Core Audioの概要

Audio > Core Audio

2008-11-13

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
U.S.A.

アップルジャパン株式会社
〒163-1450 東京都新宿区西新宿
3 丁目20 番2 号
東京オペラシティタワー
http://www.apple.com/jp/

目次

図、表、リスト

第 1 章 Core Audioとは 13

第 2 章 Core Audioの基礎 23

第 3 章　Mac OS Xでの共通の作業　59

付録 C　Mac OS Xのシステム付属オーディオユニット　83

付録 D　Mac OS Xでサポートされているオーディオファイルフォーマットとオーディオデータフォーマット　87

はじめに

Core Audioは、iPhone OSおよびMac OS X向けに作成するアプリケーションにオーディオ機能を実装するためのソフトウェアインターフェイスを提供します。これらの各プラットフォームにおいて、Core Audioは見えないところでオーディオのあらゆる側面を処理します。iPhone OSでは、Core Audioには、録音、再生、サウンドエフェクト、定位操作、フォーマット変換、ファイルストリーム解析のほか、以下の機能が含まれています。

- アプリケーションで使用できる内蔵のイコライザとミキサー
- オーディオ入出力ハードウェアへの自動アクセス
- 電話の受信が可能なデバイスのもとでアプリケーションのオーディオ面を管理できるAPI
- オーディオ品質に影響することなくバッテリーの持続時間を延ばすための最適化

デスクトップコンピュータやノートブックコンピュータ向けのMac OS Xでは、Core Audioには、録音、編集、再生、圧縮と伸長、MIDI、信号処理、ファイルストリーム解析、およびオーディオの合成の機能が含まれています。Core Audioを使用して、単独のアプリケーションを作成したり、既存の製品と連携して機能するモジュール型のエフェクトプラグインやコーデックプラグインを作成したりできます。

Core Audioは、CとObjective-Cのプログラミングインターフェイスと密接なシステム統合機能を兼ね備え、信号チェーンを通じた低レイテンシを維持する柔軟なプログラミング環境をもたらします。iPhone OSでは、Objective-C言語をベースとするCocoa Touchアプリケーションの中でCore Audioを使用します。Mac OS Xでは、C、Objective-C、またはC++アプリケーションの中でCore Audioのインターフェイスを使用できます。

Core AudioはMac OS Xのすべてのバージョンで使用できますが、古いバージョンにはこの文書で説明する機能の一部が含まれていない場合があります。Core Audioはバージョン2.0以降のiPhone OSで使用できます。この文書では、iPhone OS 2.2およびMac OS X v10.5の時点で使用できるCore Audioの機能について説明します。

注： Core Audioでは、オーディオのデジタル権利管理(DRM)を直接サポートしていません。オーディオファイルでDRMのサポートを必要とする場合は、独自に実装する必要があります。

『*Core Audioの概要*』は、iPhone、iPod touch、またはMac OS Xを実行するコンピュータ向けのオーディオソフトウェアの開発に興味のあるすべてのデベロッパを対象としています。この文書は、オーディオ全般、デジタルオーディオ、MIDI用語の基本的な知識を前提として記述されています。また、オブジェクト指向プログラミングの概念やAppleの開発環境であるXcodeにある程度慣れていることを前提として記述されています。iPhone OSをベースとするデバイスを開発対象とする場合は、Cocoa Touchの開発についての知識が必要です。詳細については、『*iPhone Application Programming Guide*』を参照してください。

この書類の構成

この文書は次の章で構成されています。

- 「Core Audioとは」（13 ページ）では、Core Audioの機能とその使用目的について説明します。
- 「Core Audioの基礎」（23 ページ）では、Core Audioのアーキテクチャについて説明し、そのプログラミングパターンやイディオムを紹介します。また、アプリケーションでのCore Audioの基本的な使いかたを示します。
- 「Mac OS Xでの共通の作業」（59 ページ）では、Core Audioを使用してMac OS Xでいくつかのオーディオ作業を実行する方法について概説します。

また、この文書には次の4つの付録があります。

- 「Core Audioフレームワーク」（71 ページ）では、Core Audioを定義するフレームワークおよびヘッダを列挙します。
- 「Core Audioサービス」（77 ページ）では、Core Audioを別の視点から眺め、iPhone OS、Mac OS X、および両方のプラットフォームで使用できるサービスを列挙します。
- 「Mac OS Xのシステム付属オーディオユニット」（83 ページ）では、Mac OS X v10.5に付属のオーディオユニットを列挙します。
- 「Mac OS Xでサポートされているオーディオファイルフォーマットとオーディオデータフォーマット」（87 ページ）では、Mac OS X v10.5のCore Audioでサポートされているオーディオファイルおよびデータの形式を列挙します。

関連項目

オーディオとCore Audioの詳細については、以下のリソースを参照してください。

- 『*AVAudioPlayer Class Reference*』。iPhone OSアプリケーションでオーディオを再生するための簡易Objective-Cインターフェイスについて説明します。
- 『*Audio Session Programming Guide*』。iPhone OSアプリケーションのオーディオ動作の重要な側面を指定する方法について説明します。
- 『*Audio Queue Services Programming Guide*』。アプリケーションで録音と再生を実装する方法について説明します。
- 『*Core Audio Data Types Reference*』。Core Audio全体を通じて使用されるデータ型について説明します。
- 『*Audio File Stream Services Reference*』。ストリーミングされたオーディオの操作に使用するインターフェイスについて説明します。
- 『*Audio Unit Programming Guide*』。Mac OS X用のオーディオユニットの作成について詳しく説明します。
- 『*Core Audio Glossary*』。Core Audioのドキュメントセット全体を通じて使用される用語を定義します。
- 『*Apple Core Audio Format Specification 1.0*』。Appleの汎用オーディオコンテナ形式であるCAF (Core Audio File)フォーマットについて説明します。

- Core Audioメーリングリスト：http://lists.apple.com/mailman/listinfo/coreaudio-api
- Mac OS Xオーディオデベロッパサイト：http://developer.apple.com/audio/
- Core Audio SDK（ソフトウェア開発キット）の入手先：http://developer.apple.com/sdk/

Core Audioとは

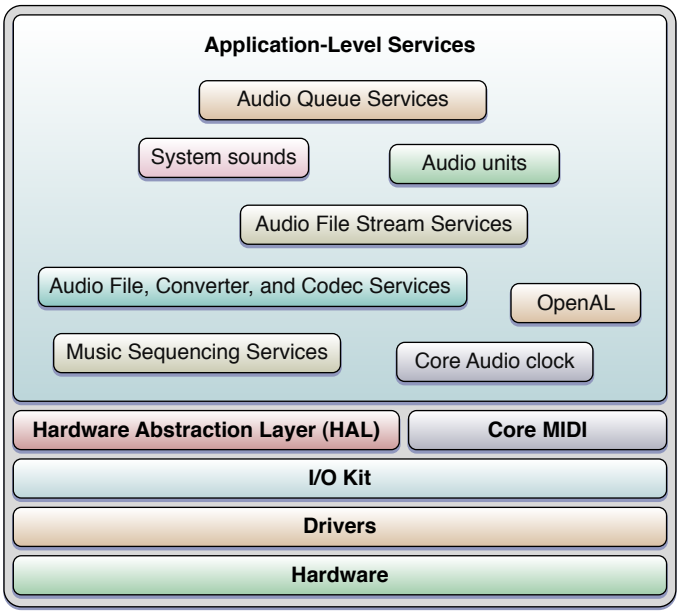
Core Audioは、iPhone OSおよびMac OS Xのデジタルオーディオインフラストラクチャです。アプリケーションに必要なオーディオ処理のために開発された一連のソフトウェアフレームワークが含まれています。この章では、iPhone OSおよびMac OS XでCore Audioを使用してできることについて説明します。

iPhone OSおよびMac OS XのCore Audio

Core AudioはiPhone OSおよびMac OS Xに密接に統合されており、高パフォーマンスと低レイテンシを実現します。

Mac OS Xでは、図 1-1に示すように、Core Audioサービスの大半がハードウェア抽象化層(HAL)の上に位置しています。オーディオ信号はHALを通じてハードウェアを行き来します。リアルタイムオーディオを必要とする場合は、Core AudioフレームワークのAudio Hardware Servicesを使用してHALにアクセスできます。Core MIDI (Musical Instrument Digital Interface)フレームワークは、MIDIデータおよびMIDIデバイスを操作するための同様のインターフェイスを提供します。

図 1-1 Mac OS XのCore Audioアーキテクチャ

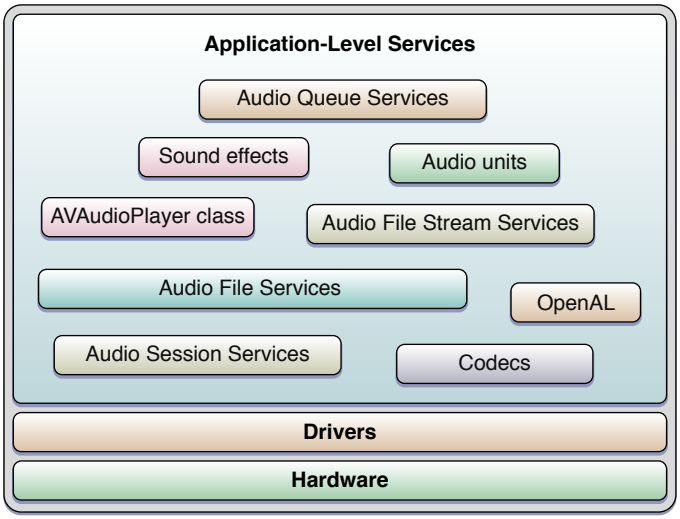


Core Audioのアプリケーションレベルのサービスは、Audio ToolboxおよびAudio Unitフレームワークの中にあります。

- Audio Queue Servicesは、オーディオの録音、再生、一時停止、ループ再生、および同期に使用します。
- Audio File Services、Converter Services、およびCodec Servicesは、ディスクからの読み取り、ディスクへの書き込み、およびオーディオデータフォーマットの変換に使用します。Mac OS Xでは、カスタムのコーデックを作成することもできます。
- Audio Unit ServicesとAudio Processing Graph Services（図では「オーディオユニット(Audio units)」と表記しています）は、アプリケーションでのオーディオユニット（オーディオプラグイン）のホストに使用します。Mac OS Xでは、カスタムのオーディオユニットを作成して、自分のアプリケーションで使用したり、ほかのアプリケーションで使用できるように提供したりできます。
- Music Sequencing Servicesは、MIDIをベースとする制御データや音楽データの再生に使用します。
- Core Audio Clock Servicesは、オーディオとMIDIの同期、および時間形式の管理に使用します。
- System Sound Services（図では「システムサウンド(System sounds)」と表記しています）は、システムサウンドやユーザインターフェイスのサウンドエフェクトの再生に使用します。

iPhone OSのCore Audioは、バッテリー駆動のモバイルプラットフォームで使用できるコンピューティングリソースに合わせて最適化されています。オペレーティングシステムによる非常に密接な管理を必要とするサービス、特に、HALやI/O KitのAPIはありません。しかし、iPhone OSでは、Mac OS Xにはないサービスが追加されています。たとえば、Audio Session Servicesを使用して、携帯電話やiPodとして機能するデバイスのもとで、アプリケーションのオーディオ動作を管理できます。図1-2は、iPhone OSのオーディオアーキテクチャの概要を示しています。

図 1-2 iPhone OSのCore Audioアーキテクチャ



デジタルオーディオとリニアPCMについて

ほとんどのCore Audioサービスは、最も一般的な非圧縮デジタルオーディオデータフォーマットであるリニアパルス符号変調（**リニアPCM**）フォーマットでオーディオを使用し、操作します。デジタルオーディオの録音では、アナログ（実世界の）オーディオ信号の振幅が決められた間隔（**サンプリングレート**）で計測され、各サンプルが数値に変換されることによって、PCMデータが生成されます。標準のコンパクトディスク(CD)オーディオでは、44.1 kHzのサンプリングレートが使用され、各サンプルが16ビット整数の解像度（**ビット深度**）で表されます。

- 1チャネル分を表す単一の数値を**サンプル**と呼び、
- 時間的に同じ位置にあるサンプルの集まりを**フレーム**と呼びます。たとえば、ステレオサウンドファイルにはフレームあたり2つのサンプルがあり、1つは左チャネル用、もう1つは右チャネル用です。
- 1つまたは複数の連続するフレームの集まりを**パケット**と呼びます。リニアPCMオーディオでは、パケットは常に単一のフレームです。圧縮されたフォーマットでは、通常は複数のフレームになります。パケットによって、対象のオーディオデータフォーマットにおいて意味のある最小のフレームセットが定義されます。

リニアPCMオーディオでは、自身が表現する元の信号の振幅に応じてサンプル値がリニア（直線的）に変化します。たとえば、標準のCDオーディオにおける16ビット整数のサンプルでは、無音から最大レベル音までの間で65,536とおりの値を使用できます。あるデジタル値から次のデジタル値までの振幅の差は常に同じです。

`CoreAudioTypes.h`ヘッダファイルで宣言されているCore Audioデータ構造体では、リニアPCMを任意のサンプルレートとビット深度で表現できます。「オーディオデータフォーマット」（28 ページ）では、このトピックについてさらに詳しく説明します。

Mac OS Xでは、Core Audioのオーディオデータとして、ネイティブなエンディアン、32ビット浮動小数点のリニアPCMフォーマットが想定されています。Audio Converter Servicesを使用すると、オーディオデータをさまざまなリニアPCMフォーマット間で変換できます。また、これらのコンバータを使用して、リニアPCMと圧縮オーディオフォーマット（MP3やApple Losslessなど）間の変換も行えます。Mac OS XのCore Audioでは、最も一般的なデジタルオーディオフォーマットを変換するための各種コーデックが提供されています（ただし、MP3に変換するためのエンコーダは提供されていません）。

iPhone OSでは、整数および固定小数点のオーディオデータが使用されます。そのため、オーディオ処理の際に計算が高速になり、バッテリーの消費が少なくなります。iPhone OSではConverterというオーディオユニットが提供されており、Audio Converter Servicesからのインターフェイスが組み込まれています。iPhone OSおよびMac OS X用のいわゆる*正準形*のオーディオデータフォーマットの詳細については、「正準形のオーディオデータフォーマット」（30 ページ）を参照してください。

iPhone OSおよびMac OS XのCore Audioでは、オーディオデータを格納および再生するための最も一般的なファイルフォーマットがサポートされています。詳細については、「iPhoneのオーディオデータフォーマット」（36 ページ）および「Mac OS Xでサポートされているオーディオファイルフォーマットとオーディオデータフォーマット」（87 ページ）を参照してください。

オーディオユニット

オーディオユニットは、オーディオデータを処理するソフトウェアプラグインです。Mac OS Xでは、単一のオーディオユニットを無数のチャネルおよびアプリケーションが同時に使用できます。

iPhone OSでは、モバイルプラットフォーム用に効率とパフォーマンスが最適化された一連のオーディオユニットが提供されています。iPhone OSアプリケーションで使用するオーディオユニットを開発できます。カスタムのオーディオユニットコードはアプリケーションに静的にリンクする必要があります。そのため、自分で作成したオーディオユニットをiPhone OSのほかのアプリケーションで使用することはできません。

iPhone OSに含まれているオーディオユニットにはユーザインターフェイスがありません。iPhoneのオーディオユニットの主要な用途は、アプリケーションにおいて低レイテンシのオーディオを実現することです。詳細については、「Core Audioプラグイン：オーディオユニットとコーデック」（49 ページ）を参照してください。

自分で開発するMac OS Xアプリケーションでは、システムが提供するオーディオユニットやサードパーティが提供するオーディオユニットを使用できます。また、オーディオユニットを、自分が権利を有する製品として開発することもできます。開発したオーディオユニットは、ユーザがGarageBandやLogic Studioなどのアプリケーションで利用できるほか、オーディオユニットをホストするそれ以外の多くのアプリケーションでも利用できます。

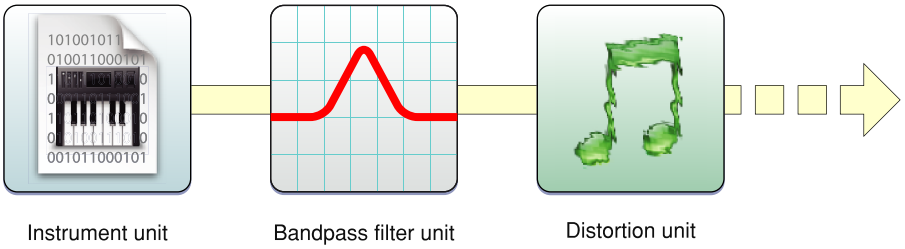
Mac OS Xのオーディオユニットには、信号の分割やハードウェアとのインターフェイスといった共通の作業を単純化するために、見えないところで動作しているものがあります。または、画面上に表示され、独自のインターフェイスを持ち、信号の処理や操作を可能にするものもあります。たとえば、ギタリストのディストーションボックスなど、実世界のエフェクトを模倣できるエフェクトユニットがあります。さらに、プログラミングを通じて、あるいはMIDI入力に応答して、信号を生成するオーディオユニットもあります。

以下は、Mac OS Xで提供されているオーディオユニットの例です。

- 信号プロセッサ（ハイパスフィルタ、リバーブ、コンプレッサ、ディストーションユニットなど）。これらはそれぞれ、一般に**エフェクトユニット**と呼ばれ、ハードウェアエフェクトボックスや外部の信号プロセッサと同様の方法でデジタル信号処理(DSP)を実行します。
- 楽器やソフトウェアシンセサイザ。これらは**インスツルメントユニット**（またはミュージックデバイス）と呼ばれ、通常はMIDI入力に応答して音を生成します。
- 信号ソース。インスツルメントユニットとは異なり、**ジェネレータユニット**はMIDI入力ではなくコードによって起動されます。たとえば、正弦波の計算や生成を行ったり、ファイルやネットワークストリームからのデータを供給したりします。
- ハードウェア入力または出力へのインターフェイス。**I/Oユニット**の詳細については、「ハードウェア抽象化層」（17 ページ）および「Mac OS Xでのハードウェアとのインターフェイス」（60 ページ）を参照してください。
- フォーマットコンバータ。**コンバータユニット**は、2つのリニアPCMフォーマット間でのデータの変換、オーディオストリームの結合または分割、時間およびピッチの変更を実行できます。詳細については、「Core Audioプラグイン：オーディオユニットとコーデック」（49 ページ）を参照してください。
- ミキサーまたはパンナー。**ミキサーユニット**は、複数のオーディオトラックを結合できます。**パンナーユニット**は、ステレオまたは3Dのパニングエフェクトを適用できます。
- オフラインで動作するエフェクトユニット。**オフラインエフェクトユニット**は、プロセッサへの負荷が大きすぎる処理や、単純にリアルタイムでは不可能な処理を実行します。たとえば、ファイルに対して時間反転処理を実行するエフェクトは、オフラインで適用する必要があります。

Mac OS Xでは、デベロッパやエンドユーザが必要とするオーディオユニットを自由に組み合わせて使用できます。図1-3は、Mac OS Xでのオーディオユニットの単純な連なりを示しています。インスツルメントユニットは、外部のMIDIキーボードから受信した制御データに基づいて、オーディオ信号を生成します。生成されたオーディオは次にエフェクトユニットを通過し、バンドパスフィルタリングとディストーションの処理が適用されます。一連のオーディオユニットのことを**オーディオ処理グラフ**と呼びます。

図 1-3 Mac OS Xでの単純なオーディオ処理グラフ



Mac OS Xの複数のアプリケーションで使用可能なオーディオDSPコードを開発する場合は、コードをオーディオユニットとしてパッケージ化する必要があります。

Mac OS X向けにオーディオアプリケーションを開発する場合は、オーディオユニットをサポートすることによって、デベロッパやユーザは既存のオーディオユニットのライブラリ（サードパーティが提供するものとAppleが提供するものの両方）を利用してアプリケーションの機能を拡張できます。

Mac OS Xでオーディオユニットを試す場合は、`/Developer/Applications/Audio`のXcode Toolsに含まれているAU Labアプリケーションを参照してください。AU Labでは、オーディオユニットをうまく組み合わせて、音源から出力デバイスまでの信号チェーンを構築できます。

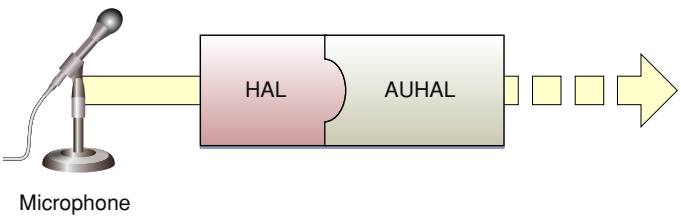
Mac OS X v10.5およびiPhone OS 2.0に付属のオーディオユニットの一覧については、「Mac OS Xのシステム付属オーディオユニット」（83 ページ）を参照してください。

ハードウェア抽象化層

Core Audioは、ハードウェア抽象化層(HAL)を使って、アプリケーションがハードウェアとやり取りするための、一貫性があり予測可能なインターフェイスを提供しています。HALはまた、同期の単純化やレイテンシの調整を行うためのタイミング情報をアプリケーションに提供できます。

ほとんどの場合、コードでHALと直接やり取りすることはありません。Appleは、AUHALユニット(Mac OS X)およびAURemoteIOユニット(iPhone OS)と呼ばれる特殊なオーディオユニットを提供しています。これらを使用して、オーディオを別のオーディオユニットからハードウェアに渡すことができます。同様に、ハードウェアからの入力はAUHALユニット（またはiPhone OSではAURemoteIOユニット）を通じて転送され、後続のオーディオユニットで使用できるようになります（図 1-4を参照）。

図 1-4 HALとAUHALユニットを経由したハードウェア入力



AUHALユニット（またはAURemoteIOユニット）は、オーディオユニットとハードウェア間のオーディオデータの変換に必要なデータ変換やチャネルマッピングも処理します。

Mac OS XでのMIDIのサポート

Mac OS Xでは、Core MIDIはMIDIプロトコルをサポートするCore Audioの一部です（iPhone OSではMIDIを使用できません）。アプリケーションはCore MIDIを使用することで、キーボードやギターなどのMIDIデバイスと通信できます。MIDIデバイスからの入力は、MIDIデータとして格納したり、インストゥルメントユニットの制御に使用したりできます。また、アプリケーションからMIDIデータをMIDIデバイスに送信することもできます。

Core MIDIは、抽象化を使用してMIDIデバイスを表現し、標準のMIDIケーブル接続(MIDI In、MIDI Out、MIDI Thru)を模倣しつつ、低レイテンシの入出力を実現します。Core Audioはまた、ミュージックプレーヤーのプログラミングインターフェイスもサポートしており、MIDIベースの制御データや曲データの再生に使用できます。

MIDIプロトコルの機能の詳細については、MIDI Manufacturers Associationのサイト(http://midi.org)を参照してください。

Audio MIDI設定アプリケーション

Mac OS Xでは、ユーザはAudio MIDI設定アプリケーションを使用して次を実行できます。

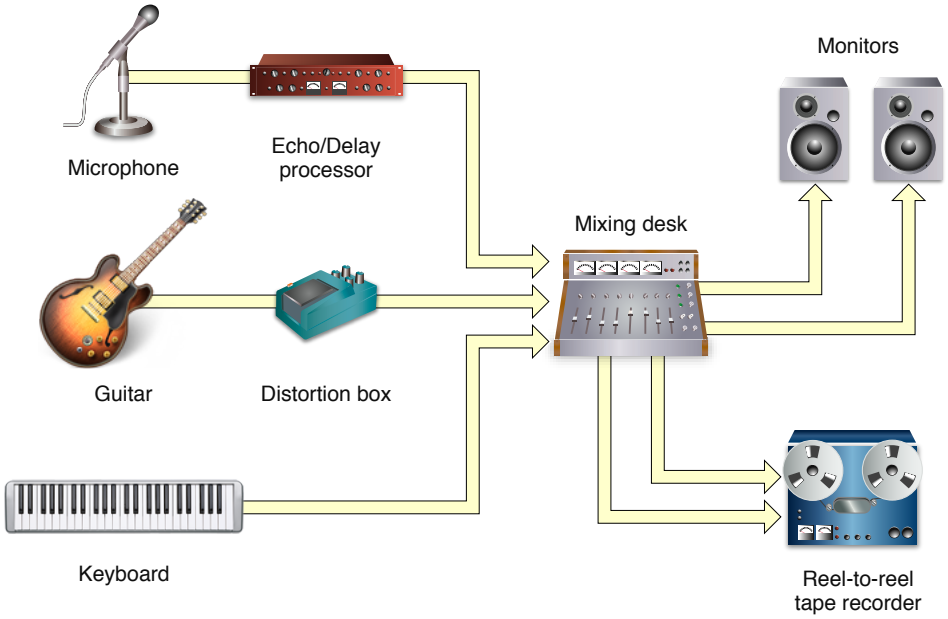
- デフォルトのオーディオ入力および出力デバイスの指定。
- 入力および出力デバイスのプロパティ（サンプリングレートやビット深度など）の設定。
- 使用可能なスピーカーへのオーディオチャネルのマッピング（ステレオや5.1サラウンドの場合など）。
- 機器セットの作成（機器セットの詳細については、「Mac OS Xでの機器セットの使用」（63 ページ）を参照してください）。
- MIDIネットワークとMIDIデバイスの設定。

