Delivering an Exceptional Audio Experience

A guide to audio best practices and APIs

Session 507

Saleem Mohammed Audio Craftsman
Doug Wyatt Audio Plumber
Introduction

Overview

CoreAudio and Drivers

Application

AVAudioSession
AVAudioPlayer
AVPlayer
AVFoundation
AVAudioRecorder

CoreAudio and Drivers
Introduction

Overview

CoreAudio and Drivers

Application

AVFoundation

AVAudioPlayer

AVAudioRecorder

AVAudioSession

AudioToolbox

AudioUnits

AUGraph

CoreAudio and Drivers
Introduction
Overview

CoreAudio and Drivers

Application
- AVAudioSession
- AVAudioPlayer
- AVPlayer
- AVAudioRecorder

AVFoundation

AudioToolbox

AudioUnits

CoreAudio and Drivers

CoreMIDI

OpenAL
Introduction

Agenda
Introduction

Agenda

Essential Setup
Simple Playback and Recording Scenarios
Advanced Playback and Recording Scenarios
Multichannel Audio
Introduction

Agenda

Essential Setup
Simple Playback and Recording Scenarios
Advanced Playback and Recording Scenarios
Multichannel Audio
Real-time Audio
Instruments, Effects, and Generators
MIDI
Essential Setup
Configuration for iOS, tvOS, and watchOS
Audio Is a Managed Service

Device
Operating System
Drivers
Audio Is a Managed Service

Device
Operating System
Drivers

Inputs
Audio Is a Managed Service

- Inputs
- Operating System
- Device
- Drivers
- Outputs
Audio Is a Managed Service
Audio Is a Managed Service

- Device
- Operating System
- Drivers
- AVAudioSession
- Some App
- Your App
- Another App

Inputs to Device, Operating System, and Drivers, Outputs from Device and AVAudioSession.
AVAudioSession

Express app’s high-level audio needs

- Device
- Operating System
- Drivers

Inputs

- Some App
- AVAudioSession

Your App
- AVAudioSession
- Category
- Mode
- Options

Outputs

- Another App
- AVAudioSession
AVAudioSession
Advanced configuration

- Device
- Operating System
- Drivers

Inputs

- Some App
- AVAudioSession

Your App
- AVAudioSession
- Input Selection
- Channel Count
- Sample Rate

Outputs

- Another App
- AVAudioSession
AVAudioSession
Interact with other audio apps

Inputs

Device
Operating System
Drivers

Outputs

Your App
Activate AVAudioSession
Category
Session Properties

Some App
AVAudioSession

Another App
AVAudioSession
AVAudioSession

Essential steps

Sign up for notifications
Set category, mode, and options
Manage activation
Handle notifications
AVAudioSession

Essential steps

Sign up for notifications
Set category, mode, and options
Manage activation
Handle notifications
Sign up for Notifications

AVAudioSession Interruption Notification
AVAudioSession Route Change Notification
AVAudioSession Media Services Were Reset Notification
AVAudioSession

Essential steps

Sign up for notifications
Set category, mode, and options
Manage activation
Handle notifications
do {
    try AVAudioSession.sharedInstance().setCategory(AVAudioSessionCategoryAmbient)
} catch {
    print(error)
}
do {
  try AVAudioSession.sharedInstance().setCategory(AVAudioSessionCategoryAmbient)
}
catch {
  print(error)
}
Set Category, Mode, and Options

Productivity app

double {
    try AVAudioSession.sharedInstance().setCategory(AVAudioSessionCategoryAmbient)
}
catch {
    print(error)
}

Will obey ringer switch
Will not play audio in the background
Will always mix with others
Set Category, Mode, and Options

Podcast app

do {
    try AVAudioSession.sharedInstance().setCategory(AVAudioSessionCategoryPlayback)
    try AVAudioSession.sharedInstance().setMode(AVAudioSessionModeSpokenAudio)
} 
catch {
    print(error)
}
Set Category, Mode, and Options

Podcast app

do {
  try AVAudioSession.sharedInstance().setCategory(AVAudioSessionCategoryPlayback)
  try AVAudioSession.sharedInstance().setMode(AVAudioSessionModeSpokenAudio)
}
catch {
  print(error)
}
do {
    try AVAudioSession.sharedInstance().setCategory(AVAudioSessionCategoryPlayback)
    try AVAudioSession.sharedInstance().setMode(AVAudioSessionModeSpokenAudio)
}
catch {
    print(error)
}
Set Category, Mode, and Options

Podcast app

do {
    try AVAudioSession.sharedInstance().setCategory(AVAudioSessionCategoryPlayback)
    try AVAudioSession.sharedInstance().setMode(AVAudioSessionModeSpokenAudio)
}
catch {
    print(error)
}

Will interrupt other applications

*Background audio key
do {
    let myAudioSession = AVAudioSession.sharedInstance()
    try myAudioSession.setCategory(AVAudioSessionCategoryPlayback,
        mode: AVAudioSessionModeDefault,
        options: [.interruptSpokenAudioAndMixWithOthers, .duckOthers])
}

catch {
    print(error)
}
do {
    let myAudioSession = AVAudioSession.sharedInstance()
    try myAudioSession.setCategory(AVAudioSessionCategoryPlayback,
        mode: AVAudioSessionModeDefault,
        options: [.interruptSpokenAudioAndMixWithOthers, .duckOthers])
}

catch {
    print(error)
}
do {
    let myAudioSession = AVAudioSession.sharedInstance()
    try myAudioSession.setCategory(AVAudioSessionCategoryPlayback,
        mode: AVAudioSessionModeDefault,
        options:[.interruptSpokenAudioAndMixWithOthers, .duckOthers])
}
catch {
    print(error)
}
do {
    let myAudioSession = AVAudioSession.sharedInstance()
    try myAudioSession.setCategory(AVAudioSessionCategoryPlayback,
    mode: AVAudioSessionModeDefault,
    options: [.interruptSpokenAudioAndMixWithOthers, .duckOthers])
}

catch {
    print(error)
}
do {
    let myAudioSession = AVAudioSession.sharedInstance()
    try myAudioSession.setCategory(AVAudioSessionCategoryPlayback,
        mode: AVAudioSessionModeDefault,
        options: [.interruptSpokenAudioAndMixWithOthers, .duckOthers])
}
catch {
    print(error)
}"Background audio key"
AVAudioSession

Essential steps

Sign up for notifications
Set category, mode, and options
Manage activation
Handle notifications
Manage Session Activation

- Device
- Operating System
- Drivers

Your App
- Activate AVAudioSession
- Category: PlayAndRecord
- Session Properties

