AV Foundationプログラミングガイド

Cocoa

2011-10-12

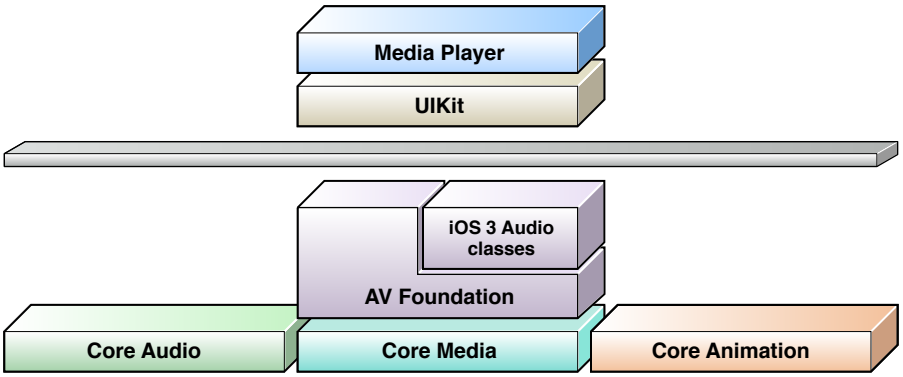
Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
U.S.A.

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

AV Foundationフレームワークについて

AV Foundationは、時間ベースのオーディオビジュアルメディアの再生と作成に使用できるフレームワークで、時間ベースのオーディオビジュアルデータに関する作業を細かいレベルで行うためのObjective-Cのインターフェイスを提供します。たとえば、メディアファイルの検査、作成、編集、再エンコードなどができます。デバイスから入力ストリームを取得して、リアルタイムでキャプチャ中および再生中のビデオを操作することもできます。



必要な作業を行うには、一般に、使用できる最上位の抽象クラスを使用する必要があります。iOSでは、たとえば次のようにします。

- ムービーを再生するだけの場合は、Media Playerフレームワーク（`MPMoviePlayerController`または`MPMoviePlayerViewController`）を使用できます。Webベースのメディアには、`UIWebView`オブジェクトを使用することもできます。
- ビデオを録画するときに最小限のフォーマット制御だけ必要な場合は、UIKitフレームワーク(`UIImagePickerController`)を使用します。

ただし、AV Foundationで使用する基本的なデータ構造の中には、Core Mediaフレームワークで宣言されるものがあります。たとえば、時間に関するデータ構造や、メディアデータの格納と記述に使用する不透過オブジェクトなどです。

AV Foundationは、iOS 4以降、Mac OS X v10.7以降で利用できます。この文書では、iOS 4.0で導入された時点でのAV Foundationについて説明します。それ以降の版のフレームワークにおける変更点や追加機能については、次の該当するリリースノートを参照してください。

- 『*AV Foundation Release Notes*』にはiOS 5での変更点を記載してあります。
- 『*AV Foundation Release Notes (iOS 4.3)*』には、iOS 4.3、OS X 10.7での変更点を記載してあります。

AV Foundationは高度なCocoaフレームワークです。このフレームワークを効果的に使用するには、次の知識が必要です。

- Cocoaの基本的な開発ツールおよび手法

- ブロックの基本
- キー値コーディングおよびキー値監視の基本
- 再生に関しては、Core Animationの基本（『*Core Animation Programming Guide*』を参照）

関連する章： 「時間およびメディアの表現」 （47 ページ）

AV Foundationでのメディアの表現と使用

AV Foundationフレームワークでメディアを表すために使用する主要なクラスは、`AVAsset`です。フレームワークの大部分は、この表現に基づいて設計されています。このクラスの構造を理解すると、フレームワークの動作を理解しやすくなります。`AVAsset`インスタンスは、1つ以上のメディアデータのコレクションを集約して表したものです。コレクションのタイトル、再生時間、本来の表示サイズなど、コレクションに関する情報をまとめて提供します。`AVAsset`は特定のデータフォーマットに関連付けられていません。`AVAsset`は、あるURLのメディアからアセットインスタンスを作成したり（「アセットの使用」 （9 ページ）を参照）、新しいコンポジションを作成したり（「編集」 （7 ページ）を参照）するために使用するほかのクラスのスーパークラスです。

アセット内の個々のメディアデータは、タイプが同じであり、**トラック**と呼ばれます。通常の簡単なケースでは、オーディオコンポーネントを表すトラックとビデオコンポーネントを表すトラックが別個に存在しますが、複雑なコンポジションでは、オーディオとビデオのトラックが複数重複して存在する場合もあります。アセットにはメタデータも格納できます。

AV Foundationで非常に重要なことは、アセットやトラックを初期化しただけでは必ずしも使用できる状態にはならないということです。場合によっては、アイテムの再生時間も計算する必要があります（たとえば、MP3ファイルには要約情報が含まれていない場合があります）。現在のスレッドをブロックしてその間に値を計算するのではなく、ブロックを使用してデベロッパが定義するコールバックを通じて値を問い合わせ、非同期に応答を取得します。

関連する章： 「アセットの使用」 （9 ページ）
「時間およびメディアの表現」 （47 ページ）

Playback

AV Foundationでは、アセットの再生を洗練された方法で管理できます。これに対応するため、アセット自体からアセットの表示状態が分離されています。これにより、たとえば、同じアセットの2つの異なるセグメントを異なる解像度でレンダリングして同時に再生できます。アセットの表示状態は**プレーヤーアイテム**オブジェクトによって管理され、アセット内の各トラックの表示状態は**プレーヤーアイテムトラック**オブジェクトによって管理されます。プレーヤーアイテムとプレーヤーアイテムトラックを使用して、たとえば、アイテムの表示部分をプレーヤーに表示する際のサイズを設定したり、再生中に適用するオーディオミックスパラメータやビデオコンポジションを設定したり、再生中にアセットのコンポーネントを無効にしたりできます。

プレーヤーオブジェクトを使用してプレーヤーアイテムを再生し、プレーヤーの出力をCore Animationレイヤに送ります。iOS 4.1以降では、**プレーヤーキュー**を使用してプレーヤーアイテムのコレクションの再生を順にスケジューリングできます。

関連する章： 「再生」（17 ページ）

アセットの読み取り、書き込み、および再エンコード

AV Foundationでは、アセットの新しい表現を複数の方法で作成できます。既存のアセットを再エンコードすることもできますが、iOS 4.1以降では、アセットの内容に対して操作を実行し、その結果を新しいアセットとして保存することもできます。

既存のアセットを再エンコードして、あらかじめ用意されている定義済みのフォーマットの1つに変換するには、**エクスポートセッション**を使用します。iOS 4.1以降では、変換方法を詳細に制御する必要がある場合に、**アセットリーダー**オブジェクトと**アセットライター**オブジェクトの組み合わせを使用して、アセットをある表現から別の表現に変換できます。これらのオブジェクトを使用して、たとえば、出力ファイルで表現するトラックを選んだり、独自の出力フォーマットを指定したり、変換処理中にアセットを変更したりできます。

波形の視覚表現を作成するには、アセットリーダーを使用してアセットのオーディオトラックを読み取ります。

関連する章： 「アセットの使用」（9 ページ）

サムネイル

ビデオ表現のサムネイル画像を生成するには、サムネイルの作成元となるアセットを使用して`AVAssetImageGenerator`のインスタンスを初期化します。`AVAssetImageGenerator`は、デフォルトで有効になっているビデオトラックを使用して画像を生成します。

関連する章： 「アセットの使用」（9 ページ）

編集

AV Foundationでは、**コンポジション**を使用して既存のメディア（通常は1つ以上のビデオおよびオーディオトラック）から新しいアセットを作成します。トラックの追加、トラックの削除、およびトラックの時間順序の調整を行うには、可変コンポジションを使用します。オーディオトラックの相対的なボリュームとランピングを設定したり、ビデオトラックの不透明度（および不透明度のランピング）を設定したりすることもできます。コンポジションは、メディアのメモリに格納されている部分を組み合わせたものです。**エクスポートセッション**を使用してコンポジションをエクスポートすると、コンポジションが1つのファイルに集約されます。

iOS 4.1以降では、**アセットライター**を使用してサンプルバッファや静止画像などのメディアからアセットを作成することもできます。

メディアキャプチャとカメラへのアクセス

カメラとマイクからの入力の録音・録画は、**キャプチャセッションによって**管理されます。キャプチャセッションは、入力デバイスから出力（ムービーファイルなど）へのデータの流れを調整します。1つのセッションに複数の入力および出力を設定できます。設定はセッションの実行中でも可能です。データの流れを開始したり停止したりするには、セッションにメッセージを送信します。

さらに、**プレビューレイヤ**のインスタンスを使用して、カメラで録画している内容をユーザに表示することもできます。

関連する章： 「メディアキャプチャ」（29 ページ）

AV Foundationのオーディオ関連クラス

AV Foundationフレームワークには2つの側面があります。1つはiOS 4より前から用意されていたオーディオのみに関するAPIであり、もう1つはiOS 4以降で導入されたAPIです。古いオーディオ関連のクラスは、オーディオを簡単に扱うための手段を提供します。これらについては、この文書ではなく、『*Multimedia Programming Guide*』で説明しています。

- サウンドファイルを再生するには、`AVAudioPlayer`を使用します。
- オーディオを録音するには、`AVAudioRecorder`を使用します。

`AVAudioSession`を使用してアプリケーションのオーディオ動作を設定することもできます。これについては、『*Audio Session Programming Guide*』で説明しています。

AV Foundationでの並列プログラミング

特定のスレッドやキューでAV Foundationからの呼び出し（ブロック、キー値オブザーバ、または通知ハンドラの呼び出し）が行われるという保証はありません。代わりに、AV Foundationは内部タスクを実行するスレッドまたはキューでこれらのハンドラを呼び出します。ハンドラが呼び出されたスレッドまたはキューが実行するタスクにとって適切かどうかを確認するのはデベロッパの責任です。適切でない場合（たとえば、ユーザインターフェイスを更新する必要があり、呼び出しがメインスレッドで行われなかった場合）は、デベロッパが認識している（または目的に合わせて作成した）安全なスレッドまたはキューにタスクの実行をリダイレクトする必要があります。

マルチスレッドアプリケーションを作成する場合は、`NSThread`のメソッド`isMainThread`または`[[NSThread currentThread] isEqual:<#A stored thread reference#>]`を使用して、呼び出し元のスレッドが作業の実行環境として期待したスレッドかどうかを確認できます。`performSelectorOnMainThread:withObject:waitUntilDone:`や`performSelector:onThread:withObject:waitUntilDone:modes:`などのメソッドを使用して、メッセージを適切なスレッドにリダイレクトできます。また、`dispatch_async(3) Mac OS X Manual Page`を使用してブロックを適切なキュー（UIタスク用のメインキュー、または並列操作のために設定したキュー）に「バウンス」することもできます。並列操作については『*Concurrency Programming Guide*』、ブロックについては『*Blocks Programming Topics*』を参照してください。

アセットの使用

アセットは、ファイルまたはユーザのiPodライブラリまたはフォトライブラリ内のメディアから入力されます。しかし、アセットオブジェクトを作成しただけで、そのアイテムに関して取得したいすべての情報がすぐに得られるとは限りません。ムービーアセットを作成した後で、アセットから静止画像を抽出したり、別のフォーマットに変換したり、コンテンツをトリミングしたりできます。

アセットオブジェクトの作成

URLで指定できるリソースを表すアセットを作成するには、`AVURLAsset`を使用します。最も簡単なケースでは、次のようにファイルからアセットを作成します。

```
NSURL *url = <#A URL that identifies an audiovisual asset such as a movie file#>;
AVURLAsset *anAsset = [[AVURLAsset alloc] initWithURL:url options:nil];
```

アセットの初期化オプション

`AVURLAsset`の初期化メソッドは、第2引数としてoptionsディクショナリを取ります。このディクショナリで使用される唯一のキーは`AVURLAssetPreferPreciseDurationAndTimingKey`です。対応する値は、正確な再生時間を指定し、時間単位の正確なランダムアクセスを許可するようにアセットを準備する必要があるかどうかを示す（`NSValue`オブジェクトに格納された）ブール値です

アセットの正確な再生時間を取得すると、かなりの処理オーバーヘッドが生じる可能性があります。大まかな再生時間を使用することで、通常は操作の負担が軽くなり、再生には十分です。したがって、次のようにします。

- アセットの再生だけを目的とする場合は、ディクショナリの代わりに`nil`を渡すか、`AVURLAssetPreferPreciseDurationAndTimingKey`キーとその値として（`NSValue`オブジェクトに格納された）`NO`を含むディクショナリを渡します。
- アセットをコンポジション（`AVMutableComposition`）に追加する場合は、通常、正確なランダムアクセスが必要です。次のように、`AVURLAssetPreferPreciseDurationAndTimingKey`キーとその値として（`NSValue`オブジェクトに格納された — 先に説明したように、`NSNumber`は`NSValue`から継承している）`YES`を含むディクショナリを渡します。

  ```
  NSURL *url = <#A URL that identifies an audiovisual asset such as a movie file#>;
  NSDictionary *options = [NSDictionary dictionaryWithObject:[NSNumber
  numberWithBool:YES]

  forKey:AVURLAssetPreferPreciseDurationAndTimingKey];
  AVURLAsset *anAssetToUseInAComposition = [[AVURLAsset alloc] initWithURL:url
  options:options];
  ```

ユーザのアセットへのアクセス

iPodライブラリや「写真(Photo)」アプリケーションで管理されているアセットにアクセスするには、アセットのURLを取得する必要があります。

- iPodライブラリにアクセスするには、`MPMediaQuery`インスタンスを作成して必要なアイテムを見つけ、`MPMediaItemPropertyAssetURL`でURLを取得します。