Audio MIDI設定は`/Applications/Utilities`フォルダにあります。

Mac OS XのCore Audioレコーディングスタジオ

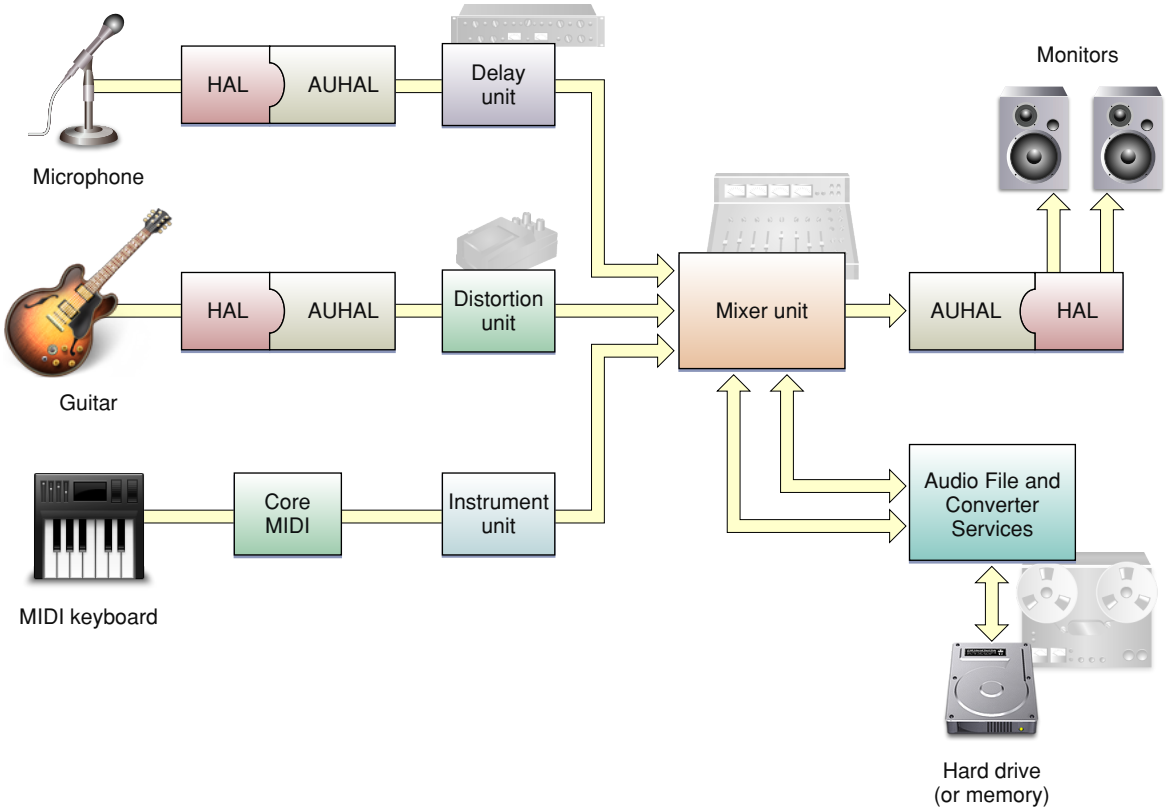
Core Audioを理解するための概念的な枠組みとして、コンピュータを使用しない従来のレコーディングスタジオを考えることができます。図 1-5に示すように、こうしたスタジオには、ミキシングデスクを供給する「実際の」楽器やエフェクトユニットが何台か置かれている場合があります。ミキサーは、出力をスタジオモニタやレコーディングデバイス（ここではやや古風なテープレコーダーで示しています）に転送できます。

図 1-5 コンピュータを使用しない簡易レコーディングスタジオ



従来のスタジオにある数多くの要素を、それと等価なソフトウェアベースの要素に置き換えることができます。この章ではそれらのすべてをすでに紹介しました。デスクトップコンピューティングプラットフォームにおいて、デジタルオーディオアプリケーションは、オーディオの録音、合成、編集、ミックス、処理、および再生を行うことができます。また、MIDIデータの録音、編集、処理、および再生もでき、ハードウェアとソフトウェアの両方のMIDI楽器とやり取りします。Mac OS Xでは、図 1-6に示すように、アプリケーションがCore Audioサービスを利用してこれらすべての作業を処理します。

図 1-6 Core Audioによる“レコーディングスタジオ”



図に示されているとおり、オーディオユニットはMac OS Xのオーディオ信号チェーンのほとんどを構成できます。その他のCore Audioインターフェイスは、アプリケーションがオーディオデータやMIDIデータをさまざまなフォーマットで取得してファイルや出力デバイスに出力できるように、アプリケーションレベルのサポートを提供します。Core Audioを構成するインターフェイスの詳細については、「Core Audioサービス」（77 ページ）を参照してください。

Core Audioを使用すると、デスクトップコンピュータ上にレコーディングスタジオを模倣する以上のことを実現できます。サウンドエフェクトの再生から、圧縮したオーディオファイルの作成や、ゲームプレーヤーを夢中にさせる音響効果に至るまで、あらゆる処理にCore Audioを使用できます。

iPhoneやiPod touchなどのモバイル機器では、バッテリーの持続時間が延びるように、オーディオ環境とコンピューティングリソースが最適化されます。結局のところ、iPhoneの最も本質的な姿は電話機としての存在です。開発やユーザの視点からは、iPhoneを仮想レコーディングスタジオの中心にすえることは意味を持たないでしょう。その一方で、iPhoneの特殊な機能である、高度なポータビリティ、組み込みのBonjourネットワーク、マルチタッチインターフェイス、加速度計および位置特定APIなどを利用することで、デスクトップでは決してできないオーディオアプリケーションを創造し、開発できます。

Core Audio SDKを使用したMac OS Xでの開発

オーディオデベロッパを支援するため、AppleはMac OS XのCore Audio向けのソフトウェア開発キット(SDK)を提供しています。このSDKには、オーディオサービスやMIDIサービスだけでなく、診断ツールやテストアプリケーションも網羅した、多数のコードサンプルが含まれています。たとえば、次のようなものがあります。

- システムのグローバルなオーディオ状態（接続されたハードウェアデバイスなど）とやり取りするテストアプリケーション(HALLab)。
- 基本的なオーディオユニットホスティングアプリケーション(AU Lab)。AU Labアプリケーションは、作成したオーディオユニット（「オーディオユニット」 （15 ページ）を参照）をテストする際に必要になります。
- オーディオファイルを読み込んで再生するためのサンプルコード(PlayFile)と、MIDIファイルを読み込んで再生するためのサンプルコード(PlaySequence)。

この文書では、一般的な作業を実行する方法を詳しく説明した、Core Audio SDKのその他のサンプルについて、その参照先を示しています。

このSDKにはまた、Mac OS X用のオーディオユニットを開発するためのC++フレームワークも付属しています。このフレームワークによって、Component Managerプラグインインターフェイスの詳細を知る必要がなくなるので、必要な作業量が減少します。さらに、SDKには共通の種類のオーディオユニットに対応したテンプレートも付属しており、ほとんどの場合、デベロッパ自身のカスタムのオーディオユニットに適用するメソッドをオーバーライドするだけで済みます。サンプルのオーディオユニットプロジェクトの中には、これらのテンプレートやフレームワークがすでに使用されているものがあります。フレームワークやテンプレートの使いかたの詳細については、『*Audio Unit Programming Guide*』を参照してください。

注： Appleはオーディオユニットの開発を支援するため、C++のオーディオユニットフレームワークをサンプルコードとして提供しています。このフレームワークは、必要に応じて自由に変更や改良を加えることができます。

Core Audio SDKでは、開発環境としてXcodeの使用を想定しています。

最新のSDKはhttp://developer.apple.com/sdk/からダウンロードできます。インストール後、SDKのファイルは`/Developer/Examples/CoreAudio`に置かれます。HALLabおよびAU Labアプリケーションは`/Developer/Applications/Audio`にあります。

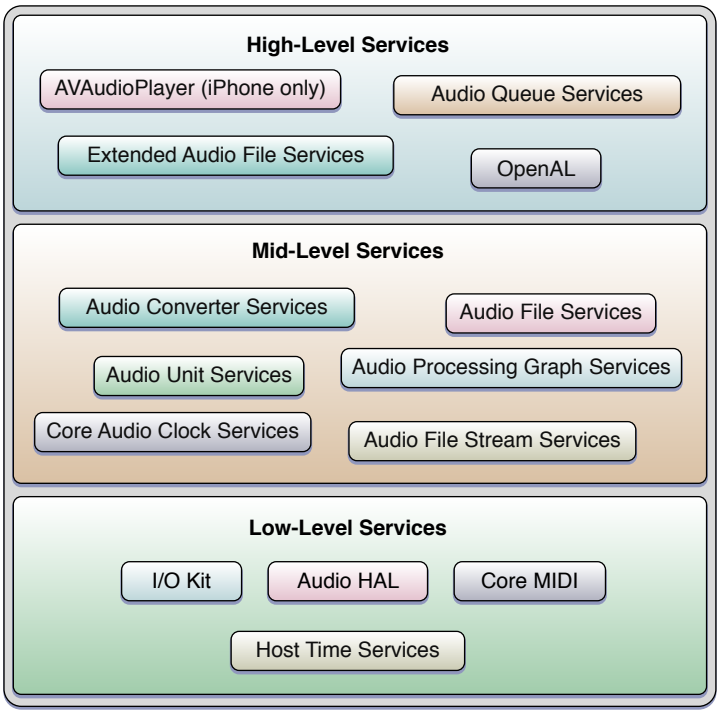
Core Audioの基礎

Appleは、Core Audioへのソフトウェアインターフェイスの設計にあたって、階層化され協調的でタスクにフォーカスしたアプローチを採用しました。この章の最初の2つのセクションでは、これらのインターフェイスの概要を紹介し、それらがどのように連携するかについて説明します。その後、Core Audio全体にわたって使用している、設計原則、使用パターン、およびプログラミングイディオムについて説明します。この章の後半の各セクションでは、Core Audioでファイル、ストリーム、録音と再生、およびプラグインの操作方法を説明します。

APIのアーキテクチャレイヤ

Core Audioのプログラミングインターフェイスは、図 2-1に示すように3つのレイヤに整理されています。

図 2-1 Core Audioの3つのAPIレイヤ



一番下のレイヤには次のサービスがあります。

- I/O Kit。ドライバとやり取りします。
- オーディオハードウェア抽象化層（オーディオHAL）。デバイスにもドライバにも依存しない、ハードウェアへのインターフェイスを提供します。

- Core MIDI。MIDIストリームやMIDIデバイスを操作するためのソフトウェア抽象化を提供します。
- Host Time Services。コンピュータのクロックへのアクセスを提供します。

Mac OS Xのアプリケーションの場合は、可能な限り最大のリアルタイムパフォーマンスを必要とする際はこれらのテクノロジーを直接利用するように記述できます。しかし、多くのオーディオアプリケーションは、このレイヤにアクセスすることはありません。実際、iPhone OSのCore Audioは、より上位のインターフェイスを使用してリアルタイムオーディオを実現するための手段を提供しています。たとえば、OpenALはダイレクトI/Oを利用して、ゲームでのリアルタイムオーディオを実現しています。その結果、モバイルプラットフォームに適した、非常に小規模の限定されたAPIセットとなっています。

Core Audioの中間のレイヤには、データフォーマットの変換、ディスクに対する読み書き、ストリームの解析、およびプラグインの操作を行うサービスがあります。

- Audio Converter Servicesは、アプリケーションでのオーディオデータフォーマットコンバータの操作を可能にします。
- Audio File Servicesは、ディスクベースのファイルに対するオーディオデータの読み書きをサポートします。
- Audio Unit ServicesとAudio Processing Graph Servicesは、アプリケーションでのデジタル信号処理(DSP)プラグイン（イコライザやミキサーなど）の操作を可能にします。
- Audio File Stream Servicesは、ストリームを解析できるアプリケーションの開発を可能にします。たとえば、ネットワーク接続を経由してストリーミングされるファイルを再生する場合などに使用します。
- Core Audio Clock Servicesは、オーディオとMIDIの同期をサポートするほか、時間基準の変換もサポートします。
- この図では示していませんが、Audio Format Servicesという小規模のAPIは、アプリケーションでのオーディオデータフォーマットの管理を支援します。

iPhone OSのCore Audioでは、図 1-2 （14 ページ）に示すように、これらのサービスのほとんどがサポートされています。

Core Audioの一番上のレイヤには、下のレイヤの機能を結合した効率的なインターフェイスがあります。

- Audio Queue Servicesは、オーディオの録音、再生、一時停止、ループ再生、および同期を可能にします。このサービスは、圧縮されたオーディオフォーマットを扱う場合に、必要に応じてコーデックを利用します。
- `AVAudioPlayer`クラスは、iPhone OSアプリケーションでオーディオの再生およびループ再生を行うためのObjective-C簡易インターフェイスを提供します。このクラスは、iPhone OSでサポートされているすべてのオーディオフォーマットを処理し、巻き戻しや早送りなどの機能を実装するための単純明快な手段を提供します。
- Extended Audio File Servicesは、Audio File ServicesとAudio Converter Servicesの機能を結合したものです。非圧縮または圧縮済みのサウンドファイルを読み書きするための統合されたインターフェイスを提供します。
- OpenALは、定位オーディオのためのオープンソースOpenAL標準のCore Audioによる実装です。OpenALは、システムが提供する3D Mixerオーディオユニットの上層に組み込まれています。OpenALはすべてのアプリケーションで使用できますが、ゲームの開発に最も適しています。

フレームワーク

/System/Library/Frameworks/にある、Core AudioのAPIフレームワークに注目することにより、Core Audioを別の視点から眺めることができます。このセクションでは、フレームワークについて簡単に触れます。ここでの説明を参考にして、Core Audioの各レイヤを構成する要素を見つける際の手がかりとしてください。

Core Audioフレームワークは、ここに示すほかのフレームワークを包括するアンブレラではなく、ピアとなるフレームワークです。

- Audio Toolboxフレームワーク(AudioToolbox.framework)は、Core Audioの中レベルおよび高レベルのサービスのためのインターフェイスを提供します。iPhone OSでは、このフレームワークにAudio Session Servicesが含まれています。Audio Session Servicesは、携帯電話やiPodとして機能するデバイスのもとで、アプリケーションのオーディオ動作を管理するためのインターフェイスです。
- Audio Unitフレームワーク(AudioUnit.framwork)は、アプリケーションでのオーディオプラグイン（オーディオユニットやコーデックを含む）の操作を可能にします。
- AV Foundationフレームワーク(AVFoundation.framework)はiPhone OSで使用できるフレームワークで、AVAudioPlayerクラスを提供します。このクラスは、オーディオ再生のための効率化されたObjective-C簡易インターフェイスです。
- Core Audioフレームワーク(CoreAudio.framework)は、Core Audio全体で使用されるデータ型を提供するほか、低レベルのサービスのインターフェイスも提供します。
- Core Audio Kitフレームワーク(CoreAudioKit.framework)は、オーディオユニットのユーザインターフェイスを作成するための小規模のAPIを提供します。このフレームワークはMac OS Xでのみ提供されています。
- Core MIDIフレームワーク(CoreMIDI.framework)は、アプリケーションでのMIDIデータの操作とMIDIネットワークの設定を可能にします。このフレームワークはMac OS Xでのみ提供されています。
- Core MIDI Serverフレームワーク(CoreMIDIServer.framework)は、MIDIドライバによるMac OS X MIDIサーバとの通信を可能にします。このフレームワークはMac OS Xでのみ提供されています。
- OpenALフレームワーク(OpenAL.framework)は、オープンソースのポジショナルオーディオ技術であるOpenALを操作するためのインターフェイスを提供します。

付録「Core Audioフレームワーク」（71 ページ）では、これらすべてのフレームワークと、各フレームワークに含まれているヘッダファイルについて説明しています。

プロキシオブジェクト

Core Audioでは、ファイル、ストリーム、オーディオプレーヤーなどを表現する手段として**プロキシオブジェクト**の概念が使用されています。たとえば、アプリケーションからディスク上のオーディオファイルを操作するなどの必要がある場合、最初の手順として、AudioFileID型のオーディオファイルオブジェクトをインスタンス化します。このオブジェクトはAudioFile.hヘッダファイルで次のように不透過のデータ構造体として宣言されています。

```
typedef struct OpaqueAudioFileID *AudioFileID;
```

オーディオファイルオブジェクトをインスタンス化し、そのオブジェクトに結び付けられた実際のオーディオファイルを作成するには、AudioFileCreateWithURL関数を呼び出します。この関数は、新しいオーディオファイルオブジェクトへの参照を返します。それ以降は、プロキシオブジェクトとやり取りすることによって実際のオーディオファイルを操作します。

この種のパターンはCore Audio全体を通じて一貫しており、操作対象がオーディオファイルでも、iPhoneのオーディオセッションでも（「オーディオセッション：Core Audioとの連携」（38 ページ）を参照）、さらにはハードウェアデバイスであっても同様です。

プロパティ、スコープ、要素

ほとんどのCore Audioインターフェイスでは、オブジェクトの状態の管理やオブジェクトの動作の詳細設定を行うために**プロパティ**のメカニズムを使用します。プロパティはキーと値のペアです。

- プロパティキーは、通常はkAudioFilePropertyFileFormatやkAudioQueueDeviceProperty_NumberChannelsなどのように、ニーモニック名の付いた列挙子定数です。
- プロパティ値は、void*、Float64、AudioChannelLayout構造体などのように、その用途に適した特定のデータ型の値です。

Appleによって定義されたプロパティは数多くあります。それらの定義はCore Audioフレームワークの各種のヘッダファイルの中にあります。Mac OS XのAudio Unit Servicesなど、一部のCore Audioインターフェイスでは、独自のプロパティも定義できます。

Core Audioインターフェイスでは、プロパティ値をオブジェクトから取得する際や、プロパティ値を変更する際（書き込み可能プロパティの場合）に、アクセサ関数を使用します。また、プロパティに関する情報を取得するためのアクセサ関数もあります。たとえば、Audio Unit Servicesの関数AudioUnitGetPropertyInfoは、指定されたプロパティ値のデータ型のサイズと、そのプロパティ値を変更できるかどうかを示します。Audio Queue Servicesの関数AudioQueueGetPropertySizeは、指定されたプロパティ値のサイズを取得します。

Core Audioインターフェイスは、プロパティが変更されたことをアプリケーションに知らせるためのメカニズムを提供しています。これについては、次のセクション「コールバック関数：Core Audioとのやり取り」（27 ページ）で説明します。

場合によっては、プロパティがオーディオオブジェクト全体に適用されることがあります。たとえば、再生オーディオキューオブジェクトでオーディオレベル測定機能を有効にするには、オブジェクトのkAudioQueueProperty_EnableLevelMeteringプロパティの値をtrueに設定します。

また、内部構造体を持ち、その各要素が独自のプロパティセットを持つことができるCore Audioオブジェクトもあります。たとえば、オーディオユニットには、入力スコープ、出力スコープ、およびグローバルスコープがあります。オーディオユニットの入力スコープまたは出力スコープは、1つまたは複数の要素で構成され、各要素はオーディオハードウェアのチャネルバスに相当します。AudioUnitGetProperty関数を呼び出すときにkAudioUnitProperty_AudioChannelLayoutプロパティを指定する場合は、情報を必要とするオーディオユニットだけでなく、スコープ（入力または出力）と要素（0、1、2など）も指定します。

コールバック関数：Core Audioとのやり取り

Core Audioインターフェイスの多くは、コールバック関数を使用してアプリケーションと通信できます。Core Audioでは、次のような場合にコールバックを使用します。

- 新しいオーディオデータのセットをアプリケーションに提供する場合（録音の場合など。その後、コールバックで新しいデータをディスクに書き込みます）。
- 新しいオーディオデータのセットをアプリケーションから要求する場合（再生の場合など。コールバックでは、ディスクからデータを読み取って提供します）。
- ソフトウェアオブジェクトの状態が変更されたことをアプリケーションに知らせる場合（コールバックで適切に対処します）。

コールバックを理解する方法の1つとして、誰が誰を呼び出すのかを逆方向から眺めてみるとよいでしょう。`AudioQueueNewOutput`などの通常の関数呼び出しでは、オペレーティングシステムの実装の中でAppleが定義した動作を、アプリケーションが呼び出します。呼び出し側では、見えないところでどのような処理が行われるのかはわからず、また、知る必要もありません。アプリケーションでは、再生オーディオキューオブジェクトを要求し、それを取得します。これが正しく処理される理由は、呼び出しの中で、関数のヘッダファイルで指定されている関数インターフェイスに従っているからです。

コールバックの場合には、デベロッパがアプリケーションで実装した動作を、オペレーティングシステムが必要と判断したときに呼び出します。アプリケーションの中でテンプレートに従ってコールバックを定義することによって、オペレーティングシステムはコールバックを正しく呼び出すことができます。たとえば、Audio Queue Servicesでは、デベロッパが実装できるコールバックのテンプレートが指定されており、それに従うことで、オーディオキューオブジェクトのプロパティが変更されたときにメッセージを取得したりメッセージに応答したりできるようになります。このコールバックテンプレートをリスト 2-1に示します。このコールバックテンプレートは`AudioQueue.h`ヘッダファイルで宣言されています。

リスト 2-1　コールバック関数のテンプレート

```
typedef void    (*AudioQueuePropertyListenerProc) (
                    void *                  inUserData,
                    AudioQueueRef           inAQ,
                    AudioQueuePropertyID    inID
                );
```

アプリケーションでコールバックを実装して使用するには、次の2つのことを実行します。

1. コールバック関数を実装します。たとえば、オーディオキューのプロパティリスナーコールバックを実装して、オーディオキューオブジェクトが実行されているか停止しているかに応じて、ユーザインターフェイスにあるボタンのタイトルと有効／無効の状態を更新することが考えられます。

2. やり取りの対象となるオブジェクトに、コールバック関数を登録します。コールバックを登録する方法の1つに、オブジェクトの作成時に行う方法があります。通常、この方法はオーディオデータの送受信を行うコールバックで使用します。オブジェクトを作成する関数呼び出しの中で、コールバックへの参照を関数パラメータとして渡します。コールバックを登録する別の方法として、専用の関数呼び出しを使用する方法があります。通常、この方法はプロパティリスナーの場合に使用します。これについてはこのセクションでこの後説明します。

リスト 2-2は、プロパティリスナーコールバック関数を実装して、再生オーディオキューオブジェクト内のプロパティの変更に応答する方法の一例を示しています。

リスト 2-2　プロパティリスナーコールバックの実装

```
static void propertyListenerCallback (
    void                    *inUserData,
    AudioQueueRef           queueObject,
    AudioQueuePropertyID    propertyID
) {
    AudioPlayer *player = (AudioPlayer *) inUserData;
        // 再生オブジェクトへの参照を取得する
    [player.notificationDelegate
updateUserInterfaceOnAudioQueueStateChange:player];
        // notificationDelegateクラスでUIの更新メソッドを実装する
}
```

（Objective-Cのクラス定義では、このコールバック定義をクラスの実装の外側に置きます。このため、関数本体には、再生オブジェクトへの参照を取得するステートメントがあります。この例では、`inUserData`パラメータがそのオブジェクトになります。この参照を、コールバックの登録時に、コールバックで使用できるようにします。これについては次の説明を参照してください。）

リスト 2-3に示すように、この特定のコールバックを登録します。

リスト 2-3　プロパティリスナーコールバックの登録

```
AudioQueueAddPropertyListener (
    self.queueObject,                // コールバックを呼び出すオブジェクト
    kAudioQueueProperty_IsRunning,   // 検知するプロパティのID
    propertyListenerCallback,        // コールバック関数への参照
    self
);
```

オーディオデータフォーマット

Core Audioによって、オーディオデータフォーマットの詳細を理解する必要性はなくなります。これにより、コードの中で特定のフォーマットを処理することが簡単になるだけでなく、オペレーティングシステムがサポートするすべてのフォーマットを1つのコードセットで操作できるようになります。

注： **オーディオデータフォーマット**は、サンプルレート、ビット深度、パケット化などを含めたオーディオデータそのものを表します。**オーディオファイルフォーマット**は、サウンドファイルのオーディオデータ、オーディオメタデータ、およびファイルシステムメタデータをディスク上にどのように配置するのかを表します。オーディオファイルフォーマットによっては、1種類のオーディオデータフォーマットしか格納できないものがあります（たとえば、MP3ファイルにはMP3オーディオデータのみ格納できます）。また、AppleのCAFフォーマットのように、各種のオーディオデータフォーマットを格納できるものもあります。

Core Audioの汎用データ型

Core Audioでは、2つの汎用データ型を使用してすべてのオーディオデータフォーマットを表現します。これらの型は、データ構造体`AudioStreamBasicDescription`（リスト 2-4）と`AudioStreamPacketDescription`（リスト 2-5）です。どちらも、`CoreAudioTypes.h`ヘッダファイルで宣言されています。詳細については、『*Core Audio Data Types Reference*』を参照してください。