Some App
- AVAudioSession

Another App
- AVAudioSession

Inputs
- Outputs
Manage Session Activation

Device
Operating System
Drivers

Your App
Activate AVAudioSession
Category: PlayAndRecord
Session Properties

Some App
AVAudioSession

Another App
AVAudioSession
Manage Session Activation

AVAudioSession

Device
Operating System
Drivers

Some App
AVAudioSession

Your App
Activate AVAudioSession
Category: PlayAndRecord
Session Properties

Another App
AVAudioSession

Inputs

Outputs
Manage Session Activation

Inputs
- Device
- Operating System
- Drivers

Outputs
- AVAudioSession
- Some App
- Another App

Your App
- Activate AVAudioSession
- Category: PlayAndRecord
- Session Properties
Manage Session Activation

AVAudioSession

Another App

Devices

Session Properties

Inputs

Your App

Session Properties

Category: PlayAndRecord

Activate AVAudioSession

Outputs

Some App

AVAudioSession

Another App

AVAudioSession
AVAudioSession

Essential steps

Sign up for notifications
Set category, mode, and options
Manage activation
Handle notifications
func handleInterruption(notification: NSNotification) {
    let interruptionType = notification.userInfo![AVAudioSessionInterruptionTypeKey] as! AVAudioSessionInterruptionType

    if interruptionType == AVAudioSessionInterruptionType.began {
        // session inactive, players have been paused; update any internal state
    }
    else { // end interruption
        // activate session, start playing, update internal state
    }
}
func handleInterruption(notification: NSNotification) {
    let interruptionType = notification.userInfo![AVAudioSessionInterruptionTypeKey] as! AVAudioSessionInterruptionType

    if interruptionType == AVAudioSessionInterruptionType.began {
        // session inactive, players have been paused; update any internal state
    }

    else {
        // end interruption
        // activate session, start playing, update internal state
    }
}

Handle Notifications
Interruptions

No Playback UI
func handleInterruption(notification: NSNotification) {
    let interruptionType = notification.userInfo![AVAudioSessionInterruptionTypeKey] as! AVAudioSessionInterruptionType

    if interruptionType == AVAudioSessionInterruptionType.began {
        // session inactive, players have been paused; update any internal state
    }
    else {
        // end interruption
        // activate session, start playing, update internal state
    }
}
func handleInterruption(notification: NSNotification) {
    let interruptionType = notification.userInfo?[AVAudioSessionInterruptionTypeKey] as! AVAudioSessionInterruptionType

    if interruptionType == AVAudioSessionInterruptionType.began {
        // session inactive, players have been paused; update any internal state
    }

    else { // end interruption
        // activate session, start playing, update internal state
    }
}
Handle Notifications

Interruptions

Playback UI

...  
if interruptionType == AVAudioSessionInterruptionType.began {
    //session inactive, update internal state as well as UI!
}
else { //end interruption
    if let interruptionOption = notification.userInfo![AVAudioSessionInterruptionOptionKey]
        as? AVAudioSessionInterruptionOptions {
        if interruptionOption == .shouldResume {
            // activate session, start playing, update UI & internal state
        }
    }
...
Handle Notifications

Interruptions

Playback UI

```swift
if interruptionType == AVAudioSessionInterruptionType.began {
    // session inactive, update internal state as well as UI!
}
else { // end interruption
    if let interruptionOption = notification.userInfo![AVAudioSessionInterruptionOptionKey]
        as? AVAudioSessionInterruptionOptions {
        if interruptionOption == .shouldResume {
            // activate session, start playing, update UI & internal state
        }
    }
}
...
Handle Notifications

Interruptions

Playback UI

```swift
... if interruptionType == AVAudioSessionInterruptionType.began {
    // session inactive, update internal state as well as UI!
}
else { // end interruption
    if let interruptionOption = notification.userInfo![AVAudioSessionInterruptionOptionKey]
        as? AVAudioSessionInterruptionOptions {
        if interruptionOption == .shouldResume {
            // activate session, start playing, update UI & internal state
        }
    }
...```
Handle Notifications

Interruptions

Not every Begin is followed by an End

- e.g., media players that interrupt each other
Handle Notifications

Route changes

```swift
let routeChangeReason = notification.userInfo?[AVAudioSessionRouteChangeReasonKey] as! AVAudioSessionRouteChangeReason

if routeChangeReason == .oldDeviceUnavailable {
    // media players stop playback; ex. headsets unplugged while listening to music
}

if routeChangeReason == .oldDeviceUnavailable || routeChangeReason == .newDeviceAvailable {
    // advanced use cases; re-evaluate session properties; ex. Sample rate
}
```
Handle Notifications

Route changes

```swift
let routeChangeReason = notification.userInfo?[AVAudioSessionRouteChangeReasonKey] as! AVAudioSessionRouteChangeReason

if routeChangeReason == .oldDeviceUnavailable {
    // media players stop playback; ex. headsets unplugged while listening to music
}

if routeChangeReason == .oldDeviceUnavailable || routeChangeReason == .newDeviceAvailable {
    // advanced use cases; re-evaluate session properties; ex. Sample rate
}
Handle Notifications

Route changes

```swift
let routeChangeReason = notification.userInfo![AVAudioSessionRouteChangeReasonKey] as! AVAudioSessionRouteChangeReason

if routeChangeReason == .oldDeviceUnavailable {
    // media players stop playback; ex. headsets unplugged while listening to music
}

if routeChangeReason == .oldDeviceUnavailable || routeChangeReason == .newDeviceAvailable {
    // advanced use cases; re-evaluate session properties; ex. Sample rate
}
```
Handle Notifications

Media services were reset
Handle Notifications

Media services were reset

Rare, but it does happen
Handle Notifications

Media services were reset

Rare, but it does happen

AVAudioSession sharedInstance pointer still valid
Handle Notifications
Media services were reset

Rare, but it does happen
AVAudioSession sharedInstance pointer still valid
Need to reset category, mode, options, etc
Handle Notifications

Media services were reset

Rare, but it does happen

AVAudioSession sharedInstance pointer still valid

Need to reset category, mode, options, etc

Need to destroy and recreate objects

• AVAudioEngine, queues, remote I/Os, players, etc
Handle Notifications

Media services were reset

Rare, but it does happen

AVAudioSession sharedInstance pointer still valid

Need to reset category, mode, options, etc

Need to destroy and recreate objects

• AVAudioEngine, queues, remote I/Os, players, etc

Test with Settings -> Developer -> Reset Media Services
AVAudioSession

Essential steps review

Sign up for notifications
Set category, mode, and options
Manage activation
Handle notifications
AirPlay and A2DP in PlayAndRecord

Stereo audio over Bluetooth and AirPlay

```
try AVAudioSession.sharedInstance().setCategory(AVAudioSessionCategoryPlayAndRecord,
  mode: AVAudioSessionModeDefault,
  options: [.allowAirPlay, .allowBluetoothA2DP])
```
AirPlay and A2DP in PlayAndRecord

Stereo audio over Bluetooth and AirPlay