  メディアライブラリの詳細については、『*Multimedia Programming Guide*』を参照してください。

- 「写真(Photo)」アプリケーションで管理されているアセットにアクセスするには、`ALAssetsLibrary`を使用します。

次の例は、「カメラロール(Saved Photos)」アルバムの最初のビデオを表すアセットを取得する方法を示しています。

```
ALAssetsLibrary *library = [[ALAssetsLibrary alloc] init];

// ALAssetsGroupSavedPhotosを使用して写真とビデオのグループを列挙する
[library enumerateGroupsWithTypes:ALAssetsGroupSavedPhotos
usingBlock:^(ALAssetsGroup *group, BOOL *stop) {

// グループ列挙ブロックの中で、ビデオだけを列挙するようにフィルタリングする
[group setAssetsFilter:[ALAssetsFilter allVideos]];

// この例では最初のアイテムだけが必要
[group enumerateAssetsAtIndexes:[NSIndexSet indexSetWithIndex:0]
                        options:0
                     usingBlock:^(ALAsset *alAsset, NSUInteger index, BOOL
*innerStop) {

                         // 列挙の終わりはasset == nilで示す
                         if (alAsset) {
                             ALAssetRepresentation *representation = [alAsset
defaultRepresentation];
                             NSURL *url = [representation url];
                            AVAsset *avAsset = [AVURLAsset URLAssetWithURL:url
 options:nil];
                             // このAVアセットを使って必要な作業を行う
                         }
                     }];
                }
                failureBlock: ^(NSError *error) {
                    // 通常はもっと丁寧にエラーを処理する必要がある
                    NSLog(@"No groups");
                }];

[library release];
```

アセットを使用する準備

アセット（またはトラック）を初期化しても、そのアイテムに関して取得したいすべての情報がすぐに得られるとは限りません。場合によっては、アイテムの再生時間も計算する必要があります（たとえば、MP3ファイルには要約情報が含まれていない場合があります）。そのような場合は、現在のスレッドを遮断してその間に値を計算するのではなく、`AVAsynchronousKeyValueLoading`プロトコルを使用して値を問い合わせ、ブロックで定義した終了ハンドラを通じて回答を得る必要があります（`AVAsset`および`AVAssetTrack`は、`AVAsynchronousKeyValueLoading`プロトコルに準拠しています）。

`statusOfValueForKey:error:`を使用してプロパティの値がロードされたかどうかを確認します。最初にアセットをロードしたときは、ほとんどまたはすべてのプロパティの値が`AVKeyValueStatusUnknown`です。1つ以上のプロパティの値をロードするには、`loadValuesAsynchronouslyForKeys:completionHandler:`を呼び出します。終了ハンドラでは、プロパティのステータスに応じて適切なアクションを実行します。ロードが何らかの理由（ネットワークベースのURLにアクセスできないなど）で失敗したり、キャンセルされたりするため、常にロードが正常に完了しない場合に備える必要があります。 .

```
NSURL *url = <#A URL that identifies an audiovisual asset such as a movie file#>;
AVURLAsset *anAsset = [[AVURLAsset alloc] initWithURL:url options:nil];
NSArray *keys = [NSArray arrayWithObject:@"duration"];

[asset loadValuesAsynchronouslyForKeys:keys completionHandler:^() {

    NSError *error = nil;
    AVKeyValueStatus tracksStatus = [asset statusOfValueForKey:@"duration" error:&error];
    switch (tracksStatus) {
        case AVKeyValueStatusLoaded:
            [self updateUserInterfaceForDuration];
            break;
        case AVKeyValueStatusFailed:
            [self reportError:error forAsset:asset];
            break;
        case AVKeyValueStatusCancelled:
            // キャンセルに対する適切な処理を実行する
            break;
    }
}];
```

再生用のアセットを準備するには、アセットの`tracks`プロパティをロードする必要があります。アセットの再生の詳細については、「再生」 （17 ページ）を参照してください。

ビデオからの静止画像の取得

再生用のアセットからサムネイルなどの静止画像を取得するには、`AVAssetImageGenerator`オブジェクトを使用します。アセットを使用して画像ジェネレータを初期化します。ただし、初期化の時点でアセットにビジュアルトラックが存在しなくても初期化に成功することがあるため、必要な場合は`tracksWithMediaCharacteristic:`を使用して、ビジュアル特性を持つトラックがアセットに存在するかどうかを確認してください。

```
AVAsset anAsset = <#Get an asset#>;
```

```
if ([anAsset tracksWithMediaCharacteristic:AVMediaTypeVideo]) {
    AVAssetImageGenerator *imageGenerator =
        [AVAssetImageGenerator assetImageGeneratorWithAsset:anAsset];
    // 実装が続く…
```

画像ジェネレータのいくつかの側面を設定できます。たとえば、maximumSizeとapertureModeを使用して、生成される画像の最大サイズと開口モードをそれぞれ指定できます。1度に1つの画像を生成することも、一連の画像を生成することもできます。必ずすべての画像が生成されるまで画像ジェネレータを保持する必要があります。

1つの画像の生成

1度に1つの画像を生成するには、copyCGImageAtTime:actualTime:error:を使用します。AV Foundationが要求された正確な時間に画像を生成できるとは限りません。そこで、2番目の引数としてCMTimeへのポインタを渡すと、戻ったときに、画像が実際に生成された時間がそこに格納されます。

```
AVAsset *myAsset = <#An asset#>];
AVAssetImageGenerator *imageGenerator = [[AVAssetImageGenerator alloc]
initWithAsset:myAsset];

Float64 durationSeconds = CMTimeGetSeconds([myAsset duration]);
CMTime midpoint = CMTimeMakeWithSeconds(durationSeconds/2.0, 600);
NSError *error = nil;
CMTime actualTime;

CGImageRef halfWayImage = [imageGenerator copyCGImageAtTime:midpoint
actualTime:&actualTime error:&error];

if (halfWayImage != NULL) {

    NSString *actualTimeString = (NSString *)CMTimeCopyDescription(NULL,
actualTime);
    NSString *requestedTimeString = (NSString *)CMTimeCopyDescription(NULL,
midpoint);
    NSLog(@"got halfWayImage: Asked for %@, got %@", requestedTimeString,
actualTimeString);
    [actualTimeString release];
    [requestedTimeString release];

    // この画像を使って必要な作業を実行する
    CGImageRelease(halfWayImage);
}
[imageGenerator release];
```

一連の画像の生成

一連の画像を生成するには、画像ジェネレータにgenerateCGImagesAsynchronouslyForTimes:completionHandler:メッセージを送信します。最初の引数は、NSValueオブジェクトの配列です。この配列には、各画像を生成するアセット時間を指定したCMTimeを格納します。2番目の引数は、生成された画像ごとに呼び出されるコールバックとして機能するブロックです。このブロック引数は、画像が正常に作成されたか、または操作がキャンセルされたかを示す結果が返されます。また、該当する場合は次の情報も提供されます。

- 画像
- 画像を要求した時間と画像が実際に生成された時間
- 生成に失敗した理由を示すエラーオブジェクト

このブロックの実装において、結果定数を確認し、画像が作成されたかどうかを判定する必要があります。また、必ず画像の作成が完了するまで画像ジェネレータを保持する必要があります。

```
AVAsset *myAsset = <#An asset#>];
// 前提：@property (retain) AVAssetImageGenerator *imageGenerator;
self.imageGenerator = [AVAssetImageGenerator
assetImageGeneratorWithAsset:myAsset];

Float64 durationSeconds = CMTimeGetSeconds([myAsset duration]);
CMTime firstThird = CMTimeMakeWithSeconds(durationSeconds/3.0, 600);
CMTime secondThird = CMTimeMakeWithSeconds(durationSeconds*2.0/3.0, 600);
CMTime end = CMTimeMakeWithSeconds(durationSeconds, 600);
NSArray *times = [NSArray arrayWithObjects:[NSValue valueWithCMTime:kCMTimeZero],
                  [NSValue valueWithCMTime:firstThird], [NSValue
valueWithCMTime:secondThird],
                  [NSValue valueWithCMTime:end], nil];

[imageGenerator generateCGImagesAsynchronouslyForTimes:times
                completionHandler:^(CMTime requestedTime, CGImageRef image,
CMTime actualTime,
                                    AVAssetImageGeneratorResult result, NSError
 *error) {

                NSString *requestedTimeString = (NSString
*)CMTimeCopyDescription(NULL, requestedTime);
                NSString *actualTimeString = (NSString
*)CMTimeCopyDescription(NULL, actualTime);
                NSLog(@"Requested: %@; actual %@", requestedTimeString,
actualTimeString);
                [requestedTimeString release];
                [actualTimeString release];

                if (result == AVAssetImageGeneratorSucceeded) {
                    // この画像を使って必要な作業を実行する
                }

                if (result == AVAssetImageGeneratorFailed) {
                 NSLog(@"Failed with error: %@", [error localizedDescription]);
                }
                if (result == AVAssetImageGeneratorCancelled) {
                    NSLog(@"Canceled");
                }
  }];
```

画像シーケンスの生成をキャンセルするには、画像ジェネレータに`cancelAllCGImageGeneration`メッセージを送信します。

ムービーのトリミングとフォーマット変換

AVAssetExportSessionオブジェクトを使用して、ムービーのフォーマット変換やムービーのトリミングを行うことができます。エクスポートセッションは、アセットの非同期エクスポートを管理するコントローラオブジェクトです。セッションを初期化するときは、エクスポートするアセットと、適用するエクスポートオプションを指定するエクスポートプリセット（allExportPresetsを参照）の名前を使用します。次に、エクスポートセッションを設定して出力のURLとファイルタイプを指定します。また、メタデータや、出力をネットワークで使用できるように最適化するかどうかなど、その他のオプションも必要に応じて指定できます。

特定のプリセットを使用して特定のアセットをエクスポートできるかどうかを確認するには、次の例に示すようにexportPresetsCompatibleWithAsset:を使用します。

```
AVAsset *anAsset = <#Get an asset#>;
NSArray *compatiblePresets = [AVAssetExportSession
exportPresetsCompatibleWithAsset:anAsset];
if ([compatiblePresets containsObject:AVAssetExportPresetLowQuality]) {
    AVAssetExportSession *exportSession = [[AVAssetExportSession alloc]
        initWithAsset:anAsset presetName:AVAssetExportPresetLowQuality];
    // 実装が続く
}
```

セッションの設定を終えるには、出力のURLを指定します（URLはファイルURLでなければなりません）。 AVAssetExportSessionは出力のファイルタイプを、URLのパス拡張子から推測できます。しかし通常は、outputFileTypeで直接設定します。時間範囲、出力ファイルの長さの制限、エクスポートしたファイルをネットワークで使用できるように最適化するかどうか、ビデオコンポジションなど、追加のプロパティを指定することもできます。次の例は、timeRangeプロパティを使用してムービーをトリミングする方法を示しています。

```
    exportSession.outputURL = <#A file URL#>;
    exportSession.outputFileType = AVFileTypeQuickTimeMovie;

    CMTime start = CMTimeMakeWithSeconds(1.0, 600);
    CMTime duration = CMTimeMakeWithSeconds(3.0, 600);
    CMTimeRange range = CMTimeRangeMake(start, duration);
    exportSession.timeRange = range;
```

新しいファイルを作成するには、exportAsynchronouslyWithCompletionHandler:を呼び出します。エクスポート操作が終了すると、終了ハンドラブロックが呼び出されます。このハンドラの実装では、セッションのstatusを確認して、エクスポートが成功したか、失敗したか、またはキャンセルされたかを判定する必要があります。

```
    [exportSession exportAsynchronouslyWithCompletionHandler:^{

        switch ([exportSession status]) {
            case AVAssetExportSessionStatusFailed:
                NSLog(@"Export failed: %@", [[exportSession error]
localizedDescription]);
```

```
                break;
            case AVAssetExportSessionStatusCancelled:
                NSLog(@"Export canceled");
                break;
            default:
                break;
        }
        [exportSession release];
    }];
```

エクスポートをキャンセルするには、セッションに`cancelExport`メッセージを送信します。

既存のファイルに上書きしようとしたり、アプリケーションのサンドボックスの外部にファイルを書き込もうとしたりすると、エクスポートに失敗します。次の場合も失敗します。

- 電話の着信があった場合
- 現在のアプリケーションがバックグラウンドにあり、別のアプリケーションが再生を開始した場合

このような状況では、通常、エクスポートに失敗したことをユーザに通知し、ユーザがエクスポートを再開できるようにする必要があります。

アセットの読み取りと書き込み

iOS 4.1以降、アセットの内容に対して操作を実行するときは、`AVAssetReader`を使用します。たとえば、アセットのオーディオトラックを読み取って、波形のビジュアル表現を作成できます。

iOS 4.1以降で、サンプルバッファや静止画像などのメディアからアセットを作成するには、`AVAssetWriter`オブジェクトを使用します。

アセットリーダーオブジェクトとアセットライターオブジェクトの組み合わせを使用して、アセットをある表現から別の表現に変換できます。これらのオブジェクトを使用すると、`AVExportSession`を使用した場合より細かく変換を制御できます。たとえば、出力ファイルで表現するトラックを選んだり、独自の出力フォーマットを指定したり、変換処理中にアセットを変更したりできます。

Playback

アセットの再生を制御するには、AVPlayerオブジェクトを使用します。再生中は、AVPlayerItemオブジェクトを使用してアセット全体の表示状態を管理し、AVPlayerItemTrackを使用して個々のトラックの表示状態を管理します。ビデオを表示するには、AVPlayerLayerを使用します。

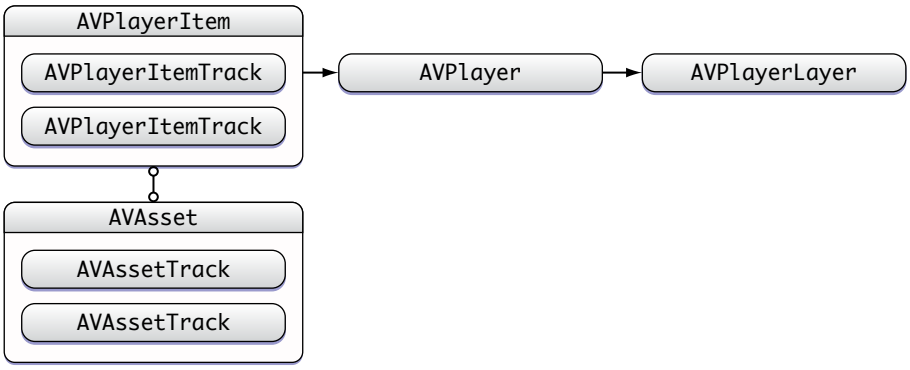
アセットの再生

プレーヤーは、アセットの再生を管理する（再生の開始と停止、特定の時間へのシークなど）コントローラオブジェクトです。1つのアセットを再生するには、AVPlayerのインスタンスを使用します。iOS 4.1以降では、AVQueuePlayerオブジェクトを使用して複数のアイテムを順に再生できます（AVQueuePlayerはAVPlayerのサブクラスです）。