リスト 2-4 `AudioStreamBasicDescription`データ型

```
struct AudioStreamBasicDescription {
    Float64 mSampleRate;
    UInt32  mFormatID;
    UInt32  mFormatFlags;
    UInt32  mBytesPerPacket;
    UInt32  mFramesPerPacket;
    UInt32  mBytesPerFrame;
    UInt32  mChannelsPerFrame;
    UInt32  mBitsPerChannel;
    UInt32  mReserved;
};
typedef struct AudioStreamBasicDescription  AudioStreamBasicDescription;
```

この構造体では、`mReserved`メンバの値は常に値`0`である必要があります。それ以外のメンバの値も`0`になる場合があります。たとえば、圧縮されたオーディオフォーマットでは、可変のサンプルあたりビット数が使用されます。そのようなフォーマットでは、`mBitsPerChannel`メンバの値は`0`になります。

名前について： このデータ型の名前には「stream」が付いていますが、Core Audioでは、非ストリーミングの標準ファイルを含め、オーディオデータフォーマットを表現する必要があるすべてのインスタンスにこのデータ型を使用します。これを「audio format basic description（オーディオフォーマット基本記述）」データ型と見なすこともできます。名前の「stream」は、オーディオデータをハードウェアやソフトウェアの周囲で移動する（つまり、ストリーミングする）必要がある場合に常にオーディオフォーマットが関与するという事実を指しています。

Core Audioに関する話題では「ASBD」という言葉（「audio stream basic description（オーディオストリーム基本記述）」の略語）をよく耳にしますが、この文書でもこの言葉を使用しています。

audio stream packet description（オーディオストリームパケット記述）型は、特定の圧縮されたオーディオデータフォーマットの場合に関与します。これについては、「オーディオデータパケット」（32 ページ）で説明します。

リスト 2-5　AudioStreamPacketDescriptionデータ型

```
struct  AudioStreamPacketDescription {
    SInt64  mStartOffset;
    UInt32  mVariableFramesInPacket;
    UInt32  mDataByteSize;
};
typedef struct AudioStreamPacketDescription AudioStreamPacketDescription;
```

固定ビットレートのオーディオデータの場合（「オーディオデータパケット」　（32 ページ）を参照）、この構造体のmVariableFramesInPacketメンバの値は0になります。

サウンドファイルのデータフォーマットの取得

ASBDのメンバはコードの中で手動で設定できます。その際、構造体のメンバの一部（または全部）について正しい値がわからない場合があります。構造体を具体的に設定するには、それらの値を0に設定してからCore Audioインターフェイスを使用します。

たとえば、リスト 2-6に示すように、Audio File Servicesを使用して、ディスク上のサウンドファイルのオーディオストリーム基本記述をすべて設定することができます。

リスト 2-6　サウンドファイルを再生するためのオーディオストリーム基本記述の取得

```
- (void) openPlaybackFile: (CFURLRef) soundFile {

    AudioFileOpenURL (
        (CFURLRef) self.audioFileURL,
        0x01,                   // 読み取り専用
        kAudioFileCAFType,
        &audioFileID
    );

    UInt32 sizeOfPlaybackFormatASBDStruct = sizeof ([self audioFormat]);

    AudioFileGetProperty (
        [self audioFileID],
        kAudioFilePropertyDataFormat,
        &sizeOfPlaybackFormatASBDStruct,
        &audioFormat          // ここにサウンドファイルのASBDが返される
    );
}
```

正準形のオーディオデータフォーマット

プラットフォームによっては、Core Audioで1つまたは2つの「正準形」のオーディオデータフォーマットが存在します。ここでの「正準形」とは、それらのフォーマットが次の条件を満たす可能性があることを意味します。

- 変換において中間フォーマットとして必要になること
- Core Audioのサービスが最適化されるフォーマットであること
- 特にASBDを指定しない場合のデフォルトの（想定される）フォーマットであること

Core Audioにおける正準形のフォーマットは次のとおりです。

- **iPhone OSの入力および出力**：リニアPCM（16ビット整数サンプル）
- **iPhone OSのオーディオユニットおよびその他のオーディオ処理**：非インターリーブのリニアPCM（8.24ビット固定小数点サンプル）
- **Mac OS Xの入力および出力**：リニアPCM（32ビット浮動小数点サンプル）
- **Mac OS Xのオーディオユニットおよびその他のオーディオ処理**：非インターリーブのリニアPCM（32ビット浮動小数点サンプル）

完全なオーディオストリーム基本記述の例を次に示します。これは、サンプルレートが44.1kHzの2チャネルの正準形のiPhoneオーディオユニットサンプルフォーマットを示しています。

```
struct AudioStreamBasicDescription {
    mSampleRate       = 44100.0;
    mFormatID         = kAudioFormatLinearPCM;
    mFormatFlags      = kAudioFormatFlagsAudioUnitCanonical;
    mBytesPerPacket   = 1 * sizeof (AudioUnitSampleType);    // 8
    mFramesPerPacket  = 1;
    mBytesPerFrame    = 1 * sizeof (AudioUnitSampleType);    // 8
    mChannelsPerFrame = 2;
    mBitsPerChannel   = 8 * sizeof (AudioUnitSampleType);    // 32
    mReserved         = 0;
};
```

ここで値として使用している定数とデータ型は`CoreAudioTypes.h`ヘッダファイルで宣言されており、『*Core Audio Data Types Reference*』で説明されています。ここで`AudioUnitSampleType`データ型（およびオーディオI/Oの処理時に`AudioSampleType`データ型）を使用することによって、ASBDがiPhone OSとMac OS Xとでプラットフォームに依存しないことが保証されます。

Core Audioのオーディオデータフォーマットについての説明の締めくくりとして、さらに2つの概念であるマジッククッキーとパケットについて次に説明します。

マジッククッキー

Core Audioの世界における**マジッククッキー**とは、圧縮されたサウンドファイルまたはストリームに添付される不透過のメタデータセットのことです。デコーダはこのメタデータから、ファイルやストリームを正しく伸長するのに必要な詳細情報を得ます。デベロッパはマジッククッキーをブラックボックスとして扱い、Core Audio関数を使って、格納されているメタデータをコピーし、読み取り、使用します。

たとえば、リスト2-7は、サウンドファイルからマジッククッキーを取得して再生オーディオキューオブジェクトに提供するメソッドを示しています。

リスト 2-7　サウンドファイル再生時のマジックマジッククッキーの使用

```
- (void) copyMagicCookieToQueue:(AudioQueueRef) queue fromFile: (AudioFileID)
file {

    UInt32 propertySize = sizeof (UInt32);

    OSStatus result = AudioFileGetPropertyInfo (
                            file,
```

```
                            kAudioFilePropertyMagicCookieData,
                            &propertySize,
                            NULL
                        );

    if (!result && propertySize) {

        char *cookie = (char *) malloc (propertySize);

        AudioFileGetProperty (
            file,
            kAudioFilePropertyMagicCookieData,
            &propertySize,
            cookie
        );

        AudioQueueSetProperty (
            queue,
            kAudioQueueProperty_MagicCookie,
            cookie,
            propertySize
        );

        free (cookie);
    }
}
```

オーディオデータパケット

前章では、パケットを、1つまたは複数のフレームの集まりとして定義しました。パケットは、対象のオーディオデータフォーマットにおいて意味のある最小のフレームセットです。この理由から、オーディオファイルの時間単位を表すのに最適なオーディオデータ単位になります。Core Audioにおける同期は、パケットの数をカウントすることによって機能します。リスト 2-8 （33 ページ）に示すように、パケットを使用して有益なオーディオデータバッファサイズを計算することができます。

すべてのオーディオデータフォーマットは、そのパケットの構成方法によって定義される側面があります。リスト 2-4 （29 ページ）に示すように、オーディオストリーム基本記述データ構造体では、フォーマットのパケットに関する基本情報が`mBytesPerPacket`および`mFramesPerPacket`メンバによって示されます。さらに情報を必要とするフォーマットの場合は、この後説明するように、オーディオストリームパケット記述データ構造体を使用します。

オーディオデータフォーマットでは次の3種類のパケットが使用されます。

- CBR（固定ビットレート）フォーマット。リニアPCMやIMA/ADPCMなど。すべてのパケットのサイズは同じです。
- VBR（可変ビットレート）フォーマット。AAC、Apple Lossless、MP3など。すべてのパケットは同じフレーム数ですが、各サンプル値のビット数は可変です。
- VFR（可変フレームレート）フォーマット。各パケットのフレーム数は可変です。この種のフォーマットで一般に使用されているものはありません。

Core AudioでVBRまたはVFRフォーマットを使用するには、オーディオストリームパケット記述構造体（リスト 2-5 （30 ページ））を使用します。このような構造体はそれぞれ、サウンドファイル内の単一のパケットを表します。VBRまたはVFRのサウンドファイルを録音または再生するには、これらの構造体の配列が必要になり、構造体がファイル内のパケットごとに1つずつ必要になります。

Audio File ServicesおよびAudio File Stream Servicesのインターフェイスを使用すると、パケットを操作できます。たとえば、AudioFile.h内のAudioFileReadPackets関数は、ディスク上のサウンドファイルから読み取られた一連のパケットを提供し、それらをバッファに格納します。また同時に、それらの各パケットを表すAudioStreamPacketDescription構造体の配列も提供します。

CBRおよびVBRフォーマット（つまり、一般に使用されているすべてのフォーマット）では、対象のオーディオファイルまたはストリームについて、毎秒パケット数が固定されています。これには、パケット化によってフォーマットの時間単位が自動的に決まる、という便利な了解事項があります。パケット化は、アプリケーションで実用上のオーディオデータバッファのサイズを計算するときに使用できます。たとえば、次のメソッドは、与えられたオーディオデータの所要時間でバッファを埋めるために読み取る必要があるパケット数を決定します。

リスト 2-8　パケット化に基づく再生バッファサイズの計算

```
- (void) calculateSizesFor:(Float64) seconds {

    UInt32 maxPacketSize;
    UInt32 propertySize = sizeof (maxPacketSize);

    AudioFileGetProperty (
        audioFileID,
        kAudioFilePropertyPacketSizeUpperBound,
        &propertySize,
        &maxPacketSize
    );

    static const int maxBufferSize = 0x10000;   // 最大サイズを64Kに制限する
    static const int minBufferSize = 0x4000;    // 最小サイズを16Kに制限する

    if (audioFormat.mFramesPerPacket) {
        Float64 numPacketsForTime =
            audioFormat.mSampleRate / audioFormat.mFramesPerPacket * seconds;
        [self setBufferByteSize:numPacketsForTime * maxPacketSize];
    } else {
        // パケットあたりフレーム数が0の場合、コーデックではパケットと時間の間の
        // 関係がわからないので、デフォルトのバッファサイズを返す
        [self setBufferByteSize:
            maxBufferSize > maxPacketSize ? maxBufferSize :maxPacketSize];
    }

    // 指定した範囲にバッファサイズを固定する
    if (bufferByteSize > maxBufferSize && bufferByteSize > maxPacketSize) {
        [self setBufferByteSize: maxBufferSize];
    } else {
        if (bufferByteSize < minBufferSize) {
            [self setBufferByteSize: minBufferSize];
        }
    }

    [self setNumPacketsToRead: self.bufferByteSize / maxPacketSize];
}
```

データフォーマット変換

あるフォーマットから別のフォーマットにオーディオデータを変換するには、オーディオコンバータを使用します。サンプルレートの変更やインターリーブ／インターリーブ解除などの単純な変換を実行できるほか、オーディオの圧縮／伸長などの複雑な変換も実行できます。次の3種類の変換を使用できます。

- オーディオフォーマット（AAC (Advanced Audio Coding)など）をリニアPCMフォーマットにデコードする。
- リニアPCMデータを別のオーディオフォーマットに変換する。
- リニアPCMのバリエーション間で変換を行う（たとえば、16ビット符号付き整数のリニアPCMを8.24固定小数点のリニアPCMに変換する）。

Audio Queue Services（「Audio Queue Servicesを使用した録音と再生」（43 ページ）を参照）を使用するときは、適切なコンバータが自動的に取得されます。Mac OS Xでは、Audio Codec Servicesを使用して特殊なオーディオコーデックを作成できます。たとえば、デジタル権利管理(DRM)や専用のオーディオフォーマットを処理するコーデックを作成できます。カスタムのコーデックを作成したら、オーディオコンバータを使用してアクセスし、使用することができます。

Mac OS Xでオーディオコンバータを明示的に使用するときは、特定のコンバータインスタンスを指定して変換関数を呼び出し、入力データを探す場所と出力を書き込む場所を指定します。ほとんどの変換では、入力データを周期的にコンバータに供給するためのコールバック関数が必要になります。オーディオコンバータの使用例については、Core Audio SDKの`Services/AudioFileTools`にある`SimpleSDK/ConvertFile`および`AFConvert`コマンドラインツールを参照してください。

「Mac OS Xでサポートされているオーディオファイルフォーマットとオーディオデータフォーマット」（87 ページ）では、圧縮されたフォーマットとリニアPCMフォーマットの間でどちらか一方向の変換を行う標準のCore Audioコーデックの一覧を記載しています。コーデックの詳細については、「コーデック」（51 ページ）を参照してください。

サウンドファイル

アプリケーションでサウンドファイルを操作する必要がある場合は、Core Audioの中レベルのサービスの1つであるAudio File Servicesをいつでも使用できます。Audio File Servicesが提供する強力な抽象化により、ファイルに格納されているオーディオデータとメタデータへのアクセスや、サウンドファイルの作成が可能です。

基本情報（一意のファイルID、ファイルタイプ、およびデータフォーマット）の操作に加えて、Audio File Servicesでは、リージョンとマーカ、ループ再生、再生の向き、SMPTEタイムコードなども操作できます。

また、システム特性を調べる場合にもAudio File Servicesを使用します。これには、`AudioFileGetGlobalInfoSize`関数（必要な情報を保持するためのメモリを割り当てることができます）および`AudioFileGetGlobalInfo`関数（その情報を取得します）を使用します。`AudioFile.h`で宣言されているプロパティの詳細な一覧を使用して、プログラムから次のようなシステム特性を取得できます。

- 読み取り可能なファイルタイプ

- 書き込み可能なファイルタイプ
- それぞれの書き込み可能なタイプについて、ファイルに書き込むことができるオーディオデータフォーマット

ここではまた、さらに2つのCore Audioテクノロジーについても紹介します。

- Audio File Stream Services。オーディオデータをディスクまたはネットワークストリームから読み取ることができるようにする、オーディオストリーム解析インターフェイスです。
- Extended Audio File Services（Mac OS Xのみ）。Audio File ServicesとAudio Converter Servicesの機能がカプセル化されたもので、コードが簡略化されます。

新しいサウンドファイルの作成

録音のための新しいサウンドファイルを作成するには、次が必要になります。

- ファイルのファイルシステムパス（`CFURL`または`NSURL`オブジェクトの形式）。
- 作成するファイルのタイプを示す識別子。`AudioFile.h`のAudio File Typesの列挙で宣言されています。たとえば、CAFファイルを作成するには、`kAudioFileCAFType`識別子を使用します。
- ファイル内に格納するオーディオデータのオーディオストリーム基本記述。多くの場合、ASBDを部分的に設定し、後でAudio File Servicesに問い合わせて自動的に設定することができます。

これら3つの情報は、`AudioFileCreateWithURL`関数へのパラメータとしてAudio File Servicesに渡します。この関数によってファイルが作成され、`AudioFileID`オブジェクトが返されます。次に、このオブジェクトを使用してサウンドファイルとの詳細なやり取りを行います。リスト 2-9 （35 ページ）は、この関数呼び出しを示しています。

リスト 2-9　サウンドファイルの作成

```
AudioFileCreateWithURL (
    audioFileURL,
    kAudioFileCAFType,
    &audioFormat,
    kAudioFileFlags_EraseFile,
    &audioFileID   // ここで関数によって新しいファイルオブジェクトが提供される
);
```

サウンドファイルを開く

サウンドファイルを再生のために開くには、`AudioFileOpenURL`関数を使用します。この関数では、使用するファイルのURL、ファイルタイプのヒント定数、およびファイルアクセスパーミッションを指定します。`AudioFileOpenURL`は、ファイルの一意のIDを返します。

次に、プロパティ識別子と、`AudioFileGetPropertyInfo`および`AudioFileGetProperty`関数を使用して、ファイルに関する必要な情報を取得します。よく使用するプロパティ識別子として、たとえば次のものがあります（これらの機能は名前から十分推測できます）。

- `kAudioFilePropertyFileFormat`
- `kAudioFilePropertyDataFormat`

- kAudioFilePropertyMagicCookieData
- kAudioFilePropertyChannelLayout

こうした識別子はAudio File Servicesで数多く使用でき、ファイルに存在する可能性のあるメタデータ（リージョンマーカ、著作権情報、再生テンポなど）を取得できます。

Podcastなど、VBRファイルが長い場合には、パケットテーブル全体を取得するのにかなりの時間がかかる可能性があります。そのような場合は、kAudioFilePropertyPacketSizeUpperBoundおよびkAudioFilePropertyEstimatedDurationの2つのプロパティ識別子が特に便利です。これらを使用すると、ファイル全体を解析して正確な数値を取得する代わりに、VBRサウンドファイルの大まかな所要時間やパケット数を見積もることができます。

サウンドファイルに対する読み書き

iPhone OSでは、通常はAudio File Servicesを使用してサウンドファイルに対するオーディオデータの読み書きを行います。Audio File Servicesを使用する場合の読み取りと書き込みは、基本的には互いに類似した操作です。どちらも、完了するまで操作がブロックされ、どちらもバイトまたはパケットを使用して処理できます。ただし、特別な必要がない限り、常にパケットを使用してください。

- VBRデータの場合は、パケットによる読み書きが唯一の選択肢です。
- パケットベースの操作を使用することで、所要時間の計算がはるかに簡単になります。

iPhone OSでは、ディスクからオーディオデータを読み取る場合のもう1つの選択肢として、Audio File Stream Servicesがあります。このテクノロジーの概要については、この章で後述する「サウンドストリーム」（37 ページ）を参照してください。

Audio Queue Servicesは、Core Audioの録音および再生用のインターフェイスであり、Audio Toolboxフレームワークの`AudioQueue.h`ヘッダファイルで宣言されています。Audio Queue Servicesの概要については、この章で後述する「Audio Queue Servicesを使用した録音と再生」（43 ページ）を参照してください。

Extended Audio File Services

Core AudioにはExtended Audio File Servicesと呼ばれる便利なAPIがあります。このインターフェイスにはAudio File ServicesとAudio Converter Servicesの重要な関数が含まれており、リニアPCMからの、またはリニアPCMへの自動データ変換機能を実現します。この詳細については、「Mac OS Xでのオーディオデータの読み書き」（59 ページ）を参照してください。

iPhoneのオーディオファイルフォーマット

iPhone OSでは、表 2-1（37 ページ）に示すオーディオファイルフォーマットがサポートされています。iPhone OSで使用できるオーディオデータフォーマットの詳細については、「コーデック」（51 ページ）を参照してください。

表 2-1 iPhone OSのオーディオファイルフォーマット

フォーマット名	フォーマットファイル名拡張子
AIFF	`.aif`, `.aiff`
CAF	`.caf`
MPEG-1 Layer 3	`.mp3`
MPEG-2またはMPEG-4 ADTS	`.aac`
MPEG-4	`.m4a`, `.mp4`
WAV	`.wav`

CAFファイル

iPhone OSおよびMac OS Xには、ネイティブのオーディオファイルフォーマットであるCore Audio File (*CAF*)ファイルフォーマットがあります。CAFフォーマットはMac OS X v10.4「Tiger」で導入されたもので、iPhone OS 2.0以降で使用できます。CAFには、プラットフォームでサポートされているどのオーディオデータフォーマットも格納することができるというユニークな特長があります。

AIFFファイルやWAVEファイルとは異なり、CAFファイルにはサイズの制限がありません。また、CAFファイルはチャネル情報やテキスト注釈などの広範なメタデータをサポートできます。CAFファイルフォーマットの詳細については、『*Apple Core Audio Format Specification 1.0*』を参照してください。

サウンドストリーム

ディスクベースのサウンドファイルとは異なり、**オーディオファイルストリーム**は、その先頭や末尾にアクセスできない場合があるオーディオデータです。たとえば、インターネットラジオプレーヤーアプリケーションを開発する場合などに、ストリームを取り扱います。プロバイダは、通常は各自のストリームを絶えず送信しています。ユーザが「再生(Play)」を押してラジオなどを聴くとき、アプリケーションでは、その時点で送られているデータの内容（オーディオパケットの先頭、途中、または末尾、あるいはマジッククッキーの可能性もあります）に関係なく、ストリームに追随して再生する必要があります。

また、サウンドファイルとは異なり、ストリームのデータは信頼性をもって使用できるとは限りません。ストリームの取得元であるネットワークの変動によって、欠落、不連続、または一時停止が生じている可能性があります。

Audio File Stream Servicesを使用すると、アプリケーションでストリームとそのすべての複雑な処理を操作でき、解析を任せることができます。

Audio File Stream Servicesを使用するには、`AudioFileStreamID`型のオーディオファイルストリームオブジェクトを作成します。このオブジェクトはストリームそれ自体のプロキシとして機能します。また、アプリケーションでこのオブジェクトを使用し、プロパティを通じてストリームの状況を把握することができます（「プロパティ、スコープ、要素」（26ページ）を参照）。たとえば、Audio File Stream Servicesによってストリームのビットレートが判別されると、オーディオファイルストリームオブジェクトの`kAudioFileStreamProperty_BitRate`プロパティが設定されます。

解析はAudio File Stream Servicesが実行するので、提供されるオーディオデータのセットやその他の情報に応答することがアプリケーションの役割になります。この方法でアプリケーションが応答できるようにするには、2つのコールバック関数を定義します。

まず必要なのは、オーディオファイルストリームオブジェクトのプロパティの変更に応答するコールバックです。このコールバックは、少なくとも`kAudioFileStreamProperty_ReadyToProducePackets`プロパティの変更に応答するように記述します。このプロパティを使用するシナリオは、通常は次のようになります。

1. ユーザが「再生(Play)」を押すか、そうでなければストリームの再生の開始を要求します。
2. Audio File Stream Servicesによってストリームの解析が開始されます。
3. 十分な数のオーディオデータパケットが解析されて、まとめてアプリケーションに送られると、Audio File Stream Servicesによって、オーディオファイルストリームオブジェクトの`kAudioFileStreamProperty_ReadyToProducePackets`プロパティが`true`（実際には値1）に設定されます。
4. Audio File Stream Servicesによって、プロパティIDの値を`kAudioFileStreamProperty_ReadyToProducePackets`としてアプリケーションのプロパティリスナーコールバックが呼び出されます。
5. プロパティリスナーコールバックで適切な処理（ストリームを再生するためのオーディオキューオブジェクトの設定など）を行います。

次に必要なのは、オーディオデータを処理するコールバックです。Audio File Stream Servicesによる完全なオーディオデータパケットのセットの収集が完了するたびに、Audio File Stream Servicesからこのコールバックが呼び出されます。このコールバックは、受信したオーディオを処理するように定義します。通常は、受信したオーディオをまとめてAudio Queue Servicesに送り、すぐに再生します。再生の詳細については、セクション「Audio Queue Servicesを使用した録音と再生」（43 ページ）を参照してください。

オーディオセッション：Core Audioとの連携

iPhone OSでは、アプリケーションは電話の着信に応えるなど、場合によってはアプリケーションの実行よりも重要なことをデバイス上で実行します。アプリケーションでサウンドを再生しているときに電話の着信があった場合、iPhoneでは正しい対応をする必要があります。

第1段階では、この「正しい対応」とは、ユーザが予測していることを満たすことを意味します。第2段階では、iPhoneが要求の競合を解決する際に、個々の実行中のアプリケーションについてそれぞれの現在の状態を考慮する必要があることも意味します。

オーディオセッションは、アプリケーションとiPhone OSとのやり取りを仲介するオブジェクトです。すべてのiPhone OSアプリケーションには、それぞれ1つだけオーディオセッションが与えられます。デベロッパは、アプリケーションのオーディオの意図が伝わるように、オーディオセッションを設定します。まず、アプリケーションの動作をどのようにする必要があるのかに関して、いくつかの質問に答えてみましょう。

- 割り込み（電話の着信など）に対してアプリケーションがどのように反応するか。

- アプリケーションのサウンドを、ほかに実行中のアプリケーションから出力されるアプリケーションの音とミックスするか、それとも、ほかのアプリケーションの音は鳴らないようにするか。
- オーディオ経路の変化（ヘッドセットの抜き差しなど）に対してアプリケーションがどのように反応するか。

これらの質問への答えを念頭に置きながら、オーディオセッションインターフェイス（AudioToolbox/AudioServices.hで宣言されています）を利用してオーディオセッションとアプリケーションを設定します。このインターフェイスによって提供されるプログラミング上の機能を表 2-2に示します。

