オーディオセッションプログラミングガイド

Audio > Core Audio

2010-11-15

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

オーディオ動作の設定について

iOSは、オーディオ動作をアプリケーションレベル、アプリケーション間レベル、およびデバイスレベルで、*オーディオセッション*という概念を使って処理します。以下の問いに対する答えを、オーディオセッションAPIを使って実装します。

- 着信／サイレントスイッチでアプリケーションのオーディオを消音する必要があるか。アプリケーションを効果的に使うためにオーディオが不可欠でなければ、この質問への答えは「はい」となります。たとえば、会議中でも音で他人に迷惑をかけずに使えるメモ取りアプリケーションやゲームなどがこれにあたります。一方、発音辞典などは着信／サイレントスイッチを無視して再生できる必要があります。このアプリケーションの主たる目的を果たすためには音が必要だからです。
- オーディオが始まったときに、iPodオーディオの再生を継続させるのか。iPodライブラリにある曲に合わせてユーザが演奏できるようなバーチャルピアノのアプリケーションであれば、この質問への回答は確実に「はい」です。しかし、ストリーミングインターネットラジオを再生するアプリケーションであれば、アプリケーションがスタートしたらiPodオーディオは停止する必要があります。

ユーザがヘッドセットを抜き差ししたり、電話がかかってきたり、アラームが鳴ったりすることが考えられます。実際、iOSデバイスのオーディオ環境は非常に複雑です。iOS OSが大部分の処理を担いますが、デベロッパはその一方でオーディオセッションAPIを使って、ごくわずかなコードで設定を指定し、システムの要求にきちんと対応できます。

一連の動作をカプセル化するオーディオセッション

オーディオセッション は、オーディオ動作を設定するためにアプリケーションとiOSの間の仲介役として機能します。アプリケーションが起動すると、そのアプリケーション用のシングルトンオーディオセッションが自動的に提供されます。指定する動作は、「Using Sound」 in *iOS Human Interface Guidelines* （*iOS Human Interface Guidelines*）で説明しているように、ユーザの期待に応えられるものでなければなりません。

関連する章： 「オーディオセッションの基礎」 （13 ページ） と 「オーディオセッションの設定」 （21 ページ）。

カテゴリによるオーディオの役割の表現

オーディオの意図を表現する基本的な方法は、オーディオセッションカテゴリを設定することです。 **カテゴリ** は、一連のオーディオ動作を示すキーです。カテゴリを設定することによって、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や、オーディオに合わせてiPodオーディオの再生を続けるかどうかなどを指定します。

6つのオーディオセッションカテゴリと、一連のオーバーライドスイッチと修飾スイッチを使って、アプリケーションの特徴や役割に応じてオーディオ動作をカスタマイズできます。さまざまなカテゴリが再生、録音、再生録音、オフラインオーディオ処理に対応しています。アプリケーションのオーディオの役割をシステムが認識していれば、ハードウェア資源への適切なアクセスができます。また、デバイス上のその他のオーディオも、確実にアプリケーションにとって適切な方法で動作します。たとえば、iPodのオーディオを消音する必要がある場合は、そのとおりになります。

関連する章： 「オーディオセッションの設定」（21 ページ）と「オーディオセッションのカテゴリ」（69 ページ）。

デリゲートによる割り込み処理のサポート

オーディオ割り込みは、アプリケーションのオーディオセッションが非アクティブ化されることであり、これにより再生中のオーディオは即座に停止します。競合するオーディオセッションが組み込みのアプリケーションによってアクティブ化された場合、そのセッションにこちらのアプリケーションとミックスできない種類のカテゴリがシステムによって指定されていると、割り込みが発生します。アプリケーションのセッションが非アクティブ状態になった後、そのアプリケーションに「割り込みが発生しました」というメッセージが送られます。アプリケーションはこの通知に対する反応として、状態の保存やユーザインターフェイスの更新などの処理を実行できます。

割り込みを処理するには、AV Foundationフレームワークに含まれている、Objective-Cの割り込みのデリゲート メソッドを実装します。ユーザの視点から最小限の中断できちんと復旧するように `beginInterruption` メソッドと `endInterruption` メソッドを記述します。

関連する章： 「オーディオ割り込みの処理方法」（27 ページ）。

コールバックによるオーディオ経路変更処理のサポート

デバイスをドックに載せたりドックから外したりすることによって、またはヘッドセットを抜き差しすることによって **オーディオ経路の変化** が生じたときに、ユーザは何らかの期待をします。「Using Sound」 in *iOS Human Interface Guidelines*（*iOS Human Interface Guidelines*）に、このようなユーザの期待と、期待に応えるためのガイドラインについて解説されています。Cのコールバック関数を記述してそれをアプリケーションのオーディオセッションに登録することによって、経路の変化を処理します。

関連する章： 「オーディオハードウェア経路の変化」（35 ページ）。

プロパティによる高度な機能のサポート

さまざまな方法でオーディオセッションカテゴリを微調整できます。カテゴリに応じて、次のことができます。

- あるカテゴリでは通常は無効化されているときに、アプリケーション以外のオーディオ（iPodからなど）とのミックスを許可する
- オーディオ出力経路をレシーバからスピーカーに変更する
- Bluetoothオーディオ入力を有効にする
- アプリケーションのオーディオが再生したら、ほかのオーディオの音量を下げる（音を小さくする）必要があることを指定する

また、プロパティは、実行時にデバイスハードウェア用にアプリケーションを最適化するためにも使用します。これによりコードを、それを実行するデバイスの特性や、アプリケーションの実行時にユーザによって開始されたハードウェア機能の変化（ヘッドセットの接続など）に合わせて調整することができます。

プロパティを使用するには、Audio Toolboxフレームワークに含まれているC言語ベースの Audio Session Services APIを使用します。

関連する章：「オーディオセッションの設定」（21 ページ）、「デバイスハードウェアに合わせた最適化」（39 ページ）、「ムービーおよびiPodミュージックの使用」（43 ページ）

本書の使いかた

AV Foundationフレームワーク、Audio Queue Services、OpenAL、I/Oオーディオユニットを使うiOSデベロッパの方はすべて、この文書全体を読むことをお勧めします。それ以外のiOSデベロッパの方は、少なくともこの序章と次の章「オーディオセッションの基礎」（13 ページ）をお読みください。オーディオセッションの概念をもう十分に把握している方は、「オーディオセッションの活用」（47 ページ）の章に進んでください。

付録の「オーディオセッションのカテゴリ」（69 ページ）では、各カテゴリごとのiOSオーディオセッションが備える動作の詳細を示します。

お読みになる前に

この文書を読むための前提として、Cocoa Touchの開発について *iOS Application Programming Guide* で紹介する内容、およびCore Audioについて同書と *Core Audio Overview* で説明する基本事項を理解していることが必要です。オーディオセッションは、デバイス上の各種のスイッチ、ボタン、コネクタなどが関わる実際のエンドユーザの使用場面を念頭に置いているため、iOSデバイスの使いかたにも精通している必要があります。

関連項目

オーディオセッションについて学ぶために役立つ参考資料を次に示します。

- *Audio Session Services Reference* は、この文書と併せて参照するためのC言語APIリファレンスです。

- *AVAudioSession Class Reference*では、オーディオセッションを設定および使用するためのObjective-Cインターフェイスについて説明しています。
- *Core Audio Overview*では、iOSとMac OS XにおけるCore Audioのアーキテクチャと設計原則を説明しています。
- *iOS Application Programming Guide*では、Cocoa Touchアプリケーションのプログラミングに不可欠な基礎知識を説明しています。
- *iOS Human Interface Guidelines*は、オーディオアプリケーションのユーザインターフェイスに関するベストプラクティスを示します。特に、「Using Sound」 in *iOS Human Interface Guidelines*のセクションを参照してください。
- *AddMusic*は再生用アプリケーションにおけるオーディオセッションオブジェクトの使いかたを示すサンプルコードプロジェクトです。このサンプルは、アプリケーションオーディオとiPodオーディオの連携も示しています。

入手方法と動作環境

Audio Session APIは、iOSデバイスの要件に合うように特化されているため、iOS上でのみ利用できます。基本的なオーディオセッションは、iOS 2.0から対応しています。この文書で説明する機能のいくつかを動作させるには、iOS 3.1が必要です。

オーディオセッションの基礎

この章では、オーディオセッションによって解決されるiOS開発関連の問題点と、オーディオセッション機能の基礎知識について説明します。

iOSがオーディオを管理しなくてはならない理由

通勤時の車内で、iPhoneのロックを解除してPodcastの新しいエピソードを再生し始めたとしましょう。オーディオが内蔵スピーカーから聞こえてきます。同乗者に迷惑な顔をされるので、急いでヘッドセットを接続しますが、Podcastの再生は続き、オーディオ出力はヘッドセットから聞こえるように途中で切り替わります。さらに"数独"ゲームを起動すれば、そのサウンドエフェクトはPodcast出力と重なって再生されます。数秒後、Podcastの音が徐々に消えてアラームが鳴り、自分の誕生日を知らせる通知が表示されます。通知を消すと、再びPodcastの音量が大きくなり、途絶えた場所から再開します。数独ゲームのサウンドも動作を再開します。

この例のように、オーディオの状況はたった1分の間にも次々に変化します。しかもユーザは一度もオーディオの設定操作をしていません。それはiOSデバイスのオーディオの使い勝手が非常に優れているからですが、このために水面下では、Mac Proよりも複雑なインフラストラクチャが機能しています。オーディオセッションオブジェクトは、この手軽さを実現するインフラストラクチャへのアクセス手段として各アプリケーションに1つずつ用意されるオブジェクトです。

デベロッパはオーディオセッションを使って、シームレスなオーディオ機能を備えたアプリケーションを作成できます。実際のところ、AV Foundationフレームワーク、Audio Queue Services、OpenAL、I/Oオーディオユニットのいずれかを使うiOSアプリケーションの場合、*iOS Human Interface Guidelines*に示されているAppleの推奨事項に適合するにはオーディオセッションプログラミングインターフェイスを使う必要があります。

オーディオセッションとは

オーディオセッション は、オーディオ動作を設定するためにアプリケーションとiOSの間の仲介役として機能します。アプリケーションが起動すると、そのアプリケーション用のシングルトンオーディオセッションが自動的に提供されます。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。以下に例を示します。

- アプリケーションのサウンドを、ほかのアプリケーション（iPodなど）のサウンドとミックスするか、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。
- 時計のアラームなどのオーディオ割り込みに対してアプリケーションがどのように反応すべきか。
- ユーザがヘッドセットを抜き差ししたときにアプリケーションはどのように反応すべきか。

アプリケーションの実行中は、オーディオセッションの設定は、ユーザインターフェイスのサウンドエフェクトを除き、すべてのオーディオ動作に影響します。アプリケーションの実行環境となるデバイスのハードウェア特性（チャネル数、サンプルレート、オーディオ入力の可否など）をオーディオセッションに照会することができます。ハードウェア特性はデバイスごとに異なる可能性があり、アプリケーションの実行中にユーザのアクションによっても変化する可能性があります。

オーディオセッションのアクティブ化と非アクティブ化を明示的に行うことができます。アプリケーションのサウンドを再生したり、録音機能を作動させたりする場合は、オーディオセッションをアクティブ化する必要があります。オーディオセッションを非アクティブ化することもできます。実際、たとえば電話の着信時やアラームが鳴ったときに非アクティブになります。このような非アクティブ化のことを **割り込み**と呼びます。オーディオセッションAPIは、割り込みに反応したり割り込みから回復する手段となります。

オーディオセッションカテゴリとは

オーディオセッション**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示すキーです。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）をシステムに伝えます。iOSの6つのオーディオセッションカテゴリと、一連のオーバーライドスイッチおよび修飾スイッチを使って、アプリケーションのオーディオ動作をカスタマイズできます。

「オーディオセッションのカテゴリ」（69 ページ）」で詳しく説明するように、オーディオセッションカテゴリごとに、次の各動作に対して「はい」と「いいえ」のパターンを指定します:

- **ミキシングを許可する**：「はい」の場合、アプリケーションがサウンドを再生したときに、ほかのアプリケーション（iPodなど）からのオーディオの再生を継続できます。
- **サイレントスイッチや画面のロックによって消音する**：「はい」の場合、サイレントスイッチをサイレントに切り替えたり、画面をロックしたりすると、アプリケーションのオーディオは鳴らなくなります（iPhoneでは、このスイッチを *着信／サイレントスイッチ*と呼びます）。
- **オーディオ入力をサポートする**：「はい」の場合、アプリケーションのオーディオ入力（録音など）が許可されます。
- **オーディオ出力をサポートする**：「はい」の場合、アプリケーションのオーディオ出力（再生など）が許可されます。

ほとんどのアプリケーションは、カテゴリを設定する必要があるのは起動時に1回だけです。ただし、必要に応じて何度でも、セッションの状態がアクティブか非アクティブかに関係なくカテゴリを変更できます。オーディオセッションが非アクティブの場合、カテゴリ変更要求は、セッションをアクティブにしたときに送られます。オーディオセッションがアクティブな場合は、カテゴリ変更要求は即座に送られます。

オーディオセッションのデフォルト動作―何もしなくても使える機能

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。具体的には次のとおりです。

- 再生は有効になっており、録音は無効になっている
- ユーザがサイレントスイッチ（iPhoneの場合は着信／サイレントスイッチ）を「サイレント」位置に切り替えると、アプリケーションのオーディオは鳴らなくなる
- 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る
- 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）は鳴らなくなる

この一連の動作は`AVAudioSessionCategorySoloAmbient`オーディオセッションカテゴリというデフォルトのカテゴリでカプセル化され、指定されます。

オーディオセッションは、アプリケーションの起動時に自動的にアクティブ化されます。これにより、再生（またはオーディオ入力をサポートするカテゴリの1つを指定した場合は録音）が可能になります。しかし、デフォルトのアクティブ化に依存するのは、アプリケーションにとって危険な状態です。たとえば、iPhoneの着信音が鳴ったとき、ユーザが電話には出ずにアプリケーションを実行し続けていても、採用している再生技術によってはオーディオは鳴らなくなる可能性があります。次のセクションでは、このような問題に対処するための方法をいくつか説明し、「オーディオ割り込みの処理方法」（27 ページ）のセクションでさらに詳しく解説します。

アプリケーションの開発中は、デフォルト動作を利用して作業を進めることができます。しかし、出荷版アプリケーションに対しては、オーディオセッションを安全に無視できるのは以下の場合に限られます。

- アプリケーションがオーディオ用にSystem Sound ServicesまたはUIKitの`playInputClick`メソッドを使用しており、ほかのオーディオAPIは使用していない

  System Sound Servicesは、ユーザインターフェイスのサウンドエフェクトの再生やバイブレーションの起動を行うためのiOSテクノロジーです。それ以外の目的での使用には適していません（*System Sound Services Reference* と、サンプルコードプロジェクト *Audio UI Sounds (SysSound)* を参照）。

  UIKitの`playInputClick`メソッドを使えば、カスタム入力ビューやキーボードアクセサリビューで、標準的なキーボードクリック音を再生できます。そのオーディオセッションの動作は、システムが自動的に処理します。「Playing Input Clicks」 in *Text, Web, and Editing Programming Guide for iOS*（*Text, Web, and Editing Programming Guide for iOS*）を参照してください。

- まったくオーディオを使わないアプリケーションの場合

これらに該当しない場合、出荷版アプリケーションでデフォルトオーディオセッションを使ってはいけません。デフォルトカテゴリの使用を選択することができますが、「solo ambient」カテゴリが指定されているオーディオセッションを明示的に使用および管理する場合、次のセクションで説明するように、これをデフォルトで使用する場合と動作が異なります。

デフォルトオーディオセッションが実際のアプリケーションでほとんど使われない理由

オーディオセッションを初期化および設定して明示的に使わないと、アプリケーションは割り込みやオーディオハードウェアの経路の変化に反応できません。また、アプリケーション間のオーディオのミキシングについてOSが判断するときに、アプリケーションは考慮されません。

次に示すシナリオは、オーディオセッションのデフォルト動作の内容とその変更方法を分かりやすく示すための例です。

- シナリオ1：オーディオブックアプリケーションの場合。ユーザが『ヴェニスの商人』を聴いていると、バサーニオが登場したとたん自動ロックの時間になり、画面のロックと共にオーディオは止まってしまいます。

  画面がロックされても聴き続けられるようにするには、その動作をサポートするようにオーディオセッションを設定します。画面がロックされても再生を続行する場合に使うカテゴリは、`AVAudioSessionCategoryPlayback` です。