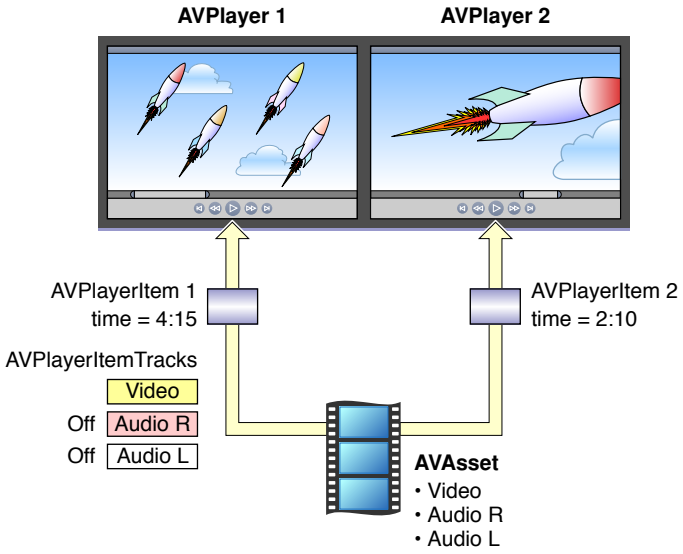
プレーヤーから再生の状態に関する情報が提供されるので、必要であれば、ユーザインターフェイスをプレーヤーの状態と同期させることができます。通常は、プレーヤーの出力を特別なCore Animationレイヤ（AVPlayerLayerまたはAVSynchronizedLayerのインスタンス）に送ります。レイヤの詳細については、『*Core Animation Programming Guide*』を参照してください。

複数のプレーヤーレイヤ： 1つのAVPlayerインスタンスから任意の数のAVPlayerLayerオブジェクトを作成できますが、画面上にビデオが表示されるのは直前に作成されたレイヤだけです。

最終的にアセットを再生する必要があっても、AVPlayerオブジェクトにはアセットを直接提供しません。代わりに、AVPlayerItemのインスタンスを提供します。プレーヤーアイテムは、そのアイテムが関連付けられたアセットの表示状態を管理します。プレーヤーアイテムには、アセット内のトラックに対応するプレーヤーアイテムトラック（AVPlayerItemTrackのインスタンス）が含まれています。



この抽象化によって、特定のアセットを複数のプレーヤーでレンダリング方法を変えながら同時に再生することができます。アイテムトラックを使用して、たとえば、再生中に特定のトラックを無効にすることができます（サウンドコンポーネントを再生したくない場合など）。



既存のアセットを使用してプレーヤーアイテムを初期化することもできますが、特定の場所でリソースを再生できるようにURLから直接プレーヤーアイテムを初期化することもできます（その場合、`AVPlayerItem`はリソースに合わせてアセットを作成および設定します）。ただし、`AVAsset`と同じように、プレーヤーアイテムを初期化しただけでは、必ずしもただちに再生できる状態にはなりません。（キー値監視を使用して）アイテムの`status`プロパティを監視することにより、再生できる状態になったかどうかを確認できます。

各種のアセットの処理

再生用のアセットの設定方法は、再生するアセットの種類によって異なる場合があります。大まかに言うと、2つの主なタイプがあります。1つはランダムアクセスが可能なファイルベースのアセット（ローカルファイル、カメラロール、メディアライブラリなどにあるファイル）で、もう1つはストリームベースのアセット（HTTPライブストリームフォーマット）です。

ファイルベースのアセットをロードして再生するには、次の手順に従います。

- `AVURLAsset`を使用してアセットを作成し、`loadValuesAsynchronouslyForKeys:completionHandler:`を使用してアセットのトラックをロードします。
- トラックがロードされたら、アセットを使用して`AVPlayerItem`のインスタンスを作成します。
- アイテムを`AVPlayer`のインスタンスに関連付けます。
- アイテムの`status`が再生可能な状態を示すまで待機します（通常は、キー値監視を使用して状態が変化したときに通知を受信します）。

この手法については、「すべてを組み合わせる：AVPlayerLayerを使用したビデオファイルの再生」（24 ページ）で具体的に示します。

再生用のHTTPライブストリームを作成して準備するには、URLを使用してAVPlayerItemのインスタンスを初期化します（HTTPライブストリーム内のメディアを表すためにAVAssetインスタンスを直接作成することはできません）。

```
NSURL *url = [NSURL URLWithString:@"<#Live stream URL#>];
// &lt;http://devimages.apple.com/iphone/samples/bipbop/bipbopall.m3u8&gt;にテストストリームがある
self.playerItem = [AVPlayerItem playerItemWithURL:url];
[playerItem addObserver:self forKeyPath:@"status" options:0
context:&ItemStatusContext];
self.player = [AVPlayer playerWithPlayerItem:playerItem];
```

プレーヤーアイテムをプレーヤーに関連付けると、再生の準備が始まります。再生の準備が完了すると、プレーヤーアイテムによってAVAssetインスタンスとAVAssetTrackインスタンスが作成されます。これらのインスタンスを使用してライブストリームのコンテンツを調べることができます。

ライブストリームを再生するだけの場合は、次のように簡単な方法でURLを使用してプレーヤーを直接作成できます。

```
self.player = [AVPlayer playerWithURL:<#Live stream URL#>];
[player addObserver:self forKeyPath:@"status" options:0
context:&PlayerStatusContext];
```

アセットやアイテムと同じように、プレーヤーを初期化しただけでは、ただちに再生できる状態にはなりません。プレーヤーのstatusプロパティを監視する必要があります。このプロパティの値は、再生できる状態になるとAVPlayerStatusReadyToPlayに変わります。また、ストリーム用に作成されたプレーヤーアイテムにアクセスする場合はcurrentItemプロパティを監視することもできます。

URLの種類がわからない場合は、次の手順に従います。

1. URLを使用してAVURLAssetを初期化し、そのtracksキーをロードします。

   トラックのロードに成功したら、アセットのプレーヤーアイテムを作成します。

2. 1に失敗したら、URLから直接AVPlayerItemを作成します。

   プレーヤーのstatusプロパティを監視して、再生可能になったかどうかを判定します。

どちらかの方法が成功すれば、プレーヤーアイテムをプレーヤーと関連付けることができます。

アイテムの再生

再生を開始するには、プレーヤーにplayメッセージを送信します。

```
- (IBAction)play:sender {
    [player play];
}
```

再生だけでなく、速度や再生ヘッドの位置など、再生に関連するさまざまな側面を管理することができます。プレーヤーの再生状態を監視することもできます。これは、たとえば、ユーザインターフェイスをアセットの表示状態と同期させる場合などに便利です（「再生の監視」 （21 ページ）を参照）。

再生速度の変更

再生速度を変更するには、プレーヤーのrateプロパティを設定します。

```
aPlayer.rate = 0.5;
aPlayer.rate = 2.0;
```

値1.0は、「現在のアイテム本来の速度での再生」を意味します。速度を0.0に設定することは、再生の一時停止と同じです。pauseを使用することもできます。

シーク：再生ヘッドの位置変更

再生ヘッドを特定の時間に移動するには、一般にseekToTime:を使用します。

```
CMTime fiveSecondsIn = CMTimeMake(5, 1);
[player seekToTime:fiveSecondsIn];
```

ただし、seekToTime:メソッドは正確さよりパフォーマンスを重視してチューニングされています。再生ヘッドを正確に移動する必要がある場合は、代わりにseekToTime:toleranceBefore:toleranceAfter:を使用します。

```
CMTime fiveSecondsIn = CMTimeMake(5, 1);
[player seekToTime:fiveSecondsIn toleranceBefore:kCMTimeZero
toleranceAfter:kCMTimeZero];
```

許容誤差としてゼロを使用すると、フレームワークが大量のデータをデコードしなければならなくなる可能性があります。ゼロを使用するのは、たとえば、厳密に制御する必要がある高度なメディア編集アプリケーションを作成する場合などに限定してください。

再生後は、プレーヤーのヘッドがアイテムの末尾にセットされ、その後はplayを呼び出しても何も起きません。再生ヘッドをアイテムの先頭に位置付けるには、アイテムからAVPlayerItemDidPlayToEndTimeNotificationを受信するように登録します。通知のコールバックメソッドで、引数kCMTimeZeroとともにseekToTime:を呼び出します。

```
// プレーヤーアイテムを作成した後で、通知センターに登録する
    [[NSNotificationCenter defaultCenter]
        addObserver:self
        selector:@selector(playerItemDidReachEnd:)
        name:AVPlayerItemDidPlayToEndTimeNotification
        object:<#The player item#>];

- (void)playerItemDidReachEnd:(NSNotification *)notification {
    [player seekToTime:kCMTimeZero];
}
```

複数のアイテムの再生

iOS 4.1以降では、AVQueuePlayerオブジェクトを使用して複数のアイテムを順に再生できます。AVQueuePlayerはAVPlayerのサブクラスです。次のようにプレーヤーアイテムの配列を使用してキュープレーヤーを初期化します。

```
NSArray *items = <#An array of player items#>;
AVQueuePlayer *queuePlayer = [[AVQueuePlayer alloc] initWithItems:items];
```

これで、AVPlayerオブジェクトと同じようにplayを使用してキューを再生できます。キュープレーヤーは各アイテムを順に再生します。次のアイテムをスキップしたい場合は、キュープレーヤーにadvanceToNextItemメッセージを送信します。

キューを変更するには、insertItem:afterItem:、removeItem:、およびremoveAllItemsを使用します。新しいアイテムを追加するときは、通常、canInsertItem:afterItem:を使用してキューに挿入できるかどうかを確認する必要があります。新しいアイテムをキューに追加できるかどうかを確認するには、次のように2番目の引数としてnilを渡します。

```
AVPlayerItem *anItem = <#Get a player item#>;
if ([queuePlayer canInsertItem:anItem afterItem:nil]) {
    [queuePlayer insertItem:anItem afterItem:nil];
}
```

再生の監視

プレーヤーおよび再生中のプレーヤーアイテムの表示状態に関するさまざまな側面を監視できます。これは、たとえば次のようにデベロッパが直接制御しない状態の変化に対して有効です。

- ユーザがマルチタスクを使用して別のアプリケーションに切り替えると、プレーヤーのrateプロパティが0.0になります。
- リモートメディアを再生している場合は、利用可能なデータが増えるにつれて、プレーヤーアイテムのloadedTimeRangesプロパティとseekableTimeRangesプロパティが変化します。

  これらのプロパティから、プレーヤーアイテムのタイムラインのどの部分が利用可能であるかがわかります。
- HTTPライブストリーム用のプレーヤーアイテムが作成されると、プレーヤーのcurrentItemプロパティが変化します。
- HTTPライブストリームの再生中は、プレーヤーアイテムのtracksプロパティが変化することがあります。

  この変化は、ストリームのコンテンツに複数のエンコードが使用されている場合に発生します。つまり、プレーヤーがエンコードを切り替えると、トラックが変化します。
- 何らかの理由で再生に失敗すると、プレーヤーまたはプレーヤーアイテムのstatusが変化することがあります。

キー値監視を使用して、これらのプロパティ値の変更を監視できます。

重要： KVO変更通知を登録し、メインスレッドのKVO変更通知を登録解除する必要があります。これにより、別のスレッドで変更が行われた場合に部分的な通知を受信する可能性を避けることができます。AV Foundationは、別のスレッドで変更操作が行われた場合でも、メインスレッド上で`observeValueForKeyPath:ofObject:change:context:`を呼び出します。

ステータスの変化に対する対応

プレーヤーまたはプレーヤーアイテムのステータスが変化すると、キー値監視の変更通知が発行されます。何らかの理由（たとえば、メディアサービスがリセットされた場合など）でオブジェクトを再生できない場合は、ステータスが状況に応じて`AVPlayerStatusFailed`または`AVPlayerItemStatusFailed`に変化します。このような場合は、オブジェクトの`error`プロパティの値が、オブジェクトが再生できなくなった理由を記述するエラーオブジェクトに変更されます。

AV Foundationは通知が送信されたときのスレッドを明示しません。ユーザインターフェイスを更新したい場合は、関連するコードを必ずメインスレッド上で呼び出す必要があります。次の例では、メインスレッド上でコードを実行するために`dispatch_async(3) Mac OS X Manual Page`を使用しています。

```
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
                        change:(NSDictionary *)change context:(void *)context {

    if (context == <#Player status context#>) {
        AVPlayer *thePlayer = (AVPlayer *)object;
        if ([thePlayer status] == AVPlayerStatusFailed) {
            NSError *error = [<#The AVPlayer object#> error];
            // エラーへの対応：警告シートの表示など
            return;
        }
        // 必要であれば、ほかのステータス変更を処理する
    }
    // 必要であれば、ほかの変更通知を処理する
    [super observeValueForKeyPath:keyPath ofObject:object
           change:change context:context];
    return;
}
```

視覚表示の準備状況の追跡

`AVPlayerLayer`オブジェクトの`readyForDisplay`プロパティを監視することで、ユーザに表示するコンテンツがレイヤに存在するかどうかを知ることができます。特に、ユーザに表示するコンテンツがある場合にのみプレーヤーレイヤをレイヤツリーに挿入し、新しいコンテンツへの移行を実行できます。