表 2-2　オーディオセッションインターフェイスによって提供される機能

オーディオセッションの機能	説明
カテゴリ	カテゴリは、アプリケーションの一連のオーディオ動作を識別するキーです。カテゴリを設定することによって、オーディオの意図（たとえば、画面がロックされたときにオーディオの再生を続けるかどうかなど）をiPhone OSに伝えます。
割り込み、および出力先の変更	オーディオ割り込みの発生時、終了時、およびハードウェアのオーディオ出力先が変更されたときに、オーディオセッションから通知を送ります。これらの通知によって、より規模の大きいオーディオ環境における変更（電話の着信による割り込みなど）に、秩序正しく応答できるようになります。
ハードウェアの特性	オーディオセッションに問い合わせて、アプリケーションが実行されているデバイスの特性（ハードウェアのサンプルレート、ハードウェアのチャネル数、オーディオ入力を使用できるかどうかなど）を調べることができます。

オーディオセッションのデフォルトの動作

オーディオセッションは、デフォルトで有効ないくつかの動作を備えています。その具体的な内容は次のとおりです。

- ユーザが着信／サイレントスイッチを「サイレント」側に動かすと、オーディオは鳴らなくなります。
- ユーザがスリープ／スリープ解除ボタンを押して画面をロックするか、自動ロック時間が経過すると、アプリケーションのオーディオは鳴らなくなります。
- アプリケーションのオーディオが鳴り始めると、そのデバイスで出力されているほかのオーディオ（すでに再生中のiPodオーディオなど）は鳴らなくなります。

これらの一連の動作は、デフォルトオーディオセッションカテゴリである kAudioSessionCategory_SoloAmbientSoundによって指定されています。iPhone OSでは、ユーザインターフェイスのサウンドエフェクトから、VOIP (Voice over Internet Protocol)アプリケーションで使用するようなオーディオの同時入出力にいたるまで、幅広いオーディオのニーズに対応したカテゴリが提供されており、必要なカテゴリをアプリケーションの起動時および実行中に指定できます。

iPhoneのオーディオ開発を始めたばかりの段階では、オーディオセッションのデフォルト動作で十分です。ただし、それらのデフォルト動作は、例外的な場合を除いて出荷版アプリケーションには向いていません。このことについて次に説明します。

割り込み：アクティブ化と非アクティブ化

デフォルトのオーディオセッションに存在しない機能のうち特に注意すべきものの1つに、割り込み後のオーディオセッション自身の再アクティブ化機能があります。オーディオセッションには、アクティブと非アクティブという2つの主要な状態があります。アプリケーションでオーディオが機能できるのは、オーディオセッションがアクティブなときだけです。

起動時には、デフォルトのオーディオセッションはアクティブになっています。しかし、電話を着信すると、セッションがすぐに非アクティブ化され、オーディオは停止されます。このことを*割り込み*と呼びます。ユーザが着信した電話に出ないことを選んだ場合は、アプリケーションは動作を続行します。しかし、オーディオセッションが非アクティブなので、オーディオは機能しません。

アプリケーションで、OpenAL、I/Oオーディオユニット、またはAudio Queue Servicesをオーディオに使用する場合は、割り込みリスナーコールバック関数を記述して、割り込みの終了時にオーディオセッションを明示的にアクティブ化しなおす必要があります。サンプルコードと詳細については、『*Audio Session Programming Guide*』を参照してください。

`AVAudioPlayer`クラスを使用する場合は、オーディオセッションの再アクティブ化処理がクラスによって自動実行されます。

オーディオ入力が使用可能かを調べる方法

iPhone OSベースのデバイス上の録音アプリケーションは、ハードウェア入力が使用できる場合にのみ録音できます。これを確かめるには、オーディオセッションの`kAudioSessionProperty_AudioInputAvailable`プロパティを使用します。アプリケーションがiPod touch（第2世代）のようなデバイス上で実行されている場合は、このことが重要になります。こうしたデバイスでは、適切なアクセサリハードウェアが接続されている場合にのみ、オーディオ入力が得られます。リスト 2-10は、このテストを実行する方法を示します。

リスト 2-10　モバイル機器で録音がサポートされているかどうかの判定方法

```
UInt32 audioInputIsAvailable;
UInt32 propertySize = sizeof (audioInputIsAvailable);

AudioSessionGetProperty (
    kAudioSessionProperty_AudioInputAvailable,
    &propertySize,
    &audioInputIsAvailable // 出力が0以外の値ならば
                           // オーディオ入力が使用できることを意味する
);
```

オーディオセッションの使用

アプリケーションで一度に使用できるオーディオセッションカテゴリは1つだけです。そのため、どの時点においても、すべてのオーディオがアクティブなカテゴリの特性に従うことになります（ただし、System Sound Services（警告とユーザインターフェイスのサウンドエフェクト用のAPI）

を使用して再生するサウンドは唯一の例外です。それらのサウンドに対しては、常に最低の優先順位のオーディオセッションカテゴリが使用されます）。『*Audio Session Programming Guide*』の「Audio Session Categories」には、すべてのカテゴリが記載されています。

アプリケーションにオーディオセッションのサポートを付加する場合にも、開発とテスト用にiPhone Simulator上でアプリケーションを実行することは可能です。ただし、iPhone Simulatorではオーディオセッションの動作はシミュレートできません。オーディオセッションコードの動作をテストするには、デバイス上で実行する必要があります。

注： Audio Session Servicesを無視しても、アプリケーションの実行が妨げられることはありませんが、アプリケーションの動作が期待どおりにならない場合があります。このセクションですでに説明したように、ほとんどの場合において、このインターフェイスを使用せずにオーディオを使用するiPhoneまたはiPod touchアプリケーションを製品として出荷しないでください。

AVAudioPlayerクラスを使用した再生

`AVAudioPlayer`クラスは、オーディオを再生するためのObjective-C簡易インターフェイスを提供します。アプリケーションでステレオの定位や正確な同期を必要とせず、ネットワークストリームからキャプチャしたオーディオを再生しない場合は、このクラスを再生に使用することを推奨します。

オーディオプレーヤーを使用すると、次を実行できます。

- 任意の所要時間のサウンドの再生
- ファイルまたはメモリバッファからのサウンドの再生
- サウンドのループ再生
- 複数のサウンドの同時再生
- 再生中の各サウンドの相対的な再生レベルの制御
- サウンドファイル内での特定の位置へのシーク（早送りや巻き戻しなどのアプリケーション機能をサポートします）
- オーディオレベル測定機能に使用できるデータの取得

`AVAudioPlayer`クラスを使用すると、iPhone OSで使用可能なすべてのオーディオフォーマットのサウンドを再生できます。このクラスのインターフェイスの詳細については、『*AVAudioPlayer Class Reference*』を参照してください。

OpenAL、I/O Audio Unit、およびAudio Queue Servicesとは異なり、`AVAudioPlayer`ではAudio Session Servicesを使用する必要はありません。オーディオプレーヤーは割り込み後に自分自身でアクティブ化しなおします。デフォルトのオーディオセッションカテゴリ（「オーディオセッション：Core Audioとの連携」（38 ページ）を参照）で指定された動作（たとえば、画面がロックされたときにオーディオを停止させるなど）を必要とする場合は、オーディオプレーヤーでデフォルトのオーディオセッションを問題なく使用できます。

オーディオプレーヤーを再生用に設定するには、サウンドファイルを割り当てて、再生の準備を行い、デリゲートオブジェクトを指定します。リスト 2-11 に示すコードの場合、通常は、アプリケーションのコントローラクラスの初期化メソッドに移動します。

リスト 2-11 AVAudioPlayerオブジェクトの設定

```
NSString *soundFilePath =
                [[NSBundle mainBundle] pathForResource: @"sound"
                                                ofType: @"wav"];

NSURL *fileURL = [[NSURL alloc] initFileURLWithPath: soundFilePath];

AVAudioPlayer *newPlayer =
                [[AVAudioPlayer alloc] initWithContentsOfURL: fileURL
                                                       error: nil];
[fileURL release];

self.player = newPlayer;
[newPlayer release];

[self.player prepareToPlay];
[self.player setDelegate: self];
```

サウンドの再生が完了したときに、デリゲートオブジェクト（コントローラオブジェクトを使用できます）を使用して割り込みを処理し、ユーザインターフェイスを更新します。`AVAudioPlayer`クラスのデリゲートメソッドについては、『*AVAudioPlayerDelegate Protocol Reference*』を参照してください。リスト 2-12は、1つのデリゲートメソッドを簡単に実装したものです。このコードは、サウンドの再生が完了したときに、「再生／一時停止(Play/Pause)」トグルボタンのタイトルを更新します。

リスト 2-12 AVAudioPlayerのデリゲートメソッドの実装

```
- (void) audioPlayerDidFinishPlaying: (AVAudioPlayer *) player
                        successfully: (BOOL) flag {
    if (flag == YES) {
        [self.button setTitle: @"Play" forState: UIControlStateNormal];
    }
}
```

`AVAudioPlayer`オブジェクトを再生、一時停止、または停止するには、そのいずれかの再生制御メソッドを呼び出します。再生が実行中かどうかを調べるには、`playing`プロパティを使用します。リスト 2-13は、再生を制御し、`UIButton`オブジェクトのタイトルを更新する、基本的な再生／一時停止切り替えメソッドを示しています。

リスト 2-13 AVAudioPlayerオブジェクトの制御

```
- (IBAction) playOrPause: (id) sender {

    // すでに再生中の場合は一時停止する
    if (self.player.playing) {
        [self.button setTitle: @"Play" forState: UIControlStateHighlighted];
        [self.button setTitle: @"Play" forState: UIControlStateNormal];
        [self.player pause];

    // 停止または一時停止状態の場合は、再生を開始する
    } else {
        [self.button setTitle: @"Pause" forState: UIControlStateHighlighted];
        [self.button setTitle: @"Pause" forState: UIControlStateNormal];
        [self.player play];
    }
}
```

AVAudioPlayerクラスでは、サウンドに関する情報（サウンドのタイムライン内の再生ポイントなど）を管理し、再生オプション（音量やループ再生など）にアクセスするために、Objective-Cで宣言されたプロパティ機能を使用します。たとえば、オーディオプレーヤーの再生音量を設定するには次のようにします。

```
[self.player setVolume: 1.0];     // 使用可能な範囲は0.0～1.0
```

AVAudioPlayerクラスの詳細については、『*AVAudioPlayer Class Reference*』を参照してください。

Audio Queue Servicesを使用した録音と再生

Audio Queue Servicesは、iPhone OSおよびMac OS Xでオーディオの録音と再生を行うための、単純明快でオーバーヘッドの少ない手段を提供します。Audio Queue Servicesにより、ハードウェアインターフェイスに関する基本知識がなくても、アプリケーションで録音および再生用のハードウェアデバイス（マイクやスピーカーなど）を使用できるようになります。また、コーデックのしくみがわからなくても洗練されたコーデックを使用できるようになります。

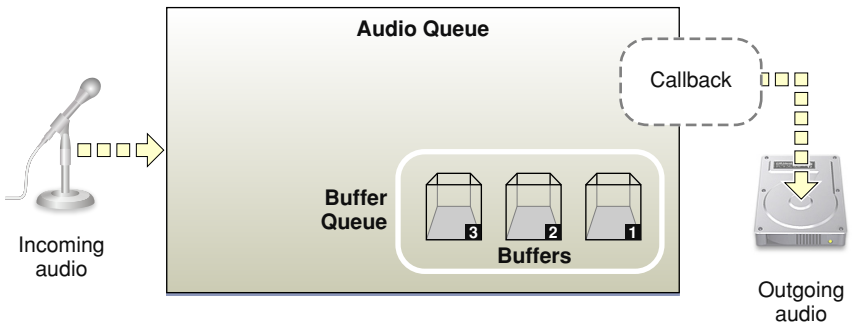
Audio Queue Servicesは高レベルのインターフェイスですが、いくつかの高度な機能をサポートしています。タイミングをきめ細かく制御でき、スケジューリングされた再生と同期をサポートします。また、Audio Queue Servicesを使用して、複数のオーディオキューの再生の同期、サウンドの同時再生、複数のサウンドの再生レベルの独立制御、およびループ再生を実行できます。圧縮されたオーディオをiPhoneまたはiPod touchで再生するには、Audio Queue ServicesとAVAudioPlayerクラス（「AVAudioPlayer クラスを使用した再生」 （41 ページ））を使用する方法のみとなります。

通常、Audio Queue ServicesはAudio File Services（「サウンドファイル」 （34 ページ）を参照）またはAudio File Stream Services（「サウンドストリーム」 （37 ページ）を参照）と組み合わせて使用します。

録音と再生のためのオーディオキューコールバック関数

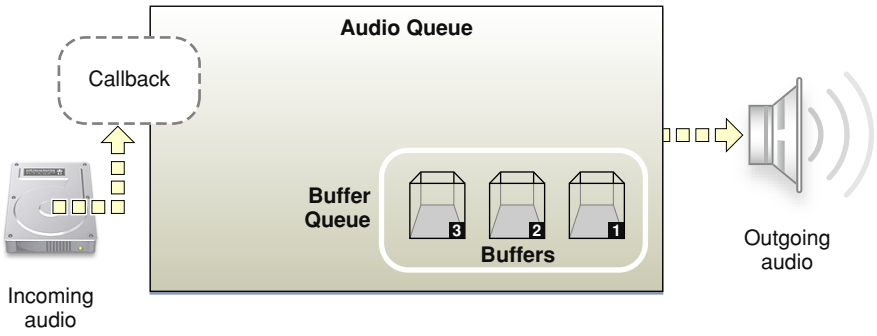
Audio File Stream Servicesの場合と同様に、オーディオキューオブジェクトとのやり取りにはコールバックとプロパティを使用します。録音では、オーディオデータバッファ（録音オーディオキューオブジェクトが提供します）を受け取り、それらをディスクに書き込むコールバック関数を実装します。録音すべき入力データの新しいバッファが発生すると、オーディオキューオブジェクトからコールバックが呼び出されます。図 2-2は、この処理の簡単な概要を図示しています。

図 2-2 オーディオキューオブジェクトを使用した録音



再生では、オーディオコールバックが逆の役割を果たします。再生オーディオキューオブジェクトで、再生すべきオーディオの別のバッファが必要になると、コールバックが呼び出されます。すると、コールバックによって、指定された数のオーディオパケットがディスクから読み取られ、それらがオーディオキューオブジェクトのいずれかのバッファに渡されます。そして、オーディオキューオブジェクトによってそのバッファが再生されます。図 2-3は、この処理を図示しています。

図 2-3 オーディオキューオブジェクトを使用した再生



オーディオキューオブジェクトの作成

Audio Queue Servicesを使用するには、まず、オーディオキューオブジェクトを作成します。これには2つの種類がありますが、どちらも`AudioQueueRef`のデータ型を持ちます。

- 録音オーディオキューオブジェクトを作成するには、`AudioQueueNewInput`関数を使用します。
- 再生オーディオキューオブジェクトを作成するには、`AudioQueueNewOutput`関数を使用します。

再生のためのオーディオキューオブジェクトを作成するには、次の手順を実行します。

1. 再生するデータのオーディオフォーマットなど、オーディオキューで必要な情報を管理するデータ構造を作成します。
2. オーディオキューバッファを管理するコールバック関数を定義します。このコールバック関数は、Audio File Servicesを使用して再生するファイルを読み込みます。
3. `AudioQueueNewOutput`関数を使用して、再生オーディオキューをインスタンス化します。

リスト 2-14に、これらの手順を示します。

リスト 2-14 オーディオキューオブジェクトの作成

```
static const int kNumberBuffers = 3;
// オーディオキューで必要な情報を管理するためのデータ構造を作成する
struct myAQStruct {
    AudioFileID                   mAudioFile;
    CAStreamBasicDescription      mDataFormat;
    AudioQueueRef                 mQueue;
    AudioQueueBufferRef           mBuffers[kNumberBuffers];
    SInt64                        mCurrentPacket;
    UInt32                        mNumPacketsToRead;
```

```
    AudioStreamPacketDescription    *mPacketDescs;
    bool                            mDone;
};
// 再生オーディオキューのコールバック関数を定義する
static void AQTestBufferCallback(
    void                    *inUserData,
    AudioQueueRef           inAQ,
    AudioQueueBufferRef     inCompleteAQBuffer
) {
    myAQStruct *myInfo = (myAQStruct *)inUserData;
    if (myInfo->mDone) return;
    UInt32 numBytes;
    UInt32 nPackets = myInfo->mNumPacketsToRead;

    AudioFileReadPackets (
        myInfo->mAudioFile,
        false,
        &numBytes,
        myInfo->mPacketDescs,
        myInfo->mCurrentPacket,
        &nPackets,
        inCompleteAQBuffer->mAudioData
    );
    if (nPackets > 0) {
        inCompleteAQBuffer->mAudioDataByteSize = numBytes;
        AudioQueueEnqueueBuffer (
            inAQ,
            inCompleteAQBuffer,
            (myInfo->mPacketDescs ? nPackets :0),
            myInfo->mPacketDescs
        );
        myInfo->mCurrentPacket += nPackets;
    } else {
        AudioQueueStop (
            myInfo->mQueue,
            false
        );
        myInfo->mDone = true;
    }
}
// オーディオキューオブジェクトをインスタンス化する
AudioQueueNewOutput (
    &myInfo.mDataFormat,
    AQTestBufferCallback,
    &myInfo,
    CFRunLoopGetCurrent(),
    kCFRunLoopCommonModes,
    0,
    &myInfo.mQueue
);
```

オーディオキューの再生レベルの制御

オーディオキューオブジェクトには、再生レベルを制御するための方法が2つあります。

再生レベルを直接設定するには、AudioQueueSetParameter関数にkAudioQueueParam_Volumeパラメータを渡します。その例をリスト 2-15に示します。レベルの変更はすぐに有効になります。

リスト 2-15　再生レベルの直接設定

```
Float32 volume = 1;
AudioQueueSetParameter (
    myAQstruct.audioQueueObject,
    kAudioQueueParam_Volume,
    volume
);
```

AudioQueueEnqueueBufferWithParameters関数を使用して、オーディオキューバッファの再生レベルを設定することもできます。これを利用すると、実際にオーディオキューバッファが保持するオーディオキューの設定を、エンキュー時に割り当てることができます。このような変更は、オーディオキューバッファが再生を開始するときに有効になります。

どちらの場合も、オーディオキューのレベル変更は、再びそれを変更するまで有効です。

オーディオキューの再生レベルの指示

現在の再生レベルは、オーディオキューオブジェクトのkAudioQueueProperty_CurrentLevelMeterDBプロパティを問い合わせて取得できます。このプロパティの値はAudioQueueLevelMeterState構造体の配列であり、チャネルごとに1つあります。リスト 2-16に、この構造体を示します。

リスト 2-16　AudioQueueLevelMeterState構造体

```
typedef struct AudioQueueLevelMeterState {
    Float32     mAveragePower;
    Float32     mPeakPower;
};  AudioQueueLevelMeterState;
```

複数のサウンドの同時再生

複数のサウンドを同時に再生するには、サウンドごとに再生オーディオキューオブジェクトを1つずつ作成します。各オーディオキューに対して、AudioQueueEnqueueBufferWithParameters関数を使用して、オーディオの最初のバッファが同時に開始するように指定します。

iPhoneまたはiPod touchでサウンドを同時に再生する場合、オーディオ形式が非常に重要になります。これは、iPhone OSでは、特定の圧縮されたフォーマットの再生において、効率的なハードウェアコーデックが利用されるからです。デバイスで一度に再生できるのは、次のいずれかのフォーマットの単一インスタンスのみになります。

- AAC
- ALAC (Apple Lossless)
- MP3

高品質のサウンドを同時に再生するには、リニアPCMまたはIMA4のオーディオを使用してください。

以下のリストは、iPhone OSが、個別再生または複数再生のためにサポートしているオーディオフォーマットについての説明です。

- **リニアPCMおよびIMA/ADPCM (IMA4)のオーディオ**。リニアPCMフォーマットおよびIMA4フォーマットのサウンドは、iPhone OSでCPUリソースを気にせずに複数同時に再生できます。
- **AAC、MP3、およびApple Lossless (ALAC)のオーディオ**。AAC、MP3、およびApple Lossless (ALAC)のサウンドの再生には、iPhoneおよびiPod touch上のハードウェアベースの効率的なデコード機能が使用されます。ただし、このようなサウンドは一度に1つしか再生できません。

Audio Queue Servicesを使用した録音と再生の方法、およびコード例については、『*Audio Queue Services Programming Guide*』を参照してください。

OpenALを使用した定位操作を伴う再生

オープンソース化されたOpenALオーディオAPIは、OpenALフレームワークで使用でき、Core Audioの上層に構築されていますが、このAPIは再生中のサウンドの定位操作に対応するように最適化されています。OpenALでは、OpenGLのモデルを継承したインターフェイスを使用することで、サウンドの再生、定位操作、ミックス、および移動が簡単に行えます。OpenALとOpenGLは共通の座標系を共有しているので、サウンドの動きと画面上のグラフィカルオブジェクトとを同期させることができます。OpenALはCore AudioのIOオーディオユニット（「オーディオユニット」（49 ページ）を参照）を直接使用するため、再生時の遅延が最小になります。

これらすべての理由から、iPhoneおよびiPod touch上のゲームアプリケーションでサウンドエフェクトを再生する場合は、OpenALが最適な選択といえます。また、OpenALは一般的なiPhone OSアプリケーションの再生ニーズにも適しています。

iPhone OSのOpenAL 1.1では、オーディオのキャプチャはサポートされていません。iPhone OSでの録音には、Audio Queue Servicesを使用してください。

Mac OS XのOpenALでは、OpenAL 1.1の仕様だけでなく、その拡張も実装されています。iPhone OSのOpenALには、Appleによる2つの拡張があります。

- `alBufferDataStaticProcPtr`。`alBufferData`の使用パターンに従ったものですが、バッファデータのコピーが削除されています。
- `alcMacOSXMixerOutputRateProcPtr`。基になるミキサーのサンプルレートを制御できるようにします。

Mac OS XにおけるCore AudioでのOpenALの使用例については、Core Audio SDKの`Services/OpenALExample`を参照してください。OpenALのドキュメントについては、OpenALのWebサイト(http://openal.org)を参照してください。

システムサウンド：警告とサウンドエフェクト

レベル設定、定位操作、オーディオセッションその他の制御を必要としない、短いサウンドファイルを再生するには、Audio ToolboxフレームワークのAudioServices.hヘッダファイルで宣言されているSystem Sound Servicesを使用します。このインターフェイスは、可能な限り最も簡単な方法で、単に短いサウンドを再生したいだけの場合に最適です。iPhone OSでは、System Sound Servicesを使用して再生されるサウンドの最大持続時間が30秒に制限されています。

iPhone OSでは、AudioServicesPlaySystemSound関数を呼び出すことで、指定したサウンドエフェクトファイルをすぐに再生できます。あるいは、ユーザに警告を示す必要がある場合に、AudioServicesPlayAlertSoundを呼び出すことができます。これらの関数はそれぞれ、iPhoneでユーザがイレントスイッチを設定している場合に、バイブレーションを起動します（もちろん、iPhone Simulatorを使用している場合やデスクトップの場合には、バイブレーションやブザー音は起こりません）。

AudioServicesPlaySystemSound関数を呼び出すときにkSystemSoundID_Vibrate定数を使用すると、iPhoneでバイブレーションを明示的に起動できます。

AudioServicesPlaySystemSound関数を使用してサウンドを再生するには、まず、サウンドIDオブジェクトを作成して、サウンドエフェクトファイルをシステムサウンドとして登録します。その後で、そのサウンドを再生できます。たびたびまたは繰り返し再生するなどの典型的な使いかたでは、アプリケーションが終了するまでサウンドIDオブジェクトを保持します。起動時のサウンドなど、一度しかサウンドを使用しないことがわかっている場合は、サウンドを再生したらすぐにサウンドIDオブジェクトを破棄してメモリを解放します。

リスト 2-17に、System Sound Servicesのインターフェイスを使用してサウンドを再生するための最小限のプログラムを示します。サウンド完了コールバックと、それをインストールする呼び出しは、このサウンドを再び再生することがない場合に、サウンドの再生後にメモリを解放したいときに主に役立ちます。

リスト 2-17　短いサウンドの再生

```
#include <AudioToolbox/AudioToolbox.h>
#include <CoreFoundation/CoreFoundation.h>

// サウンドの再生が終了したときに呼び出されるコールバックを定義する。
// 再生後にメモリを解放する必要がある場合に役立つ
static void MyCompletionCallback (
    SystemSoundID  mySSID,
    void * myURLRef
) {
        AudioServicesDisposeSystemSoundID (mySSID);
        CFRelease (myURLRef);
        CFRunLoopStop (CFRunLoopGetCurrent());
}

int main (int argc, const char * argv[]) {
    // サウンドの再生に必要な要素を用意する
    SystemSoundID    mySSID;
    CFURLRef        myURLRef;
    myURLRef = CFURLCreateWithFileSystemPath (
        kCFAllocatorDefault,
        CFSTR ("../../ComedyHorns.aif"),
        kCFURLPOSIXPathStyle,
        FALSE
    );

    // このサウンドファイルを表すシステムサウンドIDを作成する
    OSStatus error = AudioServicesCreateSystemSoundID (myURLRef, &mySSID);

    // サウンド完了コールバックを登録する。
    // これも、再生後にメモリを解放する必要がある場合に役立つ
    AudioServicesAddSystemSoundCompletion (
        mySSID,
        NULL,
```

```
        NULL,
        MyCompletionCallback,
        (void *) myURLRef
    );

    // サウンドファイルを再生する
    AudioServicesPlaySystemSound (mySSID);

    // サウンドを再生するのに十分な時間、アプリケーションの実行を継続するために
    // 現在のスレッド上で実行ループを起動する。
    // 後で、サウンド完了コールバックによって、この実行ループは停止する
    CFRunLoopRun ();
    return 0;
}
```

