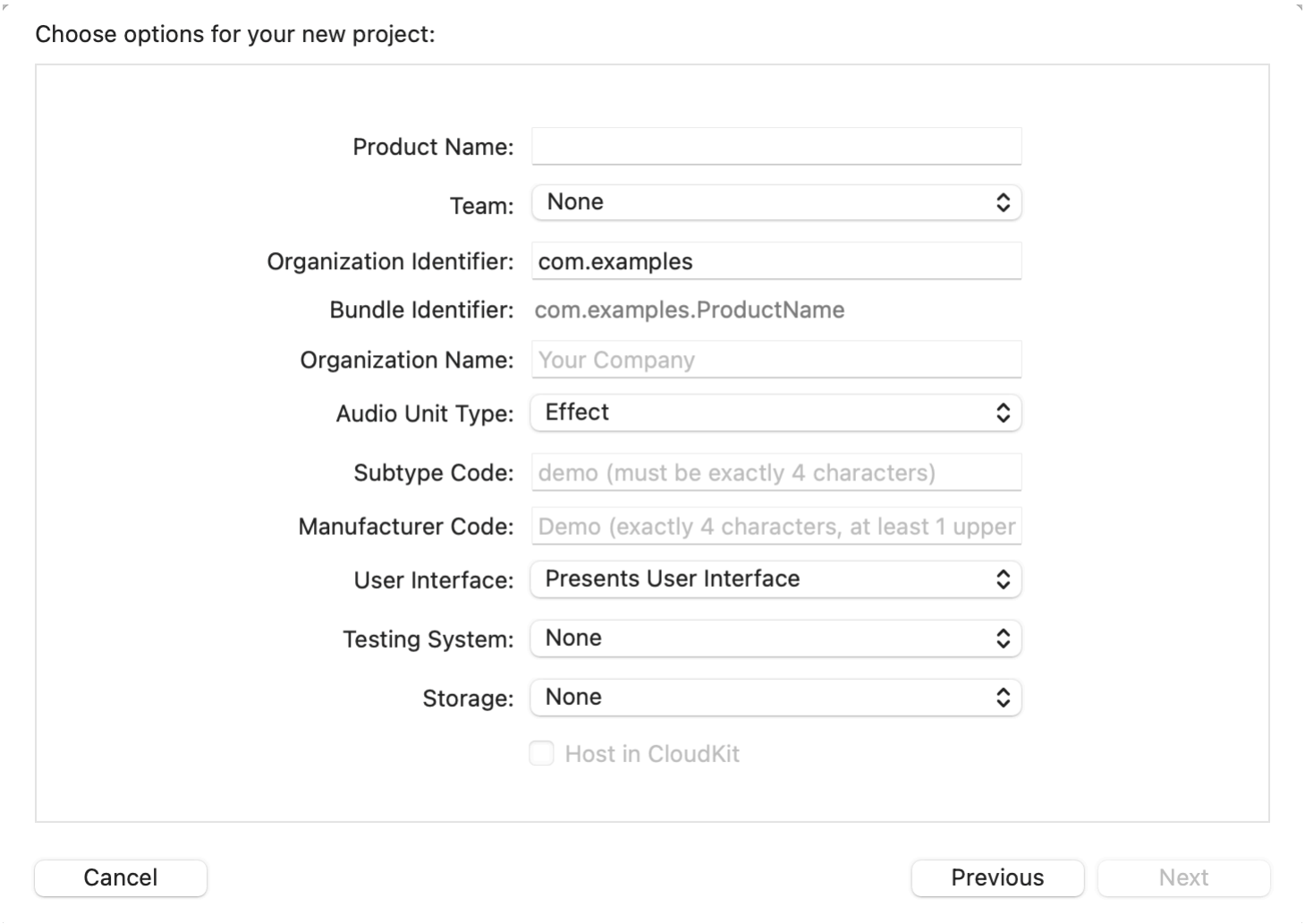
To create a new project in Xcode, choose File > New > Project. In the template chooser, select multiplatform. Scroll to the end of the template chooser and select the Audio Unit Extension App template under the Application section.



After clicking Next, configure the options for the project — including choosing what kind of audio unit to generate.

Configure the project options

Configure the options for your new audio unit extension application. The template creates an extension and a host application for your audio unit.



For an Audio Unit Extension App template, Xcode provides a starting point for the type of audio unit you're creating.

Instrument
: An audio unit that accepts incoming MIDI data and produces only audio output. The template provides a basic mono sine wave synthesizer with a parameter to adjust the gain. Incoming MIDI data sets the pitch and volume of the output sine wave signal.

Generator
: An audio unit that provides a basic sine wave generator and produces only audio output. It has a parameter to adjust the gain, and produces a continuous sine wave signal.

Effect
: An audio unit that accepts audio input and produces 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.

Music Effect
: An audio unit that accepts audio and MIDI input, and produces audio output. The template provides a MIDI-controlled audio gate with one signal parameter to adjust the gain. It allows audio to pass through only when it receives a MIDI note-on message.

MIDI Processor

An audio unit that accepts MIDI input and produces MIDI output. The template provides a MIDI note-on and note-off message generator.