- シナリオ2：FPSシューティングゲームの場合（OpenALベースのサウンドエフェクト、BGMサウンドトラック付き。ただしBGMを切り、代わりにiPod Library Accessを使ってiPodライブラリの曲を再生することも可能）。ユーザはお気に入りの死闘ソングの再生を始めます。ところが、敵艦に向けて1発目の光子魚雷を射出したとたん、iPodの曲は停止してしまいます。

  iPodの音楽を中断させないためには、ミキシングを許可するようにオーディオセッションを設定します。 `AVAudioSessionCategoryAmbient` カテゴリを使うか、ミキシングをサポートするように再生カテゴリを変更します。

- シナリオ3：ストリーミングラジオアプリケーションの場合（再生のためにAudio Queue Servicesを使用）。ラジオを聴いている最中に電話がかかってくると、ラジオの音は止まります（これは正しい反応です）。ユーザは、この電話には出ないことにして警告を閉じ、ストリーミングで音楽の続きを聴こうと「再生(Play)」をタップしますが、何も起きません。再生を再開するには、いったんアプリケーションを終了して立ち上げ直す必要があります。

  オーディオキューの割り込みにきちんと対応するには、アプリケーションのオーディオ再生を自動的に、またはユーザが手動で再開できるように、デリゲートメソッドまたはオーディオセッションコールバック関数を記述します。「オーディオセッション割り込みの対応」（54 ページ）を参照してください。

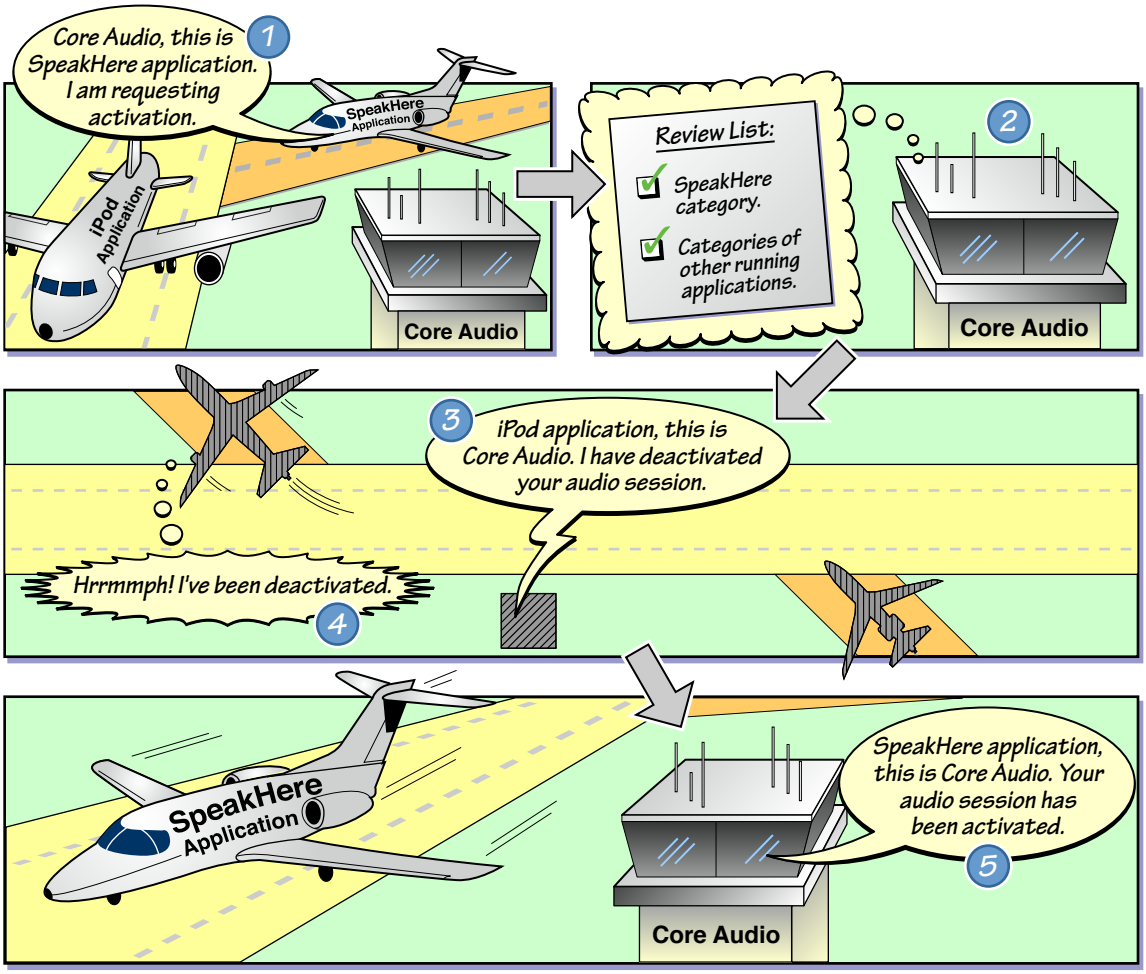
競合するオーディオ要求へのシステムの対処方法

iOSアプリケーションを起動したときに、バックグラウンドではすでに組み込みのアプリケーション（「SMS/MMS(Messages)」、「iPod」、「Safari」、「電話(Phone)」）が実行されている可能性があります。そしてそれらがオーディオを出力する必要があるかもしれません。たとえば、テキストメッセージの着信音や10分前に開始した再生中のPodcastなどです。

iPhoneを空港に例えるなら、アプリケーションは離陸許可を待つ飛行機、システムは管制塔のようなものといえます。アプリケーションはオーディオ要求を出したり、希望する優先度を表明したりできますが、実際に“滑走路”上で行われることを取りしきる最終決定権はシステムにあります。ア

プリケーションと“管制塔”との通信は、オーディオセッションを使って行います。 図 1-1 に、典型的なシナリオ、すなわちiPodが再生中のときにオーディオを使用したいというアプリケーションのシナリオを示します。このシナリオでは、アプリケーションは実際にiPodに割り込みます。

図 1-1　　競合するオーディオ要求をシステムが管理



図のステップ1では、アプリケーションがオーディオセッションのアクティブ化を要求します。これは、たとえばアプリケーションの起動時や、オーディオ録音再生アプリケーションの「再生(Play)」ボタンをユーザがタップした操作に対応して行われます。ステップ2では、アプリケーションからのアクティブ化要求の内容、つまり、オーディオセッションに割り当てられたカテゴリの扱いをシステムが検討します。図1-1では、SpeakHereというアプリケーションが、ほかのオーディオを止めるよう要求するカテゴリを指定しています。

ステップ3と4では、システムがiPodアプリケーションのオーディオセッションを非アクティブにしてそのオーディオ再生を停止します。最後のステップ5では、システムがSpeakHereアプリケーションのオーディオセッションをアクティブ化し、再生を開始します。

システムは、デバイス上に存在する任意のオーディオセッションをアクティブまたは非アクティブにする最終決定権を行使して、競合するさまざまなオーディオ要求の間を調整します。決定にあたっては、「常に電話が優先する」という破ることのできないルールに従います。どのようなアプ

リケーションがどれだけ高い優先度を求めても、電話の優先度を上回ることはできません。電話が着信したときは、進行中のオーディオ操作や設定されているカテゴリに関係なく、ユーザに着信が通知されてアプリケーションには割り込みがかかります。

アプリケーションのオーディオが電話の着信や時計のアラームに邪魔されないようにするには、ユーザの操作で機内モードに切り替えてもらう必要があります。これは、「デバイスを制御できるのはユーザであってアプリケーションではない」という、破ることのできないもう1つの規則があることを示しています。たとえば、時計のアラームが鳴らないようにプログラムから設定する方法はありません。アラームが録音を台無しにしないようにするには、予定されているアラームをユーザ側でオフにしておく必要があります。同様に、ハードウェアの再生音量をプログラムから設定する方法はありません。ハードウェアの音量は、デバイスの音量ボタンを使って常にユーザが調整します。

2つのオーディオセッションAPI

iOSは、オーディオセッションオブジェクトを操作する2つのAPIを提供しています。APIにはそれぞれ固有の長所があります。

- **AVAudioSessionクラス**。 *AVAudioSession Class Reference*に解説されているとおり、開発するアプリケーションのほかのObjective-Cコードとうまく連携する、便利なObjective-Cインターフェイスを提供します。このAPIを通じて、オーディオセッションの中核となる機能群にアクセスできます。

  `AVAudioSession`には2つの重要な長所があります。1つは、このAPIを使ってオーディオセッションの共有インスタンスを取得すると、セッションは暗黙的に初期化されることです。CのAPIを使った場合のように独立に初期化を実行する必要がないことです。もう1つは、単純なデリゲートメソッドを利用して、オーディオ割り込みやハードウェア設定（サンプルレートやチャネル数）への変更に対応できることです。デリゲートメソッドは、再生、録音、処理に使用しているオーディオテクノロジーに関係なく利用できます。

- **Audio Session Services**。 *Audio Session Services Reference*に解説されているとおり、オーディオセッションの基本機能と高度な機能のすべてにアクセスする手段となる、完全機能を備えたCのAPIです。

  オーディオハードウェア経路の変化を処理する場合や、オーディオセッションカテゴリの標準的な動作に変更を加える場合にはこのAPIが必要です。たとえば、`kAudioSessionProperty_OverrideAudioRoute`プロパティを使うと、「play and record」オーディオセッションカテゴリを使っているときにレシーバからスピーカーへ再生を切り替えることができます。このAPIはまた、割り込み処理のためのコールバックのメカニズムも備えています。

この2つのオーディオセッションAPIは、相互に完全に互換性があるため、これらのAPIの呼び出しをさまざまに組み合わせることができます。以降の章では、これらのAPIのさまざまな機能の使用について深く掘り下げていきます。

オーディオセッションAPIを使ったアプリケーション開発

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追加したときに、シミュレータやデバイス上でアプリケーションを実行することは可能です。しかし、シミュレータではオーディオセッション動作はシミュレートできず、デバイスのハードウェア機能にアクセスすることもできません。シミュレータで実行する場合、以下の操作はできません。

- 割り込みの呼び出し
- サイレントスイッチの設定変更
- 画面ロックのシミュレート
- ヘッドセットの抜き差しのシミュレート
- オーディオ経路情報の照会や、オーディオセッションカテゴリ動作のテスト
- オーディオミキシング動作（たとえばアプリケーションのオーディオと、iPodなどのほかのアプリケーションからのオーディオを一緒に再生するなど）のテスト

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。シミュレータを使う場合の開発のヒントについては、「シミュレータ上でのアプリケーションの実行」（66 ページ）を参照してください。

オーディオセッションの設定

オーディオセッションを設定して、アプリケーションに対して基本的なオーディオ動作を定めます。最も重要なことは、アプリケーションが何をし、デバイスやシステムとどのようにやり取りする必要があるかに応じて、オーディオセッションのカテゴリを設定することです。設定したカテゴリは、アプリケーションの実行時に必要に応じて変更することができます。

ただし、設定のための前提条件は初期化を行うことであり、これについてまず説明します。

オーディオセッションの初期化

システムは、アプリケーションの起動時にオーディオセッションオブジェクトを用意します。セッションを操作する前に、まずこれを初期化する必要があります。初期化を処理する方法は、主に、オーディオ割り込みをどのように処理する必要があるかによって異なります。

- 割り込みの管理にAV Foundationフレームワークを使っている場合は（詳細は「デリゲートメソッドを使った割り込み処理」（30 ページ）を参照）、以下に示すように `AVAudioSession` オブジェクトへの参照を取得するときに暗黙的に行われる初期化を利用します。

  ```
  // オーディオセッションを暗黙的に初期化
  AVAudioSession *session = [AVAudioSession sharedInstance];
  ```

  `session` 変数は、この時点で初期化済みのオーディオセッションを表しており、すぐに使用できます。Appleは、 `AVAudioSession` クラスの割り込みデリゲートメソッド、または `AVAudioPlayer` クラスと `AVAudioRecorder` クラスのデリゲートプロトコルを使ってオーディオ割り込みを処理するときには、暗黙的な初期化を使うことを推奨しています。

- あるいは、オーディオ割り込みを処理するCのコールバック関数を作成できます（詳しくは「コールバック関数を使った割り込み処理」（31 ページ）を参照）。その場合は、`AudioSessionInitialize` 関数による明示的な初期化を通じて、そのコードをオーディオセッションに対応付ける必要があります。通常、オーディオセッションの初期化は、アプリケーションのメインコントローラクラスにおいて、そのクラスの初期化メソッドの内側で明示的に行う必要があります。「オーディオセッションの初期化」（47 ページ）にサンプルコードを示します。

オーディオセッションのアクティブ化と非アクティブ化

システムは、アプリケーションの起動時にオーディオセッションをアクティブにしますが、それでもAppleは、アプリケーションのオーディオセッションを、一般的にはアプリケーションの `viewDidLoad` メソッドの中で明示的にアクティブにすることを推奨しています。これにより、アクティブ化が成功したかどうかを確認する機会が得られます。同様に、オーディオセッションのアク

ティブ／非アクティブの状態を変更したときには、その要求が成功したかを確認します。セッションのアクティブ化がシステムに拒否された場合にきちんと対処できるようにコードを作成してください。

システムは、時計やカレンダーのアラーム、または電話の着信を受けてオーディオセッションを非アクティブにします。ユーザがアラームを消したり電話に出なかった場合、システムはアプリケーションのオーディオセッションを再びアクティブにします。割り込みが終了したら、オーディオが確実に動作するように再度アクティブにしてください。サンプルコードについては「オーディオセッションのアクティブ化と非アクティブ化」（48 ページ）を参照してください。

`AVAudioPlayer`オブジェクトまたは`AVAudioRecorder`オブジェクトを使った、特定のケースのオーディオ再生やオーディオ録音では、システムがオーディオセッションの再アクティブ化を行います。その場合でも、`AVAudioSessionDelegate`プロトコルの割り込みデリゲートメソッドを実装して、オーディオセッションを明示的に再アクティブ化するよう推奨します。このメソッドで、再アクティブ化が成功したかどうか確認した上で、アプリケーションの状態やユーザインターフェイスを更新することができます。詳しくは「オーディオ割り込みの処理方法」（27 ページ）を参照してください。

ほとんどのアプリケーションでは、オーディオセッションを明示的に非アクティブにする必要はありません。重要な例外として、VoIP（Voice over Internet Protocol）アプリケーションや、特別な録音アプリケーションがあります。

- VoIPアプリケーションは、ほとんどの時間にわたってバックグラウンドで動作し、発着信を処理する間だけオーディオセッションをアクティブにする必要があります。バックグラウンドで着信を待機している状態では、オーディオセッションをアクティブにしてはならないのです。
- 「録音」カテゴリを用いるアプリケーションでは、録音中に限ってオーディオセッションをアクティブにしなければなりません。録音の開始前や停止後は、セッションを非アクティブにして、テキストメッセージによるアラートの着信など、ほかのサウンドが再生できる状態にすることができます。あるいは、録音アプリケーション自身にも再生機能を組み込む場合、録音中以外は再生カテゴリに切り替える、という方法もあります。これによっても、ほかのサウンドを再生できることになります。

最適なカテゴリの選択

iOSのオーディオセッションカテゴリには6つの種類があります。

- 再生用に3つ
- 録音用に1つ
- 再生と録音（必ずしも同時に行うわけではない）が可能なカテゴリ
- オフラインオーディオ処理用に1つ

最適なカテゴリを選択するには、次のいくつかの要素を検討します。

- 最適なカテゴリは、アプリケーションのオーディオの要求事項を最も的確に満たすカテゴリです。オーディオの再生と録音のどちらが必要か、両方とも必要か、あるいは単にオフラインオーディオ処理を行うだけでいいのか。

- 再生するオーディオは、アプリケーションを使う上で不可欠なものか、重要でないものか。不可欠であれば、最適なカテゴリは着信／サイレントスイッチがサイレントに設定されても再生が可能なカテゴリです。重要でなければ、着信／サイレントスイッチがサイレントに設定されたときに音を消すカテゴリを選択します。

- ユーザがアプリケーションを起動したときに、ほかのオーディオ（iPodオーディオなど）が再生されているか。起動中にチェックすることで分岐できます。たとえば、ゲームアプリケーションでは、iPodオーディオがすでに再生していれば再生を続けるようなカテゴリ設定を選択したり、あるいはアプリケーションに組み込まれているサウンドトラックをサポートするほかのカテゴリ設定を選択したりできます。

次のリストはカテゴリの説明です。最初のカテゴリでは、アプリケーション以外のオーディオの再生を継続できます。残りのカテゴリでは、アプリケーションのオーディオセッションがアクティブになった時点でほかのオーディオを止める必要があることを指定できます。ただし、「カテゴリの微調整」（25 ページ）で解説しているように、ミックス不可能な「再生」カテゴリと「録音」カテゴリがミックス可能になるようにカスタマイズできます。