```swift
try AVAudioSession.sharedInstance().setCategory(AVAudioSessionCategoryPlayAndRecord, mode: AVAudioSessionModeDefault, options: [.allowAirPlay, .allowBluetoothA2DP])
```
AirPlay and A2DP in PlayAndRecord

Stereo audio over Bluetooth and AirPlay

```swift
try AVAudioSession.sharedInstance().setCategory(AVAudioSessionCategoryPlayAndRecord,
    mode: AVAudioSessionModeDefault,
    options: [.allowAirPlay, .allowBluetoothA2DP])
```

Now you can use microphone while playing to Bluetooth/AirPlay
AirPlay and A2DP in PlayAndRecord

Stereo audio over Bluetooth and AirPlay

```swift
try AVAudioSession.sharedInstance().setCategory(AVAudioSessionCategoryPlayAndRecord, 
    mode: AVAudioSessionModeDefault,
    options: [.allowAirPlay, .allowBluetoothA2DP])
```

Now you can use microphone while playing to Bluetooth/AirPlay
Let the user pick the route from Control Center or a MPVolumeView
let audioSession = AVAudioSession.sharedInstance()
if let currentPort:AVAudioSessionPortDescription = audioSession.currentRoute.inputs.first {
    let disableSoftwareVoiceProcessing = currentPort.hasHardwareVoiceCallProcessing
}
let audioSession = AVAudioSession.sharedInstance()

if let currentPort: AVAudioSessionPortDescription = audioSession.currentRoute.inputs.first {
    let disableSoftwareVoiceProcessing = currentPort.hasHardwareVoiceCallProcessing
}

New Property for VoIP Apps
New Property for VoIP Apps

```swift
let audioSession = AVAudioSession.sharedInstance()
if let currentPort: AVAudioSessionPortDescription = audioSession.currentRoute.inputs.first {
    let disableSoftwareVoiceProcessing = currentPort.hasHardwareVoiceCallProcessing
}
```

Not needed if using Apple's Voice Processing IO

```swift
kAudioUnitSubType_VoiceProcessingIO
```
New Property for VoIP Apps

Not needed if using Apple’s Voice Processing IO

```swift
let audioSession = AVAudioSession.sharedInstance()
if let currentPort: AVAudioSessionPortDescription = audioSession.currentRoute.inputs.first {
    let disableSoftwareVoiceProcessing = currentPort.hasHardwareVoiceCallProcessing
}
```

Enhancing VoIP Apps with CallKit
Further Reference

<table>
<thead>
<tr>
<th>Topic</th>
<th>WWDC Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio Session and Multiroute Audio in iOS</td>
<td>WWDC 2012</td>
</tr>
<tr>
<td>What's New in Core Audio</td>
<td>WWDC 2014</td>
</tr>
<tr>
<td>What's New in Core Audio</td>
<td>WWDC 2015</td>
</tr>
</tbody>
</table>

Audio session programming guide

Simple Playback and Recording
AVFoundation Framework
Simple playback and recording
AVAudioPlayer
Simplest way to play an audio file
AVAudioPlayer

Simplest way to play an audio file

Plays file types supported by AudioFile API

• e.g., wav, caf, m4a, aac, mp3, aif
AVAudioPlayer

Simplest way to play an audio file

Plays file types supported by AudioFile API
• e.g., wav, caf, m4a, aac, mp3, aif

Provides basic playback operations
AVAudioPlayer

Simplest way to play an audio file

Plays file types supported by AudioFile API
- e.g., wav, caf, m4a, aac, mp3, aif

Provides basic playback operations

Supports volume, metering, looping, playback rate, panning
AVAudioPlayer

Simplest way to play an audio file

Plays file types supported by AudioFile API
  • e.g., wav, caf, m4a, aac, mp3, aif

Provides basic playback operations

Supports volume, metering, looping, playback rate, panning

Channel assignment (iOS/tvOS)
AVAudioPlayer

Simplest way to play an audio file

Plays file types supported by AudioFile API
  • e.g., wav, caf, m4a, aac, mp3, aif

Provides basic playback operations

Supports volume, metering, looping, playback rate, panning

Channel assignment (iOS/tvOS)

Multiple AVAudioPlayer objects for multiple sounds; synchronized playback
AVAudioPlayer
Simplest way to play an audio file

Plays file types supported by AudioFile API
• e.g., wav, caf, m4a, aac, mp3, aif

Provides basic playback operations
Supports volume, metering, looping, playback rate, panning
Channel assignment (iOS/tvOS)
Multiple AVAudioPlayer objects for multiple sounds; synchronized playback

public func setVolume(_ volume: Float, fadeDuration duration: TimeInterval)
/* fade to a new volume over a duration */
class ViewController: UIViewController {
    var successSoundPlayer: AVAudioPlayer!
    let successSoundURL = Bundle.main().urlForResource("success", withExtension: "caf")

    override func viewDidLoad() {
        do {
            // setup AVAudioSession if necessary (Ambient); setup other members
            successSoundPlayer = try AVAudioPlayer.init(contentsOf: successSoundURL!)
            successSoundPlayer.prepareToPlay()
        } catch {
            // handle error
        }
    }

    @IBAction func saveDocument() {
        // do some other work; if successful, play success sound!
        successSoundPlayer.play() // check return value
    }
}
class ViewController: UIViewController {

    var successSoundPlayer: AVAudioPlayer!
    let successSoundURL = Bundle.main().urlForResource("success", withExtension: "caf")

    override func viewDidLoad() {
        do { // setup AVAudioSession if necessary (Ambient); setup other members
            successSoundPlayer = try AVAudioPlayer.init(contentsOf: successSoundURL!)
            successSoundPlayer.prepareToPlay()
        }
        catch {
            // handle error
        }
    }

    @IBAction func saveDocument() {
        // do some other work; if successful, play success sound!
        successSoundPlayer.play() //check return value
    }
}
```swift
// AVAudioPlayer Example - Productivity App

class ViewController: UIViewController {
    var successSoundPlayer: AVAudioPlayer!
    let successSoundURL = Bundle.main().urlForResource("success", withExtension: "caf")

    override func viewDidLoad() {
        do { // setup AVAudioSession if necessary (Ambient); setup other members
            successSoundPlayer = try AVAudioPlayer.init(contentsOf: successSoundURL!)
            successSoundPlayer.prepareToPlay()
        }
        catch {
            // handle error
        }
    }