Core Audioプラグイン：オーディオユニットとコーデック

Core Audioには、オーディオデータを処理するためのプラグインのメカニズムがあります。iPhone OSでは、システムがこれらのプラグインを提供しています。Mac OS Xでは、組み込みのプラグインがあるほか、独自のプラグインを作成することもできます。

複数のホストアプリケーションがそれぞれ、オーディオユニットまたはコーデックの複数のインスタンスを使用できます。

オーディオユニット

iPhone OSでは、オーディオユニットは、アプリケーションで低レイテンシの入出力を実現するためのメカニズムを提供します。また、ここで説明するように、特定のDSP機能も提供します。

Mac OS Xでは、オーディオユニットは、GarageBandやLogic Studioなどのオーディオアプリケーションへのアドオンとして特に際立った存在になっています。この役割を担うオーディオユニットには、**ビュー**と呼ばれる独自のユーザインターフェイスがあります。iPhone OSのオーディオユニットにはビューはありません。

どちらのプラットフォームでも、組み込みのオーディオユニットにはプログラミング上の名前が付いており、それらはAudio Unitフレームワークの`AUComponent.h`ヘッダファイルにあります。

iPhone OSのオーディオユニットでは、8.24ビットの固定小数点リニアPCMオーディオデータが入出力に使用されます。ただし、次のリストで説明するように、データフォーマットコンバータユニットはこの唯一の例外です。iPhone OSのオーディオユニットには次のものがあります。

- **3Dミキサーユニット**—任意の数のモノラル入力を使用できます。各入力には8ビットまたは16ビットのリニアPCMを使用できます。8.24ビットの固定小数点PCMによるステレオ出力を1つ提供します。3Dミキサーユニットの入力ではサンプルレート変換が実行され、各入力チャネルを詳細に制御できます。たとえば、音量、ミュート、パン、距離減衰などを制御でき、それらの変更のレートも制御できます。プログラミング上は、このユニットは`kAudioUnitSubType_AU3DMixerEmbedded`です。

- **マルチチャネルミキサーユニット**—任意数のモノラル入力またはステレオ入力を使用できます。各入力には16ビットリニアまたは8.24ビットの固定小数点PCMを使用できます。8.24ビットの固定小数点PCMによるステレオ出力を1つ提供します。アプリケーションでは、各入力チャネルのミュートおよびミュート解除を実行できるほか、音量も制御できます。プログラミング上は、このユニットは`kAudioUnitSubType_MultiChannelMixer`です。
- **コンバータユニット**—サンプルレート、ビット深度、およびビット形式（リニアから固定小数点へ）の変換を提供します。iPhoneのコンバータユニットの正準形のデータフォーマットは、8.24ビットの固定小数点PCMです。このフォーマットへの変換、およびこのフォーマットからの変換を行います。プログラミング上は、このユニットは`kAudioUnitSubType_AUConverter`です。
- **I/Oユニット**—リアルタイムのオーディオ入出力を提供し、必要に応じてサンプルレート変換を実行します。プログラミング上は、このユニットは`kAudioUnitSubType_RemoteIO`です。
- **iPod EQユニット**—アプリケーションで使用できる簡易イコライザを提供します。組み込みのiPhone iPodアプリケーションで使用できるものと同じプリセットがあります。iPod EQユニットでは、8.24ビットの固定小数点PCMによる入出力が使用されます。プログラミング上は、このユニットは`kAudioUnitSubType_AUiPodEQ`です。

*I/O*ユニットはオーディオユニットの特殊なカテゴリであり、そうしたオーディオユニットには重要な役割があるので、「オーディオ処理グラフ」（52 ページ）で詳しく説明しています。

Mac OS Xでは40あまりのオーディオユニットが提供されています。それらすべての一覧については、「Mac OS Xのシステム付属オーディオユニット」（83 ページ）を参照してください。Mac OS Xでは、独自のオーディオユニットを作成して、自分のアプリケーションの部品として組み込んだり、自分が権利を有する製品として顧客に提供したりできます。詳細については、『*Audio Unit Programming Guide*』を参照してください。

Mac OS Xのオーディオユニットでは、非インターリーブの32ビット浮動小数点リニアPCMデータが入出力に使用されます。ただし、オーディオユニットとしてのデータフォーマットコンバータの場合は例外で、このフォーマットへの変換、およびこのフォーマットからの変換を行います。また、Mac OS Xのオーディオユニットの中には、リニアPCMのその他のバリエーションをサポートしているものがありますが、リニアPCM以外のオーディオユニットはサポートしていません。これにより、Appleが提供するオーディオユニットとほかのベンダが提供するオーディオユニットとの互換性が保証されます。異なるフォーマットのオーディオデータをMac OS Xのリニア PCMに変換するには、オーディオコンバータを使用できます（「データフォーマット変換」（34 ページ）を参照）。

iPhone OSおよびMac OS Xでは、ホストアプリケーションはAudio Unit Servicesの関数を使用して、オーディオユニットを検出し、読み込みます。各オーディオユニットは、タイプ、サブタイプ、製造元コードの、3つの要素の組み合わせによって一意に識別されます。タイプコードはAppleにより指定され、ユニットの一般的な目的（エフェクトやジェネレータなど）を示します。サブタイプはオーディオユニットの動作をより正確に示しますが、プログラミング上はオーディオユニットにとって重要ではありません。ただし、製造元でエフェクトユニットを複数提供する場合は、各ユニットに別々のサブタイプを付けてほかと区別する必要があります。製造元コードはオーディオユニットのデベロッパを識別するものです。Appleではデベロッパに対し、Data Type Registrationページで製造元コードを「クリエータコード」として登録するよう求めています。

「プロパティ、スコープ、要素」（26 ページ）で説明しているように、オーディオユニットでは機能や設定情報を表すのにプロパティが使用されています。各オーディオユニットタイプには、Appleにより定義されたいくつかの必須プロパティがありますが、オーディオユニットのニーズに応じてさらにプロパティを追加しても構いません。Mac OS Xでは、ホストアプリケーションでプロパティ情報を使用してオーディオユニットの汎用のビューを作成でき、開発したオーディオユニットの必須部分としてカスタムのビューを提供できます。

オーディオユニットではまた、しばしばエンドユーザがリアルタイムに調整することが可能な設定に対して、**パラメータ**のメカニズムが使用されています。これらのパラメータは、関数への引数ではなく、ユーザによるオーディオユニットの動作の調整をサポートするメカニズムです。たとえば、Mac OS X用のパラメトリックフィルタユニットの中には、中央周波数とフィルタ応答幅を決めるためのパラメータを持つものがあり、それらのパラメータはビューを通じて設定できます。

Mac OS Xでオーディオユニットの状態の変更を監視する場合、アプリケーションでコールバック関数（*リスナー*とも呼ばれます）を登録できます。リスナーは、特定のオーディオユニットイベントが発生したときに呼び出されます。たとえば、Mac OS Xアプリケーションで、ユーザがオーディオユニットのビューにあるスライダを動かしたことや、オーディオデータフローが中断されたことを検知したい場合があります。詳細については、「Technical Note TN2104: Handling Audio Unit Events」を参照してください。

Core Audio SDKの`AudioUnits`フォルダでは、共通のオーディオユニットタイプ（エフェクトユニットやインスツルメントユニットなど）に対応したテンプレートが提供されており、ほとんどのComponent Managerプラグインインターフェイスを自動的に実装するC++フレームワークも一緒に提供されています。SDKを使用したMac OS X用オーディオユニットの開発の詳細については、『*Audio Unit Programming Guide*』を参照してください。

コーデック

iPhone OSで使用できる録音および再生コーデックは、オーディオ品質、アプリケーション開発における柔軟性、ハードウェア機能、そしてバッテリー持続時間について、バランスが保たれるように選ばれています。

iPhone OSの再生コーデックには2つのグループがあります。1番目のグループは表 2-3に列挙しています。このグループには、制限なく使用できる効率の高いフォーマットが含まれています。つまり、これらの各フォーマットは、複数のインスタンスを同時に再生できます。

iPhone OSでサポートされているオーディオファイルフォーマットの詳細については、「iPhoneのオーディオファイルフォーマット」（36 ページ）を参照してください。

表 2-3　iPhone OS：制限のない再生オーディオフォーマット

AMR（Adaptive Multi-Rate、スピーチ用コーデック）
iLBC（internet Low Bitrate Codec、スピーチ用コーデック）
IMA/ADPCM（IMA-4とも呼ばれます）
リニアPCM
μLawおよびaLaw

iPhone OS再生コーデックの2番目のグループでは、すべてのフォーマットで単一のハードウェアパスが共有されます。したがって、一度に再生できるのは、いずれかのフォーマットの単一のインスタンスのみとなります。

表 2-4　iPhone OS：制限のある再生オーディオフォーマット

AAC

Apple Lossless
MP3

iPhone OSには、表2-5に列挙した録音コーデックが含まれています。ご覧のように、MP3またはAACによる録音はできません。これらのフォーマットではCPUのオーバーヘッドが高く、その結果バッテリーの消耗が激しくなるからです。

表 2-5　　iPhone OS：録音オーディオフォーマット

Apple Lossless
iLBC（internet Low Bitrate Codec、スピーチ用コーデック）
IMA/ADPCM（IMA-4とも呼ばれます）
リニアPCM
μLawおよびaLaw

Mac OS Xでは、「Mac OS Xでサポートされているオーディオファイルフォーマットとオーディオデータフォーマット」（87 ページ）で説明している広範なコーデックやフォーマットがサポートされています。

また、Mac OS Xではオーディオデータコーデックの使用と作成のためのインターフェイスも提供されています。これらのインターフェイスは`AudioConverter.h`ヘッダファイル（Audio Toolboxフレームワーク内）および`AudioCodec.h`ヘッダファイル（Audio Unitフレームワーク内）で宣言されています。これらのサービスの使用例については、「Mac OS Xでの共通の作業」（59 ページ）を参照してください。

オーディオ処理グラフ

オーディオ処理グラフ（*AUGraph*とも呼ばれます）は、複雑な作業を実行するためにまとめて並べられたオーディオユニットの集まりを定義するものです。たとえば、グラフで信号を歪ませ、コンプレッサを掛けた後、パンを音場の特定の位置に設定することができます。グラフを定義すると、再利用可能な処理モジュールが得られます。このモジュールは、アプリケーションで信号の連なりに対して追加および削除することができます。

通常、処理グラフの終端はI/Oユニット（*出力ユニット*とも呼ばれます）になります。多くの場合、I/Oユニットはハードウェアとのインターフェイスとなりますが（間接的に低レベルのサービスを経由します）、必ずしもそうでなければならないわけではありません。代わりに、I/Oユニットから出力をアプリケーションに送り返すことができます。

I/Oユニットはまた、処理グラフで**ヘッドノード**とも呼ばれます。グラフ内でデータフローを開始および停止できるユニットは、I/Oユニットだけです。これは、オーディオユニットがデータの取得に使用する、いわゆる**プル**メカニズムの本質的な側面です。グラフ内の各オーディオユニットは、レンダリングコールバックを後続のユニットに登録します。これにより、後続のユニットはオーディオデータを要求できるようになります。アプリケーションによってI/Oユニットが起動され、I/Oユニットによってデータフローが開始されると、I/Oユニットのレンダリングメソッドから、連なりにおける先行のオーディオユニットに対して、データを要求するためのコールバックが実行され、その結果さらに先行のユニットが呼び出される、というようになります。

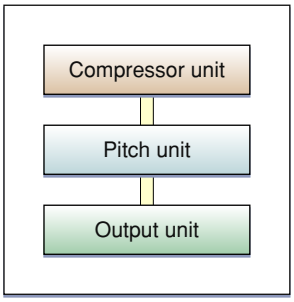
「オーディオユニット」（49 ページ）で説明しているように、iPhone OSには単一のI/Oユニットがあります。Mac OS Xでは、状況がもう少し複雑になります。

- **AUHALユニット**は、特定のハードウェアデバイスに対する専用の入出力として使用します。詳細については、*Device input using the HAL Output Audio Unit*を参照してください。
- **汎用I/Oユニット**は、オーディオ処理グラフの出力をアプリケーションに接続できるようにします。図 2-6（54 ページ）に示すように、汎用I/Oユニットをサブグラフのヘッドノードとして使用することもできます。
- **システムI/Oユニット**は、警告とユーザインターフェイスのサウンドエフェクトに使用します。
- **デフォルトI/Oユニット**は、それ以外のすべてのオーディオの入出力に使用します。

Mac OS Xでは、「ユーティリティ(Utilities)」フォルダにあるAudio MIDI設定アプリケーションを使用して、システムI/OユニットとデフォルトI/Oユニットの接続先を別々に設定できます。

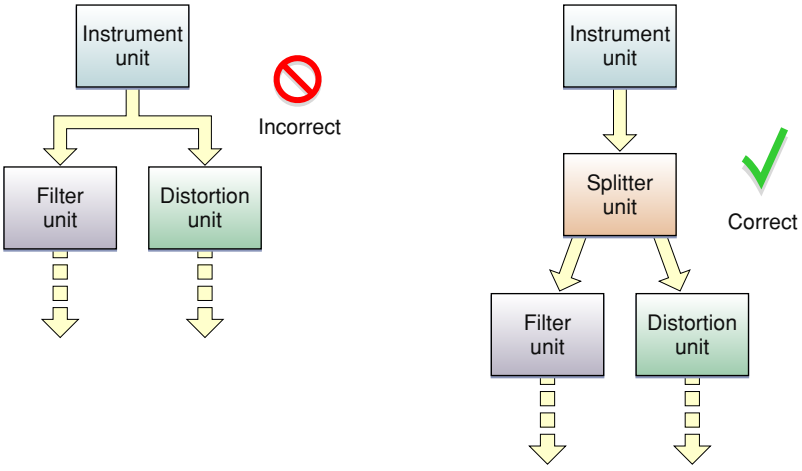
図 2-4は、シグナルフローが上から下へと向かう単純なオーディオ処理グラフを示しています。

図 2-4 Mac OS Xでの単純なオーディオ処理グラフ



オーディオ処理グラフ内の各オーディオユニットは、**ノード**と呼ばれることがあります。あるノードの出力を別のノードの入力につなげることによって、処理グラフの接続を作成します。図 2-5に示すように、単一の出力をそのまま複数の入力に接続することはできません。分割ユニットを間にはさむ必要があります。

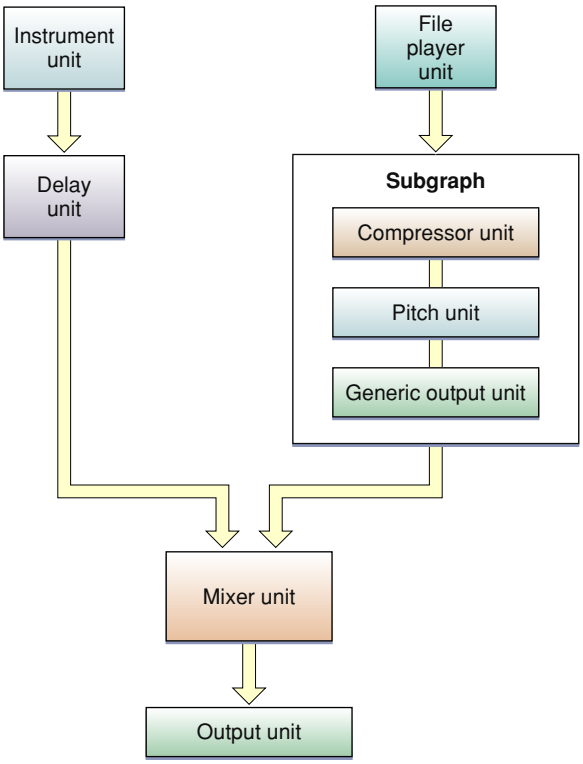
図 2-5 Mac OS Xでオーディオユニットの接続を分岐させる方法



ただし、オーディオユニットのタイプによっては、複数の出力または入力を提供するように設計されたものもあります。たとえば、分割ユニットがその一例です。

Audio Processing Graph Servicesを使用すると、サブグラフどうしを結合して、より大きいグラフにすることができます。その場合、サブグラフは大きいグラフ内で単一のノードとして表されます。この例を図 2-6に示します。

図 2-6 Mac OS Xでのサブグラフ接続



各グラフまたはサブグラフの終端は、I/Oユニットにする必要があります。サブグラフ、またはアプリケーションに出力を供給するグラフは、その終端を汎用I/Oユニットにしてください。汎用I/Oユニットはハードウェアには接続しません。

オーディオ処理グラフを使用せずに、プログラムを通じてオーディオユニットを連結することもできますが、ほとんどの場合、よい方法ではありません。グラフには重要な利点がいくつかあります。グラフには動的に変更を加えることができるので、データの処理中にシグナルパスを変更することが可能です。また、グラフによってオーディオユニットの相互接続がカプセル化されるので、グラフから参照している各オーディオユニットを明示的にインスタンス化するのではなく、グラフを1回の手順でインスタンス化します。

Mac OS XのMIDI Services

Mac OS XのCore Audioでは、MIDIのサポートにCore MIDI Servicesが使用されます。これらのサービスは、`CoreMIDI.framework`の次のヘッダファイルで宣言されている関数、データ型、および定数で構成されています。

- MIDIServices.h
- MIDISetup.h
- MIDIThruConnection.h
- MIDIDriver.h

Core MIDI Servicesでは、アプリケーションやオーディオユニットがMIDIデバイスとの通信に使用できるインターフェイスが定義されています。また、アプリケーションがMIDIネットワークとやり取りできるようにする、いくつかの抽象化が使用されています。

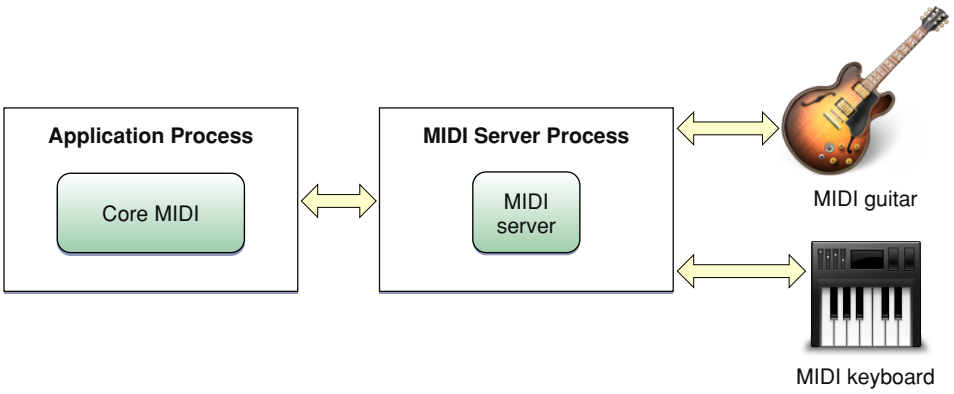
MIDIエンドポイント（不透過型のMIDIEndpointRefによって定義されています）は、標準の16チャネルMIDIデータストリームのソースまたはデスティネーションを表します。エンドポイントを、Music Player Servicesが使用するトラックに関連付けることによって、MIDIデータの録音や再生が可能になります。MIDIエンドポイントは、標準のMIDIケーブル接続を表すプロキシオブジェクトです。ただし、MIDIエンドポイントは必ずしも物理デバイスに対応している必要はなく、アプリケーションでは、MIDIデータを送受信する仮想のソースまたはデスティネーションとして、アプリケーション自身を設定できます。

多くの場合、MIDIドライバでは複数のエンドポイントが**MIDIエンティティ**(MIDIEntityRef)と呼ばれる論理的なグループに結合されます。たとえば、MIDI入力エンドポイントとMIDI出力エンドポイントをMIDIエンティティとしてグループ化することには意味があります。そうすることで、デバイスやアプリケーションとの双方向通信用として容易に参照できるようになります。

物理MIDIデバイス（単一のMIDI接続ではないもの）はそれぞれ、Core MIDIデバイスオブジェクト(MIDIDeviceRef)によって表されます。デバイスオブジェクトにはそれぞれ、1つまたは複数のMIDIエンティティが含まれる可能性があります。

Core MIDIはMIDI Serverと通信します。MIDI Serverは、アプリケーションとデバイスの間で実際にMIDIデータを受け渡す処理を行います。MIDI Serverは独自のプロセスで実行され、どのアプリケーションにも依存しません。図 2-7は、Core MIDIとMIDI Serverの間の関係を示しています。

図 2-7 Core MIDIとCore MIDI Server

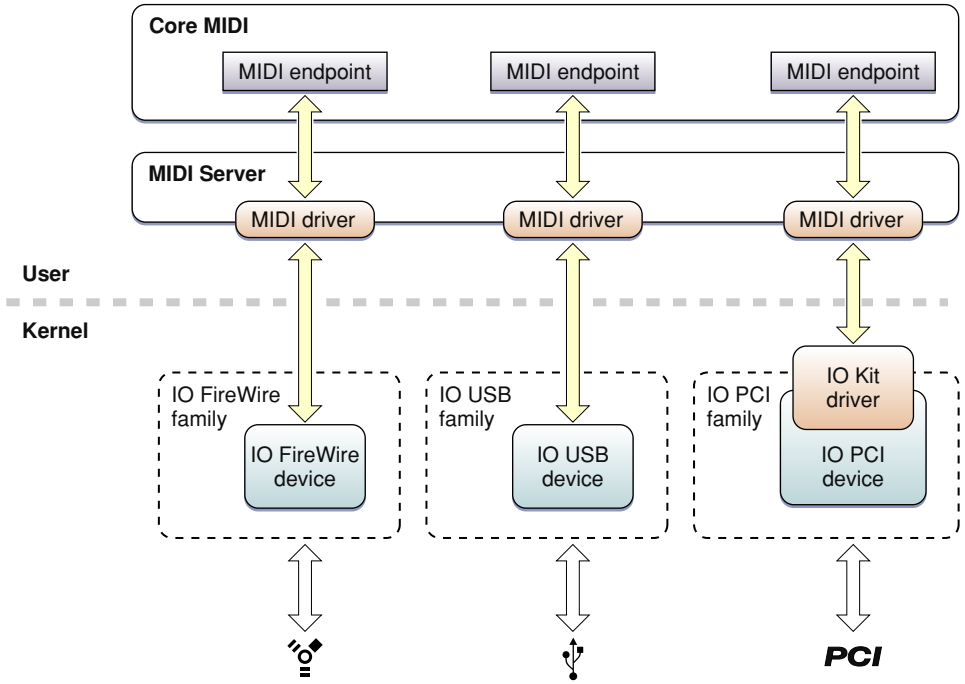


アプリケーションに依存しないMIDI通信の土台を提供することに加えて、MIDI Serverはすべてのスルー接続も処理します。これにより、ホストアプリケーションが関与しないデバイスどうしの連結が可能になります。

MIDIデバイスの製造元は、場合によっては、カーネルレベルのI/O Kitドライバとやり取りするために、MIDI Server用のCFPluginプラグインをCFBundleにパッケージ化して提供する必要があります。図2-8は、Core MIDIおよびCore MIDI Serverが基になるハードウェアとどのようにやり取りするのかを示しています。

注： USB MIDIクラスに準拠したデバイスを開発する場合、独自のドライバを作成する必要はありません。これは、Appleが提供するUSBドライバによってハードウェアがサポートされるからです。

図 2-8 MIDI ServerによるI/O Kitとのインターフェイス



各MIDIデバイス用のドライバは、一般にカーネルの外部に存在し、MIDI Serverプロセスの中で実行されます。これらのドライバは、基になるプロトコル（USBやFireWireなど）に対応したデフォルトのI/O Kitドライバとやり取りします。MIDIドライバは、未加工のデバイスデータを、使用可能なフォーマットでCore MIDIに提供する役割を果たします。次に、Core MIDIは指定されたMIDIエンドポイントを通じて、MIDI情報をアプリケーションに渡します。MIDIエンドポイントは、外部デバイス上のMIDIポートを抽象的に表現したものです。

ただし、PCIカード上のMIDIデバイスを、ユーザ空間ドライバを通じて完全に制御することはできません。PCIカードの場合、カーネル拡張を作成してカスタムのユーザクライアントを提供する必要があります。このクライアントでは、PCIデバイスそのものを制御するか（簡易メッセージキューをユーザ空間ドライバに提供します）、ユーザ空間ドライバからの要求があったときにPCIデバイスのアドレス範囲をMIDIサーバのアドレスにマップする必要があります。そうすることで、ユーザ空間ドライバがPCIデバイスを直接制御できるようになります。

ユーザ空間MIDIドライバの実装例については、Core Audio SDKの`MIDI/SampleUSBDriver`を参照してください。

Mac OS XのMusic Player Services

Music Player ServicesはMac OS Xで使用できるサービスで、MIDI楽曲トラックの集まりの整理や再生を可能にします。

トラックとは、MIDIデータやイベントデータのストリームであり、`MusicTrack`データ型によって表されます。トラックには一連の時間ベースのイベントが含まれています。これらのイベントは、MIDIデータ、Core Audioのイベントデータ、またはカスタムのイベントメッセージです。トラックは、1つの楽器の楽譜として捉えることができます。