- `AVAudioSessionCategoryAmbient` またはこれと同等の `kAudioSessionCategory_AmbientSound`— このカテゴリは、アプリケーションを洗練させたり、興味を掻き立てたりするサウンドを再生するけれども、そのサウンドはアプリケーションを使う上で必須ではないというときに使います。このカテゴリを使うと、アプリケーションのオーディオは着信／サイレントスイッチによって、および画面がロックされたときに消音されます。

  このカテゴリでは、アプリケーションがオーディオを再生している間、iPod、SafariなどiPhoneに組み込まれているアプリケーションのオーディオを再生できます。たとえば、ユーザがiPodオーディオに合わせて演奏するバーチャル楽器を提供するアプリケーションなどに対してこのカテゴリを使います。

- `AVAudioSessionCategorySoloAmbient` またはこれと同等の `kAudioSessionCategory_SoloAmbientSound`— このカテゴリは、ユーザが着信／サイレントスイッチを「サイレント」の位置に切り替えたときや画面がロックされたときに音を消す必要があるアプリケーションに対して使います。これがデフォルトのカテゴリです。`AVAudioSessionCategoryAmbient` カテゴリとの違いは、アプリケーション以外のオーディオを消音する点です。

- `AVAudioSessionCategoryPlayback` またはこれと同等の `kAudioSessionCategory_MediaPlayback`— このカテゴリは、オーディオ再生が最も重要なアプリケーションに対して使います。アプリケーションのオーディオは、画面がロックされた状態や、着信／サイレントスイッチがサイレントに設定された状態でも再生します。

  注： 画面がロックされたときでもオーディオの再生を続けることができるオーディオセッションカテゴリを選んだ場合、通常はシステムのスリープタイマを無効に しない でください。スリープタイマは、ユーザの指定した時間が経過すると画面を暗くしてバッテリ電力を節約する機能です。

- `AVAudioSessionCategoryRecord` またはこれと同等の `kAudioSessionCategory_RecordAudio`— このカテゴリは、オーディオの録音用に使います。電話の呼び出し音と、時計やカレンダーのアラームを除き、iPhone上のすべての再生が消音されます。

- `AVAudioSessionCategoryPlayAndRecord` またはこれと同等の `kAudioSessionCategory_PlayAndRecord`— このカテゴリは、オーディオの入出力を行うアプリケーションに対して使います。入出力は必ずしも同時に行う必要はありませんが、必要な場合は可能です。このカテゴリは、オーディオチャットアプリケーションでの使用に適しています。

- `AVAudioSessionCategoryAudioProcessing` またはこれと同等の `kAudioSessionCategory_AudioProcessing`— このカテゴリは、再生や録音ではなくオフラインオーディオ処理を行う際に使います。

各カテゴリに対応する実際の動作は、アプリケーションで制御することはできず、最終的にはオペレーティングシステムの判断によって決まります。また、将来のiOSバージョンでは、改良によってカテゴリの動作が変更される可能性もあります。作成するアプリケーションの用途に最もよく合ったオーディオ動作のカテゴリを選ぶよう心がけてください。カテゴリごとのオーディオ動作の詳細については、付録の 「オーディオセッションのカテゴリ」（69 ページ）を参照してください。

アプリケーションの起動時にほかのオーディオが再生中かどうかをチェックし、それに基づいてカテゴリを選ぶ方法があります。起動時（`viewDidLoad` メソッド内など）に、`kAudioSessionProperty_OtherAudioIsPlaying` プロパティの値をチェックします。

エンコードとデコードへのカテゴリの影響

iOSでは、圧縮されていないオーディオをさまざまな圧縮フォーマットにエンコードできます。また、圧縮フォーマットを再生用にデコードすることもできます。デバイス上のハードウェアコーデックは、ソフトウェアコーデックと比べて電力を消費しません。さらに、ハードウェアコーデックは、本来メインプロセッサが行う作業を実行するため、CPUサイクルをほかのタスクのために解放できます。最良のパフォーマンスとバッテリ持続時間のために、圧縮したオーディオフォーマットの操作にはできる限りハードウェアコーデックを使います。これは、ゲームのようなグラフィックを多用するアプリケーションでは特に重要になる場合があります。ソフトウェアによるオーディオデコード処理を使うと、アプリケーションの最大ビデオフレームレートが制限される可能性があるためです。

デバイスのハードウェアコーデックは、アプリケーションのオーディオセッションカテゴリを、ほかのオーディオ（iPodオーディオなど）を消音するように設定する場合に限り、アプリケーションで利用できます。つまり、ハードウェアによるオーディオエンコードやデコードの処理にアクセスできるようにするには、オーディオ再生のための排他的な権限を取得する必要があるということです。同様に、圧縮オーディオフォーマットへのハードウェアエンコード処理を使うには、アプリケーションはオーディオ入力に対する排他的な権限を取得する必要があります。

次のカテゴリでは、ハードウェアによるオーディオエンコードとデコードの使用が可能です。

- `AVAudioSessionCategorySoloAmbient` またはこれと同等の `kAudioSessionCategory_SoloAmbientSound`

- `AVAudioSessionCategoryPlayback` またはこれと同等の `kAudioSessionCategory_MediaPlayback`

- `AVAudioSessionCategoryPlayAndRecord` またはこれと同等の `kAudioSessionCategory_PlayAndRecord`

- AVAudioSessionCategoryAudioProcessing またはこれと同等の kAudioSessionCategory_AudioProcessing

「カテゴリの微調整」（25 ページ）で解説しているように、再生カテゴリのミックス不可能の特性をオーバーライドする場合はハードウェアコーデックは使用できません。ハードウェアコーデックを使ってオフラインオーディオ変換を行っており、オーディオを同時に再生する必要がない場合は、AVAudioSessionCategoryAudioProcessing（またはこれと同等の kAudioSessionCategory_AudioProcessing）カテゴリを使います。

オーディオセッションのカテゴリの設定と変更

ほとんどのiOSアプリケーションの場合、オーディオセッションのカテゴリを起動時に設定する（なおかつそれを変更しない）とうまく動作します。これによりアプリケーションが実行している間はデバイスのオーディオ動作が一貫した状態に保たれるため、ユーザ体験を高めることになります。例外として、オーディオ録音アプリケーションの場合は状態に応じてカテゴリを変更する必要があります。

録音の場合は、AVAudioSessionCategoryRecord（またはこれと同等の kAudioSessionCategory_RecordAudio）カテゴリを使います。これで、録音対象のオーディオがテキストメッセージの着信のようなデバイスのサウンドに妨げられることはありません。録音が終わったら（ユーザが「停止」ボタンをタップしたときなど）、アプリケーションの必要性に応じてカテゴリを適切な再生カテゴリに変更します。再生カテゴリのいずれを使用しても、テキストメッセージ通知のサウンドは聞こえます。

同じことが、オーディオ入力を使うけれども必ずしも録音を行わない、オーディオスペクトラムアナライザのようなオーディオ測定アプリケーションにも当てはまります。

オーディオセッションのカテゴリを設定する方法を示したサンプルコードについては、「カテゴリの設定」（48 ページ）（「オーディオセッションの活用」の章）を参照してください。

カテゴリの微調整

さまざまな方法でオーディオセッションカテゴリを微調整できます。カテゴリに応じて、次のことができます。

- あるカテゴリでは通常は無効化されているときに、アプリケーション以外のオーディオ（iPod からなど）とのミックスを許可する
- オーディオ出力経路をレシーバからスピーカーに変更する
- Bluetoothオーディオ入力を有効にする
- アプリケーションのオーディオが再生したら、ほかのオーディオの音量を下げる（音を小さくする）必要があることを指定する

AVAudioSessionCategoryPlayback（またはこれと同等の kAudioSessionCategory_MediaPlayback）カテゴリと AVAudioSessionCategoryPlayAndRecord（またはこれと同等のkAudioSessionCategory_PlayAndRecord）カテゴリのミックス不可の特性

をオーバーライドできます。オーバーライドを実行するには、kAudioSessionProperty_OverrideCategoryMixWithOthers プロパティをオーディオセッションに適用します。サンプルコードについては、「再生のミックス動作の変更」（50 ページ）を参照してください。

アプリケーションとほかのオーディオ（iPodオーディオなど）をミックスできるように再生カテゴリを変更すると、圧縮フォーマットのハードウェアによるオーディオデコード処理は使用できません。

アプリケーションのプログラムがオーディオ出力経路を変更することはできます。AVAudioSessionCategoryPlayAndRecord（またはこれと同等のkAudioSessionCategory_PlayAndRecord）カテゴリを使うと、オーディオは通常はレシーバ（電話がかかってきたときに耳に付ける小型のスピーカー）へ送られます。カテゴリ経路のオーバーライドを使って、iPhoneの下部にあるスピーカーへオーディオをリダイレクトすることができます。サンプルコードについては「出力オーディオのリダイレクト」（53 ページ）を参照してください。

iOS 3.1から、HFPをサポートするペア登録済みBluetoothデバイスからの入力を許可するように録音カテゴリを設定できます。サンプルコードについては、「Bluetoothオーディオ入力のサポート」（54 ページ）を参照してください。

最後に、アプリケーションのオーディオが再生されている間、ほかのオーディオの音量が自動的に下がるようにカテゴリを調整できます。これは、たとえばエクササイズアプリケーションなどに使用できます。ユーザがiPodに合わせてエクササイズを行っているときに、アプリケーションから「ボートこぎを開始して10分経過しました」といったメッセージを伝える必要があるとします。アプリケーションからのメッセージが確実に分かるようにするには、オーディオセッションにkAudioSessionProperty_OtherMixableAudioShouldDuck プロパティを適用します。音を下げる（音を小さくする）と、デバイス上のほかのすべてのオーディオ（電話の音以外）の音量が下がります。

オーディオ割り込みの処理方法

割り込みを処理するオーディオセッションコードを追加すると、電話を着信したときや、時計やカレンダーのアラームが鳴ったときでもアプリケーションのオーディオはきちんと動作を継続します。

オーディオ割り込み は、アプリケーションのオーディオセッションを非アクティブにすることです。採用するテクノロジーに応じて、割り込みにより再生中のオーディオは即座に停止するか一時停止します。競合するオーディオセッションが組み込みのアプリケーションによってアクティブ化された場合、そのセッションにこちらのアプリケーションとミックスできない種類のカテゴリがシステムによって指定されていると、割り込みが発生します。アプリケーションのセッションが非アクティブ状態になった後、そのアプリケーションに「割り込みが発生しました」というメッセージが送られます。アプリケーションはこの通知に対する反応として、状態の保存やユーザインターフェイスの更新などの処理を実行できます。

割り込みが発生した後には、アプリケーションが終了させられることもあります。これは、着信した電話をユーザが受けた場合に発生します。電話に出なかったり、アラームを消したりした場合、システムは割り込み終了メッセージを発行し、アプリケーションは動作を続行します。オーディオを再開するには、オーディオセッションをアクティブ化しなおす必要があります。

オーディオ割り込み処理のテクニック

割り込みに対応するには2つのアプローチがあります。

- AV Foundationフレームワークに含まれている、Objective-Cの割り込みの デリゲート メソッドを実装します。ほとんどの場合、このアプローチの方がシンプルで、アプリケーションの残りのコードとの適合性もより高いです。
- Audio Session Servicesで宣言されている、C言語ベースの割り込みコールバック関数を記述します。この選択肢の場合は、明示的にオーディオセッション初期化コールを使ってオーディオセッションにコールバックを登録する必要があります。

どちらのアプローチでも、割り込みコード内で行う処理は、採用しているオーディオテクノロジーとその用途（再生、録音、オーディオフォーマット変換、ストリーミングオーディオパケットの読み取りなど）に大きく依存します。一般に、ユーザの視点から最小限の中断できちんと復旧するようにします。

表 3-1 に、割り込み時に実行する必要がある手順を要約します。 `AVAudioPlayer` オブジェクトまたは `AVAudioRecorder` オブジェクトを使うときには、これらの手順のいくつかはシステムによって自動的に処理されます。

表 3-1　オーディオセッション割り込み時に必要な手順

割り込みの開始後	• オーディオプロセスの再開がサポートされているかを確認する • 状態とコンテキストを保存する • ユーザインターフェイスを更新する
割り込みの終了後	• 状態とコンテキストを復元する • オーディオセッションを再度アクティブにする • ユーザインターフェイスを更新する

表 3-2 に、テクノロジーに応じたオーディオ割り込みの処理方法を簡単にまとめます。この章の残りのセクションで詳細を説明します。

表 3-2　オーディオテクノロジーに応じた、オーディオ割り込みの処理方法

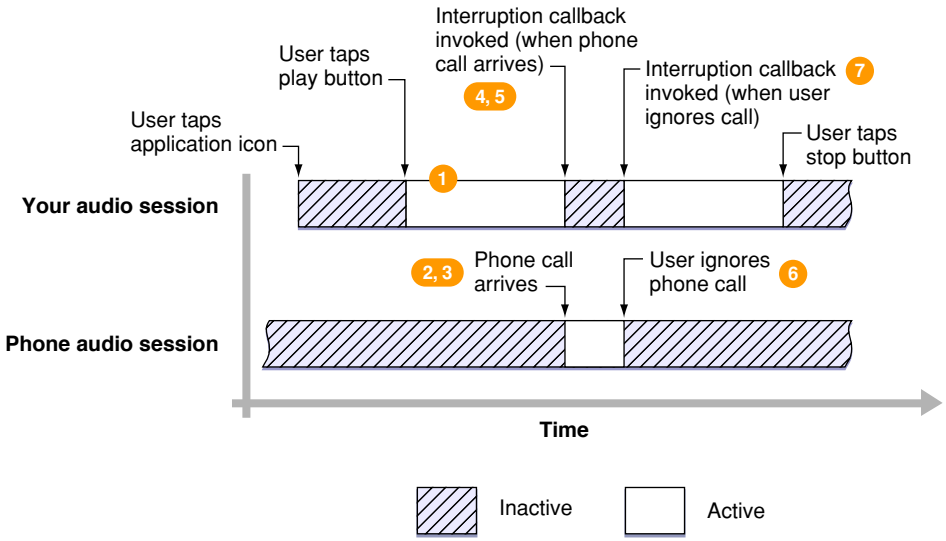
オーディオテクノロジー	割り込み時の動作
AV Foundationフレームワーク	`AVAudioPlayer` クラスと `AVAudioRecorder` クラスは、割り込みの開始と終了に関するデリゲートメソッドを備えています。ユーザインターフェイスを更新し、割り込み終了後、必要であれば一時停止していた再生を再開するためにこれらのメソッドを実装します。割り込み時に、システムは自動的に再生や録音を一時停止し、その後再生や録音が再開されるとオーディオセッションを再度アクティブにします。 アプリケーションを次回起動するまでの間、再生位置を保存して復元したい場合は、割り込み時だけでなく、アプリケーションを終了するときにも再生位置を保存します。
Audio Queue Services、I/Oオーディオユニット	これらを使うと、割り込み時の処理をアプリケーションが制御できます。再生や録音の位置を保存し、割り込み終了後にオーディオセッションを再度アクティブにする処理は、デベロッパが面倒を見なければなりません。「オーディオセッション割り込みの対応」（54 ページ）で説明するように、`AVAudioSession` 割り込みデリゲートメソッドを実装するか、割り込みリスナーコールバック関数を作成します。 ハードウェアによるAACへのエンコーディングの技術を使っている場合は、「ハードウェアコーデックとオーディオ割り込み」（32 ページ）を参照してください。
OpenAL	再生にOpenALを使う場合は、Audio Queue Servicesを使う場合と同様、`AVAudioSession` 割り込みデリゲートメソッドを実装するか、割り込みリスナーコールバック関数を作成します。しかし、デリゲートやコールバックではさらに、OpenALのコンテキストを管理する必要があります。「OpenALとオーディオ割り込み」（32 ページ）を参照してください。

オーディオテクノロジー	割り込み時の動作
System Sound Services	割り込みが開始すると、System Sound Servicesを使って再生されるサウンドは鳴らなくなります。割り込みが終了すると、これらのサウンドは再度自動的に使用可能な状態に戻ります。この再生テクノロジーを使うサウンドについて、割り込み時の動作をアプリケーションが変更することはできません。

割り込みのライフサイクル

図 3-1 は、再生アプリケーションにオーディオセッションの割り込みが発生する前、発生している最中および終了後の一連の流れを図示したものです。

図 3-1 オーディオセッションに割り込みが発生するときの動作



割り込みイベント（この例では電話の着信）は次のように進行します。番号付きのステップは、図中の番号に対応しています。

1. アプリケーションはアクティブで、オーディオを再生している状態です。
2. 電話が着信しました。システムは電話アプリケーションのオーディオセッションをアクティブにします。
3. システムはオーディオセッションを非アクティブにします。この時点で、アプリケーションが実行している再生の処理は停止します。
4. システムは割り込みリスナーコールバック関数（または割り込みを開始したデリゲートメソッド）を呼び出し、アプリケーションのオーディオセッションが非アクティブにされたことを示します。