    @IBAction func saveDocument() {
        // do some other work; if successful, play success sound!
        successSoundPlayer.play() //check return value
    }
}
```
class ViewController: UIViewController {
    var successSoundPlayer: AVAudioPlayer!
    let successSoundURL = Bundle.main().urlForResource("success", withExtension: "caf")

    override func viewDidLoad() {
        do {
            // setup AVAudioSession if necessary (Ambient); setup other members
            successSoundPlayer = try AVAudioPlayer.init(contentsOf: successSoundURL!)
            successSoundPlayer.prepareToPlay()
        } catch {
            // handle error
        }
    }

    @IBAction func saveDocument() {
        // do some other work; if successful, play success sound!
        successSoundPlayer.play() //check return value
    }
}
AVAudioRecorder
Simplest way to record an audio file

Record for a specific duration
Records until stopped
Metering
Supports a variety of encoding formats
• AAC, HE-AAC, HE-AACv2, ALAC, LPCM
AVAudioRecorder
Settings dictionary

Format
Sample rate
Number of channels
For LPCM
• Bit depth, endian-ness
For encoded formats
• Quality, bit rate
// AVAudioRecorder Example

do { // setup AVAudioSession (Record/PlayAndRecord); user permission; input selection

    let formatSettings = [AVSampleRateKey : 44100.0,
                          AVNumberOfChannelsKey : 1,
                          AVFormatIDKey : Int(kAudioFormatMPEG4AAC),
                          AVEncoderBitRateKey : 192000,
                          AVEncoderAudioQualityKey : AVAudioQuality.high.rawValue]

    recorder = try AVAudioRecorder.init(url: recordSoundURL, settings: formatSettings)
    recorder.prepareToRecord()
}

catch { /* handle error */ }

@IBAction func toggleRecorder() {
    if recorder.isRecording {
        recorder.stop()
    } else {
        recorder.record()
        // provide feedback using meters for example
    }
}
// AVAudioRecorder Example

do { // setup AVAudioSession (Record/PlayAndRecord); user permission; input selection

    let formatSettings = [AVSampleRateKey : 44100.0,
                          AVNumberOfChannelsKey : 1,
                          AVFormatIDKey : Int(kAudioFormatMPEG4AAC),
                          AVEncoderBitRateKey : 192000,
                          AVEncoderAudioQualityKey : AVAudioQuality.high.rawValue]

    recorder = try AVAudioRecorder.init(url: recordSoundURL, settings: formatSettings)
    recorder.prepareToRecord()
}

catch { /* handle error */ }
...

@IBAction func toggleRecorder() {
    if recorder.isRecording {
        recorder.stop()
    }
    else {
        recorder.record()
        // provide feedback using meters for example
    }
}
// AVAudioRecorder Example

do { // setup AVAudioSession (Record/PlayAndRecord); user permission; input selection
    let formatSettings = [AVSampleRateKey : 44100.0,
                          AVNumberOfChannelsKey : 1,
                          AVFormatIDKey : Int(kAudioFormatMPEG4AAC),
                          AVEncoderBitRateKey : 192000,
                          AVEncoderAudioQualityKey : AVAudioQuality.high.rawValue]

    guard let recorder = try AVAudioRecorder.init(url: recordSoundURL, settings: formatSettings) else { /* handle error */ }
    recorder.prepareToRecord()
}

@IBAction func toggleRecorder() {
    if recorder.isRecording {
        recorder.stop()
    } else {
        recorder.record()
        // provide feedback using meters for example
    }
}
// AVAudioRecorder Example

do { // setup AVAudioSession (Record/PlayAndRecord); user permission; input selection
    let formatSettings = [AVSampleRateKey: 44100.0,
                         AVNumberOfChannelsKey: 1,
                         AVFormatIDKey: Int(kAudioFormatMPEG4AAC),
                         AVEncoderBitRateKey: 192000,
                         AVEncoderAudioQualityKey: AVAudioQuality.high.rawValue]

    try AVAudioRecorder.init(url: recordSoundURL, settings: formatSettings)
    recorder.prepareToRecord()
}
catch { /* handle error */ }

@IBAction func toggleRecorder() {
    if recorder.isRecording {
        recorder.stop()
    }
    else {
        recorder.record()
        // provide feedback using meters for example
    }
}
AVPlayer

Playback of local and stream audio

Works with file and streaming content

Standard player controls available

AVPlayerView / AVPlayerViewController

Works with both audio and video media

Advances in AVFoundation Playback

Mission

Wednesday 9:00AM
Advanced Playback and Recording
Advanced Use Cases
Advanced Use Cases

Playback and recording—files, buffers
Advanced Use Cases

Playback and recording—files, buffers
Audio processing—effects, mixing
Advanced Use Cases

Playback and recording—files, buffers
Audio processing—effects, mixing
3D audio
Advanced Use Cases

Playback and recording—files, buffers
Audio processing—effects, mixing
3D audio

Examples
• Karaoke app
• DJ app
• Game
AVFoundation Framework
Advanced playback and recording

- AVFoundation
- AVAudioEngine
- AudioToolbox
- CoreMIDI
- OpenAL

CoreAudio and Drivers
- CoreAudio
- Drivers
AVAudioEngine
AVAudioEngine

Powerful, feature-rich Objective-C/Swift API set
Powerful, feature-rich Objective-C/Swift API set
Simplifies real-time audio
AVAudioEngine

Powerful, feature-rich Objective-C/Swift API set
Simplifies real-time audio
Manages a graph of nodes
AVAudioEngine

Powerful, feature-rich Objective-C/Swift API set
Simplifies real-time audio
Manages a graph of nodes
Features
- Play and record audio
- Connect audio processing chains, perform mixing