トラックが集まったものが**シーケンス**で、`MusicSequence`データ型によって表されます。シーケンスには必ずテンポトラックが別に含まれていて、すべてのトラックの同期に使用されます。シーケンスは、複数の楽器の楽譜を集めた1つの譜面として捉えることができます。Mac OS Xアプリケーションでは、シーケンス内のトラックの追加、削除、または編集を動的に行うことができます。

シーケンスを再生するには、シーケンスを**ミュージックプレーヤー**オブジェクト（`MusicPlayer`型）に割り当てます。このオブジェクトはいわば指揮者として機能し、シーケンスの再生を制御します。サウンドを生成するには、各トラックをインスツルメントユニットまたは外部MIDIデバイスに送ります。

トラックは必ずしも音楽情報を表す必要はありません。代わりに、トラックを使用してオーディオユニットのオートメーションを実装することもできます。たとえば、パンナーユニットに割り当てられたトラックで、音場におけるサウンドソースの見かけの定位を制御できます。また、アプリケーション定義のコールバックを起動する専用のユーザイベントをトラックに含めることもできます。

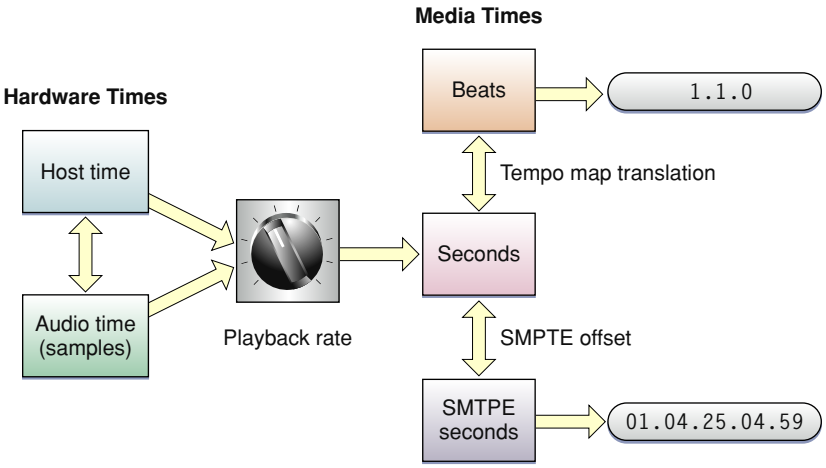
Music Player Servicesを使用したMIDIデータの再生の詳細については、「Mac OS XでのMIDIデータの処理」（66 ページ）を参照してください。

Mac OS XのTiming Services

Mac OS Xでは、Core Audio Clock Servicesは、アプリケーションやデバイスの同期に使用できる基準クロックを提供します。このクロックは、スタンドアロンのタイミングソースにしたり、MIDIビートクロックやMIDIタイムコードなどの外部のトリガと同期させたりできます。クロックそのものを開始および停止するか、特定のイベントに応答してアクティブ化または非アクティブ化するようにクロックを設定することができます。

生成されるクロック時間は、秒数、拍数、SMPTE時間、オーディオサンプル時間、1小節あたりの1拍子の時間など、いくつかの形式で取得できます。後のものほど、小節、拍子、および副拍子の観点から見て画面に表示しやすい方法で時間を表します。Core Audio Clock Servicesには、ある時間形式を別の時間形式に変換したり、1小節あたりの1拍子の時間やSMPTE時間を表示したりする、ユーティリティ関数も含まれています。図 2-9は、Core Audioの各種のクロック形式間の相互関係を示しています。

図 2-9 Core Audioのクロック形式の例



ハードウェア時間は、ホスト時間（システムクロック）からの絶対時間値か、外部オーディオデバイスから取得されるオーディオ時間（HAL内の`AudioDevice`オブジェクトによって表されます）のどちらかを表します。現在のホスト時間を調べるには、`mach_absolute_time`または`UpTime`を呼び出します。オーディオ時間は、サンプル数によって表される、オーディオデバイスの現在の時間です。サンプル数の変動率は、オーディオデバイスのサンプリングレートに依存します。

メディア時間は、オーディオデータの共通のタイミング方法を表します。正準形の表現は秒数であり、倍精度の浮動小数点値として表されます。ただし、テンポマップを使用して、秒数を音楽上の1小節あたりの1拍子の時間に変換したり、あるいはSMPTEオフセットを適用して、秒数をSMPTE秒数に変換したりできます。

メディア時間は必ずしも実際の時間に対応している必要はありません。たとえば、長さが10秒のオーディオファイルでも、再生速度を倍にした場合は再生に5秒しかかかりません。「iPhone OSでのみ使用できるサービス」（80 ページ）で示しているノブは、絶対（「現実の」）時間とメディアベースの時間との相互関係を調整できることを示しています。たとえば、小節と拍子による表記法は、楽曲全体のリズムや、どの音符をいつ再生するのかを示していますが、再生にかかる時間は示していません。再生にかかる時間を調べるには、再生速度（たとえば、1秒あたりの拍数など）を知る必要があります。同様に、SMPTE時間と実際の時間との対応は、フレームレートや、フレームがドロップされるかどうかなどの要因によって左右されます。

Mac OS Xでの共通の作業

この章では、Core Audioのパーツを組み合わせてMac OS Xでいくつかの共通の作業を実行する方法について説明します。

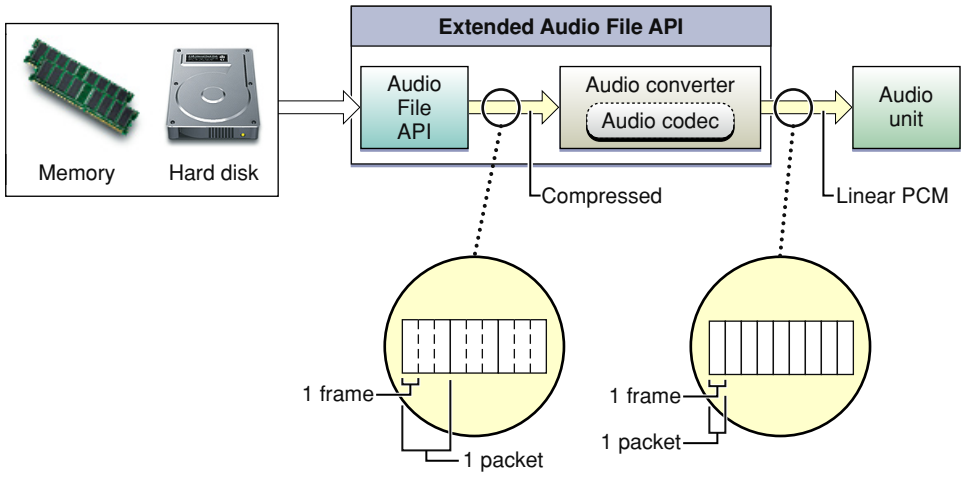
Core Audioは徹底したモジュール形式であり、その各種のパーツの使いかたについての制限はほとんどありません。つまり、あることを実行するのに複数の方法で実行できる場合が多くあります。たとえば、Mac OS Xアプリケーションでは、Audio File Servicesを呼び出して、圧縮されたサウンドファイルをディスクから読み取り、Audio Converter Servicesを呼び出してデータをリニアPCMに変換し、Audio Queue Servicesを呼び出して再生することができます。あるいは、Mac OS Xが提供するFile Playerオーディオユニットをアプリケーションに読み込むこともできます。どちらの方法にも、それぞれに利点と長所があります。

Mac OS Xでのオーディオデータの読み書き

オーディオを処理するアプリケーションの多くは、ディスクかバッファのどちらかに対してデータを読み書きする必要があります。通常は、ファイルに格納されているデータを読み取ってリニアPCMに変換する必要があります。この処理は、Extended Audio File Servicesを使用して1つの手順で実行できます。

図 3-1に示すように、Extended Audio File ServicesはAudio File Servicesを使用してオーディオデータを読み取った後、Audio Converter Servicesを呼び出してデータをリニアPCMに変換します（データがまだそのフォーマットになっていない場合）。

図 3-1　オーディオデータの読み取り



ファイルの読み取りと変換の手順を、より細かく制御する必要がある場合は、Audio FileまたはAudio Converterの関数を直接呼び出すことができます。Audio File Servicesは、ディスクまたはバッファからファイルを読み取る場合に使用します。このデータは圧縮されたフォーマットになっている可能

性があります。その場合は、オーディオコンバータを使用してリニアPCMに変換できます。また、オーディオコンバータを使用して、リニアPCM形式内部のビット深度やサンプリングレートなどの変更を処理することもできます。変換を処理するには、Audio Converter Servicesを使用してオーディオコンバータオブジェクトを作成し、必要な入力および出力フォーマットを指定します。各フォーマットはASBDで定義します（「Core Audioの汎用データ型」 （29 ページ）を参照）。

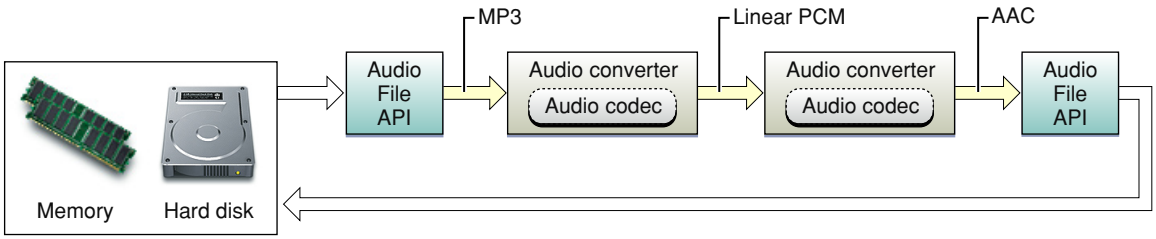
リニアPCMに変換されたデータは、オーディオユニットなどですぐに処理できます。オーディオユニットを使用するには、コールバックをオーディオユニットの入力に登録します。コールバックは、呼び出されたときにPCMデータのバッファを提供するように設計します。オーディオユニットで、より多くのデータが処理のために必要になると、オーディオユニットからコールバックが呼び出されます。

オーディオデータを出力する必要がある場合は、オーディオデータをI/Oユニットに送ります。I/Oユニットは、リニアPCMフォーマットのデータだけを受け取ることができます。このようなオーディオユニットは、通常はハードウェアデバイスのプロキシですが、必ずしもそうでなければならないわけではありません。

Mac OS Xでのオーディオデータフォーマットの変換

Core Audioでは、リニアPCMが中間フォーマットとして使用されます。そのため、数多くの変換の組み合わせが可能になります。特定のフォーマット変換が可能かを調べるには、デコーダ（あるフォーマットからリニアPCMへの変換）とエンコーダ（リニアPCMから別のフォーマットへの変換）の両方が使用できることを確かめる必要があります。たとえば、データをMP3からAACに変換する必要がある場合、2つのオーディオコンバータが必要になります。1つはMP3からリニアPCMに変換するコンバータで、もう1つはリニアPCMからAACに変換するコンバータです（図 3-2を参照）。

図 3-2 2つのコンバータを使用したオーディオデータの変換



Audio FileおよびAudio ConverterのAPIの使用例については、Core Audio SDKのサンプル`SimpleSDK/ConvertFile`および`Services/AudioFileTools`を参照してください。カスタムのオーディオコンバータコーデックの作成に興味がある場合は、`AudioCodec`フォルダにあるサンプルを参照してください。

Mac OS Xでのハードウェアとのインターフェイス

ほとんどのオーディオアプリケーションは外部のハードウェアに接続して、サウンドを出力するか（たとえば、アンプやスピーカーに出力します）、サウンドを取得します（たとえば、マイクを通じて取得します）。こうした接続は、Core Audioの一番下のレイヤによって、I/O Kitやドライバにア

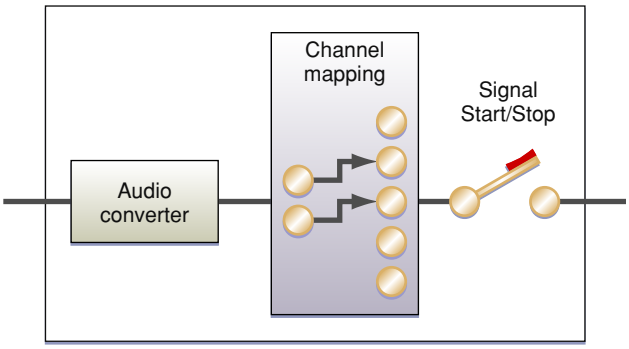
クセスすることで実行されます。Mac OS Xではこれらの接続へのインターフェイスが提供されており、*オーディオハードウェア抽象化層*（オーディオHAL）と呼ばれます。オーディオHALのインターフェイスはCore AudioフレームワークのAudioHardware.hヘッダファイルで宣言されています。

しかし、多くの場合、アプリケーションでオーディオHALを直接使用する必要はありません。Appleでは、ハードウェアに関するほとんどのニーズに対応する標準のオーディオユニットとして、デフォルト出力ユニット、システム出力ユニット、およびAUHALユニットの3つを提供しています。アプリケーションでこれらのオーディオユニットを使用するには、あらかじめ明示的に読み込んでおく必要があります。

デフォルトI/Oユニットとシステム I/Oユニット

デフォルトI/Oユニットとシステム I/Oユニットは、それぞれ、オーディオデータをデフォルトの出力（ユーザが選択する出力）またはシステム出力（警告やその他のシステムサウンドが再生される出力）に送ります。オーディオユニットの出力をこれらのいずれかのI/Oユニット（オーディオ処理グラフ内のものなど）に接続した場合は、出力でデータが必要になったときにオーディオユニットのレンダリングコールバック関数が呼び出されます。図 3-3に示すように、I/Oユニットは、該当する出力デバイスにHALを通じてデータを送り、次の作業を自動的に処理します。

図 3-3　I/Oユニットの内側



- リニアPCMデータの必要な変換。出力ユニットにはオーディオコンバータが組み込まれており、オーディオデータを、ハードウェアで必要となる各種のリニアPCMフォーマットに変換できます。
- 必要なチャネルマッピング。たとえば、ユニットから2チャネルのデータを供給していて、出力デバイスで5チャネルを処理できる場合、どちらのチャネルをどこにマップするのかを決める必要があります。これには、出力ユニットのkAudioOutputUnitProperty_ChannelMapプロパティを使用してチャネルマップを指定します。チャネルマップを指定しない場合は、デフォルトで、1番目のオーディオチャネルが1番目のデバイスチャネルにマップされ、2番目のオーディオチャネルが2番目のデバイスチャネルにマップされる、というようになります。実際に聞こえる出力は、ユーザがAudio MIDI設定アプリケーションでデバイススピーカーをどのように設定しているのかによって決まります。
- 信号の開始および停止。信号チェーンにおけるオーディオデータの流れを制御できるオーディオユニットは、出力ユニットだけです。

デフォルト出力ユニットを使用したオーディオの再生の例については、Core Audio SDKのSimpleSDK/DefaultOutputUnitを参照してください。

AUHALユニット

入力デバイスに接続する必要がある場合、またはデフォルトの出力デバイス以外のハードウェアデバイスに接続する必要がある場合は、AUHALを使用する必要があります。AUHALは出力デバイスとして指定されていますが、入力の`kAudioOutputUnitProperty_EnableIO`プロパティを設定することによって、入力を受け付けるように設定することもできます。詳細については、「Technical Note TN2091: Device Input Using the HAL Output Audio Unit」を参照してください。入力を受け付ける場合のAUHALは、入力のチャネルマッピングをサポートし、必要に応じてオーディオコンバータを使用して入力データをリニアPCMフォーマットに変換します。

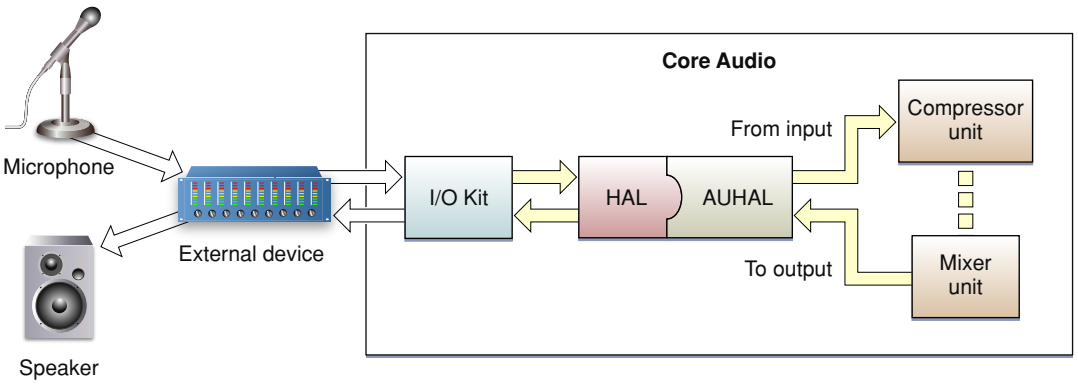
AUHALは、デフォルト出力ユニットをさらに汎用化したものです。オーディオコンバータとチャネルマッピングの機能に加えて、`kAudioOutputUnitProperty_CurrentDevice`プロパティをHAL内の`AudioDevice`オブジェクトのIDに設定することによって、接続先のデバイスを指定できます。接続後、AUHALにアクセスすることによって、`AudioDevice`オブジェクトに関連付けられたプロパティを操作することもできます。AUHALでは、オーディオデバイスを対象としたプロパティの呼び出しが自動的にまとめて渡されます。

AUHALのインスタンスから接続できるデバイスは一度に1つだけです。そのため、入力と出力の両方を有効にできるのは、デバイスが両方を受け付けることができる場合のみです。たとえば、PowerPCベースのMacintoshコンピュータの内蔵オーディオは、単一のデバイスとして設定されていますが、これは入力オーディオデータ（マイク入力などを通じて）と出力オーディオ（スピーカーを通じて）の両方を受け付けることができます。

注： 現行のIntelベースのMacintoshコンピュータにおけるUSBオーディオデバイスや内蔵オーディオなど、一部のオーディオハードウェアは、入力と出力で別々のオーディオデバイスとして表されることがあります。これらの別々のデバイスを単一の`AudioDevice`オブジェクトに結合する方法の詳細については、「Mac OS Xでの機器セットの使用」（63 ページ）を参照してください。

信号フローの目的上、入力と出力の両方用に設定されたAUHALは2つのオーディオユニットとして動作します。たとえば、出力が有効になっているときは、AUHALから、その前のオーディオユニットのレンダリングコールバックが呼び出されます。オーディオユニットでデバイスからの入力データが必要になった場合は、そのオーディオユニットから、AUHALのレンダリングコールバックが呼び出されます。図 3-4は、入力および出力の両方に使用されるAUHALを示しています。

図 3-4　入力および出力に使用されるAUHAL



外部デバイスを通じて入力されるオーディオ信号は、オーディオデータストリームに変換されてAUHALに渡されます。AUHALでは、受け取ったオーディオデータストリームを別のオーディオユニットに送ることができます。データの処理（エフェクトの追加や、ほかのオーディオデータとのミックスなど）が済むと、出力がAUHALに送り返され、同じ外部デバイスを通じてそのオーディオを出力できます。

入力および出力のためのAUHALの使用例については、ADC Reference Libraryの*SimplePlayThru*および*CAPlayThrough*のサンプルコードを参照してください。`SimplePlayThru`では、単一のAUHALインスタンスを通じて入力と出力を処理する方法を示しています。`CAPlayThrough`では、AUHALを入力に使用し、デフォルト出力ユニットを出力に使用した、入出力の実装方法を示しています。

Mac OS Xでの機器セットの使用

ハードウェアオーディオデバイスとやり取りする際、Core Audioでは、抽象化のレベルをさらに追加して、機器セットを作成できます。機器セットは、複数の入力および出力デバイスを組み合わせて、見かけ上単一のデバイスとして扱うものです。たとえば、5チャネルのオーディオ出力に対応する必要がある場合、あるデバイスに出力の2チャネルを割り当て、別のデバイスに残りの3チャネルを割り当てることができます。データフローはCore Audioによって両方のデバイスに自動的に送られますが、アプリケーションではあたかも単一のデバイスとして出力とやり取りすることができます。また、Core Audioによって、オーディオの適切な同期を確保するための処理やレイテンシを最小限に抑えるための処理が自動的に行われるので、アプリケーションやプラグインに固有の詳細部分に専念できるようになります。

ユーザはAudio MIDI設定アプリケーションで、「オーディオ」>「機器セットエディタを開く」メニュー項目を選択することで、機器セットを作成できます。サブデバイスを選択して機器セットとして組み合わせると、ほかのハードウェアデバイスと同様に、デバイスの入力チャネルと出力チャネルを設定できるようになります。また、どのサブデバイスのクロックを同期のマスタとして機能させるかを指定する必要もあります。

ユーザが作成した機器セットはすべて、システムに対してグローバルになります。プログラミングでHAL Servicesの関数呼び出しを使用すれば、アプリケーションプロセスにのみ有効なローカルの機器セットを作成することもできます。機器セットはHAL内で`AudioAggregateDevice`オブジェクト（`AudioDevice`のサブクラス）として表されます。

注： 機器セットを使用して実装の詳細を隠すことができます。たとえば、USBオーディオデバイスでは、通常は入力と出力で別々のドライバが必要であり、これらは別々の`AudioDevice`オブジェクトとして表されます。しかし、グローバルの機器セットを作成することで、HALではそれらのドライバを単一の`AudioDevice`オブジェクトとして表すことができます。

機器セットではサブデバイスの情報が保持されます。ユーザがサブデバイスを削除した場合（または互換性のない方法で設定した場合）、それらのチャネルは機器セットから消失しますが、サブデバイスを再度接続または再設定するとチャネルが再び出現します。

機器セットには次に示す制限があります。

- 機器セットを構成するすべてのサブデバイスが同じサンプリングレートで実行されている必要があり、それらのデータストリームがミックス可能である必要があります。
- ボリューム、ミュート、入力ソースの選択など、設定が可能な制御機能はありません。

- すべてのサブデバイスをデフォルトのデバイスにできる場合を除いて、機器セットをデフォルトの入力または出力デバイスとして指定することはできません。デフォルトのデバイスにできないサブデバイスがある場合は、アプリケーションで機器セットを明示的に選択して使用できるようにする必要があります。
- 現時点で機器セットに追加できるデバイスは、IOAudioファミリ（つまり、カーネルレベル）ドライバによって表されるデバイスのみです。

Mac OS Xでのオーディオユニットの作成

オーディオユニットの作成の詳細については、『*Audio Unit Programming Guide*』を参照してください。

オーディオユニットのホスティング

プラグインであるオーディオユニットを使用するためには、それを読み込んで制御するためのホストアプリケーションが必要です。

オーディオユニットはComponent Managerのコンポーネントであるため、ホストアプリケーションではComponent Managerを呼び出してオーディオユニットを読み込む必要があります。ホストアプリケーションは、Audio Unitが次のいずれかのフォルダにインストールされている場合にそれらを検索してインスタンス化できます。

- ~/ライブラリ/Audio/Plug-Ins/Components。ここにインストールされたオーディオユニットは、ホームフォルダの所有者だけが使用できます。
- /ライブラリ/Audio/Plug-Ins/Components。ここにインストールされたオーディオユニットは、すべてのユーザが使用できます。
- /システム/ライブラリ/Components。Appleが提供するオーディオユニットのデフォルトの場所です。

使用可能なオーディオユニットの一覧を（ユーザに表示するなどの目的で）取得する必要がある場合は、Component Managerの関数CountComponentsを呼び出して、特定のタイプのオーディオユニットがいくつ使用できるのかを調べ、それからFindNextComponentを使用して反復処理を行い、各ユニットに関する情報を取得する必要があります。ComponentDescription構造体には、各オーディオユニットの識別子（タイプ、サブタイプ、および製造元コード）が格納されます。Appleが提供するオーディオユニットのComponent Managerのタイプおよびサブタイプの一覧については、「Mac OS Xのシステム付属オーディオユニット」（83 ページ）を参照してください。ホストではまた、（OpenComponentを呼び出して）各ユニットを開き、各種のプロパティ情報（オーディオユニットのデフォルトの入力および出力データ形式など）を問い合わせて、取得した情報をキャッシュしてユーザに提示することもできます。

ほとんどの場合、オーディオユニットに接続するにはオーディオ処理グラフが最も簡単な方法です。処理グラフを使用すると、オーディオユニットのインスタンス化または破棄を行う個々のComponent Manager呼び出しがAPIの側で実行されるという利点があります。グラフを作成するには、NewAUGraphを呼び出します。このAPIは新しいグラフオブジェクトを返します。その後、

AUGraphNewNodeを呼び出してオーディオユニットをグラフに追加できます。グラフの終端は出力ユニットにする必要があり、ハードウェアインターフェイス（デフォルト出力ユニットやAUHALなど）か汎用の出力ユニットである必要があります。

処理グラフを構成するユニットを追加したら、AUGraphOpenを呼び出します。この関数は、グラフ内の各オーディオユニットに対してOpenComponentを呼び出すことと同じ効果があります。この時点で、オーディオユニットのプロパティ（チャネル配置やサンプリングレートなど）、または特定のユニットに固有のプロパティ（ユニットの入出力数など）を設定できます。