5. コールバック（またはデリゲートメソッド）が適切に対応します。たとえば、ユーザインターフェイスを更新して、停止した位置から再生を再開するために必要な情報を保存することなどが考えられます。

6. ユーザが割り込みを終了させた場合（ここでは、着信した電話に出ないことをユーザが選択した）、システムはコールバックまたはデリゲートメソッドを呼び出して、割り込みが終了したことを示します。

7. コールバックまたはデリゲートメソッドが、割り込みの終了に対応して適切な処理を行います。たとえば、ユーザインターフェイスを更新し、オーディオセッションを再びアクティブにし、再生を再開することなどが考えられます。

8. （図には示されていません）ステップ6でユーザが割り込みを無視せずに電話に出た場合は、アプリケーションはそこで終了しなければなりません。このように、割り込みライフサイクルは2種類いずれかの形をとって終了します。

デリゲートメソッドを使った割り込み処理

`AVAudioSession` クラスは、割り込みに対応する デリゲート メソッドを提供します（*AVAudioSessionDelegate Protocol Reference*を参照）。これらのメソッドを実装および使用して、使用しているオーディオテクノロジーに関係なく割り込みに対応できます。

- `beginInterruption`。オーディオセッションが割り込まれた後に呼び出されます。このメソッドは、アプリケーションが終了される直前（たとえば、ユーザが電話に出るときなど）であると想定して実装します。たとえば、アプリケーションを次回起動したときに、割り込みが発生したところから再生を再開できるようにするには、ファイル識別子と現在のファイルフレーム番号を保存します。

- `endInterruption`。オーディオセッションの割り込みが終了した後に呼び出されます。オーディオセッションを自動的に再びアクティブにする `AVAudioPlayer` オブジェクトや `AVAudioRecorder` オブジェクトを使っていない場合は、オーディオセッションを明示的に再びアクティブにする必要があります。

これらのメソッドの中で行う処理は、採用しているオーディオテクノロジーと、それを使って何を行っているかによって異なります。少なくとも、アプリケーションの内部の状態とユーザインターフェイスが一貫性を保ち、 `endInterruption` メッセージを受け取った場合はオーディオセッションが再びアクティブになるようにします。

`AVAudioRecorder` クラスと `AVAudioPlayback` クラスは、以下に示すように、オーディオに対する割り込みに対処するための独自のデリゲートメソッドを提供します。

- `audioPlayerBeginInterruption:`。再生中にオーディオセッションに対して割り込みが発生したときに呼び出されます。

- `audioPlayerEndInterruption:`。再生に対する割り込みが終了したときに呼び出されます。

- `audioRecorderBeginInterruption:`。録音中にオーディオセッションに対する割り込みが発生したときに呼び出されます。

- `audioRecorderEndInterruption:`。録音に対する割り込みが終了したときに呼び出されます。

AVAudioRecorder クラスと AVAudioPlayback クラスは、割り込み時に自動的に一時停止します。録音や再生の位置は、アプリケーションを次回起動するときまで維持する必要がない限り、保存する必要はありません。また、割り込み終了後に再生や録音を再開すると、システムがオーディオセッションを自動的に再びアクティブにします。

AV Foundation割り込みデリゲートメソッドの使いかたを示したサンプルコードについては、「AVAudioPlayerクラスを使った割り込み処理」（55 ページ）を参照してください。

コールバック関数を使った割り込み処理

Objective-Cのデリゲートメソッドの代わりにC言語の割り込みリスナーコールバック関数を使う場合は、Audio Session Servicesに含まれているコールバックプロトタイプの AudioSessionInterruptionListener をベースにします。

システムは、そのコールバックを呼び出す際に2つの値を渡します。1つは、アプリケーションによるセッション初期化時に指定（任意）されたデータへの参照、もう1つは、割り込みの状態を示す定数値です。 リスト 3-1 に AudioSessionInterruptionListener コールバックの宣言を示します。

リスト 3-1　割り込みコールバックの宣言

```
typedef void (*AudioSessionInterruptionListener) (
                void *inClientData,
                UInt32 inInterruptionState
            );
```

割り込みの状態は、次に示す2つのいずれかです。

- kAudioSessionBeginInterruption。オーディオセッションに割り込みが発生し、オーディオセッションが非アクティブにされたことを意味します。このメッセージが送られたときには、アプリケーションが終了させられること（割り込みの原因である着信をユーザが受けるなど）を前提にして処理を実行する必要があります。そのため、終了の前に必要なアクションを実行するように割り込みコールバック関数を記述してください。

- kAudioSessionEndInterruption。割り込みが終了したことを意味します。ユーザが着信した電話に出ないことを選択した場合などに、この定数値がコールバックに送られます。コールバックでは、アプリケーションのオーディオセッションを再アクティブ化して再生を再開するか、その他の適切な状態を回復する処理を実行します。ただし、ユーザが電話に出た場合はアプリケーションは終了します。この場合、この定数値を渡してコールバックが呼び出されることはありません。

コールバックを使う際に、アプリケーションに割り込み対応の能力を付加するには、次の3つの作業を行います。

1. 割り込み開始と割り込み終了の各時点で適切な処理を実行するためのメソッドを定義します。それらのアクションの中には、いずれにせよアプリケーションのどこかで必ず実行しなくてはならない処理（録音の停止など）と、コールバックが呼び出されたとき以外には必ずしも実行しなくていい処理（再生の一時停止など）があります。「割り込みメソッドの定義」（56 ページ）を参照してください。

2. 割り込みの開始時と終了時に呼び出されるコールバック関数を定義します。コールバック関数では、上のメソッドを必要に応じて呼び出すようにします。「割り込みリスナーコールバック関数の定義」（57 ページ）を参照してください。

3. コールバック関数を、アプリケーションのオーディオセッションに登録します。これはセッションの初期化の一環として行います。「オーディオセッションの初期化」（47 ページ）を参照してください。

OpenALとオーディオ割り込み

オーディオの再生にOpenALを使う場合は、Audio Queue Servicesを使う場合と同様、`AVAudioSession` AVAudioSessionデリゲートメソッドを実装するか、割り込みリスナーコールバック関数を作成します。しかし、作成する割り込みコードではさらに、OpenALのコンテキストを管理する必要があります。割り込み時に、OpenALコンテキストを `NULL`に設定します。割り込みが終了したら、コンテキストを前の状態に設定します。

iOS 3.0よりも前のバージョンでは、OpenAL割り込みの処理がもっと多く関与していました。割り込みが発生したら、OpenALコンテキストを保存してから破壊し、その後割り込みが終了したらコンテキストを再度作成する必要がありました。現在のコードに古いバージョンのiOSとの互換性を持たせるために、実行時にOSのバージョンを判定することによってコードを条件分岐させることができます。

オーディオ割り込みの際にOpenALコンテキストを管理する方法を示すサンプルコードについては、「OpenAL使用時の割り込みの対応」（59 ページ）を参照してください。

ハードウェアコーデックとオーディオ割り込み

この章のこれまでのセクションでは、最も一般的なオーディオ操作である再生と録音に対する割り込みに対処する方法を説明しました。割り込み処理を必要とする別の操作として、オフラインオーディオフォーマット変換があります。この場合は、必要な作業が若干増えます。具体的には、オーディオデータバッファレベルで割り込みを処理する必要があります。

特定のデバイス上では、バックグラウンド処理を通じて、ハードウェアコーデックを使ってリニアPCMオーディオをAACフォーマットにエンコードすることができます。コーデックは、iPhone 3GS以降とiPod touch（第2世代）以降で利用できます。 コーデックは、（型が `AudioConverterRef`の）オーディオ変換オブジェクトの一部として使用します。これは、（型が`ExtAudioFileRef`の）拡張オーディオファイルオブジェクトの一部です。これらの不透過型の詳細については、*Audio Converter Services Reference* および *Extended Audio File Services Reference*を参照してください。

ハードウェアエンコード処理中に割り込みに対処するには、次の2点を考慮する必要があります。

1. コーデックは、割り込み終了後にエンコードを再開できる場合とできない場合がある。

2. 割り込み前にコーデックに渡した最後のバッファは、ディスクに正常に書き出される場合と書き出されない場合がある。

エンコードは、 `ExtAudioFileWrite` 関数を繰り返し呼び出して、オーディオデータの新しいバッファを渡すたびに実行されます。割り込みを処理するには、以下に示すように、関数の結果コードに応じて対応します。

- `kExtAudioFileError_CodecUnavailableInputConsumed`—この結果コードは、割り込み前に渡した最後のバッファが正常にディスクに書き出されたことを示します。

  この結果コードを受け取ったら、`ExtAudioFileWrite` 関数の呼び出しをやめます。変換を再開する可能性がある場合は、割り込み終了の通知を受け取るまで待ちます。割り込み終了ハンドラ内で、セッションを再度アクティブにしてからファイルへの書き込みを再開します。

  割り込み前の最後のバッファはディスクに正常に書き出されたので、次のバッファへの書き込みに進みます。

- `kExtAudioFileError_CodecUnavailableInputNotConsumed`— この結果コードは、割り込み前に渡した最後のバッファが正常にディスクに書き出され *なかった* ことを示します。

  前述の結果コードの場合とまったく同じように、この結果コードを受け取ったら、`ExtAudioFileWrite` 関数の呼び出しをやめます。変換を再開する可能性がある場合は、割り込み終了の通知を受け取るまで待ちます。割り込み終了ハンドラ内で、セッションを再度アクティブにしてからファイルへの書き込みを再開します。

  前述のもう1つの結果コードの場合と割り込み処理が異なる点は、割り込み前の最後のバッファはディスクに正常に *書き出されなかった*ので、再度、同じバッファへの書き込みから始める必要がある点です。

AACコーデックが再開可能かを確認するには、対応するコンバータの `kAudioConverterPropertyCanResumeFromInterruption` プロパティの値を取得します。このプロパティの値は1（再開可能）または0（再開不可能）のどちらかです。この値は、コンバータをインスタンス化した後であればいつでも（インスタンス化の直後、割り込み発生時、割り込み終了後などに）取得できます。

コンバータを再開できない場合は、割り込み発生時に変換処理をあきらめなければなりません。割り込み終了後、またはユーザが変換を再開することを目的としてアプリケーションを再度立ち上げたら、拡張オーディオファイルオブジェクトを再度インスタンス化して、変換を再度行います。

オーディオハードウェア経路の変化

アプリケーションが実行している間、ユーザはヘッドセットを抜き差ししたり、オーディオ接続のあるドッキングステーションを使ったりすることが考えられます。 *iOS Human Interface Guidelines* では、このようなイベントにiPhoneアプリケーションがどのように対応すべきかについて解説しています。同書で示している推奨事項を実装するには、オーディオハードウェア経路の変化に対処するオーディオセッションコードを作成します。

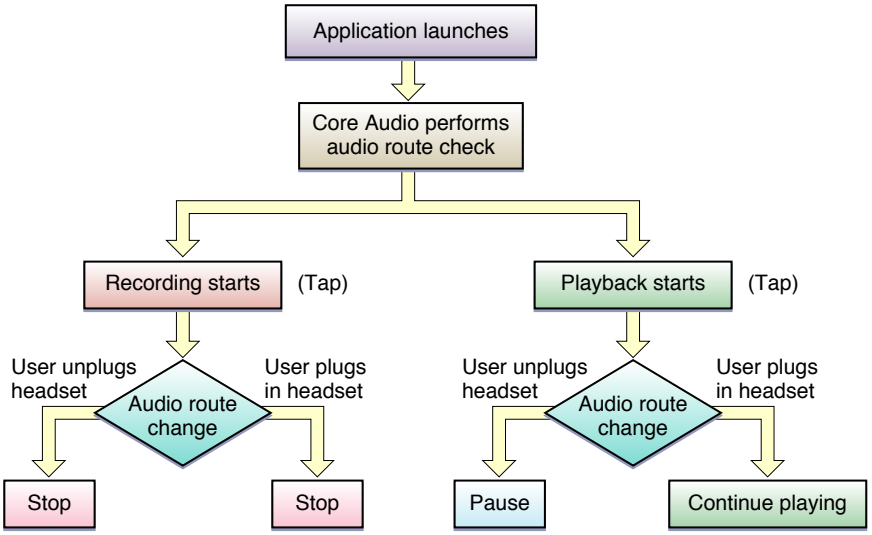
注： 経路の変化に対処するには、CベースのAudio Session Servicesインターフェイスを採用してください。それ以外のオーディオセッションコードにObjective-CベースのAV Foundationフレームワークを採用していても構いません。

さまざまなオーディオハードウェア経路の変化

オーディオハードウェア経路 は、オーディオ信号を電気的に伝えるための有線の通り道です。iOSデバイスのユーザがヘッドセットを抜き差しすると、システムはオーディオハードウェア経路を自動的に変更します。アプリケーションでは、オーディオセッションプロパティの仕組みを使ってこのような変化を検知することができます。

図 4-1 は、録音や再生の実行中に生じるさまざまな経路の変化に対する一連のイベントを示しています。作成したプロパティリスナーコールバック関数で実行される処理の内容に応じて、図の下辺に示す4通りの結果が考えられます。

図 4-1 オーディオハードウェア経路の変化に対する処理



図では、アプリケーションが起動した後、まずシステムがオーディオ経路を判定します。それ以降も、アプリケーションが動作する間、Core Audioはアクティブな経路の監視を続けます。たとえば、アプリケーションの「録音(Record)」ボタンをユーザがタップした場合を考えてみましょう。図では、左側にある「録音を開始する(Recording starts)」の四角がこれに該当します。

録音の実行中、ユーザはヘッドセットを抜き差しする可能性があります。図中の左下付近にある菱形の図形を参照してください。システムは変化に反応して、アプリケーションのプロパティリスナーコールバック関数を呼び出します。Appleの推奨事項に従うには、このときコールバックはアプリケーションに対して録音を停止するように伝える必要があります。

再生の場合も録音と同様の流れになりますが、図の右側に示すとおり結果が異なります。再生中にユーザがヘッドセットを抜いた場合は、コールバックはオーディオを一時停止する必要があります。また、ユーザがヘッドセットを接続した場合には、コールバックでは単に再生を続行できるようにする必要があります。

サンプルコードプロジェクト *AddMusic* は、この動作のうち再生部分を実装する方法を示します。

オーディオハードウェア経路の変化への対応

経路の変化に対応できるようにアプリケーションを設定するには、次の3つの作業を行います。

1. 経路が変化したときに呼び出されるメソッドを実装する。
2. 経路の変化に対応し、手順1で実装したメソッドを使用するプロパティリスナーコールバック関数を実装する。
3. 作成したコールバック関数をオーディオセッションに登録する。

たとえば、ユーザがヘッドセットを抜いた場合に再生を一時停止するようにコールバックを（Appleのガイドラインに従って）作成したとします。一時停止した後、コールバックは、アラートと共に停止か再開をユーザが指定できるボタンを表示できます。ユーザとのこのようなやり取りをサポートするために、アラートを表示するメソッドを作成し、コールバックの中からそのメソッドを呼び出すことができます。アプリケーションの起動時にオーディオセッションを初期化する際に、このコールバックを登録できます。

システムは、経路の変化に対処するコールバックを呼び出すときに、取るべきアクションを判断するために必要となる情報を提供します。作成するコールバックは、以下に示すAudio Session Servicesに含まれている `AudioSessionPropertyListener` プロトタイプをベースにしてください。

```
void MyPropertyListener (
   void *inClientData,
   AudioSessionPropertyID inID,
   UInt32 inDataSize,
   const void *inData
);
```

経路変化のイベントを受けて、システムは `kAudioSessionProperty_AudioRouteChange` を *`inID`* パラメータで渡します。

コールバックに渡される *`inData`* パラメータには、次の情報を記述した `CFDictionaryRef` オブジェクトが含まれています。

- 経路が変化した理由
- 以前の経路

辞書のキーは、 Audio Route Change Dictionary Keysに説明があります。ハードウェアのオーディオ経路が変化する可能性のあるさまざまな原因（kAudioSession_AudioRouteChangeKey_Reason キーでアクセスできます）は、 Audio Session Route Change Reasonsにその一覧と説明があります。以前の経路の情報（kAudioSession_AudioRouteChangeKey_OldRoute キーでアクセスできます）は、「Headphone」や「Speaker」といった、以前の経路を示す文字列値です。