- Capture audio at any point in the processing chain
- 3D Spatialization
AVAudioEngine
AVAudioNode
AVAudioEngine
AVAudioNode

**Source Nodes**

Provide data for rendering

- AVAudioPlayerNode
- AVAudioInputNode
- AVAudioUnitSampler
# AVAudioEngine

## AVAudioNode

<table>
<thead>
<tr>
<th>Source Nodes</th>
<th>Processing Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provide data for rendering</td>
<td>Process audio data</td>
</tr>
<tr>
<td>AVAudioPlayerNode</td>
<td>AVAudioUnitEffect</td>
</tr>
<tr>
<td>AVAudioInputNode</td>
<td>AVAudioMixerNode</td>
</tr>
<tr>
<td>AVAudioUnitSampler</td>
<td>AVAudioEnvironmentNode</td>
</tr>
</tbody>
</table>
### AVAudioEngine

#### AVAudioNode

<table>
<thead>
<tr>
<th>Source Nodes</th>
<th>Processing Nodes</th>
<th>Destination Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provide data for rendering</td>
<td>Process audio data</td>
<td>Terminating node connected to output hardware</td>
</tr>
<tr>
<td>AVAudioPlayerNode</td>
<td>AVAudioUnitEffect</td>
<td>AVAudioOutputNode</td>
</tr>
<tr>
<td>AVAudioInputNode</td>
<td>AVAudioMixerNode</td>
<td></td>
</tr>
<tr>
<td>AVAudioUnitSampler</td>
<td>AVAudioEnvironmentNode</td>
<td></td>
</tr>
</tbody>
</table>
Sample Engine Setup
Karaoke

InputNode → EffectNode (EQ) → PlayerNode (Backing Track) → MixerNode → OutputNode

NodeTapBlock (Analyze) → PlayerNode (Sound Effects)
Sample Engine Setup
Karaoke

- **InputNode**
- **EffectNode (EQ)**
- **PlayerNode (Backing Track)**
- **PlayerNode (Sound Effects)**
- **NodeTapBlock (Analyze)**
- **MixerNode**
- **OutputNode**
Sample Engine Setup

Karaoke

- **InputNode**
- **EffectNode (EQ)**
- **PlayerNode (Backing Track)**
- **PlayerNode (Sound Effects)**
- **NodeTapBlock (Analyze)**
- **MixerNode**
- **OutputNode**
Sample Engine Setup

Karaoke

- **InputNode**
- **EffectNode** (EQ)
- **PlayerNode** (Backing Track)
- **PlayerNode** (Sound Effects)
- **NodeTapBlock** (Analyze)
- **MixerNode**
- **OutputNode**
Sample Engine Setup
Karaoke

Diagram:
- InputNode
- EffectNode (EQ)
- PlayerNode (Backing Track)
- NodeTapBlock (Analyze)
- PlayerNode (Sound Effects)
- MixerNode
- OutputNode

Connections:
- InputNode connects to EffectNode (EQ)
- EffectNode (EQ) connects to PlayerNode (Backing Track)
- PlayerNode (Sound Effects) connects to PlayerNode (Backing Track)
- PlayerNode (Backing Track) connects to MixerNode
- MixerNode connects to OutputNode
Sample Engine Setup

Game

3D Space

InputNode

PlayerNode

EnvironmentNode (3D)

MixerNode

OutputNode

PlayerNode (Backing Track)
Sample Engine Setup

Game

3D Space

InputNode

PlayerNode

EnvironmentNode (3D)

MixerNode

OutputNode

PlayerNode (Backing Track)
Sample Engine Setup

Game

3D Space

InputNode → EnvironmentNode (3D) → MixerNode → OutputNode

PlayerNode (Backing Track)
Sample Engine Setup

Game

3D Space

InputNode → volume → EnvironmentNode (3D)

PlayerNode → position → ListenerPosition

reverbParameters

MixerNode → OutputNode

PlayerNode (Backing Track)
Sample Engine Setup

Game

3D Space

- InputNode
- volume
- EnvironmentNode (3D)
- listenerPosition
- reverbParameters

- PlayerNode
- position
- occlusion

- MixerNode

- OutputNode

PlayerNode (Backing Track)
Core Classes
AVAudioFormat

Core classes
AVAudioFormat

Core classes

Describes data format in an audio file or stream

- Standard format—non-interleaved Float32
- Common formats—Int16/32, Float32/64
- Compressed formats—settings dictionary
AVAudioFormat

Core classes

Describes data format in an audio file or stream

- Standard format—non-interleaved Float32
- Common formats—Int16/32, Float32/64
- Compressed formats—settings dictionary

Contains AVAudioChannelLayout
AVAudioFormat
Core classes

Describes data format in an audio file or stream

• Standard format—non-interleaved Float32
• Common formats—Int16/32, Float32/64
• Compressed formats—settings dictionary

Contains AVAudioChannelLayout

Modern interface to

• AudioStreamBasicDescription
• AudioChannelLayout
AVAudioBuffer

Core classes
AVAudioBuffer
Core classes

AVAudioPCMBuffer
AVAudioCompressedBuffer
AVAudioBuffer

Core classes

AVAudioPCMBuffer
AVAudioCompressedBuffer

Modern interface to
• AudioBufferList
• AudioStreamPacketDescription
AVAudioFile
Core classes
AVAudioFile
Core classes

Read and write files of any supported format
Takes/provides data in the form of AVAudioPCMBuffer
Transparently decodes while reading, encodes while writing
AVAudioFile
Core classes

Read and write files of any supported format
Takes/provides data in the form of AVAudioPCMBuffer
Transparencyly decodes while reading, encodes while writing
Supersedes
• AudioFile
• ExtAudioFile
AVAudioConverter

Core classes
AVAudioConverter
Core classes

Audio format conversion
AVAudioConverter

Core classes

Audio format conversion

• PCM to PCM
  - Integer/float, bit depth, endian swap, interleave/de-interleave, sample rate conversion
Core classes

Audio format conversion

• PCM to PCM
  - Integer/float, bit depth, endian swap, interleave/de-interleave, sample rate conversion
• PCM to/from compressed
  - Encoding, decoding
AVAudioConverter

Core classes

Audio format conversion

• PCM to PCM
  - Integer/float, bit depth, endian swap, interleave/de-interleave, sample rate conversion
• PCM to/from compressed
  - Encoding, decoding

Supersedes

• AudioConverter
AVAudioConverter

Core classes

Audio format conversion

• PCM to PCM
  - Integer/float, bit depth, endian swap, interleave/de-interleave, sample rate conversion
• PCM to/from compressed
  - Encoding, decoding

Supersedes

• AudioConverter