時間の追跡

AVPlayerオブジェクトに含まれる再生ヘッドの位置の変化を追跡するには、addPeriodicTimeObserverForInterval:queue:usingBlock:またはaddBoundaryTimeObserverForTimes:queue:usingBlock:を使用します。これによって、たとえば、ユーザインターフェイスに表示した経過時間や残り時間の情報を更新したり、その他のユーザインターフェイス同期処理を実行したりできます。

- addPeriodicTimeObserverForInterval:queue:usingBlock:を使用すると、時間がジャンプした場合や、再生の開始時または終了時に、指定したブロックが指定した間隔で呼び出されます。
- addBoundaryTimeObserverForTimes:queue:usingBlock:を使用すると、NSValueオブジェクトに格納されたCMTimeの配列を渡すことができます。配列に含まれる時間になると、指定したブロックが呼び出されます。

これらのメソッドは、いずれもオブザーバとして機能する不透過オブジェクトを返します。プレーヤーから時間監視ブロックを呼び出す必要がある間は、返されたオブジェクトを保持する必要があります。また、これらのメソッドの呼び出しと、対応するremoveTimeObserver:の呼び出しのバランスを取る必要があります。

AV Foundationでは、これらのメソッドのどちらを使用した場合でも、渡された間隔または境界ごとにブロックが呼び出されるという保証はありません。AV Foundationは、直前に呼び出されたブロックの実行が完了しなかった場合、ブロックを呼び出しません。したがって、ブロック内で実行する作業がシステムに過大な負荷を掛けないようにする必要があります。

```
// 前提となるプロパティ：@property (retain) id playerObserver;

Float64 durationSeconds = CMTimeGetSeconds([<#An asset#> duration]);
CMTime firstThird = CMTimeMakeWithSeconds(durationSeconds/3.0, 1);
CMTime secondThird = CMTimeMakeWithSeconds(durationSeconds*2.0/3.0, 1);
NSArray *times = [NSArray arrayWithObjects:[NSValue valueWithCMTime:firstThird],
 [NSValue valueWithCMTime:secondThird], nil];

self.playerObserver = [<#A player#> addBoundaryTimeObserverForTimes:times
queue:NULL usingBlock:^{

    NSString *timeDescription = (NSString *)CMTimeCopyDescription(NULL,
[self.player currentTime]);
    NSLog(@"Passed a boundary at %@", timeDescription);
    [timeDescription release];
}];
```

アイテムの末尾への到達

プレーヤーアイテムの再生が完了したときに、AVPlayerItemDidPlayToEndTimeNotificationの通知を受信するように登録できます。

```
[[NSNotificationCenter defaultCenter] addObserver:<#The observer, typically
self#>
                                         selector:@selector(<#The selector
name#>)
```

```
name:AVPlayerItemDidPlayToEndTimeNotification
                                            object:<#A player item#>];
```

すべてを組み合わせる：AVPlayerLayerを使用したビデオファイルの再生

次の簡単なコード例は、AVPlayerオブジェクトを使用してビデオファイルを再生する方法を示しています。具体的な方法は次のとおりです。

- AVPlayerLayerレイヤを使用するようにビューを設定する
- AVPlayerオブジェクトを作成する
- ファイルベースのアセット用のAVPlayerItemオブジェクトを作成し、キー値監視を使用してそのステータスを監視する
- 再生できる状態になったアイテムに対応してボタンを有効にする
- アイテムを再生し、プレーヤーのヘッドを先頭に戻す

注： この例では最も重要なコードに注目するため、完全なアプリケーションからメモリ管理や（キー値監視用または通知センター用の）オブザーバの登録解除など、いくつかの部分を省略しています。AV Foundationを使用するには、欠落した部分を推測できるように、Cocoaに関する十分な経験を積んでいることが期待されます。

再生の概念については、「アセットの再生」（17 ページ）を参照してください。

プレーヤービュー

アセットのビジュアルコンポーネントを再生するには、AVPlayerの出力の送り先となるAVPlayerLayerレイヤを含むビューが必要です。これに対応するため、次のようにUIViewの簡単なサブクラスを作成します。

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
@interface PlayerView : UIView {
}
@property (nonatomic, retain) AVPlayer *player;
@end

@implementation PlayerView
+ (Class)layerClass {
    return [AVPlayerLayer class];
}
- (AVPlayer*)player {
    return [(AVPlayerLayer *)[self layer] player];
}
- (void)setPlayer:(AVPlayer *)player {
```

```
    [(AVPlayerLayer *)[self layer] setPlayer:player];
}
@end
```

簡単なビューコントローラ

次のように宣言された簡単なView Controllerがあるとします。

```
@class PlayerView;
@interface PlayerViewController : UIViewController {
}
@property (nonatomic, retain) AVPlayer *player;
@property (retain) AVPlayerItem *playerItem;
@property (nonatomic, retain) IBOutlet PlayerView *playerView;
@property (nonatomic, retain) IBOutlet UIButton *playButton;
- (IBAction)loadAssetFromFile:sender;
- (IBAction)play:sender;
- (void)syncUI;
@end
```

`syncUI`メソッドは、次のようにボタンの状態をプレーヤーの状態と同期させます。

```
- (void)syncUI {
    if ((player.currentItem != nil) &&
        ([player.currentItem status] == AVPlayerItemStatusReadyToPlay)) {
        playButton.enabled = YES;
    }
    else {
        playButton.enabled = NO;
    }
}
```

View Controllerの`viewDidLoad`メソッドで`syncUI`を呼び出すことにより、ビューが最初に表示されたときに一貫性のあるユーザインターフェイスを提供します。

```
- (void)viewDidLoad {
    [super viewDidLoad];
    [self syncUI];
}
```

残りのセクションでは、その他のプロパティおよびメソッドについて説明します。

アセットの作成

`AVURLAsset`を使用してURLからアセットを作成します。ただし、アセットを作成しただけでは、必ずしも使用できる状態にはなりません。アセットを使用するには、そのトラックをロードする必要があります。現在のスレッドがブロックされないように、`loadValuesAsynchronouslyForKeys:completionHandler:`を使用して非同期にアセットのトラックをロードします（次の例では、プロジェクトに適切なビデオリソースが含まれているとします）。

```
- (IBAction)loadAssetFromFile:sender {

    NSURL *fileURL = [[NSBundle mainBundle]
        URLForResource:<#@"VideoFileName"#> withExtension:<#@"extension"#>];
```

```
    AVURLAsset *asset = [AVURLAsset URLAssetWithURL:fileURL options:nil];
    NSString *tracksKey = @"tracks";

    [asset loadValuesAsynchronouslyForKeys:[NSArray arrayWithObject:tracksKey]
completionHandler:
     ^{
         // ここに完了ブロックを記述する
     }];
}
```

完了ブロックでは、アセット用のAVPlayerItemのインスタンスを作成し、それをプレーヤービューのプレーヤーとして設定します。アセットの作成と同じように、プレーヤーアイテムを作成しただけでは、ただちに使用できる状態にはなりません。再生できる状態になったかどうかを判定するには、アイテムのstatusプロパティを監視します。アイテムをプレーヤーに関連付けたときに、アイテムを再生するための準備が開始されます。

```
// キー値監視コンテキストのために次の定数を定義する
static const NSString *ItemStatusContext;

// 完了ハンドラブロック
          dispatch_async(dispatch_get_main_queue(),
             ^{
                 NSError *error = nil;
                 AVKeyValueStatus status = [asset statusOfValueForKey:tracksKey
 error:&error];

                 if (status == AVKeyValueStatusLoaded) {
                     self.playerItem = [AVPlayerItem playerItemWithAsset:asset];
                     [playerItem addObserver:self forKeyPath:@"status"
                                     options:0 context:&ItemStatusContext];
                     [[NSNotificationCenter defaultCenter] addObserver:self

selector:@selector(playerItemDidReachEnd:)

name:AVPlayerItemDidPlayToEndTimeNotification

object:playerItem];
                     self.player = [AVPlayer playerWithPlayerItem:playerItem];
                     [playerView setPlayer:player];
                 }
                 else {
                     // エラーを適切に処理しなければならない
                     NSLog(@"The asset's tracks were not loaded:\n%@", [error
localizedDescription]);
                 }
             });
```

プレーヤーアイテムのステータス変更に対する対応

プレーヤーアイテムのステータスが変化すると、ビューコントローラはキー値監視の変更通知を受け取ります。AV Foundationは通知が送信されたときのスレッドを明示しません。ユーザインターフェイスを更新したい場合は、関連するコードを必ずメインスレッド上で呼び出す必要があります。次の例では、メインスレッド上でメッセージをキューに入れ、ユーザインターフェイスを同期するために、dispatch_async(3) Mac OS X Manual Pageを使用しています。

```
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
                        change:(NSDictionary *)change context:(void *)context {

    if (context == &ItemStatusContext) {
        dispatch_async(dispatch_get_main_queue(),
                       ^{
                           [self syncUI];
                       });
        return;
    }
    [super observeValueForKeyPath:keyPath ofObject:object
           change:change context:context];
    return;
}
```

アイテムの再生

アイテムの再生は簡単です。プレーヤーにplayメッセージを送信します。

```
- (IBAction)play:sender {
    [player play];
}
```

ただし、これではアイテムが1回再生されるだけです。再生後は、プレーヤーのヘッドがアイテムの末尾にセットされ、その後はplayを呼び出しても何も起きません。再生ヘッドをアイテムの先頭に位置付けるには、アイテムからAVPlayerItemDidPlayToEndTimeNotificationを受信するように登録します。通知のコールバックメソッドで、引数kCMTimeZeroとともにseekToTime:を呼び出します。

```
// プレーヤーアイテムを作成した後で、通知センターに登録する
    [[NSNotificationCenter defaultCenter]
        addObserver:self
        selector:@selector(playerItemDidReachEnd:)
        name:AVPlayerItemDidPlayToEndTimeNotification
        object:[player currentItem]];

- (void)playerItemDidReachEnd:(NSNotification *)notification {
    [player seekToTime:kCMTimeZero];
}
```

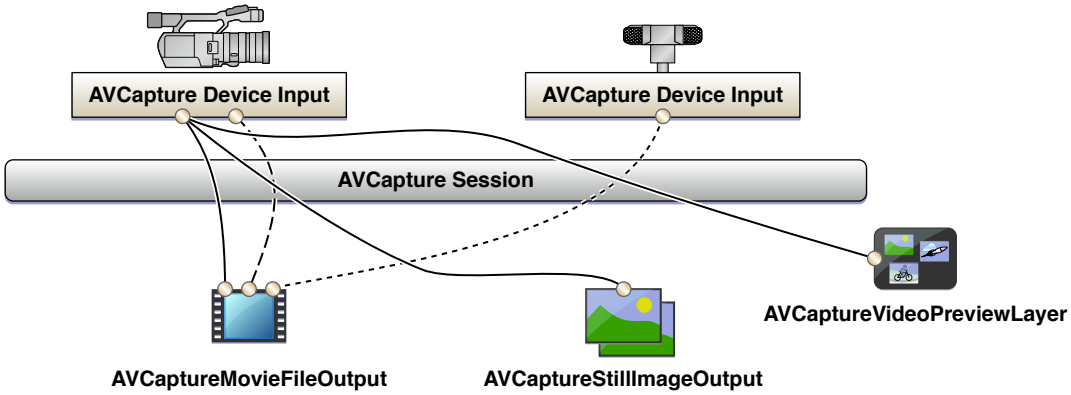
メディアキャプチャ

カメラやマイクなどのデバイスからのキャプチャを処理するには、入力と出力を表すオブジェクトを組み合わせ、AVCaptureSessionのインスタンスを使用して各オブジェクト間のデータの流れを調整します。少なくとも次のインスタンスが必要です。

- カメラやマイクなどの入力デバイスを表すAVCaptureDeviceのインスタンス
- 入力デバイスのポートを設定するAVCaptureInputの具象サブクラスのインスタンス
- ムービーファイルや静止画像への出力を管理するAVCaptureOutputの具象サブクラスのインスタンス
- 入力から出力へのデータの流れを調整するAVCaptureSessionの具象サブクラスのインスタンス

カメラで録画している内容をユーザに表示するには、AVCaptureVideoPreviewLayer（CALayerのサブクラス）のインスタンスを使用します。

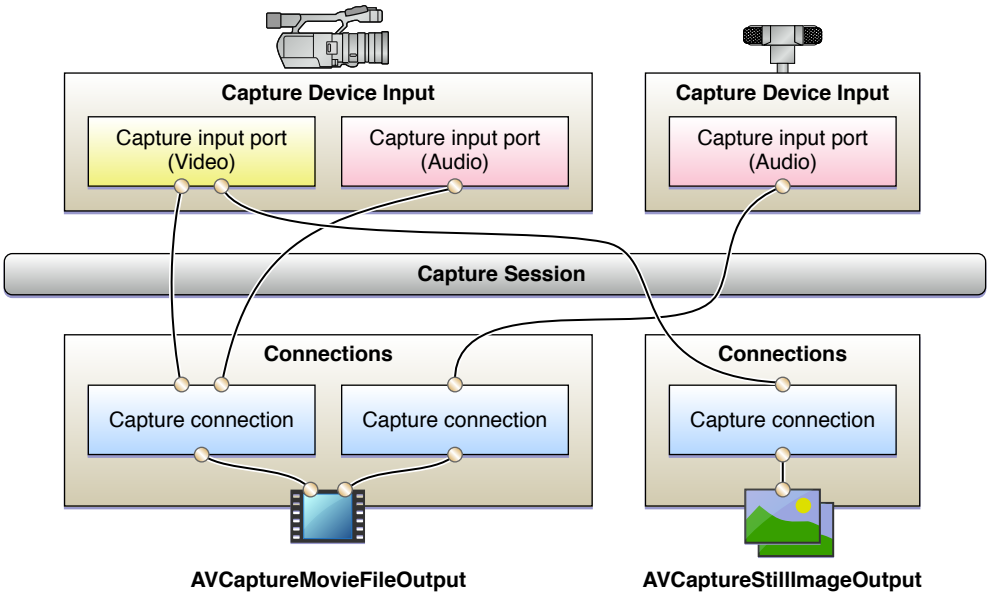
次のように、複数の入力と出力を設定し、それらを1つのセッションによって調整できます。



多くのアプリケーションは、このレベルで十分対応できます。しかし、操作の種類によっては（たとえば、オーディオチャンネルの出力レベルを監視したい場合など）、入力デバイスの各ポートの表現方法や各ポートの出力への接続方法を考慮する必要があります。

キャプチャセッションにおけるキャプチャ入力とキャプチャ出力の接続は、AVCaptureConnectionオブジェクトで表されます。キャプチャ入力（AVCaptureInputのインスタンス）には、1つ以上の入力ポート（AVCaptureInputPortのインスタンス）があります。キャプチャ出力（AVCaptureOutputのインスタンス）は、1つ以上のソースからデータを受け取ります（たとえば、AVCaptureMovieFileOutputオブジェクトはビデオデータとオーディオデータの両方を受け取ります）。

セッションに入力または出力を追加すると、そのセッションは互換性のあるすべてのキャプチャ入力ポートとキャプチャ出力の接続を「積極的に」作成します。キャプチャ入力とキャプチャ出力の接続は、AVCaptureConnectionオブジェクトで表されます。



キャプチャ接続を使用して、特定の入力から特定の出力へのデータの流れを有効または無効にすることができます。また、接続を使用して、オーディオチャンネルの平均出力レベルやピーク出力レベルを監視することもできます。

キャプチャセッションを使用したデータの流れの調整

`AVCaptureSession`オブジェクトは、データキャプチャの管理に使用する中心的な調整オブジェクトです。このオブジェクトのインスタンスを使用して、AV入力デバイスから出力へのデータの流れを調整します。必要なキャプチャデバイスと出力をセッションに追加した後、データの流れを開始するときにはセッションに`startRunning`メッセージを、録音・録画を終了するときには`stopRunning`メッセージを送信します。