コールバックに渡された情報は、通常はこれで十分であり、この情報で実行すべきアクションを決めることができます。たとえば、経路の変化の原因が kAudioSessionRouteChangeReason_OldDeviceUnavailable で、以前の経路が Headphoneであった場合、ユーザがヘッドセットを単に抜いたためだということが分かります。また、新しい経路が何であるかを知る必要がある場合は、コールバック内で AudioSessionGetProperty 関数を呼び出して kAudioSessionProperty_AudioRoute プロパティを問い合わせます。

iOSにおけるオーディオハードウェア経路の変化の原因の1つとして、kAudioSessionRouteChangeReason_CategoryChangeがあります。つまり、この場合のオーディオセッションカテゴリ内の変化は、システムによって経路の変化と見なされ、経路変化プロパティリスナーコールバックが呼び出されます。したがってこのようなコールバックは、ヘッドセットの抜き差しだけに対応することを目的としている場合は、この種の経路の変化を明示的に無視しなければなりません。「プロパティリスナーコールバック関数の定義」（61 ページ）のサンプルコードにこれを示します。

次に、実用的な例を示します。録音／再生アプリケーションでは、録音または再生の開始直前にオーディオセッションカテゴリを設定するのが正しい作法です。そのため、経路変化プロパティリスナーコールバックは、この再生開始（その前の処理が録音だった場合）または録音開始（その前の処理が再生だった場合）の際に呼び出されます。このようなアプリケーションは、録音ボタンや再生ボタンがタップされるたびに一時停止したり停止したりするべきではありません。不適切な一時停止や停止を避けるために、この例のコールバックは、経路変化の原因に応じて条件分岐し、原因がカテゴリの変化だった場合は単に復帰するべきです。

経路変化への対処方法の完全な例については、「オーディオハードウェア経路の変化への対応」（60 ページ）（「オーディオセッションの活用」）を参照してください。このようなコールバック動作を見るには、 *AddMusic* をダウンロードしてください。

デバイスハードウェアに合わせた最適化

オーディオセッションプロパティを使って、デバイスハードウェアに合わせてアプリケーションのオーディオ動作を実行時に最適化できます。コードを実行するデバイスの特性や、アプリケーションの実行時にユーザによってなされる変更（ヘッドセットの接続やデバイスのドッキングなど）にコードを適合させることができます。

オーディオセッションプロパティの仕組みを使って、以下のことが可能です。

- サンプルレートとI/Oバッファ時間に対して最適なハードウェア設定を指定する
- さまざまなハードウェア特性を問い合わせる。たとえば、入出力の遅延時間、入出力のチャネル数、ハードウェアのサンプルレート、ハードウェアの音量設定、オーディオ入力を利用できるかどうかなど
- プロパティ値の変更イベントに対処するコールバック関数を作成する

最もよく使われるプロパティ値変更イベントとして、前章で取り上げた経路の変化があります。また、ハードウェア出力音量の変化や、（マイクが内蔵されていないデバイスに対応するために）オーディオ入力の利用可否の変化を検知するコールバックを作成することもできます。

最適なオーディオハードウェア設定の指定

オーディオセッションAPIを使って、ハードウェアのサンプルレートとI/Oバッファ時間に対して最適な設定を指定できます。 表 5-1 に、このような設定を指定する場合に検討するべき得失を説明します。

表 5-1　最適なハードウェア設定の選択

設定	最適なサンプルレート	最適なI/Oバッファ時間
高い値	*例：44.1 kHz* + オーディオ品質が高い – ファイルサイズが大きい	*例：500 ミリ秒* + ディスクアクセスの頻度が低い – 遅延時間が長い
低い値	*例：8 kHz* + ファイルサイズが小さい – オーディオ品質が低い	*例：5 ミリ秒* + 遅延時間が短い – ディスクアクセスの頻度が高い

たとえば、表の中央最上部のセルに示すように、アプリケーションでオーディオ品質が非常に重要で、ファイルサイズの大きさがそれほど問題にならない場合は、サンプルレートに高い値を指定することが考えられます。

デフォルトのオーディオI/Oバッファ時間（44.1 kHzのオーディオで約0.02秒）は、ほとんどのアプリケーションに適しています。テンポの速いゲームも含め、ユーザとのやり取りなどに対する応答性はこれで十分です。VoIP（Voice over IP）のようなアプリケーションで要求されない限り、I/O時間にはこれよりも低い値を設定しないでください。

ハードウェア設定を指定するには、`AudioSessionSetProperty` 関数とプロパティ識別子の `kAudioSessionProperty_PreferredHardwareSampleRate` と `kAudioSessionProperty_PreferredHardwareIOBufferDuration` を使います。サンプルコードについては、「最適なハードウェアI/Oバッファ時間の指定」（64 ページ）を参照してください。

重要： ハードウェアの最適値は、オーディオセッションを初期化した後であればいつでも安全に指定できますが、セッションが非アクティブのときに行うのが最適です。ハードウェアの最適値を設定した後は、オーディオセッションをアクティブにしてから、オーディオセッションに問い合わせをして実際の値を取得します。システムが要求に応えられない場合もあるため、この最後の手順は重要です。

ハードウェア特性の問い合わせ

アプリケーションのオーディオセッションには、デバイスが備えるハードウェア特性の多くについて情報を取得する機能があります。ハードウェア特性はアプリケーションの実行中にも変化する可能性があります。たとえば、ユーザがヘッドセットを接続すると入力のサンプルレートが変わることがあります。このようないくつかの特性に対してはコールバックを使用できます。その他の特性に対しては、値を直接問い合わせます。

重要： ハードウェア特性に関する意味のある値を取得するために、問い合せを発行する前に確実にオーディオセッションの初期化とアクティブ化を行ってください。

オーディオセッションの各種ハードウェア特性のうち、特に役立つものを 表 5-2に示します。

表 5-2　有用なオーディオセッションハードウェア特性の一部

オーディオセッションプロパティ	説明
`kAudioSessionProperty_-CurrentHardwareSampleRate`	デバイスのハードウェアサンプルレート。オーディオの録音形式を設定する際にこのプロパティを使います（「ハードウェアサンプルレートの取得と使用」（64 ページ）を参照）。 このプロパティに対してはコールバックの呼び出しは使用できません。
`kAudioSessionProperty_-CurrentHardwareOutputVolume`	デバイスの再生音量。
`kAudioSessionProperty_-CurrentHardwareOutputLatency`	デバイスの再生遅延。 このプロパティに対してはコールバックの呼び出しは使用できません。

オーディオセッションプロパティ	説明
kAudioSessionProperty_-AudioInputAvailable	デバイスのオーディオ入力を利用できるかどうか。オーディオの録音が可能かどうかを判断する際にこのプロパティを使います。サンプルコードについては、「デバイスで録音がサポートされているかどうかを調べる方法」（65 ページ）を参照してください。

オーディオセッションプロパティの完全なリストは、*Audio Session Services Reference*の Audio Session Services Property Identifiers に示されています。

プロパティ変更イベントへの対応

前章（「オーディオハードウェア経路の変化」（35 ページ））では、特定のプロパティ値変更イベント、すなわちオーディオハードウェア経路の変化への対応について深く掘り下げました。「オーディオセッションの活用」の章には「オーディオハードウェア経路の変化への対応」（60 ページ）という完全なサンプルを掲載しています。その他のイベントに対しても、同じ関数コールバックの手法を使って対応できます。特に、以下のようなイベントに対応できます。

- ユーザによって行われた、ハードウェアの再生音量に対する変更
- オーディオ入力を利用できるかどうかに対する変化。これは、iPod touch（第2世代）のように、ヘッドセットが接続されているときに限り録音が可能なデバイス上でのマイクの追加や削除を意味します。

再生音量の変化に対応するには、kAudioSessionProperty_CurrentHardwareOutputVolume プロパティを検知するプロパティリスナーコールバック関数を作成します。オーディオ入力を利用できるかどうかに対する変化については、kAudioSessionProperty_AudioInputAvailable プロパティの変更を検知するようにします。

ムービーおよびiPodミュージックの使用

ムービープレーヤーを使って、ファイルまたはネットワークストリームからムービーを再生できます。ミュージックプレーヤーを使うと、ユーザのiPodライブラリにあるオーディオコンテンツを再生できます。これらのオブジェクトをアプリケーションのオーディオと連動させて使うには、オーディオセッションの特性を考慮に入れる必要があります。

- ミュージックプレーヤー（`MPMusicPlayerController` クラスのインスタンス）は、常にシステムが提供するオーディオセッションを使う
- ムービープレーヤー（`MPMoviePlayerController` クラスのインスタンス）は、デフォルトではアプリケーションのオーディオセッションを使うが、システムが提供するオーディオセッションを使うように設定することもできる

重要： iOS 3.1.3およびそれ以前のバージョンでは、ムービープレーヤーは常にシステムが提供するオーディオセッションを使います。iOS 3.2およびそれ以降のバージョンでも同じように動作させるには、ムービープレーヤーの `useApplicationAudioSession` プロパティ値を `NO`に設定する必要があります。

詳細については、「ムービープレーヤーの使用」（44 ページ）を参照してください。

ミュージックプレーヤーの使用

アプリケーション独自のサウンドと一緒にユーザのiPodライブラリにあるオーディオを再生するには（*iPod Library Access Programming Guide*を参照）、アプリケーションのオーディオセッションに対して、いわゆる*ミックス可能*カテゴリ設定を使う必要があります。オーディオセッションをミックス可能として設定する方法は2通りあります。

- 常にミックス可能な、`AVAudioSessionCategoryAmbient`（またはこれと同等の `kAudioSessionCategory_AmbientSound`）カテゴリを使う
- ミックス可能カテゴリのオーバーライドプロパティである `kAudioSessionProperty_OverrideCategoryMixWithOthers`を使い、ミックス不可能なその他の再生カテゴリをミックス可能にする（「カテゴリの微調整」（25 ページ）を参照）

上記の選択肢の1つを使えば、アプリケーションのサウンドがミュージックプレーヤーを中断させることも、ミュージックプレーヤーのサウンドがアプリケーションのサウンドを中断させることもありません。

重要： アプリケーションのオーディオセッションに対してミックス可能カテゴリを設定しないままミュージックプレーヤーを使うことは避けてください。

ミックス可能カテゴリ設定を使う必要があるため、再生や録音のためにハードウェアコーデックにはアクセスできません。詳細については、「エンコードとデコードへのカテゴリの影響」（24 ページ）を参照してください。

システムは、ミュージックプレーヤーに対するハードウェア経路の変化、およびオーディオ割り込みを自動的に処理します。組み込まれているこの動作に影響を及ぼすことはできません。アプリケーションのオーディオセッションを、この章やこれまでの章で述べたとおり適切に管理していれば、ユーザがヘッドセットを接続したり、アラームが鳴ったり、電話がかかってきたりしたときの対応をミュージックプレーヤーに任せることができます。

アプリケーションのオーディオの再生時に、ミュージックプレーヤーからの音を小さくする（音量を下げる）ようにオーディオセッションを設定できます。音量を下げる機能と、これを有効にする方法については、「カテゴリの微調整」（25 ページ）を参照してください。

ミュージックプレーヤークラスの詳細については、*MPMusicPlayerController Class Reference*を参照してください。

ムービープレーヤーの使用

デフォルトでは、ムービープレーヤーはアプリケーションのオーディオセッションを共有します。つまり、ムービープレーヤーは、実際にはアプリケーションのオーディオとのミックスという概念にとどまらず、ムービープレーヤーのオーディオはアプリケーションに属しているかのように動作します。選択する再生カテゴリや、そのカテゴリの設定方法に関わらず、アプリケーションのオーディオとムービープレーヤーのオーディオはどちらも互いを中断させることはありません。

オーディオセッションを共有することで、ムービーとほかのアプリケーション（iPodなど）からのオーディオとの相互作用の方法も制御できるようになっています。たとえば、カテゴリを`AVAudioSessionCategoryAmbient`に設定してセッションを共有すると、アプリケーションでムービーが開始してもiPodのオーディオは中断しません。また、オーディオセッションを共有すると、ムービーオーディオが着信／サイレントスイッチに従うかどうかを指定することもできます。

重要： iOS 3.1.3およびそれ以前のバージョンでは、ムービープレーヤーは常にシステムが提供するオーディオセッションを使います。iOS 3.2およびそれ以降のバージョンでも同じように動作させるには、ムービープレーヤーの`useApplicationAudioSession`プロパティ値を`NO`にする必要があります（表 6-1（45 ページ）の1行目を参照）。

ムービー用にオーディオの動作を設定するためには、どのような動作にしたいかを決め、適切にオーディオセッションの設定をする必要があります（表 6-1を参照）。オーディオセッションの設定に関する詳細については、「オーディオセッションの設定」（21 ページ）を参照してください。

表 6-1 ムービープレーヤー使用時のオーディオセッションの設定

必要な動作	オーディオセッションの設定
ムービーの再生により、その他のオーディオをすべて止める	● アプリケーション自身がオーディオの再生を行わない場合は、オーディオセッションは設定しない。 ● アプリケーションがオーディオの再生を行う場合、iPodなどのオーディオとのミックスが必要かどうかに応じて、オーディオセッションをミックス可能または不可能として設定する。 ● どちらの場合も、ムービープレーヤーに専用のオーディオセッションを使用するように指定する： `myMoviePlayer.useApplicationAudioSession = NO`
ムービーとアプリケーションのオーディオをミックスするが、iPodなどのほかのオーディオは止める	● *ミックス不可能*カテゴリを使ってオーディオセッションを設定する。 ● ムービープレーヤーの`useApplicationAudioSession`の値としてデフォルトの`YES`を使う。
すべてのオーディオをミックスする	● *ミックス可能*カテゴリ設定を使ってオーディオセッションを設定する。 ● ムービープレーヤーの`useApplicationAudioSession`の値としてデフォルトの`YES`を使う。

アプリケーションのオーディオセッションは、「オーディオハードウェア経路の変化」（35 ページ）および「オーディオ割り込みの処理方法」（27 ページ）で述べたように、通常どおり経路の変化と割り込みの点から管理してください。必要であれば、「カテゴリの微調整」（25 ページ）で説明したとおり、音量を下げる機能を有効化してください。

独自のオーディオセッションを使うようにムービープレーヤーを設定した場合は、いくつかの後処理を行う必要があります。ムービーの終了後、またはユーザがムービーを止めたら、以下の2つの手順を順番どおりに実行し、オーディオを再生可能な状態に戻します。

1. ムービープレーヤーを破棄する（同じムービーを後で再び再生する予定がある場合でも）。
2. アプリケーションのオーディオセッションを再度アクティブにする。

ムービープレーヤークラスの解説については、*MPMoviePlayerController Class Reference*を参照してください。

メディアプレーヤーフレームワークのみを使用する場合

アプリケーションがムービープレーヤーのみ、またはミュージックプレーヤーのみを使い、なおかつアプリケーション独自のサウンドは再生しない場合は、オーディオセッションを設定しないでください。

ムービープレーヤーのみを使用する場合は、次のように指定して、ムービープレーヤー自身のオーディオセッションを使うように伝える必要があります。

```
myMoviePlayer.useApplicationAudioSession = NO
```

ムービープレーヤー と ミュージックプレーヤーの両方を使っている場合は、おそらくこの2つの相互作用の方法を設定するのが望ましいでしょう。そのため、アプリケーションのオーディオ自体は再生しないという場合でも、オーディオセッションを設定する必要があります。 表 6-1 （45 ページ）のガイダンスを参考にしてください。

オーディオセッションの活用

この章の各セクションには、オーディオセッションを使って行われる一般的なタスクの実装コード例を示します。オーディオセッションに関するコードのほとんどは View Controllerクラスに配置するのが一般的であり、 *AddMusic* のサンプルもそのように作られています。

これらのタスクの相互関係やアプリケーションライフサイクルとの関係を理解するには、 「オーディオセッションの設定」 （21 ページ）を参照してください。

製品版コードにおいては、APIを呼び出したら必ずエラーがないかチェックします。サンプルについては、 リスト 7-4 （48 ページ）を参照してください。この章のほとんどのサンプルではエラーチェックを示していません。

オーディオセッションの初期化

AV Foundationデリゲートメソッドを使ってオーディオ割り込みを処理する場合は、 「オーディオセッションの初期化」 （21 ページ）で説明したように、オーディオセッションの暗黙的な初期化で対応できます。

あるいは、オーディオ割り込みを処理するCのコールバック関数を作成することもできます。作成したコードをオーディオセッションに対応付けるには、 `AudioSessionInitialize` 関数を使ってセッションを明示的に初期化する必要があります。初期化は、アプリケーションの起動時に1回だけ実行します。

> **注：** アプリケーションでは、オーディオセッションのインスタンス化操作は実行しません。オーディオセッションオブジェクトはiOSから与えられるシングルトンオブジェクトであり、アプリケーションの起動が完了した時点で利用可能な状態になっています。

リスト 7-1 は、アプリケーションでオーディオセッションの初期化と割り込みリスナーコールバック関数の登録を実行する方法を示します。

リスト 7-1 セッションの初期化と割り込みコールバックの登録

```
AudioSessionInitialize (
    NULL, // 1
    NULL, // 2
    interruptionListenerCallback, // 3
    userData // 4
);
```