オーディオユニットとの間で個々の接続を作成するには、AUGraphConnectNodeInputを呼び出し、接続する出力および入力を指定します。オーディオユニットチェーンの終端は出力ユニットにする必要があります。そうしないと、ホストアプリケーションではオーディオ処理を開始および停止する手段がなくなってしまいます。

オーディオユニットにユーザインターフェイスがある場合、その表示はホストアプリケーションで行います。オーディオユニットでは、Cocoaベースのインターフェイス、Carbonベースのインターフェイス、またはその両方を提供している可能性があります。ユーザインターフェイスのコードは、通常はオーディオユニットと一緒に組み込まれています。

- Cocoaベースのインターフェイスの場合、ホストアプリケーションでは、ユニットのプロパティkAudioUnitProperty_CocoaUIを問い合わせて、インターフェイスを実装しているカスタムクラス（NSViewのサブクラス）を見つけ、そのクラスのインスタンスを作成する必要があります。
- Carbonベースのインターフェイスの場合、ユーザインターフェイスは1つまたは複数のComponent Managerコンポーネントとして格納されています。コンポーネントの識別子（タイプ、サブタイプ、製造元）はkAudioUnitProperty_GetUIComponentListプロパティを問い合わせることで取得できます。その後、ホストアプリケーションから、対象のコンポーネントに対してAudioUnitCarbonViewCreateを呼び出すことによって、ユーザインターフェイスをインスタンス化できます。すると、インターフェイスがHIViewとしてウインドウに表示されます。

信号チェーンを構築したら、AUGraphInitializeを呼び出してオーディオユニットを初期化できます。この関数を呼び出すと、各オーディオユニットの初期化関数が呼び出され、オーディオユニットによるレンダリング用のメモリの割り当てやチャネル情報の設定などが可能になります。続いてAUGraphStartを呼び出し、処理を開始します。すると、出力ユニットが、チェーンの中で前のユニットのオーディオデータを（コールバックを通じて）要求します。その結果、その前のユニットが呼び出され、以下同様に処理が続きます。オーディオのソースとしてオーディオユニット（ジェネレータユニットやAUHALなど）を指定することもできます。または、ホストアプリケーションでオーディオデータそのものを供給することもできます。それには、信号チェーンの中で先頭のオーディオユニットにコールバックを登録します（ユニットのkAudioUnitProperty_SetRenderCallbackプロパティを設定します）。

オーディオユニットをインスタンス化する際、ホストアプリケーションでパラメータやプロパティの値の変更について知りたい場合があります。その場合は、変更が発生したときに通知が送られるリスナーオブジェクトを登録できます。こうしたリスナーを実装する方法の詳細については、「Technical Note TN2104: Handling Audio Unit Events」を参照してください。

ホストで信号処理を停止する必要が生じたときは、AUGraphStopを呼び出します。

グラフ内のすべてのオーディオユニットの初期化を解除するには、AUGraphUninitializeを呼び出します。未初期化状態に戻ったときも、引き続きオーディオユニットのプロパティを変更したり、接続の作成や変更を行ったりできます。AUGraphCloseを呼び出すと、CloseComponent呼び出しによって、グラフ内の各オーディオユニットの割り当てが解除されます。ただし、グラフに含まれているユニットに関するノード情報は引き続き保持されます。

処理グラフを破棄するには、AUGraphDisposeを呼び出します。グラフを破棄すると、グラフに含まれている、インスタンス化されたすべてのオーディオユニットが、自動的に破棄されます。

オーディオユニットのホスティングの例については、Core Audio SDKのServices/AudioUnitHostingおよびServices/CocoaAUHostのサンプルを参照してください。

オーディオユニットのユーザインターフェイスの実装例については、Core Audio SDKのAudioUnits/CarbonGenericViewのサンプルを参照してください。このサンプルは、ユーザによる調整が可能なパラメータを含む任意のオーディオユニットで使用できます。

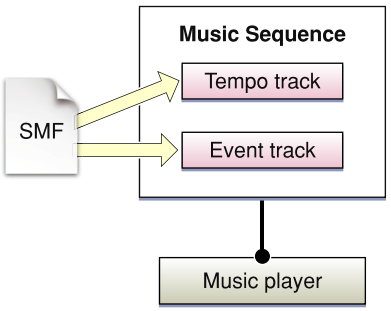
Component Managerの使用の詳細については、次のドキュメントを参照してください。

- *Component Manager Reference*
- *Component Manager for QuickTime*
- 「Inside Macintosh: More Macintosh Toolbox」にあるComponent Managerのドキュメント。これは旧来のドキュメントですが、Component Managerの概念を把握するのに役立ちます。

Mac OS XでのMIDIデータの処理

MIDIデータを操作する際、場合によっては、アプリケーションで標準MIDIファイル(SMF)からトラックデータを読み込む必要があります。図 3-5に示すように、Music Playerの関数（MusicSequenceLoadSMFWithFlagsまたはMusicSequenceLoadSMFDataWithFlags）を呼び出して、標準MIDIファイル内のデータを読み取ることができます。

図 3-5 標準MIDIファイルの読み取り

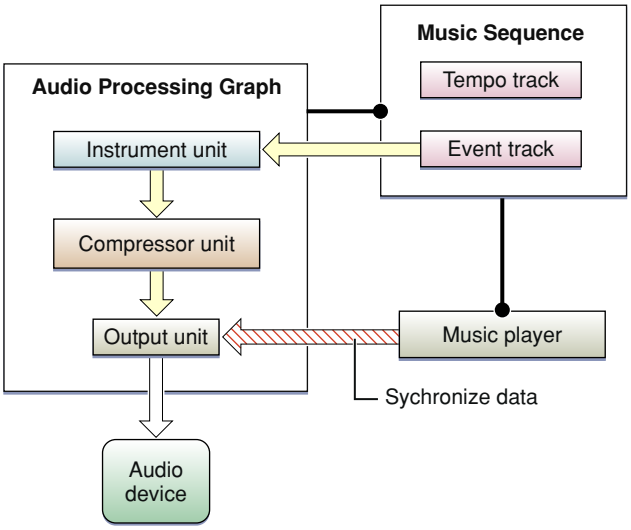


MIDIファイルのタイプや、ファイルを読み込むときに設定したフラグに応じて、すべてのMIDIデータを単一のトラックに格納したり、各MIDIチャネルをシーケンス内の別々のトラックとして格納したりできます。デフォルトでは、各MIDIチャネルがシーケンス内の新しいトラックに順にマップされます。たとえば、MIDIデータにチャネル1、3、および4が含まれている場合は、3つの新しいトラックがシーケンスに追加され、それぞれチャネル1、3、および4のデータが格納されます。これらのトラックは、シーケンスで既存のトラックの後に追加されます。シーケンス内の各トラックには、0から始まるインデックス値が割り当てられます。

タイミング情報（つまり、テンポイベント）は、テンポトラックに格納されます。

MIDIデータをシーケンスに読み込んだら、図3-6に示すように、ミュージックプレーヤーのインスタンスを割り当てて再生できます。

図 3-6 MIDIデータの再生

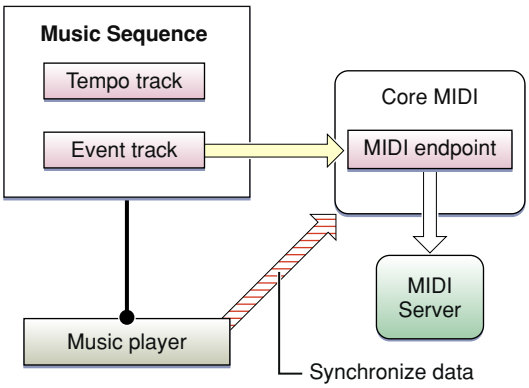


シーケンスは特定のオーディオ処理グラフに関連付ける必要があり、グラフ内の1つまたは複数のインスツルメントユニットをシーケンス内のトラックに割り当てることができます（トラックのマッピングを指定しない場合、ミュージックプレーヤーからは、プレーヤーによってグラフ内で見つけられた最初のインスツルメントユニットに、すべてのMIDIデータが送られます）。シーケンスに割り当てられたミュージックプレーヤーは、グラフの出力ユニットと自動的に通信して、出力されるオーディオデータの適切な同期を保ちます。コンプレッサユニットは必須ではありませんが、インスツルメントユニットの出力のダイナミックレンジを一定に保つのに役立ちます。

図3-7に示すように、シーケンス内のMIDIデータを外部MIDIハードウェア（または仮想MIDIデスティネーションとして設定されたソフトウェア）に送ることもできます。

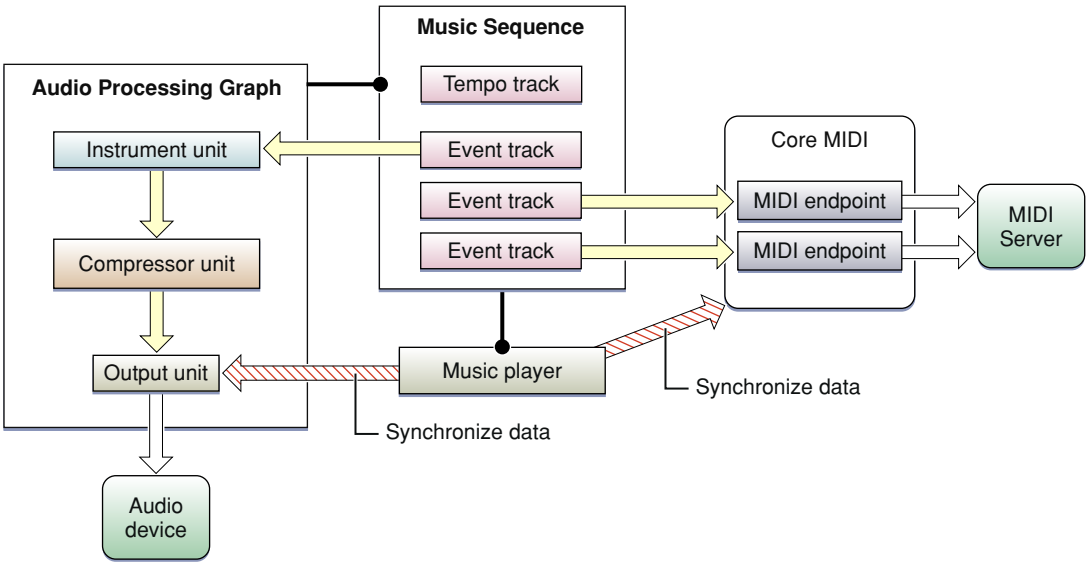
MIDI出力を目的とするトラックには、MIDIエンドポイントを割り当てる必要があります。ミュージックプレーヤーはCore MIDIと通信して、MIDIデバイスへのデータフローが正常に同期されるようにします。そして、Core MIDIはMIDI Serverと連携して、データをMIDI楽器に送信します。

図 3-7 MIDIデバイスへのMIDIデータの送信



トラックのシーケンスを、インスツルメントユニットとMIDIデバイスの組み合わせに割り当てることができます。たとえば、図 3-8に示すように、いくつかのトラックをインスツルメントユニットに割り当てて再生しながら、残りのトラックをCore MIDIに送って外部MIDIデバイスを再生することができます。

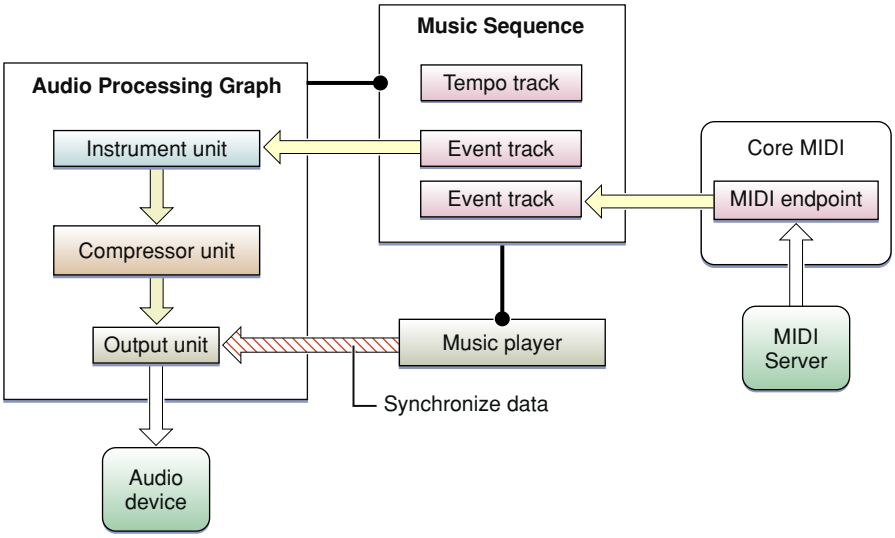
図 3-8 MIDIデバイスと仮想楽器の両方の再生



ミュージックプレーヤーは、出力が確実に同期されるように、オーディオ処理グラフの出力ユニットとCore MIDIとの間で自動的に調整を行います。

もう1つの共通のシナリオとして、図 3-9に示すように、新しいMIDI入力を受け付けながら既存のトラックデータを再生するというシナリオがあります。

図 3-9 新しいトラック入力の受け付け



既存のデータの再生はオーディオ処理グラフを通じて通常どおりに処理され、グラフから出力ユニットにオーディオデータが送られます。外部MIDIデバイスからの新しいデータは、Core MIDIを通じて入力され、割り当てられたエンドポイントを通じて転送されます。アプリケーションでは、この入力データについて反復処理を行い、MIDIイベントを新規または既存のトラックに書き込む必要があります。Music Player APIには、新しいトラックをシーケンスに追加する関数と、タイムスタンプ付きのMIDIイベントまたはその他のメッセージをトラックに書き込む関数が含まれています。

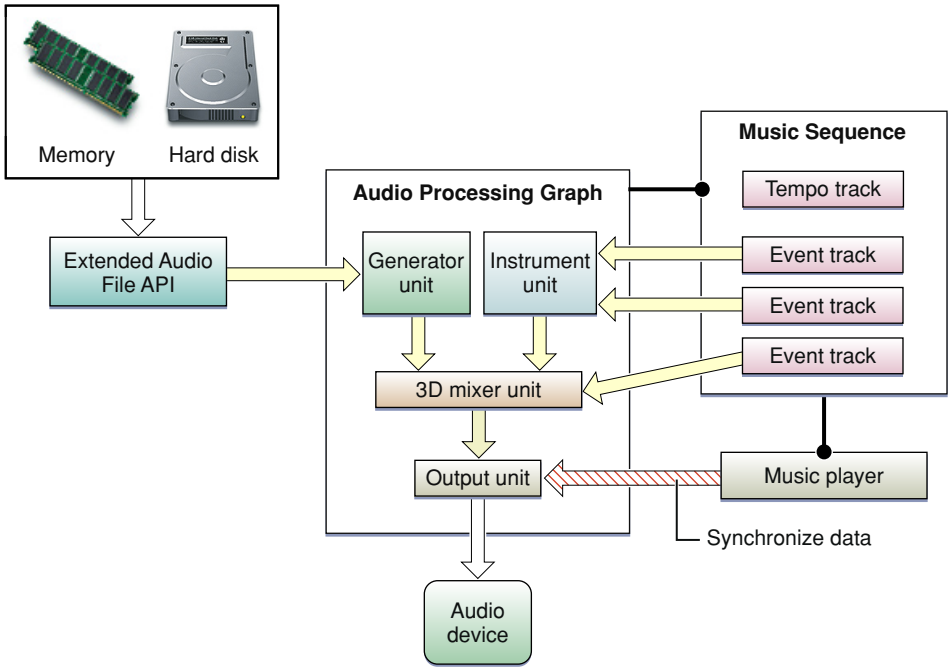
MIDIデータの処理と再生の例については、Core Audio SDKの次のサンプルを参照してください。

- `MIDI/SampleTools`。MIDIデータを送受信する簡単な方法を示します。
- `SimpleSDK/PlaySoftMIDI`。インスツルメントユニットと出力ユニットで構成される簡単な処理グラフに、MIDIデータを送信します。
- `SimpleSDK/PlaySequence`。MIDIファイルをシーケンスに読み取り、ミュージックプレーヤーを使用して再生します。

Mac OS XでのオーディオデータとMIDIデータの同時処理

場合によっては、オーディオデータと、MIDIデータから合成されたオーディオとをミックスして、その結果を再生したいことがあります。たとえば、多くのゲームのオーディオはバックグラウンドミュージックで構成されていますが、そのバックグラウンドミュージックはディスク上のオーディオファイルとして格納されています。また、イベントによって起動されるノイズ（足音や発砲音など）も一緒に格納されていますが、それらはMIDIデータとして生成されます。図3-10は、Core Audioを使用してこれらの2種類を混合できることを示しています。

図 3-10　オーディオとMIDIデータの混合



サウンドトラックのオーディオデータは、ディスクまたはメモリから取得され、Extended Audio File APIを使用してリニアPCMに変換されます。MIDIデータは、ミュージックシーケンス内でトラックとして格納されており、仮想インスツルメントユニットに送られます。仮想インスツルメントユニットからの出力はリニアPCMフォーマットであり、サウンドトラックのデータと混合できます。この例では3Dミキサーユニットを使用しています。このユニットは、3次元空間におけるオーディオソースの定位を操作できます。シーケンス内のトラックの1つからは、ミキサーユニットにイベントデータが送られています。ミキサーユニットでは定位パラメータが変化し、サウンドが時間とともに空間を移動するかのような効果が得られます。こうしたアプリケーションでは、競技者の動きを監視し、必要に応じて、特別な「動きトラック」にイベントを追加する必要があります。

ファイルベースのオーディオデータの読み込みと再生の例については、Core Audio SDKの`SimpleSDK/PlayFile`を参照してください。

Core Audioフレームワーク

Core Audioはいくつかの別々のフレームワークで構成されており、それらは/System/Library/Frameworksにあります。これらのフレームワークはアンブレラフレームワークの下で分類されているわけではないため、特定のヘッダを探すのは厄介かもしれません。この付録では、それぞれのCore Audioフレームワークと関連するヘッダファイルについて説明します。

iPhone OSおよびMac OS Xで使用できるフレームワーク

ここに列挙されているフレームワークはiPhone OS 2.0およびMac OS X v10.5で使用できます。

AudioToolbox.framework

Audio Toolboxフレームワークには、アプリケーションレベルのサービスを提供するAPIが含まれています。iPhone OSおよびMac OS XのAudio Toolboxフレームワークには、次のヘッダファイルが含まれています。

- AudioConverter.h：Audio Converter API。オーディオコンバータを作成および使用する場合に使用するインターフェイスを定義します。
- AudioFile.h：ファイル内のオーディオデータの読み取りと書き込みを行うためのインターフェイスを定義します。
- AudioFileStream.h：オーディオファイルストリームを解析するためのインターフェイスを定義します。
- AudioFormat.h：オーディオファイル内のオーディオフォーマットメタデータの割り当てと読み取りに使用するインターフェイスを定義します。
- AudioQueue.h：オーディオの再生と録音を行うためのインターフェイスを定義します。
- AudioServices.h：3つのインターフェイスを定義します。System Sound Servicesは、短いサウンドや警告音を再生できるようにします。Audio Hardware Servicesは、オーディオハードウェアとやり取りするための軽量のインターフェイスを提供します。Audio Session Servicesは、iPhoneおよびiPod touchアプリケーションでオーディオセッションを管理できるようにします。
- AudioToolbox.h：Audio Toolboxフレームワークの最上位のインクルードファイルです。
- AUGraph.h：オーディオ処理グラフを作成および使用する場合に使用するインターフェイスを定義します。
- ExtendedAudioFile.h：ファイルのオーディオデータを直接リニアPCMに変換する（またはその逆に変換する）場合に使用するインターフェイスを定義します。

Mac OS Xでは、さらに次のヘッダファイルがあります。

- AudioFileComponents.h：Audio File Component Managerコンポーネントのインターフェイスを定義します。オーディオファイルコンポーネントは、カスタムのファイル形式の読み書きを実装する場合に使用します。
- AudioUnitUtilities.h：オーディオユニットとやり取りするためのユーティリティ関数です。オーディオユニットパラメータ変換関数、およびリスナーオブジェクトを作成するためのオーディオユニットイベント関数が含まれています。これらの関数は、指定されたオーディオユニットパラメータが変更された場合にコールバックを呼び出します。
- CAFFile.h：Core Audio Formatオーディオファイルフォーマットを定義します。詳細については、『*Apple Core Audio Format Specification 1.0*』を参照してください。
- CoreAudioClock.h：アプリケーションやデバイスを同期させるためのタイミングソースを指定できるようにします。
- MusicPlayer.h：ミュージックシーケンス内のイベントトラックの管理と再生に使用するインターフェイスを定義します。
- AUMIDIController.h：廃止されました：使用しないでください。オーディオユニットで、指定されたMIDIソースからデータを受信できるようにするインターフェイスです。標準のMIDIメッセージはオーディオユニットパラメータ値に変換されます。このインターフェイスはMusic Player APIの関数に取って代わられています。
- DefaultAudioOutput.h：廃止されました：使用しないでください。デフォルトの出力ユニットにアクセスするための古いインターフェイスを定義します（Mac OS X v10.3以降では廃止されました）。

AudioUnit.framework

オーディオユニットフレームワークには、Core Audioのプラグインの管理に使用するAPIが含まれています。特に明記した場合を除いて、iPhone OSおよびMac OS Xのオーディオユニットフレームワークには、次のヘッダファイルが含まれています。

- AUComponent.h：オーディオユニットのタイプを定義します。
- AudioComponent.h：（iPhone OSのみ）オーディオコンポーネントを使用するためのインターフェイスを定義します。
- AudioOutputUnit.h：出力ユニットのオンまたはオフの切り替えに使用するインターフェイスを定義します。
- AudioUnit.h：Audio Unitフレームワークのインクルードファイルです。
- AudioUnitParameters.h：Appleのオーディオユニットが使用する定義済みのパラメータ定数です。サードパーティでも各自のオーディオユニットでこれらの定数を使用できます。
- AudioUnitProperties.h：一般的なオーディオユニットタイプやAppleのオーディオユニットのための定義済みのオーディオユニットプロパティです。

Mac OS Xでは、さらに次のヘッダファイルがあります。

- AUCocoaUIView.h：Cocoaのカスタムビューのプロトコルを定義します。Cocoaのカスタムビューを使用して、オーディオユニットのユーザインターフェイスを保持できます。CoreAudioKit.framework/AUGenericView.hも参照してください。
- AudioCodec.h：特にオーディオコーデックコンポーネントの作成に使用するインターフェイスを定義します。

- AudioUnitCarbonView.h：Carbonベースのオーディオユニットユーザインターフェイスを読み込んでやり取りするためのインターフェイスを定義します。Carbonインターフェイスは、Component Managerコンポーネントとしてパッケージ化されており、HIViewとして表されます。
- AUNTComponent.h：廃止されました：使用しないでください。古い「v1」オーディオユニットのインターフェイスを定義します。Mac OS X v10.3以降では廃止されました。AUComponent.hに置き換えられています。
- LogicAUProperties.h：Logic StudioアプリケーションのLogic Node環境で実行するオーディオユニットのインターフェイスです。
- MusicDevice.h：インスツルメントユニット（ソフトウェアベースのミュージックシンセサイザ）を作成するためのインターフェイスです。

CoreAudio.framework

Core Audioフレームワークには、すべてのCore Audioサービスに共通のデータ型が含まれています。また、ハードウェアとのやり取りに使用する低レベルのAPIも含まれています。Mac OS Xでは、HAL（ハードウェア抽象化層）Servicesのインターフェイスがこのフレームワークに含まれています。

iPhone OSおよびMac OS Xでは、このフレームワークに次のヘッダファイルが含まれています。

- CoreAudioTypes.h：すべてのCore Audioで使用されるデータ型を定義します。

Mac OS Xでは、さらに次のヘッダファイルがあります。

- AudioDriverPlugin.h：オーディオドライバプラグインとの通信に使用するインターフェイスを定義します。
- AudioHardware.h：オーディオデバイスオブジェクトとやり取りするためのインターフェイスを定義します。オーディオデバイスオブジェクトは、ハードウェア抽象化層(HAL)内の外部デバイスを表します。
- AudioHardwarePlugin.h：HALプラグインに必要なCFPluginインターフェイスを定義します。プラグインのインスタンスはHAL内でオーディオデバイスオブジェクトとして表されます。
- CoreAudio.h：Core Audioフレームワークの最上位のインクルードファイルです。
- HostTime.h：ホストの時間基準を取得および変換するための関数が含まれています。

OpenAL.framework

OpenALフレームワークは、OpenAL仕様を実装したものです。iPhone OSおよびMac OS Xでは、このフレームワークに次の2つのヘッダファイルが含まれています。

- al.h
- alc.h

iPhone OSでは、さらに次のヘッダファイルがあります。

- oalMacOSX_OALExtensions.h
- oalStaticBufferExtension.h

Mac OS Xでは、さらに次のヘッダファイルがあります。

- MacOSX_OALExtensions.h

iPhone OSでのみ使用できるフレームワーク

ここに列挙されているフレームワークはiPhone OSでのみ使用できます。

AVFoundation.framework

AV Foundationフレームワークは、ほとんどのアプリケーションで必要となる制御機能を付けてオーディオを再生するための、Objective-Cインターフェイスを提供します。iPhone OSのAV Foundationフレームワークには次の1つのヘッダファイルが含まれています。