```swift
public let AVSampleRateConverterAlgorithm_MinimumPhase: String
```
Core Classes with AVAudioEngine

- AVAudioNodeTapBlock
- AVAudioFormat
- AVAudioFile
- AVAudioPCMBuffer
- AVAudioConverter
- AVAudioCompressedBuffer
- AVAudioNode
- AVAudioPlayerNode
Core Classes with AVAudioEngine

- AVAudioFormat
  - has
  - AVAudioNode
  - AVAudioFile
  - AVAudioPCMBuffer
  - AVAudioConverter
  - AVAudioCompressedBuffer
  - AVAudioNodeTapBlock
  - AVAudioPlayerNode
Core Classes with AVAudioEngine

- AVAudioFormat
- AVAudioFile
- AVAudioPCMBuffer
- AVAudioConverter
- AVAudioCompressedBuffer

AVAudioNode

AVAudioPlayerNode

AVAUDIO_NODE_TAP_BLOCK
Core Classes with AVAudioEngine

- AVAudioNode
  - AVAudioFile
    - AVAudioFormat
      - AVAudioConverter
      - AVAudioCompressedBuffer
        - AVAudioNodeTapBlock
          - AVAudioPCMBuffer
            - AVAudioPlayerNode
Core Classes with AVAudioEngine

- AVAudioNodeTapBlock
  - provides
  - AVAudioFormat
    - has
    - AVAudioNode
      - is a
      - AVAudioPlayerNode
  - AVAudioFile
    - uses
    - AVAudioPCMBuffer
      - provides
      - AVAudioConverter
        - takes
        - AVAudioCompressedBuffer
Core Classes with AVAudioEngine

- AVAudioFormat
  - has
  - AVAudioNode
    - is a
      - AVAudioPlayerNode
- AVAudioFile
  - uses
- AVAudioPCMBuffer
  - provides
    - AVAudioConverter
      - takes
      - AVAudioCompressedBuffer
- AVAudioNodeTapBlock
  - provides
watchOS

Playback and mixing

AVAudioSession

Core classes

• AVAudioFormat/Buffer/File/Converter

AVAudioEngine

• AVAudioPlayerNode
• AVAudioMixerNode
• AVAudioOutputNode
Demo
AVAudioEngine on watchOS
// AVAudioEngine - watchOS Gaming Example
// Class setup

class GameController: WKInterfaceController {
    // other members ex. SceneKit scene
    // engine and nodes
    var audioEngine = AVAudioEngine()
    let explosionPlayer = AVAudioPlayerNode()
    let launchPlayer = AVAudioPlayerNode()

    // URLs to our audio assets
    let explosionAudioURL = URL.init(fileURLWithPath: "/path/to/explosion.caf")
    let launchAudioURL = URL.init(fileURLWithPath: "/path/to/launch.caf")

    // buffers for playback
    var explosionBuffer: AVAudioPCMBuffer?
    var launchBuffer: AVAudioPCMBuffer?
}
/ AVAudioEngine - watchOS Gaming Example
// Class setup
class GameController: WKInterfaceController {
    // other members ex. SceneKit scene
    // engine and nodes
    var audioEngine = AVAudioEngine()
    let explosionPlayer = AVAudioPlayerNode()
    let launchPlayer = AVAudioPlayerNode()

    // URLs to our audio assets
    let explosionAudioURL = URL.init(fileURLWithPath: "/path/to/explosion.caf")
    let launchAudioURL = URL.init(fileURLWithPath: "/path/to/launch.caf")

    // buffers for playback
    var explosionBuffer:AVAudioPCMBuffer?
    var launchBuffer:AVAudioPCMBuffer?
}
// AVAudioEngine - watchOS Gaming Example
// Class setup

class GameController: WKInterfaceController {
    // other members ex. SceneKit scene
    // engine and nodes
    var audioEngine = AVAudioEngine()
    let explosionPlayer = AVAudioPlayerNode()
    let launchPlayer = AVAudioPlayerNode()

    // URLs to our audio assets
    let explosionAudioURL = URL.init(fileURLWithPath: "/path/to/explosion.caf")
    let launchAudioURL = URL.init(fileURLWithPath: "/path/to/launch.caf")

    // buffers for playback
    var explosionBuffer: AVAudioPCMBuffer?
    var launchBuffer: AVAudioPCMBuffer?
}
// AVAudioEngine - watchOS Gaming Example
// Class setup

class GameController: WKInterfaceController {
    // other members ex. SceneKit scene
    // engine and nodes
    var audioEngine = AVAudioEngine()
    let explosionPlayer = AVAudioPlayerNode()
    let launchPlayer = AVAudioPlayerNode()

    // URLs to our audio assets
    let explosionAudioURL = URL.init(fileURLWithPath: "/path/to/explosion.caf")
    let launchAudioURL = URL.init(fileURLWithPath: "/path/to/launch.caf")

    // buffers for playback
    var explosionBuffer: AVAudioPCMBuffer?
    var launchBuffer: AVAudioPCMBuffer?
audioEngine.attach(explosionPlayer)
audioEngine.attach(launchPlayer)

do {
    // for each of my url assets
    let explosionAudioFile = try AVAudioFile.init(forReading: explosionAudioURL)
    explosionBuffer = AVAudioPCMBuffer.init(pcmFormat: explosionAudioFile.processingFormat,
                                           frameCapacity: AVAudioFrameCount(explosionAudioFile.length))
    try explosionAudioFile.read(into: explosionBuffer!)

    // make connections
    audioEngine.connect(explosionPlayer, to: audioEngine.mainMixerNode,
                         format: explosionAudioFile.processingFormat)
    audioEngine.connect(launchPlayer, to: audioEngine.mainMixerNode,
                         format: launchAudioFile.processingFormat)
}
catch { /* handle error */ }
audioEngine.attach(explosionPlayer)
audioEngine.attach(launchPlayer)

do {
    // for each of my url assets
    let explosionAudioFile = try AVAudioFile.init(forReading: explosionAudioURL)
    explosionBuffer = AVAudioPCMBuffer.init(pcmFormat: explosionAudioFile.processingFormat,
                                            frameCapacity: AVAudioFrameCount(explosionAudioFile.length))
    try explosionAudioFile.read(into: explosionBuffer!)

    // make connections
    audioEngine.connect(explosionPlayer, to: audioEngine.mainMixerNode,
                                format: explosionAudioFile.processingFormat)
    audioEngine.connect(launchPlayer, to: audioEngine.mainMixerNode,
                                format: launchAudioFile.processingFormat)
}
catch { /* handle error */ }
audioEngine.attach(explosionPlayer)
audioEngine.attach(launchPlayer)

do {
    // for each of my url assets
    let explosionAudioFile = try AVAudioFile.init(forReading: explosionAudioURL)
    explosionBuffer = AVAudioPCMBuffer.init(pcmFormat: explosionAudioFile.processingFormat,
                                            frameCapacity: AVAudioFrameCount(explosionAudioFile.length))
    try explosionAudioFile.read(into: explosionBuffer!)

    // make connections
    audioEngine.connect(explosionPlayer, to: audioEngine.mainMixerNode,
                        format: explosionAudioFile.processingFormat)
    audioEngine.connect(launchPlayer, to: audioEngine.mainMixerNode,
                        format: launchAudioFile.processingFormat)
}
catch { /* handle error */ }
audioEngine.attach(explosionPlayer)
audioEngine.attach(launchPlayer)

do {
    // for each of my url assets
    let explosionAudioFile = try AVAudioFile.init(forReading: explosionAudioURL)
    explosionBuffer = AVAudioPCMBuffer.init(pcmFormat: explosionAudioFile.processingFormat,
                                             frameCapacity: AVAudioFrameCount(explosionAudioFile.length))
    try explosionAudioFile.read(into: explosionBuffer!)