```
AVCaptureSession *session = [[AVCaptureSession alloc] init];
// 入力と出力を追加する
[session startRunning];
```

セッションの設定

画像の品質や解像度を指定するには、セッションの**プリセット**を使用します。プリセットは、考えられるいくつかの設定を識別する定数です。実際の設定は次のようにデバイスごとに異なる場合があります。

シンボル	解像度	備考
`AVCaptureSessionPresetHigh`	高	最高の録画品質 デバイスによって異なる

シンボル	解像度	備考
AVCaptureSessionPresetMedium	中	WiFi共有に適する 実際の値は変わる可能性がある
AVCaptureSessionPresetLow	低	3G共有に適する 実際の値は変わる可能性がある
AVCaptureSessionPreset640x480	640x480	VGA
AVCaptureSessionPreset1280x720	1280x720	720p HD
AVCaptureSessionPresetPhoto	写真	写真用の最大解像度 ビデオ出力ではサポートされない

これらのプリセットが表す各種デバイスの実際の値については、「ムービーファイルへの保存」（38 ページ）および「静止画像のキャプチャ」（41 ページ）を参照してください。

サイズ固有の設定を行う場合は、設定の前に、次のようにそのサイズがサポートされていることを確認する必要があります。

```
if ([session canSetSessionPreset:AVCaptureSessionPreset1280x720]) {
    session.sessionPreset = AVCaptureSessionPreset1280x720;
}
else {
    // エラーを処理する
}
```

多くの場合、セッションと各種の入力および出力をすべて同時に作成します。ただし、異なる入力デバイスが利用可能になった場合や、ユーザからの要求に対応する場合などは、実行中のセッションを再設定することもできます。問題は、設定を1つずつ変更した場合、新しい設定が既存の設定と矛盾するおそれがあることです。この問題に対処するには、`beginConfiguration`と`commitConfiguration`を使用して複数の設定操作をアトミックな更新操作にまとめて一括処理します。`beginConfiguration`を呼び出した後は、出力を追加または削除したり、`sessionPreset`を変更したり、入力または出力の個々のプロパティを設定したりできます。実際の変更は、`commitConfiguration`を呼び出すまで行われません。このメソッドを呼び出した時点で、変更がまとめて適用されます。

```
[session beginConfiguration];
// 既存のキャプチャデバイスを削除する
// 新しいキャプチャデバイスを追加する
// プリセットをリセットする
[session commitConfiguration];
```

キャプチャセッションの状態監視

キャプチャセッションは通知を送信します。この通知を監視することで、実行の開始、停止、中断などを知ることができます。また、ランタイムエラーが発生した場合に`AVCaptureSessionRuntimeErrorNotification`を受信するように登録することもできます。セッションが実行中かどうかを確認するには、セッションの`running`プロパティを調べます。セッションが中断されたかどうかを確認するには、セッションの`interrupted`プロパティを調べます。

入力デバイスを表すAVCaptureDeviceオブジェクト

AVCaptureDeviceオブジェクトは、AVCaptureSessionオブジェクトに入力データ（オーディオやビデオなど）を提供する物理的なキャプチャデバイスを抽象化したものです。オブジェクトは入力デバイスごとに1つずつ存在します。たとえば、iPhone 3GSには、カメラ用のビデオ入力が1つと、マイク用のオーディオ入力が1つあります。iPhone 4には、ビデオ入力が2つ（正面のカメラ用と背面のカメラ用）と、マイク用のオーディオ入力が1つあります。

どのキャプチャデバイスを現在使用できるかを調べるには、AVCaptureDeviceクラスのメソッドdevicesおよびdevicesWithMediaType:を使用します。必要な場合は、デバイスが持っている機能を調べることもできます（「デバイスキャプチャの設定」（33 ページ）を参照）。ただし、利用可能なデバイスのリストは変わる可能性があります。（別のアプリケーションが使用しているために）現在のデバイスが使用できなくなったり、（別のアプリケーションが手放したために）新しいデバイスが使用できるようになったりする可能性があります。利用可能なデバイスのリストが変更されたことを知るには、AVCaptureDeviceWasConnectedNotificationおよびAVCaptureDeviceWasDisconnectedNotificationの 通知を受信するように登録する必要があります。

キャプチャセッションにデバイスを追加するには、キャプチャ入力を使用します（「キャプチャ入力を使用したセッションへのキャプチャデバイスの追加」（37 ページ）を参照）。

デバイスの特性

デバイスに対してさまざまな特性を問い合わせることができます。特定のメディアタイプを提供しているかを確認するには、hasMediaType:を使用します。特定のキャプチャセッションプリセットをサポートしているかを確認するには、supportsAVCaptureSessionPreset:を使用します。ユーザに情報を提供するために、キャプチャデバイスの位置（ユーザが使用しているユニットの前面と背面のどちらにあるか）とローカライズされたデバイス名を特定できます。これは、キャプチャデバイスのリストを表示して、ユーザがデバイスを選べるようにしたい場合に便利です。

次のコード例では、利用可能なすべてのデバイスに対して反復処理を実行し、デバイス名とユニット上の位置（ビデオデバイスの場合）をログに記録しています。

```
NSArray *devices = [AVCaptureDevice devices];

for (AVCaptureDevice *device in devices) {

    NSLog(@"Device name: %@", [device localizedName]);

    if ([device hasMediaType:AVMediaTypeVideo]) {

        if ([device position] == AVCaptureDevicePositionBack) {
            NSLog(@"Device position : back");
        }
        else {
            NSLog(@"Device position : front");
        }
    }
}
```

さらに、デバイスの機種IDや固有IDを特定することもできます。

デバイスキャプチャの設定

デバイスの機能はデバイスごとに異なります。たとえば、複数のフォーカスモードまたはフラッシュモードをサポートするデバイスや、関心のある場所へのフォーカスをサポートするデバイスがあります。

機能	iPhone 3G	iPhone 3GS	iPhone 4（背面）	iPhone 4（前面）
フォーカスモード	×	○	○	×
関心のある場所へのフォーカス	×	○	○	×
露出モード	○	○	○	○
関心のある場所の露出	×	○	○	○
ホワイトバランスモード	○	○	○	○
フラッシュモード	×	×	○	×
トーチモード	×	×	○	×

次のコードは、トーチモードを備え、特定のキャプチャセッションプリセットをサポートするビデオ入力デバイスを見つける方法を示しています。

```
NSArray *devices = [AVCaptureDevice devicesWithMediaType:AVMediaTypeVideo];
NSMutableArray *torchDevices = [[NSMutableArray alloc] init];

for (AVCaptureDevice *device in devices) {
    [if ([device hasTorch] &&
         [device supportsAVCaptureSessionPreset:AVCaptureSessionPreset640x480])
 {
        [torchDevices addObject:device];
    }
}
```

条件を満たすデバイスが複数見つかった場合は、使用するデバイスをユーザに選ばせることもできます。ユーザに対してデバイスの説明を表示するには、デバイスの`localizedName`プロパティを使用します。

異なる複数の機能を同じような方法で使用します。特定のモードを指定するための定数があり、特定のモードをサポートしているかどうかをデバイスに問い合わせることができます。いくつかのケースでは、プロセスを監視して、機能が変更されたかどうかを知ることができます。どのケースでも、「デバイスの設定」（36 ページ）で説明しているように、特定の機能のモードを変更する前にデバイスをロックする必要があります。

注： フォーカスモードと露出モードは相互に排他的であるため、関心のある場所へのフォーカスと関心のある場所の露出も相互に排他的です。

フォーカスモード

次の3つのフォーカスモードがあります。

- `AVCaptureFocusModeLocked`：焦点距離が固定されます。

  これは、ユーザがシーンを作ってから焦点をロックできるようにする場合に便利です。

- `AVCaptureFocusModeAutoFocus`：カメラがシングルスキャンフォーカスを実行してからロック状態に戻ります。

  これは、フォーカスの対象を選択し、対象がシーンの中心から外れても対象への焦点を維持したい場合に適しています。

- `AVCaptureFocusModeContinuousAutoFocus`：カメラが必要に応じて継続的にオートフォーカスを実行します。

デバイスが特定のフォーカスモードをサポートするかどうかを確認するには、`isFocusModeSupported:`メソッドを使用します。モードを設定するには、`focusMode`プロパティを使用します。

さらに、関心のある場所へのフォーカスをデバイスがサポートしている場合もあります。サポートしているかどうかは、`focusPointOfInterestSupported`を使用して確認します。サポートしている場合は、`focusPointOfInterest`を使用して焦点を設定します。`CGPoint`を渡します。*ホームボタンが右側になる横長モードでは*、`{0,0}`が画像領域の左上を表し、`{1,1}`が右下を表します。これは、デバイスが縦長モードであっても同じです。

デバイスが現在焦点を合わせているかどうかを確認するには、`adjustingFocus`プロパティを使用します。キー値監視を使用してこのプロパティを監視することにより、デバイスによるフォーカスの開始と停止を知ることができます。

フォーカスモードの設定を変更した場合、次のようにすればデフォルト設定に戻ります。

```
if ([currentDevice isFocusModeSupported:AVCaptureFocusModeContinuousAutoFocus])
 {
    CGPoint autofocusPoint = CGPointMake(0.5f, 0.5f);
    [currentDevice setFocusPointOfInterest:autofocusPoint];
    [currentDevice setFocusMode:AVCaptureFocusModeContinuousAutoFocus];
}
```

露出モード

次の2つの露出モードがあります。

- `AVCaptureExposureModeLocked`：露出モードが固定されます。

- `AVCaptureExposureModeAutoExpose`：カメラが必要に応じて継続的に露出レベルを変更します。

デバイスが特定の露出モードをサポートするかどうかを確認するには、`isExposureModeSupported:`メソッドを使用します。モードを設定するには、`exposureMode`プロパティを使用します。

さらに、デバイスが関心のある場所の露出をサポートしている場合もあります。サポートしているかどうかは、`exposurePointOfInterestSupported`を使用して確認します。サポートしている場合は、`exposurePointOfInterest`を使用して露出を設定します。`CGPoint`を渡します。*ホームボタンが右側になる横長モードでは*、`{0,0}`が画像領域の左上を表し、`{1,1}`が右下を表します。これは、デバイスが縦長モードであっても同じです。

デバイスが現在露出の設定を変更しているかどうかを確認するには、adjustingExposureプロパティを使用します。キー値監視を使用してこのプロパティを監視することにより、デバイスによる露出の設定変更の開始と停止を知ることができます。

露出の設定を変更しても、次のようにすればデフォルト設定に戻ります。

```
if ([currentDevice
isExposureModeSupported:AVCaptureExposureModeContinuousAutoExposure]) {
    CGPoint exposurePoint = CGPointMake(0.5f, 0.5f);
    [currentDevice setExposurePointOfInterest:exposurePoint];
    [currentDevice setExposureMode:AVCaptureExposureModeContinuousAutoExposure];
}
```

フラッシュモード

次の3つのフラッシュモードがあります。

- AVCaptureFlashModeOff：フラッシュが点灯しません。
- AVCaptureFlashModeOn：フラッシュが常に点灯します。
- AVCaptureFlashModeAuto：フラッシュが必要に応じて点灯します。

デバイスにフラッシュ機能があるかを確認するには、hasFlashを使用します。デバイスが特定のフラッシュモードをサポートするかを確認するには、isFlashModeSupported:メソッドを使用します。モードを設定するには、flashModeプロパティを使用します。

トーチモード

トーチモードでは、ビデオキャプチャに照明を当てるために、カメラのフラッシュが低電力で点灯したままになります。次の3つのトーチモードがあります。

- AVCaptureTorchModeOff：トーチが常にオフになります。
- AVCaptureTorchModeOn：トーチが常にオンになります。
- AVCaptureTorchModeAuto：必要に応じてトーチのオンとオフが切り替わります。

デバイスにフラッシュ機能があるかを確認するには、hasTorchを使用します。デバイスが特定のフラッシュモードをサポートするかを確認するには、isTorchModeSupported:メソッドを使用します。モードを設定するには、torchModeプロパティを使用します。

トーチがあるデバイスでは、デバイスが実行中のキャプチャセッションに関連付けられている場合にのみトーチがオンになります。

ホワイトバランス

次の2つのホワイトバランスモードがあります。

- AVCaptureWhiteBalanceModeLocked：ホワイトバランスモードが固定されます。

- AVCaptureWhiteBalanceModeContinuousAutoWhiteBalance：カメラが必要に応じて継続的にホワイトバランスを変更します。

デバイスが特定のホワイトバランスモードをサポートするかを確認するには、isWhiteBalanceModeSupported:メソッドを使用します。モードを設定するには、whiteBalanceModeプロパティを使用します。

デバイスが現在ホワイトバランスの設定を変更しているかを確認するには、adjustingWhiteBalanceプロパティを使用します。キー値監視を使用してこのプロパティを監視することにより、デバイスによるホワイトバランスの設定変更の開始と停止を知ることができます。

デバイスの設定

デバイスのキャプチャプロパティを設定するには、最初にlockForConfiguration:を使用してデバイスのロックを取得する必要があります。これにより、ほかのアプリケーションの設定と矛盾する変更を避けることができます。次のコードは、デバイスのフォーカスモードを変更する方法を示しています。具体的には、最初にモードがサポートされているかどうかを確認し、次にデバイスをロックして再設定できるようにします。ロックを取得した場合にのみフォーカスモードが変更され、その後ただちにロックが解放されます。