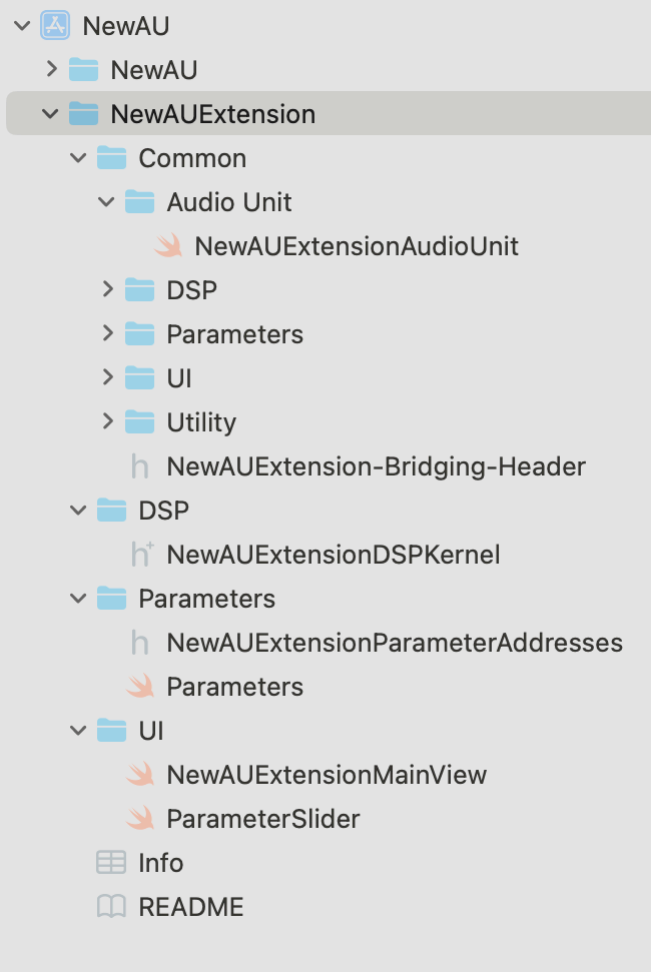
Speech Synthesis
: An audio unit for custom spoken voice generation. The template provides the setup for a speech synthesizer that's ready for customization.

Choose a subtype code that reflects the type of extension, and a manufacturer code that's unique to your company.

If your extension doesn't need a user interface, choose No User Interface; otherwise, Xcode creates a view for you to customize.

Explore the generated extension project

Xcode generates two targets for you — the host app and the extension. The template uses Swift and SwiftUI for the business logic and user interface, C++ for real-time processing, and Objective-C for interacting between Swift and C++.



The Common group contains code — organized by functionality — that you rarely need to change. In the above image, the project name NewAU is prefixed to many files.

In most cases, you only need to edit the extension files within the top-level groups Parameters, DSP, and UI.

`NewAUExtensionParameterAddress.h`
: An enumeration that contains parameter addresses that Swift and C++ use to reference parameters.

`Parameters.swift`
: A source code file where you describe your extension's parameters and their layout.

`NewAUExtensionDSPKernel.hpp`
: A class that handles the real-time aspects of the extension and is where you implement signal processing.

`NewAUExtensionMainView.swift`
: A view you use to customize your user interface, if your audio unit presents one.

Add a new parameter address

The template project contains a parameter that allows you to adjust the audio gain. To add a signal parameter to the template project, navigate to `NewAUExtensionParameterAddress.h` and add it to the enumeration.

```
typedef NS_ENUM(AUParameterAddress, NewAUExtensionParameterAddress) {
    gain = 0,
    attack
}
```

To allow your host app to interact with the parameter, describe its default value, value range, name, and identifier in `Parameters.swift`. The identifier value you specify is what you use to reference the parameter from your host app.

```
ParameterSpec(address: .attack,
              identifier: "attack",
              name: "Attack",
              units: .milliseconds,
              valueRange: 0.0…1000.0,
              defaultValue: 100.0)
```

Navigate to `NewAUExtensionDSPKernel.hpp` to expose the parameter for digital signal processing. Add the custom member variable at the end of the source file, and use the `setParameter` and `getParameter` functions to access it.

```
void setParameter(AUParameterAddress address, AUValue value) {
    switch (address) {
        case NewAUExtensionParameterAddress::attack:
            mAttack = value;
    }
}

AUValue getParameter(AUParameterAddress address) {
    switch (address) {
        case NewAUExtensionParameterAddress::attack:
            return (AUValue)mAttack;
        default:
            return 0.f;
    }
}
```

Use the `process` function to implement your custom signal-processing algorithm. For audio units that present a user interface, you access the parameter by using the `parameterTree` value in the main view.

```
// Get the attack parameter value.
value.let attack = parameterTree.global.attack.value

// Set the attack parameter value.
value.parameterTree.global.attack.value = 0.5

// Bind the parameter to a slider.
var body: some Value {
    ParameterSlider(param: parameterTree.global.attack)
}
```

See Also

Essentials

{} Using voice processing

Add voice-processing capabilities to your app by using audio engine.

`class` AVAudioUnit

A subclass of the audio node class that, processes audio either in real time or nonreal time, depending on the type of the audio unit.