    // make connections
    audioEngine.connect(explosionPlayer, to: audioEngine.mainMixerNode,
                        format: explosionAudioFile.processingFormat)
    audioEngine.connect(launchPlayer, to: audioEngine.mainMixerNode,
                        format: launchAudioFile.processingFormat)
}

catch { /* handle error */ }
do {
    // start engine and players
    try audioEngine.start()
    explosionPlayer.play()
    launchPlayer.play()
}

catch { /* handle error */ }

// create an asteroid and launch
launchPlayer.scheduleBuffer(launchBuffer!, completionHandler: nil)
// wait to launch again

// asteroid is destroyed
explosionPlayer.scheduleBuffer(explosionBuffer!, completionHandler: nil)
// clean up scene and destroy the node
do {
    // start engine and players
    try audioEngine.start()
    explosionPlayer.play()
    launchPlayer.play()
}
catch { /* handle error */ }

// create an asteroid and launch
launchPlayer.scheduleBuffer(launchBuffer!, completionHandler: nil)
// wait to launch again

// asteroid is destroyed
explosionPlayer.scheduleBuffer(explosionBuffer!, completionHandler: nil)
// clean up scene and destroy the node
do {
    // start engine and players
    try audioEngine.start()
    explosionPlayer.play()
    launchPlayer.play()
}
catch { /* handle error */ }

// create an asteroid and launch
launchPlayer.scheduleBuffer(launchBuffer!, completionHandler: nil)
// wait to launch again

// asteroid is destroyed
explosionPlayer.scheduleBuffer(explosionBuffer!, completionHandler: nil)
// clean up scene and destroy the node
do {
    // start engine and players
    try audioEngine.start()
    explosionPlayer.play()
    launchPlayer.play()
}
catch { /* handle error */ }

// create an asteroid and launch
launchPlayer.scheduleBuffer(launchBuffer!, completionHandler: nil)
// wait to launch again

// asteroid is destroyed
explosionPlayer.scheduleBuffer(explosionBuffer!, completionHandler: nil)
// clean up scene and destroy the node
Multichannel Audio
Multichannel Audio on tvOS

- Apple tv
- HDMI Receiver
- Surround Sound System: 5.1/7.1
do {
    // set category, mode & options etc...
    let audioSession = AVAudioSession.sharedInstance()
    try audioSession.setActive(true)
    let desiredNumberOfChannels = 6 // 5.1 surround rendering
    if audioSession.maximumOutputNumberOfChannels >= desiredNumberOfChannels {
        try audioSession.setPreferredOutputNumberOfChannels(desiredNumberOfChannels)
    }
    let actualNumberOfChannels = audioSession.outputNumberOfChannels
    /* ... */
}
catch { /* handle error*/ }
AVAudioSession Channel Count

Review

do {
    //set category, mode & options etc...

    let audioSession = AVAudioSession.sharedInstance()
    try audioSession.setActive(true)

    let desiredNumberOfChannels = 6 // 5.1 surround rendering
    if audioSession.maximumOutputNumberOfChannels >= desiredNumberOfChannels {
        try audioSession.setPreferredOutputNumberOfChannels(desiredNumberOfChannels)
    }

    let actualNumberOfChannels = audioSession.outputNumberOfChannels
    /* ... */
}
catch { /*handle error*/ }
do {
    //set category, mode & options etc...
    let audioSession = AVAudioSession.sharedInstance()
    try audioSession.setActive(true)

    let desiredNumberOfChannels = 6 // 5.1 surround rendering
    if audioSession.maximumOutputNumberOfChannels >= desiredNumberOfChannels {
        try audioSession.setPreferredOutputNumberOfChannels(desiredNumberOfChannels)
    }

    let actualNumberOfChannels = audioSession.outputNumberOfChannels
    /* ... */
    } catch { /*handle error*/ }
do {
    //set category, mode & options etc...
    let audioSession = AVAudioSession.sharedInstance()
    try audioSession.setActive(true)
    let desiredNumberOfChannels = 6 // 5.1 surround rendering
    if audioSession.maximumOutputNumberOfChannels >= desiredNumberOfChannels {
        try audioSession.setPreferredOutputNumberOfChannels(desiredNumberOfChannels)
    }
    let actualNumberOfChannels = audioSession.outputNumberOfChannels
    /* ... */
} catch { /*handle error*/ }
Port Description
type: HDMI
AVAudioSession Channel Labels

Port Description
type: HDMI

ChannelDescription

ChannelDescription

ChannelDescription

...
AVAudioSession Channel Labels

Port Description
type: HDMI

ChannelDescription

channelNumber 1

channelLabel kAudioChannelLabel_Left

...
AVAudioEngine Setup

Multichannel content
Mono content -> Spatialize (games)
AVAudioEngine Setup
Multichannel content
AVAudioEngine Setup

Multichannel content

- Multichannel Content
- PlayerNode
- MixerNode
- OutputNode
- Multichannel Hardware

Operations:
- Get Content Format
- Set Connection Format
- Set Connection Format
- Get Hardware Format

Channel Mapping
AVAudioEngine Setup

Spatialize content with multiple mono sources

Mono Content → PlayerNode → OutputNode → Multichannel Hardware

Mono Content → PlayerNode → Environment Node

Mono Content → PlayerNode
AVAudioEngine Setup

Spatialize content with multiple mono sources
AVAudioEngine Setup

Spatialize content with multiple mono sources
AVAudioEngine Setup

Spatialize content with multiple mono sources
AVAudioEngine Setup

Spatialize content with multiple mono sources

Get Content Format → Set Connection Format → Set Compatible Format → Get Hardware Format

Mono Content → PlayerNode → Environment Node → OutputNode → Multichannel Hardware

Mono Content → PlayerNode → Spatial Mapping

Mono Content → PlayerNode → Spatial Mapping
AVAudioEngine Setup

Spatialize content with multiple mono sources

Get Content Format → Set Connection Format → Set Compatible Format → Get Hardware Format

Mono Content → PlayerNode → SoundField → Environment Node → OutputNode → Multichannel Hardware

Mono Content → PlayerNode → SoundField

Mono Content → PlayerNode → SoundField
AVAudioEngine Setup

Spatialize content with multiple mono sources

Get Content Format → Set Connection Format → Set Compatible Format → Get Hardware Format

Mono Content

PlayerNode → Volume

PlayerNode → Position

PlayerNode → Occlusion

Environment Node

Spatial Mapping

OutputNode

Multichannel Hardware
AVAudioEngine
Summary
AVAudioEngine

Summary

Powerful, feature-rich
AVAudioEngine

Summary

Powerful, feature-rich
Simplifies real-time audio
AVAudioEngine

Summary

Powerful, feature-rich
Simplifies real-time audio
Multichannel, 3D audio
AVAudioEngine

Summary

Powerful, feature-rich
Simplifies real-time audio
Multichannel, 3D audio
Supersedes
  • AUGHraph
  • OpenAL
# References

<table>
<thead>
<tr>
<th>Title</th>
<th>Conference Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>What’s New in Core Audio</td>
<td>WWDC 2014</td>
</tr>
<tr>
<td>AVAudioEngine in Practice</td>
<td>WWDC 2014</td>
</tr>
<tr>
<td>What’s New in Core Audio</td>