```
if ([device isFocusModeSupported:AVCaptureFocusModeLocked]) {
    NSError *error = nil;
    if ([device lockForConfiguration:&error]) {
        device.focusMode = AVCaptureFocusModeLocked;
        [device unlockForConfiguration];
    }
    else {
        // 必要に応じてエラーに対応する
```

デバイスのロックは、設定可能なデバイスプロパティを変更されないようにする必要がある場合にのみ保持してください。デバイスのロックを不必要に保持すると、そのデバイスを共有するほかのアプリケーションのキャプチャ品質が低下することがあります。

デバイスの切り替え

ユーザが入力デバイスを切り替えられるようにしたい場合があります。たとえば、iPhone 4では前面カメラから背面カメラに切り替えることができます。セッションの実行中もセッションを再設定できますが、処理の中断や不連続を避けるため、beginConfigurationおよびcommitConfigurationを使用して設定変更をひとまとめにする必要があります。

```
AVCaptureSession *session = <#A capture session#>;
[session beginConfiguration];

[session removeInput:frontFacingCameraDeviceInput];
[session addInput:backFacingCameraDeviceInput];

[session commitConfiguration];
```

一番外側のcommitConfigurationが呼び出されると、すべての変更がまとめて行われます。これによって切り替えをスムーズに実行できます。

キャプチャ入力を使用したセッションへのキャプチャデバイスの追加

キャプチャデバイスをキャプチャセッションに追加するには、AVCaptureDeviceInput（AVCaptureInput抽象クラスの具象サブクラス）のインスタンスを使用します。キャプチャデバイス入力は、デバイスのポートを管理します。

```
NSError *error = nil;
AVCaptureDeviceInput *input =
        [AVCaptureDeviceInput deviceInputWithDevice:device error:&error];
if (!input) {
    // エラーを適切に処理する
}
```

セッションに入力を追加するには、addInput:を使用します。必要な場合は、canAddInput:を使用して、キャプチャ入力が既存のセッションに対応しているかどうかを確認します。

```
AVCaptureSession *captureSession = <#Get a capture session#>;
AVCaptureDeviceInput *captureDeviceInput = <#Get a capture device input#>;
if ([captureSession canAddInput:captureDeviceInput]) {
    [captureSession addInput:captureDeviceInput];
}
else {
    // エラーを処理する
}
```

実行中のセッションを再設定する方法の詳細については、「セッションの設定」（30 ページ）を参照してください。

AVCaptureInputは、メディアデータの1つ以上のストリームを提供します。たとえば、入力デバイスはオーディオデータとビデオデータの両方を提供できます。入力によって提供される各メディアストリームは、AVCaptureInputPortオブジェクトで表されます。キャプチャセッションは、AVCaptureConnectionオブジェクトを使用して一連のAVCaptureInputPortオブジェクトと1つのAVCaptureOutputとのマッピングを定義します。

キャプチャ出力を使用したセッションからの出力の取得

キャプチャセッションから出力を取得するには、1つ以上の出力を追加します。出力は、AVCaptureOutputの具象サブクラスのインスタンスです。

- AVCaptureMovieFileOutputは、ムービーファイルに出力する場合に使用します。
- AVCaptureVideoDataOutputは、キャプチャ中のビデオのフレームを処理する場合に使用します。
- AVCaptureAudioDataOutputは、キャプチャ中のオーディオデータを処理する場合に使用します。
- AVCaptureStillImageOutputは、付随するメタデータを使用して静止画像をキャプチャする場合に使用します。

キャプチャセッションに出力を追加するには、addOutput:を使用します。canAddOutput:を使用して、キャプチャ出力が既存のセッションに対応しているかどうかを確認します。セッションの実行中も、必要に応じて出力を追加または削除できます。

```
AVCaptureSession *captureSession = <#Get a capture session#>;
AVCaptureMovieFileOutput *movieInput = <#Create and configure a movie output#>;
if ([captureSession canAddOutput:movieInput]) {
    [captureSession addOutput:movieInput];
}
else {
    // エラーを処理する
}
```

ムービーファイルへの保存

ムービーデータをファイルに保存するには、AVCaptureMovieFileOutputオブジェクトを使用します（AVCaptureMovieFileOutputは、ほとんどの基本動作を定義するAVCaptureFileOutputの具象サブクラスです）。最大録画時間や最大ファイルサイズなど、ムービーファイル出力のさまざまな側面を設定できます。ディスクの残り容量が少なくなった場合に録画を禁止することもできます。

```
AVCaptureMovieFileOutput *aMovieFileOutput = [[AVCaptureMovieFileOutput alloc]
 init];
CMTime maxDuration = <#Create a CMTime to represent the maximum duration#>;
aMovieFileOutput.maxRecordedDuration = maxDuration;
aMovieFileOutput.minFreeDiskSpaceLimit = <#An appropriate minimum given the
quality of the movie format and the duration#>;
```

出力の解像度とビットレートは、キャプチャセッションのsessionPresetによって異なります。通常、ビデオエンコードはH.264、オーディオエンコードはAACです。実際の値は、次の表に示すようにデバイスによって異なります。

プリセット	iPhone 3G	iPhone 3GS	iPhone 4（背面）	iPhone 4（前面）
高	ビデオなし Apple Lossless	640x480 3.5 mbps	1280x720 10.5 mbps	640x480 3.5 mbps
中	ビデオなし Apple Lossless	480x360 700 kbps	480x360 700 kbps	480x360 700 kbps
低	ビデオなし Apple Lossless	192x144 128 kbps	192x144 128 kbps	192x144 128 kbps
640x480	ビデオなし Apple Lossless	640x480 3.5 mbps	640x480 3.5 mbps	640x480 3.5 mbps
1280x720	ビデオなし Apple Lossless	ビデオなし 64 kbps AAC	ビデオなし 64 kbps AAC	ビデオなし 64 kbps AAC
写真	ビデオ出力ではサポートされない	ビデオ出力ではサポートされない	ビデオ出力ではサポートされない	ビデオ出力ではサポートされない

録画の開始

QuickTimeムービーの録画を開始するには、`startRecordingToOutputFileURL:recordingDelegate:`を使用します。ファイルベースのURLとデリゲートを指定する必要があります。ムービーファイル出力は既存のリソースを上書きしないため、URLに既存のファイルを指定してはいけません。また、指定した場所に書き込むためのアクセス権が必要です。デリゲートは、`AVCaptureFileOutputRecordingDelegate`プロトコルに準拠し、`captureOutput:didFinishRecordingToOutputFileAtURL:fromConnections:error:`メソッドを実装する必要があります。

```
AVCaptureMovieFileOutput *aMovieFileOutput = <#Get a movie file output#>;
NSURL *fileURL = <#A file URL that identifies the output location#>;
[aMovieFileOutput startRecordingToOutputFileURL:fileURL recordingDelegate:<#The
 delegate#>];
```

デリゲートの`captureOutput:didFinishRecordingToOutputFileAtURL:fromConnections:error:`の実装では、作成されたムービーをカメラロールに書き込むこともできます。また、エラーが発生したかどうかも確認する必要があります。

ファイルが正常に書き込まれたことの確認

ファイルが正常に保存されたことを確認するには、`captureOutput:didFinishRecordingToOutputFileAtURL:fromConnections:error:`の実装で、エラーをチェックするだけなく、次のようにエラーのユーザ情報ディクショナリに含まれる`AVErrorRecordingSuccessfullyFinishedKey`の値も確認します。

```
- (void)captureOutput:(AVCaptureFileOutput *)captureOutput
        didFinishRecordingToOutputFileAtURL:(NSURL *)outputFileURL
        fromConnections:(NSArray *)connections
        error:(NSError *)error {

    BOOL recordedSuccessfully = YES;
    if ([error code] != noErr) {
        // 問題が発生した：正常に録画されたかどうかを確認する
        id value = [[error userInfo]
objectForKey:AVErrorRecordingSuccessfullyFinishedKey];
        if (value) {
            recordedSuccessfully = [value boolValue];
        }
    }
    // 必要に応じて処理を続行する...
```

エラーが発生してもファイルが正常に保存されている場合があるため、エラーのユーザ情報ディクショナリに含まれる`AVErrorRecordingSuccessfullyFinishedKey`の値を確認する必要があります。`AVErrorMaximumDurationReached`や`AVErrorMaximumFileSizeReached`など、録画に関するいずれかの制限に達したことを示すエラーもあります。録画が停止する理由として、ほかにも次のようなものがあります。

- ディスクがいっぱいになった－`AVErrorDiskFull`
- 録画デバイスが切断された（iPod touchからマイクが外された場合など）－`AVErrorDeviceWasDisconnected`
- セッションが中断された（電話がかかってきた場合など）－`AVErrorSessionWasInterrupted`

ファイルへのメタデータの追加

ムービーファイルのメタデータは、録画中を含めていつでも設定できます。これは、場所の情報など、録画の開始時に情報を取得できない場合に便利です。ファイル出力のメタデータは、AVMetadataItemオブジェクトの配列で表されます。このオブジェクトの可変サブクラスであるAVMutableMetadataItemのインスタンスを使用して独自のメタデータを作成します。

```
AVCaptureMovieFileOutput *aMovieFileOutput = <#Get a movie file output#>;
NSArray *existingMetadataArray = aMovieFileOutput.metadata;
AVMutableArray *newMetadataArray = nil;
if (existingMetadataArray) {
    newMetadataArray = [existingMetadataArray mutableCopy];
}
else {
    newMetadataArray = [[AVMutableArray alloc] init];
}

AVMutableMetadataItem *item = [[AVMutableMetadataItem alloc] init];
item.keySpace = AVMetadataKeySpaceCommon;
item.key = AVMetadataCommonKeyLocation;

CLLocation *location - <#The location to set#>;
item.value = [NSString stringWithFormat:@"%+08.4lf%+09.4lf/"
    location.coordinate.latitude, location.coordinate.longitude];

[newMetadataArray addObject:item];

aMovieFileOutput.metadata = newMetadataArray;
```

ビデオのフレームの処理

AVCaptureVideoDataOutputオブジェクトは、デリゲートを使用してビデオフレームを提供します。デリゲートを設定するにはsetSampleBufferDelegate:queue:を使用します。デリゲートに加えて、デリゲートメソッドを呼び出すシリアルキューを指定します。フレームが適切な順序でデリゲートに配信されるように、シリアルキューを使用する必要があります。現在のキューが特定のスレッドで実行されているという保証はないため、dispatch_get_current_queueから返されたキューを渡さないでください。このキューを使用して、ビデオフレームの配信と処理の優先度を変更できます。

フレームは、デリゲートメソッドcaptureOutput:didOutputSampleBuffer:fromConnection:にCMSampleBuffer不透過型のインスタンスとして提供されます（「メディアの表現」 （51 ページ）を参照）。デフォルトでは、カメラの最も効率的なフォーマットでバッファが送出されます。カスタム出力フォーマットを指定するには、videoSettingsプロパティを使用します。ビデオ設定プロパティはディクショナリです。現在サポートされている唯一のキーはkCVPixelBufferPixelFormatTypeKeyです。iPhone 4で推奨されるピクセルフォーマットは、kCVPixelFormatType_420YpCbCr8BiPlanarVideoRangeまたはkCVPixelFormatType_32BGRAです。iPhone 3Gで推奨されるピクセルフォーマットは、kCVPixelFormatType_422YpCbCr8またはkCVPixelFormatType_32BGRAです。BGRAフォーマットはCore GraphicsでもOpenGLでも正常に動作します。

```
AVCaptureSession *captureSession = <#Get a capture session#>;
NSDictionary *newSettings = [NSDictionary
dictionaryWithObject:(id)kCVPixelFormatType_32BGRA
```

```
forKey:(id)kCVPixelBufferPixelFormatTypeKey];
captureSession.videoSettings = newSettings;
```

ビデオ処理のパフォーマンスの検討事項

セッション出力をアプリケーションの最も低い実用解像度に設定してください。出力を必要以上に高い解像度に設定すると、処理サイクルが浪費され、電力の無駄遣いになります。

`captureOutput:didOutputSampleBuffer:fromConnection:`の実装がフレームに割り当てられた時間内に確実にサンプルバッファを処理できるようにする必要があります。この処理に時間がかかりすぎて、ビデオフレームを手放さないでいると、AV Foundationはデリゲートだけでなくほかの出力（プレビューレイヤなど）に対するフレームの配信も停止します。

キャプチャビデオデータ出力の`minFrameDuration`プロパティを使用して、フレームを処理する十分な時間を確保できるようにフレームレートを下げることができます。`alwaysDiscardsLateVideoFrames`プロパティを`YES`（デフォルト）に設定することもできます。これにより、遅いビデオフレームが処理対象として渡されずに破棄されます。また、録画中に出力フレームが多少遅れても問題ない場合は、このプロパティを`NO`に設定してすべてのフレームを取得することもできます。これはフレームが破棄されないという意味ではありません（つまり、この場合もフレームが破棄されることはあります）が、ただちに（効率的に）破棄されなくなります。

静止画像のキャプチャ

付随するメタデータを使用して静止画像をキャプチャしたい場合は、`AVCaptureStillImageOutput`出力を使用します。画像の解像度は、次の表に示すように、セッションのプリセットによって異なります。

プリセット	iPhone 3G	iPhone 3GS	iPhone 4（背面）	iPhone 4（前面）
高	400x304	640x480	1280x720	640x480
中	400x304	480x360	480x360	480x360
低	400x304	192x144	192x144	192x144
640x480	特になし	640x480	640x480	640x480
1280x720	特になし	特になし	1280x720	特になし
写真	1600x1200	2048x1536	2592x1936	640x480

ピクセルとエンコードフォーマット

次のように、デバイスによってサポートされる画像フォーマットが異なります。

iPhone 3G	iPhone 3GS	iPhone 4
yuvs、2vuy、BGRA、jpeg	420f、420v、BGRA、jpeg	420f、420v、BGRA、jpeg

サポートされるピクセルとコーデックの種類を調べるには、availableImageDataCVPixelFormatTypesとavailableImageDataCodecTypesをそれぞれ使用します。たとえば、次のようにoutputSettingsディクショナリを設定して必要な画像フォーマットを指定します。