このコードは次のように動作します。

1. `NULL` は、デフォルト（メイン）実行ループを使うことを示します。
2. `NULL` は、デフォルトの実行ループモードを使うことを示します。

3. 割り込みリスナーコールバック関数への参照です。割り込みコールバック関数を記述し、使用する方法については、「オーディオセッション割り込みの対応」（54 ページ）を参照してください。

4. 割り込みリスナーコールバック関数に渡すためのデータです。オーディオセッションからコールバックが呼び出される際にこのデータが渡されます。

オーディオセッションのアクティブ化と非アクティブ化

リスト 7-2に示すように、オーディオセッションをアクティブにします。

リスト 7-2　AV Foundationフレームワークを使ったオーディオセッションのアクティブ化

```
NSError *activationError = nil;
[[AVAudioSession sharedInstance] setActive: YES error: &activationError];
```

オーディオセッションを非アクティブにする必要がある場合は、`NO` を *`setActive`* パラメータとして渡します。

CベースのAudio Session Servicesインターフェイスを使いたい場合は、`AudioSessionSetActive` 関数を呼び出してオーディオセッションをアクティブにします。リスト 7-3 に呼び出し方法を示します。

リスト 7-3　Audio Session Servicesを使ったオーディオセッションのアクティブ化

```
OSStatus activationResult = NULL;
result = AudioSessionSetActive (true);
```

オーディオセッションを非アクティブにする必要がある場合は、`false` を `AudioSessionSetActive` 関数に渡します。

カテゴリの設定

オーディオセッションのカテゴリを設定するには、`setCategory:error:` メソッドを呼び出します（リスト 7-4を参照）。すべてのカテゴリの詳細については、「オーディオセッションのカテゴリ」（69 ページ）を参照してください。

リスト 7-4　AV Foundationフレームワークを使ったオーディオセッションカテゴリの設定

```
NSError *setCategoryError = nil;
[[AVAudioSession sharedInstance]
                setCategory: AVAudioSessionCategoryAmbient
                      error: &setCategoryError];

if (setCategoryError) { /* エラー状態を処理 */ }
```

リスト 7-5 に、CベースのAudio Session Servicesインターフェイスを使った場合の同等の処理を示します。

リスト 7-5　Audio Session Servicesを使ったオーディオセッションカテゴリの設定

```
UInt32 sessionCategory = kAudioSessionCategory_AmbientSound; // 1

AudioSessionSetProperty (
    kAudioSessionProperty_AudioCategory, // 2
    sizeof (sessionCategory), // 3
    &sessionCategory // 4
);
```

このコードは次のように動作します。

1. UInt32型の新しい変数を定義し、オーディオセッションに適用しようとするカテゴリの識別子を渡して初期化します。
2. 設定対象となるオーディオセッションプロパティの識別子（キー）です。設定するプロパティ値のデータサイズ（バイト単位）です。その他のオーディオセッションプロパティについては、Audio Session Services Property Identifiers列挙を参照してください。
3. オーディオセッションのカテゴリに適用したいオーディオ経路オーバーライドの値です。
4. オーディオセッションに設定したいカテゴリの値です。

アプリケーションの起動時にほかのオーディオが再生中かをチェックする

ユーザがアプリケーションを起動したときに、デバイス上でサウンド再生が行われている可能性があります。アプリケーションがゲームの場合、これは特に重要です。たとえば、iPodで音楽を再生していたり、Safariでオーディオのストリーミングを行っていたりすることなどが考えられます。このような場合、「Sound」 in *iOS Human Interface Guidelines*（*iOS Human Interface Guidelines*）では、ユーザはゲームをしながらほかのオーディオも鳴り続けることを期待していると想定するよう推奨しています。

このような場合、ほかのオーディオとのミックスをサポートするAmbientカテゴリを使い、通常のバックグラウンドサウンドトラックの再生は行いません。

リスト 7-6 に、ほかのオーディオが再生中かどうかをチェックし、典型的なゲームアプリケーション用のカテゴリを適切に設定する方法を示します。オーディオセッションを初期化した後、このチェックを実行します。

リスト 7-6　ほかのオーディオが再生中かをチェックする

```
UInt32 otherAudioIsPlaying; // 1
UInt32 propertySize = sizeof (otherAudioIsPlaying);

AudioSessionGetProperty ( // 2
    kAudioSessionProperty_OtherAudioIsPlaying,
    &propertySize,
    &otherAudioIsPlaying
);

if (otherAudioIsPlaying) { // 3
```

```
    [[AVAudioSession sharedInstance]
                    setCategory: AVAudioSessionCategoryAmbient
                          error: nil];
} else {
    [[AVAudioSession sharedInstance]
                    setCategory: AVAudioSessionCategorySoloAmbient
                          error: nil];
}
```

このコードは次のように動作します。

1. オーディオセッションの kAudioSessionProperty_OtherAudioIsPlaying プロパティの値を格納するための変数を定義します。
2. kAudioSessionProperty_OtherAudioIsPlaying プロパティの値を取得し、これを otherAudioIsPlaying 変数に代入します。
3. アプリケーションの起動時に、ほかのオーディオが再生中の場合は、アプリケーションのオーディオセッションに AVAudioSessionCategoryAmbient カテゴリを割り当てます。ほかのオーディオが再生中でない場合は、 AVAudioSessionCategorySoloAmbient カテゴリを割り当てます。

再生のミックス動作の変更

通常はデバイス上のほかのオーディオを消音する2つのカテゴリを変更して、ミックスをサポートすることができます。対象となるカテゴリは、 AVAudioSessionCategoryPlayback （またはこれと同等の kAudioSessionCategory_MediaPlayback）カテゴリ、および AVAudioSessionCategoryPlayAndRecord （またはこれと同等の kAudioSessionCategory_PlayAndRecord）カテゴリです。ミックスを許可するには、オーディオセッションプロパティを1つ設定します。 リスト 7-7 にこの方法を示します。

リスト 7-7　オーディオのミックス動作のオーバーライド

```
OSStatus propertySetError = 0;
UInt32 allowMixing = true;

propertySetError = AudioSessionSetProperty (
                       kAudioSessionProperty_OverrideCategoryMixWithOthers, //
 1
                       sizeof (allowMixing), // 2
                       &allowMixing // 3
                   );
```

このコードは次のように動作します。

1. 設定対象となるオーディオセッションプロパティの識別子（キー）です。設定するプロパティ値のデータサイズ（バイト単位）です。その他のオーディオセッションプロパティについては、Audio Session Services Property Identifiers列挙を参照してください。
2. オーディオセッションのカテゴリに適用したいオーディオ経路オーバーライドの値です。
3. プロパティに適用する値です。

デフォルトでは、このプロパティの値は `false` （0）です。割り込み時などにオーディオセッションカテゴリが変わると、このプロパティの値は `false`に戻ります。ミックス動作ができるようにするには、このプロパティを再度設定する必要があります。

このプロパティの設定が成功したか失敗したかを必ず確認し、適切に対応してください。動作は、iOSの将来のリリースにおいて変更される可能性があります。

画面のロック中もオーディオを使い続ける方法

画面がロックされた場合や、サイレントスイッチ（iPhoneの場合は *着信／サイレントスイッチ*）が「サイレント」位置に切り替えられた場合にオーディオを鳴らし続けるには、適切なカテゴリをオーディオセッションに割り当てる必要があります。「画面がロックされても再生を続けるようにオーディオユニットを設定する」 （51 ページ）に説明するように、オーディオユニットを再生用に使っている場合は、オーディオユニットを特別に設定する必要もあります。

カテゴリ（iOS 2.2以降）が `AVAudioSessionCategorySoloAmbient` （またはこれと同等の `kAudioSessionCategory_SoloAmbientSound`）のデフォルトのオーディオセッションで対処することはできません。

`AVAudioSessionCategoryPlayback` （またはこれと同等の `kAudioSessionCategory_MediaPlayback`）カテゴリを使います。「カテゴリの設定」 （48 ページ）で説明したとおり、このカテゴリをオーディオセッションに割り当てます。すべてのオーディオセッションカテゴリの一覧とそれらの特性については、「オーディオセッションのカテゴリ」 （69 ページ）を参照してください。

画面がロックされても再生を続けるようにオーディオユニットを設定する

再生用にオーディオユニットを使っており、画面がロックされてもオーディオの再生を継続するようにしたい場合は、いくつか追加の設定を行う必要があります。このセクションでは、その理由を簡単に説明し、設定方法を示します。

- オーディオユニットの遅延がシステムデフォルト値である23ミリ秒（44.1 kHzのオーディオの場合）でアプリケーションにとって適切に動作する場合は、画面がロックされたときに使用可能な、低消費電力による再生を利用できます。これを行うには、「電力消費を最小限に最適化する」 （52 ページ）で説明するように、使用中のオーディオユニット上で設定を行う必要があります。
- オーディオ遅延を短くする必要がある場合は、「低遅延のための最適化」 （53 ページ）で説明するように、オーディオユニット上ではなく、オーディオセッションオブジェクトを使って設定を行う必要があります。

VoIPのような低遅延を必要とする特別なニーズがない限り、Appleはアプリケーションの消費電力を最小に抑えるように最適化することを推奨しています。

電力消費を最小限に最適化する

オーディオ再生の電力消費を最小限に最適化するには、ここで説明するように、使用中のオーディオユニットの `kAudioUnitProperty_MaximumFramesPerSlice` プロパティ値を調整する必要があります。

デフォルトでは、システムはI/Oユニットのレンダリングコールバックを、1,024フレームのスライスサイズで呼び出します（スライスについて理解するには、「slice」 in *Core Audio Glossary*、および `kAudioUnitProperty_MaximumFramesPerSlice` プロパティを参照してください）。これは、44.1 kHzのオーディオでは約23ミリ秒の遅延に相当します。0.02秒の遅延は（特別に低い数字ではありませんが）、メディア再生、インターネットストリーミング、ゲームなど、ほとんどのオーディオでうまくいきます。

画面がロックされると、システムはI/Oユニットのスライスサイズを4,096フレーム（44.1 kHzのオーディオでは約93ミリ秒に相当）まで増やすことで、電力を節約します。つまり、スライスのサイズが大きくなると、システムがI/Oユニットのレンダリングコールバックを呼び出す頻度が少なくなり、電力が節約されます。画面がロックされている間はユーザとのやり取りは行われないため、遅延を長くしても問題になることはありません。I/Oユニットのスライスサイズはシステムが管理します。このためI/Oユニットのスライスサイズの設定を行わないでください。

ただし、システムは、再生中に呼び出されるほかのオーディオユニットの最大スライスサイズのプロパティは調整しません。複数のほかのオーディオユニットからI/Oユニットにオーディオを送る場合、それらのオーディオユニットが画面ロック時の大きなスライスサイズに対応できるようにするのはデベロッパの責任です。これを行わないと、画面のロック時にオーディオが停止したりつっかえたりします。

再生に関わる各オーディオユニット（I/Oユニット以外）に対して、`kAudioUnitProperty_MaximumFramesPerSlice` プロパティの値を4,096に設定します。この設定は、オーディオユニットの初期化前の設定時に行います。オーディオ処理グラフを使っている場合は、グラフ初期化の前のグラフ設定の際に行ってください。

> **重要：** 電力消費を最小限に抑えるために最適化する際は、最適なI/Oハードウェアバッファ時間は設定しないでください。バッファ時間を設定すると、画面ロック時に再生を低消費電力モードに切り替えることができません。

リスト 7-8 に、オーディオユニットの `kAudioUnitProperty_MaximumFramesPerSlice` プロパティを `AudioUnitSetProperty` 関数を使って設定し、低電力モードをサポートする方法を示します。

リスト 7-8　オーディオユニットのmaximum-frames-per-sliceプロパティの設定

```
UInt32 maximumFramesPerSlice = 4096;

AudioUnitSetProperty (
    mixerUnit,
    kAudioUnitProperty_MaximumFramesPerSlice,
    kAudioUnitScope_Global,
    0, // グローバルスコープは常に要素0を使用
    &maximumFramesPerSlice,
    sizeof (maximumFramesPerSlice)
);
```

オーディオユニットの検索、インスタンス化、設定の方法に関する例については、*iPhoneMultichannelMixerTest* のサンプルを参照してください。

低遅延のための最適化

システムで指定されている23ミリ秒のレンダリング遅延（44.1 kHzのオーディオの場合）は、VoIP（voice over IP）のように時間が重視される特定のアプリケーションにとっては長すぎます。遅延を削減するため、最適なI/Oハードウェアバッファ時間を設定します。「デバイスハードウェアに合わせた最適化」（39 ページ）および「最適なハードウェアI/Oバッファ時間の指定」（64 ページ）を参照してください。ただし、I/Oバッファ時間を設定した後は、設定を明示的に変更するまで、その値がシステムによって使われます。つまり、画面ロック時にオーディオ再生は低電力モードにならないということです。

出力オーディオのリダイレクト

「再生と録音」のカテゴリを使っている（つまり入力と出力を両方とも使おうとしている）場合、オーディオは通常はレシーバに出力されますが、次の2つの方法で、このオーディオをiPhoneの下部にあるスピーカーにリダイレクトできます。

1. カテゴリ経路のオーバーライドを使う。オーバーライドは、割り込み時やユーザがヘッドセットの抜き差しによってオーディオ経路を変更した時点でレシーバに戻ります。スピーカーを使い続ける場合は、オーバーライドを再度呼び出す必要があります。

2. デフォルトの出力経路を変更する。オーディオセッションカテゴリを変更しない限り、新しい出力経路が有効なままです。この選択肢はiOS 3.1から使用可能です。

リスト 7-9 に、カテゴリ経路のオーバーライドを実行する方法を示します。このコードを実行する前に、オーディオセッションカテゴリを `AVAudioSessionCategoryRecord`（またはこれと同等の `kAudioSessionCategory_PlayAndRecord`）に設定します（「カテゴリの設定」（48 ページ）を参照）。

リスト 7-9　出力オーディオ経路のオーバーライド

```
UInt32 audioRouteOverride = kAudioSessionOverrideAudioRoute_Speaker; // 1

AudioSessionSetProperty (
    kAudioSessionProperty_OverrideAudioRoute, // 2
    sizeof (audioRouteOverride), // 3
    &audioRouteOverride // 4
);
```

このコードは次のように動作します。

1. 出力のオーディオのリダイレクトに使用できる識別子については、「オーディオセッションのカテゴリ経路のオーバーライド」を参照してください。

2. 設定対象となるオーディオセッションプロパティの識別子（キー）です。

3. 設定するプロパティ値のデータサイズ（バイト単位）です。

4. オーディオセッションのカテゴリに適用したいオーディオ経路オーバーライドの値です。

出力オーディオをリダイレクトしたままの状態にしたい場合は、リスト 7-10に示すように、デフォルトの出力を変更します。ここでも、まずオーディオセッションカテゴリを「再生と録音」に設定します。

リスト 7-10　デフォルトの出力オーディオ経路を変更する

```
UInt32 doChangeDefaultRoute = 1;

AudioSessionSetProperty (
    kAudioSessionProperty_OverrideCategoryDefaultToSpeaker,
    sizeof (doChangeDefaultRoute),
    &doChangeDefaultRoute
);
```

Bluetoothオーディオ入力のサポート

iOS 3.1から、録音のオーディオセッションカテゴリを変更して、ペア登録済みBluetoothデバイスからのオーディオ入力ができるようになっています。 リスト 7-11 にその方法を示します。

リスト 7-11　Bluetooth入力に対応するように録音のカテゴリを変更する

```
// まず、入力が可能なオーディオセッションカテゴリを設定する。
// AVAudioSessionCategoryRecordまたはこれと同等のkAudioSessionCategory_Recordカテゴリ、
// あるいはAVAudioSessionCategoryPlayAndRecordまたはこれと同等の
// kAudioSessionCategory_PlayAndRecordカテゴリを使う。その後、以下のように進める。

UInt32 allowBluetoothInput = 1;

AudioSessionSetProperty (
    kAudioSessionProperty_OverrideCategoryEnableBluetoothInput,
    sizeof (allowBluetoothInput),
    &allowBluetoothInput
);
```

この技法は前述のサンプルコード、「出力オーディオのリダイレクト」（53 ページ）と類似しています。リスト 7-11 （54 ページ）のコードを実行する前に、入力が可能なオーディオセッションカテゴリを設定します。 `AVAudioSessionCategoryRecord` （またはこれと同等の `kAudioSessionCategory_RecordAudio`）カテゴリ、および`AVAudioSessionCategoryPlayAndRecord`（またはこれと同等の `kAudioSessionCategory_PlayAndRecord`）カテゴリを使います。