<td>WWDC 2015</td>
</tr>
</tbody>
</table>
Real-Time Audio

Doug Wyatt Audio Plumber
What Is Real-Time Audio?

Low latency

Needed for

• Music: Software synthesizers, effects
• Telephony (VoIP)
• Other interactive engines
What Is Real-Time Audio?

Deadline: You have N milliseconds to produce N milliseconds of audio
Typically ~3–20 ms
Real-Time Constraints
Real-Time Constraints

Must not block by

• Allocating memory
• Taking a mutex or waiting on a semaphore
• Reading from a file
• Logging
• Calling libdispatch (notably async)
• Calling ObjC and Swift runtimes
• Doing anything else that involves memory allocation or a mutex
Real-Time Audio
Real-Time Audio

Audio Units

• Modular, reusable signal generation/processing blocks
Real-Time Audio

Audio Units

• Modular, reusable signal generation/processing blocks
You can host them
• System built-in units
• Units chosen by user
Real-Time Audio

Audio Units
• Modular, reusable signal generation/processing blocks
You can host them
• System built-in units
• Units chosen by user
Build your own
• As plug-ins (extensions)
• Registered privately to your app
Audio Units: Components
Audio Units: Components

AudioToolbox maintains a system registry
Audio Units: Components

AudioToolbox maintains a system registry
Component key: Type/subtype/manufacturer (4-character codes)
Audio Units: Components

AudioToolbox maintains a system registry
Component key: Type/subtype/manufacturer (4-character codes)
Audio Component types
• Audio Units: Input/output, generators, instruments, effects, converters
• Codecs: Encoders, decoders
Audio Units: Components
Implementations
Audio Units: Components

Implementations

Audio Unit application extensions (macOS, iOS)
Audio Units: Components
Implementations

Audio Unit application extensions (macOS, iOS)
Component bundles (macOS only)
Audio Units: Components

Implementations

Audio Unit application extensions (macOS, iOS)
Component bundles (macOS only)
Inter-app audio nodes (iOS only)
Audio Units: Components

Implementations

Audio Unit application extensions (macOS, iOS)
Component bundles (macOS only)
Inter-app audio nodes (iOS only)
Registered at runtime
Audio Units: Components

Implementations

Audio Unit application extensions (macOS, iOS)
Component bundles (macOS only)
Inter-app audio nodes (iOS only)
Registered at runtime
Apple built-in
Audio Input/Output Units
Audio Input/Output Units

Most commonly used
Audio Input/Output Units

Most commonly used

Preferred higher-level interface to the system’s basic I/O path
Audio Input/Output Units

Most commonly used

Preferred higher-level interface to the system's basic I/O path

AUAudioUnit (AudioToolbox/AUAudioUnit.h)

• Modern interface to version 2 Audio Units: AUHAL, AUREmoteIO
Demo

Using AUAudioUnit
Effects, Instruments, and Generators
Effects, Instruments, and Generators

Hosting

Also AUAudioUnit

Chain render blocks

Parameters

AU-provided views
Effects, Instruments, and Generators

Writing your own
Effects, Instruments, and Generators
Writing your own

Within your own app
• Subclass AUAudioUnit
• +[AUAudioUnit registerSubclass: ... ]
Effects, Instruments, and Generators
Writing your own

Within your own app
• Subclass AUAudioUnit
• +[AUAudioUnit registerSubclass: ... ]

To write a plug-in for distribution
• Also AUAudioUnit subclass
• Packaged as Audio Unit Extension
Demo
Audio Unit Extensions

Torrey Holbrook Walker The Demonstrator
AUAudioUnit
For more information

Audio Unit Extensions

WWDC 2015
MIDI

Music Instrument Digital Interface
<table>
<thead>
<tr>
<th>Task</th>
<th>Preferred APIs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## MIDI

Music Instrument Digital Interface

<table>
<thead>
<tr>
<th>Task</th>
<th>Preferred APIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Play standard MIDI file (or your own sequences)</td>
<td>AVAudioSequencer</td>
</tr>
</tbody>
</table>
# MIDI

**Music Instrument Digital Interface**

<table>
<thead>
<tr>
<th>Task</th>
<th>Preferred APIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Play standard MIDI file</td>
<td>AVAudioSequencer</td>
</tr>
<tr>
<td>(or your own sequences)</td>
<td></td>
</tr>
<tr>
<td>Control software synth</td>
<td>AVAudioUnitMIDIInstrument</td>
</tr>
<tr>
<td></td>
<td>AUAudioUnit</td>
</tr>
</tbody>
</table>
# MIDI

## Music Instrument Digital Interface

<table>
<thead>
<tr>
<th>Task</th>
<th>Preferred APIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Play standard MIDI file (or your own sequences)</td>
<td>AVAudioSequencer</td>
</tr>
<tr>
<td>Control software synth</td>
<td>AVAudioUnitMIDIInstrument</td>
</tr>
<tr>
<td>Communicate with MIDI hardware (e.g., USB, Bluetooth), MIDI network</td>
<td>AUAudioUnit</td>
</tr>
<tr>
<td></td>
<td>CoreMIDI</td>
</tr>
</tbody>
</table>
# MIDI

**Music Instrument Digital Interface**

<table>
<thead>
<tr>
<th>Task</th>
<th>Preferred APIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Play standard MIDI file (or your own sequences)</td>
<td>AVAudioSequencer</td>
</tr>
<tr>
<td>Control software synth</td>
<td>AVAudioUnitMIDIInstrument</td>
</tr>
<tr>
<td>Communicate with MIDI hardware (e.g., USB, Bluetooth), MIDI network</td>
<td>AUAudioUnit</td>
</tr>
<tr>
<td>Inter-process real-time MIDI</td>
<td>CoreMIDI</td>
</tr>
<tr>
<td></td>
<td>CoreMIDI</td>
</tr>
</tbody>
</table>
Recap

Application

CoreAudio and Drivers
Recap

CoreAudio and Drivers

Application

AVFoundation

AVAudioSession

AVAudioPlayer

AVPlayer

AVAudioEngine

AVAudioRecorder

CoreAudio and Drivers
Recap

CoreAudio and Drivers

AudioToolbox

AVFoundation

AVPlayer
AVAudioSession
AVAudioEngine
AVAudioRecorder

AUAudioUnit

Application

CoreAudio and Drivers
Recap

CoreAudio and Drivers

Application

CoreMIDI

AVAudioPlayer

AVFoundation

AVAudioSession

AVAudioEngine

AVAudioRecorder

AudioToolbox

AUAudioUnit

CoreAudio and Drivers
More Information

# Related Sessions

<table>
<thead>
<tr>
<th>Session Title</th>
<th>Venue</th>
<th>Date/Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enhancing VoIP Apps with CallKit</td>
<td>Mission</td>
<td>Thursday 5:00PM</td>
</tr>
<tr>
<td>Advances in AVFoundation Playback</td>
<td>Mission</td>
<td>Wednesday 9:00AM</td>
</tr>
<tr>
<td>Audio Session and Multiroute Audio in iOS</td>
<td>WWDC 2012</td>
<td></td>
</tr>
<tr>
<td>What’s New in Core Audio</td>
<td>WWDC 2014</td>
<td></td>
</tr>
<tr>
<td>AVAudioEngine in Practice</td>
<td>WWDC 2014</td>
<td></td>
</tr>
<tr>
<td>What’s New in Core Audio</td>
<td>WWDC 2015</td>
<td></td>
</tr>
<tr>
<td>Audio Unit Extensions</td>
<td>WWDC 2015</td>
<td></td>
</tr>
<tr>
<td>Labs</td>
<td>Graphics, Games, and Media Lab D</td>
<td>Friday 3:00PM</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Audio Lab</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>