```
AVCaptureStillImageOutput *stillImageOutput = [[AVCaptureStillImageOutput alloc]
 init];
NSDictionary *outputSettings = [[NSDictionary alloc] initWithObjectsAndKeys:
                                                        AVVideoCodecJPEG,
AVVideoCodecKey, nil];
[stillImageOutput setOutputSettings:outputSettings];
```

JPEG画像をキャプチャする場合は、通常、独自の圧縮フォーマットを指定するべきではありません。代わりに、ハードウェアで高速化される静止画像出力固有の圧縮を使用するべきです。画像のデータ表現が必要な場合は、jpegStillImageNSDataRepresentation:を使用すると、画像のメタデータを変更した場合でもデータを再圧縮せずにNSDataオブジェクトを取得できます。

画像のキャプチャ

画像をキャプチャするときは、出力にcaptureStillImageAsynchronouslyFromConnection:completionHandler:メッセージを送信します。最初の引数は、キャプチャに使用する接続です。次のように、入力ポートでビデオが収集されている接続を探す必要があります。

```
AVCaptureConnection *videoConnection = nil;
for (AVCaptureConnection *connection in stillImageOutput.connections) {
    for (AVCaptureInputPort *port in [connection inputPorts]) {
        if ([[port mediaType] isEqual:AVMediaTypeVideo] ) {
            videoConnection = connection;
            break;
        }
    }
    if (videoConnection) { break; }
}
```

captureStillImageAsynchronouslyFromConnection:completionHandler:に対する2番目の引数は、画像データを格納したCMSampleBufferとエラーの2つの引数を取るブロックです。サンプルバッファ自体にアタッチメントとしてメタデータ（Exifディクショナリなど）が含まれる場合があります。必要な場合はアタッチメントを変更できますが、「ピクセルとエンコードフォーマット」（41 ページ）で説明したJPEG画像の最適化に注意してください。

```
[stillImageOutput captureStillImageAsynchronouslyFromConnection:videoConnection
 completionHandler:
    ^(CMSampleBufferRef imageSampleBuffer, NSError *error) {
        CFDictionaryRef exifAttachments =
            CMGetAttachment(imageSampleBuffer, kCGImagePropertyExifDictionary,
 NULL);
        if (exifAttachments) {
            // 受け取ったアタッチメントを使って何か処理をする
        }
        // 必要に応じて処理を続行する
    }];
```

ユーザに対する録画内容の表示

カメラで録画している内容をユーザに提供するには、プレビューレイヤを使用します。マイクで録音している内容をユーザに提供するには、オーディオチャンネルを監視します。

ビデオのプレビュー

録画している内容のプレビューをユーザに対して表示するには、`AVCaptureVideoPreviewLayer`オブジェクトを使用します。`AVCaptureVideoPreviewLayer`は`CALayer`のサブクラスです（『*Core Animation Programming Guide*』を参照）。プレビューを表示するための出力は必要ありません。

キャプチャ出力とは異なり、ビデオのプレビューレイヤは関連付けられたセッションを保持します。これは、レイヤがビデオを表示しようとしている間にセッションが割り当て解除されないようにするためです。これは、次のように、プレビューレイヤを初期化する方法に反映されています。

```
AVCaptureSession *captureSession = <#Get a capture session#>;
CALayer *viewLayer = <#Get a layer from the view in which you want to present
the preview#>;

AVCaptureVideoPreviewLayer *captureVideoPreviewLayer =
[[AVCaptureVideoPreviewLayer alloc] initWithSession:captureSession];
[viewLayer addSublayer:captureVideoPreviewLayer];
```

一般に、プレビューレイヤはレンダリングツリー内のほかの`CALayer`オブジェクトと同じように動作します（『*Core Animation Programming Guide*』を参照）。ほかのレイヤと同じように、画像の拡大縮小、変換、回転などを実行できます。1つ異なる点として、カメラから入力された画像の回転方法を指定するためにレイヤの`orientation`プロパティを設定する必要がある場合があります。また、iPhone 4では、プレビューレイヤでミラーリングがサポートされます（これは、前面カメラをプレビューするときのデフォルトです）。

ビデオ映像の重心設定モード

プレビューレイヤでは、`videoGravity`を使用して設定する重心位置として、次の3つがサポートされます。

- `AVLayerVideoGravityResizeAspect`：アスペクト比が維持され、利用可能な画面領域いっぱいにビデオ映像が表示されないときは黒いバーが残ります。
- `AVLayerVideoGravityResizeAspectFill`：アスペクト比が維持されますが、必要に応じてビデオ映像はトリミングされ、利用可能な画面領域いっぱいに表示されます。
- `AVLayerVideoGravityResize`：利用可能な画面領域いっぱいに表示されるようにビデオが引き伸ばされます（画像が変形することもあります）。

プレビューでの「タップしてフォーカス」の使用

「タップしてフォーカス」をプレビューレイヤと組み合わせて実装するときは、注意が必要です。プレビューの向きとレイヤの重心位置、およびプレビューがミラーリングされている可能性を考慮する必要があります。

オーディオレベルの表示

キャプチャ接続に含まれるオーディオチャンネルの平均出力レベルやピーク出力レベルを監視するには、AVCaptureAudioChannelオブジェクトを使用します。オーディオレベルのキー値監視はできないため、ユーザインターフェイスを更新する頻度に合わせて（たとえば毎秒10回）レベルの更新をポーリングする必要があります。

```
AVCaptureAudioDataOutput *audioDataOutput = <#Get the audio data output#>;
NSArray *connections = audioDataOutput.connections;
if ([connections count] > 0) {
    // AVCaptureAudioDataOutputへの接続が1つだけ存在するはず
    AVCaptureConnection *connection = [connections objectAtIndex:0];

    NSArray *audioChannels = connection.audioChannels;

    for (AVCaptureAudioChannel *channel in audioChannels) {
        float avg = channel.averagePowerLevel;
        float peak = channel.peakHoldLevel;
        // レベルメータのユーザインターフェイスを更新する
    }
}
```

すべてを組み合わせる：UIImageオブジェクトとしてのビデオフレームのキャプチャ

次の簡単なコード例は、ビデオをキャプチャし、取得したフレームをUIImageオブジェクトに変換する方法を示しています。具体的な方法は次のとおりです。

- AVCaptureSessionオブジェクトを作成して、AV入力デバイスから出力へのデータの流れを調整する
- 必要な入力タイプのAVCaptureDeviceオブジェクトを見つける
- デバイスのAVCaptureDeviceInputオブジェクトを作成する
- AVCaptureVideoDataOutputオブジェクトを作成して、ビデオフレームを生成する
- AVCaptureVideoDataOutputオブジェクトのデリゲートを実装して、ビデオフレームを処理する
- デリゲートから受け取ったCMSampleBufferをUIImageオブジェクトに変換する関数を実装する

注： この例では最も重要なコードに注目するため、完全なアプリケーションからメモリ管理など、いくつかの部分を省略しています。AV Foundationを使用するには、欠落した部分を推測できるように、Cocoaに関する十分な経験を積んでいることが期待されます。

キャプチャセッションの作成と設定

AV入力デバイスから出力へのデータの流れを調整するには、`AVCaptureSession`オブジェクトを使用します。セッションを作成し、中程度の解像度を持つビデオフレームを生成するように設定します。

```
AVCaptureSession *session = [[AVCaptureSession alloc] init];
session.sessionPreset = AVCaptureSessionPresetMedium;
```

デバイスおよびデバイス入力の作成と設定

キャプチャデバイスは`AVCaptureDevice`オブジェクトで表されます。このクラスには、必要な入力タイプのオブジェクトを取得するためのメソッドが用意されています。デバイスには1つ以上のポートがあり、`AVCaptureInput`オブジェクトを使用して各ポートを設定します。通常、キャプチャ入力はデフォルト設定で使用します。

ビデオキャプチャデバイスを見つけ、そのデバイスを使用してデバイス入力を作成し、そのデバイス入力にセッションを追加します。

```
AVCaptureDevice *device =
        [AVCaptureDevice defaultDeviceWithMediaType:AVMediaTypeVideo];

NSError *error = nil;
AVCaptureDeviceInput *input =
        [AVCaptureDeviceInput deviceInputWithDevice:device error:&error];
if (!input) {
    // エラーを適切に処理する
}
[session addInput:input];
```

データ出力の作成と設定

キャプチャ中のビデオから入力された非圧縮フレームを処理するには、`AVCaptureVideoDataOutput`を使用します。通常は、出力のいくつかの側面を設定します。たとえば、ビデオの場合は、`videoSettings`プロパティを使用してピクセルフォーマットを指定したり、`minFrameDuration`プロパティを設定してフレームレートを制限したりできます。

次のように、ビデオデータ用の出力を作成して設定し、それをセッションに追加します。また、`minFrameDuration`プロパティを1/15秒に設定してフレームレートを15fpsまでに制限します。

```
AVCaptureVideoDataOutput *output = [[[AVCaptureVideoDataOutput alloc] init]
autorelease];
[session addOutput:output];
output.videoSettings =
                [NSDictionary dictionaryWithObject:[NSNumber
numberWithInt:kCVPixelFormatType_32BGRA]
```

```
forKey:(id)kCVPixelBufferPixelFormatTypeKey];
output.minFrameDuration = CMTimeMake(1, 15);
```

データ出力オブジェクトは、デリゲートを使用してビデオフレームを提供します。デリゲートは、AVCaptureVideoDataOutputSampleBufferDelegateプロトコルを採用する必要があります。データ出力のデリゲートを設定するときは、コールバックを呼び出すキューも指定する必要があります。

```
dispatch_queue_t queue = dispatch_queue_create("MyQueue", NULL);
[output setSampleBufferDelegate:self queue:queue];
dispatch_release(queue);
```

キューを使用して、ビデオフレームの配信と処理の優先度を変更します。

サンプルバッファのデリゲートメソッドの実装

デリゲートクラスには、サンプルバッファが書き込まれたときに呼び出されるメソッド(captureOutput:didOutputSampleBuffer:fromConnection:)を実装します。ビデオデータ出力オブジェクトはフレームをCMSampleBufferとして配信するため、CMSampleBufferをUIImageオブジェクトに変換する必要がありますこの操作の機能は、「CMSampleBufferからUIImageへの変換」（51 ページ）に示しています。

```
- (void)captureOutput:(AVCaptureOutput *)captureOutput
         didOutputSampleBuffer:(CMSampleBufferRef)sampleBuffer
         fromConnection:(AVCaptureConnection *)connection {

    UIImage *image = imageFromSampleBuffer(sampleBuffer);
    // 画像を使用するコードをここに追加する
}
```

このデリゲートメソッドはsetSampleBufferDelegate:queue:で指定したキュー上で呼び出されます。ユーザインターフェイスを更新したい場合は、関連するコードをメインスレッド上で呼び出す必要があります。

録画の開始と停止

キャプチャセッションを設定したら、セッションにstartRunningメッセージを送信して録画を開始します。