オーディオセッション割り込みの対応

AV Foundationフレームワークを使って割り込みに対応できます。これを行うには、「割り込み開始」と「割り込み終了」のデリゲートメソッドを実装します。あるいは、Cのコールバック関数を使うこともできますが、より複雑になります。コールバックを使う場合は、次の作業を行います。

- 割り込みの発生時に呼び出されるメソッドを定義する
- 割り込みリスナーコールバック関数を定義する

- オーディオセッションに対するコールバック関数を登録する

AVAudioSessionDelegateプロトコルを使ったオーディオ割り込みの処理

このセクションでは、Audio Queue Services、OpenAL、I/Oオーディオユニットを使ってオーディオ関連作業を行う際に適している、 `AVAudioSessionDelegate` プロトコルの使用について説明します。次のセクションでは、オーディオプレーヤー（*AVAudioPlayerDelegate Protocol Reference*参照）とオーディオレコーダー（*AVAudioRecorderDelegate Protocol Reference*参照）が直接提供する割り込みデリゲートメソッドを代わりに使用する方法を説明します。

リスト 7-12 に、 `AVAudioSessionDelegate` プロトコルに含まれている、割り込みの「開始」および「終了」メソッドのシンプルな実装例を示します（*AVAudioSessionDelegate Protocol Reference*を参照）。割り込み「開始」のデリゲートメソッドは、アプリケーションのオーディオが停止され、オーディオセッションが非アクティブにされた後に呼び出されます。

リスト 7-12　AVAudioSessionDelegate割り込みメソッドの実装

```
- (void) beginInterruption {
    if (playing) {
        playing = NO;
        interruptedWhilePlaying = YES;
        [self updateUserInterface];
    }
}

NSError *activationError = nil;
- (void) endInterruption {
    if (interruptedWhilePlaying) {
       [[AVAudioSession sharedInstance] setActive: YES error: &activationError];
        [player play];
        playing = YES;
        interruptedWhilePlaying = NO;
        [self updateUserInterface];
    }
}
```

`beginInterruption` メソッドは、アプリケーションの状態とユーザインターフェイスを更新します。

`endInterruption` メソッドはオーディオセッションを再びアクティブにし、再生を再開し、アプリケーションの状態とユーザインターフェイスを更新します。

AVAudioPlayerクラスを使った割り込み処理

iOS 2.2から使用可能になった `AVAudioPlayer` クラスは、独自の割り込みデリゲートメソッドを提供します（*AVAudioPlayerDelegate Protocol Reference*を参照）。iOS 3.0から使用可能になった `AVAudioRecorder` クラスは、独自の割り込みデリゲートメソッドを提供します（*AVAudioRecorderDelegate Protocol Reference*を参照）。このセクションでは `AVAudioPlayer` クラスを使って割り込みを処理する方法を説明します。 `AVAudioRecorder` クラスの場合と非常によく似たアプローチを使います。

その1つ、割り込み開始のデリゲートメソッドは、アプリケーションのオーディオプレーヤーが一時停止になり、オーディオセッションが非アクティブ化された後に呼び出されます。リスト 7-13に示すような実装を用意できます。

リスト 7-13　オーディオプレーヤーの割り込み開始のデリゲートメソッド

```
- (void) audioPlayerBeginInterruption: (AVAudioPlayer *) player {
    if (playing) {
        playing = NO;
        interruptedOnPlayback = YES;
        [self updateUserInterface];
    }
}
```

このメソッドは、割り込み発生時にオーディオプレーヤーが実際に再生していたかをチェックした後、アプリケーションの状態を更新し、さらにユーザインターフェイスを更新します（システムは自動的にオーディオプレーヤーを一時停止します）。

着信した電話をユーザが受けない場合は、システムによってアプリケーションのオーディオセッションが自動的に再度アクティブにされ、割り込み終了のデリゲートメソッドが呼び出されます。リスト 7-14 にこのメソッドの簡単な実装例を示します。

リスト 7-14　オーディオプレーヤーの割り込み終了のデリゲートメソッド

```
- (void) audioPlayerEndInterruption: (AVAudioPlayer *) player {
    if (interruptedOnPlayback) {
        [player prepareToPlay];
        [player play];
        playing = YES;
        interruptedOnPlayback = NO;
    }
}
```

割り込みメソッドの定義

注： オーディオ割り込みへの対応に関するこのセクションの以降の内容は、C言語の割り込みリスナーコールバック関数を使って再生または録音への割り込みに対応する場合に該当します。

オーディオ割り込みに対して効果的に対応するには、割り込み開始と割り込み終了の各時点で適切な処理を実行するためのメソッドをアプリケーション内に定義する必要があります。それらのアクションの中には、いずれにせよアプリケーションのどこかで必ず実行しなくてはならない処理（録音の停止など）と、割り込みリスナーコールバック関数が呼び出されたとき以外には必ずしも実行しなくていい処理（再生の一時停止など）があります。

コールバックによってのみ呼び出されるメソッドは、関連するユーザインターフェイスを持っていません。それ以外の点はほかのコントロールメソッドと同様です。ユーザインターフェイスに関連付けられたオーディオコントロールメソッド、関連付けられていないオーディオコントロールメソッドの例については、`recordOrStop:`、`playOrStop:`、`pausePlayback:`、`resumePlayback:` の各メソッドを参照してください。`AudioViewController.m` クラスファイル（*SpeakHere* サンプルコードプロジェクト）にあります。これらのうち、ユーザインターフェイス要素からは直接呼び出されない `resumePlayback:` メソッドのコードを リスト 7-15に転載します。

リスト 7-15　割り込みメソッド

```
- (void) resumePlayback {

    UInt32 sessionCategory = kAudioSessionCategory_MediaPlayback; // 1
    AudioSessionSetProperty ( // 2
        kAudioSessionProperty_AudioCategory, // 3
        sizeof (sessionCategory), // 4
        &sessionCategory // 5
    );
    AudioSessionSetActive (true); // 6

    [self.audioPlayer resume]; // 7
}
```

このコードは次のように動作します。

1. オーディオセッションカテゴリを表す変数を定義し、「メディア再生」カテゴリの識別子を代入して初期化します。これは、アプリケーションのオーディオセッションをアクティブにするときはデバイス上で使われているほかのオーディオを止める、という意図を示すカテゴリです。第6行でオーディオセッションをアクティブにする前に、ここであらかじめカテゴリを設定します。

2. `AudioSessionSetProperty` 関数を呼び出し、オーディオセッションオブジェクトの特定のプロパティに値を設定します。

3. 値を設定する対象のオーディオセッションプロパティです。ここでは、「オーディオカテゴリ (audio category)」プロパティを指定しています。

4. プロパティに設定する値のデータサイズです。

5. カテゴリの識別子を格納した変数のアドレスです。

6. 再生を再開する直前に、オーディオセッションをアクティブにします。

   注： オーディオセッションは割り込みの開始時に非アクティブ化されているため、ここでアクティブにする必要があります。割り込みの終了時にオーディオセッションをアクティブ化しないと、再生も録音を含むアプリケーションのオーディオ機能は動作しません。

7. `resume` メソッド（`audioPlayer` オブジェクト）を呼び出します。

割り込みリスナーコールバック関数の定義

割り込みリスナーコールバック関数は、オーディオセッションの定数 `kAudioSessionBeginInterruption` および `kAudioSessionEndInterruption` で表される2つの割り込み状態に対応した処理を行うように定義します。 リスト 7-16 に基本的な実装例を示します。

リスト 7-16　割り込みリスナーコールバック関数

```
void interruptionListenerCallback (
    void *inUserData, // 1
    UInt32 interruptionState // 2
) {
```

```
    AudioViewController *controller =
        (AudioViewController *) inUserData; // 3

    if (interruptionState == kAudioSessionBeginInterruption) { // 4

        if (controller.audioRecorder) {
            [controller recordOrStop: (id) controller]; // 5
        } else if (controller.audioPlayer) {
            [controller pausePlayback]; // 6
            controller.interruptedOnPlayback = YES; // 7
        }

    } else if ((interruptionState == kAudioSessionEndInterruption) &&
                                    controller.interruptedOnPlayback) { // 8
        [controller resumePlayback];
        controller.interruptedOnPlayback = NO;
    }
}
```

このコードは次のように動作します。

1. オーディオセッションを初期化する際に指定したデータへのポインタです。

   Objective-Cのクラスファイル（View Controllerクラスなど）では、割り込みリスナーコールバック関数のコードはクラス実装ブロックの外側に配置します。したがって、コールバック内からコントローラオブジェクトにメッセージを送るには、送信先オブジェクトへの参照を取得する必要があります。リスト 7-1 （47 ページ）で説明したオーディオセッションの初期化処理で参照を渡しておいたのは、このためです。

2. `Audio Session Interruption States`列挙で定義されている、割り込み状態を表す定数です。

3. `AudioViewController`オブジェクトのインスタンスを、 *`inUserData`* パラメータで渡されてきた参照を使って初期化します。

4. オーディオセッション割り込みの開始時は、この判定が真になります。アプリケーションの録音または再生処理は停止されています。

5. 録音中だった場合は録音バッファをフラッシュし、停止状態を示すためにユーザインターフェイスを更新します。

6. 再生中だった場合は再生の状態を保存し、停止状態（または、実質的には一時停止）を示すためにユーザインターフェイスを更新します。

7. 再生中に割り込みが発生したことを示すフラグを設定します。この後、「割り込み終了」状態でコールバックが呼び出された場合に、このフラグを使います。

8. 割り込みが終了した場合、以前の状態が再生中だったのであれば、再生を再開する必要があります。

オーディオセッションに対する割り込みコールバック関数の登録

オーディオセッションから送られるメッセージを割り込みリスナーコールバック関数で受信するには、コールバックをオーディオセッションに登録しておく必要があります。コールバックの登録は、「オーディオセッションの初期化」（47 ページ）で説明したとおり、オーディオセッションを初期化する際に行います。

OpenAL使用時の割り込みの対応

オーディオ再生にOpenALを使っている場合は、リスト 7-17に示すように、OpenALコンテキストを管理する必要があります。「オーディオセッションの初期化」（47 ページ）で説明したように、このコールバック関数をオーディオセッションに登録します。

リスト 7-17　オーディオ割り込み時のOpenALコンテキストの管理

```
void openALInterruptionListener (
    void *inClientData,
    UInt32 inInterruptionState
) {
    if (inInterruptionState == kAudioSessionBeginInterruption) {
        alcMakeContextCurrent (NULL);
     } else if (inInterruptionState == kAudioSessionEndInterruption) {
        alcMakeContextCurrent (myContext);
     }
    // その他の割り込みリスナー処理コード
}
```

現在のコードに古いバージョンのiPhone OSとの互換性を持たせるために、リスト 7-18に示すようにコードを条件分岐させます。

リスト 7-18　OpenALの割り込みコードに古いバージョンのiOSとの互換性を持たせる

```
- (NSInteger) getMajorOSVersion {
    NSString *versionString = [[UIDevice currentDevice] systemVersion];
    return [[[versionString componentsSeparateByString:@"."] objectAtIndex:0]
intValue];
}

void platformIndependentOpenALInterruptionListener (
    void *inClientData,
    UInt32 inInterruptionState
) {
    if ([self getMajorOSVersion] >= 3) {
        // 3.0 OSのメカニズム -- alcMakeContextCurrentを使用
    } else {
        if (inInterruptionState == kAudioSessionBeginInterruption) {
            // OpenALコンテキストの状態および関連するソースを保存
            StoreMyContextState (myContext);
            alcDestroyContext (myContext);
        } if (inInterruptionState == kAudioSessionEndInterruption) {
            ALCcontext *myContext = alcCreateContext (myOpenALDevice, NULL);
```

```
            // OALContextの状態と関連するOALSourcesを復元する
            RestoreMyContextState (myContext);
        }
    }
    // その他の割り込みリスナー処理コード
}
```

オーディオハードウェア経路の変化への対応

経路の変化に対応するには、プロパティリスナーコールバック関数を使います。システムは、ユーザがヘッドセットを抜き差ししたり、デバイスをドックに載せたりドックから外したりした結果、オーディオ接続の追加や削除が行われると、このコールバックを呼び出します。システムはまた、Bluetoothデバイスの接続または切断時に、プロパティリスナーコールバックも呼び出します。

このセクションのサンプルコードには、経路の変化の後でオーディオ再生が一時停止したときの対処方法をユーザに選択させるアラートの設定と表示が含まれています。

経路変化メソッドの定義

再開または停止の選択肢を提供するアラートを表示するには、`UIAlertView`デリゲートメソッドを定義します（リスト 7-19を参照）。「プロパティリスナーコールバック関数の定義」（61 ページ）に示すように、このメソッドをプロパティリスナーコールバック関数内で使うことができます（*UIAlertViewDelegate Protocol Reference* にも `alertView:clickedButtonAtIndex:` メソッドの解説があります）。これと同様のコードが *AddMusic* のサンプルで使われています。

リスト 7-19　UIAlertViewデリゲートメソッド

```
- (void) alertView: routeChangeAlertView
             clickedButtonAtIndex: buttonIndex { // 1

    if ((NSInteger) buttonIndex == 1) {
        [self resumePlayback]; // 2
    } else {
        [self playOrStop: self]; // 3
    }

    [routeChangeAlertView release]; // 4
}
```

このコードは次のように動作します。

1. *buttonIndex* パラメータには、ユーザのタップしたボタンに対応する整数値（0で始まるインデックス値）が渡されます。アラートの左端のボタンのインデックスが0です。
2. ユーザが右端のボタンをタップした場合は、現在のオブジェクトの `resumePlayback` メソッドを呼び出します。
3. 通知に表示したボタン2つのうち、左のボタンがタップされた場合に対応します。現在のオブジェクトの `playOrStop:` メソッドを呼び出します。

4. プロパティリスナーコールバック関数内で以前に割り当てられたUIAlertViewオブジェクトをここで解放します（「プロパティリスナーコールバック関数の定義」を参照）。

プロパティリスナーコールバック関数の定義

経路の変化に対応するには、プロパティリスナーコールバック関数で次の処理を実行する必要があります。

- 経路にどのような変化が発生したかの確認
- 経路変化の内容と現時点のオーディオコンテキスト（録音中、再生中、停止中など）に応じた場合分け処理
- 適切な処理の実行または呼び出し

リスト7-20は、経路の変化に対応するコールバックのコード例です。サンプルコードプロジェクト*AddMusic*では、同様のコールバック関数を実際のアプリケーション内に実装した例を見ることができます。また、ハードウェアのオーディオ出力の音量や、オーディオ入力が利用できるかどうかが変化した場合など、ほかのオーディオセッションの状態の変化についても、このようなコールバックのコードに相応の変更をほどこすことで対応できます。

リスト 7-20　経路の変化に対応するプロパティリスナーコールバック関数

```
void audioRouteChangeListenerCallback (
    void *inUserData, // 1
    AudioSessionPropertyID inPropertyID, // 2
    UInt32 inPropertyValueSize, // 3
    const void *inPropertyValue // 4
) {
    if (inPropertyID != kAudioSessionProperty_AudioRouteChange) return; // 5

    MainViewController *controller = (MainViewController *) inUserData; // 6

    if (controller.appSoundPlayer.playing == 0 ) { // 7
        return;
    } else {
        CFDictionaryRef routeChangeDictionary = inPropertyValue; // 8
        CFNumberRef routeChangeReasonRef =
                CFDictionaryGetValue (
                    routeChangeDictionary,
                    CFSTR (kAudioSession_AudioRouteChangeKey_Reason)
                );

        SInt32 routeChangeReason;
        CFNumberGetValue (
            routeChangeReasonRef, kCFNumberSInt32Type, &routeChangeReason
        );

        if (routeChangeReason ==
                kAudioSessionRouteChangeReason_OldDeviceUnavailable) { // 9

            [controller.appSoundPlayer pause];
```

```
            UIAlertView *routeChangeAlertView =
                [[UIAlertView alloc]
                    initWithTitle: @"Playback Paused"
                          message: @"Audio output was changed."
                         delegate: controller
                cancelButtonTitle: @"Stop"
                otherButtonTitles: @"Play", nil];
            [routeChangeAlertView show];
        }
    }
}
```

このコードは次のように動作します。

1. オーディオセッションを初期化する際に指定したデータへのポインタです。

   Objective-Cのクラスファイル（View Controllerクラスなど）では、プロパティリスナーコールバック関数のコードはクラス実装ブロックの外側に配置します。したがって、コールバック内からコントローラオブジェクトにメッセージを送るには、送信先オブジェクトへの参照を取得する必要があります。このために、リスト 7-21 （63 ページ）で説明するオーディオセッションの初期化処理で参照を渡しています。

2. このコールバック関数で通知を受ける対象プロパティの識別子です。

3. *inPropertyValue* パラメータに渡されたデータのサイズ（バイト単位）です。

4. このコールバック関数で監視しているプロパティの現在の値です。監視対象プロパティが `kAudioSessionProperty_AudioRouteChange` なので、この値は `CFDictionary` オブジェクトです。