- AVAudioPlayer.h：オーディオをファイルまたはメモリから再生するためのインターフェイスを定義します。

Mac OS Xでのみ使用できるフレームワーク

ここに列挙されているフレームワークはMac OS Xでのみ使用できます。

CoreAudioKit.framework

Core Audio Kitフレームワークには、オーディオユニット用のCocoaユーザインターフェイスを作成するためのAPIが含まれています。

- CoreAudioKit.h：Core Audio Kitフレームワークの最上位のインクルードファイルです。
- AUGenericView.h：オーディオユニットで使用する汎用のCocoaビュークラスを定義します。これは、オーディオユニットが独自のカスタムインターフェイスを作成しない場合に表示される最低限のユーザインターフェイスです。
- AUPannerView.h：パンナーオーディオユニットで使用する汎用のビューを定義し、インスタンス化します。

CoreMIDI.framework

Core MIDIフレームワークには、アプリケーションでMIDIのサポートを実装するために使用するすべてのCore MIDI Services APIが含まれています。

- CoreMIDI.h：Core MIDIフレームワークの最上位のインクルードファイルです。
- MIDIServices.h：MIDIデバイスと（MIDIエンドポイントや通知などを通じて）通信するアプリケーションのセットアップと設定に使用するインターフェイスを定義します。

- `MIDISetup.h`：MIDIシステムのグローバルな状態（使用可能なMIDIデバイスやMIDIエンドポイントなど）の設定またはカスタマイズに使用するインターフェイスを定義します。
- `MIDIThruConnection.h`：MIDIソースとMIDIデスティネーション間でMIDIプレイスルー接続を作成するための関数を定義します。MIDIスルー接続を使用すると、MIDIデバイスの連結が可能になり、あるデバイスへの入力を別のデバイスにパススルーすることもできるようになります。

CoreMIDIServer.framework

Core MIDI Serverフレームワークには、MIDIドライバのインターフェイスが含まれています。

- `CoreMIDIServer.h`：Core MIDI Serverフレームワークの最上位のインクルードファイルです。
- `MIDIDriver.h`：MIDIドライバがCore MIDI Serverとのやり取りに使用するCFPluginインターフェイスを定義します。

Core Audioサービス

この章では、Core Audioで使用できるサービスの一覧を示します。iPhone OSでは、これらのサービスが次のフレームワークに割り当てられています。

- **Audio Toolbox**—アプリケーションレベルのサービス：ファイル、ストリーム、警告、再生と録音。iPhone OSでは、Audio Session Servicesが含まれます。
- **Audio Unit**—オーディオユニットおよびオーディオコーデックサービス。
- **AV Foundation**—Objective-Cオーディオ再生インターフェイス。IPhone OSのみです。
- **Core Audio**—データ型。Mac OS Xでは、ハードウェアサービスが含まれます。
- **OpenAL**—定位、および低レイテンシのオーディオサービス。

Mac OS XのCore Audioには、これら4つのフレームワークに加えて、さらに次の3つが含まれます。

- **Core Audio Kit**—オーディオユニットユーザインターフェイスサービス。
- **Core MIDI**—アプリケーションレベルのMIDIサポート。
- **Core MIDI Server**—MIDIサーバおよびドライバのサポート。

フレームワークを中心として眺めたCore Audioのヘッダファイルの一覧については、付録「Core Audioフレームワーク」（71 ページ）を参照してください。

以降、この章ではサービスを中心として眺めたCore Audioを示します。まず、iPhone OSとMac OS Xの両方で使用できるサービスを示します。

iPhone OSおよびMac OS Xで使用できるサービス

ここに列挙されているサービスはiPhone OS 2.0およびMac OS X v10.5で使用できます。

Audio Converter Services

Audio Converter Servicesは、データフォーマットの変換を可能にします。このインターフェイスは、`AudioToolbox.framework`の`AudioConverter.h`ヘッダファイルで宣言されている関数、データ型、および定数で構成されています。

Audio File Services

Audio File Servicesは、ファイルまたはバッファに対するオーディオデータの読み書きを可能にします。このサービスをAudio Queue Servicesと組み合わせて使用し、オーディオの録音または再生を行います。iPhone OSおよびMac OS Xでは、Audio File Servicesは、`AudioToolbox.framework`の`AudioFile.h`ヘッダファイルで宣言されている関数、データ型、および定数で構成されています。

Audio File Stream Services

Audio File Stream Servicesは、オーディオファイルストリーム（つまり、必ずしもファイル全体にアクセスできるとは限らないオーディオデータ）の解析を可能にします。このサービスを使用してディスクからのファイルデータを解析することもできますが、その用途に設計されているのはAudio File Servicesです。

Audio File Stream Servicesは、コールバックを通じてオーディオデータとメタデータをアプリケーションに返します。通常は、その次にAudio Queue Servicesを使用して再生します。iPhone OSおよびMac OS Xでは、Audio File Stream Servicesは、`AudioToolbox.framework`の`AudioFileStream.h`ヘッダファイルで宣言されている関数、データ型、および定数で構成されています。

Audio Format Services

Audio Format Servicesは、オーディオデータフォーマット情報の操作を可能にします。Audio File Servicesなどのほかのサービスにも、その用途に使用するための関数があります。Audio Format Servicesは、オーディオデータフォーマット情報の取得のみを必要とする場合に使用します。Mac OS Xでは、このサービスを使用してシステム特性（エンコードで使用可能なサンプルレートなど）を取得することもできます。Audio Format Servicesは、`AudioToolbox.framework`の`AudioFormat.h`ヘッダファイルで宣言されている関数、データ型、および定数で構成されています。

Audio Processing Graph Services

Audio Processing Graph Servicesは、アプリケーションでのオーディオ処理グラフの作成と操作を可能にします。iPhone OSおよびMac OS Xでは、Audio Processing Graph Servicesは、`AudioToolbox.framework`の`AUGraph.h`ヘッダファイルで宣言されている関数、データ型、および定数で構成されています。

Audio Queue Services

Audio Queue Servicesは、オーディオの再生または録音を可能にします。また、再生の一時停止と再開、ループ再生の実行、複数チャネルのオーディオの同期も可能にします。iPhone OSおよびMac OS Xでは、Audio Queue Servicesは、`AudioToolbox.framework`の`AudioQueue.h`ヘッダファイルで宣言されている関数、データ型、および定数で構成されています。

Audio Unit Services

Audio Unit Servicesは、アプリケーションでのオーディオユニットの読み込みと使用を可能にします。

iPhone OSでは、Audio Unit Servicesは、AudioUnit.frameworkの次のヘッダファイルで宣言されている関数、データ型、および定数で構成されています。

- AUComponent.h
- AudioComponent.h（iPhone OSのみ）
- AudioOutputUnit.h
- AudioUnitParameters.h
- AudioUnitProperties.h

Mac OS Xでは、前述のヘッダファイルに加えて、AudioUnit.frameworkおよびAudioToolbox.frameworkの次のヘッダファイルも含まれます。

- AUCocoaUIView.h
- AudioUnitCarbonView.h
- AudioUnitUtilities.h（AudioToolbox.framework内）
- LogicAUProperties.h
- MusicDevice.h

OpenAL

OpenALは、ゲームアプリケーションでの使用を目的として開発されたオープンソースの定位オーディオテクノロジーです。iPhone OSおよびMac OS Xでは、OpenAL 1.1の仕様が実装されています。OpenALのヘッダには、OpenALフレームワークの次のヘッダファイルが含まれています。

- al.h
- alc.h

iPhone OSでは、さらに次のヘッダファイルがあります。

- oalMacOSX_OALExtensions.h
- oalStaticBufferExtension.h

Mac OS Xでは、さらに次のヘッダファイルがあります。

- MacOSX_OALExtensions.h

System Sound Services

System Sound Servicesは、短いサウンドや警告音を再生できるようにします。iPhoneでは、バイブレーションの起動をできるようにします。System Sound Servicesは、AudioToolbox.frameworkのAudioServices.hヘッダファイルで宣言されている関数、データ型、および定数のサブセットで構成されています。

iPhone OSでのみ使用できるサービス

ここに列挙されているサービスはiPhone OSでのみ使用できます。

Audio Session Services

Audio Session Servicesは、アプリケーションでのオーディオセッションの管理を可能にします。アプリケーションでのオーディオ動作を、iPhoneまたはiPod touch上のバックグランドアプリケーションと連携して調整します。Audio Session Servicesは、`AudioToolbox.framework`の`AudioServices.h`ヘッダファイルで宣言されている関数、データ型、および定数のサブセットで構成されています。

AVAudioPlayerクラス

`AVAudioPlayer`クラスは、サウンドを再生するためのObjective-C簡易インターフェイスを提供します。アプリケーションでステレオの定位や正確な同期を必要とせず、ネットワークストリームからキャプチャしたオーディオを再生しない場合は、このクラスを再生に使用することを推奨します。このクラスは`AVFoundation.framework`の`AVAudioPlayer.h`ヘッダファイルで宣言されています。

Mac OS Xでのみ使用できるサービス

ここに列挙されているサービスはMac OS Xでのみ使用できます。

Audio Codec Services

Audio Codec Servicesは、データフォーマットの変換を可能にします。このインターフェイスは、`AudioUnit.framework`の`AudioCodec.h`ヘッダファイルで宣言されている関数、データ型、および定数で構成されています。

- `AudioCodec.h`（`AudioUnit.framework`にあります）

Audio Hardware Services

Audio Hardware Servicesは、オーディオHAL（ハードウェア抽象化層）のいくつかの重要な機能に対する、小さい軽量のインターフェイスを提供します。Audio Hardware Servicesは、`AudioToolbox.framework`の`AudioServices.h`ヘッダファイルで宣言されている関数、データ型、および定数のサブセットで構成されています。

Core Audio Clock Services

Core Audio Clock Servicesは、アプリケーションやデバイスの同期に使用できる基準クロックを提供します。このサービスは、AudioToolbox.frameworkのCoreAudioClock.hヘッダファイルで宣言されている関数、データ型、および定数で構成されています。

Core MIDI Services

Mac OS XのCore Audioは、Core MIDI Servicesを通じてMIDIをサポートします。このサービスは、CoreMIDI.frameworkの次のヘッダファイルで宣言されている関数、データ型、および定数で構成されています。

- MIDIServices.h
- MIDISetup.h
- MIDIThruConnection.h
- MIDIDriver.h

Core MIDI Server Services

Core MIDI Server Servicesは、MIDIドライバによるMac OS X MIDIサーバとの通信を可能にします。このインターフェイスは、CoreMIDIServer.frameworkの次のヘッダファイルで宣言されている関数、データ型、および定数で構成されています。

- CoreMIDIServer.h
- MIDIDriver.h

Extended Audio File Services

Extended Audio File Servicesについて

多くの場合、Extended Audio File Servicesを使用します。このサービスは、オーディオデータの読み取りと書き込みを行うための最も単純なインターフェイスを提供します。このAPIを使用して読み取られたファイルは、自動的に伸長されたり、自動的にリニアPCMフォーマット（オーディオユニットのネイティブの形式）に変換されたりします。同様に、1つの関数呼び出しを使用して、リニアPCMオーディオデータを圧縮されたフォーマットまたは変換されたフォーマットでファイルに書き込むことができます。Core Audioでデフォルトでサポートされているファイルフォーマットの一覧については、「Mac OS Xでサポートされているオーディオファイルフォーマットとオーディオデータフォーマット」（87 ページ）を参照してください。フォーマットによっては制限があるものもあります。たとえば、デフォルトでは、Core AudioはMP3ファイルの読み取りはできますが、MP3ファイルの書き込みはできません。

HAL（ハードウェア抽象化層）Services

Mac OS XのCore Audioでは、アプリケーションがハードウェアを取り扱うための、結果を予測できる一貫したインターフェイスを提供するために、ハードウェア抽象化層(HAL)が使用されています。ハードウェアはそれぞれ、HAL内のオーディオデバイスオブジェクト（`AudioDevice`型）によって表されます。アプリケーションはオーディオデバイスオブジェクトに問い合わせて、タイミング情報を取得し、その情報を同期やレイテンシの調整に使用できます。

HAL Servicesは、`CoreAudio.framework`の次のヘッダファイルで宣言されている関数、データ型、および定数で構成されています。

- `AudioDriverPlugin.h`
- `AudioHardware.h`
- `AudioHardwarePlugin.h`
- `CoreAudioTypes.h`（すべてのCore Audioインターフェイスで使用されるデータ型と定数があります）
- `HostTime.h`

AppleのAUHALユニットは、ほとんどのデベロッパのハードウェアインターフェイスのニーズに対応しています。そのため、デベロッパがHAL Servicesと直接やり取りする必要はありません。AUHALは、指定されたオーディオデバイスオブジェクトに対して、必要なチャネルマッピングを含めてオーディオデータを送信する役割を担います。AUHALユニットやその他の出力ユニットの使用の詳細については、「Mac OS Xでのハードウェアとのインターフェイス」（60 ページ）を参照してください。

Music Player Services

Music Player ServicesはMac OS Xで使用できるサービスで、MIDI楽曲トラックの集まりの整理や再生を可能にします。このサービスは、`AudioToolbox.framework`の`MusicPlayer.h`ヘッダファイルで宣言されている関数、データ型、および定数で構成されています。

Mac OS Xのシステム付属オーディオユニット

この付録の表では、Mac OS X v10.5に付属のオーディオユニットの一覧を、Component Managerのタイプ別に分類して示します。これらすべてのユニットのComponent Manager製造元識別子は、`kAudioUnitManufacturer_Apple`です。

表 C-1 システム付属のエフェクトユニット(`kAudioUnitType_Effect`)

エフェクトユニット	サブタイプ	説明
AUBandpass	`kAudioUnitSubType_BandPassFilter`	1バンドのバンドパスフィルタ。
AUDynamicsProcessor	`kAudioUnitSubType_DynamicsProcessor`	ヘッドルーム、圧縮量、アタック時間、リリース時間などのパラメータを設定できるダイナミックプロセッサ。
AUDelay	`kAudioUnitSubType_Delay`	ディレイユニット。
AUFilter	`kAudioUnitSubType_AUFilter`	5バンドフィルタ。高周波および低周波のカットオフに加えて、3つのバンドパスフィルタを使用できます。
AUGraphicEQ	`kAudioUnitSubType_GraphicEQ`	10バンドまたは31バンドのグラフィックイコライザ。
AUHiPass	`kAudioUnitSubType_HighPassFilter`	レゾナンスのピークを調整できるハイパスフィルタ。
AUHighShelfFilter	`kAudioUnitSubType_HighShelfFilter`	高周波を固定量だけブーストまたはカットできるフィルタ。
AUPeakLimiter	`kAudioUnitSubType_PeakLimiter`	ピークリミッタ。
AULowPass	`kAudioUnitSubType_LowPassFilter`	レゾナンスのピークを調整できるローパスフィルタ。
AULowShelfFilter	`kAudioUnitSubType_LowShelfFilter`	低周波を固定量だけブーストまたはカットできるフィルタ。
AUMultibandCompressor	`kAudioUnitSubType_MultiBandCompressor`	4バンドコンプレッサ。
AUMatrixReverb	`kAudioUnitSubType_MatrixReverb`	ルームサイズや吸音特性などの空間特性を指定できるリバーブユニット。

エフェクトユニット	サブタイプ	説明
AUNetSend	kAudioUnitSubType_NetSend	オーディオデータをネットワーク経由でストリーミングするユニット。AUNetReceiveジェネレータオーディオユニットと組み合わせて使用します。
AUParametricEQ	kAudioUnitSubType_ParametricEQ	パラメトリックイコライザ。
AUSampleDelay	kAudioUnitSubType_SampleDelay	時間ではなくサンプル数でディレイを設定できるディレイユニット。
AUPitch	kAudioUnitSubType_Pitch	再生速度を変化させずにサウンドのピッチを変更できるエフェクトユニット。

表 C-2 システム付属のインスツルメントユニット(kAudioUnitType_MusicDevice)

インスツルメントユニット	サブタイプ	説明
DLSMusicDevice	kAudioUnitSubType_DLSSynth	SoundFontまたはDLS (Downloadable Sound)フォーマットのサウンドバンクを使用してMIDIデータを再生できる仮想インスツルメントユニット。サウンドバンクは、ホームディレクトリかシステムディレクトリのどちらかの/Library/Audio/Sounds/Banksフォルダに格納する必要があります。

表 C-3 システム付属のミキサーユニット(kAudioUnitType_Mixer)

ミキサーユニット	サブタイプ	説明
AUMixer3D	kAudioUnitSubType_3DMixer	複数の異なる信号を受信し、3次元空間に配置されるようにミックスできる、特殊なミキシングユニット。このユニットの使いかたの詳細については、「Technical Note TN2112: Using the 3DMixer Audio Unit」を参照してください。
AUMatrixMixer	kAudioUnitSubType_MatrixMixer	任意の数の入力を任意の数の出力にミックスするユニット。
AUMixer	kAudioUnitSubType_StereoMixer	任意の数のモノラルまたはステレオ入力を単一のステレオ出力にミックスするユニット。

表 C-4 システム付属のコンバータユニット(kAudioUnitType_FormatConverter)

コンバータユニット	サブタイプ	説明
AUConverter	kAudioUnitSubType_AUConverter	リニアPCMフォーマット内部のデータ変換を処理する汎用コンバータ。つまり、サンプルレート変換、整数から浮動小数点への変換（およびその逆）、インターリーブなどを処理できます。このオーディオユニットは、基本的にはオーディオコンバータを包むラッパーです。
AUDeferredRenderer	kAudioUnitSubType_DeferredRenderer	あるスレッドからの入力を取得し、その出力を別のスレッドに送信するオーディオユニット。このユニットを使用して、オーディオ処理の連なりを複数のスレッド間で分割できます。
AUMerger	kAudioUnitSubType_Merger	2つの別々のオーディオ入力を結合するユニット。
AUSplitter	kAudioUnitSubType_Splitter	オーディオ入力を2つの別々のオーディオ出力に分割するユニット。
AUTimePitch	kAudioUnitSubType_TimePitch	ピッチを変化させずに再生速度を変更できる、または再生速度を変化させずにピッチを変更できるユニット。
AUVarispeed	kAudioUnitSubType_Varispeed	再生速度を変更できるユニット（その結果ピッチも変更されます）。

表 C-5 システム付属の出力ユニット(kAudioUnitType_Output)

出力ユニット	サブタイプ	説明
AudioDeviceOutput	kAudioUnitSubType_HALOutput	ハードウェア抽象化層を使用してオーディオデバイスとのインターフェイスとなるユニット。AUHALとも呼ばれます。名前とは機能が異なりますが、AudioDeviceOutputユニットはデバイス入力を受け付けるように設定することもできます。詳細については、「Mac OS Xでのハードウェアとのインターフェイス」（60 ページ）を参照してください。
DefaultOutputUnit	kAudioUnitSubType_DefaultOutput	入力データを、ユーザが指定するデフォルトの出力（コンピュータのスピーカーなど）に送信する出力ユニット。
GenericOutput	kAudioUnitSubType_GenericOutput	汎用出力ユニット。出力ユニットの信号形式の制御および変換機能がありますが、出力デバイスとのインターフェイスはありません。通常は、オーディオ処理サブグラフの出力に使用します。「オーディオ処理グラフ」（52 ページ）を参照してください。

出力ユニット	サブタイプ	説明
SystemOutputUnit	kAudioUnitSubType_-SystemOutput	入力データを標準のシステム出力に送信する出力ユニット。システム出力とは、システムのサウンドやエフェクト用に指定された出力のことです。システム出力は、「サウンド」環境設定パネルの「サウンドエフェクト」タブで設定できます。

表 C-6 システム付属のジェネレータユニット(kAudioUnitType_Generator)

ジェネレータユニット	サブタイプ	説明
AUAudioFilePlayer	kAudioUnitSubType_-AudioFilePlayer	オーディオデータをファイルから取得して再生するユニット。
AUNetReceive	kAudioUnitSubType_-NetReceive	ストリーミングされたオーディオデータをネットワークから受信するユニット。AUNetSendオーディオユニットと組み合わせて使用します。
AUScheduledSoundPlayer	kAudioUnitSubType_-ScheduledSoundPlayer	1つまたは複数のメモリ内バッファからのオーディオデータを再生するユニット。

Mac OS Xでサポートされているオーディオファイルフォーマットとオーディオデータフォーマット

この付録では、Mac OS X v10.5のCore Audioでサポートされているオーディオデータフォーマットとオーディオファイルフォーマットについて説明します。

各オーディオファイルタイプには、そのタイプでサポートされているデータフォーマットの一覧があります。つまり、特定のファイルフォーマットから、一覧に示されたどのデータフォーマットへも変換するコンバータが存在します。AC3などの一部のデータフォーマットは、リニアPCMフォーマットに変換できず、したがって、標準のオーディオユニットで処理できません。

CAF (Core Audio Format)ファイルには、任意のフォーマットのオーディオデータを格納できます。CAFファイルフォーマットをサポートするどのアプリケーションも、オーディオデータをファイルに書き込んだり、ファイルに格納されているデータを取り出したりできます。ただし、CAFファイルの中に格納されているオーディオデータをエンコードまたはデコードできるかどうかは、システムでオーディオコーデックを使用できるかどうかによって決まります。

表 D-1　各ファイル形式で使用できるデータフォーマット

ファイル形式	データ形式
AAC (.aac、 .adts)	'aac '
AC3 (.ac3)	'ac-3'
AIFC (.aif、 .aiff、 .aifc)	BEI8, BEI16, BEI24, BEI32, BEF32, BEF64, 'ulaw', 'alaw', 'MAC3', 'MAC6', 'ima4' , 'QDMC', 'QDM2', 'Qclp', 'agsm'
AIFF (.aiff)	BEI8, BEI16, BEI24, BEI32
Apple Core Audio Format (.caf)	'.mp3', 'MAC3', 'MAC6', 'QDM2', 'QDMC', 'Qclp', 'Qclq', 'aac ', 'agsm', 'alac', 'alaw', 'drms', 'dvi ', 'ima4', 'lpc ', BEI8, BEI16, BEI24, BEI32, BEF32, BEF64, LEI16, LEI24, LEI32, LEF32, LEF64, 'ms\x00\x02', 'ms\x00\x11', 'ms\x001', 'ms\x00U', 'ms\x00', 'samr', 'ulaw'
MPEG Layer 3 (.mp3)	'.mp3'
MPEG 4 Audio (.mp4)	'aac '
MPEG 4 Audio (.m4a)	'aac ', alac'
NeXT/Sun Audio (.snd, .au)	BEI8, BEI16, BEI24, BEI32, BEF32, BEF64, 'ulaw'
Sound Designer II (.sd2)	BEI8, BEI16, BEI24, BEI32
WAVE (.wav)	LEUI8, LEI16, LEI24, LEI32, LEF32, LEF64, 'ulaw', 'alaw'

リニアPCMフォーマットのキー。例：BEF32 = ビッグエンディアンのリニアPCM 32ビット浮動小数点。

表 D-2 リニアPCMフォーマットのキー

LE	リトルエンディアン
BE	ビッグエンディアン
F	浮動小数点
I	整数
UI	符号なし整数
8/16/24/32/64	ビット数

Core Audioには、オーディオデータとリニアPCM間の変換を行うオーディオコーデックがいくつか組み込まれています。次のオーディオデータタイプ用のコーデックはMac OS X v10.4で使用できるものです。オーディオアプリケーションで追加のエンコーダおよびデコーダをインストールすることもできます。

オーディオデータタイプ	リニアPCMからのエンコード	リニアPCMへのデコード
MPEG Layer 3 ('.mp3')	×	○
MACE 3:1 ('MAC3')	○	○
MACE 6:1 ('MAC6')	○	○
QDesign Music 2 ('QDM2')	○	○
QDesign ('QDMC')	×	○
Qualcomm PureVoice ('Qclp')	○	○
Qualcomm QCELP ('qclq')	×	○
AAC ('aac ')	○	○
Apple Lossless ('alac')	○	○
Apple GSM 10:1 ('agsm')	×	○
ALaw 2:1 ('alaw')	○	○
Apple DRMオーディオデコーダ('drms')	×	○
AC-3	×	×
DVI 4:1 ('dvi ')	×	○
Apple IMA 4:1 ('ima4')	○	○
LPC 23:1 ('lpc ')	×	○
Microsoft ADPCM	×	○

オーディオデータタイプ	リニアPCMからのエンコード	リニアPCMへのデコード
DVI ADPCM	○	○
GSM610	×	○
AMR Narrowband (`'samr'`)	○	○
µLaw 2:1 (`'ulaw'`)	○	○

Mac OS Xでサポートされているオーディオファイルフォーマットとオーディオデータフォーマット

書類の改訂履歴

この表は「*Core Audioの概要*」の改訂履歴です。

日付	メモ
2008-11-13	iPhone OS 2.2用に更新。
2008-07-08	iPhone OS 2.0およびMac OS X v10.5用に更新。
2007-01-08	若干の訂正と言い回しの変更。
2006-08-07	Core Audioフレームワークの基本的な概念とアーキテクチャを紹介する新規文書。

書類の改訂履歴