```
[session startRunning];
```

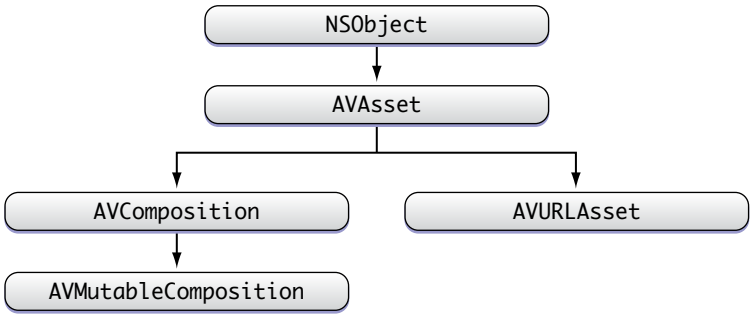
録画を停止するには、セッションにstopRunningメッセージを送信します。

時間およびメディアの表現

AV Foundationフレームワークでは、時間ベースのオーディオビジュアルデータ（ムービーファイルやビデオストリームなど）をAVAssetで表します。フレームワークのほとんどの作業は、アセットの構造によって規定されます。AV Foundationが時間とメディアの表現に使用するいくつかの低レベルデータ構造（サンプルバッファなど）は、Core Mediaフレームワークによって提供されます。

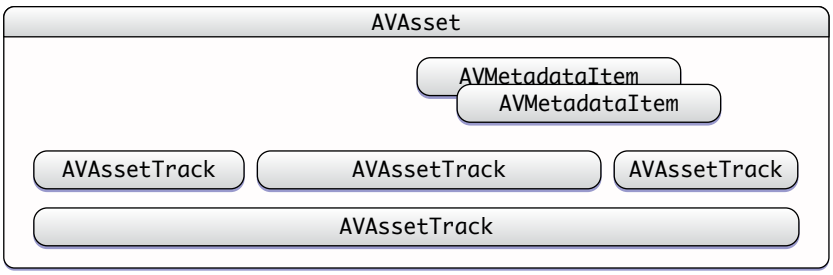
アセットの表現

AVAssetは、AV Foundationフレームワークのコアクラスです。時間ベースのオーディオビジュアルデータ（ムービーファイルやビデオストリーム）のフォーマットに依存しない抽象表現を提供します。多くの場合は、サブクラスのいずれかを使用します。新しいアセットを作成するときは、コンポジションサブクラスを使用します（「編集」 （7 ページ）を参照）。また、特定のURLにあるメディアから新しいアセットインスタンスを作成するには、AVURLAssetを使用します（MPMediaフレームワークまたはAsset Libraryフレームワークのアセットを含む。「アセットの使用」 （9 ページ）を参照）。



アセットには、まとめて表示または処理するトラックのコレクションが含まれています。各トラックのメディアタイプは同じであり、メディアタイプにはオーディオ、ビデオ、テキスト、クローズドキャプション、サブタイトルなどがあります（ただし、これらに限定されません）。アセットオブジェクトは、リソース全体に関する情報（再生時間やタイトルなど）とともに、表示に関するヒント（固有のサイズなど）を提供します。アセットには、AVMetadataItemのインスタンスで表されるメタデータも格納できます。

トラックは、AVAssetTrackのインスタンスで表されます。通常の簡単なケースでは、オーディオコンポーネントを表すトラックとビデオコンポーネントを表すトラックが別個に存在しますが、複雑なコンポジションでは、オーディオとビデオのトラックが複数重複して存在する場合もあります。



トラックは、トラックのタイプ（ビデオまたはオーディオ）、ビデオまたはオーディオの特性（該当する場合）、メタデータ、タイムライン（親アセットに関して表される）などのいくつかのプロパティと、フォーマット記述の配列を持っています。この配列には、トラックが参照するメディアサンプルのフォーマットを記述するCMFormatDescriptionが格納されます（`CMFormatDescriptionRef`を参照）。同じメディアを格納しているトラック（たとえば、すべて同じ設定を使用してエンコードされた場合など）は、個数1の配列を提供します。

トラック自体が`AVAssetTrackSegment`のインスタンスで表される複数のセグメントに分割されることもあります。セグメントは、ソースからアセットトラックのタイムラインにマッピングされる時間です。

時間の表現

AV Foundationでは、時間をCore Mediaフレームワークの基本的な構造体で表します。

時間の長さを表すCMTime

`CMTime`は、時間を分子（`int64_t`の値）と分母（`int32_t`の時間係数）からなる有理数で表すCの構造体です。時間係数は、概念的には分子の各単位が1秒間に占める割合を示します。したがって、時間係数が4であれば、各単位は1/4秒を表し、時間係数が10であれば、各単位は1/10秒を表します。時間係数としては、よく使用されるフレームレートの公倍数である600が頻繁に使用されます。よく使用されるフレームレートとは、映画の24fps（1秒あたりのフレーム数）、NTSCの30fps（北米および日本のTVで使用される）、およびPALの25fps（ヨーロッパのTVで使用される）です。時間係数として600を使用することで、これらのシステムのフレーム数を正確に表すことができます。

CMTimeは、単純な時間値に加えて、非数値である正の無限値、負の無限値、および不定値を表すこともできます。また、時間の丸め処理が行われたかどうかを示したり、エポック番号を保持することもできます。

CMTimeの使用

時間を作成するには、`CMTimeMake`または関連する関数のいずれか（たとえば、float型の値にして時間を作成し、任意の時間スケールを指定できる`CMTimeMakeWithSeconds`など）を使用します。次の例に示すように、時間ベースの算術演算や時間の比較を行うための関数がいくつか用意されています。

```
CMTime time1 = CMTimeMake(200, 2); // 1/2秒単位で200
CMTime time2 = CMTimeMake(400, 4); // 1/4秒単位で400
```

```
// time1とtime2はどちらも100秒を表すが、使用している時間係数が異なる
if (CMTimeCompare(time1, time2) == 0) {
    NSLog(@"time1 and time2 are the same");
}

Float64 float64Seconds = 200.0 / 3;
CMTime time3 = CMTimeMakeWithSeconds(float64Seconds , 3); // 66.66...  (1/3秒単
位で)
time3 = CMTimeMultiply(time3, 3);
// time3は200秒を表す。次に、time1 (100秒) を差し引く
time3 = CMTimeSubtract(time3, time1);
CMTimeShow(time3);

if (CMTIME_COMPARE_INLINE(time2, ==, time3)) {
    NSLog(@"time2 and time3 are the same");
}
```

利用可能なすべての関数の一覧については、『*CMTime Reference*』を参照してください。

CMTimeの特殊な値

Core Mediaには、特殊な値を表すための定数として、`kCMTimeZero`、`kCMTimeInvalid`、`kCMTimePositiveInfinity`、および`kCMTimeNegativeInfinity`が用意されています。ただし、CMTimeでは、たとえば時間として無効な値をさまざまな方法で表すことができます。CMTimeが無効または非数値かどうかを確認する必要がある場合は、適切なマクロ（`CMTIME_IS_INVALID`、`CMTIME_IS_POSITIVE_INFINITY`、`CMTIME_IS_INDEFINITE`など）を使用してください。

```
CMTime myTime = <#Get a CMTime#>;
if (CMTIME_IS_INVALID(myTime)) {
    // これをおそらくエラーとして処理し、ユーザに対して適切な警告を表示する
}
```

任意のCMTimeの値を`kCMTimeInvalid`と比較しないでください。

オブジェクトとしてのCMTimeの表現

注釈またはCore Foundationコンテナで CMTimeを使用する必要がある場合は、`CMTimeCopyAsDictionary`を使用してCMTimeをCFDictionary（`CFDictionaryRef`を参照）に変換し、`CMTimeMakeFromDictionary`を使用してCFDictionaryをCMTimeに変換することができます。`CMTimeCopyDescription`を使用してCMTimeの文字列表現を取得することもできます。

エポック

CMTimeのエポック番号は通常0に設定されていますが、エポック番号を使用して関係のないタイムラインを区別することができます。たとえば、表示ループを1回通すたびにエポックをインクリメントすることにより、ループ0の時間Nとループ1の時間Nとを区別できます。

時間範囲を表すCMTimeRange

`CMTimeRange`はCの構造体で、CMTimeで表される開始時刻と持続時間を保持します。時間範囲には、開始時刻に持続時間を加えた時刻は含まれません。

時間範囲を作成するには、CMTimeRangeMakeまたはCMTimeRangeFromTimeToTimeを使用します。CMTimeのエポックの値には次の制約があります。

- 異なるエポックにまたがるCMTimeRangeは作成できません。
- タイムスタンプを表すCMTimeのエポックは、0以外にすることができますが、範囲操作（CMTimeRangeGetUnionなど）の実行対象になるのは開始フィールドが同じエポックを持つ範囲だけです。
- 持続時間を表すCMTimeのエポックは、常に0にする必要があり、負数でない値にする必要があります。

時間範囲の操作

Core Mediaには、時間範囲に特定の時間や他の時間範囲が含まれているかどうかを判定したり、2つの時間範囲が等しいかどうかを判定したり、時間範囲の論理和や論理積を計算したりする関数（CMTimeRangeContainsTime、CMTimeRangeEqual、CMTimeRangeContainsTimeRange、CMTimeRangeGetUnionなど）が用意されています。

開始時刻に持続時間を加えた時刻は時間範囲に含まれないため、次の式は常にfalseとして評価されます。

```
CMTimeRangeContainsTime(range, CMTimeRangeGetEnd(range))
```

利用可能なすべての関数の一覧については、『*CMTimeRange Reference*』を参照してください。

CMTimeRangeの特殊な値

Core Mediaには、長さ0の範囲を表す定数(kCMTimeRangeZero)と無効な範囲を表す定数(kCMTimeRangeInvalid)が用意されています。ただし、CMTimeRangeでは、無効、ゼロ、または不定（CMTimeのいずれかが不定の場合）をさまざまな方法で表すことができます。CMTimeRangeが無効、ゼロ、または不定かを確認する必要がある場合は、適切なマクロ（CMTIMERANGE_IS_VALID、CMTIMERANGE_IS_INVALID、CMTIMERANGE_IS_EMPTY、CMTIMERANGE_IS_EMPTY）を使用してください。

```
CMTimeRange myTimeRange = <#Get a CMTimeRange#>;
if (CMTIMERANGE_IS_EMPTY(myTimeRange)) {
    // 時間範囲がゼロである
}
```

任意のCMTimeRangeの値をkCMTimeRangeInvalidと比較しないでください。

オブジェクトとしてのCMTimeRangeの表現

注釈またはCore Foundationコンテナで CMTimeRangeを使用する必要がある場合は、CMTimeRangeCopyAsDictionaryを使用してCMTimeRangeをCFDictionary（CFDictionaryRefを参照）に変換し、CMTimeRangeMakeFromDictionaryを使用してCFDictionaryをCMTimeRangeに変換することができます。CMTimeRangeCopyDescriptionを使用してCMTimeの文字列表現を取得することもできます。

メディアの表現

AV Foundationでは、ビデオデータと関連するメタデータをCore Mediaフレームワークの不透過オブジェクトで表します。Core Mediaでは、CMSampleBufferを使用してビデオデータが表されます（CMSampleBufferRefを参照）。CMSampleBufferは、Core Foundationスタイルの不透過型です。インスタンスにはビデオデータのフレームのサンプルバッファがCore Videoのピクセルバッファ（CVPixelBufferRefを参照）として格納されます。サンプルバッファからピクセルバッファにアクセスするには、次のようにCMSampleBufferGetImageBufferを使用します。

```
CVPixelBufferRef pixelBuffer = CMSampleBufferGetImageBuffer(<#A CMSampleBuffer#>);
```

ピクセルバッファから実際のビデオデータにアクセスできます。具体例については、「CMSampleBufferからUIImageへの変換」（51 ページ）を参照してください。

ビデオデータに加えて、次のようなビデオフレームのほかの側面も取得できます。

- **タイミング情報**：元の表示時間およびデコード時間の正確なタイムスタンプを取得するには、CMSampleBufferGetPresentationTimeStampおよびCMSampleBufferGetDecodeTimeStampをそれぞれ使用します。
- **フォーマット情報**：フォーマット情報は、CMFormatDescriptionオブジェクトにカプセル化されています（CMFormatDescriptionRefを参照）。このフォーマット記述から、たとえば、CMVideoFormatDescriptionGetCodecTypeを使用してピクセルタイプを取得したり、CMVideoFormatDescriptionGetDimensionsを使用してビデオの寸法を取得したりできます。
- **メタデータ**：メタデータは、ディクショナリに**アタッチメント**として格納されています。このディクショナリを取得するには、次のようにCMGetAttachmentを使用します。

```
CMSampleBufferRef sampleBuffer = <#Get a sample buffer#>;
CFDictionaryRef metadataDictionary =
    CMGetAttachment(sampleBuffer, CFSTR("MetadataDictionary", NULL);
if (metadataDictionary) {
    // メタデータを使って何か処理をする
}
```

CMSampleBufferからUIImageへの変換

次の関数は、CMSampleBufferをUIImageオブジェクトに変換する方法を示しています。この関数を使用する前に、要件を慎重に検討してください。この変換は比較的負荷の高い操作です。たとえば、1秒程度の間隔で取得したビデオデータのフレームから静止画像を作成するのが適切です。この関数を、キャプチャデバイスから入力されるすべてのフレームをリアルタイムで操作する手段として使用しないでください。

```
UIImage *imageFromSampleBuffer(CMSampleBufferRef sampleBuffer) {

    CVImageBufferRef imageBuffer = CMSampleBufferGetImageBuffer(sampleBuffer);
    // ピクセルバッファのベースアドレスをロックする
    CVPixelBufferLockBaseAddress(imageBuffer,0);

    // ピクセルバッファの1行あたりのバイト数を取得する
    size_t bytesPerRow = CVPixelBufferGetBytesPerRow(imageBuffer);
```

```
    // ピクセルバッファの幅と高さを取得する
    size_t width = CVPixelBufferGetWidth(imageBuffer);
    size_t height = CVPixelBufferGetHeight(imageBuffer);

    // デバイス依存のRGB色空間を作成する
    static CGColorSpaceRef colorSpace = NULL;
    if (colorSpace == NULL) {
        colorSpace = CGColorSpaceCreateDeviceRGB();
            if (colorSpace == NULL) {
            // エラーを適切に処理する
            return nil;
        }
    }

    // ピクセルバッファのベースアドレスを取得する
    void *baseAddress = CVPixelBufferGetBaseAddress(imageBuffer);
    // ピクセルバッファの連続したプレーンのデータサイズを取得する
    size_t bufferSize = CVPixelBufferGetDataSize(imageBuffer);

    // 用意されたデータを使用するQuartzの直接アクセスデータプロバイダを作成する
    CGDataProviderRef dataProvider =
        CGDataProviderCreateWithData(NULL, baseAddress, bufferSize, NULL);
    // データプロバイダが提供するデータからビットマップ画像を作成する
    CGImageRef cgImage =
        CGImageCreate(width, height, 8, 32, bytesPerRow,
                      colorSpace, kCGImageAlphaNoneSkipFirst |
kCGBitmapByteOrder32Little,
                      dataProvider, NULL, true, kCGRenderingIntentDefault);
    CGDataProviderRelease(dataProvider);

    // Quartz画像を表現するための画像オブジェクトを作成して返す
    UIImage *image = [UIImage imageWithCGImage:cgImage];
    CGImageRelease(cgImage);

    CVPixelBufferUnlockBaseAddress(imageBuffer, 0);

    return image;
}
```

書類の改訂履歴

この表は「*AV Foundation プログラミングガイド*」の改訂履歴です。

日付	メモ
2011-10-12	iOS 5用に更新し、リリースノートへの参照を含めました。
2011-04-28	Mac OS X 10.7向けの初版。
2010-11-15	キャプチャに関する内容を追加しました。
2010-09-08	未定
2010-08-16	メディアアセットの再生、検査、作成、編集、キャプチャ、変換を行う下位レベルのフレームワークについて説明した文書の初版。

書類の改訂履歴