5. コールバックの呼び出された理由が、目的のオーディオセッションプロパティの変更によるものかどうか確認します。

6. `MainViewController` オブジェクトのインスタンスを、 *inUserData* パラメータで渡されてきた参照を使って初期化します。これにより、コールバックからView Controllerオブジェクト（通常はコールバックを実装している同じファイル内で定義されています）へメッセージを送ることができます。

7. アプリケーションでサウンドの再生を行っていない場合は、何もせずに復帰します。

8. この行と次のいくつかの行では、経路の変化の原因を特定します。この再生専用の例で注目する経路の変化は、ヘッドセットのような出力デバイスの取り外しです。

9. 出力デバイスが実際に取り外された場合、再生を一時停止して、ユーザ側で再生の停止や再開を行うためのアラートを表示します。メソッド内で `UIAlertView` オブジェクトの解放が行われます（リスト 7-19 （60 ページ）を参照）。

オーディオセッションに対するプロパティリスナーコールバック関数の登録

ハードウェアや経路の変化に関するイベントをアプリケーションで検知するには、Audio Session Servicesに用意されたプロパティメカニズムを使います。たとえば、経路変化のイベントを検知するには、リスト7-21のようにしてコールバック関数をオーディオセッションオブジェクトに登録します。

リスト 7-21　Audio Session Servicesに対するプロパティリスナーコールバックの登録

```
AudioSessionPropertyID routeChangeID =
                        kAudioSessionProperty_AudioRouteChange; // 1
AudioSessionAddPropertyListener ( // 2
    routeChangeID, // 3
    propertyListenerCallback, // 4
    userData // 5
);
```

このコードは次のように動作します。

1. 変数を宣言し、監視したいプロパティの識別子を代入して初期化します。
2. 初期化済みのオーディオセッションに対して、プロパティリスナーコールバック関数を登録します。
3. 監視対象となるプロパティの識別子です。
4. 登録するハードウェアリスナーコールバック関数への参照です。
5. オーディオセッションから呼び出されるコールバック関数に渡すためのデータです。

オーディオハードウェア特性の問い合わせと使用

iOSデバイスのハードウェア特性は、アプリケーションの動作中に変化することがあり、デバイスによって異なることもあります。たとえば、オリジナルのiPhoneの内蔵マイクを使う場合、録音サンプルレートは8 kHzに制限されますが、ヘッドセットを接続してヘッドセットのマイクを使う場合はもっと高いサンプルレートを使うことができます。新しいiOSデバイスは、内蔵マイク用にはより高いハードウェアサンプルレートをサポートしています。

最適なハードウェア特性を指定する前に、オーディオセッションがアクティブでないことを確認します。必要な設定した後、オーディオセッションをアクティブにしてから、オーディオセッションに問い合わせて実際の特性を決定します。システムが要求に応えられない場合もあるため、この最後の手順は重要です。

最適なハードウェアI/Oバッファ時間の指定

AVAudioSessionクラスを使うか、またはAudio Session Servicesを使うことによって、最適なハードウェアサンプルレートとハードウェアI/Oバッファ時間を指定できます。 リスト 7-22 に、AVAudioSessionクラスを使って最適なI/Oバッファ時間を設定する方法を示します。最適なサンプルレートを設定する場合も、同じようなコードを使います。

リスト 7-22　AVAudioSessionクラスを使った最適なI/Oバッファ時間の指定

```
NSError *setPreferenceError = nil;
NSTimeInterval preferredBufferDuration = 0.005;
[[AVAudiosession sharedInstance]
            setPreferredIOBufferDuration: preferredBufferDuration
                                  error: &setPreferenceError];
```

リスト 7-23 に、C言語ベースのAudio Session Servicesを使って同じことを実現する方法を示します。最適なサンプルレートを設定する場合も、同じようなコードを使います。

リスト 7-23　Audio Session Servicesを使った最適なI/Oバッファ時間の指定

```
Float32 preferredBufferDuration = 0.005; // 1
AudioSessionSetProperty ( // 2
    kAudioSessionProperty_PreferredHardwareIOBufferDuration,
    sizeof (preferredBufferDuration),
    &preferredBufferDuration
);
```

このコードは次のように動作します。

1. I/Oバッファ時間の値（秒単位）を宣言し、初期化します。
2. 望みのI/Oバッファ時間を設定します。

システムが要求に応えられない場合もあるため、ハードウェアの設定を行った後は、リスト 7-25 （65 ページ）に示すように、必ず実際の値をハードウェアに問い合わせます。

ハードウェアサンプルレートの取得と使用

オーディオの録音に備える設定を行う際には、現在のオーディオハードウェアサンプルレートを取得し、それに応じたオーディオデータ形式を使う必要があります。 リスト 7-24 にその方法を示します。このコードは、録音を実行するクラスの実装ファイル内に配置するのが普通です。入出力のチャネル数など、ほかのハードウェア特性を取得する場合も同様のコードを使います。

オーディオセッションに現在のハードウェア特性を問い合わせる前に、オーディオセッションがアクティブであることを確認してください。

リスト 7-24　AVAudioSessionクラスを使った現在のオーディオハードウェアサンプルレートの取得

```
double sampleRate;
sampleRate = [[AVAudiosession sharedInstance] currentHardwareSampleRate];
```

リスト 7-25に示すように、C言語ベースのAudio Session Servicesを使って同じことを実現できます。

リスト 7-25　Audio Session Servicesを使った現在のオーディオハードウェアサンプルレートの取得

```
UInt32 propertySize = sizeof (self.hardwareSampleRate); // 1

AudioSessionGetProperty ( // 2
    kAudioSessionProperty_CurrentHardwareSampleRate,
    &propertySize,
    &hardwareSampleRate
);
```

このコードは次のように動作します。

1. `UInt32`型の変数を宣言し、このメソッドを定義しているオブジェクトに属するインスタンス変数 `hardwareSampleRate` のサイズ（バイト単位）を代入して初期化します。このインスタンス変数は、ヘッダファイル内で `Float64` 型の値として宣言されています。完全なコード例としては、サンプルコードプロジェクト *SpeakHere* を参照してください。
2. プロパティ識別子 `kAudioSessionProperty_CurrentHardwareSampleRate` を使ってオーディオセッションオブジェクトへの問い合わせを実行し、現在のオーディオハードウェアサンプルレートを取得します。サンプルレートは、インスタンス変数 `hardwareSampleRate` に格納されます。

デバイスで録音がサポートされているかどうかを調べる方法

アプリケーションがマイク組み込みのiOSデバイス上で動作している場合は常に録音がサポートされます。しかし、iPod touch上では、適切なハードウェアアクセサリが接続されている場合のみ録音がサポートされます。 リスト 7-27 に、デバイス上で録音が可能かを調べる方法を示します。

オーディオセッションの初期化処理（一般的に、アプリケーションの起動時に実行）の後、オーディオ入力のサポート状況を判定します。オーディオ入力の可否の変更についてのメッセージを受け取るには、 `inputIsAvailableChanged:` メソッドも実装する必要があります。これは、`AVAudioSessionDelegate` （*AVAudioSessionDelegate Protocol Reference*を参照）プロトコルに含まれているものです。

リスト 7-26　デバイスでオーディオの録音がサポートされているかをAVAudioSessionクラスを使って判定する方法

```
BOOL inputAvailable;
inputAvailable = [[AVAudioSession sharedInstance] inputIsAvailable];
```

リスト 7-27に示すように、C言語ベースのAudio Session Servicesを使って同じことを実現できます。

リスト 7-27　デバイスでオーディオの録音がサポートされているかをAudio Session Servicesを使って判定する方法

```
UInt32 audioInputIsAvailable; // 1
UInt32 propertySize = sizeof (audioInputIsAvailable); // 2

AudioSessionGetProperty ( // 3
    kAudioSessionProperty_AudioInputAvailable,
    &propertySize,
    &audioInputIsAvailable
);
```

このコードは次のように動作します。

1. `kAudioSessionProperty_AudioInputAvailable` プロパティの値を格納するための変数を宣言します。
2. 変数を宣言し、プロパティ値のサイズを代入して初期化します。
3. 初期化済みのオーディオセッションから、`kAudioSessionProperty_AudioInputAvailable` プロパティの値を取得します。オーディオ入力を利用できる場合は、`audioInputIsAvailable` 変数に0以外の値が格納されます。

シミュレータ上でのアプリケーションの実行

シミュレータの特性のため（「オーディオセッションAPIを使ったアプリケーション開発」（19ページ）を参照）、作成したコードをシミュレータ上で部分的にテストできるように条件分岐させる必要が生じることもあります。

1つの方法として、API呼び出しの戻り値に基づいて分岐する方法があります。たとえば、`AudioSessionGetProperty` 関数を、プロパティ識別子 `kAudioSessionProperty_AudioRoute` を渡して呼び出し、現在のオーディオ経路を取得しようとしても、シミュレータでは失敗します。すべてのオーディオセッション関数呼び出しからの結果コードをチェックして適切に対処するようにしてください。結果コードは、アプリケーションがあるデバイス上では適切に動作し、シミュレータ上では失敗した原因を示している可能性があります。

オーディオセッションの結果コードを適切に使うことに加えて、プリプロセッサの条件文を使って、シミュレータ上で実行するときに特定のコードを隠すこともできます。 リスト 7-28 にその方法を示します。

リスト 7-28　プリプロセッサの条件文の使用

```
#if TARGET_IPHONE_SIMULATOR
#warning *** シミュレータモード：オーディオセッションコードはデバイス上でのみ動作します
    // シミュレータ上で動作するコードのサブセットを実行
#else
    // デバイス専用コードとほかのコードを実行
#endif
```

使用方法ガイドラインの提示

アプリケーションの動作がほかの競合するセッションによって割り込まれると、ユーザにとって不都合が生じる場合があります。たとえば、講演を録音しているときに割り込みが入ることは望ましくないと考えられます。

しかし、オーディオセッションの割り込みをプログラムで確実に防止する方法はありません。これは、iOSでは常に電話が最優先の扱いを受けるからです。また、ある種のアラームやアラートもiOSでは高い優先度を持ちます（飛行機に乗り遅れないためわざわざアラームを設定したのに、鳴らないということは許されません）。

したがって、録音中の割り込みを確実に防ぐには、ユーザが意識的に次の操作を行ってiOSデバイスのオーディオを止める必要があります。

1. 「設定(Settings)」アプリケーションで、機内モードであるべきデバイスについて、実際にそうなっていることを確認します。
2. 「カレンダー(Calendar)」アプリケーションで、録音の終了予定時刻までのイベントアラームをすべて解除します。
3. 「時計(Clock)」アプリケーションで、録音の終了予定時刻までの時計アラームをすべて解除します。
4. サイレントスイッチ（iPhoneでは *着信／サイレントスイッチ*）付きのデバイスの場合、録音中はこのスイッチを操作しないように注意します。サイレントモードを変更すると、ユーザ設定によっては、たとえばバイブレーションが作動することがあります。
5. 録音中はヘッドセットを抜き差ししないように注意します。デバイスをドックに載せたりドックから外したりする操作も、同じく録音中は行わないようにします。
6. 録音中はデバイスの電源アダプタを接続しないように注意します。iOSデバイスに電源アダプタを接続すると、デバイスやユーザ設定によってはビープ音が鳴ることや、バイブレーションが作動することがあります。

アプリケーションのユーザガイドなどに、これらの注意事項を明示しておくことをお勧めします。

オーディオセッションのカテゴリ

アプリケーションは、オーディオセッションのカテゴリを設定することによって、オーディオの使いかたに関する意図をiOSに伝えます。 表 A-1 に、カテゴリごとの詳細を示します。影付きの行はデフォルトのカテゴリ kAudioSessionCategory_SoloAmbientSoundです。カテゴリが機能する仕組みについての説明は、 「最適なカテゴリの選択」 （22 ページ）を参照してください。ミックスオーバーライドスイッチの詳細については、 「カテゴリの微調整」 （25 ページ）を参照してください。

表 A-1 各オーディオセッションカテゴリの動作

カテゴリ識別子*	着信／サイレントスイッチや画面のロックによって消音する	ほかのアプリケーションからのオーディオを許可	オーディオ入力（録音）と出力（再生）を許可
AVAudioSessionCategoryAmbient kAudioSessionCategory_AmbientSound	○	○	出力のみ
AVAudioSessionCategorySoloAmbient kAudioSessionCategory_SoloAmbientSound	○	×	出力のみ
AVAudioSessionCategoryPlayback kAudioSessionCategory_MediaPlayback	×	デフォルトでは×、オーバーライドスイッチを使うことで○	出力のみ
AVAudioSessionCategoryRecord kAudioSessionCategory_RecordAudio	×（画面がロックした状態でも録音は継続）	×	入力のみ
AVAudioSessionCategoryPlayAndRecord kAudioSessionCategory_PlayAndRecord	×	デフォルトでは×、オーバーライドスイッチを使うことで○	入力および出力
AVAudioSessionCategoryAudioProcessing kAudioSessionCategory_AudioProcessing	–	×	入力、出力とも×

*各行において、1つ目の識別子はObjective-CベースのAVAudioSessionインターフェイス用で、2つ目の識別子はCベースのAudio Session Servicesインターフェイス用です。

メモ： iOS 2.2よりも前のiOSには `kAudioSessionCategory_SoloAmbientSound` カテゴリがなく、`kAudioSessionCategory_MediaPlayback` カテゴリがデフォルトとして使われていました。使用が廃止された2つのカテゴリは、表には示されていません。廃止された `kAudioSessionCategory_UserInterfaceSoundEffects` カテゴリは `AVAudioSessionCategoryAmbient`と同等、廃止された `kAudioSessionCategory_LiveAudio` カテゴリは `AVAudioSessionCategoryPlayback`と同等です。

書類の改訂履歴

この表は「*オーディオセッションプログラミングガイド*」の改訂履歴です。

日付	メモ
2010-11-15	`playInputClick` メソッドに関する情報を「オーディオセッションの基礎」（13 ページ）に追加しました。
2010-09-01	iOS 4でのアプリケーションのライフサイクルをサポートする方法の説明を更新しました。
	「オーディオセッションのアクティブ化と非アクティブ化」（21 ページ）に、アプリケーションのオーディオセッションを非アクティブ化するべき状況の説明を記載しました。
2010-07-09	細部の変更。
2010-04-12	iOS 3.2向けに、ムービープレーヤーのオーディオセッションに関する変更点を記述して更新しました。「ムービーおよびiPodミュージックの使用」（43 ページ）を参照してください。
2010-01-29	「電力消費を最小限に最適化する」（52 ページ）、「最適なオーディオハードウェア設定の指定」（39ページ）の説明を若干変更しました。
2010-01-20	「オーディオ動作の設定について」（9 ページ）を修正しました。
	コードレシピを追加しました：「アプリケーションの起動時にほかのオーディオが再生中かをチェックする」（49 ページ）
	コードレシピを追加しました：「画面がロックされても再生を続けるようにオーディオユニットを設定する」（51 ページ）
	以下のとおり、オーディオハードウェアを操作するためのベストプラクティス情報を追加しました。「最適なオーディオハードウェア設定の指定」（39 ページ）に「重要」のメモを追加しました。「ハードウェア特性の問い合わせ」（40 ページ）に「重要」のメモを追加しました。「オーディオハードウェア特性の問い合わせと使用」（63 ページ）のコードレシピを改善しました。
2009-10-19	「オーディオセッションAPIを使ったアプリケーション開発」（19 ページ）のオーディオセッションに関して、シミュレータの動作についての補足情報を追加しました。
	「シミュレータ上でのアプリケーションの実行」（66 ページ）に、シミュレータの特性に対応するためのヒントをいくつか追加しました。
2009-09-09	iOS 3.1向けに更新して、新しいオーディオセッションプロパティの説明を追加しました。

日付	メモ
	文書に次の4つの章を新たに追加しました。「オーディオ割り込みの処理方法」（27 ページ）、「オーディオハードウェア経路の変化」（35 ページ）、「デバイスハードウェアに合わせた最適化」（39 ページ）、「ムービーおよびiPodミュージックの使用」（43 ページ）。
	「オーディオセッションの設定」（21 ページ）に、「エンコードとデコードへのカテゴリの影響」（24 ページ）と「カテゴリの微調整」（25 ページ）を含む、新しいテーマを追加しました。
	「オーディオセッションの基礎」（13ページ）、「オーディオセッションカテゴリとは」（14 ページ）、「2つのオーディオセッションAPI」（18 ページ）に新しいセクションを追加しました。
	「オーディオセッションの活用」（47ページ）を増強して、新しいテーマに対応するサンプルを追加しました。
2008-11-13	iOSアプリケーションのオーディオセッションの使いかたを説明する、iOS 2.2向け新規文書。